UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

RICARDO LUIS DOS SANTOS

Deploying and Managing Network Services over Programmable Virtual Networks

Thesis presented in partial fulfillment
of the requirements for the degree of
Doctor of Computer Science

Advisor: Prof. Dr. Lisandro Zambenedetti Granville
Coadvisor: Prof.ª Dr.ª Liane M. Rockenbach Tarouco

Porto Alegre
September 2018

CIP – CATALOGING-IN-PUBLICATION

Santos, Ricardo Luis dos

Deploying and Managing Network Services over Programmable Virtual Networks / Ricardo Luis dos Santos. – Porto Alegre: PPGC da UFRGS, 2018.

177 f.: il.

Thesis (Ph.D.) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2018. Advisor: Lisandro Zambenedetti Granville; Coadvisor: Liane M. Rockenbach Tarouco.

1. Network Programmability. 2. Network Marketplace. 3. Network Applications. 4. Programmable Virtual Networks. I. Granville, Lisandro Zambenedetti. II. Tarouco, Liane M. Rockenbach. III. Título.

“I don’t believe in magic. I believe in hard work.”

— RICHIE MCCAW (ALL BLACKS CAPTAIN)

ACKNOWLEDGEMENTS

Acknowledgements are written in Portuguese because of my family. Meus sinceros agradecimentos para todos aqueles que confiaram em mim e que me apoiaram nos momentos mais difíceis, a saber, minha mãe, meu pai, meu irmão, minha namorada, meu orientador, os verdadeiros amigos, os colegas (de trabalho e da UFRGS) e também todos os conhecidos. Saibam que todos vocês tiveram uma parcela de suma importância e me ajudaram a seguir em frente. Ainda, gostaria de agradecer também todos aqueles que, em algum momento, duvidaram ou colocaram alguma barreira para eu atingir meus objetivos. Saibam que vocês me fortaleceram nos momentos que eu mais precisei.

Muito obrigado!

ABSTRACT

Several companies and standardization bodies have attempted to define standards and find solutions for networks that support virtualization and programmability (also known as Programmable Virtual Networks - PVNs), such as Juniper Junos, Cisco ONE, OpenFlow, ETSI NFV, and IETF ForCES. These solutions provide an Execution Environment (EE) that supports the deployment of Network Applications (NetApps). However, in the case of PVNs, several tasks are complex and repetitive, mainly because the PVN owners must have an extensive knowledge of the device instructions to deploy and manage NetApps. A few years ago, Apple and Google overcame similar shortcomings by introducing online marketplaces. The successful paradigm for mobile application stores could be applied to the networking market, and this could help PVN owners to deploy and manage NetApps in their PVNs. In this thesis, we review the state-of-the-art of network programmability and virtualization, as well as discuss their main drawbacks. Following this, we analyze the historical background of marketplaces with regard to networking paradigms to define the essential design goals for a reference network marketplace. Thus, inspired by the main drawbacks and essential design goals, we propose the App2net ecosystem. App2net empowers PVN owners, who do not know the specific features of the underlying infrastructure, to deploy and manage NetApps in PVNs formed of different EEs. We conducted a case study to evaluate our ecosystem, which contains all the steps for a third-party developer to describe and publish an innovative network service. Then, we describe the main interactions for a PVN owner when deploying and managing this network service in a PVN with heterogenous EEs. Also, we implemented a prototype of App2net to evaluate the feasibility and the impact in terms of the distribution time, CPU usage, and network overhead. When the baseline results are taken into account, App2net demonstrates an ability to reduce the distribution time and the total network traffic generated. At the same time, our ecosystem increases the CPU usage (required to execute it) and also had minimal network overhead.

Keywords: Network Programmability. Network Marketplace. Network Applications. Programmable Virtual Networks.

Implantando e Gerenciando Serviços de Rede em Redes Virtuais Programáveis

RESUMO

Várias empresas e órgãos de normalização têm tentado definir padrões e buscar soluções para redes que suportam virtualização e programabilidade (também conhecidas como Redes Virtuais Programáveis - *Programmable Virtual Networks* - PVNs), tais como Juniper Junos, Cisco ONE, OpenFlow, ETSI NFV e IETF ForCES. Essas soluções fornecem um ambiente de execução (*Execution Environment* - EE) que suporta a implantação de Aplicações de Rede (*Network Applications* - NetApps). No entanto, no caso das PVNs, diversas tarefas são complexas e repetitivas, principalmente porque os proprietários de PVNs devem ter um amplo conhecimento das instruções dos dispositivos para implantar e gerenciar os NetApps. Há alguns anos, a Apple e o Google superaram deficiências semelhantes ao introduzir lojas *on-line*. O paradigma de sucesso das lojas de aplicativos móveis poderia ser aplicado ao mercado de redes, e isso poderia auxiliar os proprietários de PVN a implantar e gerenciar NetApps em suas PVNs. Nesta tese, nós revisamos o estado da arte da virtualização e programabilidade de redes, bem como discutimos as suas principais desvantagens. Em seguida, nós analisamos o histórico das lojas em relação aos paradigmas de rede para definir os objetivos essenciais de projeto para uma loja de rede de referência. Assim, inspirados pelas principais desvantagens e pelos objetivos essenciais de projeto, nós propomos o ecosistema App2net. O App2net capacita os proprietários de PVN, que não conhecem os recursos específicos da infraestrutura subjacente, a implantar e gerenciar NetApps em PVNs formadas por diferentes EEs. Nós conduzimos um estudo de caso para avaliar nosso ecosistema, o qual contém todos os passos para um desenvolvedor terceirizado descrever e publicar um serviço de rede inovador. Então, nós descrevemos as principais interações para um proprietário de PVN, ao implantar e gerenciar este serviço de rede em uma PVN com EEs heterogêneos. Ainda, nós implementamos um protótipo do App2net para avaliar a viabilidade e o impacto em termos do tempo de distribuição, uso de CPU e sobrecarga da rede. Quando os resultados de base são considerados, App2net demonstra uma habilidade para reduzir o tempo de distribuição e o tráfego total de rede gerado. Ao mesmo tempo, nosso ecosistema incrementa o uso da CPU (necessário para executá-lo) e também teve uma sobrecarga de rede mínima.

Palavras-chave: Programabilidade de Redes. Loja de Rede. Aplicações de Rede. Redes Virtuais Programáveis..

LIST OF ABBREVIATIONS AND ACRONYMS

3G	Third Generation
3GPP	3rd Generation Partnership Project
6Bone	IPv6 Backbone
ANTS	Active Node Transfer System
AON	Application-Oriented Networking
API	Application Programming Interface
app	Application
app2net	Applications to Network
ASIC	Application Specific Integrated Circuits
AWS	Amazon Web Service
CAIDA	Center for Applied Internet Data Analysis
CAPEX	Capital Expenditure
CE	Control Element
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit
DARPA	Defense Advanced Research Projects Agency
DB	Database
DDoS	Distributed Denial-of-Service
DoS	Denial-of-Service
DPI	Deep Packet Inspection
DSL	Digital Subscriber Line
EC2	Elastic Compute Cloud
EE	Execution Environment
ETSI	European Telecommunications Standards Institute

FDM	Frequency Division Multiplexing
FE	Forwarding Element
FIFO	First In First Out
ForCES	Forwarding and Control Element Separation
FPGA	Field Programmable Gate Array
GB	Gigabyte
Gbps	Gigabits per second
GHz	Gigahertz
GRE	Generic Routing Encapsulation
HPE	Hewlett-Packard Enterprise
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Task Force
iMPROVE	Marketplace for Programmable Virtual Network
IoT	Internet of Things
iOS	iPhone Operating System
IP	Internet Protocol
IPSec	Internet Protocol Security
IPv6	Internet Protocol version 6
ISG	Industry Specification Group
ISP	Internet Service Provider
IT	Information Technology
JDN	Juniper Developer Network
JPSM	Juniper Professional Services Marketplace
JSON	JavaScript Object Notation
JunOS	Junos Operating System
KB	Kilobyte

L2	Layer 2
L3	Layer 3
L1VPN	Layer 1 Virtual Private Network
L2VPN	Layer 2 Virtual Private Network
L3VPN	Layer 3 Virtual Private Network
LAN	Local Area Network
MANO	Management & Orchestration
MB	Megabyte
MBone	Mutlticast Backbone
MIB	Management Information Base
NACR	Network Application Code Repository
NAD	Network Application Descriptor
NAT	Network Address Translation
NetApp	Network Application
NETCONF	Network configuration
NetFPGA	Network Field Programmable Gate Array
NFV	Network Functions Virtualization
NIC	Network Interface Card
NIST	National Institute of Standards and Technology
NS2	Network Simulator version 2
NS3	Network Simulator version 3
NSD	Network Service Descriptor
NTP	Network Time Protocol
NVE	Network Virtualization Environment
ONE	Open Network Environment
ONF	Open Network Foundation

onePK	Open Network Environment Platform Kit
OPEX	Operational Expenditure
OPNFV	Open Platform for Network Functions Virtualization
OS	Operating System
OSI	Open Systems Interconnection
P4	Programming Protocol-Independent Packet Processors
POF	Protocol-Oblivious Forwarding
POF-FIS	Protocol-Oblivious Forwarding Flow Instruction Set
PVN	Programmable Virtual Network
QEMU	Quick Emulator
QoS	Quality of Service
REST	Representational State Transfer
RFC	Request for Comments
RSA	Rivest-Shamir-Adleman
RTT	Round-Trip Time
SDK	Software Development Kit
SDN	Software-Defined Networking
SFC	Service Function Chaining
SHA	Secure Hash Algorithm
SIG	Softgoal Interdependency Graph
SLA	Service Level Agreement
SNMP	Simple Network Management Protocol
SR-IOV	Single Root Input/Output Virtualization
TCP	Transmission Control Protocol
TDM	Time Division Multiplexing
TTL	Time To Live

URI	Uniform Resource Identifier
VIM	Virtualized Infrastructure Manager
VLAN	Virtual Local Area Network
VM	Virtual Machine
VN	Virtual Network
VNA	Virtual Network Appliance
VNF	Virtualized Network Function
VNFM	Virtualized Network Function Manager
VoIP	Voice over Internet Protocol
VPN	Virtual Private Network
VXDL	Virtual Infrastructure Description Language
vNIC	virtual Network Interface Card
WDM	Wavelength Division Multiplexing
XML	eXtensible Markup Language
XNC	eXtensible Network Controller
YAML	YAML Ain't Markup Language

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

Computer networks are built up using dedicated devices based on purpose-specific hardware (FEAMSTER; REXFORD; ZEGURA, 2014). These devices (also known as middleboxes) generally run closed source and proprietary software that carry out a wide range of network services, such as traffic filtering, intrusion detection, routing, and switching. Both the middleboxes and combined software have undergone years of development and testing to achieve stability and a reliable performance. As a result, there has been an increase in capital expenditure (CAPEX) and operational expenditure (OPEX) which restricts networking market to a few large companies. In addition, the growing number of heterogeneous devices and the wide range of protocols, services, and technologies have made computer networks an unfriendly environment for innovation. Thus, even straightforward and necessary changes (*e.g.*, IPv6 and IPSec) depend on the commercial interests of companies and replacement of middleboxes in the whole network, which sets up a huge barrier for innovation, and is often referred to as "Internet Ossification" (KIM; FEAMSTER, 2013).

In the 1990s, the concept of network programmability (LAZAR, 1997) arose and made it possible to rapidly build, deploy, and manage new services in the network (CAMPBELL et al., 1999) and thus overcome the problem of "Internet Ossification" (FEAMSTER; REXFORD; ZEGURA, 2014). Since then, several programmability-related technologies have emerged, such as Active Networks (TENNENHOUSE et al., 1997), Mobile Agents (CHESS; HARRISON; KERSHENBAUM, 1997), and Script MIB (LEVI; SCHOENWAELDER, 1999), and these allow developers to implement and deploy their programs in the core network. However, none of these technologies were in fact adopted, because they lack firm guarantees about the settings and applications deployed in the devices, and there was a risk that failures could either interfere with or lead to a collapse of the production network (MOORE; NETTLES, 2001). All this has hampered the rapid introduction of innovative network services that can meet the demands of new users and, as a result, network programmability still remains an open problem (NUNES et al., 2014) (MACEDO et al., 2015) (MIJUMBI et al., 2016).

In the last few years, the main vendors of the networking devices (such as Cisco (CISCO SYSTEMS, 2016), Juniper (JUNIPER NETWORKS, 2016b), and Extreme Networks (EXTREME NETWORKS, 2016)) have begun employing network virtualization technologies. In the Network Virtualization Environment (NVE), it is possible to have

multiple virtual networks (VNs) running simultaneously on top of a shared physical infrastructure (ESTEVES; GRANVILLE; BOUTABA, 2013). One essential feature of network virtualization is isolation, which allows diverse network architectures to coexist in a single physical infrastructure without affecting the production services, as well as virtual network nodes to be created, moved, and deleted. The provision of these nodes to network operators allows the deployment and execution of programs in an isolated way (CHOWDHURY; BOUTABA, 2009). Thus, if there is a misbehaving code, these programs do not affect the production network. Finally, the network operator can easily delete a set of devices that run misbehaving programs, and in this way prevent the physical infrastructure from collapsing.

Currently, academia, industry (*e.g.*, Cisco and Juniper), and standardization bodies (*e.g.*, ETSI and the IETF) are seeking to define standards and solutions for networks that support virtualization and programmability (also known as Programmable Virtual Networks - PVNs) (FEAMSTER; REXFORD; ZEGURA, 2014) (MACEDO et al., 2015) (CHOWDHURY; BOUTABA, 2009), such as OpenFlow (MCKEOWN et al., 2008), ETSI NFV (NFV ISG, 2012), IETF ForCES (DANTU; ANDERSON; GOPAL, 2004), Juniper Junos SDK (JUNIPER NETWORKS, 2015), and Cisco onePK (CISCO, 2012b). These solutions provide an Execution Environment (EE) that supports the deployment of innovative network services as diverse as multicasting (ZHANG et al., 2015), adaptive video distribution (JIN; WEN; WESTPHAL, 2015), intrusion prevention system (XING et al., 2014), and video transcoding (EGILMEZ; CIVANLAR; TEKALP, 2013). The benefits of PVNs include: (*i*) the ability to rapidly introduce new network services, (*ii*) a more open networking market for small companies and third-party developers, and (*iii*) the possibility of having multiple virtual network architectures on the same physical substrate. In addition, PVNs make the network more flexible and encourage innovation through the so-called network softwarization (MANZALINI; CRESPI, 2016) (RAMOS et al., 2014).

PVNs enable network services to migrate from middleboxes and proprietary software to Virtual Network Appliances (VNAs), hosted on commercial off-the-shelf (COTS) servers. Essentially, VNAs are virtual machine instances that contain EEs and Network Applications (NetApps), encoded by operators or third-party developers, who provide innovative network services. A single VNA can host just one NetApp (*e.g.*, Firewall or Load-Balancer) or an entire framework containing several NetApps (*e.g.*, Firewall, Antivirus, Encryption, Intrusion Protection System, and Deep Packet Inspection). Although

PVNs support a broad range of NetApps, it is not feasible to have a network that is solely comprised of VNAs. The performance of COTS servers is limited, and the overhead introduced by virtualization has proved to be an obstacle to high-speed network processing because it reduces throughput and increases latency (WOOD et al., 2015). Because of this, some NetApps (*e.g.*, switching, routing, and error control) must still remain hosted in middleboxes so that they can overcome performance constraints. Thus, VNAs and middleboxes coexist in the same network, where they can achieve a good performance and offer flexibility (SEZER et al., 2013). Moreover, it should be noted that since there is no common glossary of terms for PVNs, each technology refers to the same concepts in different ways; for example, NFV refers to NetApps as "network functions" whereas OpenFlow calls them by different names such as "business applications" and "SDN control software". To overcome this problem, we have compiled a glossary that brings together terms related to the same concepts but found in different technologies (see Subsection 2.3).

The programmability-related solutions mentioned above have attracted a good deal of attention in recent years. The main research efforts in this area can be divided into two groups. The first involves aggregating new features for each solution, including (but not limited to) scalability (MA et al., 2015) (TOOTOONCHIAN; GANJALI, 2010), performance (BRONSTEIN et al., 2015) (GELBERGER; YEMINI; GILADI, 2013) (TARIQ et al., 2014), isolation (SHERWOOD et al., 2009) (SHIH et al., 2016), security (SON et al., 2013) (BENTON; CAMP; SMALL, 2013) (BERNARDO; CHUA, 2015), management (PALKAR et al., 2015) (WICKBOLDT et al., 2015), and debugging & tests (CANINI et al., 2012) (VEITCH; MCGRATH; BAYON, 2015). The second group is concerned with the development of new NetApps such as reliable multicasting (CEN; LI; WANG, 2015) (SHEN et al., 2015) (EDWARDS; GIULIANO; WRIGHT, 2002), topology discovery (TARNARAS; HALEPLIDIS; DENAZIS, 2015) (PAKZAD et al., 2016), deep packet inspection (ZHOU et al., 2010) (BREMLER-BARR et al., 2014), quality of experience (GEORGOPOULOS et al., 2013), adaptive video distribution (JIN; WEN; WESTPHAL, 2015), and an intrusion prevention system (XING et al., 2014).

Despite its appeal, there are key factors that prevent the fast and broad adoption of PVNs, including (SANTOS et al., 2015): (*i*) network end-users (hereinafter called PVN owners) have to be able to match and understand the capacities of the network devices that host NetApps, to meet the requirements of different NetApps; (*ii*) PVN owners must themselves be able to sort out issues related to dependencies and conflicts among

the NetApps; (*iii*) the heterogeneous of execution environments force PVN owners to manually tune NetApps before running them; and (*iv*) NetApps are generally written with a specific network architecture in mind (*e.g.*, SDN and NFV) and, as a result, NetApps are not usually agnostic with regard to the underlying networking technology. However, so far, no effort has been carried out the deployment and management of network services within heterogeneous EEs has not been carryed out from the standpoint of the PVN owner. Hence, integrating VNAs or even NetApps with real-world production networks is an arduous task, mainly because the PVN owners must have an extensive knowledge of the device instructions so that they can deploy and manage them. Moreover, owing to a lack of NetApp documentation, several tasks (*e.g.*, transferring or configuring a new NetApp) have become extremely complex and repetitive.

In the recent past, vendors of mobile devices successfully overcame similar shortcomings by deploying online *marketplaces*. As the name suggests, these are platform-specific online infrastructures in which developers offer applications (or just apps), of which PVN owners make use. Although the first marketplaces targeted apps that were only provided by the vendors themselves (*i.e.*, third-party developers were not allowed), Google Play for Android (GOOGLE, 2018), and Apple App Store for iOS (APPLE INC, 2018) devices introduced a business model where third-party developers were able to offer apps to users of mobile devices.

As in the case of the mobile device market, we argue that recent advances in PVNs and network softwarization have the potential to open up the networking market to third-party developers as well. In our view, the successful model of today's marketplaces for mobile devices can provide an inspiration and benefits to the networking market. A marketplace can significantly accelerate the introduction of new NetApps, for instance, by automating tasks related to deployment and management. However, network softwarization has particular features that require online marketplaces to be targeted at PVN ecosystems, which include: network infrastructures based on heterogeneous paradigms (*e.g.*, Cloud, NFV, and SDN), diverse technologies (*e.g.*, OpenStack, OPNFV, and OpenFlow), and the fact that network services often combine different NetApps, which may cause dependency issues and/or conflicts. In light of this, the overall goal of this thesis is to investigate whether, from the standpoint of the PVN owner, it is feasible to have a platform that is inspired in the online marketplaces and can deploy and manage innovative NetApps regarding PVNs that contain heterogeneous EEs.

1.1 Hypothesis and Research Questions

In this thesis, the following hypothesis is raised to overcome the limitations discussed in the context of PVNs, particularly with regard to the deployment and management of NetApps.

***Hypothesis*: a platform inspired in online marketplaces may enable PVN owners with a limited knowledge of the underlying infrastructure to overcome the main hurdles to introducing new NetApps into PVNs**

In the past, the main reason why network programmability solutions were not adopted, was that they raised concerns about security (MOORE; NETTLES, 2001). There was a risk that the settings and NetApps deployed on the devices could carry out harmful activities or have bugs, even if the EE was correctly implemented. Integrating virtualization and programmability ensures isolation among VNs, and this prevents these kinds of misbehaving programs (or even devices) from interfering with the production network; however, it also imposes constraints on performance. Although this is a shortcoming, it has not prevented new NetApps from being designed. Despite of this, even today, only a tiny number of NetApps are deployed in PVNs. Thus, our first research question can be defined as follows:

Research Question #1: *What are currently the main hurdles to introducing new network applications in Programmable Virtual Networks?*

We will answer this Research Question #1 in Chapter 2, in which we determine the main factors that hamper the introduction of a new NetApp within PVNs. Among these factors, it should be noted that the PVN owner must have an extensive knowledge of device instructions and NetApps, before being able to deploy and manage new NetApps. As a result, simple tasks, such as transferring or configuring a new NetApp, can become extremely complex and repetitive. Moreover, the initial settings must be manually replicated in each EE to set up the logic rule for delegating the flow of packets to each new service. All this makes integrating VNAs or even NetApps with production networks, an arduous task. In our view, a platform that helps to deploy and manage NetApps could simplify the processes described above, and improve the way that PVN owners adopt them.

A short time ago, the market for mobile applications was closed, and only mobile vendors were able to develop new applications for cell phones. However, as a result of the introduction of Google Play Store (GOOGLE, 2018) for Android and Apple App Store (APPLE INC, 2018) for iOS, the market for mobile applications was opened up. This has allowed third-party developers to program and offer a wide range of new mobile applications directly to the PVN owners. Moreover, these PVN owners are able to install and manage distinct apps, even if they do not know the particular features of cell phones. Similarly, the networking market was closed and proprietary; however, recent advances in PVNs have allowed the networking market to be opened up to third-party developers too and, furthermore, today, device vendors and academia are in a position to recommend some marketplaces to PVN owners (JUNIPER NETWORKS, 2016a) (HPE, 2016) (RAMOS et al., 2014) (PINHEIRO et al., 2014). However, these marketplaces are technology-dependent and in nearly all cases, do not support a carefully detailed NetApp descriptor (*i.e.*, one that supports relationships between NetApps such as conflicts and dependencies). In our view, we could apply the successful paradigm of mobile app stores in the networking market, and this could help PVN owners, who are ignorant of the underlying infrastructure, to deploy and manage NetApps and VNAs in their PVNs. All this leads on to our second research question which is as follows:

Research Question #2: *In light of the successful paradigm of OS-specific app stores, what are the essential design goals that can enable PVN owners, who lack any knowledge of the underlying infrastructure, to deploy and manage the network applications?*

We will answer this Research Question #2 in Chapters 3 and 4. First of all, in Chapter 3, we will determine and outline what marketplaces there are for different contexts, as well as their essential design goals, and open research challenges. This includes a historical and comprehensive roadmap of networking paradigms for a better understanding of the online marketplaces and to show how paradigms and technologies have evolved over the years. Another reason for discussing current online marketplaces is to define the essential design goals and the research challenges that must be overcome to make the network marketplaces a reality. In light of the design goals and open research challenges, we propose App2net ecosystem as a means of enabling PVN owners to deploy and manage NetApps in PVNs with heterogeneous EEs. In addition, the App2net ecosystem enables third-party developers to publish and describe the NetApps using several programmability-related

technologies.

We divide our ecosystem into two main platforms, namely Marketplace for PROgrammable Virtual nEtworks (`iMPROVE`) and Applications to Network (`app2net core`). The `iMPROVE` is a network marketplace for different technologies, such as NFV, OpenFlow-based, and VyOS. This marketplace allows PVN owners to select, download, and trigger the deployment of new network services. We extend the ETSI Network Service Descriptor (NSD) (NFV ISG, 2014) to cover these processes and provide a detailed account of the different technologies of NetApps and VNAs, as well as to describe issues of conflict. To complete the lifecycle of a NetApp, we also introduce the `app2net core` for carrying out processes in the NetApps. In this way, `app2net core` can assist PVN owners to transfer, install, and configure the NetApp package among heterogeneous EEs in a transparent way. Moreover, we design different transfer models based on a set of code transfer techniques. These models allow us to transfer NetApp packages and configuration files among PVN nodes to achieve different goals. Although App2net simplifies the deployment and management of NetApps from the standpoint of the PVN owner, it is essential to evaluate the impact of our ecosystem and determine whether or not it is acceptable. All this forms the basis of our third research question:

Research Question #3: *Considering the distribution time, CPU usage, and network overhead, what is the impact of using a platform on deploying applications in heterogeneous Programmable Virtual Networks?*

To address this Research Question #3, in Chapter 5, we will make two different evaluations of our App2net ecosystem. In the first of these, we conducted a case study to provide a detailed description of the interactions and outline the necessary activities for a developer when publishing a network service in the App2net ecosystem. Next, we will explore the interactions of a PVN owner, who desires to deploy this network service in his/her PVN. In this case study, it is possible to check if our ecosystem is feasible and whether or not it has enough simplicity to allow PVN owners, without any knowledge of its underlying infrastructure, to deploy and manage NetApps for a PVN with different EEs. Thus, we also discuss the management of this new network service by using the App2net RESTful API. In the second evaluation, we implement and test a prototype through realistic network topologies that are commonly found in the Internet. The results allow us to determine which models can improve the code transfer. Following this, we

also evaluate the impact introduced by our App2net ecosystem in terms of the distribution time, CPU usage, and network overhead.

1.2 Thesis Roadmap

The remainder of this thesis is structured as follows.

- In **Chapter 2**, we review the research studies on network virtualization and programmability technologies. This is followed by a comparison of some of the features of the programmability-related technologies. Finally, we compile and introduce a glossary for this thesis. This glossary brings together the terms that are related to the same concepts but found in different technologies.

- In **Chapter 3**, we discuss about current online marketplaces in different areas. From a historical perspective, we point out the essential design goals and main stakeholders. We also disccus the main research challenges that must be overcome to allow the adoption of marketplaces by future networks. Finally, we review the current landscape of marketplaces on the basis fo the previously identified design goals.

- In **Chapter 4**, we introduce the App2net ecosystem, which is split into two main platforms, namely `iMPROVE` and `app2net core`. Throughout this chapter, we extend the ETSI NSD to describe the particular features of NetApps within a large number of technologies as well as examining issus of dependency and conflict. Next, we specify a RESTful API to enable external software elements (*e.g.*, another system, platform, module, script, or network service) to request data, trigger actions, and send notifications and messages. Following this, we analyze a set of code transfer techniques; and, then we design different models for code transfer based on the identified techniques. Finally, we discuss our ecosystem with regard to the design goals and research challenges outlined in Chapter 3.

- In **Chapter 5**, we carry out two main evaluations of the App2net ecosystem. First, we conduct a case study to describe the main tasks of a third-party developer when distributing and describing a network service for different technologies. We also explain the main interactions of a PVN owner that wants to deploy and manage a network service in a PVN with EEs of several technologies. In the second evaluation, we implement a prototype and evaluate it through the realistic network topologies

that are generally found in the Internet. The results make clear the impact introduced by App2net in terms of distribution time, CPU usage, and network overhead.

- In **Chapter 6**, we summarize some of the conclusions about matters related to this thesis. Following this, we examine in detail and explain the answers to the research questions. Next, we summarize the main contributions of this thesis. Finally, we make suggestions for further research in this area.

- The **Appendix** includes a list of publications that show the results achieved in the development of this thesis. This list also contains the main collaborations carried out in papers that focus on related subjects.

2 PROGRAMMABLE VIRTUAL NETWORKS: A HISTORICAL PERSPECTIVE, TECHNOLOGIES, AND CURRENT SHORTCOMINGS

This chapter[1] presents an outline of the main research topics related to this thesis. First, in Section 2.1 we describe the network virtualization and its benefits and, then, we review the main technologies employed to create virtual networks. Next, in Section 2.2, we provide a historical perspective of the area of network programmability, by examining the kinds of technology which have evolved over time. Following this, we compare some features of these technologies in so far as they are related to the management and deployment of NetApps in heterogeneous PVNs. Finally, in Section 2.3, we summarize the chapter and compile a glossary for this thesis, which brings together terms related to the same concepts but found in different technologies.

2.1 Network Virtualization

As mentioned in the previous chapter, the scientific community regards network virtualization as a feasible means of proposing, investigating, and deploying new features and solutions in computer networks, by making them an integral part of the next-generation networking paradigm. In fact, there is no general consensus about how network virtualization should be defined and several definitions have been put forward (ANDERSON et al., 2005) (ESTEVES; GRANVILLE; BOUTABA, 2013) (MERWE; KALMANEK, 2009) (KHAN et al., 2012). Wang *et al.* discuss some of these definitions and propose one of their own that can combine different perspectives (WANG et al., 2013):

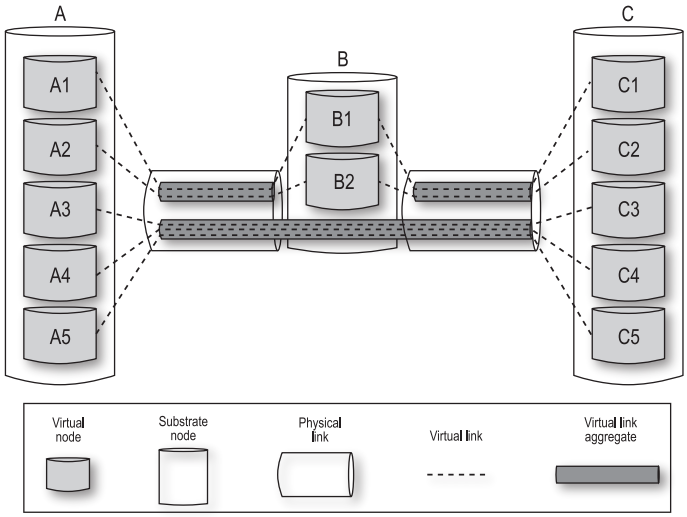
> Network virtualization is any form of partitioning or combining a set of network resources, and presenting (abstracting) it to users such that each user, through its set of partitioned or combined resources has a unique, separate view of the network. Resources can be fundamental (nodes, links) or derived (topologies), and can be virtualized recursively. Node and link virtualization involve resource partition/combination/abstraction; and topology virtualization involves new address (another fundamental resource) spaces.

It was decided to base this thesis on the above definition because it covers a broad spectrum of current network virtualization technologies. These technologies have been grouped together in the following three categories: link virtualization, node virtualization, and virtual networks (CARAPINHA; JIMÉNEZ, 2009) (WANG et al., 2013). Figure 2.1

[1]This chapter is based on the following paper: "A Survey on Programmable Virtual Networks: Where Did We Come From, and Where Are We Going To?". For more information about this paper see the Appendix.

provides a simple view of the main elements of a virtual network.

Figure 2.1: Main elements of a virtual network.



Source: adapted from Carapinha and Jiménez (CARAPINHA; JIMÉNEZ, 2009).

Link virtualization allows multiple separate virtual links to share the same physical link (substrate node). On the basis of this definition, there are many ways to create virtual links. First of all, there are multiplexing technologies (*e.g.*, Time Division Multiplexing - TDM, Frequency Division Multiplexing - FDM, and Wavelength Division Multiplexing - WDM), which are widely used to split a single physical link into separate virtual links. Likewise, physical links can be combined to create a single high-speed virtual link (*i.e.*, inverse multiplexing) while some virtual links can be grouped into a virtual link aggregation. In addition, virtual links can be created by techniques that allow the packets to be tagged (*e.g.*, Virtual Local Area Network - VLAN) and by tunnels that connect the devices that are not physically adjacent (*e.g.*, Generic Routing Encapsulation - GRE) (CHOWDHURY; BOUTABA, 2010).

Node virtualization involves the isolation and partitioning of hardware resources. The physical resources of a substrate node (*e.g.*, CPU, memory, and link bandwidth) are partitioned per virtual node in accordance with a set of requirements. In most cases, two devices are virtualized: (*i*) Network Interface Cards (NICs), at the network edge; and, (*ii*) routers, in the network core (WANG et al., 2013). VMware, QEMU, Citrix XenServer, and Oracle VirtualBox share NIC hardware through a type of emulation software called virtual NIC (vNIC). Hence, each virtual machine has an assigned vNIC and a virtual switch connects several vNICs to a physical NIC. Single-root I/O virtualization (SR-IOV) is a hardware enhancement that replaces the virtual switch, and thus provides better throughput and scalability, and reduces CPU usage (DONG et al., 2010).

There are two approaches that can be adopted to virtualize routers. In the first of these, Linux-based network operating systems (*e.g.*, Mikrotik, VyOS, Vyatta, and Quagga), which provide software-based routing solutions, are installed in x86 and x86-64 virtual machines. It should be noted that these virtual machines can be hosted on common hypervisors (*e.g.*, VMware, Citrix XenServer, and Oracle VirtualBox) that run on COTS servers. The second approach involves physical routers that support hardware partitioning and are able to host a number of routing instances in a single device. These routing devices are called logical routers by Cisco (CISCO, 2008) and protected system domains by Juniper Networks (JUNIPER, 2012).

Virtual networks are collections of virtual network devices that are connected through virtual links. Three kinds of collections are commonly implemented, namely Overlay Networks, Virtual Private Networks, and Virtual Sharing Networks (CARAPINHA; JIMÉNEZ, 2009). An overlay network is built on an existing network, largely through tunneling and encapsulation technologies. Overlay networks have often been created to provide new services for an existing infrastructure such as Internet access networks (Digital Subscriber Line - DSL), multicast (Multicast Backbone - MBone), and IPv6 (IPv6 Backbone - 6Bone). A Virtual Private Network (VPN) is a network built on the infrastructure of a public network. That is, instead of using dedicated links to connect remote devices, it uses the Internet infrastructure. VPN gives each node the illusion that there is a direct connection to another node. Many companies deploy VPNs to connect their offices to geographically distant areas, whereas employees or clients can use a VPN to gain access to their company's internal network remotely. VPNs can be created in different layers such as Layer 2 VPN (L2VPN) and Layer 3 VPN (L3VPN) which are widely deployed; but also in Layer 1 VPN (L1VPN).

Virtual sharing networks entail the use of technologies (*e.g.*, VLAN or VPN) that allow physical resources to be shared with multiple virtual networks (VNs) instances while causing segmentation between them. For example, in a university, there are two VNs: students and teachers. Students can only have access to the file server whereas teachers have access to the Internet, file server, and mail server. In this way, two VNs can share physical routers and servers. However, these networks are segmented and have limited connectivity to students. In this example, two VNs form one virtual shared network because they share the same physical infrastructure, although they are separated from each other. It is worth mentioning that a VN can form a part of more than one category, thus, in the example outlined above, the VNs are classified in virtual private networks too.

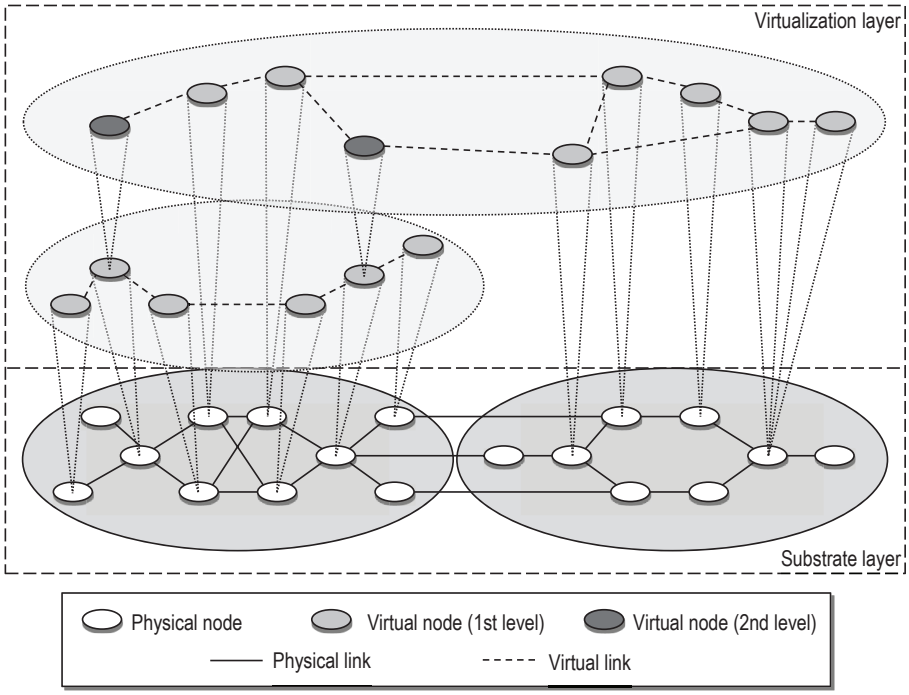
According to Chowdhury and Boutaba (CHOWDHURY; BOUTABA, 2010), a networking environment supports network virtualization if it allows the coexistence of multiple VNs within the same physical infrastructure (also known as the substrate layer). Each VN in a network virtualization environment (NVE) is a collection of virtual nodes and virtual links. In other words, a virtual network is a subset of resources from the substrate layer, which hosts the key virtual elements.

In Figure 2.2, there is a generic model of a network virtualization environment (ESTEVES; GRANVILLE; BOUTABA, 2013). The white ellipses in the diagram represent physical elements, while the light gray (first level) and dark gray (second level) ellipses represent virtual nodes. The continuous lines indicate the existence of a physical link whereas the dashed lines indicate the virtual links between the virtual nodes. As can be observed, the VN is a complete topology because it has all the elements that comprise a production network such as routers, hosts, and links. The virtual routers are usually connected to each other by IP tunnels that allow the use of a large number of application protocols, transport protocols, routing algorithms, and other features included in the current Internet (TOUCH et al., 2003). Network virtualization enables a single physical component to form multiple VNs and at the same time, allows a VN to host the nodes of another VN. Thus, novel network architectures can be created without significantly interfering with the substrate layer, and also provide a high degree of flexibility to the virtualization layer.

According to Anderson *et al.* (ANDERSON et al., 2005) and Merwe and Kalmanek (MERWE; KALMANEK, 2009), the network virtualization has several benefits and provides the scientific community with the means to overcome the problem of "Internet Ossification" (CHOWDHURY; BOUTABA, 2010). Below, there is a discussion of the benefits of network virtualization and how it requires a better way to introduce network programmability:

- **Isolation** - Network virtualization ensures that there is isolation between NVE and the physical network resources, as well as between the VNs that share the same NVE. Thus, it is possible to host multiple VNs within a single physical network without affecting the network operations or interfering with the existing services. This isolation is of paramount importance when deciding how to introduce the network programmability. In the past, some solutions were rejected because misbehaving programs could lead to a collapse of the production network. Hence, the use of a VN to support a programmability technology would reduce the degree of

Figure 2.2: A generic network virtualization environment.



Source: adapted from Esteves *et al.* (ESTEVES; GRANVILLE; BOUTABA, 2013).

interference from misbehaving programs within the production network.

- **Flexibility** - A virtual network node can be created, moved, and deleted without disturbing other virtual nodes. Moreover, in the case of hardware failure or system overloading, virtual nodes can migrate to another physical resource in a transparent way, in accordance with the policies defined by operators. This means the PVN owners can deploy and run their programs on virtual nodes. Thus, if new nodes could be dynamically created, they would provide the PVN owners and their programs with more resources. As well as this, the virtual nodes can be moved to another server in case of failures, and thus offer the PVN owner a more resilient solution.

- **Availability** - Restarting and updating a given network service can be performed in one virtual node without affecting other network services in the same NVE. In an environment without virtualization, the update procedure affects all the users and traffic in the network. Simple incremental updates are almost impossible since there are conflicting interests between both the users and the Internet Service Providers (ISPs). By supporting both virtualization and programmability, the PVN owners can instantiate services which are of value to them, including different versions of the same services that are used by other PVN owners without any interference.

- **Scalability** - Initially, virtual network nodes only provide a small set of features, but, as new demands arise, other features can be added, which results in a more robust network architecture. A programmable platform can add new services in an incremental way, and thus avoid the need for a further implementation of network services while at the same time, enhancing the use of the NetApps that have already been designed. It is worth noting that this kind of platform would make it possible to examine and overcome issues of dependency and conflict among the NetApps, and thus avoid the problem of PVN owners having to carry out new and complex tasks.

- **Reduced Costs** - A single physical resource, like a LAN port, can be shared by several VNs as this avoids the need to acquire dedicated equipment for each service and, hence, reduces the CAPEX and OPEX. This is important because at the outset, the devices of programmability technologies must be replaced by new ones in the whole network, and this leads to prohibitive costs. With the introduction of virtualization, no changes are required at the substrate layer; all that is needed is for the virtual nodes to be replaced to support the new technologies, and thus make it easier to carry out the update procedure of the devices.

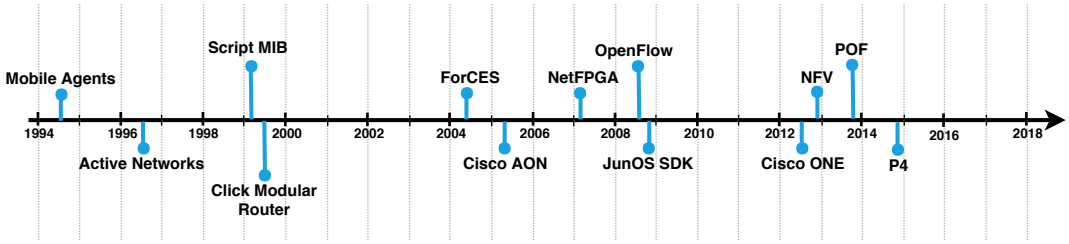
2.2 Network Programmability

In the 1990s, DARPA recognized the shortcomings of IP networks, which were later called “Internet Ossification” (FEAMSTER; REXFORD; ZEGURA, 2014). To overcome this problem, the scientific community coined the term “network programmability” to define the ability of a network device (*e.g.*, router or switch) to run a program on the network. This ability allows new features and services to be developed (*e.g.*, new protocols, management solutions, monitoring systems, and deep packet inspection) without having to replace the physical hardware (MERWE; KALMANEK, 2009). What is of particular importance, according to Campbell *et al.* , is that network programmability has the ability to build, deploy, and manage new services in the network at great speed (CAMPBELL et al., 1999).

Today, academia, companies, and standardization bodies have acquired enough technological skills to explore and design new services, protocols, and network architectures. Since the 1990s several programmability-related technologies have emerged with the

same goal (such as Active Networks (TENNENHOUSE et al., 1997), Mobile Agents (CHESS; HARRISON; KERSHENBAUM, 1997), and Script MIB (LEVI; SCHOENWAELDER, 1999)). However, these technologies were prevented from being adopted by two constraints: (*i*) the lack of safeguards or guarantees about the settings and applications deployed in the devices, which could interfere with, or lead to a collapse of, the production network; and (*ii*) the increase in CAPEX and OPEX of the ISPs caused by the continuous need to replace the network devices that underpin these technologies. Below, in Figure 2.3, we summarize the main programmability-related technologies that have emerged over the years.

Figure 2.3: Timeline of network programmability technologies.



Source: the author (2018).

As previously mentioned, the introduction of virtualization provides several benefits to computer networks, since it gives rise to new ideas about the network programmability. Moreover, it means that network devices can only be replaced in a virtual way. Thus, new technologies can be introduced without any changes in the physical devices, and thus reduce the CAPEX and OPEX of the ISPs. In addition, the isolation enables VNs to coexist with other VNs and the physical network and, in case of failures or misfunction, these VNs or specific virtual nodes can be stopped or removed. In this way, they will affect neither the other VNs nor the physical network. This means that the programmability can be reintroduced and empower developers to create innovative services for the Future Internet. In the next section, there is a description of the main programmability-related technologies and their features.

