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

EDER JOHN SCHEID

INSpIRE: an Integrated NFV-baSed Intent Refinement Environment

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

Advisor: Prof. Dr. Lisandro Zambenedetti
Granville

Porto Alegre
May 2017

CIP – CATALOGING-IN-PUBLICATION

Scheid, Eder John

INSpIRE: an Integrated NFV-baSed
Intent Refinement Environment / Eder John Scheid. – Porto Alegre: PPGC da UFRGS, 2017.

83 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2017. Advisor: Lisandro Zambenedetti Granville.

1. Policy-based management. 2. Policy refinement. 3. Software-defined networking. 4. Network functions virtualization. 5. Intent-based Networking. I. Granville, Lisandro Zambenedetti. II. Título.

ACKNOWLEDGMENTS

Only in Brazilian Portuguese. See page 4.

AGRADECIMENTOS

Em primeiro lugar, gostaria de agradecer à minha família, a qual é a base de tudo. Ao meu pai e minha mãe, que são exemplo a serem seguidos de integridade, perseverança, honestidade, trabalho duro e por me ensinarem que o estudo é uma fundação essencial no crescimento do indivíduo. Ao meu irmão, companheiro desde sempre, exemplo de dedicação e que sempre torceu por mim e pelo Internacional :). A minha namorada, que esteve presente em todos os momentos, bons ou ruins e sempre acreditou em mim. Sem vocês nunca teria chegado até aqui. Obrigado!

Gostaria de também agradecer ao meu orientador, Lisandro, que durante todo o decorrer de meu mestrado contribuiu com diversos elementos essenciais para a conclusão desta dissertação, oportunidades e pela valiosa orientação tanto acadêmica e pessoal. Destaco as reuniões e encontros que foram fonte de esclarecimento para problemas de pesquisa e para indicar a direção na qual seguir durante toda minha formação. Meus agradecimentos ao corpo docente e técnicos administrativos da Universidade Federal do Rio Grande do Sul (UFRGS), em especial aos professores das disciplinas que cursei durante o mestrado, obrigado pelos ensinamentos.

Estendo meus agradecimentos à todo o grupo de Redes de Computadores do Instituto de Informática da UFRGS, grupo que mais que colegas, se tornaram amigos. Expresso minha gratidão ao Lab. 210 pela fiel parceria, amizade, e churrascos durante estes dois anos, em especial ao Cristian Machado (Batman), Muriel Franco (Mumu), Ricardo Pfitscher (Tocaio) e Ricardo Santos (Ricardão) pelas múltiplas contribuições nos artigos e nesta dissertação. Valeu, gurizada!

Uma menção à todos meus amigos que estão do meu lado até hoje, desde o ensino fundamental, ensino médio e graduação. Espero que toda a amizade que temos continue se perpetuando.

Por fim, obrigado à todos as pessoas, departamentos e órgãos federais que contribuíram de alguma maneira, direta ou indireta, para minha formação.

"Desire to know why, and how, curiosity; such as is in no living creature but man: so that man is distinguished, not only by his reason, but also by this singular passion from other animals; in whom the appetite of food, and other pleasures of sense, by predominance, take away the care of knowing causes; which is a lust of the mind, that by a perseverance of delight in the continual and indefatigable generation of knowledge, exceedeth the short vehemence of any carnal pleasure".

— THOMAS HOBBES - LEVIATHAN.

ABSTRACT

Many aspects of the management of computer networks, such as quality of service and security, must be taken into consideration to ensure that the network meets the users and clients demands. Fortunately, management solutions were developed to address these aspects, such as Intent-Based Networking (IBN). IBN is a novel networking paradigm that abstracts network configurations by allowing administrators to specify how the network should behave and not what it should do. In this dissertation, we introduce an IBN solution called `INSpIRE` (Integrated NFV-based Intent Refinement Environment). `INSpIRE` implements a refinement technique to translate intents into a set of configurations to perform a desired service chain in both homogeneous environments (virtualized functions only) and heterogeneous environments (virtualized functions and physical middleboxes). This refinement technique relies on Non-Functional Requirements (NFRs) and clustering to determine the network functions that will compose the service chain. Our solution is capable of (*i*) determining the specific functions required to fulfill an intent, (*ii*) chaining these functions according to their dependencies, and (*iii*) presenting enough low-level information to network devices for posterior traffic steering. Furthermore, to assess the feasibility of our solution we detail case studies that reflects real-world management situations and evaluate the scalability of the refinement process. Finally, the results showed that `INSpIRE` is capable of delivering a service chain that meets the requirements specified in the *intent* in small and large scenarios.

Keywords: Policy-based management. Policy refinement. Software-defined networking. Network functions virtualization. Intent-based Networking.

INSpIRE - Ambiente Integrado de Refinamento de *Intenções* Baseado em NFV

RESUMO

Muitos aspectos da gestão de redes de computadores, como a Qualidade de Serviço (QoS) e segurança, devem ser levados em consideração para garantir que a rede atenda às exigências de usuários e clientes. Felizmente, soluções de gestão de rede foram desenvolvidas para lidar com estes aspectos, tais como Redes Baseadas em Intenção (*Intent-based Networking* - IBN). IBN é um novo paradigma de rede que abstrai configurações de rede, permitindo que administradores especifiquem como a rede deve se comportar e não o que ele deve fazer. Nesta dissertação, apresentamos uma solução de IBN chamada `INSpIRE` (Integrated NFV-based Intent Refinement Envirorment). `INSpIRE` implementa uma técnica de refinamento para traduzir *intenções* em um conjunto de configurações para executar uma desejada cadeia de serviço em ambos, ambientes homogêneos (somente funções virtualizadas) e ambientes heterogêneos (funções virtualizadas e *middleboxes* físicas). A técnica de refinamento baseia-se em Requisitos Não Funcionais (*Non-Functional Requirements* - NFRs) e *clustering* para determinar quais funções de rede deverão compor a cadeia de serviços. Nossa solução é capaz de (*i*) determinar as funções específicas necessárias para o cumprimento de uma *intenção*, (*ii*) encadear estas funções de acordo com suas dependências e (*iii*) apresentar informações de baixo nível suficientes para que dispositivos de rede possam posteriormente orientar o tráfego de rede por essa cadeia de serviço. Além disso, para avaliar a viabilidade da nossa solução, estudos de caso no qual refletem situações de gestão do mundo real e uma avaliação da escalabilidade do processo de refinamento são detalhados. Por fim, os resultados mostraram que `INSpIRE` é capaz de fornecer uma cadeia de serviços que atende aos requisitos especificados na *intenção* em cenários pequenos e grandes.

Palavras-chave: Gerenciamento de Rede Baseado em Políticas. Refinamento de Políticas. Redes Definidas por Sofware. Virtualização de Funções de Rede. Redes Baseadas em Intenções.

LIST OF FIGURES

2.1 SDN OpenFlow Architecture 18
2.2 NFV-enabled Infrastructure 20
2.3 Synergy between NFV and SDN with Service Chaining 21
2.4 Static and Dynamic Service Chaining Examples 22
2.5 The IETF Policy-Based Management Architecture 22
2.6 Diagram of the Types of Policies 23
2.7 ECA Policy and *Intent* Example 24

4.1 `INSpIRE` Architecture and Main Components 31
4.2 Valid *Intent* Parsing Example 34
4.3 `INSpIRE` Index Page 35
4.4 Intent Authoring Screen 36
4.5 `INSpIRE` Web-Interface Views 37

5.1 SIG Example 40
5.2 Pre-defined SIG for Middlebox Security 41
5.3 INSpIRE Intent Refinement Flow 45
5.4 VNF Clustering Example 46

6.1 Modeled SIG for Middlebox Privacy 50
6.2 Example of a SIG in YAML Format 51
6.3 Operationalizations Tree View 52
6.4 Time to discover 3 clusters in different scenarios 58
6.5 Time to calculate the scores in different SIGs 58

LIST OF TABLES

4.1 Regular Expressions Examples 33
4.2 Exposed `INSpIRE` REST Functions 36

5.1 Quantitative Contribuitions from Affleck and Krishna (AFFLECK; KRISHNA, 2012) 42

6.1 Impact of *Operationalizations* towards Leaf-*Softgoals* 49
6.2 User Domains and Respective IP Ranges 54
6.3 Results Obtained for the Case Study 1 54
6.4 Results Obtained for the Case Study 2 55
6.5 Results Obtained for the Case Study 3 56

LIST OF ABBREVIATIONS AND ACRONYMS

BSS	*Business Support System*
CAPEX	*CApital EXpenditure*
CNL	*Controlled Natural Language*
COTS	*Commerical-Of-The-Shelf Server*
CPU	*Central Processing Unit*
ECA	*Event-Condition-Action*
ETSI	*European Telecommunications Standards Institute*
FTP	*File Transfer Protocol*
GiB	*Gigabyte*
GUI	*Graphical User Interface*
HTTP	*Hypertext Transfer Protocol*
IBN	*Intent-Based Networking*
IDPS	*Intrusion Detection and Prevention System*
IDS	*Intrusion Detection System*
IETF	*Internet Engineering Task Force*
IMAP	*Internet Message Access Protocol*
IP	*Internet Protocol*
IPS	*Intrusion Prevention System*
ISG	*Industry Specification Groups*
ISP	*Internet Service Providers*
JSON	*JavaScript Object Notation*
KAOS	*Knowledge Acquisition in autOmated Specification*
MANO	*Management and Orchestration*
MD5	*Message-Digest algorithm 5*
MVC	*Model View Controller*
NAT	*Network Address Translation*

NB	*North Bound*
NFR	*Non-Functional Requirement*
NFV	*Network Functions Virtualization*
NS	*Network Service*
OPEX	*OPerational EXpenditure*
OSS	*Operational Support System*
PBNM	*Policy-Based Network Management*
PDP	*Policy Decision Point*
PEP	*Policy Enforcement Point*
PIN	*Personal Identification Number*
QoS	*Quality of Service*
RAM	*Random Access Memory*
RFC	*Request For Comments*
SCaaS	*Service Chain as a Service*
SDN	*Software-Defined Networking*
SFC	*Service Function Chaining*
SHA	*Secure Hash Algorithm*
SLA	*Service Level Agreement*
SMTP	*Simple Mail Transfer Protocol*
SNMP	*Simple Network Management Protocol*
SSL	*Secure Socket Layer*
SSH	*Secure Shell*
SIG	*Softgoal Interdependency Graph*
VIM	*Virtualized Infrastructure Manager*
VM	*Virtual Machine*
VNF	*Virtual Network Function*
VNF	*Virtual Network Function Descriptor*
VoIP	*Voice over Internet Protocol*
XML	*eXtensible Markup Language*

YAML *YAML Ain’t Markup Language*

CONTENTS

1 INTRODUCTION 15

2 BACKGROUND 18
2.1 Software-Defined Networks (SDN) 18
2.2 Network Functions Virtualization (NFV) 19
2.3 Traffic Steering and Service Chaining 21
2.4 Policy-Based Network Management (PBNM) 22
2.4.1 Policy Refinement 23
2.5 Intent-based Networking (IBN) 24

3 RELATED WORK 26
3.1 Network Functions Virtualization 26
3.2 SDN-based Traffic Steering and Service Chaining 26
3.3 Policy-Based Network Management 27
3.3.1 Policy Refinement 28
3.4 Intent-Based Networking 29
3.5 Discussion of Related Work 29

4 INTEGRATED NFV-BASED INTENT REFINEMENT ENVIRONMENT 30
4.1 Integrated NFV-baSed Intent Refinement Environment Overview 30
4.1.1 Main Components 30
4.2 Controlled Natural Language (CNL) 32
4.3 Intent Translation 33
4.3.1 *Intent* Validation 33
4.3.2 Conflict Detection 33
4.4 `INSpIRE` Prototype GUI Implementation 34

5 REFINEMENT TECHNIQUE 38
5.1 Non-Functional Requirements Framework 38
5.2 Softgoal Interdependency Graphs (SIGs) 39
5.3 Quantitative Calculation of NFR 40
5.4 Intent Refinement 44
5.4.1 Clustering 44
5.4.2 VNF Selection 46
5.4.3 VNF Dependency Ordering 47

6 PROTOTYPE AND EVALUATION 49
6.1 INSpIRE Prototype Case Study 49
6.1.1 SIG Modeling 49
6.1.2 VNF Insertion 52
6.1.3 *Intent* Authoring 53
6.2 Intent Refinenemt Case Studies 53
6.2.1 Case Study 1 - Generic Academic Network 53
6.2.2 Case Study 2 - Common Company Network 55
6.2.3 Case Study 3 - VNF Service Chain as a Service (VNF-SCaaS) 56
6.3 Evaluation 57
6.3.1 Clustering Evaluation 57
6.3.2 Score Calculation Evaluation 57

7 CONCLUDING REMARKS 60
7.1 Summary of Contributions 60
7.2 Final Remarks and Future Work 61

REFERENCES 63

APPENDIXA PUBLISHED PAPER – ISCC 2016 67

APPENDIXB PUBLISHED PAPER – IM 2017 74

1 INTRODUCTION

Many aspects of computer networks management, such as Quality of Service (QoS) and security, must be taken into account to ensure that the network meets the requirements of the users and clients. The devices that form the network must be individually configured to achieve a satisfactory performance, and this requires time and effort on the part of the network administrators (VERMA, 2009). This activity generally leads to an interruption of the network and results in rules that depend, on a great extend, on the physical network topology. Fortunately, many approaches have been adopted to tackle these issues, including well-known and widely employed solutions such as Simple Network Management Protocol (SNMP) (CASE et al., 1990) and Policy-Based Network Management (PBNM) (STRASSNER, 2003) (VERMA, 2002), as well as more recent techniques based on Intent-Based Networking (IBN) (BEHRINGER et al., 2015).

IBN is a novel networking paradigm that abstracts network configurations by allowing administrators to specify how the network should behave and not what it should do. For example, in IBN solutions, one must write an *intent* (*e.g.* "All outgoing network traffic is encrypted and secure") and not an instruction (*e.g.* "If a packet's destination IP is not in the company's subnet IP range, then encrypt it using the $f()$ function with the SHA3 parameter"). *Intent* introduces a context and is not vendor-specific, which means that the underlying mechanisms must be capable of translating it to low-level configurations and maintaining the desired state throughout the entire network operation. Given the dynamicity involved in IBN, the underlying technologies must be flexible enough to cope with an ever-changing network environment, by scaling and moving accordingly. Novel technologies have arisen as alternatives to provide this flexibility, and include Network Functions Virtualization (NFV) (ETSI, 2012), and Software-Defined Networking (SDN) (FEAMSTER; REXFORD; ZEGURA, 2014).

Network functions, such as load balancing, firewalls, and Intrusion Detection Systems (IDSes) are traditionally implemented in physical devices (often referred to as middleboxes). Middleboxes tend to be proprietary and vendor-specific, and thus force network operators to learn about their peculiarities from different vendors, which is counter-productive. Moreover, physical middleboxes are not flexible enough to accommodate bursts of demand, which intrinsically hinders their scalability. NFV is a novel technology that addresses the lack of flexibility of physical middleboxes and makes use of Commercial Off-The-Shelf (COTS) hardware to host virtualized network services. By adopting this approach, the CApital EXpenditure (CAPEX) and OPerational EXpenditure (OPEX) can be significantly reduced. In addition, with NFV, service provisioning can be easily scaled up or down depending on the requirements of the network. These benefits are driving the application of innovative network functions and accelerating the adoption of NFV by large companies.

NFV allows the chaining of multiple Virtual Network Functions (VNFs). This VNF chaining enables network operators to decide which sequence of VNFs a packet should undergo. The

act of specifying the sequence of VNFs is called *Service Function Chaining* (SFC) (QUINN; NADEAU, 2015). Service chaining in current network infrastructures is statically defined and dependent on the network's topology. This imposes a challenge to the operator when adding or removing services, since earlier technologies are difficult to redeploy (JOHN et al., 2013). When NFV is allied with SDN, this chaining can be performed dynamically. SDN decouples the control plane from the data plane, and provides *(a)* a global view of the network and *(b)* a controller that makes decisions about traffic forwarding (FEAMSTER; REXFORD; ZEGURA, 2014). As a result of this separation, a controller can be implemented to steer the traffic dynamically during runtime. This means that service chaining can be easily adapted to the administrator's needs. The chaining is created from existing VNFs and middleboxes, and thus ensures that network resources are used efficiently (BLENDIN et al., 2014). However, the management of network functions, service chains, and other network resources becomes a challenging task as the dynamicity of the network increases. Thus, the employment of *intents* and IBN is appropriate in the context of service chaining. Moreover, *intents* can be used to decouple management strategies from implementation details, and this way reduce the amount of specific knowledge from system-level administrators when configuring low-level settings, *e.g.* the chaining of VNFs.

Even though IBN is a novel networking paradigm, *intents* can still be regarded as a system of high-level abstract policies. In addition, they do not involve specific requirements or configurations, *e.g.* OpenFlow rules (MCKEOWN et al., 2008). Thus, IBN solutions must be able to translate these high-level policies into specific lower-level configurations, *e.g.* IPTables rules or routing tables. This translation process is referred to as *"policy refinement"* and has been investigated for several years within the context of PBNM (MOFFETT; SLOMAN, 1993) (CRAVEN et al., 2011). However, to the best of our knowledge, refinement techniques employed alongside IBN and NFV, have not been exploited in any other solution, which means that there is an opportunity to investigate refinement techniques in the context of IBN. Moreover, network administrators have different needs depending on the network traffic that they manage and different customers and end-users of the network do not necessarily have the same security concerns. For example, SSH packets exchanged inside a company's network can be allowed to pass through a less secure firewall, instead of passing through a more sophisticated firewall and a DPI. However, these premises give rise to an important question: how can these security concerns can be specified with regard to service chaining? One approach to solving this problem is the use of IBN and policy refinement to assist the administrator in translating these concerns into service chains in a straightforward, and uncomplicated manner.

In this dissertation, we introduce an IBN solution called `INSpIRE` (Integrated NFV-based Intent Refinement Environment). `INSpIRE` implements a refinement technique to translate *intents* , constrained by a Controlled Natural Language - CNL (SCHEID et al., 2016), into a set of configurations to provide the required service chain in both homogeneous environments (VNFs

only) and heterogeneous environments (VNFs, physical network functions, and middleboxes)[1]. Our solution involves a refinement process that is based on non-functional requirements and *softgoals*, to decompose the *intents* and calculate the values that are relied on as selection criteria for the choice of the middleboxes that will compose the service chain – ultimately, by satisfying the desired *intent*. `INSpIRE` is capable of (*i*) determining the specific VNFs required to fulfill an *intent*, (*ii*) pre-chaining these VNFs according to their dependencies, and (*iii*) providing enough low-level information to the network devices for posterior traffic steering.

Moreover, a prototype has been implemented to provide an evaluation of the feasibility of our solution. The prototype contains a *Graphical User Interface (GUI)* and a computing module. In the GUI, a network operator can write *intents*, manage the available VNFs in its infrastructure, and also configure all the variables and constraints involved in the refinement process. The compute module is responsible for carrying out all the heavy-duty tasks, such as calculating the values that make up the selection criteria and are used for cluster analysis. However, our prototype does not tackle the question of the traffic steering process to provide a more consistent evaluation of the refinement procedure, as this can be undertaken by external elements. Finally, we describe three case studies and conduct experiments to demonstrate the scalability of the refinement features and evaluate our solution in real-world scenarios.

The remainder of this dissertation is structured as follows. Chapter 2 provides a brief overview of the main concepts employed in our solution. Next, in Chapter 3 we discuss the related work of the area. `INSpIRE` is examined in detail, in Chapter 4, together with the elements, techniques, and prototype. In Chapter 5 we provide a formal definition and representation of *softgoals* and the quantitative calculation of the softgoals' values. There is a prototype case study, together with case studies, and experiments in Chapter 6. Finally, in Chapter 7 we conclude this dissertation with some final remarks and make suggestions for future work.