2.2.1 Improved Nodes

Traditionally, the primary purpose of the network has been to deliver packets from their source to the destination host without changing the payload of the packets. The network devices check the headers, perform the routing, and correct any errors. The aim

of the first programmability-related technologies was to open up and improve the network devices. This means they define the elements that can enable the network nodes to run a program. In these technologies, the programs were transmitted by the network or else transferred to the specific nodes, in which they were executed. Three key technologies are described below that allow the network to be programmed in this way: Active Networks, Mobile Agents, and Script MIB.

Active Networks

In 1995/1996, the concept of "active networking" emerged from discussions within the broad DARPA research community about the future direction of networking systems (AUBRECHT; KOUTNY, 2012). According to Tennenhouse *et al.*, **Active Networks** represent a new approach to the network architecture. A network becomes active in two situations: (*i*) when the network devices (*e.g.*, routers or switches) can execute custom programs on the packets that flow through them; and, (*ii*) when users can inject their custom programs into the routers which examine the packets and run the appropriate program. In an extreme case, there will be no difference between the internal and end-user nodes since, if necessary, both will be able to run the same programs (TENNENHOUSE et al., 1997).

Active Networks allow new services to be developed as well as improving their existing functionalities and performance. Various applications can be enhanced such as network management, web proxies, reliable and efficient multicasting, and active caching (PSOUNIS, 1999). Essentially, active networks adopt three approaches: integrated (active packets), discrete (active nodes), and hybrid. In the integrated approach, every message, or capsule that travels between the nodes, contains a fragment of a program (of at least one instruction) that may include embedded data. Thus, the program is installed when it is used. In the discrete approach, the packets only contain the identifiers of the programs (routines) that must be run. In this way, the active applications are installed into the node on demand by means of a download scheme. In the hybrid approach, a network based on the integrated approach can be programmed to emulate the discrete approach and vice versa. In other words, programs can either be encapsulated in packages or be installed via download (TENNENHOUSE; WETHERALL, 2002). There are several significant studies on active networks: (*i*) SwitchWare architecture which addresses security concerns (ALEXANDER et al., 1998), (*ii*) ANTS toolkit that allows new protocols to

be developed (WETHERALL; GUTTAG; TENNENHOUSE, 1998), (*iii*) Smart Packets for network management (SCHWARTZ et al., 1999); and, (*iv*) PLAN, a programming language for management applications (HICKS et al., 1998).

Mobile Agents

An agent is a type of software that is able to carry out tasks in an execution environment as well as to learn and cooperate with other agents. In addition, an agent is autonomic, proactive, and reactive (LANGE; MITSURU, 1998). According to Chess *et al.*, **Mobile Agents** are programs, that are generally written in a scripting language, which may be dispatched from one node and then transported to another for execution. In this way, a mobile agent is independent of the execution environment which it sets out from, since it is able to travel to the nodes in the network (CHESS; HARRISON; KERSHENBAUM, 1997).

After it has been created, the mobile agent takes its "state" and "code" with it to another execution environment, where it resumes its task of executing of activities. In this context, the word "state" refers to the attribute values of the mobile agent that determine which actions must be performed and what the next node will be. The "code" relates to the routines that are necessary for execution. It should be noted that the transfer of a mobile agent can be either partial or complete depending on the requirements for execution (BIESZCZAD; PAGUREK; WHITE, 1998). Mobile agents offer a wide range of benefits to applications, including the following: a reduction of network traffic and latency, fault tolerance, encapsulation of protocols, and asynchronous and autonomous execution (LANGE; OSHIMA, 1999).

In the past, mobile agents were studied as a viable alternative for programming the network. However, they raise serious concerns about security and performance which means that their use is avoided (CHESS; HARRISON; KERSHENBAUM, 1997). Today, mobile agents are attracting a good deal of attention but in different areas such as sensor networks (LIN et al., 2012) (IYENGAR; BROOKS, 2016), vehicular cloud computing (GERLA, 2012), and smart grids (BERA; MISRA; RODRIGUES, 2015).

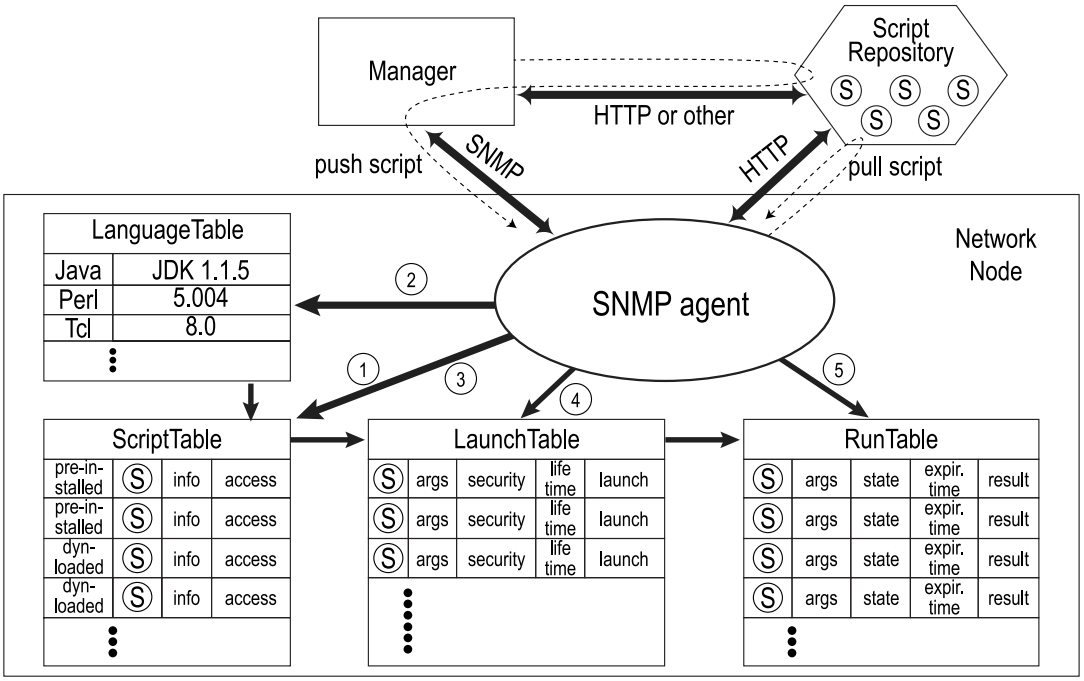
Script MIB

The Management Information Base (MIB) is a set of managed objects that represents

the entities in a computer network, that allow a large number of management tasks to be automated. Every object is an abstract view of a real resource of the system. Hence, all the network resources that must be managed are modeled, and the resulting data structures represent the managed objects. Users or managers can read and change these objects in compliance with specific rules that grant permission (LEVI; SCHOENWAELDER, 1999).

The **Script MIB** module defined in RFC 2592 (LEVI; SCHOENWAELDER, 1999) and RFC 3165 (LEVI; SCHOENWAELDER, 2001) can delegate and install management functions (also known as "scripts") in the network nodes. This MIB makes no requirements about the programming language; thus, these scripts can be written in any programming language that is supported by the Script MIB implementation. As a result of the SNMP requests, the manager can carry out several operations: (*i*) transferring scripts; (*ii*) managing (*i.e.*, installing, suspending, summarizing, and terminating) scripts; (*iii*) sending parameters; (*iv*) monitoring and controlling the execution; and, (*v*) requesting information about the results obtained.

Figure 2.4: A network node supporting the Script MIB.



Source: adapted from Schönwalder *et al.* (SCHONWALDER; QUITTEK; KAPPLER, 2000).

When running and controlling a script in Script MIB (as shown in Figure 2.4), the manager first checks if the desired script has been installed in the node, which means finding out if the ScriptTable has a record of this particular script (step 1). When installing a script, the manager checks the available programming languages in the LanguageTable (step 2). Additionally, the manager also checks the installed extensions in

the ExtensionTable if necessary. Next, the manager stores the data about the script in the ScriptTable (step 3). At this time, the transfer can occur through two methods: “pull” and “push’. In the pull method, a node receives the URL repository, and then, downloads the script through the HTTP protocol. In the push method, a manager sends the script to a node via SNMP messages. Following this, the node stores the received script in the CodeTable. After the installation, the manager sends the settings to the node which stores it in the LaunchTable (step 4). At this stage, a manager can start the script execution. Finally, the node stores the results in the RunTable (step 5) and the manager can request this data during a time interval.

Although the Script MIB provides flexibility with regard to the programming languages used, transfer methods, and ways of collecting data from the execution of the distributed management scripts, it has serious limitations that prevent its wide adoption as a form of network programmability technology (SCHONWALDER; QUITTEK, 1999) (STRAUß, 2000). These include the following: (*i*) it focuses solely on distributed management, which makes it difficult to program applications with other purposes (*e.g.*, video transcoding and multicast); (*ii*) administrators have been discouraged from using third-party scripts because they must grant root access to install and run the management scripts in their devices; (*iii*) a lack of isolation among new applications and production networks, which, in the case of a misbehaving code, could interfere with the whole network or lead to its collapse; (*iv*) the management of scripts is a very complex task because distributing, installing, and updating scripts in several devices require a considerable effort to ensure they are synchronized and made usable; and, (*v*) security issues primarily related to the integrity of the scripts, the traffic of sensible data, and the control to run scripts.

2.2.2 Software Routers

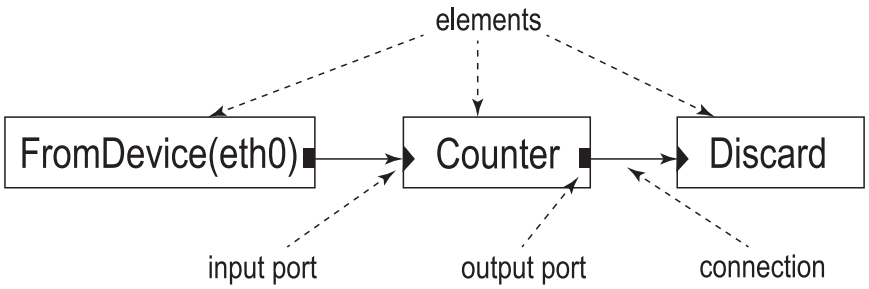
Software routers perform packet processing in software (BOLLA; BRUSCHI, 2007) (FALL et al., 2011). Moreover, they are easily extensible; in other words, developers can program and supplement new functionalities that can provide new services or support other technologies. These software routers often consist of Linux-based network operating systems installed in x86/x86-64 virtual machines and hosted in COTS servers (BOLLA; BRUSCHI, 2008). There are several implementations of software routers, for instance, Mikrotik, Cisco CSR 1000v, Juniper vMX, VyOS, Vyatta, ClickOS, Quagga, and Click Modular Router. Although these examples can be programmed, we wish to

draw attention to the Click Modular Router because of its popularity and modularity.

Click Modular Router

Click Modular Router runs C++ small modules (also known as elements), which perform simple router functions such as the reduction of the TTL of the packet, packet classification, queuing, scheduling, and interfacing with a network device. Several elements are grouped and linked, to create a graph of the elements that represents a software router (MORRIS et al., 1999). In this graph, the vertices are the elements and the edges describe the path of the packets. Figure 2.5 shows a simple example of a router that first counts the incoming packets, and then discards them. The *FromDevice(eth0)* element receives the packets from the *eth0* interface and forwards them to the following element. After this, the *Counter* counts the number of packets. Finally, the *Discard* element is responsible for dropping the packets.

Figure 2.5: A simple example of a router in Click.



Source: adapted from Kohler *et al.* (KOHLER et al., 2000).

Click has three operational modes: simulation, user level, and kernel (KOHLER et al., 2000). In the simulation mode, the click router works as an NS3 or NS2 module, called NSClick. In the second mode, the user level, the click router runs as a user process in a Linux system. Finally, in the kernel model, the Linux system is modified to enable it to forward the received packets directly to the click router and then to the network interfaces. Initially, Click was proposed as a single-threaded application, but currently, there are parallel implementations in the kernel mode that reach up to 40 Gbps of packet processing throughput (FALL et al., 2011). Although Click provides a custom language service to specify the connections and dependencies among the elements, there is no support to describe any conflicts between them.

2.2.3 Customized Hardware Routers

Some solutions focus on the growing demand for high performance to packet processing, and thus, they enable network operators and engineers to built up a customized hardware router. In such solutions, a "developer" can modify all architecture of a router since increasing memory buffers until implementing new protocols to L2/L3 (in hardware level). Following, we describe the NetFPGA because it is the main solution in this context.

NetFPGA

NetFPGA is the most popular open-source software and hardware platform for a fast prototyping of networking hardware (*e.g.*, modified Ethernet switches and IP routers). NetFPGA uses a programmable NIC based on Field Programmable Gate Array (FPGA) to run programs designed by multiple users such as researchers, teachers, or students (GIBB et al., 2008). The users implement their programs in Verilog code, where they are called modules. A pipeline organizes the system in several stages and this allows these modules to be run. When a packet arrives, the NetFPGA processes it at each stage of the pipeline. It should be noted that the stages are interconnected by means of two buses: the packet bus and the register bus. The packet bus transfers the packets from one stage to the next through a synchronous FIFO packet-based push interface. The register bus provides another channel of communication that does not consume the bandwidth of the packet bus. This allows that the information travels in both directions through the pipeline but with a much lower bandwidth (NAOUS et al., 2008).

NetFPGA enables a wide range of users to create new prototypes in a modular way. As a result, users can reuse, modify, and add new functionalities to pre-built modules, or create a new one without the need for any pre-built modules (ZILBERMAN et al., 2014). Several modules are available online for free download[2]. Some examples are OpenFlow switch, common IPv4 and IPv6 routers, and monitoring systems. A solution for installing or even arranging pre-built modules into new service modules, could ease the management of these modules and thereby improve the adoption of NetFPGA as a programmable solution in experimental networks; however, it is a very laborious task, and

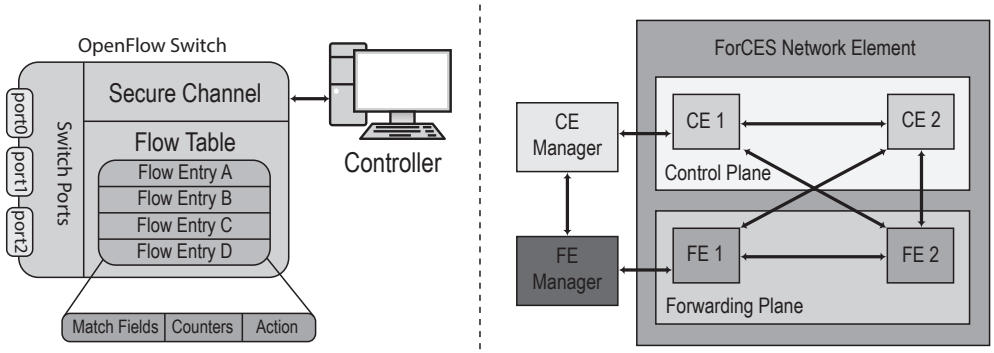
[2]See the projects list for each version at the NetFPGA site: <http://netfpga.org/site/#/systems/4netfpga-1g/applications/>

there are no studies in the literature that deal with this issue. Despite its flexibility and modularity, some significant concerns have been expressed by the NetFPGA community, such as high latency, high memory usage, and the bandwidth restrictions of the registered bus (NAOUS et al., 2008).

2.2.4 Software-Defined Networking

Software-Defined Networking (SDN) is an emerging network architecture in which network control (the control plane) is decoupled from forwarding functions (the data plane). This enables the network control to be directly programmable and the underlying infrastructure to be abstracted for applications and network services (FEAMSTER; REXFORD; ZEGURA, 2014) (ONF, 2016). SDN provides programmability through well-defined interfaces, and can lead to innovative network applications such as new protocols, network management solutions, server load balancing, and energy-efficient networking (XIA et al., 2015). In the next subsections, there is a review of two well-known SDN architectures, namely OpenFlow (MCKEOWN et al., 2008) and ForCES (YANG et al., 2004). Figure 2.6 shows the main elements of both architectures OpenFlow (left) and ForCES (right). Due mainly to its disruptive characteristics (see below), ForCES is not widely adopted as the OpenFlow (XIA et al., 2015) (HARES, 2013).

Figure 2.6: OpenFlow architecture (left) and ForCES architecture (right).



(a) OpenFlow (b) ForCES

Source: adapted from McKeown *et al.* (MCKEOWN et al., 2008) and Yang *et al.* (YANG et al., 2013).

OpenFlow

OpenFlow emerged in 2008 as a means of allowing researchers to run experimental

protocols in the campus networks (MCKEOWN et al., 2008). Essentially, OpenFlow consists of four key elements: (*i*) the controller, a remote control entity that makes the traffic forwarding decisions, and adds or deletes the flow entries to determine the path of the flows across the network; (*ii*) a flow table which is a set of several flow entries; each entry consists of match fields (used to match the incoming flows), an action (this defines how to handle matching flows), and counters (used to collect statistics for each flow); (*iii*) a secure channel, this connects switches to the controller, and ensures that commands and packets are sent in a safe and reliable way; and, (*iv*) the OpenFlow protocol, which provides an open and standard way for the controller to communicate with the switches.

When a packet arrives, the OpenFlow switch extracts the packet headers and matches them with the flow table. If a matching entry is found, the switch handles the flow by means of an action in the flow entry. If no flow entry matches the packet headers, the switch handles the flow in accordance with a table-miss flow entry. The "table-miss" defines which actions must be performed when no match is found for the flow. The possible courses of action include dropping the flow, analyzing the next flow table, or sending the packet to a controller via the secure channel. The controller uses the applications created by operators, to determine which actions must be performed in this flow, for example, forwarding to a specific port, dropping, modifying a header, or forwarding the flow to the controller. After this, the controller adds, updates, or deletes the flow entries in the switch flow table (ONF, 2015).

It should be noted that the OpenFlow controller can handle switches from different network vendors, including virtual and software routers. Thus, several studies in both academia and industry have drawn on OpenFlow and SDN concepts. In the first place, there are many controllers with different features (MACEDO et al., 2015) such as NOX - easy learning (GUDE et al., 2008), Onix - scalability and reliability (KOPONEN et al., 2010), Beacon - dynamic application management (ERICKSON, 2013), and OpenDaylight - which gives support to several protocols (MEDVED et al., 2014). In addition, there are many applications in the literature such as reliable multicasting (SHEN et al., 2015), management and visualization solutions (ISOLANI et al., 2015), adaptive video distribution (JIN; WEN; WESTPHAL, 2015), and an intrusion prevention system (XING et al., 2014). However, owing to the characteristics of OpenFlow and the performance of the controller, applications that focus on data plane and packet data processing (*e.g.*, deep packet inspection, cache, and transcoding) are beyond the scope of the OpenFlow controller (NAKAO, 2013) (KUKLIŃSKI, 2014) (MACEDO et al., 2015).

ForCES

In 2004, IETF proposed the **Forwarding and Control Element Separation (ForCES)** Framework to redefine the architecture of network nodes by applying the concepts of SDN (control and data plane separation and abstraction of physical elements) (YANG et al., 2013). A ForCES network node consists of multiple Forwarding Elements (FEs) and multiple Control Elements (CEs). FEs use the underlying hardware to provide per-packet processing. In turn, CEs executes control/signaling functions and employs the ForCES protocol to instruct FEs how to handle packets. Although FEs and CEs can be combined in a single network device, this is not a requirement, which means these elements can be physically separated. In addition, FE and CE managers are proposed. The first determines which CEs a FE should communicate (CE discovery). Similarly, the CE manager determines which FEs a CE should communicate (FE discovery) (HALEPLIDIS et al., 2015).

A particular characteristic of ForCES is the "pre-association phase", when an FE manager and a CE manager defines which FE(s) and CE(s) should be a part of the same network element (CE discovery and FE discovery). Packet forwarding in an FE is based on the abstraction of logical function blocks, which can be dynamically specified to create new packet processing activities (HALEPLIDIS, 2015) (SALIM, 2015). ForCES provides a more flexible forwarding model, redundancy, CE-FE discovery, and additional mechanisms for security. However, owing to the fact that the business model for the logical function blocks is disruptive and there is a lack of support for open source, ForCES is not widely adopted as an OpenFlow (XIA et al., 2015) (HARES, 2013).

P4

In the area of SDN, the aim of programmable data planes is to provide more flexible programmability because they are protocol-independent (COSTA CORDEIRO; MARQUES; GASPARY, 2017). In this context, we would like to highlight two key solutions, namely P4 (released in 2014) (BOSSHART et al., 2014) and POF (which appeared in 2013) (SONG, 2013). **Programming Protocol-Independent Packet Processors (P4)** enables developers to specify in detail how the network forwarding elements of the data plane will process the packets. The abstraction model supports a programmable parser,

which helps developers to define the new headers that will be used to create the match actions for the packets. In addition, the model assumes that actions consist of protocol-independent primitives supported by distinct forwarding elements (WANG, 2017). In this way, developers create programs using the P4 language and, then, compile these programs for each forwarding element. The P4 compiler takes the capabilities of these elements into account to turn a device-independent program (written in P4) into a device-dependent program (used to configure the forwarding elements). Thus, developers do not need to know the specifications of the underlying forwarding elements concerning the device type (*e.g.*, switch or router) or even the technological systems (*e.g.*, ethernet switches, software switches, FPGAs, smart NICs or fixed-function switch Application Specific Integrated Circuits (ASICs)) (DARGAHI et al., 2017).

P4 provides two kinds of operations to control the data plane: configure and populate. Configure operations program the parser, order the stages of the match actions, and specify the header fields processed by each stage. In that way, these operations make it possible to determine the supported protocols and show how the forwarding elements will process packets (MARTINS et al., 2018). Populate operations add and remove the entries in the match action tables which were specified during the configuration phase. Note that P4 creates the match actions by means of five primitives (BOSSHART et al., 2014): (*i*) the `set_field`, this sets a value to a specific header field; (*ii*) the `copy_field`, which copies the value of a header field for another field; (*iii*) `add_header`, this sets a specific header instance and all its fields; (*iv*) the `remove_header`, which deletes a header and all its fields from a packet; and, (*v*) the `increment` for incrementing or decreasing the value in a field. Today, in P4 version 16, some devices can support additional primitives, but, in these cases, the programs require external libraries coded by manufacturers (YANG et al., 2017b). Although P4 offers a more flexible model to program the network than OpenFlow, it focuses on the control of the forwarding elements. Thus, again, applications that require packet data processing are beyond the scope of this technology.

Protocol-Oblivious Forwarding (POF)

While P4 designs a programming language to program different devices, **Protocol-Oblivious Forwarding (POF)** introduces forwarding elements that do not know any forwarding protocols (DARGAHI et al., 2017). In addition, POF specifies a protocol-

independent instruction set, which allows a network developer to define the protocol stack and packet processing procedure (SONG, 2013). The instructions can be classified into five different categories (YU et al., 2014): (*i*) editing, this kind of instructions assists in editing the packet data by carrying out tasks such as writing, storing, copying, and calculating values from different fields; (*ii*) forwarding, means these instructions can enable the packets to move forwards, backwards, or to a specific location; (*iii*) entry, this category contains instructions to add, set, and remove flow entries into the match tables of the forwarding elements; (*iv*) jumpm this category makes it possible to change the packet data processing order; and, (*v*) flow instructions provide some operations about the global status of the data flow. By following the instructions from the categories described above, the POF controller can perform the matching of data flows through a tuple `<offset, length>`, in which offset indicates the bit location in a packet that starts the matching fields, and length specifies the size of the field in bits (LI et al., 2017). In this way, POF provides a controller (based on OpenFlow controller) that can create new protocol-independent network applications (SONG, 2013). However, as POF is an OpenFlow-based solution, several features limit its ability to create and adopt applications in a wide area that are targeted at data processing. These features include, but are not limited to, a centralized controller, the emphasis being only on controlling the network, and controller performance.

2.2.5 Proprietary Solutions

Even though there are several programmability-related technologies, the vendors of the main network devices have their own views about what can enable developers to program the network. With regard to these approaches, two companies should be highlighted, Juniper (JUNIPER NETWORKS, 2016b) and Cisco (CISCO SYSTEMS, 2016).

Juniper

The Juniper approach is divided into two solutions: the **Junos Operating System (OS)** and **Junos Software Development Kit (SDK)**. Junos OS is a unified network operating system based on FreeBSD which runs on most Juniper devices; it provides security, robustness, resilience to failure, and a broad platform for the development of applications (Junos SDK). In addition, the basic logical system of Junos OS is divided into three el-

ements: (*i*) the control plane, this manages and controls the device which includes two other planes; (*ii*) the data plane, this is dedicated hardware that filters and forwards the traffic in compliance with a forwarding table; and (*iii*) the services plane, this is hosted on optional and installable hardware (also known as service modules) that allows the running of service applications (KELLY, 2013).

Since Junos SDK enables applications to be developed and used again, it is wholly integrated in the Junos OS. Applications can be built, deployed, and finally, run natively on the Juniper devices like other native processes. Junos SDK provides C and C++ APIs for both the service plane and control plane. Hence, the applications can either control the devices or perform packet data processing such as recording VoIP calls and monitoring the quality of video streams (KELLY; ARAUJO; BANERJEE, 2009). Each application runs on an isolated service module, which ensures the execution is carried out appropriately and securely while avoiding disastrous failures in the network. Moreover, Juniper adopts a rigorous procedure to determine if a developer is a viable candidate to publish an application. Once that process is complete, the developer must have a valid signature and an authorized administrator must run the application. As a result, the Junos OS not only protects the applications running on that network but the underlying network as well (GRILICHES, 2009).

Cisco

The Cisco strategy for network programmability is rather different from that of Juniper. Initially, Cisco introduced the concept of **Application-Oriented Networking (AON)** since it allows programs to be created that focus on the application layer (CISCO, 2005). When seeking to support AON, it was necessary to add a new hardware module into the Cisco devices like the service planning in the Juniper devices that restrict the usage of Cisco AON. After this, in 2012, Cisco introduced the **Open Network Environment (ONE)**, which is a customizable framework that enables developers to create programs to control the network. Cisco ONE supplements traditional SDN approaches (that decouple the control and data planes), while also giving strong support to other deployment models, ranging from traditional Cisco network devices with integrated control and data forwarding capabilities to virtual overlays with pure SDN (CISCO, 2012a).

Cisco ONE consists of two main elements: Open Network Environment Platform Kit (onePK) API and eXtensible Network Controller (XNC). OnePK API abstracts the un-

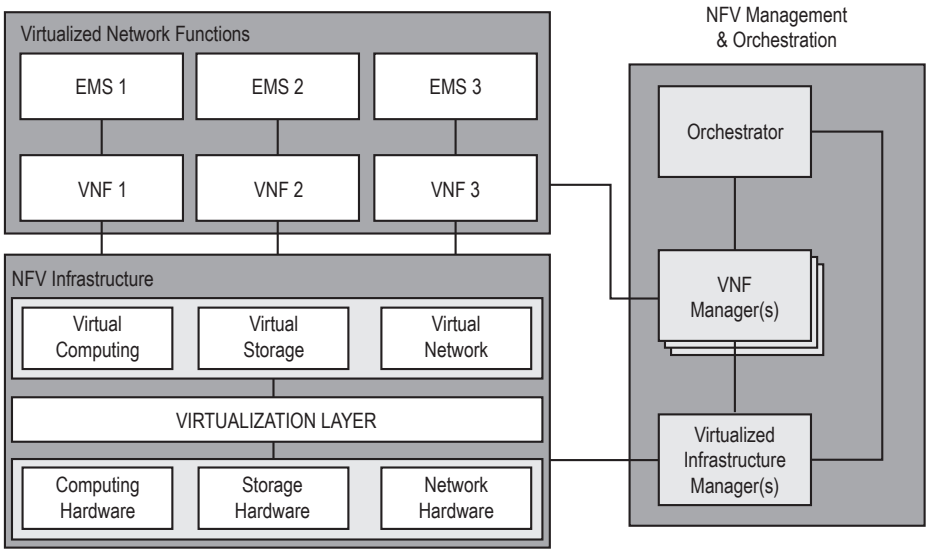
derlying differences between the operating systems and hardware, by providing access to functions that are within the network devices (KIRAN; KINGHORN, 2015). For example, if an application uses a onePK function call to read interface statistics, the same function call will work across all the Cisco network devices. So far, onePK has been able to support three programming languages, including Python, Java, and C. Cisco XNC is an OpenDaylight-based controller which is capable of supporting multiple protocols for device communication such as OpenFlow, onePK, and NETCONF (CISCO, 2012b). Finally, Cisco XNC also provides a northbound Representational State Transfer (REST) API for an application development (CISCO, 2014).

2.2.6 Network Functions Virtualization

In 2012/2013, the Industry Specification Group (ISG), under the auspices of the European Telecommunications Standards Institute (ETSI), published the **Network Functions Virtualization (NFV)** architectural framework. NFV enhances the flexibility of network service provisioning and reduces the time needed for deployment services. NFV moves packet processing from dedicated hardware middleboxes to network functions, called Virtualized Network Functions (VNFs), by running virtual machines hosted on COTS servers. This means that NFV decouples network functions from the underlying vendor-specific hardware, which enables the VNF software to evolve separately from the hardware and vice versa. The benefits offered by NFV allow the network operators to create innovative services, and reduce CAPEX and OPEX. Furthermore, by using NFV, the operator can easily scale the network services to meet the demands of multiple tenants. These can also make it technically feasible and provide potential business opportunities while at the same time, making the network more innovative (NFV ISG, 2012).

Figure 2.7 shows the NFV architecture according to ETSI ISG (NFV ISG, 2013). As can be seen, the main elements of NFV include: (*i*) the NFV Infrastructure which represents all the hardware and software components that comprise the environment in which the VNFs are deployed, managed, and executed; it also ensures that the VNFs are decoupled from the hardware resources, which means the software can be deployed with different physical hardware resources; (*ii*) VNFs, which are network functions running on virtual machines hosted on COTS servers; and (*iii*) NFV Management & Orchestration (MANO), which is reponsible for the orchestration and the lifecycle management of the physical and software resources. To be more specific, NFV MANO is divided into

Figure 2.7: The NFV architectural framework.



Source: adapted from NFV ISG (NFV ISG, 2013).

three components: (*i*) Virtualized Infrastructure Manager (VIM) which monitors, controls, manages, and allocates computing, storage, and network resources with VNFs; (*ii*) VNF Manager (VNFM) which performs the VNF lifecycle management; in other words, VNFM is responsible for instantiating, configuring, updating, scaling up/down, and terminating the VNFs; and, (*iii*) the Orchestrator which coordinates and manages several VIMs and VNFMs. The NFV MANO enables to create a Network Service Chaining, also known as Service Function Chaining (SFC), which is a capability to create a service chain of connected network services (such as firewalls, network address translation (NAT), and intrusion protection) and connect them in a VN. This capability can be used by network operators to set up suites or catalogs of connected services that enable the use of a single network connection for many services, with different characteristics.

When seeking to instantiate a new network service for a tenant, an operator interacts with the Orchestrator that requests a new virtual machine instance to the VIM in accordance with the pre-defined policies and requirements set out in the network service descriptor. After allocating the virtual resources to the new virtual machine, the Orchestrator sends configuration information to VNFM, which takes the necessary steps to instantiate and configure the VNF properly. Assuming this tenant already has multiple VNFs that are instantiated, then, on the basis of all the descriptors, the Orchestrator must define the new network connectivity among the VNFs, which is called a "forwarding graph" (NFV ISG, 2014). With regard to this graph, the VIM configures the virtual links and virtual interfaces so that it can forward the packets correctly. It is worth noting that a single VNF

can represent the degree of cooperation between multiple VNFs; for example, a VNF "security" can represent a set of VNFs from different vendors, including firewall, encryption, the intrusion protection system, and deep packet inspection.

2.3 Summary and Discussion

In this chapter, we examined the main concepts and benefits of both network virtualization and network programmability. We also sought to describe the well-known programmability-related technologies and their main features. A comparison will now be made of these technologies with regard to seven aspects linked to the management and deployment of NetApps. The results of this comparison are shown in Table 2.1. It should be noted that the expression "partial" means that the technology partially supports the particular aspect or that some studies support it. Following this, the aspects in question are examined in detail.

Programs - When the area of network programmability was analyzed, it was found that each technology refers to developed programs that make use of different terms. For example, when a network that supports Active Networks technology is used, a developer can install routines into the nodes to add some functionalities. In the same way, if a network is used to support Script MIB, a developer can install scripts into nodes to carry out the management. After this, the terms were listed that refer to the elements that the developers implement. Thus, the ways in which these elements are installed and managed could be compared.

Level of Programmability - All the technologies mentioned above enable the network to be programmed. However, each technology focuses on different levels. For instance, OpenFlow focuses on controlling the flows in a network. Thus, a developer can define new paths for packets that are in accordance with the developed application. On the other hand, the Click Modular Router is concerned with providing new functionalities for the nodes; for example, a developer can use Click Modular Router to create a new software router that supports an OpenFlow protocol. Three levels of programmability were found when the specific technologies were compared: (*i*) at the *device* level, the technologies were grouped together so that the components of the network devices (physical or virtual) could be supplemented, removed or even altered; (*ii*) at the *control* level, the technologies were classified to enable the network behavior to be controlled, monitored or changed by, for example, altering the path of the packets; (*iii*) at the *service*

Table 2.1: Comparison of programmability-related technologies.

Technology	Programs	Level of Programmability	Application				
			Distribution	Installation	Configuration	Description	Management
Active Networks	Capsule; Routine	Service	Supported	Supported	Partial	Not Supported	Not Supported
Mobile Agents	Mobile Agent	Service	Supported	Not Supported	Partial	Not Supported	Not Supported
Script MIB	Script	Control	Supported	Supported	Supported	Partial	Supported
Click Modular Router	Element	Device	Not Supported	Not Supported	Not Supported	Not Supported	Not Supported
NetFPGA	Module; NetFPGA Package	Device	Not Supported	Not Supported	Not Supported	Not Supported	Not Supported
OpenFlow	Application; SDN Control Software; Business Application	Control	Partial	Partial	Not Supported	Partial	Partial
ForCES	Control Function	Control	Not Supported	Not Supported	Not Supported	Not Supported	Not Supported
P4	P4 Program	Control	Not Supported	Not Supported	Not Supported	Not Supported	Not Supported
POF	Application	Control	Not Supported	Not Supported	Not Supported	Not Supported	Not Supported
Juniper Junos OS + SDK	Juniper Extension Toolkit Application	Service; Control	Not Supported	Supported	Supported	Not Supported	Not Supported
Cisco AON	Bladelet	Service	Not Supported	Supported	Supported	Not Supported	Not Supported
Cisco ONE + onePK + XNC	Application; onePK App	Control	Partial	Supported	Not Supported	Supported	Not Supported
NFV	VNF; Network function	Service	Not Supported	Partial	Supported	Supported	Supported

level, we grouped together the technologies that can handle and use the data travelling in the network. It is worth noting that several technologies can be included at more than one level, but they have a specific focus which we have taken into account. Thus, the main focus of OpenFlow is on the control level, although a developer could program a typical application at a services level such as deep packet inspection.

Application Distribution - This aspect represents the capacity to distribute applications in the network. Thus, it can be assumed that one type of technology supports the distribution if there are mechanisms for the developers to disseminate their applications such as repositories or detailed distribution processes.

Application Installation - Here, an account is taken of the ability to install programs in the network nodes. In light of this concept, P4 offers some facilities that can be used by developers to deploy their P4 programs; for example, a compiler to turn a device-independent program into a device-dependent program. However, this compiler creates an executable file for specific devices, and it does not automate any process to install P4

programs into devices. For this reason, we believe that P4 does not support the installation process. It should be noted that there is a difference between installation and instantiation. Installation is the ability to add a new application to the existing nodes whereas instantiation is the ability to create new virtual instances of network nodes in an NVE. Hence, on the basis of this definition, NFV can instantiate programs, but it cannot install programs, even though the result of the process is similar. In view of this, we recognize that NFV partly supports the installation process.

Application Configuration - We investigated if the technologies are able to configure a program that is already being executed can be configured. This is based on the assumption that the configuration process is supported if a developer can submit a file for settings or establish parameters which customize the execution of the applications in the nodes. In addition, some technologies send the programs to the nodes combined with parameters so as to customize the execution; in these cases, it is assumed that the configuration process is partly supported.

Application Description - When seeking to automate distribution, installation, and configuration processes, some technologies define files that can describe the applications that have been developed, as well as the actions that they can carry out; these files are called descriptors. When the technology supports and defines the content of the descriptors, the application description can be regarded as fully supported. In the event of there being a simple description of applications (*e.g.*, tables in Script MIB), it is assumed that there is a partial support.

Application Management - This aspect includes the capacity to start, pause, resume, and stop the execution of programs in the nodes. Hence, it can be presumed that one technology supports the application management if there are mechanisms or processes for carrying out the actions outlined above.

It is worth observing that OpenFlow does not support any of the processes that have been analyzed. However, the OpenDaylight controller gives an opportunity to third-party developers to submit their applications to a git repository. After the applications have been reviewed by the community, new versions of the controller make them available to the other users. Thus, a user can install these third-party applications “on-the-fly”. In addition, the Beacon controller allows PVN owners to manage the installed applications, through operations such as starting, pausing, and resume. Even though the support is provided by the OpenDaylight and Beacon controllers, the partial support for distribution, installation, description, and management can be attributed to OpenFlow, because these

processes are not available if another controller is used. Furthermore, as the Cisco XNC is an OpenDaylight-based controller, it inherits these features from the OpenDaylight. Despite of this, we classified the Cisco solutions as fully supported because the XNC is the only controller available. However, in Cisco XNC, the distribution process is more limited, because the controller is only compatible with the Cisco applications.

As can be seen in this chapter, there are several technologies that enable developers to program the network. However, there is no unified glossary for PVNs, and thus each technology refers to the same concepts in different ways. For example, NFV refers to NetApps as network functions whereas OpenFlow calls them by different names such as business applications and SDN control software. In light of this, we decided to compile a glossary which brings together the terms related to the same concepts but found in several technologies. The glossary used in this thesis is shown in Table 2.2 below.

Table 2.2: Terms used in this thesis.

Term	Acronym	Definition
Execution Environments	EEs	One node (physical or virtual) or part of a node that executes a network application. In this thesis, it is assumed that one execution environment supports just one programmability technology.
Network Applications	NetApps	Programs coded by developers that provide a network service. The same network application can be coded to several programmability technologies. Thus, there are a number of network application packages (one per technology).
Network Application Package	NetApp Package	An individual file or set of files that contain the code which provides a network service. Only one package has the files that support one kind of programmability technology.
Network Service	–	The functionality provided by a network application or a virtual network appliance.
Programmable Virtual Networks	PVNs	Virtual networks that support some level of programmability.
Virtual Network Appliances	VNAs	Virtual machine instances containing one or more network applications as well as one well-defined execution environment that supports one kind of programmability-related technology.

Even in an NVE, a virtual network usually consists of virtual nodes from different vendors such as Vyatta, Quagga, and Cisco. Moreover, in many cases, different programmability-related technologies are supported. Today, several authors argue that the virtual networks which support the NFV and SDN technologies, are those that are most-commonly used (MATIAS et al., 2015) (AKHTAR; MATTA; WANG, 2016). In addition,

these technologies offer a flexible and dynamic environment, which are essential requirements to overcome the problem of Internet Ossification (FEAMSTER; REXFORD; ZEGURA, 2014) (BOUBENDIR; BERTIN; SIMONI, 2016). However, so far, none of the programmability-related technologies have been able to carry out the distribution, installation, configuration, and management of network services in heterogeneous PVNs from the perspective of the PVN owner. In addition, only a few academic research studies support some of the processes needed (*e.g.*, distribution, configuration, and management), although all of them are technology-dependent.

Integrating NetApps or even VNAs with production networks is a very hard task. The PVN owners must have an extensive knowledge of device instructions and NetApps before they can deploy and manage the network services. As a result, simple tasks, such as transferring or configuring a new NetApp, are extremely complex and repetitive. All this is due to several shortcomings that have been found in the PVNs including the following: (*i*) neither the technological nor academic world has been able to automate the NetApp distribution by taking account of heterogeneous EEs; (*ii*) the same tasks (*e.g.*, distribution of NetApps) have different and conflicting requirements (*e.g.*, minimal network interference or distribution time) in each stage of the service lifecycle (*i.e.*, deployment, operation, and optimization); (*iii*) the initial settings must be manually replicated in each EE to set up the logic for delegating data flows to each new service; (*iv*) the absence of repositories and the unavailability of NetApps for download restrict the distribution of developed services to just a few PVN owners; (*v*) to add new services, PVN owners must now address dependency and conflict issues in the VNAs and also among NetApps; and, (*vi*) there is no descriptor to detail the VNAs or NetApps of heterogeneous programmability-related technologies, and thus prevent the automation of some tasks (*e.g.*, management, configuration, and conflict detection of NetApps).

3 NETWORK MARKETPLACES: THE CURRENT LANDSCAPE, DESIGN GOALS, AND RESEARCH CHALLENGES

The emergence of PVNs (*i.e.*, networks that support virtualization and programmability) has attracted the attention of academia, industry (*e.g.*, Cisco and Juniper), and standardization bodies (*e.g.*, ETSI and IETF). This is because it offers an opportunity to use this paradigm to reduce CAPEX and OPEX, as well as to make innovations in the production network, and thus overcome the problem of Internet Ossification. Marketplaces akin to online application stores have become an essential means of enabling developers to publish and distribute network applications independently. However, research efforts are necessary to ensure these online application stores are more widely adopted in future computer networks. For a better understanding of this context, in Section 3.1, we set out a historical roadmap of networking paradigms and marketplaces to assess how paradigms and technologies have evolved over the years. Furthermore, in Section 3.2.2, we discuss current online marketplaces to identify the essential design goals and main stakeholders. In Section 3.3, we highlight the significant challenges that must be faced to make the adoption of marketplaces a reality in future networks. Finally, in Section 3.4, we summarize this chapter[1] with a brief discussion of network marketplaces.

3.1 A Historical Perspective

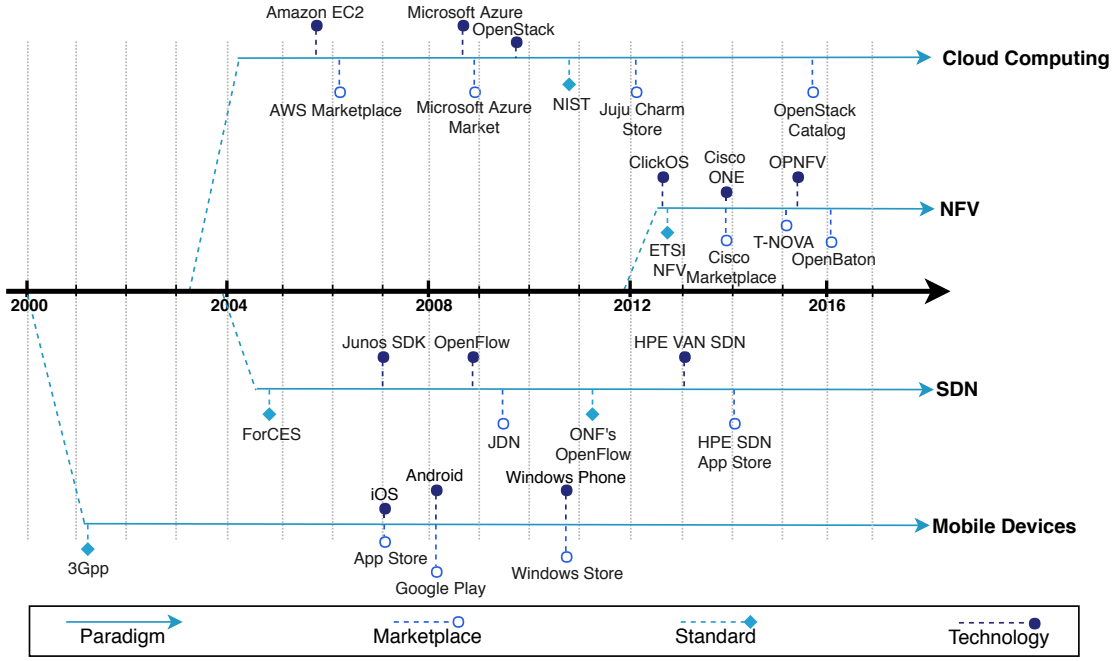
In the recent past, there a number of significant shortcomings in the mobile market that impeded the deployment of applications on mobile devices, namely: (*i*) the heterogeneity of execution environments (*i.e.*, cell phones); (*ii*) the dependence of applications on specific technologies or architectures; and, (*iii*) the fact that the installation and uninstallation processes were remarkably complex and impossible to carry out by typical users. Vendors of mobile devices have successfully overcome these shortcomings through the deployment of online *marketplaces*. As the name suggests, online marketplaces are platform-specific online infrastructures in which developers offer applications (or just apps), which are consumed by end-users. Although the first marketplaces targeted apps that were only provided only by the vendors themselves (*i.e.*, third-party developers

[1] This chapter is based on the following paper: "Network Marketplaces: The Current Landscape, Design Goals, and Research Challenges". For more information about this paper see the Appendix.

were not allowed), Google Play[2] for Android and Apple App Store[3] for iOS devices, introduced a business model where third-party developers were able to offer apps to users of mobile devices.

Online marketplaces have evolved alongside the emergence of new technologies. The popularity of smartphones, for example, created a market for innovative apps, which led the mobile vendors to make use of their marketplaces as a way to offer applications to end-users. A historical perspective of online marketplaces is illustrated in Figure 3.1. Each paradigm corresponds to a secondary horizontal line parallel to the main timeline, on which the technologies and marketplaces are plotted. These marketplaces were selected on basis of a previous analysis, which indicated that these solutions had a market dominance (with regard to the number of customers and services available) in every technology and paradigm that we examined.

Figure 3.1: Timeline of technologies and marketplaces.



Source: the author (2018).

The 3rd Generation Partnership Project (3GPP) released the 3G Networks specifications in the early 2000s, which was a milestone in the mobile networks. Later on, in the mid-2000s, the popularity of smartphones boosted the mobile market (OLIVER, 2009). Although several mobile platforms were developed (*e.g.*, BlackBerry, Symbian, and Windows Mobile), Apple and Google opened up the market of apps, by introducing,

[2]<https://play.google.com/>
[3]<https://itunes.apple.com/>

whole ecosystems comprising operational systems (*i.e.*, iOS and Android) and marketplaces (*e.g.*, Apple App Store and Google Play) in 2007 and 2008 respectively, to provide new apps and features to smartphone end-users. Later, in 2010, Microsoft also adopted this strategy by making both Windows Phone and Windows Store[4] available. As a result, the combination of mobile platforms and marketplaces allowed third-party developers to offer a broad range of new apps. By way of illustration, in 2017, Google Play offered around 3.1 million different apps, and its revenue is forecast to add nearly 10 billion U.S. dollars to the world economy (PORTAL, 2017).

Although cloud computing emerged at the beginning of the 2000s (ZHANG; CHENG; BOUTABA, 2010), it was only consolidated from 2006 onwards, when providers started to offer on-demand services over the cloud. In 2006, Amazon announced an on-demand computing platform, called Elastic Compute Cloud (EC2), and two years later, in 2008, Microsoft entered this market by making available the Microsoft Azure cloud computing platform. Later, in 2010, OpenStack was also introduced as an open-source cloud enabler. As a result of the wide adoption of this paradigm, in 2011, the National Institute of Standards and Technology (NIST) published a definition of cloud computing (MELL; GRANCE, 2011). At the same time, several companies invested in marketplaces as an easy way to distribute applications and services over the cloud. Among them, four marketplaces are worth highlighting: Amazon Web Service (AWS) Marketplace[5], Microsoft Azure Market[6], Juju Charm Store[7], and OpenStack Catalog[8]. AWS Marketplace contains a collection of cloud computing services for EC2. Azure Market provides a collection of integrated cloud services with solutions for data storage, database management, mobile services, and networking. Juju Charm is a project under the auspices of Canonical[9], which consists of a marketplace for the modeling of applications and services for clouds. Finally, the OpenStack Catalog hosts ready-to-use applications that customers can deploy within private or public OpenStack clouds.

In computer networks, SDN is a paradigm characterized by decoupling the network control (control plane) from the forwarding functions (data plane) (WICKBOLDT et al., 2015). As seen in Chapter 2, one of the first significant attempts to standardize SDN was

[4]<https://www.microsoft.com/store/apps/>

[5]<https://aws.amazon.com/>

[6]<https://azure.microsoft.com/marketplace/>

[7]<https://jujucharms.com/>

[8]<https://apps.openstack.org/>

[9]Canonical Ltd. is a UK-based privately-owned computer software company that supports several free and open-source software or tools designed to improve collaboration between free software developers and contributors, such as Juju Charm Store and Ubuntu Linux distribution.

ForCES (YANG et al., 2004), which defined an architectural framework and combined protocols for establishing communication between control and the forwarding elements. In 2007, Juniper released the Junos SDK to allow the development of applications in Junos OS and since 2009, Juniper has maintained the Juniper Developer Network (JDN[10]) and Juniper Professional Services Marketplace (JPSM[11]) to foster a community of NetApps developers. At the end of 2008, however, OpenFlow became established as the most important SDN implementation. Shortly after the first specification of OpenFlow, the Open Network Foundation (ONF) became responsible for OpenFlow standardization procedures (from 2011 onwards). In turn, Hewlett-Packard Enterprise (HPE) released the HPE VAN SDN controller at the beginning of 2013. Immediately after this, in 2014, HPE introduced a marketplace for SDN applications (HPE SDN App Store[12]), which allows PVN owners to deploy services, by aligning the network with business needs.

With regard to the virtualization of network functions, between 2012-2013, the European Telecommunications Standards Institute (ETSI) established the NFV architectural framework (MIJUMBI et al., 2016). NFV removes packet processing from dedicated hardware middleboxes to VNFs, and runs in VMs hosted on COTS servers. From 2012 onwards, several solutions were put forward for accelerating the adoption of NFV, such as open platforms (*e.g.,* ClickOS in 2012 and OPNFV in 2014) and frameworks that simplify the development of VNFs (*e.g.,* Cisco Open Network Environment (ONE) in 2013). In addition, key players have devoted a good deal of effort to facilitate the distribution of VNFs: Cisco Marketplace[13] (released in 2014) offers applications and hardware solutions from Cisco itself, as well as from partner companies, whereas the T-NOVA project[14] (started in 2015) proposes a marketplace that enables PVN owners to purchase and deploy VNFs that can meet their demands. In 2016, the Open Baton project[15] made available a marketplace for downloading VNFs that are compatible with the Open Baton NFV Orchestrator and VNF Managers.

As outlined in this section, there is a trend towards marketplaces for each technology. We believe that PVNs could similarly benefit from a marketplace, for example, by opening up the NetApps market for third-party developers. By doing this, we can capitalize on these previous efforts and identify the design goals and research challenges that are

[10]<https://developer.juniper.net/>
[11]<https://juniper.taskit.io/>
[12]<https://marketplace.saas.hpe.com/sdn/>
[13]<https://marketplace.cisco.com/>
[14]<https://www.t-nova.eu/>
[15]<https://openbaton.github.io/>

needed to achieve a marketplace for NetApps. These issues will be discussed in the next sections.

3.2 Network Marketplaces

In general, marketplaces can be used to publish and deploy apps in different environments (MARTIN et al., 2017). As well as this, typical marketplaces for smartphones, such as Google Play and Apple App Store, rely on proprietary interfaces and having full control over the APIs and the underlying technology. In view of this, these examples of marketplaces provide a sophisticated programming environment for developers, but only for a specific technology; in addition, the interactions between different apps are generally rather restricted.

In our view, *network marketplaces* are platforms that enable network operators to purchase and deploy different network services on their PVNs. It is worth to note that network marketplaces must be capable of handling a diverse and heterogeneous environment, in which there is a broad range of devices, protocols, technologies, and NetApps. In contrast to current mobile marketplaces, the network marketplaces have several additional aspects and requirements that they must handle and manage. Moreover, production networks are very sensitive for both customers and companies, because failures may lead to the disruption of services and systems, and may cause severe financial losses. Therefore, in the following, we discuss the main stakeholders, design goals, and research challenges that, in our view, must be supported by network marketplaces to enable their broad adoption in the future network.

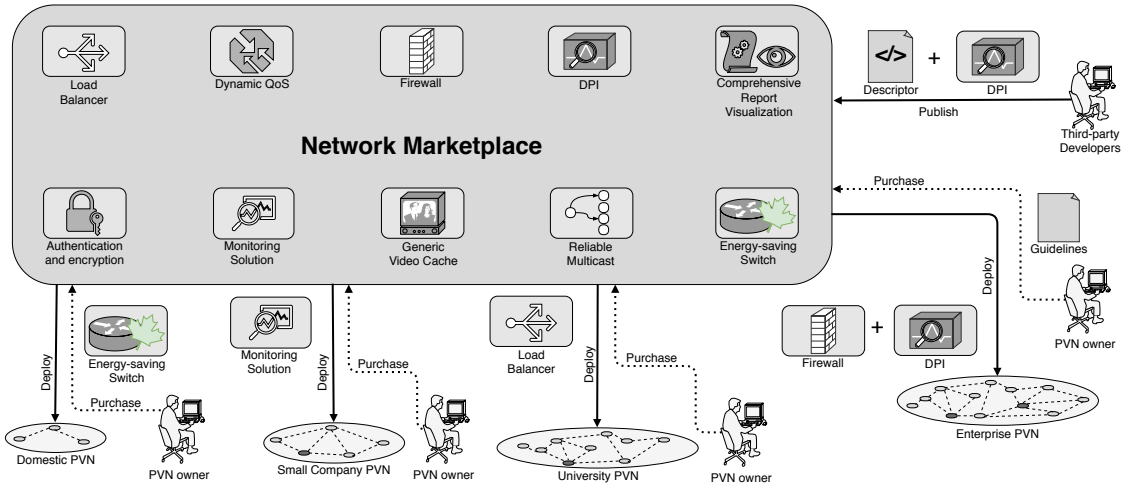
3.2.1 The Main Stakeholders

We envisage that a range of different actors will interact directly with a network marketplace, for example, PVN owners and third-party developers. We illustrate these two actors and their interactions in Figure 3.2. However, other actors must be included because, to some extent, they also take part in the network service life cycle, examples of these are service providers, infrastructure providers, and network customers. In the following section, we discuss the responsibilities of these actors, their roles, and the way they interact with a network marketplace.

PVN owners can purchase NetApps from the marketplace, which will then be responsible for deploying them in the PVN. The purchasing methods and deployment of NetApps can change significantly. For example, in a domestic environment, PVN owners might deploy NetApps as an energy-saving in their network (Figure 3.2). In a university, the owner could deploy a load balancing NetApp to improve the distribution of workloads across multiple servers. In a small company, PVN owners could seek a complete monitoring solution to observe the network behavior. In addition, the PVN owners are able to draw up guidelines to purchase NetApps that meet their customers needs. For example, in an enterprise, the PVN owner may require guidelines for improving security, which might consist of two NetApps (*e.g.*, Deep Packet Inspection (DPI) and Firewall). As a result, when a network customer hires this service, the PVN owner must automatically purchase the corresponding NetApps from the marketplace, which will subsequently deploy them in the PVN.

Third-party developers might be either vendor development teams or independent programmers that code and publish NetApps (Figure 3.2). While vendors tend to provide services for their specific technologies (*e.g.*, Cisco DPI), independent programmers usually develop general purpose NetApps that offer broad applicability (*e.g.*, a Generic Video Transcoding). In network marketplaces, the developers must be able to publish NetApps and choose what fees to levy (see "publish" and "pricing" design goals).

Figure 3.2: Interactions between different roles of the marketplace.



Source: the author (2018).

Like PVN owners, *service providers* must be able to purchase NetApps to meet both their own and their customer needs. However, a service provider could acquire a load balancer to improve the distribution of workloads across multiple mail servers. As well as this, it could purchase a DPI and a Firewall to form a new network security service,

which could then be offered to customers. In fact, from the standpoint of the marketplace, there is no difference between PVN owners and service providers.

Infrastructure providers supply the virtual machines, which will host the PVN nodes. It should be noted that this actor does not interact directly with marketplaces. Instead, PVN owners hire nodes to run NetApps. In addition, PVN owners can use their own physical/virtual infrastructure to host PVN nodes and run NetApps, thus avoiding additional costs. It is worth noting that the infrastructure providers do not have to relinquish their autonomy. The network marketplaces will just access the virtual devices that were opened by the PVN owners.

Network customers use the services transparently; thus, they have no knowledge of the NetApps, marketplaces, or developers. Although these customers do not interact with marketplaces, they are essential to the decision-making process because their needs represent the requirements of new NetApps.

3.2.2 Design Goals

In the following section, we discuss the main design goals that must be taken into account when developing of the network marketplaces; these were grouped into three key categories: *offer and distribution*, *network environment*, and *applications*. These designs goals (highlighted in bold throughout this section) were compiled on the basis of a thorough analysis of features found in the different areas of existing marketplaces, which were discussed in Section 3.1. In addition, Tables 3.1, 3.2, and 3.3 summarize the current panorama of marketplaces. With the aid of these tables, we are able to provide an overview of the key solutions and indicate how they relate to the design goals outlined here. We have obtained this information directly from the documentation related to the marketplaces. It should be pointed out that some marketplaces may require local modules installed in the PVN to achieve a particular design goal (when applicable this is made clear in Tables 3.1, 3.2, and 3.3).

Offer and Distribution

As seen above, users play different roles depending on their functions, for instance, that of a PVN owner or developer. Hence, the first design goal that a network marketplace should support is **access control**. This can ensure that the marketplaces can prevent mali-

cious users from obtaining undue access to services/resources and limit the tasks that they can carry out (SHABTAI et al., 2010). Thus, users must have different levels of access to carry out on specific actions, such as downloading NetApps, publishing updates, and managing deployed NetApps. Before taking any action performed in the marketplace, the user must first reveal his/her access credentials. If authorized, the user will, for example, be able to browse in the NetApp Catalog, filter the available NetApps, install NetApps, or even publish a new NetApp.

All the marketplaces, from different paradigms, enable access control to achieve a minimum security level and to restrict the number of actions taken, such as publishing new applications (Table 3.1). In addition, insofar as the marketplaces support more features, they create new roles and actions for their customers. For instance, Google Play, HPE SDN, JDN/JPSM, and Open Baton enable PVN owners to deploy and update the applications, whereas AWS, Juju Store, Microsoft Azure, and T-NOVA enable PVN owners to deploy, update, and manage the applications.

The network marketplaces must provide mechanisms to assist the **publishing** (ARNTZEN; JOHANSEN, 2005) and allow third-party developers to upload and describe NetApps. By interacting with the marketplace, developers can provide valuable information about new NetApp, such as supported technologies, pricing, access rights, and minimum hardware requirements. Two key aspects must be taken into account as well. First, the marketplaces must ensure the authenticity of the third-party developers, by protecting PVN owners from the installation of malicious NetApps. For example, Juniper adopts a rigorous procedure to determine whether or not a developer is a viable candidate to publish a NetApp in JDN/JPSM. Once completed, the developer must have a valid signature and an authorized administrator must run the application. As a result, Juniper not only protects the NetApps running on that network but the underlying network as well (GRILICHES, 2009). Second, the marketplace must allow third-party developers to maintain their own repositories. By having their repositories, developers and companies are able to host the NetApp packages themselves, thus enhancing the NetApp distribution, as well as preserving their confidentiality and autonomy. Both OpenStack (OPENSTACK, 2017) and T-NOVA (RAMOS et al., 2014) allow PVN owners and developers to configure additional external repositories for the installation of new applications.

Marketplaces must allow developers to decide their **pricing** model, and determine how NetApps will be charged (LLORENTE, 2017). Several pricing models are currently in use, two of which should be highlighted (KANDPAL; GAHLAWAT; PATEL, 2017):

Table 3.1: Overview of marketplaces concerning the design goals of offer and distribution.

	General Information	**Design Goals of Offer and Distribution**			
Marketplace	**Paradigm**	**Access control**	**Publish**	**Pricing**	**Notifications**
App Store Google Play Windows Store	Mobile	Publish, deploy, and update	Third-party developers	Fixed-price	Life cycle and updates
AWS	Cloud	Publish, deploy, update, and manage	Amazon services and certified developers	Pay-as-you-go and fixed-price	Life cycle, runtime, and updates
Juju Store	Cloud	Publish, deploy, update and manage	Juju community and third-party developers	Fixed-price	Not supported
Microsoft Azure	Cloud	Publish, deploy, update, and manage	Microsoft apps and certified developers	Pay-as-you-go and fixed-price	Life cycle, runtime, and updates
OpenStack Catalog	Cloud	Only to publish	Free Software and OpenStack communities	Fixed-price	Life cycle, runtime, and updates (requires local modules)
Cisco Marketplace	Cloud, NFV	Only to publish	Cisco partners and certified developers	Fixed-price	Not supported
HPE SDN	SDN	Publish, deploy, and update	HPE SDN and community apps	Fixed-price	Not supported
JDN/JPSM	Cloud, SDN, and Juniper Solutions	Publish, deploy, and update	Juniper partners and certified developers	Juniper policy and fixed-price	Not supported
T-NOVA	NFV	Publish, deploy, update, and manage	T-NOVA and certified NetApps	Pay-as-you-go, T-NOVA policy (auctions), and fixed-price	Life cycle, runtime, and updates
Open Baton	NFV	Publish, deploy, and update	Open Baton only	Fixed-price	Not supported

pay-as-you-go and fixed-price. In the first, the developers can stipulate a preset rate (prepaid or postpaid) for using every application deployed, which takes account of the time or size limits (AMAZON, 2016b). AWS and Microsoft Azure are built on the pay-as-you-go model. In the second model, developers set a price for the application, which means that the PVN owners pay a fixed price to use it without any time or size limits (CUADRADO; DUENAS, 2012). Google Play, Apple App Store, and Cisco Market are marketplaces that rely on the fixed-price model. T-NOVA also offers a new pricing model, which involves an auction, where whoever offers the highest bid will have access to the entire NetApp's code (RAMOS et al., 2014).

The marketplace should allow the configuration of different pricing models (*e.g.*, pay-as-you-go and fixed-price) and also of several parameters (*e.g.*, payment per hour of used NetApps or per packets processed). It is worth noting that in the case of third-party developers, there are also in-app payments (*e.g.*, premium instances) or additional fees (CUADRADO; DUENAS, 2012); however, network marketplaces do not need to be aware of these in-app payments.

Notifications can ensure that the stakeholders will be informed of important events in the marketplace (CUADRADO; DUENAS, 2012) (ESPOSITO; CIAMPI, 2015). On the one hand, PVN owners can register to receive notifications of relevant events. For instance, someone might be interested in events such as new NetApp updates or poor NetApp performance. On the other hand, third-party developers must be told about the feedback or NetApp failures, which are provided by PVN owners. Furthermore, life cycle events (*e.g.*, dependency failure, conflicts, and deployment status) are pushed to the PVN owner by default. As well as providing event-related notifications, the marketplace must also collect resource monitoring data (*e.g.*, CPU and memory usage).

In academia, several research studies have discussed the design of event brokers and notification systems (LIU; PLALE, 2003) (ARLEIN; BETGÉ-BREZETZ; ENSOR, 2008) (SALLAM; UDGATA, 2011) (GUSEV et al., 2014). We compare the marketplaces, which implement a notification system and the notifications that are supported (Table 3.1). For example, Google Play, Microsoft Windows Store, and Apple App Store provide notifications about life cycle events (*e.g.*, installation and deployment status) and application updates. In addition, Microsoft Azure, OpenStack Catalog, and T-NOVA also provide notifications about runtime (*e.g.*, SLA and statistical data).

The Network Environment

The network marketplace must seamlessly interact with the PVN to deploy new NetApps automatically (*i.e.*, through minimum intervention by the owner). For example, take the case where the PVN supports two technologies: SDN using a POX controller and NFV using ClickOS. In this case, the marketplace must only allow the deployment of NetApps if it is designed for these two technologies (SDN and NFV) and execution environments (POX and ClickOS). Furthermore, NetApps can be grouped into packages for particular services; for example, a "firewall load balancing service" would generally contain two individual NetApps (a Firewall and a Load Balancer).

As seen in Chapter 2, the **deployment** of NetApps can be based on *instantiation* or *installation* (YOU; JUNG, 2014). Some applications are modular and portable; in this case, the deployment procedure entails instantiating the NetApp in the host environment (*e.g.*, by deploying a DPI VM over a Xen hypervisor (HUANG et al., 2011)). Other types of applications may depend on environment configuration or source code compilation; in these cases, the deployment procedure will install the NetApp, and carry out the required configuration tasks in the execution environment (*e.g.*, by updating an SDN controller to support multicast flows (HUNGYO; PANDEY, 2016) or uploading a Juniper Extension Toolkit application to support dynamic Quality of Service (QoS) in Juniper devices (NETWORKS, 2015) (NETWORKS, 2017)). A *Generic Deployment Platform* is required for all the necessary actions related to the instantiation or installation of NetApps. The *Generic Deployment Platform* can integrate a single ecosystem that supports a range of programmable technologies or find a solution for each specific technology, such as Management and Orchestration (MANO Orchestrator) (NFV ISG, 2014) for NFV or OpenDayLight controller (MEDVED et al., 2014) that allow the installation of NFV and SDN NetApps respectively. In Table 3.2, we summarize which marketplaces can provide services for installation, instantiation, or just downloading the applications.

Marketplaces must be **infrastructure-agnostic**, in other words, they must support the coexistence of different technologies within the same infrastructure. For instance, T-NOVA seeks to support NetApps regardless of the underlying NFV technology (RAMOS et al., 2014) (*e.g.*, ClickOS and Dockers) and Cisco Marketplace allows partners and certified developers to publish NetApps in different paradigms (*e.g.*, NFV and Cloud) (CISCO, 2016) (CISCO, 2017). Adapters and drivers (*i.e.*, translation elements) that address the particular features of each technology must be used to ensure that the network market-

Table 3.2: Overview of marketplaces concerning the design goals of the network environment.

	Design Goals of the Environment		
Marketplace	**Deployment**	**Infrastructure-Agnostic**	**Monitoring and SLAs Management**
App Store Google Play Windows Store	Install third-party apps	Particular mobile OS	Not supported
AWS	Instantiate VMs and install third-party software	Amazon infrastructure	Monitoring and specification of SLAs for instances
Juju Store	Instantiate containers and install third-party NetApps	Public and hybrid clouds	Not supported
Microsoft Azure	Instantiate VMs and install third-party software	Microsoft Azure infrastructure	Monitoring and specification of SLAs for instances
OpenStack Catalog	Instantiate and install app packages, templates, and environments	Public and private OpenStack clouds	Not supported
Cisco Marketplace	Redirects to download in developer's site	Cisco technologies	Not supported
HPE SDN	Download only	HP products and SDN technology	Not supported
JDN/JPSM	Download only	Juniper Devices and Junos OS	Not supported
T-NOVA	Instantiate VMs containing the NetApps	NFV technologies	Monitoring and specification of SLAs for NetApps (VNFs)
Open Baton	Download only	NFV technologies	Not supported

places are agnostic to technologies and paradigms. Also, new adapters and drivers can be attached as a means of supporting new *Generic Deployment Platforms* and technologies, and legacy drivers can be replaced to enable the deployment of new NetApps (*i.e.*, by supporting new features).

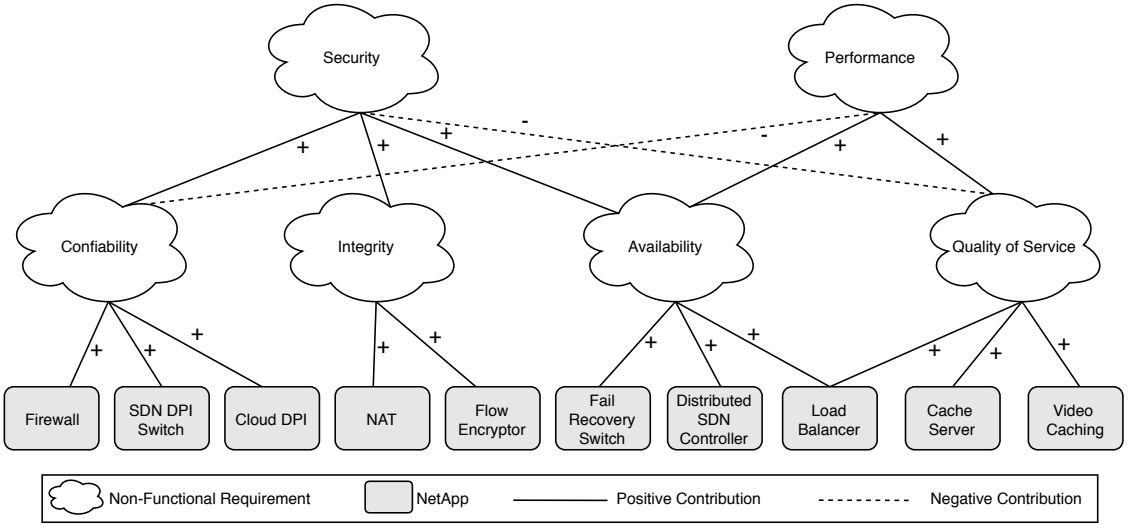
We envisage that marketplaces will have the ability to handle several adapters to interact with different *Generic Deployment Platforms* (*e.g.*, Open Source Mano (HOBAN et al., 2017), OpenDayLight (MEDVED et al., 2014), Cisco eXtensible Network Controller (CISCO, 2014), and OpenStack (OPENSTACK, 2017)). In turn, if the marketplaces comprise an entire system for deploying applications, they must manage and control the drivers, which abstract the nuances and enable network marketplaces to communicate with the underlying physical or virtual devices from different technologies (*e.g.*, OpenFlow, JunOS, and ClickOS) and vendors (*e.g.*, Cisco, Juniper, or Brocade). In our analysis (Table 3.2), we examine both adapters and drivers to observe which underlying infrastructures are supported by each marketplace. For instance, AWS only supports the Amazon infrastructure, while T-NOVA supports several NFV orchestrators.

The network marketplace should also have the ability to monitor the **Service Level Agreements** (SLAs) and detect any violations caused by deployed NetApps. For example, if a deployed NetApp does not comply with the SLAs (*e.g.*, having throughput of at least 100 Mbps), the network marketplace should detect this violation and notify the PVN owner. As one of the key ideas of marketplaces is to abstract low-level configurations and constraints, it is possible the adopt policies that define high-level goals (SCHEID et al., 2017). The specific knowledge of NetApps and technologies can be abstracted in the form of a Softgoal Interdependency Graph[16] (SIG) (CHUNG et al., 2000), which is depicted in Figure 3.3. In the SIG representation, the top non-functional requirements (*e.g.*, *Security*) are satisfied by different middle-level non-functional requirements (*e.g.*, *Inspection*, *Integrity*, and *Availability*) that satisfy this requirement, either positively (solid "+" edge) or negatively (dashed "–" edge), to this requirement. A positive indicator means that when a NetApp is deployed, it can help to meet a non-functional requirement. On the other hand, a negative indicator means that the deployment of this NetApp may have a negative effect on meeting a non-functional requirement.

Network marketplaces could enforce policies to detect SLA violations. In addition, they could lead to an alteration of the SIG and thus trigger the NetApps deployment. In this way, when a PVN owner writes a policy that specifies that his/her infrastructure

[16]It is worth mentioning that the SIG can be altered to reflect the nuances of each environment.

Figure 3.3: A refinement of non-functional requirements.



Source: the author (2018).

needs *Security*, this policy will be refined to match the modeled SIG. This means that the marketplace will select the NetApps that assists positively in satisfying the *Security* requirement and will notify the PVN owner of any possible negative effects of the deployment of related NetApps (*i.e.*, negative contributions). In our analysis, we focus on investigating whether marketplaces can monitor and detect the SLAs violations, even if they do not support the policies. As can be seen in Table 3.2, AWS and Microsoft Azure assist in both the monitoring and detection of SLAs violations. On the other hand, Open Baton, Cisco Marketplace, and HPE SDN fail to support either the monitoring of SLAs or the detection of SLAs violations.

Applications

Consistency is a paramount design goal when building a successful application (PRITCHETT, 2008). In our view, **consistency** is also essential for the success of the marketplaces. Network marketplaces need to maintain a consistent state for the deployed and available NetApps, underlying technologies, and external repositories. For example, if a NetApp is removed from the marketplace, or if it ceases to be supported by the developer, the marketplace must determine the cause of this failure, delete all the related files and entries from the repositories, and notify the PVN owners (HERBAUT et al., 2015).

Several studies in different areas claim that marketplaces must support multiple external repositories (MENYCHTAS et al., 2012) (BUREGIO et al., 2007) (SANTOS et al., 2017). In this scenario, repositories may become inconsistent (*e.g.*, have conflicting data

within the same NetApp) or stop working (*e.g.*, through loss of connection or corrupted files) (SCHEFFCZYK et al., 2003) (VOUILLON; COSMO, 2013); in view of this, marketplaces must also determine what is the consistent state of external repositories and, if necessary, repair them. It is worth noting that network marketplaces should at least be able to temporarily remove the affected NetApps from the catalog. In our analysis (Table 3.3), we investigate which actions are supported by each marketplace. For example, AWS and OpenStack fix broken repositories, notify PVN owners about instances states, and resolve the problem of unsupported applications (*i.e.*, temporarily remove them from the catalog), whereas Cisco Marketplace and HPE SDN do not support actions related to consistency.

Applications may establish several **relationships** with each other, the two main ones being *dependencies* and *conflicts* (VOUILLON; COSMO, 2013). A dependency is a condition must be satisfied for the correct execution of the NetApp (NFV ISG, 2014). This condition may be a requirement of another NetApp or even underlying technologies. In an enterprise environment, a visualization NetApp may depend on a specific monitoring NetApp to provide visual reports about the network state. If the monitoring NetApp is not deployed, the visualization NetApp may operate on a restricted mode and provide more limited reports. In another environment, a reliable multicast NetApp may solely depend on a dynamic QoS NetApp, which means, the reliable multicast will not operate if the dynamic QoS NetApp is not deployed (JUNIPER, 2010). In addition, some NetApps may require on particular technologies to be deployed (*e.g.*, a load balancer NetApp implemented for a POX controller in SDN environments).

Several conflicts can occur in a network environment (DZIUBINSKI; GOYAL; MINARSCH, 2016), such as policy conflicts (CANINI et al., 2015) (HAMED; AL-SHAER, 2006), resource conflicts (NFV ISG, 2014), or even conflicts between NetApps (SCHWABE; GUTIÉRREZ; KARL, 2016) (CANINI et al., 2014). The marketplaces must provide mechanisms that can help developers to describe known conflicts between NetApps as well as their degree of severity. For instance, in a minor conflict, a load balancer NetApp may reduce the performance of a cache server. In a critical scenario, a trust-based multicasting NetApp may lead to the improper operation of a DPI NetApp. The marketplace should analyze these conflicts and prevent incompatible NetApps in the same PVN. Furthermore, the network marketplace must check the conflicts and dependencies before triggering the deployment of new NetApps.

As can be seen in Table 3.3, we found that Juju Store, Microsoft Azure, AWS, and

OpenStack Catalog make it possible to describe conflicts and dependency issues and check them before deploying new apps. However, most network marketplaces fail to addresss the question of conflicts and dependencies, including Cisco Marketplace, JDN/JPSM, and HPE SDN. T-NOVA is the only exception to this because it addresses dependency issues among NetApps. Besides, it is worth stressing that mobile marketplaces are able to encapsulate all the dependencies (*e.g.*, libraries and others applications) in a single package.

Table 3.3: Overview of marketplaces concerning the design goals of applications.

	Design Goals of Applications		
Marketplace	**Consistency**	**Relationships**	**Management**
App Store Google Play Windows Store	Resolve the problem of unsupported apps, and fix broken repositories	Packages encapsulates apps dependencies	Partial life cycle (collecting data, updating, and deleting)
AWS	Resolve the problem of unsupported apps, give notification of instances states, and fix broken repositories	Dependency and conflict issues	Complete life cycle
Juju Store	Resolve the problem of unsupported apps and fix broken repositories	Dependency and conflict issues	Complete life cycle (requires local modules)
Microsoft Azure	Resolve the problem of unsupported apps, give notification of instances states, and fix broken repositories	Dependency and conflict issues	Complete life cycle
OpenStack Catalog	Resolve the problem of unsupported apps, give notification of instances states, and fix broken repositories	Dependency and conflict issues	Complete life cycle (requires local modules)
Cisco Marketplace	Not supported	Not supported	Not supported
HPE SDN	Not supported	Not supported	Not supported
JDN/JPSM	Not supported	Not Supported	Not supported
T-NOVA	Resolve the problem of unsupported apps, give notification of instances states, and fix broken repositories	Dependency issues	Complete life cycle
Open Baton	Not supported	Not supported	Not supported

Once NetApps are deployed over the infrastructure, we argue that the network mar-

ketplaces must assist the **management** of the NetApps life cycle (ERICKSON, 2013) (VALOCCHI et al., 2017). It is possible to pause, resume, or even configure the execution of a specific NetApp depending on the performance events of a particular network. For example, during the peak hour traffic, a PVN owner can reduce the sampling rate (*i.e.*, reconfigure) of a deployed DPI to deal with the problem of an overloaded network, and thus avoid poor customer experience. In a more extreme case, this owner could pause the execution of the DPI. When there is a reduction in the volume of network traffic, the PVN owner could set up the default configuration of the DPI execution, or even resume it, to prevent unwanted access. For this reason, the marketplace must allow PVN owners to carry out several management tasks, which include installing, starting, pausing, resuming, configuring, updating, uninstalling, and collecting data from a NetApp (RAMOS et al., 2014) (CISCO, 2014). These tasks are crucial to ensure the correct execution of the whole network environment complies with the constraints defined by the SLAs, as well as to meet the needs of the PVN owners.

The marketplace should provide an API to support the management tasks described above. Today, a number of marketplaces offer similar APIs for these kinds of tasks. For example, Microsoft Azure API (MICROSOFT, 2016) and AWS API (AMAZON, 2016a) allow customers to create programs to manage virtual hosts, snapshots, images, and virtual networks. With the aid of local modules, Juju Store (CANONICAL, 2017) and OpenStack Catalog (OPENSTACK, 2017) can offer several services in the managed nodes, such as virtual machine instantiation, application management, and underlying hardware management. In the context of the network, T-NOVA provides an API that covers the complete life cycle of applications management, whereas Cisco Marketplace, HPE SDN, JDN/JPSM, and OpenBaton do not support any management tasks in their applications.

3.3 Research Challenges and New Directions

Network marketplaces must deal with specific challenges and aspects. While conventional marketplaces (*e.g.*, Google Play and Apple App Store) are mostly concerned with publishing and deployment, network marketplaces (*e.g.*, Juju Charm Store (YANG et al., 2017a) and T-NOVA (RAMOS et al., 2014)) must address striking aspects, such as placement, auditing, and the recommendation of NetApps. Besides, there are additional challenges (*e.g.*, heterogeneity, legacy support, and the continuous introduction of new

technologies) which must be dealt with. In identifying the leading questions that remain open from our standpoint, this discussion of future research directions is based on the previous analysis of the design goals and our experiences in this thesis.

3.3.1 Business Model

Undoubtedly, designing a business model is a critical research challenge for the adoption of network marketplaces. Today, there are two methods that are widely used for NetApps acquisition: fixed-price and pay-as-you-go. The first one denotes that the customers must pay an amount that is determined in advance by NetApps and can use the service without any time or size limits. Google Play, JDN, and Cisco marketplace are examples that use this method for NetApps acquisition. The pay-as-you-go method defines a preset rate (prepaid or postpaid) to use NetApps, that takes account of the time or size limits. For example, a developer can offer a NetApp for trusted multicasting and a preset rate per flow route. AWS and Microsoft Azure are marketplaces that use this method for some NetApps.

In our point of view, the future network marketplaces must employ two other methods for NetApps acquisition: holding auctions and custom-built method. Although some marketplaces hold simple auctions (see Subsection 3.2.1), in which the highest bidder acquires the NetApp, in our opinion, both third-party developers and network customers should be able to initiate an auction. Thus, in the developers' auction, the highest bid from a customer acquires the NetApp whereas, in the customers' auction, the lowest bid will mean that the developer has to provide the NetApp for the desired network service. Finally, in the custom-built method, third-party developers negotiate with customers to develop a NetApp for specific needs. As a result of this negotiation, there will be an agreement about the final price, requirements, SLAs, deadlines, and desired features of the new NetApp, which will still need to be programmed.

3.3.2 Evolution-Aware

Network technologies are in constant evolution. There was evidence of this in the recent paradigm shifts on PVN development, which include the decoupling of data and the control plane through SDN and the virtualization of the network functions provided by

NFV. Hence, future marketplaces must find the means to support new network paradigms seamlessly. Some efforts have need made to develop middlewares that allow the execution of different NetApps in heterogeneous environments (WETTINGER et al., 2016) (GARCÍA-VALLS; BELLAVISTA; GOKHALE, 2017).

With regard to the current marketplaces, T-NOVA provides an orchestration layer to allow the deployment of VNFs, service provisioning, and resource management in NFV technologies. Juju store offers local modules that can instantiate NetApp containers over public and hybrid clouds. HPE SDN allows NetApps to be deployed in networks independently of SDN technologies. However, those efforts are tailored to particular paradigms, and still require researchers to develop middlewares with well-defined APIs that can be incorporated into marketplaces so that NetApps can be executed regardless of the network infrastructure.

Let us consider a visualization NetApp developed for both SDN and cloud environments. The marketplace must abstract the particular technological nuances so that the publishing and deployment can take place in both environments. For example, the PVN owners could use the same REST methods, *e.g.*, *publish()* and *deploy()*, for the publishing and deployment tasks in SDN and cloud environments. However, both network marketplaces and *Generic Deployment Platforms* must be able to execute the specific commands in each environment. Moreover, these commands must be comprehensive enough to allow, with minor modifications, the NetApp to operate in emerging paradigms, such as NFV.

There must also be mechanisms to handle management requests in accordance with the specified features of the environments. For example, a monitoring NetApp, developed for both SDN and NFV environments, may trigger a predefined action when detecting a saturated Web server. In an NFV, this action could start another instance of the Web server. In an SDN environment, it could divide the flows between the several Web servers (if they exist). It should be noted that third-party developers must provide these different actions or provide means for the PVN owners to configure them. However, the marketplace must provide the necessary mechanisms for these different actions by means of the same system call from the NetApp. Hence, dealing with a heterogeneous environment is a significant research challenge.

3.3.3 Auditing

PVN owners should be able to check if the deployed NetApps are providing the advertised functionalities. This means that the network marketplaces must adopt auditing mechanisms to gather information about the execution of NetApps. For example, a PVN owner may deploy a network service for Distributed Denial-of-Service (DDoS) prevention. Upon request, the markeplace must provide reports to the PVN owner with evidence that the deployed NetApp is actually preventing DDoS attacks. As well as this, PVN owners may not have a wide technical knowledge, and thus it is important that these reports should be easy to understand.

The auditing reports must determine not only if a NetApp meets the established SLAs, but also how NetApps affect the environment in which their run. Thus, research efforts should be aimed at design mechanisms that combine monitoring information (*e.g.*, traffic pattern and resource usage) and diagnostic models (*e.g.*, classification and machine learning approaches) to produce comprehensive reports. Another factor is that the consolidation of the next generation mobile broadband technologies and the expansion of the Internet of Things (IoT) are pushing NetApps close to PVN owners. This strengthens the influence of management decisions (*e.g.*, resource allocation, scaling of NetApps instances, and chaining strategies) on applications that depend on the execution environment of the PVN. In this climate, applications that deal with human health and safety (*e.g.*, decisions made for autonomous vehicles and health-care agents) depend on the precise functioning of the NetApps running on the PVN, where a single error can put a life at risk. Moreover, when something goes wrong in any part of the system, it has an impact on human life. Thus, someone should have a legal liability; the auditing mechanisms can help to assign responsibilities. For example, they can help to discover the wrong management decisions or coding errors in NetApps. However, scientific research and moral discussions must follow the same path so that they can determine these legal responsibilities.

Although in this thesis we classify auditing as an open research challenge for the entire network environment, currently, there are efforts being made to deal with specific tasks. In the context of NFV, Bless and Flittner (BLESS; FLITTNER, 2014) set out an auditing mechanism to help customers to check if the resources allocated by the service providers are in compliance with the established SLAs. Carella *et al.* (CARELLA et al., 2015) also use monitoring information to compare Key Performance Indicators when auditing QoS

and for an elastic scaling of resources. With regard to SDN, auditing mechanisms provide an effective way of determining network correctness on the basis of specifications and tracing the root cause of problems (AKHUNZADA et al., 2015).

3.3.4 Recommendation of NetApps

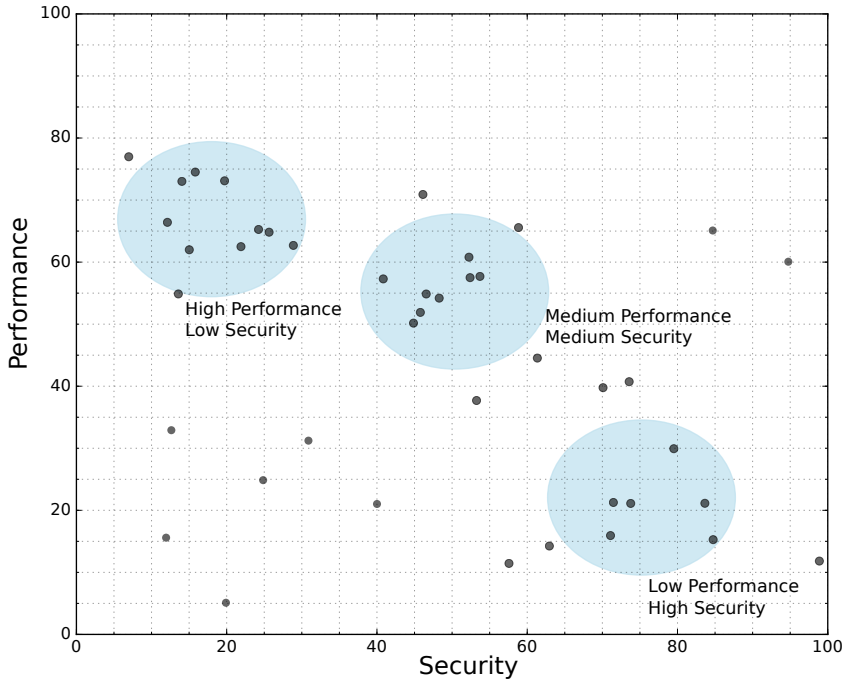
A plethora of NetApps will be available in network marketplaces, and some may share similar goals. An open research challenge is to provide the means to distinguish, or aggregate, these NetApps to meet a specific requirement. For example, security-related NetApps can offer security capabilities at different levels, such as inspection firewalls for L3 packets and intrusion prevention systems that detect malicious traffic patterns. The challenge is how to define which NetApps must be selected by PVN owners to meet their target requirements.

In the literature, there are several recommendation systems in diverse areas. Burke proposes a system that combines knowledge-based recommendation and collaborative filtering to recommend restaurants (BURKE, 2002). Also, clustering techniques have been employed to address the recommendation of applications and products (LINDEN; SMITH; YORK, 2003) (LU et al., 2015). Ricci *et al.* discuss the basic recommender systems ideas and concepts to understand the techniques used to provide suggestions for items that are most likely of interest to a particular user (RICCI; ROKACH; SHAPIRA, 2015). Bobadilla *et al.* aim to provide an overview of recommender systems as well as collaborative filtering methods and algorithms; they also provide an original classification for these systems that are incorporating social information to recommend items to the people (BOBADILLA et al., 2013).