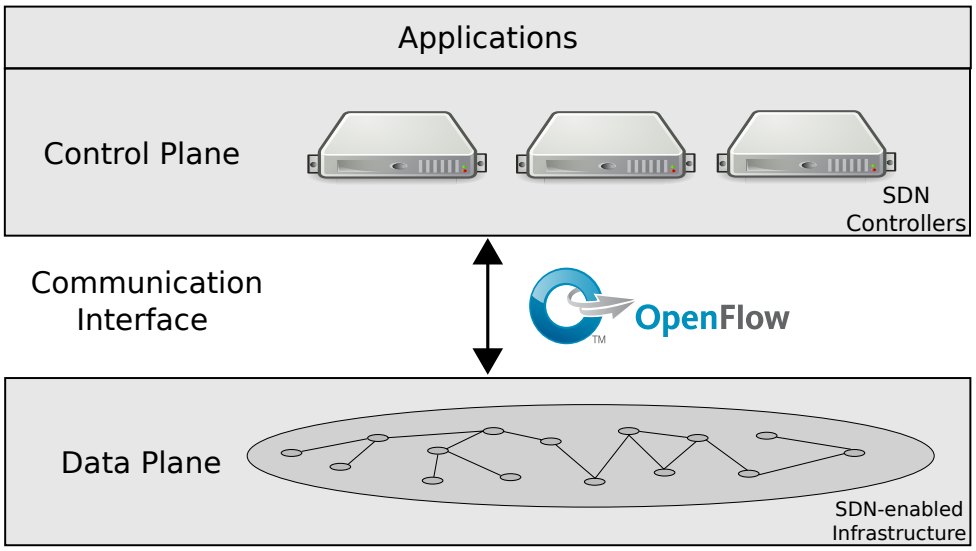
[1] We use these terms (VNFs, network functions, and middleboxes) interchangeably in this dissertation.

2 BACKGROUND

This chapter provides an overview of the main concepts, standards, and technologies involved in INSpIRE. First, there is a description of SDN in Section 2.1. Then, we set out NFV and related factors in Section 2.2. Following this, the concepts of Traffic Steering and Service Chaining are defined in Section 2.3. PBNM and the refinement of policies are described in Section 2.4 and Section 2.4.1, respectively. Finally, IBN is outlined in Section 2.5.

2.1 Software-Defined Networks (SDN)

Figure 2.1: SDN OpenFlow Architecture



Source: the Author, 2017

In traditional networks, the control plane and the data plane are united in a single network element (*e.g.* a switch). The control plane runs in every single network element in the network; this means that the forwarding decisions are processed locally, and the same element forwards the packet on the basis of a local decision. Moreover, many vendors develop their network elements and sell these solutions as black boxes. In other words, the implementation, protocols, and technology of these solutions are proprietary. As a result, the management of the different types of network elements that can be found in traditional networks, is a challenging task for administrators.

Software Defined Networks (SDN) is a new networking paradigm that was introduced a few years ago (FEAMSTER; REXFORD; ZEGURA, 2014). In its early stages, the attempt to create a new networking paradigm originated in universities. However, as a result of contributions being made from companies over the years, SDN has become an attractive alternative to traditional networks. What attracted companies to SDN was the main concept underlying it,

which is to provide a more programmable network. This makes it possible to find innovative network solutions. In addition, SDN provides a centralized (in the case of a single controlling entity) and global view of the network as well. Having a global view of the network enables companies to rapidly develop novel and personalized network management solutions, and thus increase the flexibility of the network and reduce capital and operational expenses.

The cornerstone of SDN is the clear separation of the data plane from the control plane, which is shown in Figure 2.1. The control plane (SDN Controllers) only handles traffic in the network while the data plane (SDN-enabled Infrastructure) only forwards the packets in compliance with the decisions made by the control plane. Before the controllers can make forwarding decisions, they have to possess knowledge about the infrastructure. For this reason, every switch in the network must send control messages to the controllers containing information about features, such as connected links, ports, and incoming packets. This communication between the control plane and the data plane is carried out by novel protocols, *e.g.* OpenFlow (MCKEOWN et al., 2008).

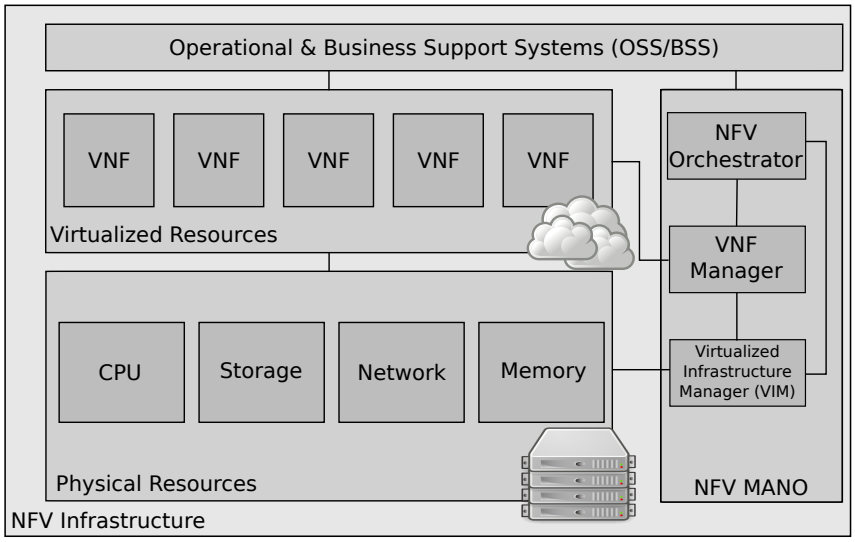
OpenFlow was designed to be the standard protocol employed by SDN-enabled infrastructures. It also provides a secure communication channel between the controllers and OpenFlow-capable switches. By means of OpenFlow, developers can create controllers that are able to communicate with different types of OpenFlow switches from different vendors, and thus assist in integrating the SDN solutions.

2.2 Network Functions Virtualization (NFV)

Network Functions Virtualization (NFV) exploits virtualization techniques to address the separation of the middlebox hardware and software. NFV is designed to use general purpose x86 servers to host Virtual Machines (VM) which contain Virtualized Network Functions (VNFs). The operations carried out by these VNFs vary from DPI to video streaming and caching. In NFV infrastructures (Figure 2.2), the hardware particularities (*e.g.* storage technology, CPU architecture, and memory size) are abstracted, and VNFs can be instantiated and deployed dynamically in the infrastructure, while leveraging the virtualization of physical resources. This means that NFV decouples network functions from the underlying vendor-specific hardware, and can enable the VNF software to evolve separately from the hardware and the other way round as well. The benefits offered by NFV allow network operators to create innovative services, reduce CAPEX and OPEX, and rapidly deliver new services to end-users.

The concept of NFV was first introduced by the European Telecommunications Standards Institute (ETSI), which formed the NFV Industry Specification Group (ISG). This committee is concerned with specifying requirements, architectures, and constraints for the successful adoption of NFV by industry. In 2012, the ETSI NFV ISG published a white paper (ETSI, 2012), that formally set out a definition of NFV, together with its benefits, and challenges. Another outcome of this group was the ETSI NFV Management and Orchestration (MANO) (European

Figure 2.2: NFV-enabled Infrastructure



Source: the Author, 2017

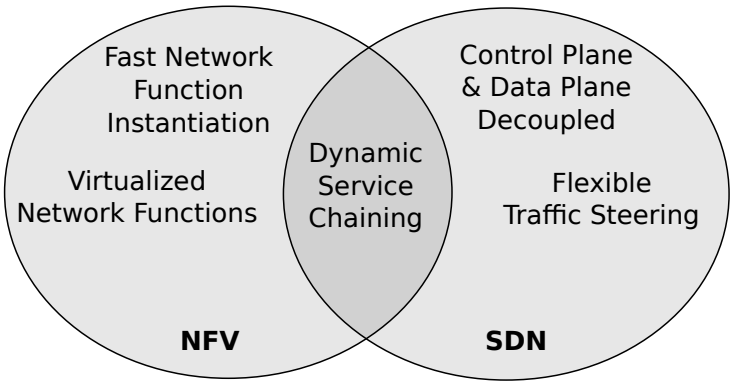
Telecommunications Standards Institute (ETSI), 2014) specification, which has established a framework for the provisioning of VNFs, and all the necessary operations for the proper configuration of VNFs in infrastructures. The main objective of MANO is to meet the particular requirements of the management and orchestration of NFV, and to provide guidance on novel requirements that may appear in this context. Figure 2.2 shows the relationship between the NFV MANO and the different components of an NFV-enabled infrastructure.

The three functional blocks of the NFV MANO are described below:

- **NFV Orchestrator**: This block is responsible for the global management of resources, such as Network Services (NS) and VNF packages and this block handles all the requests from the NFV Infrastructure. The orchestrator manages the lifecycle of NSs, as well.
- **VNF Manager**: This block governs all the lifecycle events of VNF instances. These events comprise the instantiation, scaling up/down, updating/upgrading, and termination of VNF instances. It also communicates with the VIM, and reports on VNF-related events.
- **Virtualized Infrastructure Manager (VIM)**: This block controls and manages computing, storage and network resources of the NFV Infrastructure. It is also responsible for the collection and forwarding (to the orchestrator or VNF Manager) of performance measurements and events occurring within the infrastructure.

2.3 Traffic Steering and Service Chaining

Figure 2.3: Synergy between NFV and SDN with Service Chaining



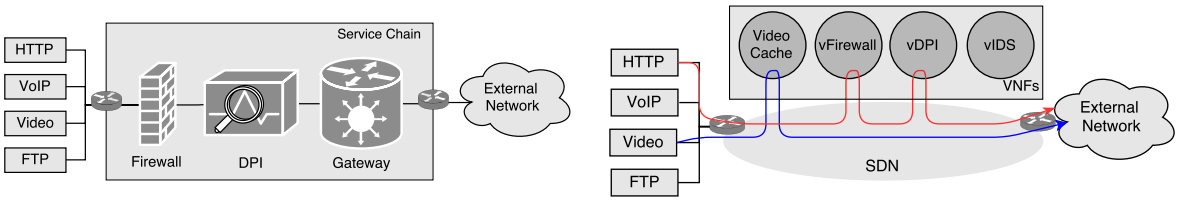
Source: the Author, 2017

After the concepts of SDN and NFV have been described, two further factors will be examined – traffic steering and service chaining – which involve both technologies.