Assigned *scores* could help to clarify how well each NetApp addresses a specific non-functional requirement. Thus, a clustering algorithm (*e.g.*, k-means (MACQUEEN, 1967)) could be used to classify the NetApps that share similar scores in a desired number of clusters (ISINKAYE; FOLAJIMI; OJOKOH, 2015) (JU; XU, 2013). For instance, let us consider a set of NetApps that has scores for security and performance. Figure 3.4 shows how these NetApps can be grouped into clusters at three different levels. The x-*axis* of the graph represents the security requirement score of the NetApp, while the y-*axis* shows the performance requirement score. These scores range from 0, meaning that the NetApp can provide 0% of such a requirement (*e.g.*, security), to 100, which means providing 100% of the same requirement. This means there are NetApps that provide

a high-level security, but provide a low-level performance; and NetApps that provide a low-level security but a high-level of performance.

Figure 3.4: Example of NetApp clustering.



Source: the author (2018).

In a recent study (SCHEID et al., 2017), Scheid *et al.* proposed an intent refinement process that clusters VNFs according to user-defined contexts. In another study (JACOBS et al., 2017), Jacobs *et al.* provide a mechanism to calculate the affinity score for each pair of VNFs in a service function chain. However, further research is still necessary to fully integrate these solutions into network marketplaces. A reliable recommendation mechanism for NetApps must address several challenges, including: (*i*) the need to consolidate the classification mechanism, (*ii*) defining the number of NetApps in each cluster, (*iii*) providing ways to assess classification correctness; and (*iv*) how to establish affinity and anti-affinity relationships between the NetApps.

3.3.5 Placement

Finding the best placement of NetApps within the substrate infrastructure is difficult because each PVN owner may have different priorities and goals. Besides, some NetApps may require a specific location for execution. For example, while firewalls are better placed at the network edge (*i.e.*, close to the external link), an IP media transcoder (*i.e.*, a NetApp that processes media streams) must remain close to the content servers. A

placement mechanism must take into account predefined criteria and provide automated and manual mechanisms to define optimal locations for NetApps. The placement criteria might include: minimal network delay, energy saving, deployment costs, and resource utilization.

The placement problem has been widely investigated in the literature on virtual networks over the years. For instance, Haas *et al.* (HAAS; DROZ; STILLER, 2001) established a framework to enable QoS-aware service deployment over programmable virtual heterogeneous networks. Wu *et al.* (WU et al., 2013) proposed an SDN-based solution to instantiate new virtual nodes and manage the virtual networks through the use of a centralized server and distributed agents. Moens and Turck (MOENS; TURCK, 2014) sought to optimally place VNFs and network services on the basis of established policies in the context of NFV. Luizelli *et al.* (LUIZELLI et al., 2015) formalize the network function placement and chaining problem and propose an Integer Linear Programming model to solve the placement in the NFV context. In SDN networks, Hock *et al.* (HOCK et al., 2013) tackled the problem of best positioning distributed controllers. However, network marketplaces must also enable the easy and flexible placement of NetApps with regard to both different technologies and concurrent or conflicting placement criteria.

3.3.6 Security

Network marketplaces have particular security issues related to stakeholders and external elements that require special attention. We expect that NetApps will be developed and published by several third-party developers and that different environments will deploy these NetApps. For these reasons, the marketplace must employ security mechanisms to prevent the environment from becoming a target of malicious attacks. For instance, if a PVN owner acquires a NetApp for energy saving in his SDN network, the marketplace must assure the integrity of the NetApp and also provide a secure communication channel (*e.g.*, by using cryptography) to deploy and send management commands to the NetApp. This would prevent malicious users from interfering with the communication (*e.g.*, man-in-the-middle attacks to steal sensitive data) or sending commands to perform undesired actions (*e.g.*, installing malicious software or stopping services). Besides, malicious users could develop NetApps to initiate attacks against third-party environments (REYNAUD et al., 2016). In view of this, marketplaces should employ tools to ensure the integrity of NetApps.

Mechanisms to deal with the three cornerstones of security (SCOTT-HAYWARD; O'CALLAGHAN; SEZER, 2013) (*i.e.*, confidentiality, integrity, and availability) are required to establish a secure relationship between the PVN owners and developers. The required functionalities include: (*i*) validating the NetApps execution, (*ii*) certifying the NetApps source code, and (*iii*) ensuring a secure communication between the execution environments and the marketplace.

3.3.7 Descriptors

NetApp descriptors are essential elements for the adoption of marketplaces. Current network marketplaces (*e.g.*, AWS and OpenStack Catalog) support these descriptors to represent the main features of NetApps, such as versioning, dependency, and pricing. However, a number of key issues remain unsupported. For example, with regard to the application relationships, descriptors must also help developers to specify conflicts between the NetApps; in the case of auditing, the descriptors must provide tags that advertise the NetApps functionalities (*e.g.*, a security NetApp that prevents DDoS attacks); regarding to recommendation, these tags must express the level at which a given NetApp satisfies a set of goals (*e.g.*, security and performance); and, with regard to placement, they must determine which placement criteria should be included. The definition of these tags is not a trivial task, owing to aspects concerning the subjectivity of the information (*e.g.*, what data should be included and how to collect it to check if a NetApp is delivering what it promises) and using significant information (*e.g.*, the placement criteria employed in the NFV paradigm could be different from those employed in the cloud paradigm).

Several research efforts have concentrated on the development of descriptors as a way of standardizing applications and their execution within a particular environment. For example, ETSI proposes the Network Service Descriptor (NSD) to specify the nuances of NetApps in NFV environments (NFV ISG, 2014) and OpenStack describes the applications in a manifest using the Murano specifications (OpenStack, 2013) for cloud environments. As a result, tags, parameters and commands are generally only compatible with the particular target technology, which makes these descriptors unsuitable for a marketplace that will help in the management of general purpose technologies. Therefore, significant research efforts are still required to investigate the provision of standardized descriptors that can fully accommodate the identified needs.

3.4 Summary and Discussion

PVNs allow innovative NetApps to coexist in heterogeneous infrastructures and to be rapidly developed by third-party developers. However, the diversity of NetApps and the number of upcoming technologies make the management and deployment of NetApps by PVN owners a laborious task, and also prevents the PVNs from becoming popular. Traditionally, marketplaces (*e.g.*, mobile application stores) have helped to overcome similar challenges. In this chapter, we identified and discussed the current research efforts, design goals, and open research challenges with regard to network marketplaces.

By analyzing the historical perspective of marketplaces, we can notice that marketplaces are inherently related to three key categories, namely: offer and distribution, network environment, and applications. On the basis of analysis and the key categories, we were able to compile a list of the main design goals for a reference network marketplace, which include the following: (*i*) access control; (*ii*) publishing guidelines; (*iii*) pricing methods; (*iv*) notifications about important events; (*v*) deployment of NetApps; (*vi*) support of different technologies; (*vii*) the monitoring/management of SLAs; (*viii*) the question of consistency with regard to NetApps and repositories; (*ix*) the relationship between NetApps, in special, dependencies and conflicts; and, (*x*) the management of the NetApps lifecycle. It is worth highlighting that, in our opinion, this set of design goals comprises the suitable features of the future network marketplaces.

We also discussed the research challenges raised by the adoption of network marketplaces. We found that issues regarding business models, evolution-awareness, auditing, recommendations, placement, security, and NetApps description, still require considerable investigation by researchers. In our view, significant research is needed to integrate the different technologies and thus provide a flexible and useful PVN. This investigation should include: (*i*) discovering the best allocation/sharing of underlying resources (*i.e.*, through computing, memory, storage, and the network), as a means to scale up/down virtual resources an thus optimize the provided service; (*ii*) defining technologies of abstractions to enable the standardization of methods to carry out specific actions (*e.g.*, deployment and management) within several infrastructures and NetApps; (*iii*) determining the best chaining among NetApps to reduce costs as well as improve the performance of the services; (*iv*) enabling the simple build up of PVNs by using virtual nodes from different virtual infrastructure providers; and, (*v*) security/privacy issues in PVNs that involve several stakeholders.

4 APP2NET ECOSYSTEM

In this chapter[1], we introduce the conceptual App2net ecosystem to enable PVN owners to install, configure, and manage NetApps. Our ecosystem also helps third-party developers to distribute and describe NetApps in the context of PVNs. In fact, we divide our ecosystem into two key platforms, namely Marketplace for PROgrammable Virtual nEtworks (`iMPROVE`) and Applications to Network (`app2net core`). It is worth noting that both platforms can operate in an isolated way, but all the features remain available when PVN owners use the whole App2net ecosystem (*i.e.*, `iMPROVE` and `app2net core` platforms).

First of all, we propose `iMPROVE` as a means of simplifying the distribution and description of network services provided by NetApps and VNAs in PVNs with multiple Execution Environments (EEs). We designed a platform for repositories where both NetApps and VNAs can be stored, published, and distributed. We also introduced a conceptual marketplace for different technologies, such as NFV, OpenFlow-based, and VyOS. This marketplace enables PVN owners to select, download, and trigger the deployment of new network services. Furthermore, the European Telecommunications Standards Institute (ETSI) Network Service Descriptor (NSD) (NFV ISG, 2014) has been extended to support the NetApps and VNAs of several programmability-related technologies, as well as to describe issues of conflict.

In completing the lifecycle of a Network Application (NetApp), we also introduce the `app2net core` platform to tackle the problem of deploying NetApps with regard to PVNs. For this reason, the aim of `app2net core` is to enable PVN owners to transfer, install, and configure the NetApp package among heterogeneous EEs in a transparent way. Moreover, we analyze the key features of a set of code transfer techniques, which are, then, grouped together. Finally, we design different transfer models based on the set of techniques previously described. These models are employed to transfer NetApp packages and configuration files among the PVN nodes to achieve particular goals (*e.g.*, minimal distribution time and minimal overhead).

The rest of this chapter is structured as follows. In Section 4.1, we present an overview of the App2net ecosystem architecture as well as the interface for the two key platforms. Section 4.2 provides a detailed account of the `iMPROVE` platform and our extension for

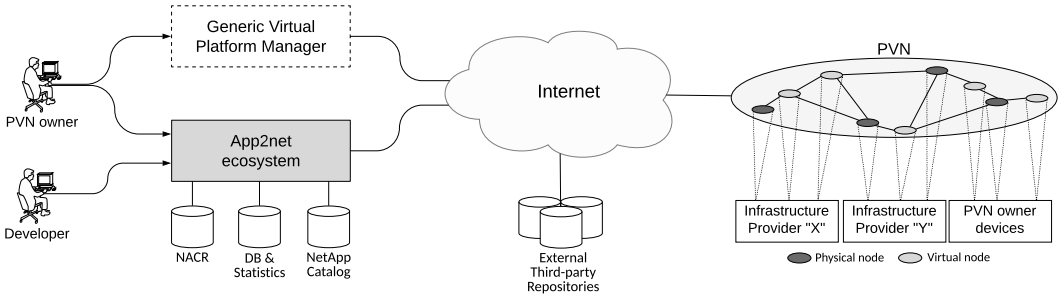
[1]This chapter is based on the following papers: “App2net: A platform to transfer and configure applications on programmable virtual networks” and “iMPROVE: Enhancing the Introduction of Services on Programmable Virtual Networks”. For more information about these papers see the Appendix.

the ETSI NSD. In Section 4.3, we describe the `app2net core` platform and the its main components. There is also a discussion of the techniques employed and the models designed for transferring NetApps packages are introduced. Section 4.5 concludes with some final remarks about this chapter.

4.1 An Overview of the App2net Ecosystem Architecture

The purpose of the App2net ecosystem is to deploy, manage, and distribute NetApps in PVNs that use heterogeneous EEs. These EEs can be hosted on virtual or physical devices. We consider that the deployment consists of three key actions, namely transfer, install, and configure NetApps. In Figure 4.1, we provide an overview of the interactions of the external stakeholders to our proposed ecosystem. At the outset, it is assumed that there is a traditional best-effort, TCP/IP network that mediates the communication between the App2net ecosystem and PVN nodes.

Figure 4.1: Overview of the external interactions of the App2net ecosystem.



Source: the author (2018).

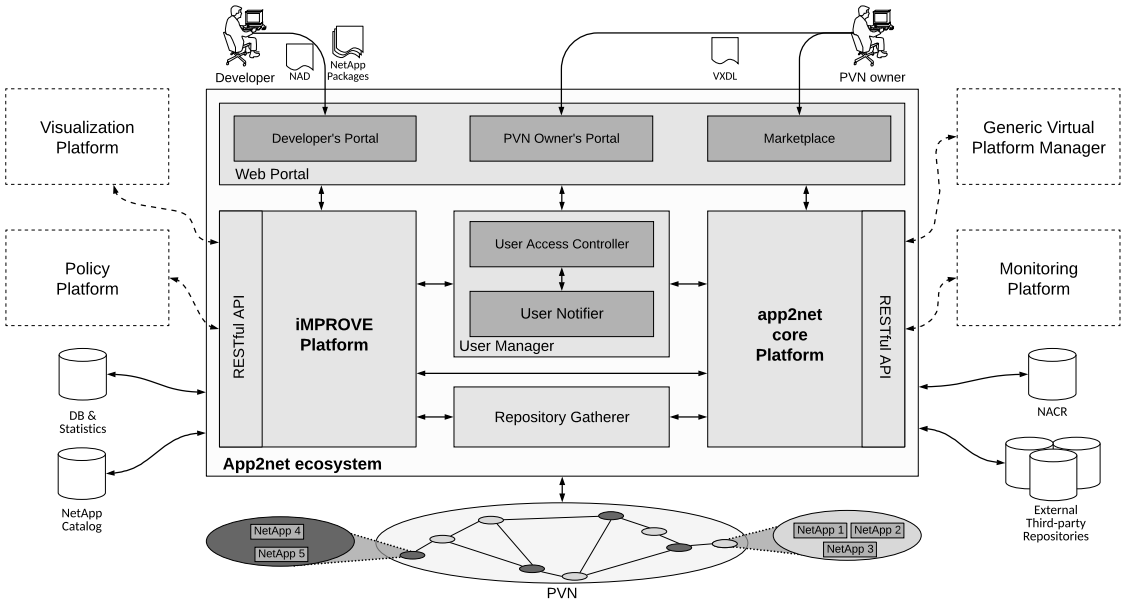
Two of the main stakeholders are discussed here: third-party developers and PVN owners (Figure 4.1). The developers can make NetApps and their initial functional settings available, as well as the VNAs. All these files are stored in different repositories (see "Databases and Repositories" in the Subsection 4.1.1), namely, DB & Statistics, NetApp Catalog, and NetApp Code Repository (NACR). The developers or even companies sometimes prefer to keep the files that provide the network service within their own infrastructures. This requires the developers to install a driver into these repositories, which are called External Third-party Repositories.

The PVN owners interact with a Generic Virtual Platform Manager (*e.g.*, OpenStack (OPENSTACK, 2016), Eucalyptus (EUCALYPTUS, 2015), or Aurora (WICKBOLDT et al., 2014)) to set up a PVN on one or several Infrastructure Providers. However, it should be pointed out that these steps are beyond the scope of our ecosystem. After this, the

PVN owners can select and trigger the deployment of a NetApp or VNA, which provide a new network service (*e.g.*, a deep packet inspection firewall, video encoding/transcoding, and trust-based broadcasting) in their PVNs. To this end, App2net offers a Web Portal that allows PVN owners to carry out actions without knowing the specific features of the underlying environment. Through this Web Portal, the PVN owner can invoke the elements needed to deploy the NetApps or VNAs, and thus avoid complex interactions with it.

In Figure 4.2, we provide an overview of the general elements of the App2net ecosystem. Let us suppose that a developer wants to release a new network service, for example, a reliable multicast. First of all, it must first send the NetApp package, VNA image, and NAD file through the Developers Portal (see "The Web Portal" in the Subsection 4.1.1) of the App2net ecosystem. Then, the `iMPROVE` platform stores these files in the corresponding repositories and makes the reliable multicast network service available in the Marketplace.

Figure 4.2: Overview of the general elements of the App2net ecosystem.



Source: the author (2018).

First of all, the PVN owner interacts with App2net through of the PVN owner's Portal. Then, the PVN owner provides information about the PVN (*e.g.*, node addresses and its supported programmability technologies). Afterward, the PVN owner selects the NetApps that must be deployed in the Marketplace. As can be seen in Figure 4.2, these NetApps and their initial functional settings are stored in NACR and NetApp Catalog repositories. Following this, the PVN owners can tune these initial functional settings so as to adjust the execution of the new network service to their PVN.

To be more specific, when installing a new network service, the `iMPROVE` platform sends the NetApps list (including the dependent packages) to the `app2net core` platform and supplies information about the nodes. After this, the `app2net core` carries out the necessary actions to install and configure these NetApps into PVN nodes following the tuned settings. Next, the PVN owner is able to collect data and statistical information from the NetApp execution. It is worth noting that a single PVN node can host several different NetApps. Below, we outline the general elements of the App2net ecosystem that enable the deployment and description of new network services transparently.

4.1.1 General Elements of the App2net Ecosystem

The App2net ecosystem requires several key elements to support its processes, including Databases and Repositories, Web Portal, User Manager, and Repository Gatherer. All these elements enable the App2net ecosystem to: (*i*) interact with Developers and PVN owners; (*ii*) store relevant data about PVNs, NetApps, VNAs, users, and the relationships between them; (*iii*) send notifications to stakeholders and run the user management; and, (*iv*) communicate and manage the repositories. The main features of each element are described in detail below.

The Web Portal

We design the Web Portal to be responsible for all the interactions between the platform and the essential stakeholders, and thus allows them to register new platforms or components, record PVN information, and trigger actions in the App2net ecosystem. We suggest dividing the Web Portal into three main modules: the Developer's Portal, the PVN Owner's Portal, and the Marketplace. The Developer's Portal provides interfaces and forms, which allow the developers to register NetApps, VNAs, and external repositories. These interfaces also enable developers to offer, withdraw, and update their NetApps and VNAs. Finally, developers can access the feedback provided by the PVN owners as well as track and manage reported bugs.

In the PVN Owner's Portal, PVN owners can register and configure their infrastructures where each PVN is formed. It is also possible to obtain an overview of all the network services deployed and a summary of how the NetApps are executed. Through this Portal, PVN owners can customize the settings of the network services deployed and,

thus adapt them to their environments and needs. It should be noted that this Portal will show the messages and notifications arising from the network services or App2net ecosystem elements. Last but not least, the Marketplace has a set of several Web interfaces that can be used by the PVN owners to find and choose the NetApps. Once the PVN has been configured, a user can trigger the deployment of the NetApps and VNAs required by using the Marketplace. If the deployment results in an error, the PVN owner will receive the message error via Marketplace and must solve the problem by trying to make the deployment again.

User Manager

The User Manager element sends notifications to stakeholders and is responsible for the user management process. This element is split into two modules, essentially User Access Controller and User Notifier. Before interacting with the platform, the PVN owner's credentials have to be checked. This verification must be made by using asymmetric keys and involves the User Access Controller carrying out actions related to user management, like authenticating and granting permission. This module also restricts access to the networks and related operations depending on the permissions assigned to each user, and prevents the PVNs and nodes from being handled improperly.

The User Notifier sends messages to both the essential stakeholders: the developers and PVN owners. When a NetApp or VNA update has been registered, the User Notifier sends a message containing data about the update, which can be done automatically (see Subsection 4.2.1). However, if an error occurs during the update process or if the PVN owner wishes to authorize this process, the Notifier module requests the PVN owner to make an intervention. It should be noted that if an error occurs and the owner does not address the problem, this module will periodically require an intervention, depending on the owner's preferences. If there is no problem, the process continues as normal. Moreover, the Notifier informs the developers when a PVN owner reports a new bug or if there is a conflict in their network services.

The use of asymmetric keys allows the App2net ecosystem to perform two important functions: (*i*) authentication, in which the public key checks if the holder of the paired private key that requests the login; and (*ii*) message encryption, in which only the paired private key holder can decode the message encrypted with the public key. Thus, an essential step for ensuring secure communication between App2net and the third-party developers,

is the generation and change of these asymmetric keys. First, a user must generate his/her pair of keys (*i.e.*, a public and private key) using, for example, a Rivest-Shamir-Adleman (RSA) algorithm implementation. Then, the developer downloads the public key from the App2net ecosystem and sends his/her public key to the registration process. After the public keys have been changed, all communication between the App2net ecosystem and developers must be established in a secure channel using the RSA keys to autenticate them, and thus mitigating classic network attacks (*e.g.*, man-in-the-middle, spoofing, eavesdropping, and other malicious actions). Likewise, PVN owners must also generate and send their RSA public keys to ensure the security of communication between PVNs and App2net ecosystem.

Databases and Repositories

We recommend that all the necessary data are stored in three different elements: DB & Statistics, NetApp Catalog, and NetApp Code Repository (NACR). In fact, these elements can be implemented in just one database; but, we advocate in favor to split them up to allow the developers and companies to host the NetApp packages and VNA images in their repositories, and thus enhance the NetApp/VNA distribution. Hence, the DB & Statistics stores all the system data and NetApp statistics, which includes the following: feedback from the users and reviews, updates, description of services, reported bugs, the number of downloads, description of the PVNs, user credentials, the repository data, and public/private keys.

The App2net ecosystem (to be more specific, the `iMPROVE` platform) saves the content of the NAD files in the NetApp Catalog, and thus includes the vendor, version, categories, dependencies, conflicts, virtual hardware requirements, internal forwarding graph, supported technologies, hashes (*i.e.*, checksums used to verify the integrity of NetApp packages), initial functional settings, and required commands. In addition, the App2net stores all the NetApp packages and VNA images in the NACR. As mentioned previously, there may be multiple external online repositories, which are managed by third-party developers or companies. It is worth pointing out that these repositories must also contain NAD files so that the Repository Gatherer can import them, and register the related NetApps and VNAs via the `iMPROVE` platform.

The Repository Gatherer

When desired NetApps are available in the NACR, the PVN owner only needs to request their installation by the Marketplace. However, the developers or even companies sometimes wish to store NetApps and VNAs in their own repository. In this case, a developer registers its repository rather than NetApps or VNAs and provides data such as the IP address, access credentials, and asymmetric keys. Then, the Repository Gatherer installs the correct driver. This driver collects information about available NetApps and VNAs and establishes the communication with the App2net ecosystem and PVN nodes.

The Repository Gatherer also configures, communicates, and manages the repositories and keeps an updated list of available NetApps and VNAs. To do this in an External Third-party Repository, the Gatherer imports all the available NAD files and sends them to the `iMPROVE` platform for registration. In addition, it must periodically check that NAD files include or update the records in the NetApp Catalog; in this way, the developer can configure the time interval required for this verification process. It should be stressed that the NetApp package or VNA image must remain in the External Third-party Repository.

Before deploying a new network service, the Repository Gatherer validates the used External Third-party Repositories that are included by using the asymmetric keys to ensure their authenticity. After this, the Gatherer checks the integrity of all the used NetApp packages or VNA images, by comparing the calculated hash with the hash stored in the NetApp Catalog. Moreover, the Gatherer sends the hash data to `app2net core` platform; this means that the `app2net core` sends this hash to the PVN nodes that will check the integrity of the NetApps after downloading them. When the deployment process has ended, the `app2net core` stores the deployment data in the DB & Statistics.

If there is no communication between the External Third-party Repositories and PVN nodes, the Repository Gatherer must be given all the necessary files from the repositories (*i.e.*, the NetApp package and its dependent packages) and send them to the PVN nodes via the `app2net core` platform. It should be noted that communication between the PVN nodes and External Third-party Repositories may not be established for several reasons, such as unauthorized access, firewall restrictions, and connectivity problems.

4.1.2 App2net RESTful API

The App2net ecosystem reveals some of its main functions through a RESTful API, which allows interested third-party developers to extend or automate a number of actions, including: (*i*) monitoring NetApps and PVNs, (*ii*) collecting data about management tasks, (*iii*) rollbacking IT changes if an error occurs; and (*iv*) triggering predefined configurations when some thresholds have been reached. As a result of the modular architecture of the App2net ecosystem, an interested developer can replace an existing element (*e.g.*, Conflict Estimator, Dependency Verifier, or Infrastructure Handler) with a new one. The new element must be able to carry out at least the same actions as the replaced element. Furthermore, third-party developers can program new modules that contain additional features. Thus, PVN owners can plug these modules into the App2net ecosystem (*e.g.*, a billing module, pricing module, or recommendation module).

As can be seen in Figure 4.2, developers can also create entirely new platforms (*e.g.*, Visualization Platform, Policy Platform, Generic Virtual Platform Manager, and Monitoring Platform). We distinguish between modules and platforms through their interaction with the App2net ecosystem. The modules provide new Web interfaces as well as some relevant data for the App2net elements and interact via App2net RESTful API. For example, a new Affinity Estimator module can collect data about deployed NetApps and PVN nodes, which are provided by the `app2net core` platform. Thus, this data could be used in a recommendation calculus to estimate which new NetApp is best suited to the environment of the deployed NetApps. In this way, the `iMPROVE` modules could use this recommendation for offering new NetApps to PVN owners via the Marketplace.

Platforms also interact via App2net RESTful API, but, they just trigger actions or consume the data made available by the App2net ecosystem. For instance, a Policy Platform could trigger management actions in nodes to set up the entire PVN and establish predefined high-level rules. This can be illustrated by the following situation, an owner observed high traffic in his/her PVN with a high rate of blocked flows during a time interval lasting 07:00 AM to 10:00 PM. However, during the night, the DPI receives almost no flows because the company is closed and none of the employees is using the network. To reduce energy expenditures, the PVN owner could write the following policy “pause the DPI from 10:00 PM to 07:00 AM”. Thus, the Policy Platform could trigger a management action to pause the DPI network service in all the PVN nodes during the predefined time interval (from 10:00 PM to 07:00 AM). In this case, the App2net ecosystem will pause

the DPI network service, and thus reduce the amount of required processing.

In another example, a Visualization Platform was able to get data about the PVN (*e.g.*, PVN nodes, deployed NetApps, and costs) to compile a new visual report that enables PVN owners to have a better understanding of how to reduce operational and capital expenditure. Although this is not specified in the methods, we would like to highlight the fact that all the requests must generate an embed token (one per session) in the login process (login method). The main exposed methods are listed below.

- **Method:** checkUpdates.

  Parameters: PVN or Network service identifier.

  Operation: Generates a list of all pending updates.

  Return: List of pending updates.

- **Method:** collectExecutionData.

  Parameters: PVN; Network service identifier.

  Operation: Returns relevant information about the NetApp execution. The returned data include those described in the NAD file by developers and the default data (see "output" in the Subsection 4.2.4).

  Return: Returns a dictionary containing all the information about the network service execution in all the PVN nodes in an aggregated way.

- **Method:** configurePVN.

  Parameters: VXDL file.

  Operation: Through this method, the App2net ecosystem can read a VXDL file and store data about the PVN in the DB & Statistics, such as node addresses, supported technologies, access credentials, the number of interfaces, supported protocols, and supported programming languages. In addition to receiving and storing PVN information, this method triggers the important tasks in our ecosystem. One of these tasks is installing drivers into the PVN nodes. These drivers make it possible to get data about the EE in each node as well as to receive system calls from the App2net ecosystem. In this way, our ecosystem can trigger processes and execute commands straight into PVN nodes, for example, by installing a NetApp, collecting data, and configuring network services.

  Return: Success or failure.

- **Method:** deployNetworkService.

  Parameters: PVN; Network service identifier; Technology identifier.

  Operation: Deploys a network service in the PVN. The deployment of a network service can result in two different actions, namely installation or instantiation. The former means that the NetApp package will be installed in the PVN nodes. The latter means that a VNA image will be started (*i.e.*, a new node) in the PVN.

Return: Success or failure.

- **Method:** getCompatibleList.

  Parameters: PVN identifier.

  Operation: Generates a list of compatible network services, in other words, network services that can be deployed in the PVN for the use of technologies and available EEs.

  Return: List containing compatible NetApps and another list containing compatible VNAs.

- **Method:** getConflicts.

  Parameters: Network service identifier; PVN identifier or Technology identifier and EE.

  Operation: Produces a conflict list in accordance with the requirements of the installed NetApps and dependency tree.

  Return: List containing all known conflicts.

- **Method:** getData.

  Parameters: PVN; Network service identifier; Data identifier.

  Operation: Returns the output about the network service execution. Among the available execution data are those described in the NAD file and the default data (see "output" in Subsection 4.2.4).

  Return: Returns a dictionary with information about the network service execution. This method only returns one parameter, but the data returned is about all the PVN nodes.

- **Method:** getDependencies.

  Parameters: Network service identifier; PVN identifier or Technology identifier and EE.

  Operation: Generates the dependency tree.

  Return: List containing all known dependencies.

- **Method:** getDeployedNetworkServices.

  Parameters: PVN identifier.

  Operation: Generates a list of deployed network services.

  Return: List containing installed NetApps and another list containing instantiated VNAs.

- **Method:** getFeedbacks.

  Parameters: Network service identifier.

  Operation: Generates a list of all the users feedbacks and scores already reported.

  Return: List containing all the users feedbacks and scores.

- **Method:** getNADFile.

  Parameters: Network service identifier.

Operation: Returns the NAD file of a network service.

Return: NAD file.

- **Method:** getStatisticsAndInfo.

  Parameters: Network service identifier.

  Operation: Returns data about the network service, including the vendor, the number of downloads, related packages, supported technologies, known conflicts, and dependencies.

  Return: Dictionary with related information.

- **Method:** listPVN.

  Parameters: credentials.

  Operation: Lists all the PVNs registered for the user.

  Return: A dictionary with data about all the PVNs registered.

- **Method:** listRepository.

  Parameters: None.

  Operation: Creates a list with information about all the registered NACRs.

  Return: A dictionary containing all the information about the registered NACRs.

- **Method:** login.

  Parameters: credentials.

  Operation: Allows a remote user or system authenticate in the App2net ecosystem. This method returns an encrypted token that must be used in all other requests.

  Return: tokenCrypt.

- **Method:** notifyEvent.

  Parameters: PVN identifier; Network service identifier; Type; Event.

  Operation: This method allows an event notification to be created for a network service.

  Return: Success or failure.

- **Method:** notifyUsers.

  Parameters: Message; Network service identifier.

  Operation: Sends notifications to subscribed users via the Web Portal.

  Return: Success or failure.

- **Method:** publish.

  Parameters: NAD file; path or URI to packages/images.

  Operation: Triggers the NetApp or VNA registration process.

Return: Success or failure.

- **Method:** registerRepository.

  Parameters: URI; Credentials; Assymetric public key.

  Operation: Registers a new external NACR maintained by third-party developers or companies. Moreover, the App2net ecosystem will import all NAD files and make these new network services available in the marketplace.

  Return: Success or failure.

- **Method:** removeNetworkService.

  Parameters: PVN; Network service identifier; Technology identifier.

  Operation: Removes a network service from the PVN. The removal of a network service can result in two different actions, namely uninstallation and deletion. The first one means that the NetApp package will be uninstalled from the PVN nodes. While the second one means that the VM that hosts the VNA image will be deleted from the PVN.

  Return: Success or failure.

- **Method:** retrieveActions.

  Parameters: Network service identifier.

  Operation: Generates a list of all supported actions and parameters.

  Return: List of available actions.

- **Method:** retrieveDeployedServices.

  Parameters: PVN identifier.

  Operation: Returns a list of all deployed network services (NetApps and VNAs) containing identifiers and technologies.

  Return: List of all deployed network services (NetApps and VNAs).

- **Method:** retrieveService.

  Parameters: Parameters; PVN identifier.

  Operation: Returns a network service list based on information about parameters.

  Return: List containing network services.

- **Method:** runAction.

  Parameters: PVN identifier; Network service identifier; Technology identifier; Action; Parameters.

  Operation: To execute a specific NetApp or VNA action through the PVN nodes that support a specific technology.

  Return: Success or failure.

- **Method:** sendFeedback.

  Parameters: Network service identifier; Feedback; Score.

  Operation: Sends the user feedback and score for a network service to the `iMPROVE` platform.

  Return: Success or failure.

- **Method:** sendMessage.

  Parameters: User; Type; Message.

  Operation: This method allows rhe network services, new platforms, or even new modules, that are plugged in the App2net ecosystem, to send messages to a user. Network services, platforms, and modules can only send messages to users that have agreed to receive messages from them.

  Return: Success or failure.

- **Method:** selectPackage.

  Parameters: Network service identifier; PVN.

  Operation: Selects the correct version of the package by taking account of conflicts, dependencies, and supported technologies.

  Return: URI to packages/images; list of dependent packages.

- **Method:** signIn.

  Parameters: Assymetric public key; credentials.

  Operation: Registers a new user in App2net.

  Return: Success or failure.

- **Method:** subscribeEvents.

  Parameters: PVN identifier, Network service identifier, or Module identifier (Source); Module identifier or Network service identifier (Destination).

  Operation: This method allows a module or network service to receive generated events from a PVN or during the execution of another network service or module.

  Return: Success or failure.

- **Method:** updateNetworkService.

  Parameters: PVN; Network service identifier; Technology identifier.

  Operation: Updates a network service in the PVN. To update a NetApp, the `app2net core` retrieves data about the installation or the last update process. By means of this data, the Package Transfer module is able to validate the NACR or External Third-party Repository; then, it informs the NACR and PVN nodes about the guidelines for the package transference. After compatible nodes have obtained the NetApp package, the Package Installer recovers the required actions from the NetApp Catalog. Following these actions, the Installer saves the current configuration, and stops the NetApp execution in all the nodes that have the network

service. Next, the Package Installer runs the necessary actions to update the NetApp. Finally, the NetApp Configurator sets up the configuration that has been saved. In the case of a VNA image, the NetApp Manager gets the current configuration from the PVN nodes; then, it deletes the VM of network service and instantiates another VM with the new VNA image version. Finally, the Configurator sets up the saved configuration for the new node.

Return: Success or failure.

- **Method:** unpublish.

  Parameters: NAD file.

  Operation: Removes a NetApp or VNA from the marketplace.

  Return: Success or failure.

- **Method:** unregisterRepository.

  Parameters: Repository identifier.

  Operation: Unregisters an external available NACR. It is worth to note that the NAD files records will be deleted from the NetApp Catalog and the network services will be removed from the marketplace.

  Return: Success or failure.

- **Method:** unsubscribeEvents.

  Parameters: PVN identifier, Network service identifier, or Module identifier (Source); Module identifier or Network service identifier (Destination).

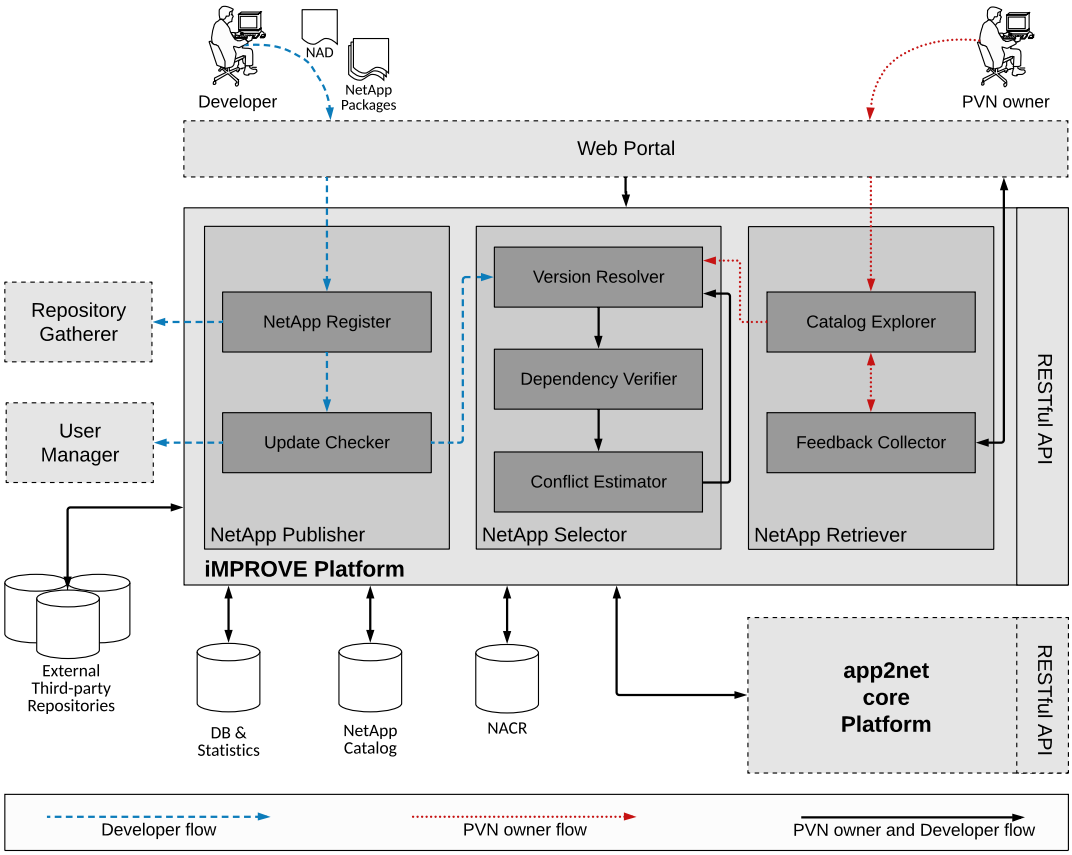
  Operation: Deletes the subscription to receive events from a PVN, network service or module. After the unsubscription, no event from the PVN or network service will be sent to the receiver.

  Return: Success or failure.

4.2 iMPROVE - Marketplace for PROgrammable Virtual nEtwork

`iMPROVE` simplifies the introduction of new services in PVNs that are based on different network technologies. It is setting up a platform that allows developers to send their network programs (*i.e.*, NetApps) to an online repository. This platform also enables PVN owners to download and install NetApps that are compatible. When a PVN owner has chosen the NetApp or Virtual Network Appliance (VNA), `iMPROVE` receives this information from the Web Portal and carries out the necessary actions to select the correct version of NetApp and solve issues of dependency and conflict. After this, `iMPROVE` triggers specific commands to install and configure the NetApp or instantiate the VNA through the `app2net core` platform (see Section 4.3).

Figure 4.3: The conceptual architecture of iMPROVE.



Source: the author (2018).

In more specific terms, `iMPROVE` carries out several actions including the following: (*i*) publishing and storing NetApps packages, VNAs images, and the NetApp Descriptor (NAD) file; (*ii*) enabling PVN owners to find and download NetApps and VNAs; (*iii*) verifying issues of conflict; (*iv*) checking dependencies; and (*v*) establishing communication with the `app2net core` platform to trigger processes. Thus, the `iMPROVE` platform acts as an orchestrator, since it contacts other elements and requests data from them. For instance, the platform retrieves information about NetApps from the NetApp Catalog, so that it can determine the compatibility of NetApps. It also attempts to match its requirements with those of the PVN (*e.g.*, supported programmability technologies and used EEs), by finding out and displaying what compatible NetApps there are in the Marketplace. Finally, the PVN owner can select which NetApps will be installed. In Figure 4.3, the main components of `iMPROVE` are shown: NetApp Publisher, NetApp Retriever, and NetApp Selector. In the next section, there is a description of these components and their modules.

4.2.1 The NetApp Publisher

We first describe how a developer releases a new network service. It must first send the NetApp package, VNA image, and NAD file through the Web Portal. Later, the NetApp Publisher registers it and makes the network service available to other users in the Marketplace, as a means of allowing the PVN owners to deploy it in their networks. The NetApp Publisher is split up into two main modules to ensure the platform is kept simple and modular, as described below: NetApp Register and Update Checker. The NetApp Register module imports data from the NAD file and stores them in the NetApp Catalog. After this, this module checks whether the network service has been registered, (*i.e.*, if there is a record containing the vendor and identification in App2net DB & Statistics). If there is no record, the Register stores the initial data about the network service lifecycle in the DB & Statistics, such as categories, the company or developers responsible, last update, and descriptive features.

When the developers update an existing network service (*i.e.*, when there is a record containing the same vendor and identification, but the version is different), the Update Checker module checks which PVNs are affected. This module obtains the preferences of the PVN owners from App2net DB & Statistics, and checks which NetApps packages or VNA instances need to be updated. Some NetApps/VNA cannot be automatically updated because of critical issues such as conflicts, unsuitable virtual hardware requirements, or even the need for user confirmation. In these cases, the Checker sends these problems to the User Notifier for human intervention.

4.2.2 The NetApp Retriever

The developers and PVN owners interact with the NetApp Retriever component, which carries out a number of actions at various stages of the process. The main function of the NetApp Retriever is to assist the users in finding the required services provided by NetApps or VNAs. Furthermore, the Retriever collects the feedback from the users and sends it to developers and thus provides important data about the NetApp lifecycle (*e.g.*, bugs found or new features requested). For the sake of simplicity, this component is divided into two modules: Catalog Explorer and Feedback Collector.

When choosing a network service, the PVN owner interacts with the Catalog Explorer through the Marketplace. First, the Explorer retrieves information about the PVN from

App2net DB & Statistics such as supported technologies, installed NetApps, feedback from the users, and virtual hardware specifications. The NetApp Catalog provides data about the available NetApps and then the Explorer module conducts a matching operation with these data, which only shows the NetApps and VNAs that are compatible. The PVN owner can specify the categories and keywords that are needed to refine the obtained results, and thus enhance the search process. Finally, after a network service has been chosen, this module triggers the NetApp Selector so that it can pick out the correct NetApp package or VNA version and checks if there are any possible conflicts and dependencies.

The Feedback Collector is responsible for obtaining the users' evaluations for each network service and storing them in App2net DB & Statistics. These evaluations contain a textual review and a rating score. The Collector module enables the users to report bugs, errors, and unknown conflicts. Thus, this module is able to notify the responsible developers of any problems that must be resolved. Likewise, when the developer submits a patch to correct a bug, the Collector informs the interested PVN owners (who then report it) about the update that has been received.

4.2.3 The NetApp Selector

This component is responsible for selecting the correct NetApp package, while taking into account the version, virtual hardware requirements, supported technologies, and other installed NetApps. It should be noted that the Selector only determines which version of the NetApp package will be used on the basis of issues of dependency and conflict. For the sake of clarity and simplicity, we have divided the Selector into three modules: Version Resolver, Dependency Verifier, and Conflict Estimator.

Both of the action flows from the developers and owners arrive at the Version Resolver (Figure 4.3). Initially, this module requests NetApp Catalog information about all the versions of the desired network service. On the basis of the installed NetApps, instantiated VNAs, and PVN data, the Resolver identifies the last version that is compatible; this means, the last version of the NetApp package or VNA image that is supported by the PVN and in accordance with both the technologies and virtual hardware requirements. Although the Version Resolver selects a compatible version, there are sometimes conflicts with the installed or dependent NetApps and, in this case, in the next iteration, the Resolver selects another NetApp version.

Once one NetApp or VNA version has been selected, the Dependency Verifier module

will check the dependency tree. The Verifier does this by obtaining the NetApp dependencies, which are given information by the developers in the NAD file and stored in the NetApp Catalog. Afterward, this module finds out if the NetApps have been installed in the PVN and then, determines which dependencies are needed. In this way, the Verifier determines the nested dependencies that form the NetApps list for each dependent NetApp. It should be stressed that if a dependent NetApp has already been installed, it is not included in the NetApps list.

On the completion of these steps, the Conflict Estimator module conducts the conflict analysis. First of all, this module obtains the necessary information, such as the installed NetApps, the dependency tree, and supported technologies. The Estimator then compiles the conflicts of all the installed and dependent NetApps in a known conflicts list. Following this, the Estimator establishes if there is any dependent or installed NetApp in it. Whenever the Estimator finds a conflict, the selection process restarts and returns to the Version Resolver which then ignores the used NetApp version in the next iteration. If by the end of the process, no available version fulfills the requirements, the PVN owner is requested to make an intervention to address the problems that have been found. In addition, the module registers a new bug for the responsible developers with the used data, and sends a message to inform about the bug. It is important to note that the developers can employ several mechanisms to identify the conflicts and find other means of mitigating them.

4.2.4 NetApp Descriptor

The ETSI specifies the NSD file (NFV ISG, 2014) which provides details of the network service in a simple XML notation, including the vendor, identification, version, dependencies, and virtual hardware requirements. In this work, the ETSI NSD has been extended to support all the features of the App2net ecosystem. Our extension, called NetApp Descriptor (NAD), makes it possible to describe the NetApps and VNAs of several programmability-related technologies as well as their specific features. This is based on ETSI NSD, which only supports the NFV technology; thus, several tags have been included to support different technologies.

Our extension includes two tags under the heading of the NSD: *categories* and *packages*. The first tag (*categories*) allows us to add several keywords related to the network service. This tag enables us to merge the similar network services together, as well as

Table 4.1: Extended tags for the NetApp descriptor.

Tag	Parent Tag	Description
category	categories	Adds a category (also known as the keyword) related to the network service. This groups together the similar network services to be grouped together as well as helping the PVN owners to find a correct service.
technology	packages	The main tag to provide information about the technologies that are supported by the network service.
identifier	technology	The string that identifies a programmability technology, for instance, openflow, nfv, and cisco-onepk.
version	technology	A supported version of the programmability technology.
location	technology	The main tag to describe the URIs of the NetApp packages or VNA images.
location_flag	technology	This flag takes into account the application-network interaction and the location of the nodes within the PVNs. Thus, it stipulates in which nodes the App2net ecosystem must install the NetApp (*e.g.*, *custom* nodes, *border*, *ingress*, *egress*, or *all* the nodes that are possible values).
ee	technology	The execution environment necessary to run the NetApp.
type	technology	Describes the type of package in detail. Two values are supported “netapp” and “vna”.
hash	technology	Informs the Secure Hash Algorithm-256 (*e.g.*, SHA-256) hash about the package. This hash is used in the validation process.
outputs	technology	Makes new relevant information available about the NetApp execution. PVN owners can get these data through the Web Portal or external platforms. It can also get it via `collectData` (in an aggregated way) or `getData` (in an individual way) methods from App2net RESTful API.
conflicts	technology	Maps all the known conflicts between the NetApps and VNAs; this means, this NetApp or VNA operates improperly when the NetApp/VNA target is used.
initial_config	technology	Describes the initial functional settings of a NetApp, which might be a simple command or a set of commands.
manage_action	technology	The main tag to specify the necessary commands and parameters that are needed to manage a NetApp (*e.g.*, install, start, restart, pause, resume, uninstall, and configure). Note that only the supported actions are described.

helping the PVN owners to find a correct service. The second tag (*packages*) describes all the information about the packages including conflicts, supported technologies, initial functional settings, available management actions, version, and the necessary EE. In Table 4.1 there is a description of the main tags proposed for our extension.

In Figure 4.4 there is an example of a NAD file for a video transcoder and cache.

Figure 4.4: An example of a NAD file that employs the proposed tags.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 ...
3     <categories>
4         <category>video</category>
5         <category>transcoding</category>
6         <category>caching</category>
7     </categories>
8     <packages>
9         <technology>
10            <identifier>openflow</identifier>
11            <version>1.0</version>
12            <ee>pox</ee>
13            <type>netapp</type>
14            <hash>29a8b9009509b39d542ecb229787cdf48f05e739a9322895e9e9858d7c487c80</hash>
15            <location>
16                <uri>file:///repository/services/openflow/pox/video-trans-caching-1.py</uri>
17            </location>
18            <location_flag>all</location_flag>
19            <initial_config>
20                <command>video-trans-caching-1.py --set-video=640 --set-cache=100MB</command>
21            </initial_config>
22            <manage_action>
23                <install>./video-trans-caching-1.py</install>
24                <start>./video-trans-caching-1.py --start</start>
25                <stop>./video-trans-caching-1.py --terminate</stop>
26                <configure>
27                    <parameter>
28                        <value>video</value>
29                        <command>./video-trans-caching-1.py --set-video=</command>
30                    </parameter>
31                    <parameter>
32                        <value>cache</value>
33                        <command>./video-trans-caching-1.py --set-cache=</command>
34                    </parameter>
35                </configure>
36            </manage_action>
37            <conflicts>
38                <conflict>
39                    <identifier>transcoding-roxio</identifier>
40                    <version>&amp;lt; 2.1</version>
41                </conflict>
42                <conflict>
43                    <identifier>squid-cache</identifier>
44                    <version>all</version>
45                </conflict>
46            </conflicts>
47            <outputs>
48                <output>flows_processed</output>
49                <output>codecs</output>
50                <output>bitrate</output>
51                <output>available_video_sizes</output>
52                <output>compulsory_misses</output>
53                <output>capacity_misses</output>
54                <output>conflict_misses</output>
55            </outputs>
56        </technology>
57    </packages>
58 ...
```

Source: the author (2018).

This NAD describes a NetApp (line 13), that means, an application that must be installed in the PVN nodes. As can be seen, there are several categories (lines 3 to 7) that assist PVN owners in finding this NetApp, such as video, transcoding, and caching. Lines 8 to 57 give information about the available package. In this case, there is just one package, which is supported by POX controller using OpenFlow 1.0 (lines 10, 11, and 12). This NetApp package is available in the NACR of `iMPROVE` because the URI points out the local file system (line 16).

With regard to the location flag, the package will be downloaded by all the PVN nodes

(line 18). After downloading corresponding files, the platform can carry out the package validation using the informed SHA-256 hash (line 14). In this NetApp, just one command can achieve the initial configuration (lines 19 to 21), which must be carried out in every PVN node. Also, PVN owners or `iMPROVE` can trigger four management tasks (lines 22 to 36), namely, install, start, stop, and configure. In specific terms, the configure task can be adjusted by two parameters: (*i*) video, which sets the resolution of video streaming (lines 27 to 30); and, (*ii*) cache, which sets the size available for the used cache (lines 31 to 34). Moreover, developers describe two known conflicts. The first one states that this NetApp is incompatible with versions older than 2.1 of NetApp transcoding-roxio (lines 38 to 41). The second one indicates that NetApp will operate improperly if the NetApp squid-cache is installed in the PVN (lines 42 to 45).

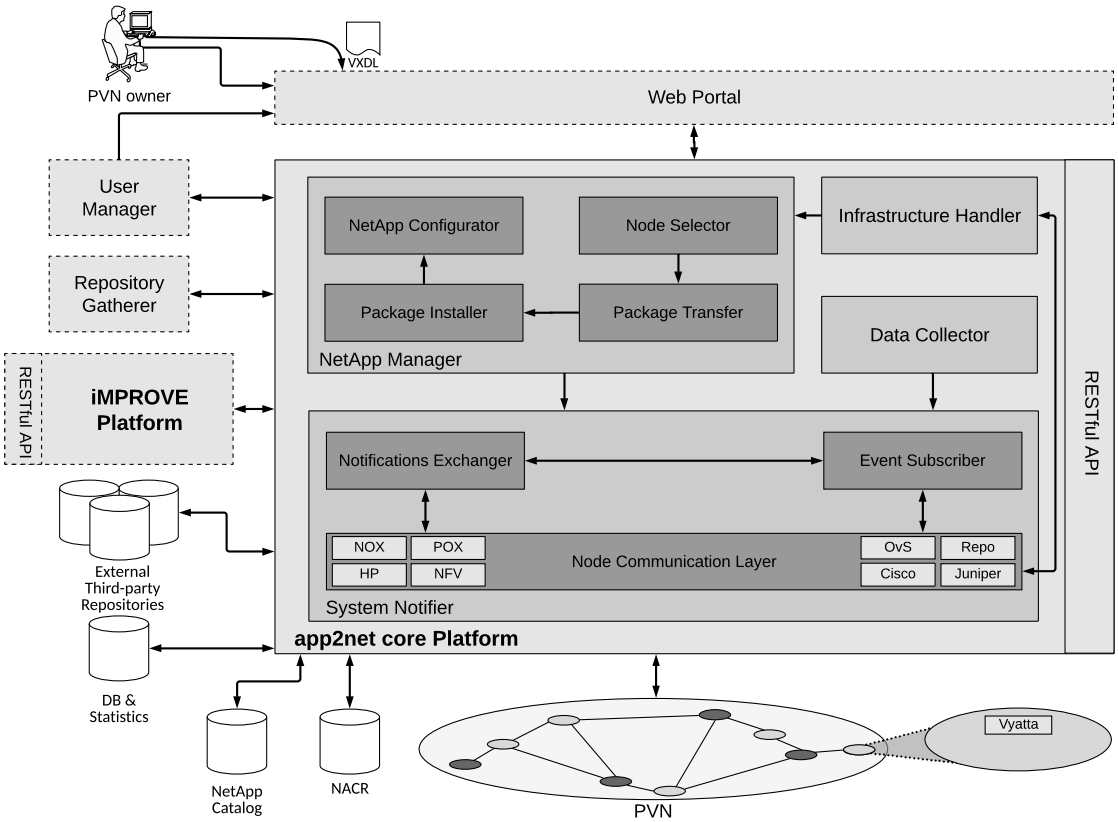
In addition to default data, third-party developers have made new output data available for the execution of the network service (lines 47 to 55), namely the following: processed flows, codecs, bitrate, available video sizes, compulsory misses, capacity misses, and conflict misses. By employing the *getData* and *collectData* methods of App2net RESTful API, PVN owners can obtain values associated with each output. As well as this, another developer (or even a PVN owner) can program a new management script or platform that relies on these values to trigger configurations or other actions. For instance, a management script can monitor all the output data; and, by observing a high number of capacity misses, it is able to trigger a request to increase the cache capacity, and hence the storage capacity of the VM. In the same way, another solution could reduce the video size or the average bitrate.

4.3 App2net Core - Applications to Network

The essential purpose of the `app2net core` is to deploy NetApps in PVNs that use heterogeneous EEs. These EEs can be hosted on virtual or physical devices. We assume that the deployment of a NetApp consists of three key actions, namely transfer, install, and configure. To this end, the App2net ecosystem offers the PVN Owner's Portal that allows owners to interact with the `app2net core` when carrying out actions without knowing the specific features of the underlying environment. In this way, the PVN owner is able to invoke the `app2net core` platform components needed to deploy the NetApps, and thus avoid complex interactions with the PVN.

In Figure 4.5, there is a detailed diagram of the conceptual architecture of the `app2net`

Figure 4.5: The conceptual architecture of App2net.



Source: the author (2018).

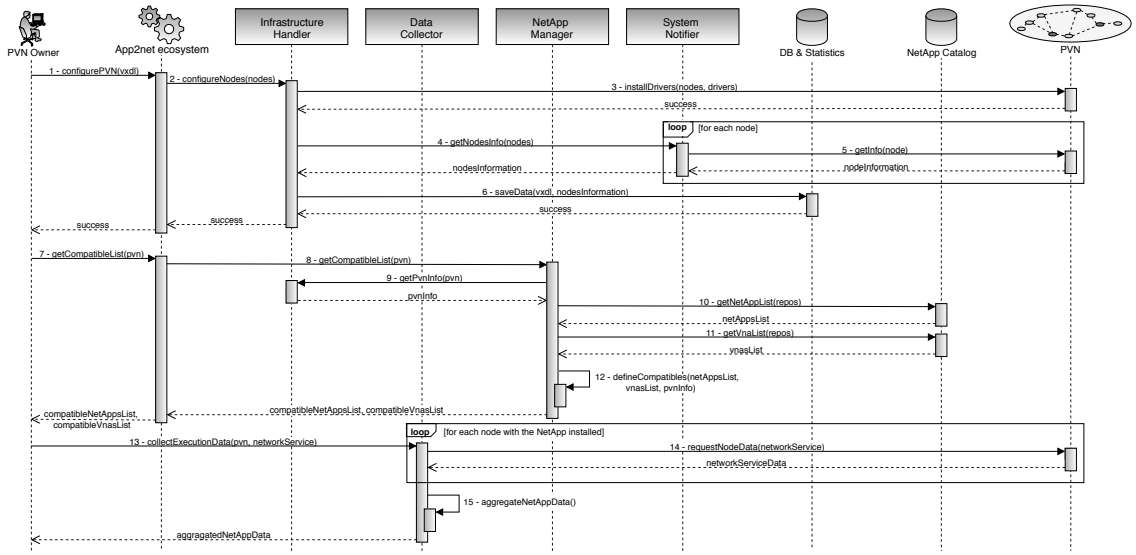
`core` and its four main components: Infrastructure Handler, NetApp Manager, Data Collector, and System Notifier. Moreover, Figure 4.6 shows the main actions performed by the PVN owner in the `app2net core`.

4.3.1 The Infrastructure Handler

After logging into the App2net ecosystem, the PVN owner interacts with the PVN Owner's Portal to configure the Infrastructure Handler with the nodes that form the PVN. This only involves supplying information about the node addresses, supported programmability technologies, and access credentials (actions 1 and 2). This configuration is either carried out manually or imported from a Virtual Infrastructure Description Language (VXDL) file (KOSLOVSKI; PRIMET; CHARAO, 2009).

We chose VXDL because it represents the infrastructure elements in a simple and clean XML notation. It should be noted that the nodes that comprise the PVN can be hosted both on the user's infrastructure and on multiple geographically distributed Infrastructure Providers, as shown in Figure 4.1. The Infrastructure Handler is responsible for:

Figure 4.6: Sequence diagram of the main actions carried out in the app2net core.



Source: the author (2018).

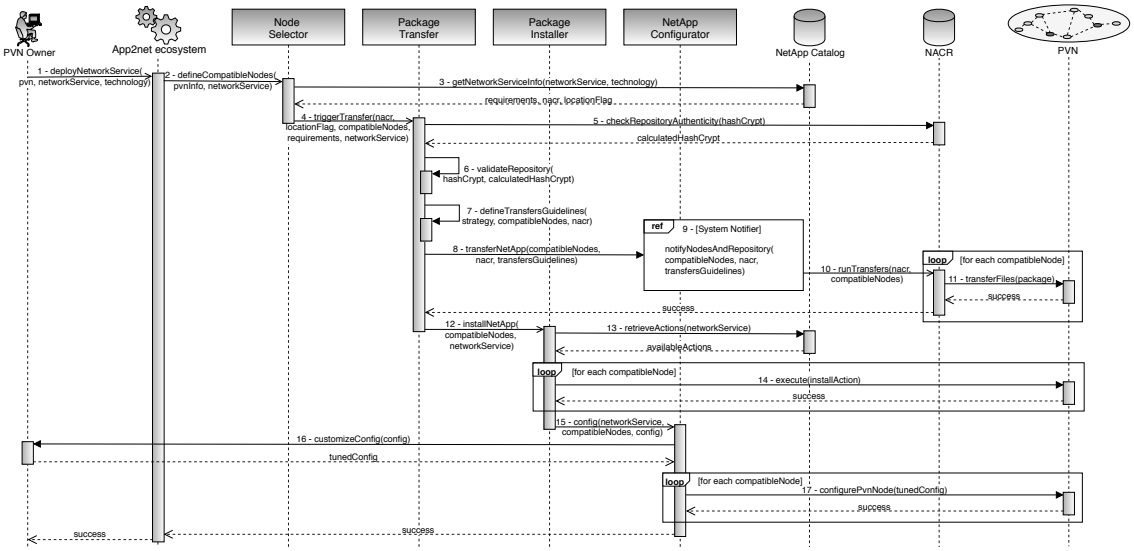
(*i*) installing drivers in the nodes (action 3); (*ii*) collecting data about the resources in the PVN nodes, for example, network interfaces or programming languages (actions 4 and 5); (*iii*) maintaining the list of available devices per PVN; and, (*iv*) storing the VXDL file and credentials for accessing devices in the DB & Statistics (action 6).

4.3.2 NetApp Manager

When required, NetApps are available in the NACR; the PVN owner only needs to request their installation by the Marketplace that triggers the deployment process in the NetApp Manager. The NetApp Manager is responsible for transferring, installing, uninstalling, and configuring NetApps. This Manager communicates and requests information from the other components (actions 7 and 8). Initially, the Manager retrieves information about the PVN nodes and their resources from the Infrastructure Handler (action 9). Then the NetApp Manager retrieves a list of available NetApps (action 10) and a list of available VNAs (action 11) from the NetApp Catalog. After this, the Manager carries out the task of matching the requirements with the resources in the nodes; this means it is only possible to deploy the NetApp or VNA that is present in the compatible NetApps List or compatible VNAs List (action 12). The NetApp Manager is formed of four modules: the Node Selector, Package Transfer, Package Installer, and NetApp Configurator. Figure 4.7 shows the actions performed by the NetApps modules when deploying and configuring a NetApp in a PVN. It should be noted that all the messages to repositories or the PVN

are sent via the System Notifier, but these interactions have been hidden for the sake of readability.

Figure 4.7: Actions carried out by the modules of the NetApp Manager when installing and configuring a new NetApp.



Source: the author (2018).

When a PVN onwer triggers the deployment of a new network service (Figure 4.7, action 1), the Node Selector module identifies the compatible nodes and the NACR, which holds the corresponding package (actions 2 and 3). A compatible node can install a specific NetApp that meets the minimum requirements for both software and hardware. After this, the Selector defines which compatible nodes must install the NetApp, depending on the requirements and the location flag (action 3) provided by the NetApp Catalog. This flag takes into account the application-network interaction and the location of the nodes within the PVNs. This means it is possible to install and run NetApps at strategic points of the PVNs. For instance, a deep packet inspection firewall uses the location flag with a "*border*" value; thus, it is only installed on the border nodes, which avoids additional and unnecessary computing.

The location flag can take on five values: (*i*) *border*, this refers to boundary nodes and all the data flows in a particular PVN; (*ii*) *ingress*, which includes the boundary nodes and input data flows; (*iii*) *egress*, which denotes that the boundary nodes and output data flows will be used; (*iv*) *custom* this allows a PVN owner to select which nodes and data flows will be used by the NetApp; and (*v*) *all* refers to all the nodes and all the data flows. It is important to mentioning that, when designating the location flag with a "*custom*" value, the NetApp developer should set out the rules for handling the data flows; however, the rules for choosing the nodes cannot be specified.

The Package Transfer module receives information about the PVN nodes from the Node Selector (action 4). Initially, the Transfer validates the authenticity of the NACR by means of a hash that is encrypted by means of asymmetric keys (actions 5 and 6), which are stored in the DB & Statistics. Next, this module sets out the guidelines (*e.g.*, order and initial transfers) that are required for transferring a copy of the NetApp package to each network node (action 7). Afterward, the Package Transfer gives information about these guidelines to the drivers of the NACR and PVN nodes, which perform the package transfer (actions 8 and 9). Finally, the driver validates the package integrity by means of the SHA-256 hashes that appear in the NetApp Catalog. For the sake of simplicity in presentation, in Figure 4.7 there is a summary of the actions to inform the drivers about the guidelines of transfers, as well as the package validation is concealed.

Once a NetApp has been transferred (actions 10 and 11), the Transfer instructs the Package Installer to install the NetApp (action 12). After this, the Package Installer obtains the available actions and adjustable parameters from the NetApp Catalog (action 13); and then, it sends the commands and parameters to install the NetApp in each compatible PVN node (action 14). With the aid of the NetApps Configurator, a PVN owner can issue input commands to adapt and tune a NetApp execution to a specific PVN environment (actions 15, 16, and 17). It should be stressed that NetApp developers provide input commands and parameters in the NAD file through the registration process previously mentioned. By ending the installation process, the NetApp Manager is able to save all the data about the deployment in the DB & Statistics.

4.3.3 Data Collector

A PVN owner can request data about the execution of a network service (Figure 4.6, action 13). In this case, the Data Collector component requests the network service data about its execution from each PVN node (action 14), namely, status, processed packets, execution time, uptime, average CPU usage, memory usage, storage usage, packet losses, bandwidth usage, and latency. In addition to these data, developers can make available new metrics about their NetApps. For this reason, they must specify new values under tag "*outputs*" in the NAD file (see Subsection 4.2.4). For instance, in a deep inspection firewall NetApp, the developers could make available other relevant information, such as rules, rule age, intrusion attempts, dropped flows, IPs blocked by Denial-of-Service (DoS) attacks, error rate, packet processing rate, and node-internal transfer delay. After

receiving data from all the nodes, the Data Collector aggregates them and sends a report to the PVN owner (action 15). Besides, these data can be used to supply a monitoring system and, in this way, check whether the NetApp is operating correctly and to provide performance statistics.

4.3.4 System Notifier

The System Notifier consists of three modules, namely, Notifications Exchanger, Event Subscriber, and Node Communication Layer. The first is responsible for sending all the messages from the `app2net core` components to the nodes, as well as receiving messages from the PVN nodes. For instance, the exchanged messages contain data about the following: (*i*) package transfers, (*ii*) node priorities, (*iii*) used strategies, (*iv*) NetApps management actions, (*v*) data for NACR authentication, (*vi*) data to validate packages, (*vii*) data from the execution of NetApps, (*viii*) busy notifications, (*ix*) deletion requests; and, (*x*) commands that must be performed in the PVN nodes. These messages are coded in the JavaScript Object Notation (JSON) format. We decided to use this format because it is lightweight and simple, and allows a quick exchange of messages.

PVN owners can set up the `app2net core` to trigger events on another platform. To this end, owners must interact with the Event Subscriber module to indicate the interested NetApps or PVNs. For example, the PVN owner registers a URL in Event Subscriber, which must be triggered if a management event (*e.g.*, deploy, install, or those detailed in the NAD file) or even an error occurs. This URL points to a Monitoring Platform. In this case, if a developer upgrades a NetApp and, the update process results in a NetApp misbehaving or instability, the Monitoring Platform is able to detect this failure and request the downgrading of the NetApp via App2net RESTful API (see Subsection 4.1.2). It is worth highlighting that the PVN owners must register URLs for each event and for the network service that they are interested; in this way, the Event Subscriber can trigger different URLs per event (in the PVN) or per network service.

The PVN nodes, NACR (internal), and External Third-party Repository (external) must have a *driver* installed, which is coded and adapted to the features of each device. Thus, different drivers are needed for each kind of technology, such as Vyatta, Cisco ONE, Juniper Junos, and POX controller. The set of used drivers forms the Node Communication Layer. This Layer carries out the following functions on PVN nodes: (*i*) executing commands to deploy and uninstall NetApps, (*ii*) collecting data from NetApps,

(*iii*) triggering commands to configure the execution of NetApps, (*iv*) establishing the communication between the nodes, (*v*) estimating the length of the delay to the NACR, (*vi*) answering requests from the `app2net core` components; and, (*vii*) managing the registration of the nodes and all the lists. It should be noted that if a PVN node is unable to host a driver, the `app2net core` server can host the driver on its behalf. Thus, communication between the `app2net core` and the PVN nodes will be guaranteed, even though delays might be significantly longer.

4.4 Code Transfer Techniques

Different research areas have employed several techniques to perform the code transfer, such as cloud computing (ZABOLOTNYI; LEITNER; DUSTDAR, 2013), controller updates to SDN (VANBEVER et al., 2013), mobile networks (NAWAF; TORBEY, 2009), and software updates (PAYNE; SHAIO; HOFF, 1999). However, these techniques have been described by different names and, no study has attempted to classify and compare them.

Regarding the scope of this thesis, diverse tasks have different and conflicting requirements at each stage of the service lifecycle; then, we aim to analyze distinct ways for transferring NetApps packages and initial configurations, as well as the main benefits and drawbacks of each technique in the PVNs context. For instance, at the developing stage, the developers want to distribute both NetApp packages and initial configurations in the shortest time possible. This is because they must perform several tests using different versions and, consequently, they must distribute the files several times. However, a PVN owner desires a minimal network interference (*i.e.*, lower total traffic in the network) when triggers the NetApp installation process (only once).

In view of this, we have sought to define the main features of the code transfer techniques and arrange them in accordance with their purposes. Hence, they can be compared and analyzed in the context of the PVNs. Initially, three groups were categorized: ranking criteria, mechanisms, and strategies. In the next subsections each of these will be examined in detail.

4.4.1 Ranking Criteria

Different ranking criteria can be used to define the node priority policies during the code transfer. These ranking criteria are metrics and rules that are adopted to calculate priorities and sort the PVN nodes into different categories; thus, it is possible to ensure that the nodes with a high priority receive the NetApp during the first transfers. In App2net ecosystem, we suggest that the nodes were ranked on the basis of three criteria: available bandwidth, average delay, and node degree. In an average delay, the node with the lowest absolute value has the highest priority. Hence, the node with the lowest average delay is the first to download the NetApp package. Instead, in the case of available bandwidth and node degree, the node with the highest absolute value has top priority, since it is the first to download the NetApp package. Of course, based on PVN owners requirements, the App2net ecosystem can use other criteria, for instance, costs, energy, betweenness centrality, closeness centrality, eigenvector centrality, throughput, packet loss, latency, or jitter.

The *available bandwidth* of a link is the serviceable capacity for data transmission in a time interval (JAIN; DOVROLIS, 2002). This value is obtained by applying Equation 4.1, in which: (*i*) $A_i^\tau(t_0)$ represents the available bandwidth of link i during a time interval $(t_0, t_0+\tau)$, (*ii*) C_i is the capacity in link i; and (*iii*) $u_i^\tau(t_0)$ is the average utilization of link i during a time interval $(t_0, t_0 + \tau)$, with $0 \leq u_i^\tau(t_0) \leq 1$. On the basis of this criterion, initial transfers can become faster because bottlenecks are avoided at the beginning of the NetApp distribution process.

$$A_i^\tau(t_0) \equiv C_i[1 - u_i^\tau(t_0)] \tag{4.1}$$

The delay is the time for a packet to travel across the network from one node (the source) to another node (the destination). Here, the criterion for *average delay* is the average time (in milliseconds) obtained through the Round-Trip Time (RTT) of packets between the source and destination nodes during a time interval. In this way, the NetApp distribution occurs earlier when it is close (short RTT) to the source and/or in idle nodes. Afterward, the NetApp package is sent to the distant and busy nodes.

According to Graph Theory, the *node degree* represents the number of links connecting to a node. On the basis of this criterion, high degree nodes are privileged and receive the NetApp package at the initial stage. Subsequently, these nodes are able to send the received NetApp package to a larger number of neighbors, which makes the NetApp

distribution faster. Two nodes can be regarded as neighbors when there is a direct link between them.

4.4.2 Mechanisms

Mechanisms define the behavior of NetApp sources, and are just called *repositories*. In PVNs, these repositories can be the NACR (also known as internal repository), External Third-party Repositories, or PVN nodes that have already received the NetApp package. We also examine three key mechanisms: push-based, pull-based, and hybrid. In *push-based*, the repositories are "*active*", which means that they start the NetApp package transfers to the nodes. The Package Transfer module creates a list of the initial transfers and priorities of the PVN nodes, called the *transfer list*. In this mechanism, the System Notifier only sends this list to the repositories involved. Next, the repositories send the NetApp package to the nodes in the transfer list, in accordance with the priorities of the nodes. It should be pointed out that the high-priority nodes (those on the top of the list) receive the NetApp at the initial stage.

In contrast, in *pull-based*, repositories are "*passive*"; that means, they just answer to the requests from the PVN nodes. The System Notifier component sends messages to nodes concerning which NetApps must be installed or upgraded. This component also gives information about the calculations made to determine the priority of the nodes. After that, each node clearly requests a given NetApp package from the repository. If the repository cannot send the NetApp package immediately, the requests from these nodes are stored in a *standby list* ordered by priorities. Then, the repository sends the NetApp to those nodes at the top of the standby list, as soon as possible.

In the *hybrid* mechanism, the repositories behave in both an active and passive way. In addition, the repositories both manage and use the transfer list and the standby list. First, with regard to the initial transfers (at the *push stage*), the repositories send the NetApp package directly to the nodes in the transfer list (active). In subsequent transfers (the *pull stage*), the nodes request the NetApp package from the repository (passive). Finally, if the requests are not answered, they are stored in the standby list and, after that, the repositories will send the requested NetApp.

4.4.3 Strategies and Models

Strategies are the guidelines that control the behavior of PVN nodes during the NetApp distribution process, as well as the number of supported simultaneous transfers. On the basis of techniques found in the literature, we set out four strategies to transfer NetApps packages in PVNs: Sequential, Parallel, Gossip, and Groups. Once the code transfer techniques (ranking criteria, mechanisms, and strategies) have been grouped and defined, we combine them in a way that allows us to build *models*. A model consists of a single strategy that is applied to one particular mechanism and only employs one criterion for ranking the nodes. In the `app2net core`, these models were implemented in the Package Transfer module. A detailed account of the strategies and developed models that are employed is given below.

Sequential

The first strategy simulates a simple script that sends one NetApp package to the PVN nodes; thus, there is only one transfer at a time. In this case, the NACR only sends the NetApp to *node B*, after completing the transfer to *node A*. This is repeated until all the nodes have the NetApp package. Since this strategy is an elementary, automatic, and realistic approach, which could be applied by administrators in a real PVN; we decided to use it as the baseline to compare its results with those obtained from the more elaborate strategies. The implemented models in this strategy have the following features. In the push-based mechanism, after receiving the transfer list, the NACR sends the NetApp package to each node, and only starts a new transfer when it has finished the previous one.

In the pull-based mechanism, the `app2net core` informs the nodes about the NetApps packages that are needed and the criterion used. After, the nodes request these packages from the NACR, which only answers one node at a time. When the NACR cannot give an immediate answer, the requests from the nodes are stored in a standby list that is arranged in order of priority. The NACR calculates the priorities on the basis of the information supplied by the nodes. When a transfer is completed, the NACR selects a high-priority node to send to the NetApp.

In the hybrid mechanism, the `app2net core` sends a transfer list to the NACR. This list only includes half of the nodes, which have higher priority. In the push stage, the

NACR sends the NetApp directly to each node in the transfer list. At the same time, the `app2net core` informs the remaining nodes about the NetApp that is needed. These nodes request the NetApp from the NACR, which stores all the download requests in the standby list. After completing the push stage, the NACR starts the pull stage and, then, answers the download requests from the nodes in the standby list in accordance with their priorities.

Parallel

The second strategy aims to speed up the time to distribution by performing several transfers at the same time (*i.e.*, transfers in parallel). Thus, this strategy optimizes the NACR bandwidth, avoiding to wait by congested links and low processing nodes. The NACR simultaneously sends the NetApp package to multiple nodes until a threshold is reached that is called the *quota*, which is calculated by Equation 4.2. In this: (*i*) n is the number of PVN nodes, (*ii*) $d(node_i, node_j)$ denotes the length of the shortest path in hops between $node_i$ and $node_j$; and (*iii*) $node_i$ and $node_j$ are the nodes of a connected graph. This equation calculates the average shortest path length, which is a standard network topology measurement. This measurement adapts the quota to the number of nodes and calculates the average distance between them. Thus, there will be a large quota of PVNs with many distant nodes; as a result, several transfers will occur in parallel, and this improves the overall distribution time. It is important to note that the quota equation considers the complete PVN topology.

$$quota = \sum_{i \neq j} \frac{d(node_i, node_j)}{n(n-1)} \tag{4.2}$$

In the push-based mechanism, after the transfer list has been received, the NACR sends the NetApp package to *x* nodes at the same time. It should be noted that *x* is the limit defined by the quota. Similarly, in the pull-based mechanism, the NACR accepts *x* simultaneous requests from the nodes. In this way, the *x* nodes download the NetApp in parallel; at the same time, the remaining requests are stored in the standby list. In both mechanisms, after completing a transfer, the NACR sends the package to the node with the highest priority in the transfer list (push-based mechanism) or in the standby list (pull-based mechanism).

The hybrid mechanism combines the two previous behavioral patterns. In the push

stage, the `app2net core` sends the transfer list to the NACR where it only includes half of the PVN nodes. Following this, the NACR simultaneously sends the NetApp package to the x nodes. After completing a transfer, the NACR sends the NetApp to the next node with the highest priority in the transfer list. This process is repeated until all the nodes that are included in the transfer list have received the package. At the same time, the NACR waits for the download requests and stores them in the standby list. When the pull stage starts, the NACR sends the NetApp package towards the x nodes in parallel. Again, after a transfer is completed, the NACR answers the request from the node that has the highest priority on the standby list.

Gossip

This strategy initially behaves like the parallel one, where the quota limits the number of simultaneous downloads. However, the nodes that receive the NetApp package become *ghost repositories*. A ghost repository sends the package to the remaining nodes, and thus acts as an extra source of the NetApp packages. Hence, the number of sources is incremental, and optimizes the NetApp distribution over a period of time. This strategy adds a shared list; this is located in the NACR and contains all the nodes which must receive the NetApp package. In the ghost repositories, there are local lists that contain all the neighbors (*i.e.*, just nodes directly connected with them), which have not received the package already. These lists allow there to be multiple sources at the same time. It should be underlined that before making a download from a ghost repository, the nodes must be removed from the shared list. The node achieves this by sending a deletion request to the NACR.

In the push-based mechanism, the NACR sends the NetApp package to nodes in the transfer list. Again, the quota restricts the number of parallel downloads. After the nodes receive the NetApp, they become ghost repositories. These nodes check which of their neighbors are in the shared transfer list, which is located in the NACR. Each ghost repository includes all the neighbors that have not yet received the NetApp in this local transfers list and, then, it sends a deletion request to the NACR to remove its neighbor from the shared transfer list, which must be confirmed. Subsequently, the ghost repository sends the NetApp package to their neighbors, which after receiving the package, become new ghost repositories. Note that the number of parallel transfers is limited by the quota computed.

In the pull-based mechanism, after becoming a ghost repository, the nodes create a local standby list, which only takes account of the neighbors that remain in the shared standby list. The ghost repositories then send a message to the nodes in their local standby lists to give information about the NetApp available. If a node responds with a download request, the ghost repository sends the NetApp package away. At the same time, the node sends a deletion request to the NACR to remove it from the shared standby list. However, if a node answers with a "busy notification", it is removed from the local standby list. A busy notification means that a node is downloading the NetApp, or else that the transfer has been completed.

The hybrid mechanism only has slight variations from the previous models. The essential difference is that the NACR only sends the NetApp via push-based calls, whereas the ghost repositories only send the NetApp through pull-based calls. It is worth underlining that ghost repositories do not send deletion requests to the NACR, and that only the PVNs nodes do this. In addition, the push and pull stages occur simultaneously.

Groups

In this strategy, the PVN nodes are divided into groups, in which the group members will be interested in the same NetApp. The number of groups is equal to the quota (see Equation 4.2); thus, in PVNs with a lot of distant nodes, there will be many groups, which leads to an increase in the number of ghost repositories and also improves the overall distribution time. First of all, Package Transfer module performs a master election among all nodes of the PVN in accordance with the ranking criteria (*i.e.*, available bandwidth, average delay, or node degree). Note two important guidelines: (*i*) each group has a single master; and, (*ii*) if there are several nodes candidates tied in the primary criterion, then, those with the shortest average delay (short RTT) will be selected until filling the vacancies.

After the Package Transfer has chosen the master nodes, the System Notifier informs the remaining nodes (called *slaves*) about this selection. Regardless of the ranking criteria used, the slaves register with the nearest master, which is identified by the shortest average delay (short RTT). However, the number of members is not fixed, and varies in each group. Once the groups have been formed, the NACR sends the NetApp to the master nodes that become ghost repositories. This means that the slave nodes can only download the NetApp from their selected masters. Again, the quota limits the number of parallel

downloads.

The pull-based and push-based mechanisms of Groups and Gossip behave in a similar way. The main difference is that, in the groups strategy, the slave nodes only receive the NetApp and will not become another ghost repository. In the hybrid mechanism, the NACR transfers the NetApp to the master nodes by push-based calls. After that, the masters receive requests from the slaves to send the NetApp package using pull-based calls. In this case, the NACR is behaving in an "*active*" way, whereas the *ghost repositories* are behaving in a "*passive*" way.

4.5 Summary and Discussion

In this chapter, we introduced the App2net ecosystem to empower PVN owners to deploy and manage NetApps in PVNs with heterogeneous EEs. Moreover, our ecosystem enables developers to distribute and describe NetApps of different technologies. For the sake of simplicity, we split up the ecosystem into two key platforms, `iMPROVE` and `app2net core`. As can be seen throughout this chapter, `iMPROVE` allows developers to store, publish, and distribute both NetApps and VNAs. We also proposed the Network Application Descriptor (NAD), an extension of the ETSI Network Service Descriptor, to describe NetApps and VNAs of different programmability-related technologies, as well as to represent the issue of conflicts. The second platform, called `app2net core`, enables PVN owners to trigger the deployment of NetApps in PVNs with heterogeneous EEs, even without knowing the specific features of the underlying infrastructure. Finally, we analyzed a set of code transfer techniques and investigated the main features for grouping them. Furthermore, we have designed models for code transfer which combined some of these techniques.

As seen in Chapter 3, we analyzed the main proposals for marketplaces in different contexts (*e.g.*, mobile market, cloud computing, and network). Following this, we compiled a list of design goals, divided into three main categories, for network marketplaces. We would now like to discuss how our ecosystem applied each of these design goals in our ecosystem. The first category (*i.e.*, *offer and distribution*) consists of four design goals, namely access control, publish, pricing, and notifications. With regard to **access control**, our ecosystem restricts access to different stakeholders through the *User Access Controller* module in the *User Manager* element. This module carries out different tasks related to access control: (*i*) authentication, in which it checks the credentials

of stakeholders before granting access to our ecosystem; and, (*ii*) message encryption, even though this action is not directly related to access control, it prevents messages (to and from our ecosystem), from being intercepted and understood by malicious users or systems. In this way, it prevents information theft and the sending of commands from unknown sources.

Inspired by mobile marketplaces (*i.e.*, Google Play Store, Apple App Store, and Windows Store), the App2net ecosystem also enables any third-party developers to **publish** their NetApps. In our ecosystem, we recommend no certification or other procedure to determine if candidates are qualified to become developers. In our view, any developer can create a "killer application" and, for this reason, all developers must be able to publish their applications in our ecosystem. However, stakeholders must create a pair of asymmetric keys to be able to carry out actions in the App2net, including to publish NetApps.

We do not propose elements or procedures for **pricing** NetApps, although it is discussed in Chapter 3 and included in our set of design goals. We believe that, in the context of networks, pricing models must be closely linked with monitoring tasks. Thus, a pricing model could be employed that includes the number of packets processed, created routes, or even the amount of connected clients. We also believe that disruptions or even service malfunction must reduce the price that is charged. In our view, this is a very complex research goal, which in itself is a subject for another thesis. For this reason, we do not provide mechanisms to support pricing in our ecosystem; however, we include a modular architecture and an extensible API that allows new components to be added, for accurate performance-based pricing models.

On the subject of **notifications**, the App2net ecosystem (*i.e.*, the User Notifier module) ensures that developers and PVN owners receive notifications about NetApp execution, runtime, and updates. Moreover, in our ecosystem (specifically, in the System Notifier component), modules or even NetApps can subscribe the events from network services (through the *subscribeEvents()* method). Thus, an event that takes place in a network service can trigger actions in another network service. For example, in a security service, an event that informs a high rate of access in a web server can trigger the initialization of the DPI service to mitigate a possible DDoS attack. Finally, any platform, network service, or module can send messages (*i.e.*, *sendMessage()* method) or events (*i.e.*, *notifyEvent* and *notifyUsers*) to the App2net ecosystem using the RESTful API.

We analyze the App2net ecosystem in terms of the *network environment* category. An essential feature of our ecosystem is the **deployment**. We design the `app2net core`

to be responsible for deploying NetApps and VNAs over different nodes (the NetApp Manager component has the means to install NetApps and instantiate VNAs). Moreover, our ecosystem allows the developers to specify the location flag, which represents the best placement for that particular NetApp. Thus, when PVN owners request the deployment of NetApps, the App2net deploys NetApps with regard to the location flag defined previously. A key feature of the App2net ecosystem is the support it gives to different technologies (*i.e.*, **infrastructure-agnostic**). The drivers that form the Node Communication Layer, enable App2net to deploy and manage network services over heterogeneous nodes from diverse technologies, such as Vyatta, Cisco ONE, POX controller, OPNFV, and P4 forwarding elements. However, the developers must program and adapt each distinct driver so that it can support the features of each device.

In App2net ecosystem, we think a basic support to the **monitoring and SLA Management**. We suggest a minimal set of procedures to cope with monitoring tasks. We describe the *Data Collector* component in detail and how it can gather different data about the NetApp execution from all the PVN nodes. Furthermore, we make the *output* tag available in the NAD file, which enables developers to reveal new data about the execution of their NetApps. Although we recommend a minimal support to collect data, we know that there are significant research opportunities to tackle both monitoring and SLA Management tasks. Moreover, as explained above, we believe that this goal together with pricing could integrate a new means of managing the NetApps execution (while also taking account of disruptions and malfunctions) and controlling the expenditure of the PVN owners.

We employ several procedures and elements to investigate the design goals of *applications*. On the subject of **consistency**, we design the *Repository Gatherer* element to deal with management and updates in the list of available NetApps and VNAs, stored in the NetApp Catalog. This element must periodically check if there has been any change in the External Third-party Repositories. When a developer requests a change in some NAD file, which lists the available NetApps and VNAs, this element updates the NetApp Catalog and notifies the PVN owners about these changes. Likewise, if a developer unpublishes a network service or if an External Third-party Repository loses connectivity, the *Repository Gatherer* shall remove all the affected NetApps and VNAs from the Catalog, even if only temporarily. As well as this, the App2net ecosystem takes measures to check the integrity of the NetApps packages and VNA images. To achieve this, the drivers must validate the package integrity through the SHA-256 hashes obtained from

the NetApp Catalog, before installing NetApps or instantiating VNA images. Moreover, the developers specify these hashes under *hash* tag in the NAD file.

We have paid special attention to **relationships** between network services; in the NAD file, the developers can describe both the dependencies and conflicts. When a PVN owner triggers the deployment of a network service or when a developer has already updated an installed NetApp, the *NetApp Selector* component takes steps to check whether there is any conflict between the necessary services as well as the NetApps dependents that must be installed to support the chosen network service. Finally, the App2net ecosystem provides support for several **management** tasks. This involves creating the tag *manage_action* through which developers to designate the necessary commands and parameters required for management tasks in NetApps. A PVN owner can trigger management actions by interacting with the App2net Web Portal or via RESTful API with *retrieveActions()* and *runAction()* methods. We believe that in this way, App2net can give support to the complete life cycle management, as well as, all the management actions listed by the developers in the NAD file.

Even though the research challenges (discussed in the Chapter 3) offer open opportunities for future studies, we have taken a step forward by addressing some of them in this thesis. For instance, in **descriptors**, our NAD enables developers to include data that were not previously envisaged, such as placement, management actions, output data, conflicts, and dependencies. However, it is clear that more in-depth research must be undertaken to include challenges such as auditing and affinity. In addition, we introduce a minimal set of actions and tags to cope with recommendation and placement. In the case of **recommendation**, we have compiled the *categories* tag, which together with the *NetApp Retriever* component can make a basic recommendation of NetApps based on the categories informed by the PVN owners. With regard to matters concerning **placement**, we propose the *location_flag* tag that allows the *Node Selector* module to choose which PVN nodes must install the network service. It must be remembered that the location flag can accept five different values, namely border, ingress, egress, custom, and all.

With regard to issues of **security**, we employ public and private keys to perform the asymmetric encryption of the messages and thus create a secure channel between the App2net ecosystem and PVN. In this way, we can prevent malicious users from interfering with the communication (*e.g.*, man-in-the-middle attacks to steal sensitive data) or sending false commands to carry out undesired actions. Of course, this is not enough and more robust elements, procedures, and tasks must be employed to ensure the security of

network marketplaces. On the question of **evolution-aware**, we encourage the creation of drivers to support new technologies or network paradigms. Although this solution encompasses underlying technologies to support new paradigms or even technologies, the new drivers must be implemented. Thus, the complexity of the Node Communication Layer is incremental and could reach such a high level, that it becomes impossible to add, modify, or delete drivers from this layer. For this reason, we argue in favor of more in-depth research that can find a transparent and sophisticated way to meet the challenge of evolution-aware.

5 EVALUATION AND ANALYSIS

In this chapter[1], our App2net ecosystem is evaluated in two distinct ways. In the Section 5.1, we conducted a case study to provide a detailed description of the interactions and the necessary actions for a third-party developer when publishing a network service in the App2net ecosystem. Following this, we explored the interactions of a PVN owner, who desires to purchase and deploy this network service in his/her PVN. As a result, the App2net ecosystem was shown to be a feasible platform which had enough simplicity to carry out the actions required by both developers and PVN owners transparently. We also discussed the management of this new network service via the App2net RESTful API. The main elements and interactions are shown in the Figure 5.1.

In the second evaluation, which is outlined in the Section 5.2, we implemented a prototype and that is tested on realistic network topologies commonly found in the Internet. The results allowed us to assess the impact of the App2net ecosystem on CPU usage, network overhead, and distribution time. Finally, in Section 5.3, we present some final remarks and a discussion about the results obtained.

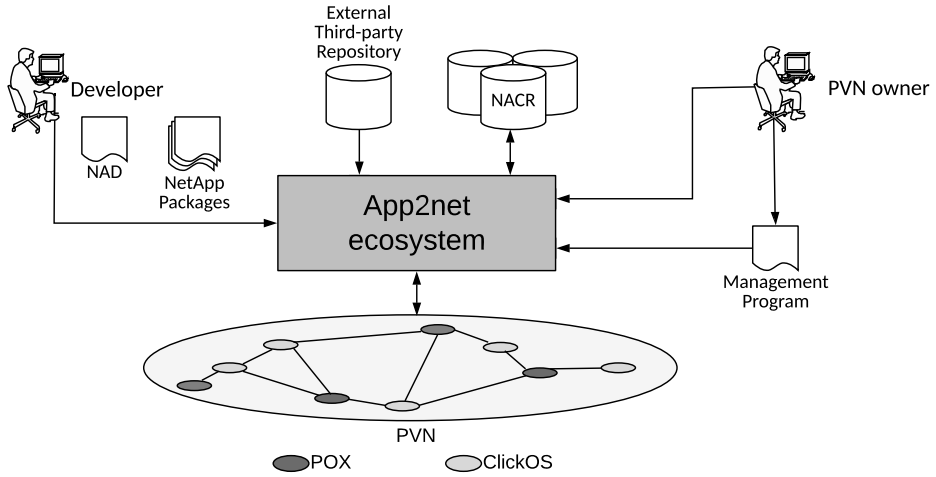
5.1 The Case Study

Let us suppose that an arbitrary third-party developer programs a new network service, which provides a *trust-based multicasting*. This network service supports two different technologies including NFV and OpenFlow. However, there are three different packages, each one implemented for a specific programmability technology: (*i*) one for general purpose NFV; (*ii*) one for NFV with ClickOS; and (*iii*) one for OpenFlow using a POX controller. As well as the packages, the developer must create a NAD file that contains all the information about the network service. Although there are three packages (one per technology), only one NAD file is required to describe all the packages that appear under the technology tag. Figures 5.2 and 5.3 show the core tags of the NAD file, especially those related to our proposed extension.

In the case of NFV, there are two packages. The first is a VNA image that contains the whole framework needed to support the network service (lines 9 to 24). This image

[1]This chapter is based on the following papers: “App2net: A platform to transfer and configure applications on programmable virtual networks” and “iMPROVE: Enhancing the Introduction of Services on Programmable Virtual Networks”. For more information about these papers see the Appendix.

Figure 5.1: Interactions between the App2net ecosystem and external elements.



Source: the author (2018).

is a virtual machine containing the NetApp package and the necessary EE. The developer does not send this image because it is available in an External Third-party Repository (Figure 5.1 and Figure 5.2 lines 14 to 16). In the NAD file, the developer makes a list of five distinct output data, namely: (*i*) "*not_forward_multicast_packets* which denotes the number of multicast packets that were not forwarded; (*ii*) "*multicast_traffic*" designates the number of active multicast flows and their sources; (*iii*) "*multicast_clients*" specifies the number of the multicast clients and their addresses; (*iv*) "*multicast_groups*" describes the data about multicast groups, such as sources, clients, and total data traffic; and, (*v*) "*trusted_nodes*" shows data about nodes marked as "*trusted*", which means, only those trusted nodes are able to join into multicast groups. It is worth noting that this output data can be obtained through App2net RESTful API via "*collectData*" and "*getData*" methods.

The second package for the NFV is a NetApp implemented for the ClickOS EE (lines 25 to 57), and the NACR will host it. As this package is a NetApp (line 29), an existing PVN node with support for NFV and ClickOS could install it. However, with regard to the location flag (line 37), all the compatible nodes in the PVN must install the NetApp that is responsible for processing all the data flows (*i.e.*, both ingress and egress traffic). In addition, the developer defines the initial settings (lines 31 to 33), which denotes that all the nodes must be regarded as "*trusted*". Furthermore, there are four different management actions (lines 38 to 48): (*i*) "*install*", which deploys the NetApp into the EE of the PVN node (line 39); (*ii*) "*start*", which initiates the execution of the NetApp according to the initial or saved settings (line 40); (*iii*) "*stop*", which shuts down the execution of the NetApp and saves the settings used (line 41); and (*iv*) the "*trusted-nodes*", which enables

a PVN owner to configure which nodes must become "*trusted*" for the network service. It should be noted that the information provided by the PVN owner ("*trusted-nodes*") will replace the expression `$value0$` in line 45.

In the case of POX (lines 58 to 111), the NetApp package will also be hosted in the NACR. Again, the developer defined the initial settings (lines 64 to 66). It is important to remember that the PVN owner can change these settings during the installation process. In addition to the management actions of the NFV NetApp (lines 38 to 48), the POX version has two additional actions: (*i*) the "*force-add-node*", which allows a "*trusted*" node to be added without any need for the verification process (lines 76 to 79); and (*ii*) the "*force-remove-node*" configuration allows a "*trusted*" node to be removed without the verification process (lines 80 to 83). The location flag (line 89) determines that the `app2net core` must install this NetApp in all the compatible nodes. Finally, there are two known conflicts in the POX version, which include all the versions of *multicast-ufrgs* (lines 91 to 94) and previous versions of the *dpi-telefonica* 4.0.2 (lines 95 to 98). In other words, this *trust-based multicasting* operates improperly when *multicast-ufrgs* or *dpi-telefonica* ($<$ 4.0.2) are used. Note that in XML notation, the "$<$" is represented by "<". Moreover, the expression `$value0$` (lines 74, 78, and 82) will also be replaced by the values attributed to each corresponding action.

When publishing the NetApp for downloading by the PVN owners, the developer must register it in `iMPROVE`. First of all, the developer must log into the App2net ecosystem via the Developer's Portal. After making the login successfully, the developer fills in the registration form and submits the two NetApp packages (one to OpenFlow using POX and one to NFV using ClickOS) and the NAD file. After this, the Portal sends these files to the NetApp Register, which reads the NAD file and stores the data in the NetApp Catalog. Then, the Register also creates a new record in the DB & Statistics with the information about the service obtained from the registration form and NAD file. In the next step, the Publisher component uploads the NetApp packages to the local NACR. By this time, the NetApp has been published and is available for a PVN owner to request its installation.

Assuming that the submitted packages update an existing network service, the Update Checker checks in the DB & Statistics to find out if any PVNs are using this network service. The Checker arranges these PVNs into two groups: (*i*) the PVNs that can update the network service automatically; and (*ii*) the PVNs that require human intervention. While the PVNs in the first group trigger the update processes for the network service, the PVNs in the second group wait for human actions. It should be pointed out that `iMPROVE`

Figure 5.2: Partial NAD file for *trust-based multicasting* (Part 1).