Service chaining is the capacity to create chains of network functions (*e.g.* firewalls, NATs, and IDSes) that are connected, in a determined order, by virtual links. In traditional networks, the chaining of these network functions is static and defined in physical terms, which means that all the traffic traverses the same middleboxes regardless of the type of packet. In Figure 2.4a, there are multiple traffic types that traverse the same chain (Firewall, DPI, and Gateway) before reaching the external network. This type of chain is dependent on the topology, and the middleboxes are connected with wires, so it is difficult to add or remove services and any change since requires the full/partial stoppage of the network connection.

The capacity to dynamically compose service chains relies on the ability of the SDN controller to manage the steering of the incoming traffic through the defined chain in a flexible way. By being able to separate the network control from the data plane, the SDN controller is provided with a global view of the network, and thus the forwarding of packets is simplified. Figure 2.4b depicts a possible dynamic service chaining, where the traffic is steered through different network functions in a SDN before reaching the external network. In addition to this flexibility, NFV ensures the fast instantiation and deployment of VNFs, which means that the functions can operate in a timely way. Figure 2.3 shows the synergy between NFV and SDN that is required to enable dynamic service chaining. Moreover, as the data plane and the control plane are decoupled from each other, Internet Service Providers (ISPs) can design different service chains that provide customized services for a particular client, or create a catalog of service chains with different features.

Figure 2.4: Static and Dynamic Service Chaining Examples



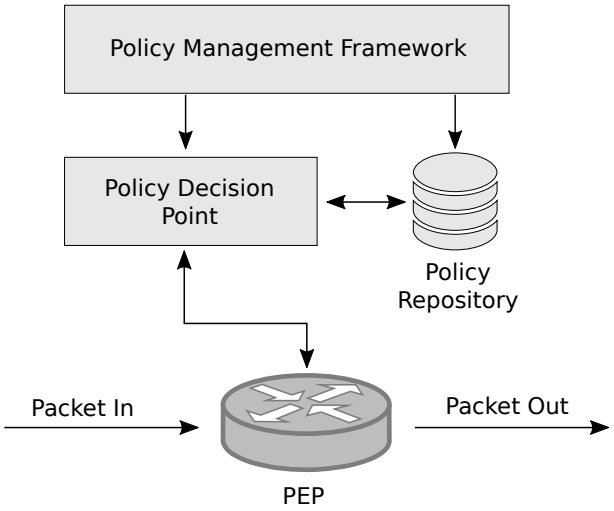
(a) Traditional Static Service Chaining

(b) Dynamic Service Chaining with SDN

Source: the Author, 2017

2.4 Policy-Based Network Management (PBNM)

Figure 2.5: The IETF Policy-Based Management Architecture



Source: the Author, 2017

Policy-Based Network Management (PBNM) is a concept that has already been widely employed and studied for years (STRASSNER, 2003) (VERMA, 2002). This management approach relies on rules to determine configurations, actions, access control, and different level of performance for systems and devices in small or large scale networks. Moreover, PBNM allows the network administrators to manage several devices and network features in an abstract manner (*i.e.* the administrator does not need specific a low-level configuration knowledge to write policies). This means that the policies are separate from the underlying technology of the target entity, and this leads to a more dynamic execution of the whole system.

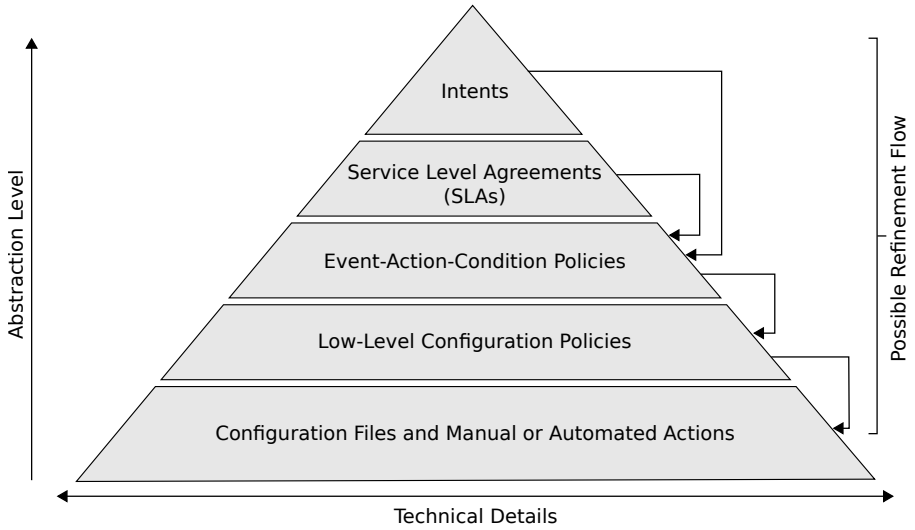
The Internet Engineering Task Force (IETF) has defined, in the document Policy Core Information Model (MOORE et al., 2001), an architecture (Figure 2.5) that supports the many

components involved in a PNBM system. Three key components were defined in this architecture, the Policy Decision Point (PDP), the Policy Enforcement Point (PEP), and the Policy Repository; these are described below.

- **Policy Decision Point (PDP):** This component decides, upon a request, which policy matches the input request and forwards it to the PEP.
- **Policy Enforcement Point (PEP):** This component is any entity capable of enforcing policy decisions in network elements, *e.g.* switches, routers, middleboxes, and so on.
- **Policy Repository:** This component stores policies, actions, conditions, and related policy data for later retrieval by the PDP.

2.4.1 Policy Refinement

Figure 2.6: Diagram of the Types of Policies



Source: the Author, 2017

Policies can have different abstraction levels that vary in accordance with the technical knowledge level required to formulate them. Figure 2.6 shows a pyramid representing some types of policies and their abstraction level related to the low-level technical knowledge contained in each one. At the top of the pyramid is the *intent* type, a high-level abstract policy that does not contain any technical details – this will be more fully described in Section 2.5. It is followed by Service Level Agreements (SLAs), which are business specifications and on the whole contain just a few technical details. On the other hand, Event-Condition-Action (ECA) policies include more details because the network operators must have specific knowledge about the action that must be carried out when an event meets a desired condition, as well as information about the event itself. Low-level configuration policies are the most raw type of policies and

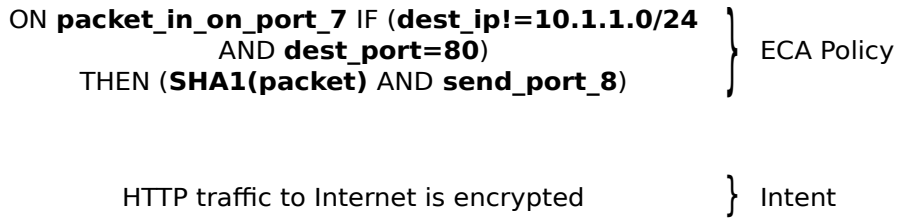
include all the specific nuances of hardware, software, and technology, which often can only be interpreted by computer programs. The bottom of the pyramid consists of configurations files, manual and automated actions, which are not policies but contain the most amount of technical detail and the lowest level of abstraction.

The process of translating high-level policies into lower-level configurations is referred to as *policy refinement* and has been investigated for several years (MOFFETT; SLOMAN, 1993) (CRAVEN et al., 2011). Even though several refinement techniques, such as those that are goal-based (BANDARA et al., 2004), and ontology-based (USZOK et al., 2003), have been put forward to address this problem, some features are still underexploited; this is mainly because of the complexity involved in the refinement process. As can be seen in Figure 2.6, a possible refinement flow passes through a series of types of policies, each with a lower level abstraction and containing as much information as possible. Ultimately, as described by Moffett and Sloman (MOFFETT; SLOMAN, 1993), refinement techniques must seek to answer a number of questions such as:

1. What resources are needed to fulfill the policy requirements?
2. How can high-level policies be translated into a set of low-level actions that a system can enforce?
3. Are the low-level policies accurate enough to satisfy the requirements laid down by the high-level policy?

2.5 Intent-based Networking (IBN)

Figure 2.7: ECA Policy and *Intent* Example



Source: the Author, 2017

Unlike traditional network management paradigms (*e.g.* SNMP, PBNM, and configuration by command line), the purpose of Intent-Based Networking (IBN) is to provide a more natural and straightforward management technique. IBN achieves these aims by allowing administrators to *(a)* configure network elements and *(b)* to formulate rules by giving information about what they wish (in the form of an *intent*). Figure 2.7 gives an example of a general ECA policy and an example of an *intent*. The former contains specific information about what the system

must do (*i.e.* match the destination IP address and port, then use an encryption algorithm and forward the packet), while the latter only describes a desire (*i.e.* that all HTTP traffic to the Internet should be encrypted).

The definition of *intent* is described in the RFC 7575 - Autonomic Networking: Definitions and Design Goals (BEHRINGER et al., 2015) as being an abstract high-level policy. Within the scope of the RFC an *intent* does not contain any low-level configuration or any information about a particular node; moreover, it is usually both defined and provided by a central entity. *Intent* is related to Autonomic Networking in so far as this policy expects the network to be self-managed, which means that the network operator should have the least possible interaction with the network configuration. Moreover, Autonomic networks should adapt to a change in the network on their own on the basis of the defined *intents*. This means that the network operators should only specify their intentions to a central entity, and the network will carry out all the low-level configurations, providing feedback to itself, and thus self-manage the network operation.

3 RELATED WORK

This chapter provides an overview of the related work in the literature. It includes a description of some works that boost the NFV research in Section 3.1. In Section 3.2 we describe the works related to Traffic Steering and Service Chaining that can be integrated with `INSpIRE`. Section 3.3 examines studies in the area of PNBM. Following this, in Section 3.3.1 some policy refinement solutions are set out. Finally, the IBN solutions are analyzed in Section 3.4, and some related work with `INSpIRE` is discussed in Section 3.5.