```
 1 <?xml version="1.0" encoding="UTF-8"?>
 2 ...
 3 <categories>
 4   <category>security</category>
 5   <category>multicasting</category>
 6   <category>communication</category>
 7 </categories>
 8 <packages>
 9   <technology>
10      <identifier>nfv</identifier>
11      <version>all</version>
12      <hash>3546705163bc40eef80ddf2cd11918aa</hash>
13      <type>vna</type>
14      <location>
15        <uri>http://myrepository.com/services/nfv/trust_multicast_45.img</uri>
16      </location>
17      <outputs>
18        <output>not_forward_multicast_packets</output>
19        <output>multicast_traffics</output>
20        <output>multicast_clients</output>
21        <output>multicast_groups</output>
22        <output>trusted_nodes</output>
23      </outputs>
24    </technology>
25    <technology>
26      <identifier>nfv</identifier>
27      <version>all</version>
28      <ee>clickos</ee>
29      <type>netapp</type>
30      <hash>e48f694fc62f4512317eda7e40ca2dac0447da10</hash>
31      <initial_config>
32        <command>export TRUSTED_NODES="all"</command>
33      </initial_config>
34      <location>
35        <uri>file:///repository/services/nfv/clickos/trust_multicast_45.click</uri>
36      </location>
37      <location_flag>all</location_flag>
38      <manage_action>
39        <install>click-install trust_multicast_45.click</install>
40        <start>click-install trust_multicast_45.click</start>
41        <stop>click-devirtualize trust_multicast_45.click</stop>
42        <configure>
43          <parameter>
44            <value0>trusted-nodes</value0>
45            <command>export TRUSTED_NODES="$value0$"</command>
46          </parameter>
47        </configure>
48      </manage_action>
49      <outputs>
50        <output>not_forward_multicast_packets</output>
51        <output>multicast_traffic</output>
52        <output>multicast_clients</output>
53        <output>multicast_groups</output>
54        <output>packet_loss_rate</output>
55        <output>trusted_nodes</output>
56      </outputs>
57    </technology>
```

Source: the author (2018).

sends notifications via the User Notifier (User Manager) to either inform the owners (first group) or request a decision from the owners (second group) about the update process. Before installing this update, NetApp Selector carries out a check of the dependencies (Dependency Verifier) and conflicts (Conflict Estimator). These processes are required to determine all the new dependencies and find out if any conflict is likely to occur between the installed network services and new NetApps packages.

If there is no problem, `iMPROVE` triggers the "*updateNetworkService*" method to ensure that the `app2net core` updates the NetApp. However, if there is any problem, `iMPROVE` requests the PVN owner to make an intervention. This intervention can in-

volve other actions, for example, the replacement of NetApps that are causing unresolved conflicts. After the PVN owner has addressed any problems found, `iMPROVE` can request the NetApps update ("*updateNetworkService*" method) for the `app2net core` on the basis of the owner's definitions. Even though the PVN owners may not have addressed the problems found or have decided not to make an update, the new NetApp is still available and can be installed in compatible PVNs.

By triggering the "*updateNetworkService*" method (available in App2net RESTful API), in PVNs that use the NetApp version of the *trust-based multicasting* network service, the `app2net core` retrieves the data about the installation or the last update from the DB & Statistics. Through this data, the Package Transfer module validates the source repository, which might be the NACR or an External Third-party Repository (lines 14 to 17, 34 to 36, and 86 to 88). Next, this module sends an "*update*" notification, via the System Notifier, to the PVN nodes that have been installed in the network service. Here, we must point out two aspects. First, the "*update*" notification contains the transfer guidelines, NACR, and new requirements. The second point is that there is no need to define compatible nodes because they were chosen in the last process. This means that the Transfer module only contacts the nodes that have installed the NetApp package. Following these actions, the Package installer recovers the *stop* and *update* actions from the NetApp Catalog and saves the current configuration of the *trust-based multicasting* network service. Next, the Installer stops the NetApp execution and takes the necessary actions to update the NetApp. Finally, the NetApp Configurator restores the saved configuration in all the compatible nodes, and the NetApp Manager initializes the network service execution.

Now, let us suppose the case of a PVN owner that wants to deploy the *trust-based multicasting* service in his/her PVN. First, the owner interacts with the PVN Owner's Portal to configure his/her PVN into the App2net ecosystem. During this process, the owner can send a VXDL file with the PVN data, such as supported technologies, nodes, interfaces, and IP addresses. In the `app2net core`, the Infrastructure Handler makes use of the VXDL file to install the corresponding drivers in each PVN node and get additional information such as supported programming languages, available EEs, and deployed network services. After ending the configuration process, the Infrastructure Handler saves all these data in the DB & Statistics (Figure 5.1 and Table 5.1 summarize some of them). Then, the owner uses the Marketplace, which is available in the App2net ecosystem, to select and trigger the deployment process of the NetApp that provides this

Figure 5.3: Partial NAD file for *trust-based multicasting* (Part 2).

```
 58    <technology>
 59      <identifier>openflow</identifier>
 60      <version>1.0</version>
 61      <ee>pox</ee>
 62      <type>netapp</type>
 63      <hash>f8af1633168833725ded1835ffdcec44</hash>
 64      <initial_config>
 65        <command>./trust_multicast_45.py --config={"trusted-nodes": "all"}</command>
 66      </initial_config>
 67      <manage_action>
 68        <install>./trust_multicast_45.py --install --file=${POX_HOME}/startup.py</install>
 69        <start>./trust_multicast_45.py --start</start>
 70        <stop>./trust_multicast_45.py --terminate</stop>
 71        <configure>
 72          <parameter>
 73            <value0>trusted-nodes</value0>
 74            <command>./trust_multicast_45.py --trusted-nodes=$value0$</command>
 75          </parameter>
 76          <parameter>
 77            <value0>force-add-node</value0>
 78            <command>./trust_multicast_45.py --add-node=$value0$</command>
 79          </parameter>
 80          <parameter>
 81            <value0>force-remove-node</value0>
 82            <command>./trust_multicast_45.py --remove-node=$value0$</command>
 83          </parameter>
 84        </configure>
 85      </manage_action>
 86      <location>
 87        <uri>file:///repository/services/openflow/pox/trust_multicast_45.py</uri>
 88      </location>
 89      <location_flag>all</location_flag>
 90      <conflicts>
 91        <conflict>
 92          <identifier>multicast-ufrgs</identifier>
 93          <version>all</version>
 94        </conflict>
 95        <conflict>
 96          <identifier>dpi-telefonica</identifier>
 97          <version>&lt; 4.0.2</version>
 98        </conflict>
 99      </conflicts>
100      <outputs>
101        <output>average_delay</output>
102        <output>not_forward_multicast_packets</output>
103        <output>multicast_traffics</output>
104        <output>multicast_clients</output>
105        <output>multicast_groups</output>
106        <output>packet_loss_rate</output>
107        <output>throughput</output>
108        <output>trusted_nodes</output>
109      </outputs>
110    </technology>
111 </packages>
112 ...
```

Source: the author (2018).

service. As can be observed in Figure 5.1, there are two technologies in the PVN, namely NFV using ClickOS EE and OpenFlow using POX controller EE. In addition, Table 5.1 displays the installed network services in the PVN. Most of the services are installed in a single technology, but, some of them (*e.g.*, *cache-squid*) are installed in both technologies.

In iMPROVE, the Catalog Explorer retrieves the PVN data registered in the previous steps from DB & Statistics. By interacting with the Marketplace, the owner is able to select two different categories: security and multicasting. He/She also supplies two keywords: trust-based and communication. Following this, Explorer uses the categories and keywords to filter the list of network services that are only OpenFlow technology-related and that use a POX controller or NFV technology-related that use ClickOS. In this

Table 5.1: Network services installed in the PVN.

Network Service	Version	Execution Environment
cache-squid	3.1	ClickOS, POX Controller
firewall-netfilter	10.0.1	ClickOS
ids-telofonica	1.9.8	ClickOS
nat-cisco	1.0	ClickOS
dpi-telefonica	4.0	POX Controller
load-balancer-ufrgs	0.98	POX Controller

way, the Marketplace only shows compatible NetApps (which use OpenFlow with POX or NFV with ClickOs) where the categories and keywords match the stored NAD files, and this enhances the search for the desired network service. Finally, the owner selects the network service *trust-based multicasting* and triggers the selection process.

In the selection process, the Version Resolver module retrieves from the Catalog the data about the last version of the NetApp and carries out checks of the dependencies (Dependency Verifier) and conflicts (Conflict Estimator). In this case, the Estimator finds a conflict with an installed NetApp (*dpi-telefonica* 4.0). Then, the process returns to the Version Selector to retrieve data about the previous version of the NetApp. As there is no previous version, User Manager notifies the PVN owner about the detected conflict. On the one hand, the PVN owner could remove the *dpi-telefonica* from his/her PVN, and thus restart the installation process. On the other hand, the PVN owner could update the *dpi-telefonica* network service, which also mitigates the conflict because the last version of the *dpi-telefonica* (4.3) is compatible with the *trust-based multicasting* (there is no known conflict with this version mapped in its NAD file).

The owner updates the *dpi-telefonica* by just triggerring the update process via the PVN Owner's Portal (or triggering "*updateNetworkService*" method in App2net RESTful API). In this way, the NetApp Selector will retrieve the last version of the *dpi-telefonica* (4.3). Then, the `app2net core` takes the necessary actions to update this NetApp, and the PVN owner can continue with the *trust-based multicasting* installation. Now that it is without conflicts, `iMPROVE` can interact with the `app2net core` to trigger the "*deployNetworkService*" method with the chosen NetApp and its dependencies.

When starting the deployment process, the Node Selector obtains the necessary data to define the compatible nodes. These data include the available EEs (ClickOS and POX controller), location flag (all), NACR, and PVN nodes. Since there are two available EEs in this PVN, the Node Selector splits them into two subprocesses, one for each technology. On receiving the necessary data, the Package Transfer module validates the NACR using a hash encrypted by asymmetric keys previously stored in the DB & Statistics. This module

generates the transfer guidelines (*e.g.*, strategy chosen, method, ranking criteria, NACR address, and URI of the package) and sends them to the NACR and compatible nodes via the System Notifier. Next, the PVN nodes that support NFV download the NetApp package (available in the NACR according to line 35) for ClickOS, whereas the nodes that support OpenFlow download the NetApp package for the POX controller (available in the NACR according to line 87).

In the next step, the Package Installer retrieves the available actions from the Catalog. Each node group recovers different actions. The nodes that support NFV get four actions (lines 38 to 48) and the nodes that support OpenFlow get six actions (lines 67 to 85). The Installer sends these "*install*" actions (line 39 for NFV and line 68 for OpenFlow) to drivers that execute them into compatible nodes. After deploying the network service, the NetApp Configurator obtains the initial functional settings to configure the *trust-based multicasting* in the nodes with ClickOS for NFV (line 32) and nodes with POX controller for OpenFlow (line 65). Although the initial functional settings are different commands (lines 32 and 65), they set up the same configuration in the network service. It is also worth noting that the PVN owner can customize these initial settings before the Configurator includes them in the nodes. Finally, even though just one network service has been selected, `app2net core` installs two NetApp packages (one in ClickOS EEs and one in POX controllers EEs).

After the deployment, the PVN owner can use the open methods of the App2net RESTful API to manage the NetApp. Thus, the PVN owner could write a management program, and in this way adapt the NetApp execution to the nuances of his/her environment. For instance, owing to performance and security constraints, the *trust-based multicasting* must be executed from 8:00 AM to 6:00 PM, and, at every hour, the PVN owner must change the nodes that can connect to the multicast groups (called trusted nodes). This means that, during the time interval, the owner must access the PVN Owner Portal to include the new trusted nodes at each hour, without a management program.

When implementing a management program, the owner first obtains all the available actions in each NetApp package from the App2net RESTful API (through the "*retrieveActions*" method). Since two different technologies form the PVN, by deploying the *trust-based multicasting*, two NetApp packages were installed (one in ClickOS EEs and another in POX controllers EEs). Although two packages implement the same network service, each can make different management actions available. As can be seen in Figures 5.2 and 5.3, in the NFV ClickOS version, there are four available actions (lines 38 to 48)

and six output data (lines 49 to 56). Similarly, in the POX version, there are six available actions (lines 67 to 85) and eight output data (lines 100 to 109).

When automating the necessary actions, the PVN owner runs a management program that triggers the necessary actions via the App2net RESTful API. Thus, when a computer executes the management program, at 08:00 AM, it triggers a "*start*" action through the "*runAction*" method in PVN nodes that have installed the *trust-based multicasting* network service. Next, at every hour, the program obtains data about which nodes are "*trusted*" (via the "*getData*" method and "*trusted-nodes*" output); then, it checks whether or not the trusted nodes must be changed. In an affirmative case, the program also triggers the "*configure*' action to change them. In this way, nodes with ClickOS for NFV, run the command given line 45, whereas nodes with POX controller for OpenFlow run the command detailed in line 74.

In the case of automatic actions, the management program is responsible for sending the required parameters. However, another platform or script can provide these parameters too. At the end of the day (at 6:00 PM), as a result of the constraints defined by the PVN owner, the management program triggers the "*stop*' action to turn off the *trust-based multicasting* execution. The owner or a program trigger a configure or start action only once even for two different technologies; thus, the `app2net core` must run different commands in the nodes depending on its supported technology (POX or ClickOS). However, the management action and parameters must be the same to do this, such as when configuring "*trusted-nodes*" in nodes that support NFV ClickOS (lines 43 to 46) and in nodes that support the POX controller (lines 72 to 75), as well as for starting the NetApp execution in nodes that support NFV ClickOS (line 40) and in nodes that support POX controller (line 69).

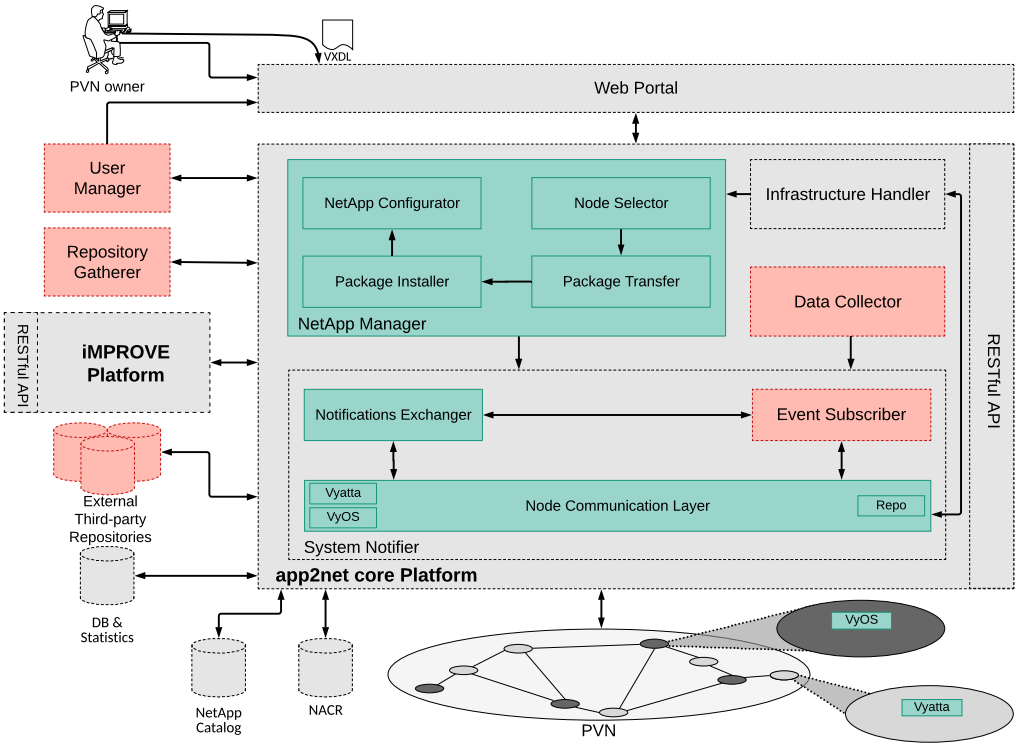
5.2 App2net Ecosystem Prototype and Experiments

In this section, we give a detailed account of the App2net ecosystem prototype[2] and the features of the test environment. Following this, we examine the feasibility of the prototype and discuss the results obtained for code transfer and deployment by means of the App2net ecosystem. In Figure 5.4, we detail the components and modules of our prototype as follows: (*i*) red boxes represent the elements did not implement; (*ii*) green boxes represent components and modules that are fully implemented; and, (*iii*) green

[2]The current version of our prototype can be download from <https://github.com/rlsantos/app2net>

dashed boxes denote components and modules partially implemented, that means, just essential features are available.

Figure 5.4: Representation of the implemented modules in App2net ecosystem prototype.



Source: the author (2018).

The App2net ecosystem prototype was implemented by means of the Django 1.4.3[3], Python 2.7.3[4], and PostgreSQL 9.1.6[5]. This prototype was installed on a server with four AMD Opteron 6276 processors (16 cores per processor) 2.30 GHz and 64 GB of RAM, running an Ubuntu Server 12.04.3. Besides, the python-igraph[6] library 0.6.5 was employed for calculating the quota. The experiments were conducted on a PVN consisting of by 51 VMs that were hosted on the server described above. These VMs were created with two 2.3 GHz cores, 1 GB of RAM, and 1 Gbps vNIC, using two different EEs: Vyatta Core 6.6 R1 and VyOS 1.0.4. Furthermore, the QEMU-KVM 1.0 hypervisor and Open vSwitch 1.10.2 were required to host and connect these VMs. The Aurora (WICKBOLDT et al., 2014) was used as the Generic Virtual Platform Manager, which means, it was designed to create and manage the PVN infrastructure. This software was installed in another server with an Intel Core 2 Duo 3.00 GHz and 2 GB of RAM, running Ubuntu 13.04.

[3]<http://www.djangoproject.com/>
[4]<http://www.python.org/>
[5]<http://www.postgresql.org/>
[6]<http://igraph.org/python/>

5.2.1 Methodology and the Test Environment

A well-accepted classification of Internet topologies (LUIZELLI et al., 2016), designed by Kamiyama *et al.* (KAMIYAMA et al., 2010), was adopted to ensure a high degree of fidelity with regard to real-world networks. Kamiyama *et al.* (KAMIYAMA et al., 2010) conducted a study that formalized the classification of ISP networks into the three topology classes, namely: *Ladder*, *Star*, and *Hub & Spoke*. In their study, the authors analyzed 23 commercial backbone networks (online and available at the CAIDA website[7]) with the number of nodes ranging from 21 to 128. Through this analysis, the authors defined a set of metrics that reflect the main topological properties present in each backbone network. These metrics include, for instance, the degree of connectivity in the network and the presence of hub nodes, which means, nodes with high connectivity and concentrated. Thus, the authors map the relationship between these metrics and the type of network topology of the backbone, so that these topologies can be categorized into one of the previously described classes.

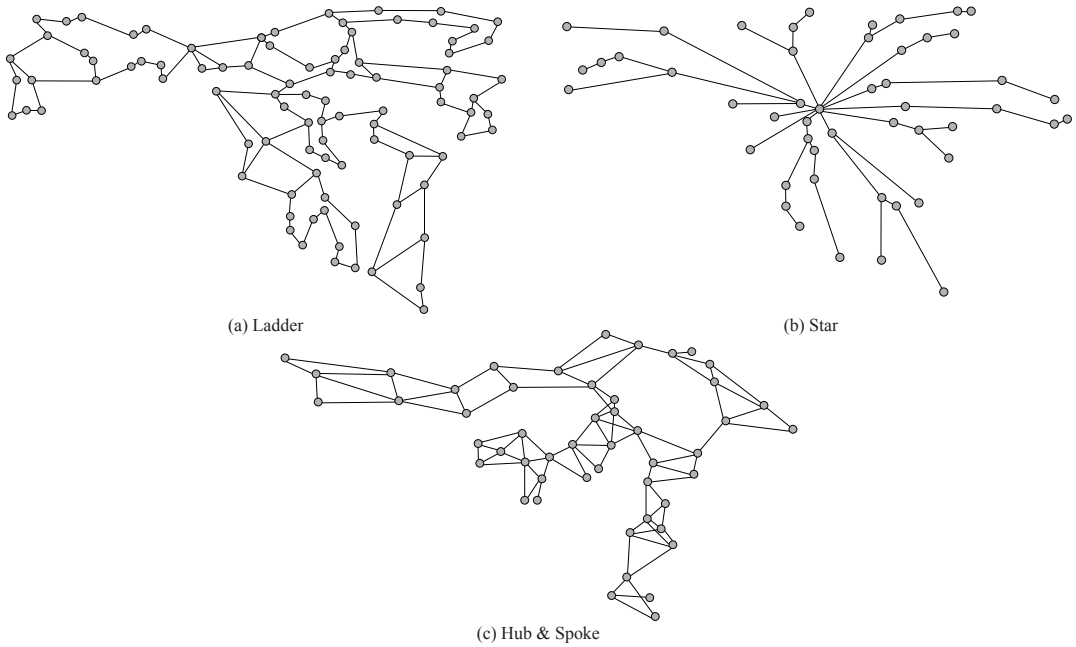
Figure 5.5 provides a simple view of three representative Internet topology classes. The network topologies arranged as *Ladder* (Figure 5.5(a)) are characterized by their lack of hubs, as well as being formed by a set of loops. As a result, the topology tends to have lower connectivity, the distance between the nodes (with regard to the number of hops) is generally high. *Star* topology class has a small number of hubs connected to numerous nodes with low connectivity (as can be seen in Figure 5.5(b)). In this class, the distance between the nodes tends to be low, while traffic tends to concentrate on the hubs. Finally, the *Hub & Spoke* class (Figure 5.5(c)) is characterized by a comparatively higher number of hubs, which are usually interconnected. A further point is that a large number of nodes are connected to one or more hubs.

In this thesis, our virtual nodes simulate programmable routers in a PVN and we have arranged them in accordance with the Internet topology classes outlined above. In this way, we were abre to produce samples of topologies using the IGen tool[8]. Each PVN sample has 50 EEs and one NACR, which only stores one NetApp but two different versions of the package (one for Vyatta and another for VyOS). We divided the VMs into three groups that simulate their geographical distribution. Thus, the connections between the groups were configured with a limited bandwidth rate of 100 Mbps and a delay of

[7] See the commercial backbone networks on the Center for Applied Internet Data Analysis (CAIDA) website: <http://www.caida.org/tools/visualization/mapnet/Data/>

[8] <http://igen.sourceforge.net/>

Figure 5.5: Three representative Internet topology classes.



Source: the author (2018).

16 ms, 24 ms, and 50 ms with a variation of 5 ms (as observed in the tests conducted at the main network sites in Brazil). We limited: (*i*) the bandwidth rate by using the configurations available in OpenVSwitch; and, (*ii*) the delay through the traffic control commands available in the kernel.

To ensure size of NetApps was realistic, we analyzed a sample of applications implemented with different network programmability-related technologies. As a result of this analysis, we concluded that the size of most of the NetApps ranges between 100 KB and 1 MB. However, some network services and firmware have sizes up to 12 MB. Hence, we included the following NetApps sizes: 100 KB, 500 KB, 1 MB, 5 MB, and 10 MB.

In our experiments, we analyzed three main evaluation metrics: distribution time, CPU usage, and network overhead. The reason for making these metrics was that they could enable us to evaluate the impact introduced by the App2net ecosystem on the deployment of network services. First, we employed as evaluation metric of the time that had elapsed until all the EEs obtained the NetApp package (regardless of what technology the nodes support), which we call the NetApp "*distribution time*". Thus, we were able to check the speed of the App2net ecosystem for transferring packages to PVN nodes. In this way, we could determine if the App2net would be more efficient than automating scripts or even a PVN owner executing these transfers manually.

We synchronized all the nodes through the same Network Time Protocol (NTP) server

to capture the distribution time. Next, we obtained the time at the start of the distribution process by employing the time module available in the Python[9]. After every node has completed the transfer, it sends a message to the App2net ecosystem prototype with the end time. Finally, at the last node, the App2net calculates the total distribution time by subtracting the time captured by the last node and the time captured at the beginning of the process.

The CPU usage allows us to check how much processing our solution requires from both repositories and PVN nodes. In some cases, the PVN owners are more interested in a minimal processing interference than a minimal distribution time because the PVNs require a high CPU usage from the nodes (*e.g.*, the video transcoding network service). We used *pidstat* command from the *sysstat* package[10] in both the NACR and PVN nodes to obtain the CPU usage rate. This command is a means of capturing the average CPU usage of a specific process in each VM at every second. Our analysis is based on the calculated average CPU usage from all the PVN nodes, which were captured using the *pidstat* in each node, and the average CPU usage from the NACR also captured by *pidstat*.

The network overhead allows us to measure how much traffic our ecosystem needs to add to the network to perform its essential functions. When estimating the overhead, first, we log the data traffic of all the interfaces of each node using the Wireshark[11]. In our prototype, the App2net ecosystem sends all the necessary messages through specific ports. These messages include lists, actions, statistics, requests, system calls to trigger actions, configurations, and others. By using the filters available in the Wireshark, we are able to isolate the App2net traffic from other types of traffics. Then, we can estimate precisely the network overhead that is introduced by our ecosystem.

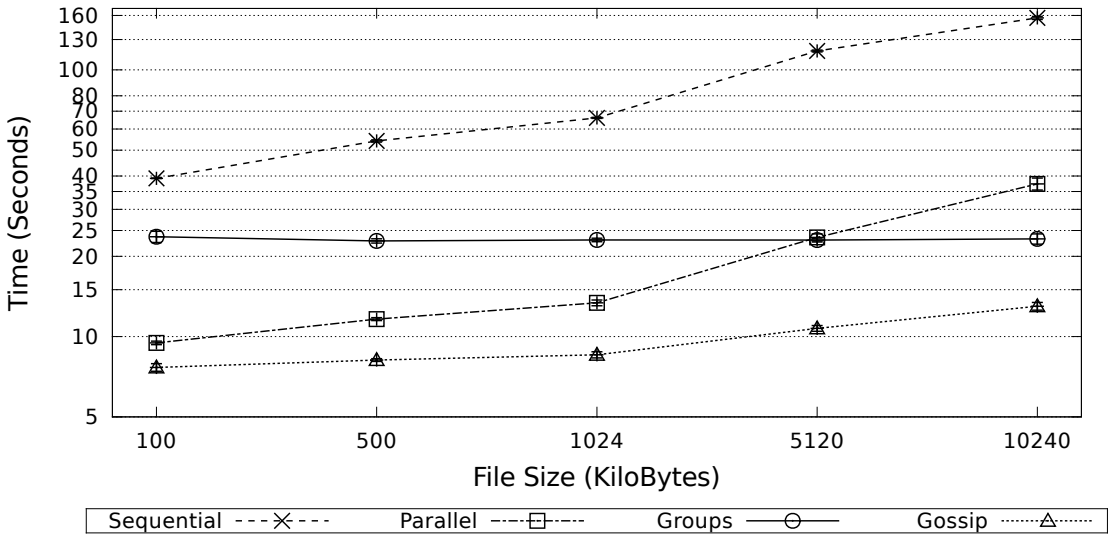
We created different setups for each experiment to evaluate the performance of our ecosystem more effectively. In every setup, we changed the topology, NetApp size, and model for code transfer. In the following experiments, we executed the setups 30 times. Hence, the values displayed in the following graphs are the averages calculated for all the executions (JAIN, 2015). Furthermore, we also showed the confidence interval (error bars) by calculating a confidence level of 99%, even though the confidence interval can be minimal in most cases.

[9]For more information, see Python documentation: <https://docs.python.org/2/library/time.html>
[10]For more information, see Sysstat repository: <https://github.com/sysstat/sysstat>
[11]<https://www.wireshark.org/>

5.2.2 Analysis of the Experiments

Figures 5.6, 5.7, and 5.8 provide an overview of the results obtained by the models which employed the hybrid mechanism and delay criterion in Ladder, Hub & Spoke, and Star topologies, respectively. We chose to examine these models because they highlight some of the differences between the strategies. Thus, the results obtained from other models were not revealed for the sake of simplicity.

Figure 5.6: Strategies in the Ladder topology (hybrid-delay).

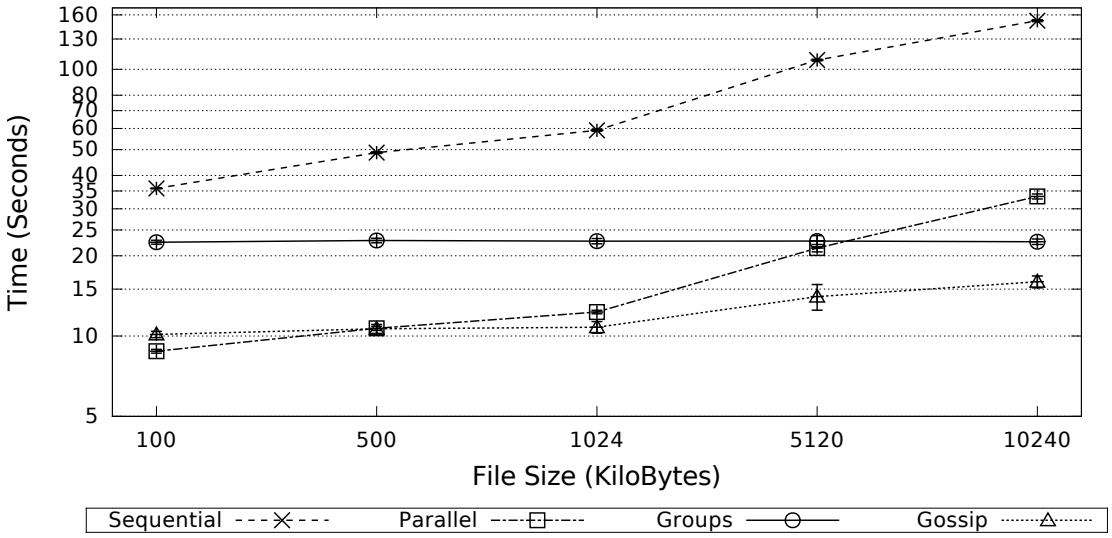


Source: the author (2018).

Figure 5.6 shows the results obtained for models using the hybrid mechanism and delay criterion in Ladder. These results reveal that Parallel has a good distribution time for small NetApps (100 KB), but, that the time increases for large NetApps (5 MB and 10 MB). However, Gossip obtains the shortest times for almost all the file sizes. With regard to the small NetApps, the reason for the different results obtained by Gossip and Parallel is that the central NACR has enough processing capacity and carried out its transfers so fast that the EEs had no time to act as repositories. However, as the package size increases, Gossip offers additional *ghost repositories* that reduce delay and prevent the central NACR from overloading. Despite of this, Groups obtains similar times in all the package sizes, mainly because each master is only able to send the packages in parallel for the nearest nodes, which prevents the congestion of links and the occurrence of high delays.

As can be seen in Figure 5.7, the results obtained in the Hub & Spoke reveal a very similar behavior to that of Ladder. However, in Hub & Spoke, two interesting cases

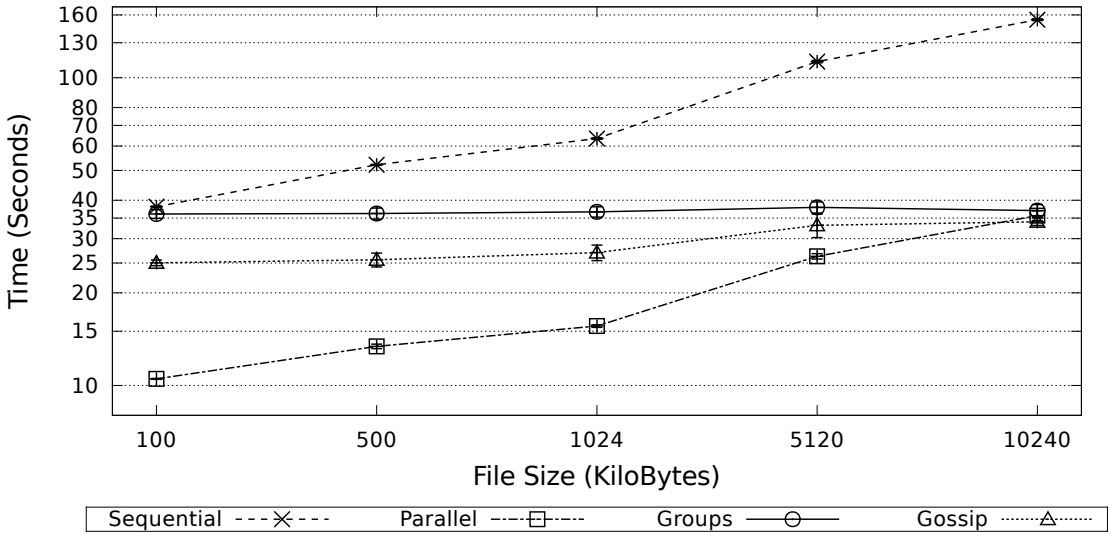
Figure 5.7: Strategies in the Hub & Spoke topology (hybrid-delay).



Source: the author (2018).

should be highlighted that are caused by the large number of links in the nodes. First of all, the distribution time for Gossip and Groups is slightly higher than in Ladder. By using the delay criterion, the ghost repositories must calculate the delay for more nodes by increasing the distribution time. Second, the distribution time for Sequential and Parallel strategies is somewhat better. Now, the NACR can send the NetApp package to multiple EEs through several links, and thus avoid the congestion of links among the groups of PVN nodes and reduce the hop distance between the NACR and EEs.

Figure 5.8: Strategies in the Star topology (hybrid-delay).

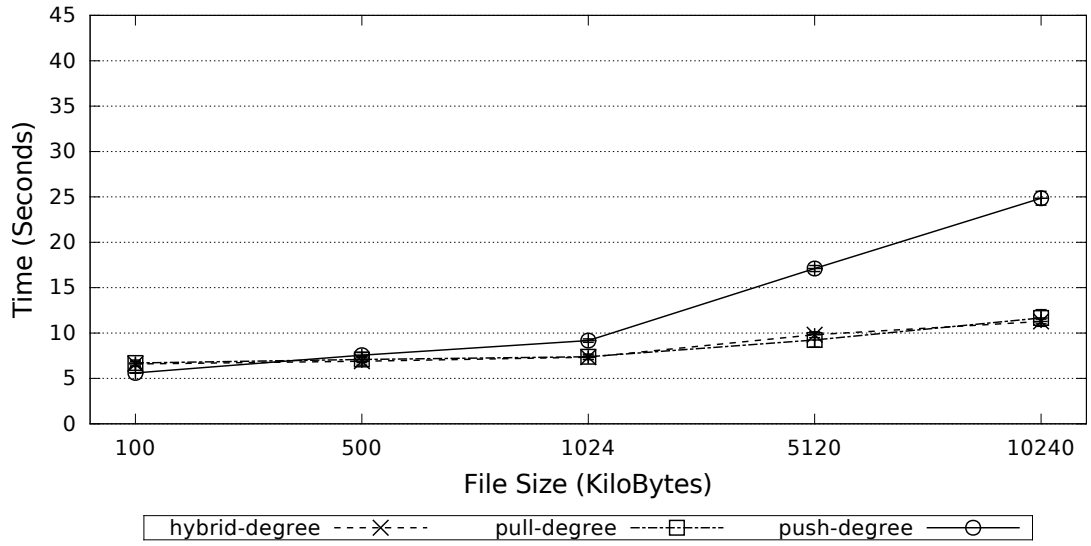


Source: the author (2018).

Figure 5.8 shows the results obtained by models based on the hybrid mechanism and delay criterion in Star. These results reveal that NetApp distribution times in Star are

longer than (or equal to) those obtained for Hub & Spoke and Ladder, which corroborates the view that the PVN topology influences the NetApp distribution process. In Star, the low number of links and the centralized feature, in which only one node interconnects with several branches, cause congestion and high delay, and thus results in a poor performance. From an analysis of the results, it is clear that the Gossip strategy is less advantageous than the others. As there is only one path to reach most of the EEs, Gossip cannot avoid delays and congestion. Similarly, questions regarding topology are also an obstacle to other strategies, and this might explain the similarities between Parallel, Groups, and Gossip for large NetApps (10 MB).

Figure 5.9: The Gossip strategy in the Hub & Spoke topology (degree).

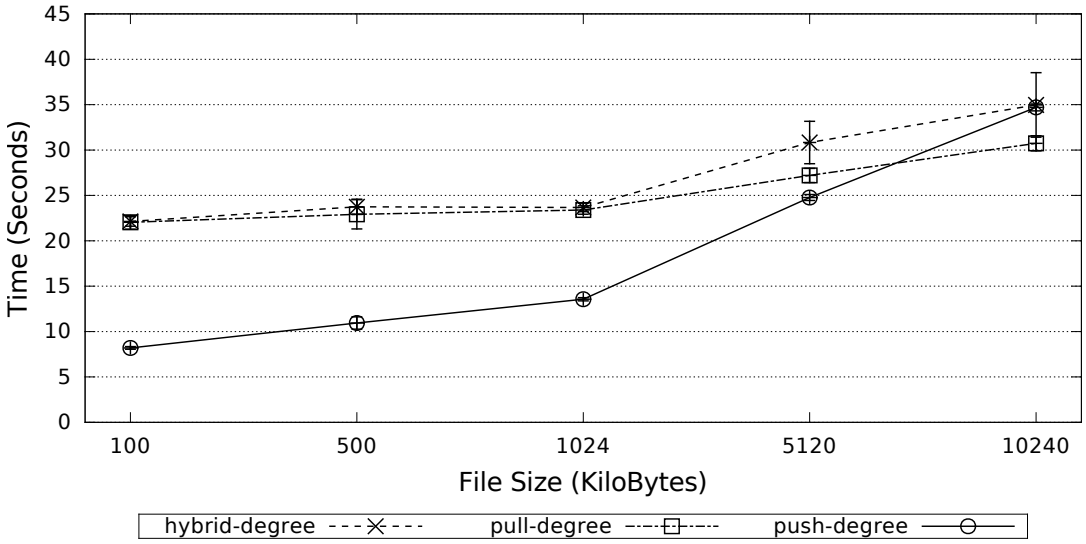


Source: the author (2018).

Figure 5.9 shows the performance results for Gossip in Hub & Spoke when following the degree criterion, and this reveals that all the mechanisms attain similar distribution times for small files. As the file size becomes larger than 1 MB, the *push-based* mechanism deteriorates; at the same time, the *pull-based* and *hybrid* mechanisms show a moderate increase in time. We have not disclosed the results in the Ladder topology because the transfer models showed identical results to those of Hub & Spoke. The degree is the only criterion that has been displayed because it achieves the best results for this strategy.

Figure 5.10 shows that the *push-based* mechanism in Star performed quite well for small files but poorly for larger ones. The reason for this is that the nodes actively transfer the file to their neighbors. In contrast, the *pull-based* and *hybrid* mechanisms started with a longer time to transfer the small files but showed a good performance for the larger ones. This was because the nodes had to act as an extra repository and communicate changes in

Figure 5.10: The Gossip strategy in the Star topology (degree).



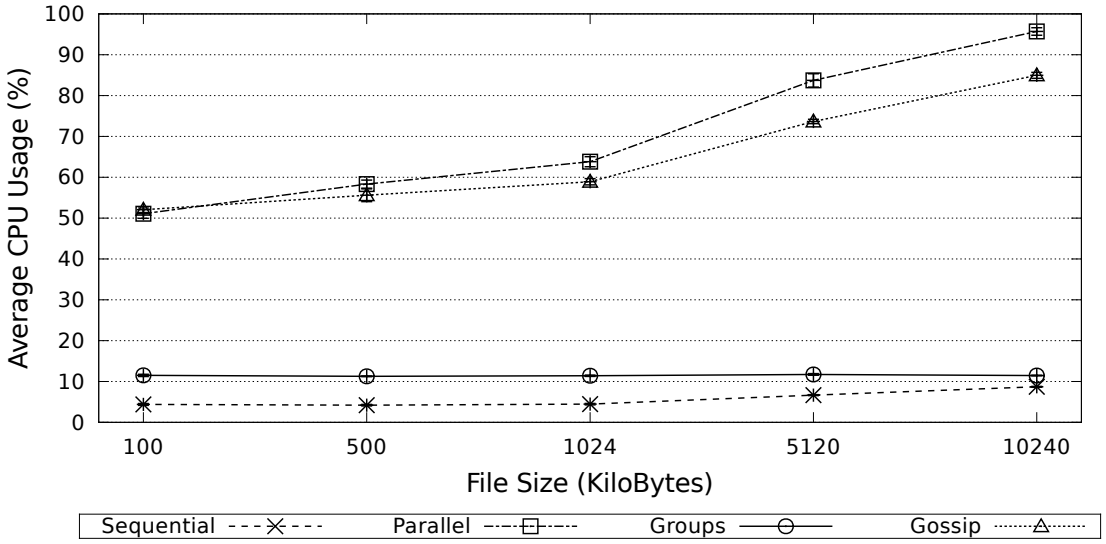
Source: the author (2018).

the standby list, as well as carrying out their regular routing tasks. Moreover, the repositories have to wait for their neighbors to request the file which leads to intervals before the starting-point of each transfer. In addition, the *hybrid* mechanism is also affected by the problem of over-processing because it uses *push* in the first stage of the transfer, as can be observed in the time it takes to make a transfer of the large files (5 MB and 10 MB).

We found the CPU usage of VMs was dissimilar while carrying out the experiments with different models. For this reason, we ran the experiments again, but this time including an analysis of the CPU usage. Here (in Figures 5.11 and 5.12), we took into account the *push-delay* (in the NACR) and *pull-links* (in the PVN nodes) models. These models were chosen because they show the higher values for the average CPU usage. In the NACR, Parallel and Gossip (Figure 5.11) show a significant and incremental average CPU usage. This results from the computing that is required by the NACR to control and perform all the parallel transfers for the PVN nodes. By contrast, Groups and Sequential have a small and steady average CPU usage rate for all the experiments. Sequential has the lowest values for the average CPU usage rate because in this strategy there is only one transfer at a time. Thus, the distribution time to all the PVN nodes received the NetApp is very high and the computing required is minimal. Finally, in Groups, the CPU usage is smoother because the NACR only transfers the NetApp to the masters.

First of all, it is important to highlight the need to take account of the average CPU usage rate in PVN nodes. Thus, the results shown are the average CPU usage in 50

Figure 5.11: Average CPU usage rate of the NACR (push-delay).

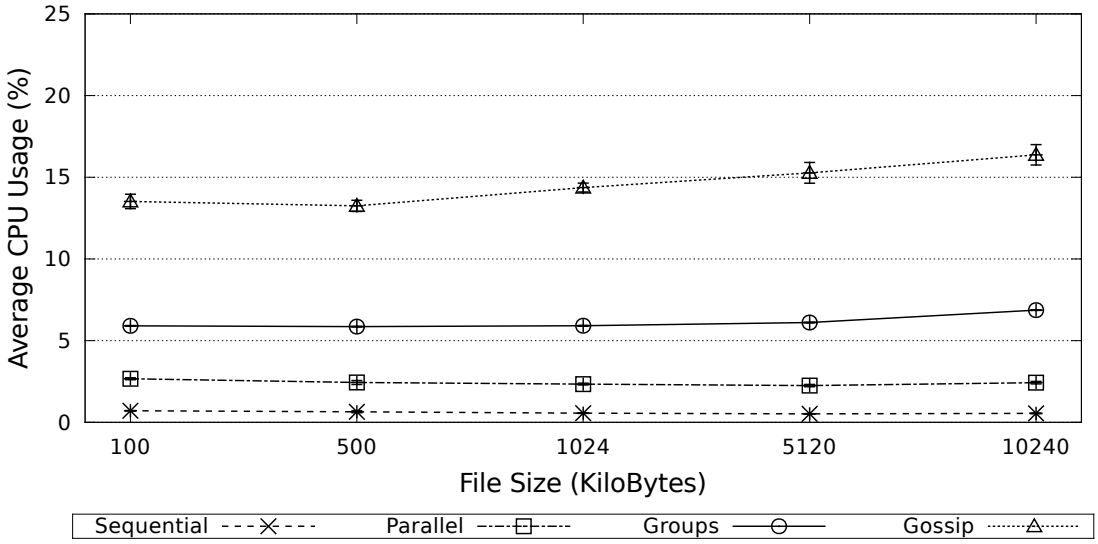


Source: the author (2018).

PVN nodes of each topology (Figure 5.12); in this way, although some nodes have a high CPU usage in different models, these rates can be softened in the average shown. In the PVN nodes (Figure 5.12), it is clear that Gossip obtains a higher CPU usage rate than the other strategies because the PVN nodes become ghost repositories after receiving the NetApp; this means, they must transfer the NetApp package to other PVN nodes. In the experiments with Gossip, we found that after becoming ghost repositories, there was a considerable increase in the PVN nodes with the CPU usage. However, as previously explained, these high values observed in the ghost repositories (few nodes) are smoothed out in the average shown by the low processing of the other PVN nodes. Likewise, in Groups, the small CPU usage of slaves conceals the high processing of the masters and reduces the average CPU usage. In another way, in Sequential and Parallel, each PVN node performs its transfer and only interacts with the NACR, which results in a small average CPU usage.

By analyzing the average CPU usage of all the nodes (Figures 5.11 and 5.12), we can determine that the Gossip models require a high processing from both the PVN nodes (ghost repositories) ($\approx$ 15%) and NACR (from $\approx$ 50% for small files to $\approx$ 85% for large files). As the number of ghost repositories increases over time, the number of PVN nodes with high processing also increases. Even though the Parallel strategy requires minimal processing in PVN nodes (below 5%), it requires a huge CPU usage from the NACR (from $\approx$ 50% for small files to $\approx$ 95% for large files). Finally, Groups strategy shows the best performance with regard to the average CPU usage. The reason for this is that the

Figure 5.12: Average CPU usage rate of PVN nodes (pull-links).



Source: the author (2018).

processing required is low in both the NACR ($\approx$ 10%) and PVN nodes (from $\approx$ 5% to $\approx$ 7%). It is noteworthy that the master nodes require high processing, but, in the overall average CPU usage, the large number of slaves with negligible processing spreads out these values.

One significant difference between the strategies analyzed, is the size and number of messages exchanged to control the behavior of the PVN nodes and to handle all the lists. For example, in Sequential, there are only a few messages exchanged between the nodes, such as requests for downloads and information about the criterion that is used. Otherwise, in Gossip, there is a significant number of messages, such as information about current ghost repositories, busy notifications, shared list, deletion requests, and quota information. In view of this, we also measured the communication overhead for each strategy by 50 PVN nodes, one NACR, and the NetApp package of 100 KB. The communication overhead is set out in detail in Table 5.2.

Table 5.2: Total network traffic and overhead generated by each strategy.

Strategies	**Total Network Traffic**	**Reduction in the Total Network Traffic**	**Percentage of Reduction**	**Overhead**	**Percentage of Overhead**
Sequential	54,629.62 KB	0.00 KB	0.00%	1,567.87 KB	2.87%
Parallel	53,619.85 KB	1,009.77 KB	1.85%	1,458.46 KB	2.72%
Groups	42,236.71 KB	12,392.91 KB	22.69%	1,875.31 KB	4.44%
Gossip	41,139.05 KB	13,490.57 KB	24.69%	2,085.75 KB	5.07%

As expected, the overhead remained almost the same for most of the analyzed models, and we only observed a small variation if the used mechanisms were changed. With regard to the strategies, Gossip registered the highest observed overhead, around 2,085.75 KB ($\approx$ 5.07%) of overhead in the total traffic generated. The Groups obtained 1,875.31 KB ($\approx$ 4.44%) of overhead, whereas Sequential and Parallel achieved 1,567.87 KB ($\approx$ 2.87%) and 1,458.46 KB ($\approx$ 2.72%) of overhead, respectively. Even though Groups and Gossip require a large number of messages to control the behavior of the nodes and to handle lists, these results demonstrate that the overhead is minimal, in light of the benefits provided by these two strategies, for example, a better distribution time than that achieved by Sequential and Parallel.

By analyzing the total traffic generated, we observed that Gossip and Groups had obtained a higher overhead. However, these strategies also showed a considerable reduction in the total amount of network traffic generated. If the Sequential strategy is taken as a baseline, in Gossip, the reduction observed was of 13,490.57 KB ($\approx$ 24.69%) and, in Groups, the reduction was 12,392.91 KB ($\approx$ 22.69%). The reason for this reduction is that both strategies created the new repositories (*i.e.*, ghost repositories and masters) along with the NetApp distribution process. Finally, Parallel obtained a tiny reduction in the total network traffic generated because there are no extra repositories, which means that the same NACR performs all the transfers. In terms of numbers, this reduction was 1,009.77 KB ($\approx$ 1.85%) in comparison with the Sequential strategy.

5.3 Summary and Discussion

We began this chapter by conducting a case study to provide a detailed description of the necessary actions needed for a third-party developer to publish and distribute an innovative network service. We have shown that a single NAD file can describe packages for three different technologies, including the initial settings, management actions, output data, and known conflicts. In addition, we showed the simple steps needed for a user to choose, deploy, and manage a new network service in the PVN. In the deployment phase, the App2net ecosystem was shown to be feasible to deploy a single network service with several packages in a PVN supporting two different technologies, namely ClickOS for NFV and POX for OpenFlow. After deploying the network service, we described the procedures for a PVN owner to obtain all the available actions in each NetApp through the "*retrieveActions*" from App2net RESTful API. Next, we explained how he/she could use

these actions to create a management program, which could analyze the execution data (via the "*getData*" method) and trigger the necessary actions (by means of the "*runAction*" method). In this way, the App2net ecosystem proved to be a viable alternative for distributing, describing, deploying, and managing network services in PVNs that support different programmability technologies.

We also implemented and tested an App2net ecosystem prototype and this took into account the code transfer models that we have designed. The results obtained corroborate the feasibility of a PVN owner using the App2net to transfer, install, and configure new NetApps without knowing the specific features of the underlying environment. We evaluated the App2net ecosystem in the light of two different EEs (Vyatta and VyOS). The evaluation results clearly reveal the influence of the PVN owner's network topology on the distribution times of NetApps for PVNs with heterogeneous EEs. We found that the behavior of the strategies was similar in both the Hub & Spoke and Ladder topologies. In these topologies, Gossip using the hybrid and pull mechanisms obtained the best distribution times in most of the tests. With the degree (number of links) as the criterion, Hub & Spoke achieved good distribution times, whereas, in the case of Ladder, the available bandwidth performed better. In Star, the best performance was obtained with the Parallel strategy and push mechanism using the degree and the available bandwidth criteria.

Finally, we also evaluated the average CPU usage and network overhead generated by each strategy. Although Gossip obtains good distribution times, it requires a significant CPU usage of both the NACR and the PVN nodes ($\approx$ 85% and $\approx$ 16.5% respectively for NetApps with a size of 10 MB). On the one hand, Gossip and Groups strategies show a high network overhead (5.07% and 4.44% respectively). On the other hand, these strategies reduce the total network traffic (24.69% for Gossip and 22.69% for Groups). Parallel and Sequential strategies require low CPU usage from PVN nodes ($\approx$ 3% and $\approx$ 1% for NetApps with a size of 10 MB). However, although Parallel achieved the best distribution time for Star, it still incurs a high CPU usage at the NACR ($\approx$ 96% NetApps with a size of 10 MB). In summary, there is no single best strategy for a good distribution time, minimal overhead, and minimal CPU usage of both NACR and PVN nodes. It is up to the PVN owner to choose the best solution and trade-offs for each environment.

6 CONCLUDING REMARKS

This thesis examined the state-of-the-art of network programmability and virtualization, and discussed the main drawbacks of deploying and managing NetApps over PVNs with different EEs. We compiled a glossary to bring together terms related to the same concepts used in different technologies. Following this, we reviewed the historical background of marketplaces with regard to networking paradigms. On the basis of this review, we discussed the essential design goals for a reference network marketplace, and then, we highlighted the research challenges that must be overcome before that these marketplaces can be established in future networks.

This thesis also introduced the App2net ecosystem to empower PVN owners, who do not know the specific features of the underlying infrastructure, to deploy and manage NetApps in PVNs formed by different EEs. We sought inspiration from the mobile market that through the use of online marketplaces enables PVN owners without any knowledge of devices to deploy apps on their smartphones too. This meant we were in a position to analyze our ecosystem in terms of the design goals previously found in the review of online marketplaces. When evaluating our ecosystem, we conducted a case study that traced the steps in which a third-party developer can describe and publish an innovative network service in the App2net ecosystem. Following this, we described the main steps needed for a PVN owner to deploy and manage this network service in a PVN with heterogenous EEs. In addition to this case study, a prototype was implemented, and its feasibility was evaluated and discussed by taking account of the distribution time, CPU usage, and network overhead.

The remainder of this chapter is structured as follows. First, the research questions are answered, and then the hypothesis is supported by an outline of the work carried out in this thesis. After this, we summarize the main contributions made by this thesis. Finally, we make some predictions about further research studies needed in this area.

6.1 Answering the Research Questions and Confirming the Hypothesis

As discussed throughout this thesis, a few years ago, the market for mobile apps was closed and limited to a few large companies that developed and deployed their apps on specific smartphones. In this way, the smartphones were “black boxes” because no one could install or change the apps that had already been deployed. However, Apple and

Google opened up the market for third-party apps, by introducing, in 2007 and 2008, respectively, online marketplaces (*i.e.*, Apple App Store and Google Play) to provide new apps and features for the smartphones end-users. Similarly, the networking market was closed and limited to a few large companies. However, recent advances in PVNs have opened it up it third-party developers too. In light of this, the investigation carried out during this thesis has been aimed at corroborating the following hypothesis.

***Hypothesis*: a platform inspired in online marketplaces may enable PVN owners with a limited knowledge of the underlying infrastructure to overcome the main hurdles to introducing new NetApps into PVNs**

At the beginning of this thesis, three research questions were defined to guide the investigation about this hypothesis and to provide evidence about its validation. Thus, on the basis of this investigation, it is possible to answer these research questions as follows.

Research Question #1: *What are currently the main hurdles to introducing new network applications in Programmable Virtual Networks?*

Answer: The isolation provided by network virtualization, allows PVNs to prevent misbehaving programs (or even devices) from interfering with the production network and leading to its collapse, which was a severe drawback in the past. Thus, in the case of misbehaving programs, the network operator can easily stop or delete a set of devices that run them. Virtualization imposes constraints on performance, but this does not prevent new NetApps from being developed. The review of the literature that was carried out in Chapter 2, showed that there were several drawbacks to introducing NetApps in PVNs. These included the following: (*i*) the PVN owners need to have an extensive knowledge of device instructions and NetApps before they can carry out simple tasks (*e.g.*, transferring and installing a NetApp), which is arduous and repetitive; (*ii*) the same tasks (*e.g.*, distribution of NetApps) have different and conflicting requirements (*e.g.*, minimal network interference or distribution time) at each stage of the service lifecycle (*i.e.*, deployment, operation, and optimization); (*iii*) the initial settings must be manually replicated in each EE to set up the logic rules for delegating the data flows for each new service; (*iv*) neither the technological nor academic world has been able to automate the NetApp management by including different types of EEs; (*v*) the lack of repositories and the unavailability of NetApps for downloading, restricts the distribution of developed

services to just a few PVN owners; (*vi*) when deploying new services, the PVN owners must now address issues of dependency and conflict between the NetApps; and, (*vii*) there is no descriptor to provide a detailed account of the VNAs or NetApps of different programmability-related technologies, which prevents some tasks from being automated (*e.g.*, management, configuration, and conflict detection in network services).

Research Question #2: *In light of the successful paradigm of OS-specific app stores, what are the essential design goals that can enable PVN owners, who lack any knowledge of the underlying infrastructure, to deploy and manage the network applications?*

Answer: In different fields, online marketplaces allowed end-users to deploy and manage applications with minimal or no knowledge about the underlying infrastructure. In Chapter 3, we reviewed the history of the marketplaces, and we identified the design goals that comprise a set of the suitable features that should be provided by network marketplaces. We argued that this set is divided into three main categories of design goals, namely: offer and distribution, network environment, and applications. In the first category, offer and distribution, we identified four design goals: access control; publishing guidelines; pricing methods; and notifications of important events. The design goals for the network environment category include: deployment of NetApps; support for different technologies, which allows NetApps to be "infrastructure-agnostic"; and monitoring and SLAs management. In addition, in the applications category, we determined that there were three crucial design goals, which include: consistency about NetApps and repositories; the relationship between NetApps, in particular, dependencies and conflicts; and, the management of the NetApps lifecycle. Finally, the survey about marketplaces involved compiling a list of research challenges, which must be overcome to make the adoption of network marketplaces a reality, in particular, with regard to challenges such as business model, evolution-aware, auditing, recommendation of NetApps, placement, security, and descriptors.

Research Question #3: *Considering the distribution time, CPU usage, and network overhead, what is the impact of using a platform on deploying applications in heterogeneous Programmable Virtual Networks?*

Answer: The App2net ecosystem, described in detail in Chapter 4, empowers PVN owners to deploy and manage NetApps in PVNs by means of heterogeneous EEs without knowing the specific features of the underlying environment. The experiments conducted in Chapter 5 show this impact in terms of the distribution time of a NetApp, CPU usage, and network overhead introduced by our ecosystem. In summary, considering the baseline results (obtained from the Sequential strategy) the distribution time was reduced, on average, by 84% for Ladder and Hub & Spoke topologies, while in the case of the Star topology, the distribution time was reduced by approximately 76%. It should be noted that reduction was observed when comparing the baseline results (Sequential strategy) with the average values obtained from the other strategies (Parallel, Gossip, and Groups). With regard to CPU usage, owing to the high values observed, two perspectives were analyzed: (*i*) the average CPU usage of the repository from "push" strategy and delay criterion and (*ii*) the average CPU usage of the PVN nodes from the "pull" strategy and link criterion. With regard to the baseline results, the strategies of Parallel, Gossip, and Groups increase the average CPU usage by $\approx 86\%$, $\approx 76\%$, and $\approx 3\%$ respectively in the NACR; and $\approx 2\%$, $\approx 16\%$, and $\approx 7\%$ in the PVN nodes. Although all the strategies increased the average CPU usage in both the NACR and PVN nodes, they also reduced the distribution time. The reason for this increase in the average CPU usage was that computing was required to control and perform the parallel transfers for the PVN nodes. Finally, the App2net ecosystem generated low traffic when communicating with the PVN nodes, and in all the analyzed cases, the total network overhead was less than 6%. As well as this, all the strategies reduce the total network traffic generated to transfer the same NetApp packages, for instance, Parallel, Groups, and Gossip reduce the total traffic generated by 1.85%, 22.69%, and 24.69% respectively, in comparison with the baseline results.

The investigation carried out during this thesis and the answers to the research questions provide us evidence of the feasibility of the App2net ecosystem for empowering PVN owners to deploy and manage NetApps into PVNs. In this way, all information provided and discussed, supports the proposed hypothesis: **"a platform inspired in online marketplaces enables PVN owners with a limited knowledge of the underlying infrastructure to overcome the main hurdles to introducing new NetApps into PVNs"**.

We designed the App2net ecosystem by addressing the current hurdles to introducing NetApps in PVNs (answer to Research Question 1) as well as the set of essential design goals obtained from our historical analysis of online marketplaces (answer to Research Question 2). For a better understanding of our proposed solution, in Section 4.5, we discussed the App2net ecosystem in light of the design goals and research challenges previously identified. In addition, we carried out two different evaluations of our ecosystem. In the first, we conducted a case study to seek evidence about whether it was feasible for third-party developers to describe and publish a network service for distinct programmability technologies. Such case study also described all the steps that a PVN owner must perform (who only has a minimal or no knowledge of the underlying infrastructure) to deploy and manage this network service in a PVN formed of different EEs using App2net ecosystem. In the second evaluation, we implemented a prototype of our ecosystem and after that, carried out the performance analysis which included the distribution time, CPU usage, and network overhead (answer to Research Question 3).

6.2 Summary of Main Contributions

This thesis has investigated whether it is feasible to design a platform, grounded in online marketplaces, that can deploy and manage new NetApps for PVNs formed of different EEs. The results of carrying out this investigation have led to the following contributions.

- **A survey of network programmability technologies.** The introduction of virtualization provides several benefits to computer networks, by making it possible to reintroduce the network programmability and overcome the problem of "Internet Ossification". Thus, the thesis included a review of the key programmability technologies and their features and made a comparison between the aspects required to deploy and manage NetApps.

- **A unified glossary of terms for network programmability.** In the area of network programmability, each technology refers to the same concepts in different ways. For instance, NFV describes NetApps as network functions, whereas OpenFlow is called by different names such as business applications and SDN control software. For this reason, a glossary was compiled and used in this thesis to bring together the different terms related to the same concepts found in multiple technologies.

- **A survey of online marketplaces.** Google Play and Apple App Store opened

up the mobile applications market to third-party developers, and this also enabled the smartphones end-users to deploy and manage applications with minimal or no knowledge of their devices. In our view, the successful model of today's marketplaces for mobile devices can provide inspiration and benefit to the networking market. For this reason, we reviewed the historical roadmap of networking paradigms and marketplaces to assess how paradigms and technologies have evolved over the years. We also investigated the current landscape regarding marketplaces for different paradigms, identified the essential design goals, and discussed the significant challenges that must be overcome to make the adoption of marketplaces in future networks a reality.

- **The NetApp concept.** This concept handled the network services as applications that could be installed, uninstalled, and managed and is essential to foresee the kind of marketplaces that are suitable for deploying and managing services in the future networks.

- **The App2net ecosystem architecture.** The App2net ecosystem empowers PVN owners to deploy and manage NetApps in PVNs by using different EEs without knowing the specific features of the underlying environment. Our integrated architecture also specifies a marketplace, in which third-party developers can describe and publish network services for different programmability technologies. Finally, the App2net RESTful API provides access to external software elements to request data, trigger actions, and send notifications and messages.

- **The App2net RESTful API.** The specification of the RESTful API that reveals the main functions of the App2net ecosystem. This API enables a modular architecture of App2net, in which an interested developer can replace an existing element with a new one that carries out at least the same actions. Furthermore, developers can program new modules that contain additional features. Thus, PVN owners can plug these new modules in the App2net and, consequently, they customize the ecosystem to better adapt to their requirements.

- **Network Application Descriptor.** An extension of the ETSI Network Service Descriptor, which makes it possible to describe the NetApps and VNAs of several programmability technologies, as well as provide all the information needed about the packages including conflicts, initial functional settings, available management actions, version, and the necessary EE.

6.3 Future Work

The investigation carried out in this thesis offers opportunities for further research. These opportunities are described in this section as future work.

- **Extending the Network Application Descriptor.** Affinities and anti-affinities between NetApps and VNAs can be employed by the PVN owners to select new NetApps. It is recommended that two NetApps with a high degree of affinity should work together to provide an improved environment for a network service. However, there is a rist that two NetApps with a high degree anti-affinity can interfere with each other's performance or even prevent their correct execution. Hence, it is of crucial importance to enable developers to map affinities and anti-affinities in our Network Application Descriptor. In addition, the App2net ecosystem must be identified and, if possible, change the order of data flows between the NetApps to a better sequence.

- **Integrating a solution for pricing and monitoring.** Although we referred to pricing and monitoring and SLA management as design goals, we did not provide the necessary mechanisms to support them in our ecosystem. We believe that disruptions or even service malfunction will inevitably reduce amount payable by the PVN owners. Thus, in our view, this is a very complex research goal, and is really a subject for another research endeavor.

- **Tacking the research challenges.** In our review of marketplaces, we found there were several research challenges, for which we sought a palliative or straightforward solution in the App2net ecosystem. However, we are in favor of carrying out more in-depth research to meet these challenges, which include the following: security issues, evolution-aware, placement, NetApp recommendations, and descriptors. Besides, there are two research challenges that must be addressed by the academia in future studies: auditing and forming a business model. Of course, new elements, procedures, tasks, or event platforms that are more robust, must be included to overcome all these research challenges.

REFERENCES

AKHTAR, N.; MATTA, I.; WANG, Y. Managing nfv using sdn and control theory. In: **NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium**. [S.l.: s.n.], 2016. p. 1005–1006.

AKHUNZADA, A. et al. Securing software defined networks: taxonomy, requirements, and open issues. **IEEE Communications Magazine**, IEEE, v. 53, n. 4, p. 36–44, 2015.

ALEXANDER, D. et al. The switchware active network architecture. **Network, IEEE**, v. 12, n. 3, p. 29–36, 1998. ISSN 0890-8044.

AMAZON. **Amazon Elastic Compute Cloud Documentation**. 2016. Available at: <http://docs.aws.amazon.com/AWSEC2/latest/APIReference/Welcome.html>. Accessed: March, 2017.

AMAZON. **How AWS Pricing Works**. [S.l.]: Amazon Web Services. White Paper, 2016. Available at: <https://d0.awsstatic.com/whitepapers/aws_pricing_overview.pdf>. Accessed: January, 2017.

ANDERSON, T. et al. Overcoming the internet impasse through virtualization. **Computer**, v. 38, n. 4, p. 34–41, April 2005. ISSN 0018-9162.

APPLE INC. **Apple App Store**. 2018. Available at: <https://itunes.apple.com>. Accessed: February, 2018.

ARLEIN, R. M.; BETGÉ-BREZETZ, S.; ENSOR, J. R. Adaptive notification framework for converged environments. **Bell Labs Technical Journal**, v. 13, n. 2, p. 155–159, Summer 2008. ISSN 1089-7089.