3.1 Network Functions Virtualization

NFV is a novel network architecture, and as it is still in the early stages of being standardized, it requires closer investigation. However, the scientific community and industry itself are involved in exploring solutions, and implementations that may accelerate the adoption of this technology.

Some examples of NFV solutions that seek to enable the development and deployment of NFV infrastructures and VNFs are OpenNF (GEMBER-JACOBSON et al., 2014), OPNFV (Linux Foundation, 2015), and OpenANFV (GE et al., 2014). While as in the case of OpenANFV, just exploit existing cloud solutions, such as OpenStack (SEFRAOUI; AISSAOUI; ELEULDJ, 2012), others, such as OpenNF and OPNFV are underpinned by SDN and able to take advantage of its adaptability as well. In addition, the European Union has launched a project with the aim of bridging the gap between Cloud and carrier networks, that is called UNIFY (Unify Project, 2014). The purpose of this is to provide a dynamic service creation architecture, which can leverage Cloud virtualization techniques and SDN. This architecture comprises three layers: Service, Orchestration, and Infrastructure.

Implementations that rely on NFV vary from the monitoring of VNFs in distributed scenarios to the visualization of different VNF features and data. Pfitscher *et al.* proposed DReAM, which employs a management by delegation system, where each agent (that is running alongside a VNF) computes a defined diagnostic model to estimate the state of the network services (PFITSCHER et al., 2016). On the other hand, Franco *et al.* proposed VISION. VISION employ a set of interactive and selective visualizations to help network operators to detect causes of problems in NFV-enabled networks (FRANCO et al., 2016). Both implementations are example of solutions originating from academia, and are an evidence of their involvement in fostering the adoption and dissemination of NFV.

3.2 SDN-based Traffic Steering and Service Chaining

With regard to service chaining, the Internet Engineering Task Force (IETF) has described an architecture for the development of Service Function Chains (SFCs) (HALPERN; PIG-

NATARO, 2015). In addition, the Network Service Headers (NSH) (QUINN; ELZUR, 2015) is an approach that introduces a new header in packets traveling through service instances. This header is designed to help in the separation of the traffic. However, its addition increases the traffic processing time, which may cause delays or even packet loss.

Qazi *et al.* developed SIMPLE, which is a solution that relies on SDN to provide middlebox traffic steering (QAZI et al., 2013). SIMPLE introduces a policy enforcement layer which translates user-defined policies into OpenFlow rules and tracks packets that have their headers altered by service instances. This solution addresses the service chaining problem with SDN but ignores the question of whether the middleboxes are deployed as VNFs, and hence is unable to leverage flexibility and scalability of NFV.

Ding *et al.* introduced OpenSCaaS, a platform to provide a tailor-made service for the introduction of service chaining as a service (DING et al., 2015). OpenSCaaS classifies the incoming flow, *e.g.* video or HTTP services, and steers this flow through a set of services depending on what type of flow is involved. For example, a flow classified as a video service is steered through a service chain composed of three functions, a cache, a firewall, and a NAT, before it reaches its destination.

Csoma *et al.* introduced ESCAPE, a prototyping framework that allows the developer to test and chain customized VNFs in an SDN environment (CSOMA et al., 2014). ESCAPE employs consolidated tools, such as Mininet (LANTZ; HELLER; MCKEOWN, 2010) and ClickOS (MARTINS et al., 2014), that form a strong basis to carry out and evaluate different types of NFV and SDN projects. Although the prototype allows the developer to compose any kind of VNF chain, it does not accept any modification of this chain during runtime.

Zhang *et al.* proposed StEERING, a framework that allows the dynamic routing of network traffic through any sequence of middleboxes (ZHANG et al., 2013). This solution takes advantage of existing OpenFlow specifications to introduce a novel solution without making any extensions to the OpenFlow specifications. Moreover, in Frenetic (FOSTER et al., 2011), the traffic steering and classification of traffic is simplified by the abstraction of packet-forwarding policies and modularization of the components.

3.3 Policy-Based Network Management

With the recent interest in both NFV and SDN among the scientific and industrial community, the concept of PBNM is emerging again. Moreover, some high-level languages for programming OpenFlow networks (*e.g.* Frenetic), simplify the steering and the classification of traffic by abstracting packet-forwarding policies and modularizing components and thus, encourage the adoption of PBNM approaches in OpenFlow networks.

Batista *et al.* designed a system for a Policy-Based OpenFlow Network Management (PBONM) (BATISTA; CAMPOS; FERNANDEZ, 2013). The paper argues that the concepts of flows and rules in OpenFlow are not very easy for companies to adopt whereas the concepts of policies

and SLAs are. Thus, this can justify setting up a framework in which these two concepts are merged.

Machado *et al.* attempt to manage an SDN environment with minimal changes for the controller implementation (MACHADO et al., 2015b). To achieve this, the authors introduce a framework that translates, within the scope of the work, Quality-of-Service (QoS) policies into a set of OpenFlow rules. The study seeks to reduce the complexity of management tasks and enable high-level policies to be written by using a CNL. However, the authors do not address the question of traffic steering policies or NFV.

3.3.1 Policy Refinement

Early work on policy refinement in the context of PBNM has achieved promising results. Bandara *et al.* attempted to decompose high-level policies down to low-level concrete policies based on goal mapping (BANDARA et al., 2004). As a result of a formal representation of a system based on Event Calculus (EC), it possible to follow a sequence of operations that will allow it to achieve the desired goal. These goals can be accomplished by reaching one or more of the underlying goals that were previously derived.

Rubio-Loyola *et al.* used linear temporal logic by conducting an analysis of reactive systems to provide a solution for goal-based policy refinement (RUBIO-LOYOLA et al., 2006). By leveraging the Knowledge Acquisition in autOmated Specification (KaOS) methodology, the solution is able to derive goals into low-level policies in the Ponder specification language (DAMIANOU et al., 2001). In addition, the authors outline their solution within a DiffServ QoS management scenario.

Craven *et al.* described a method for the refinement of two types of policies – authorization and obligation policies (CRAVEN et al., 2011). In this study, the domains are represented in UML diagrams, which are used as inputs for the refinement process, together with a policy and decomposition rules. After the decomposition, operationalization and re-refinement stages have been completed, the policy is ready for deployment.

More recently, there has been an investigation of the use of high-level abstractions for configuring and managing SDN-based infrastructures. Machado *et al.* introduced a formalism based on EC to represent high-level Service-Level Agreements (SLA) and then applied logical reasoning to refine these SLAs into low-level rules to manage an SDN network (MACHADO et al., 2015a). The authors argue that some aspects of SDN, such as information gathering about the network (*e.g.* jitter and delay) is made easy by OpenFlow controllers, and thus enhances the policy refinement process in these kinds of environments.

3.4 Intent-Based Networking

As IBN is a novel concept, research in this field is still in its early stages. However, there have been some projects regarding the introduction of *intents* in the area of SDN, such as the ONOS Intent Framework (ONOS Project, 2014) and the NeMo Project (NeMo Project Team, 2015).

The ONOS Intent Framework provides applications with the means to specify requirements of the network, such as new flow rules, link tunneling, or reservations of optical lambdas (wavelengths), in the form of *intents* and then the framework translates these *intents* into activities in the network environment. However, these *intents* are restricted to the Open Network Operating System (ONOS) control plane, and thus are not applicable to different control plane solutions.

The NeMo project provides a North Bound (NB) API that allows applications to specify, (in a domain specific policy language), which *intents* can be translated to create virtual networks. These virtual networks are composed of flows controlled by policies that are translated from the *intent*. Even though NeMo provides a "prescriptive" language for the creation of *intents*, the language still contains low-level information, and is only focused on SDN and not NFV.

3.5 Discussion of Related Work

In this chapter, we listed and described works that are related to `INSpIRE` in different contexts, such as PBNM, policy refinement, service chaining, and IBN. However, as `INSpIRE` adopts an approach to *intent* refinement we only delve into the discussion of the different shortcomings of policy refinement approaches within the *intent* refinement case.

All the works described in Section 3.3.1 address the question of the refinement of high-level policies into low-level policies. However, *intents* are abstract and subjective, and tend to vary from domain to domain. In view of this, there is a need to consider this subjectivity when refining *intents*. Furthermore, the scope of most refinement techniques is restricted to specific domains, such as QoS management in conventional IP networks or access control systems, which confines their employment to the refinement of *intents*. Moreover, a single *intent* can alter the configuration of several elements in the network. Thus, the modeled domains used for the refinement process must accurately reflect the elements and configurations of the whole network and not just be confined to a single area.

As `INSpIRE` is only focused on the refinement of *intents*, it does not handle the steering of the traffic through the composed service chain. The traffic steering can be carried out by an external solution, such as that described in Section 3.2. As a result, `INSpIRE` can be employed in any environment, by acting as a Policy Repository and PDP. For example, in a SDN-enabled network whenever a *packet-in* control message reaches the SDN controller, the controller can request `INSpIRE` to retrieve the matching *intent* together with the packet data (*e.g.* source, destination, and port) and then steer the traffic through the related service chain.

4 INTEGRATED NFV-BASED INTENT REFINEMENT ENVIRONMENT

In this chapter we describe INSpIRE (Integrated NFV-baSed Intent Refinement Environment). In Section 4.1 we provide an overview of the solution along with a description of its main components. Then, in Section 4.2, the proposed natural language to write *intents* is described. The phases of the *intent* translation are outlined in Section 4.3. Lastly, we show screens and functions from the implemented prototype in Section 4.4.

4.1 Integrated NFV-baSed Intent Refinement Environment Overview

INSpIRE address the refinement of *intents* into a set of network functions and information that satisfy a defined *intent*. In addition, INSpIRE provides a framework for the management of VNFs and related configurations, allowing network operators to insert, remove, edit, and visualize information regarding the VNFs' status.