ARNTZEN, I. M.; JOHANSEN, D. A stateful and open publish-subscribe structure for online marketplaces. In: **25th IEEE International Conference on Distributed Computing Systems Workshops**. [S.l.: s.n.], 2005. p. 431–437. ISSN 1545-0678.

AUBRECHT, V.; KOUTNY, T. Revisiting the darpa's idea of a programmable network. In: **Communications and Information Systems Conference (MCC), 2012 Military**. [S.l.: s.n.], 2012. p. 1–6.

BENTON, K.; CAMP, L. J.; SMALL, C. Openflow vulnerability assessment. In: **Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking**. New York, NY, USA: ACM, 2013. (HotSDN '13), p. 151–152. ISBN 978-1-4503-2178-5. Available from Internet: <http://doi.acm.org/10.1145/2491185.2491222>.

BERA, S.; MISRA, S.; RODRIGUES, J. J. P. C. Cloud computing applications for smart grid: A survey. **IEEE Transactions on Parallel and Distributed Systems**, v. 26, n. 5, p. 1477–1494, May 2015. ISSN 1045-9219.

BERNARDO, D. V.; CHUA, B. B. Introduction and analysis of sdn and nfv security architecture (sn-seca). In: **2015 IEEE 29th International Conference on Advanced Information Networking and Applications**. [S.l.: s.n.], 2015. p. 796–801. ISSN 1550-445X.

BIESZCZAD, A.; PAGUREK, B.; WHITE, T. Mobile agents for network management. **Communications Surveys Tutorials, IEEE**, v. 1, n. 1, p. 2 –9, 1998. ISSN 1553-877X.

BLESS, R.; FLITTNER, M. Towards corporate confidentiality preserving auditing mechanisms for clouds. In: IEEE. **Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on**. [S.l.], 2014. p. 381–387.

BOBADILLA, J. et al. Recommender systems survey. **Knowledge-Based Systems**, v. 46, p. 109 – 132, 2013. ISSN 0950-7051. Available from Internet: <http://www.sciencedirect.com/science/article/pii/S0950705113001044>.

BOLLA, R.; BRUSCHI, R. Linux software router: data plane optimization and performance evaluation. **Journal of Networks**, v. 2, n. 3, p. 6–17, 2007.

BOLLA, R.; BRUSCHI, R. Pc-based software routers: High performance and application service support. In: **Proceedings of the ACM Workshop on Programmable Routers for Extensible Services of Tomorrow**. New York, NY, USA: ACM, 2008. (PRESTO '08), p. 27–32. ISBN 978-1-60558-181-1. Available from Internet: <http://doi.acm.org/10.1145/1397718.1397725>.

BOSSHART, P. et al. P4: Programming protocol-independent packet processors. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 44, n. 3, p. 87–95, jul. 2014. ISSN 0146-4833. Available from Internet: <http://doi.acm.org/10.1145/2656877.2656890>.

BOUBENDIR, A.; BERTIN, E.; SIMONI, N. Naas architecture through sdn-enabled nfv: Network openness towards web communication service providers. In: **NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium**. [S.l.: s.n.], 2016. p. 722–726.

BREMLER-BARR, A. et al. Deep packet inspection as a service. In: **Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies**. New York, NY, USA: ACM, 2014. (CoNEXT '14), p. 271–282. ISBN 978-1-4503-3279-8. Available from Internet: <http://doi.acm.org/10.1145/2674005.2674984>.

BRONSTEIN, Z. et al. Uniform handling and abstraction of nfv hardware accelerators. **IEEE Network**, v. 29, n. 3, p. 22–29, May 2015. ISSN 0890-8044.

BUREGIO, V. A. et al. Specification, design and implementation of a reuse repository. In: **31st Annual International Computer Software and Applications Conference (COMPSAC 2007)**. [S.l.: s.n.], 2007. v. 1, p. 579–582. ISSN 0730-3157.

BURKE, R. Hybrid recommender systems: Survey and experiments. **User Modeling and User-Adapted Interaction**, v. 12, n. 4, p. 331–370, Nov 2002. ISSN 1573-1391. Available from Internet: <https://doi.org/10.1023/A:1021240730564>.

CAMPBELL, A. T. et al. A survey of programmable networks. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 29, n. 2, p. 7–23, abr. 1999. ISSN 0146-4833. Available from Internet: <http://doi.acm.org/10.1145/505733.505735>.

CANINI, M. et al. Stn: A robust and distributed sdn control plane. **Open Networking Summit**, v. 490, 2014.

CANINI, M. et al. A distributed and robust sdn control plane for transactional network updates. In: **2015 IEEE Conference on Computer Communications (INFOCOM)**. [S.l.: s.n.], 2015. p. 190–198. ISSN 0743-166X.

CANINI, M. et al. A nice way to test openflow applications. In: **Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)**. [S.l.: s.n.], 2012. p. 127–140.

CANONICAL. **Juju Charm Documentation**. 2017. Available at: <https://jujucharms.com/docs/2.2/getting-started>. Accessed: March, 2017.

CARAPINHA, J.; JIMÉNEZ, J. Network virtualization: A view from the bottom. In: **Proceedings of the 1st ACM Workshop on Virtualized Infrastructure Systems and Architectures**. New York, NY, USA: ACM, 2009. (VISA '09), p. 73–80. ISBN 978-1-60558-595-6. Available from Internet: <http://doi.acm.org/10.1145/1592648.1592660>.

CARELLA, G. et al. Quality audit and resource brokering for network functions virtualization (nfv) orchestration in hybrid clouds. In: IEEE. **Global Communications Conference (GLOBECOM), 2015 IEEE**. [S.l.], 2015. p. 1–6.

CEN, L.; LI, C.; WANG, W. **A Feedback Mechanism Based on Randomly Suppressed Timer for ForCES Protocol**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015. 103–110 p. ISBN 978-3-662-46578-3. Available from Internet: <http://dx.doi.org/10.1007/978-3-662-46578-3_13>.

CHESS, D.; HARRISON, C.; KERSHENBAUM, A. Mobile agents: Are they a good idea? In: **MOS'97**. [S.l.]: Springer Berlin / Heidelberg, 1997, (Lecture Notes in Computer Science, v. 1222). p. 25–45.

CHOWDHURY, N.; BOUTABA, R. Network virtualization: state of the art and research challenges. **Communications Magazine, IEEE**, v. 47, n. 7, p. 20–26, July 2009. ISSN 0163-6804.

CHOWDHURY, N. M. K.; BOUTABA, R. A survey of network virtualization. **Computer Networks**, v. 54, n. 5, p. 862 – 876, 2010. ISSN 1389-1286. Available from Internet: <http://www.sciencedirect.com/science/article/pii/S1389128609003387>.

CHUNG, L. et al. **Non-Functional Requirements in Software Engineering**. 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2000.

CISCO. **Cisco Application-Oriented Networking**. [S.l.]: CISCO. White Paper, 2005. Available at: <http://www.cisco.com/c/en/us/products/collateral/interfaces-modules/application-networking-services-modules/prod_white_paper0900aecd8033e9a4.pdf>. Accessed: June, 2012.

CISCO. **Router Virtualization in Service Providers**. [S.l.]: CISCO. White Paper, 2008. Available at: <http://www.cisco.com/c/en/us/td/docs/ios_xr_sw/iosxr_r3-2/interfaces/configuration/guide/int_c32/hc32logr.pdf>. Accessed: February, 2016.

CISCO. **Cisco Open Network Environment: Network Programmability and Virtual Network Overlays**. [S.l.]: CISCO. White Paper, 2012. Available at: <http://www.cisco.com/en/US/prod/collateral/iosswrel/content/white_paper_c11-707978.pdf>. Accessed: January, 2015.

CISCO. **One Platform Kit: The Power to Innovate**. [S.l.]: CISCO. White Paper, 2012. Available at: <http://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/white_paper_c11-708666.pdf>. Accessed: January, 2016.

CISCO. **Cisco Extensible Network Controller: An OpenDaylight-Based Controller**. [S.l.]: CISCO. White Paper, 2014. Available at: <http://www.cisco.com/c/en/us/products/collateral/cloud-systems-management/extensible-network-controller-xnc/data_sheet_c78-729453.pdf>. Accessed: January, 2016.

CISCO. **Transformation Through Innovation: A Strategy for Service Provider Success**. [S.l.]: CISCO. White Paper, 2016. Available at: <http://www.cisco.com/c/en/us/solutions/collateral/service-provider/network-infrastructure/white-paper-c11-732692.pdf>. Accessed: June, 2017.

CISCO. **Transformation Through Innovation: A service provider strategy to prosper from digitization**. [S.l.]: CISCO. White Paper, 2017. Available at: <http://www.cisco.com/c/dam/en/us/solutions/service-provider/transformation-through-innovation/cisco_gsp_whitepaper_interactive_v3.pdf>. Accessed: June, 2017.

CISCO SYSTEMS. **Cisco Systems Inc.** 2016. Available at: <http://www.cisco.com/>. Accessed: February, 2016.

COSTA CORDEIRO, W. L. da; MARQUES, J. A.; GASPARY, L. P. Data plane programmability beyond openflow: Opportunities and challenges for network and service operations and management. **Journal of Network and Systems Management**, v. 25, n. 4, p. 784–818, Oct 2017. ISSN 1573-7705. Available from Internet: <https://doi.org/10.1007/s10922-017-9423-2>.

CUADRADO, F.; DUENAS, J. C. Mobile application stores: success factors, existing approaches, and future developments. **IEEE Communications Magazine**, v. 50, n. 11, p. 160–167, November 2012. ISSN 0163-6804.

DANTU, R.; ANDERSON, T.; GOPAL, R. **Forwarding and Control Element Separation (ForCES) Framework (RFC 3746)**. United States: RFC Editor, 2004.

DARGAHI, T. et al. A survey on the security of stateful sdn data planes. **IEEE Communications Surveys Tutorials**, v. 19, n. 3, p. 1701–1725, thirdquarter 2017.

DONG, Y. et al. High performance network virtualization with sr-iov. In: **HPCA - 16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture**. [S.l.: s.n.], 2010. p. 1–10. ISSN 1530-0897.

DZIUBINSKI, M.; GOYAL, S.; MINARSCH, D. E. Dynamic conflict on a network. In: **Proceedings of the 2016 ACM Conference on Economics and Computation**. New York, NY, USA: ACM, 2016. (EC '16), p. 655–656. ISBN 978-1-4503-3936-0. Available from Internet: <http://doi.acm.org/10.1145/2940716.2940776>.

EDWARDS, B.; GIULIANO, L.; WRIGHT, B. **Interdomain Multicast Routing: Practical Juniper Networks and Cisco Systems Solutions**. Addison-Wesley, 2002. ISBN 9780201746129. Available from Internet: <https://books.google.com.br/books?id=2Nhv8mdg-o4C>.

EGILMEZ, H.; CIVANLAR, S.; TEKALP, A. An optimization framework for qos-enabled adaptive video streaming over openflow networks. **Multimedia, IEEE Transactions on**, v. 15, n. 3, p. 710–715, 2013. ISSN 1520-9210.

ERICKSON, D. The beacon openflow controller. In: **Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking**. New York, NY, USA: ACM, 2013. (HotSDN '13), p. 13–18. ISBN 978-1-4503-2178-5. Available from Internet: <http://doi.acm.org/10.1145/2491185.2491189>.

ESPOSITO, C.; CIAMPI, M. On security in publish/subscribe services: A survey. **IEEE Communications Surveys Tutorials**, v. 17, n. 2, p. 966–997, Secondquarter 2015. ISSN 1553-877X.

ESTEVES, R. P.; GRANVILLE, L. Z.; BOUTABA, R. On the management of virtual networks. **IEEE Communications Magazine**, v. 51, n. 7, p. 80–88, July 2013. ISSN 0163-6804.

EUCALYPTUS. **Open Source Private Cloud Software**. 2015. Available at: <http://www.eucalyptus.com/>. Accessed: August, 2015.

EXTREME NETWORKS. **Extreme Networks Inc.** . 2016. Available at: <http://extremenetworks.com/>. Accessed: February, 2016.

FALL, K. et al. Routebricks: Enabling general purpose network infrastructure. **SIGOPS Oper. Syst. Rev.**, ACM, New York, NY, USA, v. 45, n. 1, p. 112–125, feb. 2011. ISSN 0163-5980. Available from Internet: <http://doi.acm.org/10.1145/1945023.1945037>.

FEAMSTER, N.; REXFORD, J.; ZEGURA, E. The road to sdn: An intellectual history of programmable networks. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 44, n. 2, p. 87–98, abr. 2014. ISSN 0146-4833.

GARCÍA-VALLS, M.; BELLAVISTA, P.; GOKHALE, A. Reliable software technologies and communication middleware: A perspective and evolution directions for cyber-physical system, mobility, and cloud computing. **Future Generation Computer Systems**, v. 71, p. 171 – 176, 2017. ISSN 0167-739X.

GELBERGER, A.; YEMINI, N.; GILADI, R. Performance analysis of software-defined networking (sdn). In: **2013 IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems**. [S.l.: s.n.], 2013. p. 389–393. ISSN 1526-7539.

GEORGOPOULOS, P. et al. Towards network-wide qoe fairness using openflow-assisted adaptive video streaming. In: **Proceedings of the 2013 ACM SIGCOMM Workshop on Future Human-centric Multimedia Networking**. New York, NY, USA: ACM, 2013. (FhMN '13), p. 15–20. ISBN 978-1-4503-2183-9. Available from Internet: <http://doi.acm.org/10.1145/2491172.2491181>.

GERLA, M. Vehicular cloud computing. In: **Ad Hoc Networking Workshop (Med-Hoc-Net), 2012 The 11th Annual Mediterranean**. [S.l.: s.n.], 2012. p. 152–155.

GIBB, G. et al. Netfpga – an open platform for teaching how to build gigabit-rate network switches and routers. **IEEE Transactions on Education**, v. 51, n. 3, p. 364–369, Aug 2008. ISSN 0018-9359.

GOOGLE. **Google Play**. 2018. Available at: <https://play.google.com/store>. Accessed: February, 2018.

GRILICHES, E. **The Junos SDK: A Market Update on Innovation**. [S.l.]: White Paper, 2009. Available at: <http://juniper.jmsshare.com/pdfs/IDC-The_Junos_SDK-An_Update_on_Innovation.pdf>. Accessed: March, 2016.

GUDE, N. et al. Nox: Towards an operating system for networks. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 38, n. 3, p. 105–110, jul. 2008. ISSN 0146-4833. Available from Internet: <http://doi.acm.org/10.1145/1384609.1384625>.

GUSEV, M. et al. Scalable architecture of alert notification as a service. In: **International Conference on Information Society (i-Society 2014)**. [S.l.: s.n.], 2014. p. 80–85.

HAAS, R.; DROZ, P.; STILLER, B. Distributed service deployment over programmable networks. In: **Proceedings. 12th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM) 2001**. [S.l.: s.n.], 2001.

HALEPLIDIS, E. **Forwarding and Control Element Separation (ForCES) Model Extension**. RFC Editor, 2015. RFC 7408. (Request for Comments, 7408). Available from Internet: <https://rfc-editor.org/rfc/rfc7408.txt>.

HALEPLIDIS, E. et al. Network programmability with forces. **IEEE Communications Surveys Tutorials**, v. 17, n. 3, p. 1423–1440, thirdquarter 2015. ISSN 1553-877X.

HAMED, H.; AL-SHAER, E. Taxonomy of conflicts in network security policies. **IEEE Communications Magazine**, v. 44, n. 3, p. 134–141, March 2006. ISSN 0163-6804.

HARES, S. **Analysis of Comparisons between OpenFlow and ForCES**. [S.l.], 2013. Work in Progress. Available from Internet: <https://tools.ietf.org/html/draft-hares-forces-vs-openflow-00>.

HERBAUT, N. et al. **Network Function Store**. [S.l.], 2015.

HICKS, M. et al. Plan: a packet language for active networks. **SIGPLAN Not.**, ACM, New York, NY, USA, v. 34, p. 86–93, September 1998. ISSN 0362-1340. Available from Internet: <http://doi.acm.org/10.1145/291251.289431>.

HOBAN, A. et al. **OSM Release Two: A Technical Overview**. [S.l.]: ETSI OSM. White Paper, 2017. Available at: <https://osm.etsi.org/images/OSM-Whitepaper-TechContent-ReleaseTWO-FINAL.pdf>. Accessed: June, 2017.

HOCK, D. et al. Pareto-optimal resilient controller placement in sdn-based core networks. In: IEEE. **Teletraffic Congress (ITC), 2013 25th International**. [S.l.], 2013. p. 1–9.

HPE. **Hewlett Packard Enterprise SDN App Store**. 2016. Available at: <https://saas.hpe.com/marketplace/sdn>. Accessed: February, 2016.

HUANG, J. C. et al. Ally: Os-transparent packet inspection using sequestered cores. In: **2011 ACM/IEEE Seventh Symposium on Architectures for Networking and Communications Systems**. [S.l.: s.n.], 2011. p. 1–11.

HUNGYO, M.; PANDEY, M. Sdn based implementation of publish/subscribe paradigm using openflow multicast. In: **2016 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)**. [S.l.: s.n.], 2016. p. 1–6.

ISINKAYE, F.; FOLAJIMI, Y.; OJOKOH, B. Recommendation systems: Principles, methods and evaluation. **Egyptian Informatics Journal**, v. 16, n. 3, p. 261 – 273, 2015. ISSN 1110-8665. Available from Internet: <http://www.sciencedirect.com/science/article/pii/S1110866515000341>.

ISOLANI, P. H. et al. Interactive monitoring, visualization, and configuration of openflow-based sdn. In: **2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)**. [S.l.: s.n.], 2015. p. 207–215. ISSN 1573-0077.

IYENGAR, S. S.; BROOKS, R. R. **Distributed sensor networks: sensor networking and applications**. [S.l.]: CRC press, 2016.

JACOBS, A. S. et al. Affinity Measurement for NFV-enabled Networks: A Criteria-based Approach. In: **15th IFIP/IEEE International Symposium on Integrated Network Management (IM 2017)**. [S.l.: s.n.], 2017. Accepted. To be published.

JAIN, M.; DOVROLIS, C. End-to-end available bandwidth: measurement methodology, dynamics, and relation with tcp throughput. **SIGCOMM**, ACM, NY, USA, v. 32, n. 4, p. 295–308, aug 2002. ISSN 0146-4833.

JAIN, R. **Art of Computer Systems Performance Analysis: Techniques For Experimental Design Measurements Simulation and Modeling**. [S.l.]: Wiley, 2015. ISBN 9781118858424.

JIN, Y.; WEN, Y.; WESTPHAL, C. Towards joint resource allocation and routing to optimize video distribution over future internet. In: **IFIP Networking Conference (IFIP Networking), 2015**. [S.l.: s.n.], 2015. p. 1–9.

JU, C.; XU, C. A new collaborative recommendation approach based on users clustering using artificial bee colony algorithm. **The Scientific World Journal**, v. 2013, p. 1 – 9, 2013.

JUNIPER. **IP Multicast Bandwidth Adjustment for QoS Overview**. [S.l.]: Juniper. Documentation., 2010. Available at: <https://www.juniper.net/documentation/en_US/junose11.3/topics/concept/qos-parameters-ip-multicast.html>. Accessed: May, 2017.

JUNIPER. **Protected System Domain Configuration Guide**. [S.l.]: Juniper Networks. White Paper, 2012. Available at: <https://www.juniper.net/techpubs/en_US/junos12.3/information-products/topic-collections/config-guide-psd/config-guide-psd.pdf>. Accessed: April, 2016.

JUNIPER NETWORKS. **Transforming to DevOps with Junos OS**. [S.l.]: Juniper Networks. White Paper, 2015. Available at: <https://www.juniper.net/us/en/local/pdf/whitepapers/2000586-en.pdf>. Accessed: November, 2015.

JUNIPER NETWORKS. **Juniper Developer Network**. 2016. Available at: <https://developer.juniper.net/>. Accessed: February, 2016.

JUNIPER NETWORKS. **Juniper Networks Inc.** 2016. Available at: <http://www.juniper.net/us/en/>. Accessed: February, 2016.

KAMIYAMA, N. et al. Impact of topology on parallel video streaming. In: **2010 IEEE Network Operations and Management Symposium - NOMS 2010**. [S.l.: s.n.], 2010. p. 607–614. ISSN 1542-1201.

KANDPAL, M.; GAHLAWAT, M.; PATEL, K. Role of predictive modeling in cloud services pricing: A survey. In: **2017 7th International Conference on Cloud Computing, Data Science Engineering - Confluence**. [S.l.: s.n.], 2017. p. 249–254.

KELLY, J. Developing innovative embedded applications in the network with the junos sdk. In: ____. **Network-Embedded Management and Applications: Understanding Programmable Networking Infrastructure**. New York, NY: Springer New York, 2013. p. 137–157. ISBN 978-1-4419-6769-5. Available from Internet: <http://dx.doi.org/10.1007/978-1-4419-6769-5_7>.

KELLY, J.; ARAUJO, W.; BANERJEE, K. Rapid service creation using the junos sdk. In: **2nd ACM SIGCOMM Workshop Presto**. NY, USA: ACM, 2009. p. 7–12. ISBN 978-1-60558-446-1.

KHAN, A. et al. Network virtualization: a hypervisor for the internet? **IEEE Communications Magazine**, v. 50, n. 1, p. 136–143, January 2012. ISSN 0163-6804.

KIM, H.; FEAMSTER, N. Improving network management with software defined networking. **IEEE Communications Magazine**, v. 51, n. 2, p. 114–119, February 2013. ISSN 0163-6804.

KIRAN, S.; KINGHORN, G. **Cisco Open Network Environment: Bring the Network Closer to Applications**. [S.l.]: CISCO. White Paper, 2015. Available at: <http://www.cisco.com/c/en/us/products/collateral/switches/nexus-1000v-switch-vmware-vsphere/white_paper_c11-728045.pdf>. Accessed: January, 2016.

KOHLER, E. et al. The click modular router. **ACM Transactions on Computer Systems**, ACM, New York, NY, USA, v. 18, n. 3, p. 263–297, aug. 2000. ISSN 0734-2071. Available from Internet: <http://doi.acm.org/10.1145/354871.354874>.

KOPONEN, T. et al. Onix: A distributed control platform for large-scale production networks. In: **Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation**. Berkeley, CA, USA: USENIX Association, 2010. (OSDI'10), p. 351–364. Available from Internet: <http://dl.acm.org/citation.cfm?id=1924943.1924968>.

KOSLOVSKI, G. P.; PRIMET, P. V.-B.; CHARAO, A. S. VXDL: Virtual Resources and Interconnection Networks Description Language. In: **Networks for Grid Applications**. [S.l.: s.n.], 2009. v. 2, p. 138–154. ISBN 978-3-642-02080-3.

KUKLIŃSKI, S. Programmable management framework for evolved sdn. In: **2014 IEEE Network Operations and Management Symposium (NOMS)**. [S.l.: s.n.], 2014. p. 1–8. ISSN 1542-1201.

LANGE, D. B.; MITSURU, O. **Programming and Deploying Java Mobile Agents Aglets**. 1st. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1998. ISBN 0201325829.

LANGE, D. B.; OSHIMA, M. Seven good reasons for mobile agents. **Commun. ACM**, ACM, New York, NY, USA, v. 42, p. 88–89, March 1999. ISSN 0001-0782. Available from Internet: <http://doi.acm.org/10.1145/295685.298136>.

LAZAR, A. Programming telecommunication networks. **Network, IEEE**, v. 11, n. 5, p. 8–18, 1997. ISSN 0890-8044.

LEVI, D.; SCHOENWAELDER, J. **Definitions of Managed Objects for the Delegation of Management Scripts (RFC 2592)**. United States: RFC Editor, 1999.

LEVI, D.; SCHOENWAELDER, J. **Definitions of Managed Objects for the Delegation of Management Scripts (RFC 3165)**. United States: RFC Editor, 2001.

LI, S. et al. Protocol oblivious forwarding (pof): Software-defined networking with enhanced programmability. **IEEE Network**, v. 31, n. 2, p. 58–66, March 2017. ISSN 0890-8044.

LIN, K. et al. Balancing energy consumption with mobile agents in wireless sensor networks. **Future Generation Computer Systems**, v. 28, n. 2, p. 446 – 456, 2012. ISSN 0167-739X. Available from Internet: <http://www.sciencedirect.com/science/article/pii/S0167739X1100029X>.

LINDEN, G.; SMITH, B.; YORK, J. Amazon.com recommendations: item-to-item collaborative filtering. **IEEE Internet Computing**, v. 7, n. 1, p. 76–80, Jan 2003. ISSN 1089-7801.

LIU, Y.; PLALE, B. **Survey of publish subscribe event systems**. [S.l.], 2003. Available at: <ftp://cs.indiana.edu/pub/techreports/TR574.pdf>. Accessed: December, 2016.

LLORENTE, I. M. The limits to cloud price reduction. **IEEE Cloud Computing**, v. 4, n. 3, p. 8–13, 2017. ISSN 2325-6095.

LU, J. et al. Recommender system application developments: A survey. **Decision Support Systems**, v. 74, p. 12 – 32, 2015. ISSN 0167-9236. Available from Internet: <http://www.sciencedirect.com/science/article/pii/S0167923615000627>.

LUIZELLI, M. C. et al. Piecing together the nfv provisioning puzzle: Efficient placement and chaining of virtual network functions. In: **2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)**. [S.l.: s.n.], 2015. p. 98–106. ISSN 1573-0077.

LUIZELLI, M. C. et al. How physical network topologies affect virtual network embedding quality: A characterization study based on isp and datacenter networks. **Journal of Network and Computer Applications**, v. 70, p. 1 – 16, 2016. ISSN 1084-8045. Available from Internet: <http://www.sciencedirect.com/science/article/pii/S1084804516300959>.

MA, S. et al. Pdni: A distributed framework for nfv infrastructure. In: IEEE. **Parallel Architectures, Algorithms and Programming (PAAP), 2015 Seventh International Symposium on**. [S.l.], 2015. p. 34–40.

MACEDO, D. F. et al. Programmable networks – from software-defined radio to software-defined networking. **IEEE Communications Surveys Tutorials**, v. 17, n. 2, p. 1102–1125, Secondquarter 2015. ISSN 1553-877X.

MACQUEEN, J. Some Methods for Classification and Analysis of Multivariate Observations. In: **In 5-th Berkeley Symposium on Mathematical Statistics and Probability**. [S.l.: s.n.], 1967. p. 281–297.

MANZALINI, A.; CRESPI, N. An Edge Operating System Enabling Anything-as-a-Service. **IEEE Communications Magazine**, v. 54, n. 3, p. 62–67, March 2016. ISSN 0163-6804.

MARTIN, W. et al. A survey of app store analysis for software engineering. **IEEE Transactions on Software Engineering**, PP, n. 99, p. 1–1, 2017. ISSN 0098-5589.

MARTINS, R. F. T. et al. Introdução à linguagem p4 teoria e prática. In: SBC. **Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC), 2018 SBC**. [S.l.], 2018.

MATIAS, J. et al. Toward an sdn-enabled nfv architecture. **IEEE Communications Magazine**, v. 53, n. 4, p. 187–193, April 2015. ISSN 0163-6804.

MCKEOWN, N. et al. Openflow: Enabling innovation in campus networks. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 38, n. 2, p. 69–74, mar. 2008. ISSN 0146-4833. Available from Internet: <http://doi.acm.org/10.1145/1355734.1355746>.

MEDVED, J. et al. Opendaylight: Towards a model-driven sdn controller architecture. In: **World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on a**. [S.l.: s.n.], 2014. p. 1–6.

MELL, P.; GRANCE, T. **The NIST Definition of Cloud Computing**. [S.l.], 2011. Available at: <https://csrc.nist.gov/publications/detail/sp/800-145/final>. Accessed: September, 2017.

MENYCHTAS, A. et al. A marketplace framework for trading cloud-based services. In: **Proceedings of the 8th International Conference on Economics of Grids, Clouds, Systems, and Services**. Berlin, Heidelberg: Springer-Verlag, 2012. (GECON'11), p. 76–89. ISBN 978-3-642-28674-2. Available from Internet: <http://dx.doi.org/10.1007/978-3-642-28675-9_6>.

MERWE, J. V. d.; KALMANEK, C. **Control Plane Scaling and Router Virtualization**. [S.l.]: White Paper, 2009. Tech. Rep. 2000261-001-EN, Juniper Networks, Feb. 2009.

MICROSOFT. **Microsoft Azure API Management**. 2016. Available at: <https://docs.microsoft.com/en-us/rest/api/apimanagement/>. Accessed: March, 2017.

MIJUMBI, R. et al. Network function virtualization: State-of-the-art and research challenges. **IEEE Communications Surveys Tutorials**, v. 18, n. 1, p. 236–262, Firstquarter 2016. ISSN 1553-877X.

MOENS, H.; TURCK, F. D. Vnf-p: A model for efficient placement of virtualized network functions. In: IEEE. **Network and Service Management (CNSM), 2014 10th International Conference on**. [S.l.], 2014. p. 418–423.

MOORE, J. T.; NETTLES, S. M. Towards practical programmable packets. In: **in Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2001**. [S.l.: s.n.], 2001. p. 41–50.

MORRIS, R. et al. The click modular router. **SIGOPS Oper. Syst. Rev.**, ACM, New York, NY, USA, v. 33, n. 5, p. 217–231, dec. 1999. ISSN 0163-5980. Available from Internet: <http://doi.acm.org/10.1145/319344.319166>.

NAKAO, A. **Deeply Programmable Network: extending Software Defined Network to support deeper programmability**. 2013. Available at: <http://im2013.ieee-im.org/sites/default/files/presentations/IM2013_Keynote_Nakao-Aki.pdf>. Accessed: March, 2015.

NAOUS, J. et al. Netfpga: Reusable router architecture for experimental research. In: **Proceedings of the ACM Workshop on Programmable Routers for Extensible Services of Tomorrow**. New York, NY, USA: ACM, 2008. (PRESTO '08), p. 1–7. ISBN 978-1-60558-181-1. Available from Internet: <http://doi.acm.org/10.1145/1397718.1397720>.

NAWAF, M. M.; TORBEY, Z. Replica update strategy in mobile ad hoc networks. In: **MEDES**. NY, USA: ACM, 2009. p. 474–476. ISBN 978-1-60558-829-2.

NETWORKS, J. **Transforming to DevOps with Junos OS**. [S.l.], 2015. Available at: <https://www.juniper.net/assets/fr/fr/local/pdf/whitepapers/2000586-en.pdf>. Accessed: March, 2017.

NETWORKS, J. **Automating Dynamic QoS Management with Juniper Extension Toolkit**. [S.l.], 2017. Available at: <https://www.juniper.net/assets/us/en/local/pdf/whitepapers/2000650-en.pdf>. Accessed: June, 2017.

NFV ISG. **Network Functions Virtualisation(NFV): An Introduction, Benefits, Enablers, Challenges & Call for Action**. [S.l.], 2012. Available at: <https://portal.etsi.org/nfv/nfv_white_paper.pdf>. Accessed: December, 2015.

NFV ISG. **Network Functions Virtualisation(NFV): Architectural Framework**. [S.l.], 2013. Available at: <http://www.etsi.org/deliver/etsi_gs/nfv/001_099/002/01.01.01_60/gs_nfv002v010101p.pdf>. Accessed: December, 2015.

NFV ISG. **Network Functions Virtualisation(NFV): Management and Orchestration**. [S.l.], 2014. Available at: <http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_nfv-man001v010101p.pdf>. Accessed: February, 2016.

NUNES, B. A. A. et al. A survey of software-defined networking: Past, present, and future of programmable networks. **IEEE Communications Surveys Tutorials**, v. 16, n. 3, p. 1617–1634, Third 2014. ISSN 1553-877X.

OLIVER, E. A Survey of Platforms for Mobile Networks Research. **SIGMOBILE Mob. Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 12, n. 4, p. 56–63, feb. 2009. ISSN 1559-1662. Available from Internet: <http://doi.acm.org/10.1145/1508285.1508292>.

ONF. **OpenFlow Switch Specification version 1.5.1**. 2015. Available at: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.1.pdf>. Accessed: December, 2015.

ONF. **Open Networking Foundation**. 2016. Available at: <http://www.opennetworking.org/>. Accessed: January, 2016.

OpenStack. **Murano Specification API**. [S.l.], 2013. Available at: <https://docs.openstack.org/developer/murano/>. Accessed: February, 2017.

OPENSTACK. **Open source software for building private and public clouds**. 2016. Available at: <http://www.openstack.org/>. Accessed: January, 2016.

OPENSTACK. **OpenStack Documentation for Ocata**. 2017. Available at: <https://docs.openstack.org/>. Accessed: March, 2017.

PAKZAD, F. et al. Efficient topology discovery in openflow-based software defined networks. **Computer Communications**, v. 77, p. 52 – 61, 2016. ISSN 0140-3664. Available from Internet: <http://www.sciencedirect.com/science/article/pii/S0140366415003527>.

PALKAR, S. et al. E2: a framework for nfv applications. In: ACM. **Proceedings of the 25th Symposium on Operating Systems Principles**. [S.l.], 2015. p. 121–136.

PAYNE, J.; SHAIO, S.; HOFF, A. V. **Method for the distribution of code and data updates**. Google Patents, 1999. US Patent 5,919,247. Available from Internet: <http://google.com/patents/US5919247>.

PINHEIRO, R. S. et al. Reposdn: An repository organization and coordination method of software defined networks applications. In: **NOMS 2014**. [S.l.: s.n.], 2014. p. 1–4. ISSN 1542-1201.

PORTAL, S. T. S. **App Stores - Statistics & Facts**. 2017. Available at: <https://www.statista.com/topics/1729/app-stores/>. Accessed: July, 2017.

PRITCHETT, D. Base: An acid alternative. **Queue**, ACM, New York, NY, USA, v. 6, n. 3, p. 48–55, may 2008. ISSN 1542-7730. Available from Internet: <http://doi.acm.org/10.1145/1394127.1394128>.

PSOUNIS, K. Active networks: Applications, security, safety, and architectures. **Communications Surveys Tutorials, IEEE**, v. 2, n. 1, p. 2 –16, 1999. ISSN 1553-877X.

RAMOS, A. et al. **Specification of the Network Function Framework and T-NOVA Marketplace**. [S.l.], 2014. Available at: <http://www.t-nova.eu/wp-content/uploads/2016/03/TNOVA_D2.42_Specification_of_the_Network_Function_Framework_and_T-NOVA_Marketplace.pdf>. Accessed: March, 2016.

REYNAUD, F. et al. Attacks against network functions virtualization and software-defined networking: State-of-the-art. In: IEEE. **NetSoft Conference and Workshops (NetSoft), 2016 IEEE**. [S.l.], 2016. p. 471–476.

RICCI, F.; ROKACH, L.; SHAPIRA, B. Recommender systems: Introduction and challenges. In: ____. **Recommender Systems Handbook**. Boston, MA: Springer US, 2015. p. 1–34. ISBN 978-1-4899-7637-6. Available from Internet: <https://doi.org/10.1007/978-1-4899-7637-6_1>.

SALIM, J. H. **Forwarding and Control Element Separation (ForCES) Protocol Extensions**. RFC Editor, 2015. RFC 7391. (Request for Comments, 7391). Available from Internet: <https://rfc-editor.org/rfc/rfc7391.txt>.

SALLAM, A. A.; UDGATA, S. K. An integrated architecture for notification system to enhance the efficiency of mobile marketplace. In: **2011 International Conference on Business, Engineering and Industrial Applications**. [S.l.: s.n.], 2011. p. 198–202.

SANTOS, R. L. d. et al. iMPROVE: Enhancing the introduction of services on programmable virtual networks. In: **2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)**. [S.l.: s.n.], 2017. p. 500–507. ISSN 1550-445X.

SANTOS, R. L. dos et al. App2net A Platform to Transfer and Configure Applications on Programmable Virtual Networks. In: **20th IEEE Symposium on Computers and Communication (ISCC 2015)**. Larnaca, Cyprus: [s.n.], 2015. p. 335–342.

SCHEFFCZYK, J. et al. Efficient (in-)consistency management for heterogeneous repositories. In: DOSCH, W.; LEE, R. Y. (Ed.). **Proceedings of the ACIS Fourth International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD'03), October 16-18, 2003, Lübeck, Germany**. [S.l.]: ACIS, 2003. p. 370–377.

SCHEID, E. J. et al. INSpIRE: Integrated NFV-baSed Intent Refinement Environment. In: **15th IFIP/IEEE International Symposium on Integrated Network Management (IM 2017)**. [S.l.: s.n.], 2017. Accepted. To be published.

SCHONWALDER, J.; QUITTEK, J. Secure management by delegation within the internet management framework. In: **Integrated Network Management VI. Distributed Management for the Networked Millennium. Proceedings of the Sixth IFIP/IEEE International Symposium on Integrated Network Management.** [S.l.: s.n.], 1999. p. 687–700.

SCHONWALDER, J.; QUITTEK, J.; KAPPLER, C. Building distributed management applications with the ietf script mib. **Selected Areas in Communications, IEEE Journal on**, v. 18, n. 5, p. 702 –714, may 2000. ISSN 0733-8716.

SCHWABE, A.; GUTIÉRREZ, P. A. A.; KARL, H. Composition of sdn applications: Options/challenges for real implementations. In: **Proceedings of the 2016 Applied Networking Research Workshop**. New York, NY, USA: ACM, 2016. (ANRW '16), p. 26–31. ISBN 978-1-4503-4443-2. Available from Internet: <http://doi.acm.org/10.1145/2959424.2959436>.

SCHWARTZ, B. et al. Smart packets for active networks. In: **Open Architectures and Network Programming Proceedings, 1999. OPENARCH '99**. [S.l.: s.n.], 1999. p. 90 –97.

SCOTT-HAYWARD, S.; O'CALLAGHAN, G.; SEZER, S. Sdn security: A survey. In: IEEE. **Future Networks and Services (SDN4FNS), 2013 IEEE SDN For**. [S.l.], 2013. p. 1–7.

SEZER, S. et al. Are we ready for sdn? implementation challenges for software-defined networks. **Communications Magazine, IEEE**, v. 51, n. 7, p. 36–43, July 2013. ISSN 0163-6804.

SHABTAI, A. et al. Google android: A comprehensive security assessment. **IEEE Security Privacy**, v. 8, n. 2, p. 35–44, March 2010. ISSN 1540-7993.

SHEN, S. H. et al. Reliable multicast routing for software-defined networks. In: **2015 IEEE Conference on Computer Communications (INFOCOM)**. [S.l.: s.n.], 2015. p. 181–189. ISSN 0743-166X.

SHERWOOD, R. et al. **Flowvisor: A network virtualization layer**. 2009.

SHIH, M.-W. et al. S-nfv: Securing nfv states by using sgx. In: **Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization**. New York, NY, USA: ACM, 2016. (SDN-NFV Security '16), p. 45–48. ISBN 978-1-4503-4078-6. Available from Internet: <http://doi.acm.org/10.1145/2876019.2876032>.

SON, S. et al. Model checking invariant security properties in openflow. In: **2013 IEEE International Conference on Communications (ICC)**. [S.l.: s.n.], 2013. p. 1974–1979. ISSN 1550-3607.

SONG, H. Protocol-oblivious forwarding: Unleash the power of sdn through a future-proof forwarding plane. In: **Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking**. New York, NY, USA: ACM, 2013. (HotSDN '13), p. 127–132. ISBN 978-1-4503-2178-5. Available from Internet: <http://doi.acm.org/10.1145/2491185.2491190>.

STRAUß, F. **Advantages and Disadvantages of the Script MIB Infrastructure**. [S.l.], 2000. Available at: <https://www.ibr.cs.tu-bs.de/bib/xml/Str00b.html>. Accessed: November, 2016.

TARIQ, M. A. et al. Pleroma: A sdn-based high performance publish/subscribe middleware. In: **Proceedings of the 15th International Middleware Conference**. New York, NY, USA: ACM, 2014. (Middleware '14), p. 217–228. ISBN 978-1-4503-2785-5. Available from Internet: <http://doi.acm.org/10.1145/2663165.2663338>.

TARNARAS, G.; HALEPLIDIS, E.; DENAZIS, S. Sdn and forces based optimal network topology discovery. In: **Network Softwarization (NetSoft), 2015 1st IEEE Conference on**. [S.l.: s.n.], 2015. p. 1–6.

TENNENHOUSE, D.; WETHERALL, D. Towards an active network architecture. In: **DARPA Active Networks Conference and Exposition, 2002. Proceedings**. [S.l.: s.n.], 2002. p. 2–15.

TENNENHOUSE, D. L. et al. A survey of active network research. **IEEE Communications Magazine**, v. 35, n. 1, p. 80–86, Jan 1997. ISSN 0163-6804.

TOOTOONCHIAN, A.; GANJALI, Y. Hyperflow: a distributed control plane for openflow. In: **INM/WREN**. [S.l.: s.n.], 2010.

TOUCH, J. D. et al. **A Virtual Internet Architecture**. [S.l.]: ISI Technical Report: ISI-TR-2003-570, 2003. Available at: <http://www.isi.edu/div7/publication_files/tr-570.pdf>. Accessed: November, 2015.

VALOCCHI, D. et al. Sigma: Signaling framework for decentralized network management applications. **IEEE Transactions on Network and Service Management**, PP, n. 99, p. 1–1, 2017. ISSN 1932-4537.

VANBEVER, L. et al. Hotswap: Correct and efficient controller upgrades for software-defined networks. In: **Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking**. New York, NY, USA: ACM, 2013. (HotSDN '13), p. 133–138. ISBN 978-1-4503-2178-5. Available from Internet: <http://doi.acm.org/10.1145/2491185.2491194>.

VEITCH, P.; MCGRATH, M. J.; BAYON, V. An instrumentation and analytics framework for optimal and robust nfv deployment. **IEEE Communications Magazine**, v. 53, n. 2, p. 126–133, Feb 2015. ISSN 0163-6804.

VOUILLON, J.; COSMO, R. D. Broken sets in software repository evolution. In: **2013 35th International Conference on Software Engineering (ICSE)**. [S.l.: s.n.], 2013. p. 412–421. ISSN 0270-5257.

WANG, A. et al. Network virtualization: Technologies, perspectives, and frontiers. **Journal of Lightwave Technology**, v. 31, n. 4, p. 523–537, Feb 2013. ISSN 0733-8724.

WANG, H. **Towards a programmable dataplane**. Thesis (PhD) — Cornell University, The address of the publisher, 5 2017. Available at: <https://pdfs.semanticscholar.org/2b0a/918d1a30683e47f4918fca09f5781b4a35f3.pdf>. Accessed: April, 2018.

WETHERALL, D.; GUTTAG, J. V.; TENNENHOUSE, D. Ants: a toolkit for building and dynamically deploying network protocols. In: **OPENARCH, IEEE**. [S.l.: s.n.], 1998. p. 117–129.

WETTINGER, J. et al. Streamlining devops automation for cloud applications using {TOSCA} as standardized metamodel. **Future Generation Computer Systems**, v. 56, p. 317 – 332, 2016. ISSN 0167-739X.

WICKBOLDT, J. A. et al. Resource management in iaas cloud platforms made flexible through programmability. **Computer Networks**, v. 68, p. 54 – 70, 2014. ISSN 1389-1286. Communications and Networking in the Cloud. Available from Internet: <http://www.sciencedirect.com/science/article/pii/S138912861400084X>.

WICKBOLDT, J. A. et al. Software-defined networking: management requirements and challenges. **IEEE Communications Magazine**, v. 53, n. 1, p. 278–285, January 2015. ISSN 0163-6804.

WOOD, T. et al. Toward a software-based network: integrating software defined networking and network function virtualization. **Network, IEEE**, v. 29, n. 3, p. 36–41, May 2015. ISSN 0890-8044.

WU, Y. juan et al. Programmable virtual network instantiation in iaas cloud based on sdn. **The Journal of China Universities of Posts and Telecommunications**, v. 20, p. 121 – 125, 2013. ISSN 1005-8885.

XIA, W. et al. A survey on software-defined networking. **IEEE Communications Surveys Tutorials**, v. 17, n. 1, p. 27–51, Firstquarter 2015. ISSN 1553-877X.

XING, T. et al. Sdnips: Enabling software-defined networking based intrusion prevention system in clouds. In: **Network and Service Management (CNSM), 2014 10th International Conference on**. [S.l.: s.n.], 2014. p. 308–311.

YANG, L. et al. **Forwarding and Control Element Separation (ForCES) Framework**. RFC Editor, 2013. RFC 3746. (Request for Comments, 3746). Available from Internet: <https://rfc-editor.org/rfc/rfc3746.txt>.

YANG, L. et al. **Forwarding and Control Element Separation (ForCES) Framework (RFC 3746)**. United States: RFC Editor, 2004.

YANG, L. et al. **Juju Store**. [S.l.]: Canonical Ltd, 2017. Available at: <https://jujucharms.com/>. Accessed: February, 2017.

YANG, L. et al. **$P4_{16}$ Language Specification**. [S.l.]: The P4 Language COnsortium, 2017. Available at: <https://p4.org/p4-spec/docs/P4-16-v1.0.0-spec.html>. Accessed: May, 2018.

YOU, T.; JUNG, H. Exploiting in-network functions enabling sdn for evolving future internet architecture. In: **2014 International Conference on Information and Communication Technology Convergence (ICTC)**. [S.l.: s.n.], 2014. p. 780–784. ISSN 2162-1233.

YU, J. et al. Forwarding programming in protocol-oblivious instruction set. In: **2014 IEEE 22nd International Conference on Network Protocols**. [S.l.: s.n.], 2014. p. 577–582. ISSN 1092-1648.

ZABOLOTNYI, R.; LEITNER, P.; DUSTDAR, S. Dynamic program code distribution in infrastructure-as-a-service clouds. In: **2013 5th International Workshop on Principles of Engineering Service-Oriented Systems (PESOS)**. [S.l.: s.n.], 2013. p. 29–36. ISSN 2156-7921.

ZHANG, Q.; CHENG, L.; BOUTABA, R. Cloud Computing: State-of-the-art and Research Challenges. **Journal of internet services and applications**, Springer, v. 1, n. 1, p. 7–18, 2010.

ZHANG, S. Q. et al. Network function virtualization enabled multicast routing on sdn. In: **IEEE ICC 2015**. [S.l.: s.n.], 2015. p. 5595–5601.

ZHOU, F. et al. Design and implementation of stateful packet inspection firewall based on forces architecture. In: **2010 International Conference on Information, Networking and Automation (ICINA)**. [S.l.: s.n.], 2010. v. 1, p. V1–120–V1–124. ISSN 2162-5476.

ZILBERMAN, N. et al. Netfpga sume: Toward 100 gbps as research commodity. **IEEE Micro**, v. 34, n. 5, p. 32–41, Sept 2014. ISSN 0272-1732.

APPENDIX A SCIENTIFIC PRODUCTION

The research work presented in this thesis was reported to the scientific community through paper submissions to renowned conferences and journals. The process of doing research, submitting paper, gathering feedback, and improving the work helped to achieve the maturity hereby presented.

A.1 Papers: Accepted and Under Reviewing

The following list shows the main papers related to this thesis.

1. Ricardo Luis dos Santos, Oscar Mauricio Caicedo Rendón, Juliano Araújo Wickboldt, and Lisandro Zambenedetti Granville. **App2net: A platform to transfer and configure applications on programmable virtual networks**. 2015 IEEE Symposium on Computers and Communication (ISCC), July 06 - July 09, 2015, Larnaca, Cyprus.

    - **Title:** *App2net: A Platform to Transfer and Configure Applications on Programmable Virtual Networks*.
    - **Contribution:** System architecture to install and configure network applications over heterogeneous programmable virtual networks.
    - **Abstract:** In programmable virtual networks, simple tasks, like installing software, can be extremely complex. This complexity occurs mainly because the code transfer and initial functional settings in network execution environments are not automated. In addition, the same tasks have different requirements in each service lifecycle stage. In this sense, we propose the App2net platform for enabling the transfer and configuration of network applications in programmable virtual networks that use heterogeneous execution environments. We also propose a taxonomy for grouping code transfer techniques and, based on such techniques, we develop models for code transfer. A prototype has been implemented and tested on realistic network topologies commonly found on the Internet. Results allow us to identify which models improve code transfer consuming fewer resources, regarding service lifecycle stages and network topologies.

- **Status:** Published.
- **Qualis:** A2.
- **Conference:** Twentieth IEEE Symposium on Computers and Communications.
- **Date:** July 06 - July 09, 2015.
- **Local:** Larnaca, Cyprus.
- **URL:** <http://iscc2015.ieee-iscc.org/>.
- **Digital Object Identifier (DOI):** <http://dx.doi.org/10.1109/ISCC.2015.7405534>.

2. Ricardo Luis dos Santos, Muriel Figueredo Franco, Eder John Scheid, Ricardo José Pfitscher, Lisandro Zambenedetti Granville, and Liane Margarida Rockenbach Tarouco. **iMPROVE: Enhancing the Introduction of Services on Programmable Virtual Networks**. 31st IEEE International Conference on Advanced Information Networking and Applications (AINA 2017), March 27 - March 29, 2017, Taipei, Taiwan.

   - **Title:** *iMPROVE: Enhancing the Introduction of Services on Programmable Virtual Networks*.
   - **Contribution:** Integrating the successful paradigm of OS-specific app stores to enhance the distribution and description of virtual network appliances and network applications.
   - **Abstract:** Programmable Virtual Networks (PVNs) make the network more flexible and allow the fast introduction of new services. However, several shortcomings hamper their wider adoption, including: (*i*) the extensive knowledge required to configure and manage the NetApps; (*ii*) the lack of descriptors to detail all nuances of the NetApps; and (*iii*) there is no solution that enables to distribute and configure NetApps over distinct technologies. Therefore, we propose `iMPROVE` to simplify the introduction of services in PVNs, enhancing the distribution of NetApps. We also extend the ETSI network service descriptor to support distinct technologies as well as to represent conflict issues. We demonstrate evidence of `iMPROVE`'s feasibility in a case study and compare it with the main solutions for distributing and deploying applications over multiple platforms.

- **Status:** Published.
- **Qualis:** A2.
- **Conference:** 31st IEEE International Conference on Advanced Information Networking and Applications (AINA 2017).
- **Date:** March 27 - March 29, 2017.
- **Local:** Taipei, Taiwan.
- **URL:** <http://voyager.ce.fit.ac.jp/conf/aina/2017/>.
- **Digital Object Identifier (DOI):** <http://dx.doi.org/10.1109/AINA.2017.113>.

3. Ricardo Luis dos Santos, Muriel Figueredo Franco, Eder John Scheid, Ricardo José Pfitscher, Alberto Egon Schaeffer-Filho, Lisandro Zambenedetti Granville, and Liane Margarida Rockenbach Tarouco. **Network Marketplaces: The Current Landscape, Design Goals, and Research Challenges**.

   - **Title:** *Network Marketplaces: The Current Landscape, Design Goals, and Research Challenges*.
   - **Contribution:** A review of the historical roadmap of networking paradigms and marketplaces to assess how paradigms and technologies have evolved over the years. In addition, this review includes the current landscape regarding marketplaces for different paradigms, essential design goals, and the significant challenges that must be overcome to make the adoption of marketplaces in future networks a reality.
   - **Status:** To be submitted to a journal.

4. Ricardo Luis dos Santos, Muriel Figueredo Franco, Eder John Scheid, Ricardo José Pfitscher, Lisandro Zambenedetti Granville, and Liane Margarida Rockenbach Tarouco. **A Survey on Programmable Virtual Networks: Where Did We Come From, and Where Are We Going To?**.

   - **Title:** *A Survey on Programmable Virtual Networks: Where Did We Come From, and Where Are We Going To?*
   - **Contribution:** A review of the key technologies with regard to virtualization and network programmability. Besides, a comparison between the essential aspects of these technologies that are required to deploy and manage network services.

- **Status:** To be submitted to a journal.

There is another publication that, although not directly related to this thesis, was developed during the Ph.D. time.

1. Ricardo Luis dos Santos, Juliano Araújo Wickboldt, Roben Castagna Lunardi, Bruno Lopes Dalmazo, Lisandro Zambenedetti Granville, Luciano Paschoal Gaspary. **Identifying the Root Cause of Failures in IT Changes: Novel Strategies and Trade-offs**. IFIP/IEEE International Symposium on Integrated Network Management, 2013, Ghent, Belgium. pp 118-125. ISSN 1573-0077.

   - **Status:** Published.
   - **Qualis:** A2.

A.2 Collaborations: Accepted and Under Reviewing

The following list shows the main collaborations that, although not directly related to this thesis, are linked to the design of network management solutions.

1. Wanderson Paim de Jesus, Ricardo Luis dos Santos, Oscar Mauricio Caicedo Rendón, Lisandro Zambenedetti Granville. **A Platform for Programmable Virtual Network Management**. The 31st Brazilian Symposium on Computer Networks and Distributed Systems (SBRC 2013), May 6 - May 10, 2013, Brasília, Brazil.

   - **Status:** Published.
   - **Qualis:** B2.

2. Muriel Figueredo Franco, Ricardo Luis dos Santos, Alberto Egon Schaeffer-Filho, Lisandro Zambenedetti Granville. **VISION - Interactive and Selective Visualization for Management of NFV-Enabled Networks**. The 30th IEEE International Conference on Advanced Information Networking and Applications (AINA 2016), pp. 274 - 281, March 23 - March 25, 2016, Le Régent Congress Centre, Crans-Montana, Switzerland.

   - **Status:** Published.
   - **Qualis:** A2.

- **Digital Object Identifier (DOI):** <http://dx.doi.org/10.1109/aina.2016.26>.

3. Ricardo José Pfitscher, Eder John Scheid, Ricardo Luis dos Santos, Rafael Obelheiro, Mauricio Pillon, Alberto Egon Schaeffer-Filho, Lisandro Zambenedetti Granville. **DReAM - A Distributed Result-Aware Monitor for Network Functions Virtualization**. The 21st IEEE Symposium on Computers and Communication (ISCC 2016), June 27 - June 30, 2016, University of Messina, Messina, Italy.

   - **Status:** Published.
   - **Qualis:** A2.
   - **Digital Object Identifier (DOI):** <http://dx.doi.org/10.1109/iscc.2016.7543813>.

4. Eder John Scheid, Cristian Cleder Machado, Ricardo Luis dos Santos, Alberto Egon Schaeffer-Filho, Lisandro Zambenedetti Granville. **Policy-Based Dynamic Service Chaining in Network Functions Virtualization**. The 21st IEEE Symposium on Computers and Communication (ISCC 2016), June 27 - June 30, 2016, University of Messina, Messina, Italy.

   - **Status:** Published.
   - **Qualis:** A2.
   - **Digital Object Identifier (DOI):** <http://dx.doi.org/10.1109/iscc.2016.7543763>.

5. Eder John Scheid, Cristian Cleder Machado, Muriel Figueredo Franco, Ricardo Luis dos Santos, Ricardo José Pfitscher, Alberto Egon Schaeffer-Filho, Lisandro Zambenedetti Granville. **INSpIRE: Integrated NFV-based Intent Refinement Environment**. The 15th IFIP/IEEE International Symposium on Integrated Network Management (IM 2017), May 8 - May 12, 2017, Lisbon, Portugal.

   - **Status:** Published.
   - **Qualis:** A2.
   - **Digital Object Identifier (DOI):** <https://doi.org/10.23919/INM.2017.7987279>.

6. Arthur Selle Jacobs, Ricardo Luis dos Santos, Muriel Figueredo Franco, Eder John Scheid, Ricardo José Pfitscher, Lisandro Zambenedetti Granville. **Affinity measurement for NFV-enabled networks: A criteria-based approach**. The 15th IFIP/IEEE International Symposium on Integrated Network Management (IM 2017), May 8 - May 12, 2017, Lisbon, Portugal.

- **Status:** Published.
- **Qualis:** A2.
- **Digital Object Identifier (DOI):** <https://doi.org/10.23919/INM.2017.7987272>.

7. Muriel Figueredo Franco, Ricardo Luis dos Santos, Ricardo Cava, Eder John Scheid, Ricardo José Pfitscher, Carla Dal Sasso Freitas, Lisandro Zambenedetti Granville. **Interactive Visualizations for Planning and Strategic Business Decisions in NFV-Enabled Networks**. The 31st IEEE International Conference on Advanced Information Networking and Applications (AINA 2017), March 27 - March 29, 2017, Taipei, Taiwan.

   - **Status:** Published.
   - **Qualis:** A2.
   - **Digital Object Identifier (DOI):** <https://doi.org/10.1109/AINA.2017.103>.

8. Arthur Selle Jacobs, Ricardo José Pfitscher, Ricardo Luis dos Santos, Muriel Figueredo Franco, Eder John Scheid, Lisandro Zambenedetti Granville. **Artificial neural network model to predict affinity for virtual network functions**. The 2018 IEEE/IFIP Network Operations and Management Symposium (NOMS 2018), April 23 - April 27, 2018, Taipei, Taiwan.

   - **Status:** Published.
   - **Qualis:** A2.
   - **Digital Object Identifier (DOI):** <https://doi.org/10.1109/NOMS.2018.8406253>.

9. Ricardo José Pfitscher, Arthur Selle Jacobs, Eder John Scheid, Muriel Figueredo Franco, Ricardo Luis dos Santos, Alberto Egon Schaeffer-Filho, Lisandro Zambenedetti Granville. **A model for quantifying performance degradation in virtual network function service chains**. The 2018 IEEE/IFIP Network Operations and Management Symposium (NOMS 2018), April 23 - April 27, 2018, Taipei, Taiwan.

   - **Status:** Published.
   - **Qualis:** A2.
   - **Digital Object Identifier (DOI):** <https://doi.org/10.1109/NOMS.2018.8406268>.

10. Lucas Bondan, Muriel Figueredo Franco, Leonardo da Cruz Marcuzzo, Giovanni Venâncio, Ricardo Luis dos Santos, Ricardo José Pfitscher, Eder John Scheid, Burkhard Stiller, Filip De Turck, Elias P. Duarte Jr., Alberto Egon Schaeffer-Filho, Carlos Raniery Paula dos Santos, Lisandro Zambenedetti Granville. **FENDE: Marketplace-based Distribution, Execution, and Lifecycle Management of VNFs**. *This paper was submitted to "Network & Service Management Series" of the IEEE Communications Magazine*.

    - **Status:** *Submitted*.
    - **Qualis:** A1.