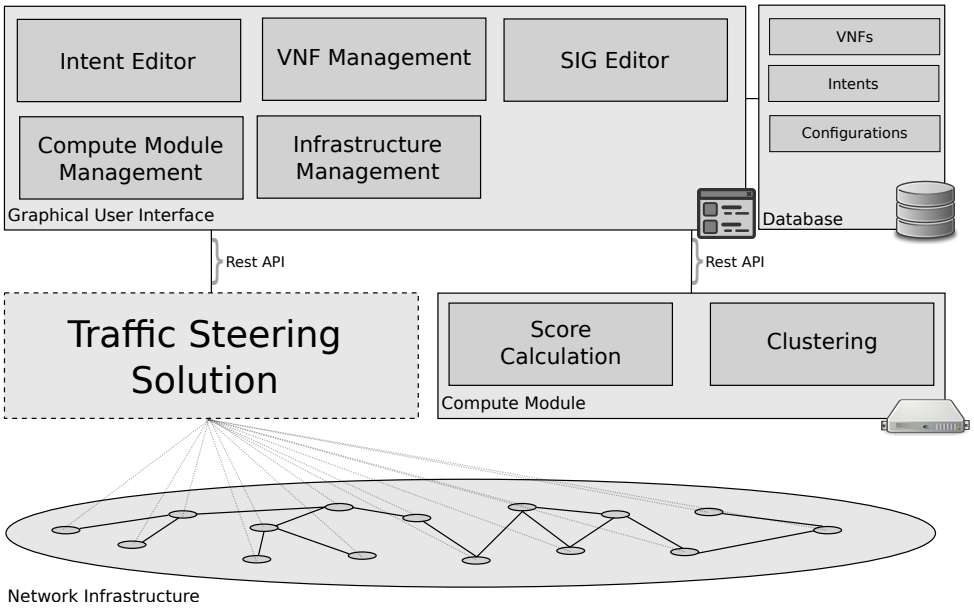
Figure 4.1 depicts the main components and related elements of INSpIRE. The conceptual architecture of INSpIRE is primarily divided into three elements, (*i*) Graphical User Interface, (*ii*) Database, and (*iii*) Compute Module. The Graphical User Interface (GUI) comprises all the elements responsible for interacting with the network operator; they are available in a Web-based interface. The Database stores all the required data about VNFs, *intents*, and configurations of INSpIRE. In order to perform the heavy-duty tasks, such as clustering and calculations, the Compute Module is detached from the GUI so that this element can be placed on a dedicated hardware. The communication between these elements is performed using a REST API. All the components that compose these elements are described in Section 4.1.1.

4.1.1 Main Components

Our solution consists of several components that act together during the task of refining a written *intent* into a chain of VNFs. They are described below.

- **Intent Editor**: This component allows business-level operators to create, retrieve, update and delete *intents*. Operators can also enable or disable *intents* accordingly to their needs. A pre-defined CNL, which is later described in Section 4.2, is used to write the *intents*.
- **VNF Management**: In order for the system to recognize the available VNFs in the infrastructure, infrastructure-level operators must inform their description and details. To manage this information (create, remove and update VNFs) a set of functions are accessible using the operator's account to login in the Web-based interface. The NFV ETSI ISG formalizes the information that a VNF should contain (European Telecommunications Standards Institute (ETSI), 2014). This information is stored in the VNF Descriptor (vnfd), which contains elements regarding requirements of the deployment and operation of VNFs, such as the number of virtual CPUs (computation_requirement),

Figure 4.1: INSpIRE Architecture and Main Components



Source: the Author, 2017

the amount of virtual network bandwidth needed (`virtual_network_bandwith_resource`) and the version of the VNF software. All this data is required by INSpIRE along with the operations that the VNF performs.

- **SIG Editor**: This component enables network operators to change the Softgoal Interdependency Graph (SIG) accordingly to the domain that the SIG will be applied. The network operator can write a SIG with the Non-Functional Requirements (NFRs) designed exclusively for the company that he/she works for, resulting in a more precise refinement process. The SIG and NFR concepts are described later in Chapter 5.
- **Compute Module Management**: This component present information about the compute module, *e.g.* system status (memory and processor usage).
- **Infrastructure Management**: This component manages the infrastructure information, where infrastructure level administrators can specify the different user domains and the IP ranges destined to them. Also, the traffic types that travel through the company's network must be informed, such as HTTP, FTP, SSH, and so on.
- **Score Calculation**: All the functions related to the calculations of NFRs' scores and the processing of the SIG is performed by this component. The mathematical basis for the calculations are presented in Section 5.3.
- **Clustering**: This component performs the task of grouping the VNFs in groups that share similar scores defined in the *intent*. The clustering function and related algorithm are described in Section 5.4.

- **Traffic Steering Solution**[1]: This external component can be an SDN controller that steers the flows through the desired set of VNFs based on the defined service chaining graph. This steering is ruled by the written *intents*, which are stored in the `INSpIRE` database.
- **Network Infrastructure**: Within this components are comprised all the physical resources and controllers. Resources are composed of machines containing compute, storage, and network resources as also their respective managers. In our solution users, VNFs, middleboxes, routers, and servers compose this layer.

4.2 Controlled Natural Language (CNL)

The syntax of many policy languages often resembles the syntax of traditional programming languages, which is the case of Ponder (DAMIANOU et al., 2001). This approach requires the network operator to have a prior knowledge of the language and to translate *intents* into a particular format. On the other hand, with the employment of CNLs (KUHN, 2014) to write policy languages, network operators can write *intents* in (a subset of) English, which diminish the need for prior specific knowledge. Machado *et al.* proved the feasibility of using a CNL to write SLAs that are translated to QoS rules and then enforced in the network elements (MACHADO et al., 2015b). Given this premise, we present a CNL to write rules for the creation of service chaining graphs. The grammar of the proposed CNL is presented in the Listing 4.1.

Listing 4.1: Proposed CNL grammar

```
1 Language: →<Service><Flow><Preposition><Expression>
2 Service: → service−regexes
3 Flow: →<Direction><Target><Direction><Target>
4 Direction: → From|To
5 Target: → user−defined−regexes
6 Preposition: →Have
7 Expression: →<Term>|<Term><Connective><Expression>
8 Term: →<ContextLevel><Context>
9 ContextLevel: → contextLevel−regexes
10 Context: → context−regexes
11 Connective: →And
```

As our *intent* language is defined as a CNL, in order to identify strings that compose a *intent* in a solid way we defined a set of regular expressions. We have classified these regular expressions into four main types according to their purposes:

- *service-regexes*: Utilized to identify the type of service.
- *user-defined-regexes*: User domains that are set by the user, *e.g.* teachers, students, and staff.
- *contextLevel-regexes*: Used to identify the level of the context provided.

[1]This component, as stated before, was not implemented.

- *context-regexes*: Regexes that identify the contexts of the *intent*.

Some examples of regular expressions are presented in Table 4.1. The "dynamic" column specifies if a regex is dynamic, where "Yes" means that a regex can be configured by a network operator, and "No" implies that it is statically defined, thus can not be modified.

Table 4.1: Regular Expressions Examples

Type	*Expression*	Dynamic
service-regexes	HTTP, SMTP, FTP, VoIP...	Yes
user-defined-regexes	teachers, staff, Internet...	Yes
contextLevel-regexes	low, medium, high	No
context-regexes	security, perfomance, inspection...	Yes

4.3 Intent Translation

The process of translating *intents* into service chains comprises three phases: *Intent* Validation, Conflict Detection, and Service Chain Graph Construction. The first two are performed by the *Intent Editor* and the last one is performed by the *Compute Module*. To provide a better organization of this dissertation, we only describe the *Intent* Validation, and Conflict Detection phases in this chapter. The Service Chain Graph Construction phase is described in depth in Chapter 5.

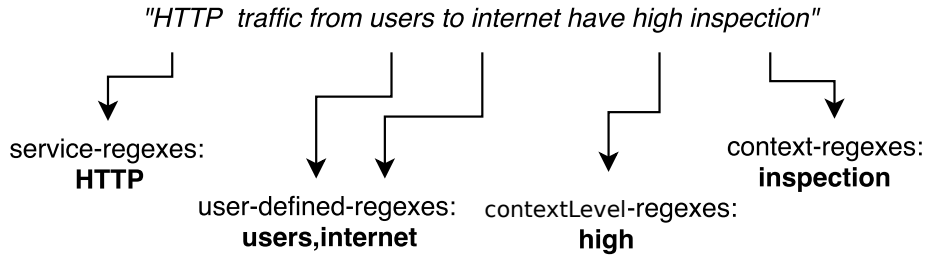
4.3.1 *Intent* Validation

To validate an *intent* written by an operator, the Intent Editor has to parse this *intent* into a set of defined regular expressions, as depict in Figure 4.2. The Intent Editor iterates over the input string to find the *service-regexes* first; then it moves to *user-defined-regexes* that specifies the source and destination of the flow. Finally, it searches for the *contextLevel-regexes* followed by the *context-regexes*. However, if the parser encounters an error (*e.g.* type of traffic not found or missing information) in any part of the parsing process, the *intent* is marked as invalid and is not stored in the database. Also, the operator receives an error message in the Web-based interface to address the error.

4.3.2 Conflict Detection

The main focus of this dissertation is not to resolve conflicts among *intents*. However, our system estimates some conflicting *intents* before its insertion in the database. The Web-based interface displays the conflicting information to the current operator, who must write a new

Figure 4.2: Valid *Intent* Parsing Example



Source: (SCHEID et al., 2016)

non-conflicting *intent*. Conflicts can vary from already defined *intents* (two identical *intents*) to priority conflicts (the inclusion of two equal *intents* but with different *contextLevel-regex*). For example, if an operator writes an *intent "HTTP traffic from teachers to students have high inspection"* and later tries to insert another *intent* informing *"HTTP traffic from teachers to students have none inspection"*, the system will notify the operator of the conflict, which in this case is the same *service-regex* (HTTP), same *user-defined-regexes* (teachers and students) and different *contextLevel-regexes* (high and none) for the same *context-regex* (inspection). Next, after the notification, the operator must resolve the conflicting *intent*.

4.4 `INSpIRE` Prototype GUI Implementation

In this section we present the prototype developed. We developed the Web-based GUI using the CakePHP Web framework[2]. We chose CakePHP due to the author's previous knowledge in working with this framework. For the interface design we used the Bootstrap front-end framework[3]. The database used for storing all the data about the VNFs, *intents*, and related information about the environment is the MySQL[4]. To develop the Compute Module we utilized the Python[5] language along with the NetworkX library[6] to construct and work with graphs.

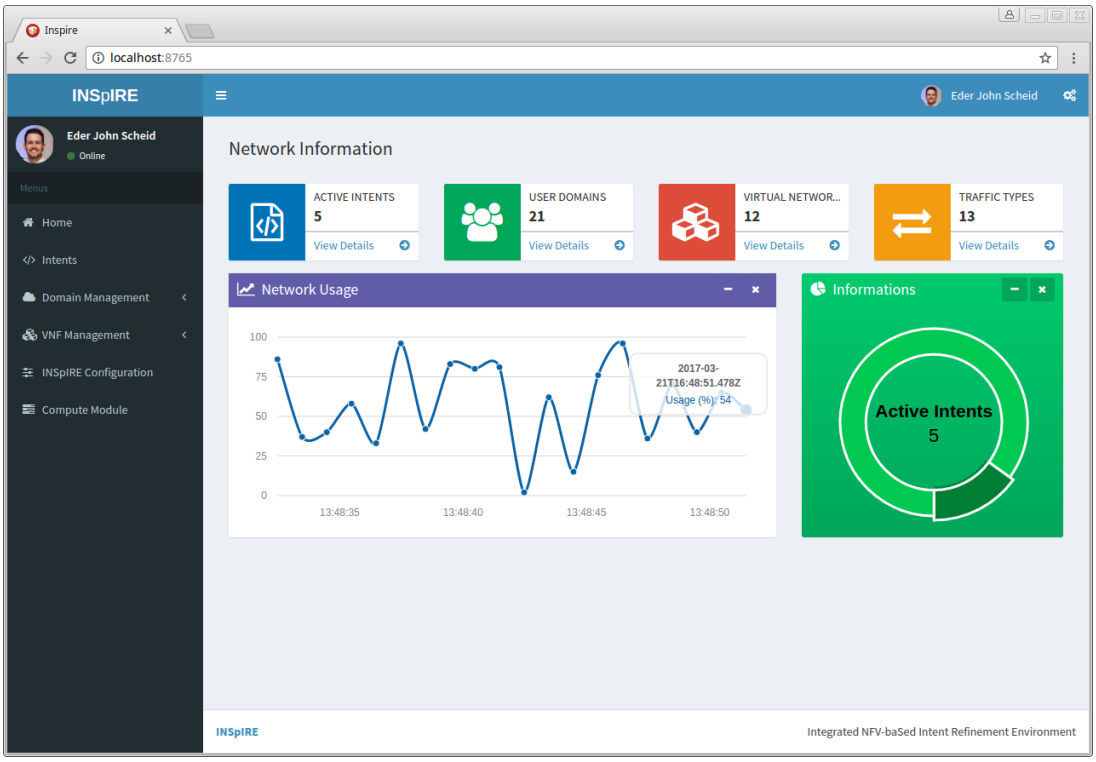
CakePHP is an open-source framework for the fast development of PHP applications. It is focused on providing all the necessary CRUD (Create, Read, Update, and Delete) functions to integrate Web applications with databases. It is built following the Model View Controller (MVC) architecture, and the applications developed on top of the CakePHP framework follows the same architecture. In `INSpIRE`, all the interaction with the user was designed and implemented as Views (*i.e.* templates in the CakePHP). The CRUD operations with the database

[2]https://cakephp.org/
[3]http://getbootstrap.com/
[4]https://www.mysql.com/
[5]http://www.python.org/
[6]https://networkx.github.io/

were implemented in the models, one model for each table in the database. In order to retrieve and deliver the information to be presented in the views, we implemented the controllers for INSpIRE. Each table in the database has its controller, and each View has its function inside a controller, *i.e.* one function for the view that composes the VNF editor, one for the view that lists the VNFs, one for the view that enables the addition of VNF in the database, and so on.

Figure 4.3 depicts the index page of INSpIRE. On the center of the index page, there is a dashboard, where the user can visualize gadgets that present information regarding the number of Active Intents, Users Domains, Virtual Network Functions, and Traffic Types. The user can click in *View Details* to be redirected to view the details of this information. Also, under these gadgets, the historical information, which is updated every second, of the Network Usage is presented in the form of a line chart. The rate between Active Intents and Available Intents is represented in a circle green gadget in the right side of the page. On the left side of the page, under a couple of information (*i.e.* username and avatar) about the user, a menu is placed allowing the user to navigate within all the components of INSpIRE.

Figure 4.3: INSpIRE Index Page

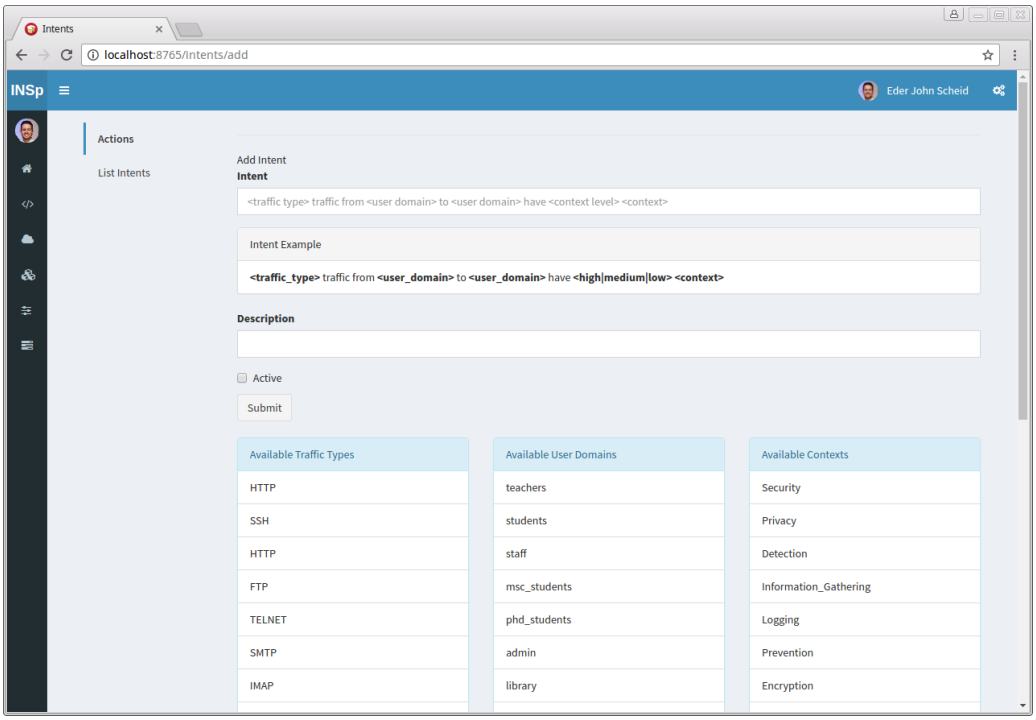


Source: the Author, 2017

Figure 4.4 depicts the page used by users to input *intents* in INSpIRE. We can notice that in Figure 4.3 the menu in the left is expanded, whereas in this screen the menu is collapsed. This option to collapse the menu provides extra space for the user to interact with the authoring screen and write *intents*. To facilitate the process of writing *intents* we provide an example

of an *intent* in both the input field and in a static example below the input field. Also, in the bottom of the figure, we placed all the regexes that can be used to write an *intent*, such as available traffic types, available user domains, and available contexts. Once the user has written the *intent*, he/she gives a full description of the *intent* and selects if it will be active or not in the *Active checkbox*. Furthermore, INSpIRE include a set of different pages, such as the Compute Module Information view (Figure 4.5a) and the SIG Editor view (Figure 4.5b).

Figure 4.4: Intent Authoring Screen



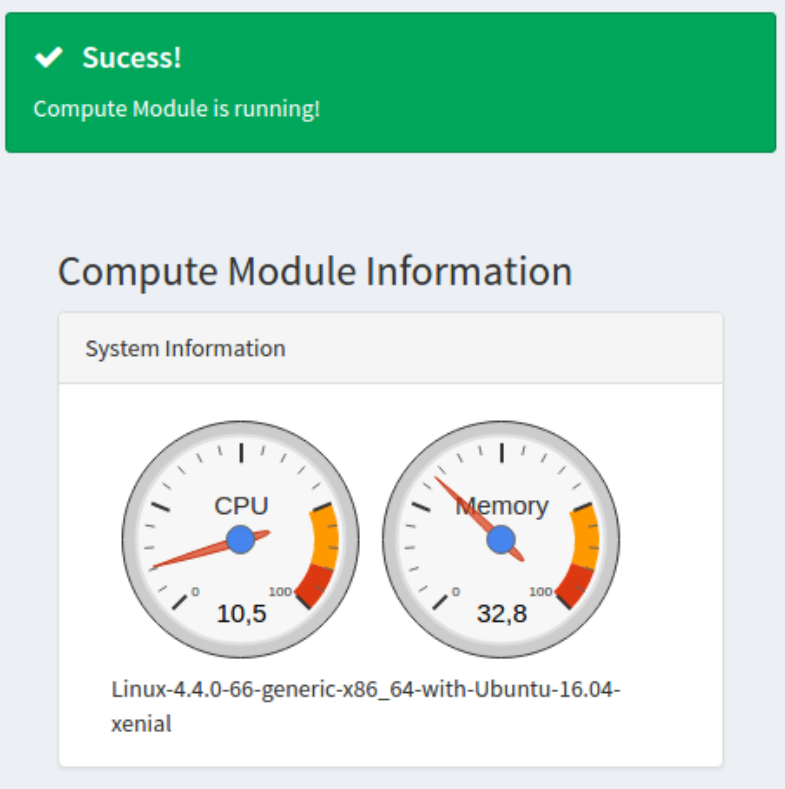
Source: the Author, 2017

Finally, the CakePHP framework allows controllers to publish REST API functions to be used as Web services. Table 4.2 shows some of the functions, and their respective outputs, published by INSpIRE. These functions can be used to create novel solutions in different contexts, such as traffic steering, or data visualization.

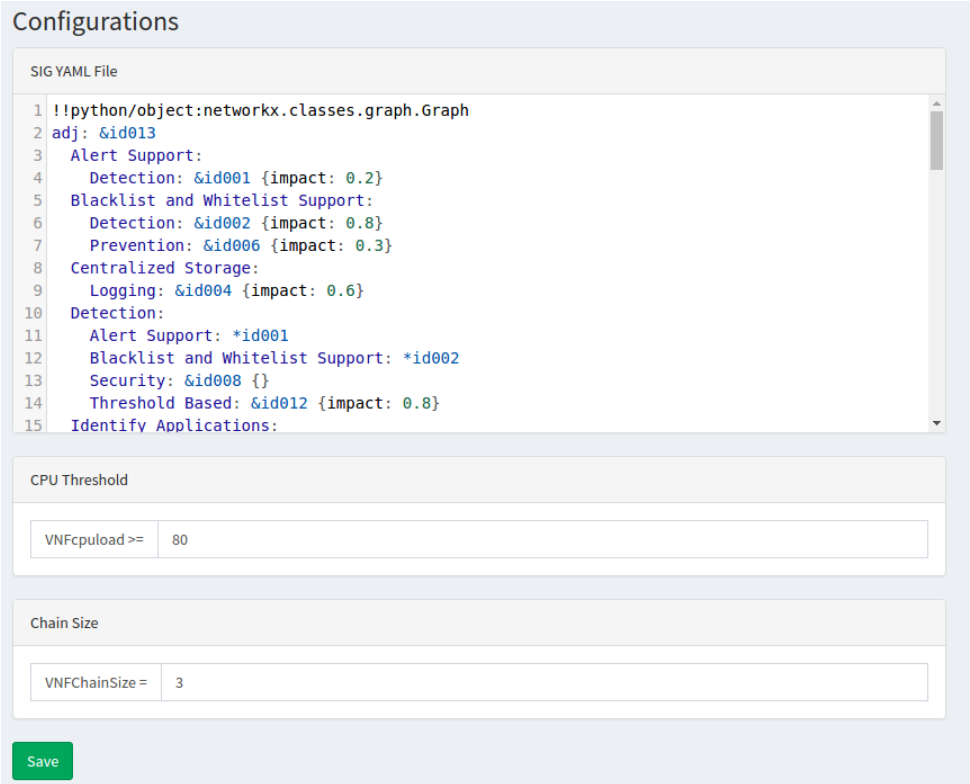
Table 4.2: Exposed INSpIRE REST Functions

Relative INSpIRE URL	Type	Output
/configurations/getSig	GET	Return the SIG saved in INSpIRE
/configurations/getSoftGoals	GET	Return all the *softgoals* of the SIG
/configurations/getLeafSoftGoals	GET	Return all the leaf-*softgoals* of the SIG
/configurations/getOperationalizations	GET	Return all the operationalizations present in the SIG
/vnfs/getVnfs	GET	Return all VNFs stored in the database in a JSON object
/getChain/SRC_IP/DST_IP/TRAFFIC_TYPE/	GET	Return the low-level policy that matches with the arguments informed in the URL

Figure 4.5: INSpIRE Web-Interface Views



(a) Compute Module CPU and Memory Status View



(b) SIG Editor and Configurations View

Source: the Author, 2017

5 REFINEMENT TECHNIQUE

In this chapter we present our refinement technique, which is implemented by `INSpIRE`. The technique is composed of three steps. The first step relates to the modeling of the domain in which the *intent* is being applied, including specifying the operations performed by network functions (*e.g.* L2 inspection and packet filtering) and non-functional requirements (*e.g.* security). We detail the modeling and requirements in Section 5.1 and Section 5.2. The second step includes the quantitative calculation of the non-functional requirements of a VNF based on the modeled domain, resulting in numerical values for these requirements. This step is described in Section 5.3. Finally, the third step includes the parsing of the intent and the clustering of VNFs based on the resultant values from the quantitative calculation, which is detailed in Section 5.4.

5.1 Non-Functional Requirements Framework

When designing software, it is crucial that software engineers consider both functional requirements and non-functional requirements. The former dictates what the software is expected to do (*e.g.* store employees data and exchange email), the latter defines the qualities of the software (*e.g.* store data securely and exchange email quickly). Non-Functional Requirements (NFRs) are, usually, informally specified during the software development process, being based on empirical observation from stakeholders, and thus are hard to model. Therefore, one of the main challenges is to define how one can model the qualities of a system in a comprehensive manner (CHUNG; LEITE, 2009).

To model NFRs, we rely on the NFR Framework (CHUNG et al., 2000). In which *softgoals* and *operationalizations* represent the requirements in a *Softgoal* Interdependency Graph (SIG). An example of a SIG is depicted in Figure 5.1. One *softgoal* (cloud shape) can have different types of contributions and relationships towards other *softgoals*, such as BREAK (−), HELP (+), HURT (−), and MAKE (++). While MAKE and HELP contribute positively to *satisfice*[1] an upper *softgoal*, BREAK and HURT contribute negatively. To satisfice these *softgoals*, one must first identify possible techniques that must be implemented in the system, named *operationalizations* (bold cloud shape). These *operationalizations* are the external nodes of the SIG.

In `INSpIRE` we formally define a SIG with the set below:

$$\mathcal{SIG} = (\mathcal{V}, \mathcal{E})$$
$$\mathcal{V} \in \{SG, LSG, OP\}$$

where

- SG: represents the primary set of *softgoals*, which are the root-node of the graph.

[1] According to the Oxford Dictionary *satisfice* is defined as "Accept an available option as satisfactory". Therefore, we utilize this word instead of *satisfy* in this context.

- LSG: is the set of refined leaf-*softgoals* from the primary *softgoal*.
- OP: contains the set of *operationalizations* that contributes to satisfice the LSG or the SG.

and

$$\mathcal{E} \in \{\uparrow^{++}, \uparrow^{+}, \uparrow^{--}, \uparrow^{-}, \wedge\}$$

where

- $\uparrow^{++}$ (MAKE): Denotes a strong positive contribution towards a *softgoal*. One single MAKE contribution fully satisfices a parent *softgoal* if the offspring is satisficed.
- $\uparrow^{+}$ (HELP): Denotes a positive contribution. Which means that a child *softgoal* partially contributes to satisfice a parent *softgoal*.
- $\uparrow^{--}$ (BREAK): Denotes a strong negative contribution. If a *softgoal* is satisficed then the parent *softgoal* is automatically denied.
- $\uparrow^{-}$ (HURT): Denotes a negative contribution. Which means that a child *softgoal* partially contributes negatively to satisfice a parent *softgoal*.
- $\wedge$ (AND): This contribution relates to a group of *softgoals* to their parent. If all child *softgoals* are satisficed then the parent is also satisficed.

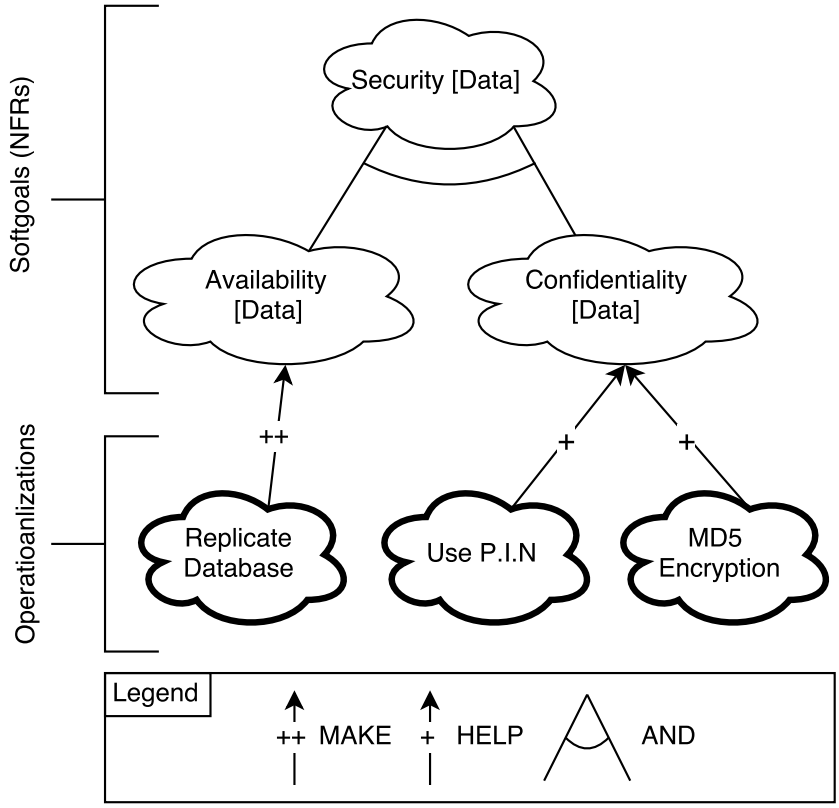
5.2 Softgoal Interdependency Graphs (SIGs)

In software engineering, the modeling of the SIG follows a top-down approach, starting with a high-level *softgoal* being refined into other *softgoals* until the *operationalizations* are defined and selected. However, in our solution, we assume the SIG is already pre-defined, and each VNF is submitted through a bottom-up process in the SIG to quantify its initial *softgoal* score, *i.e.* attributing a numerical value for the primary non-functional requirement. In order to model this pre-defined SIG, we first identify the domain in which the SIG is going to be applied, in our case, middleboxes. Then, we select the non-functional requirements that we want to measure, such as security or performance.

Let us consider the SIG depicted in Figure 5.2. We have extracted the NFRs to compose this SIG from the work Guide to Intrusion Detection and Prevention Systems (IDPS) by Scarfone and Mell (SCARFONE; MELL, 2007). In the work, the authors provide an overview of Intrusion Detection Systems (IDSes) and Intrusion Prevention Systems (IPSes) to help organizations understand such systems. We use this work as a guideline to model a pre-defined SIG, which is then used for evaluating VNFs. To cope with the subjectiveness of the *intents* and requirements, one can alter the SIG at any time to reflect its domain, middleboxes, and network.

To simplify and provide a more straightforward example, we only address the refinement and modeling of one non-functional requirement, which is *Security*. Therefore, following the traditional SIG modeling approach, we start with an initial *softgoal* (*Security*). This *softgoal*

Figure 5.1: SIG Example



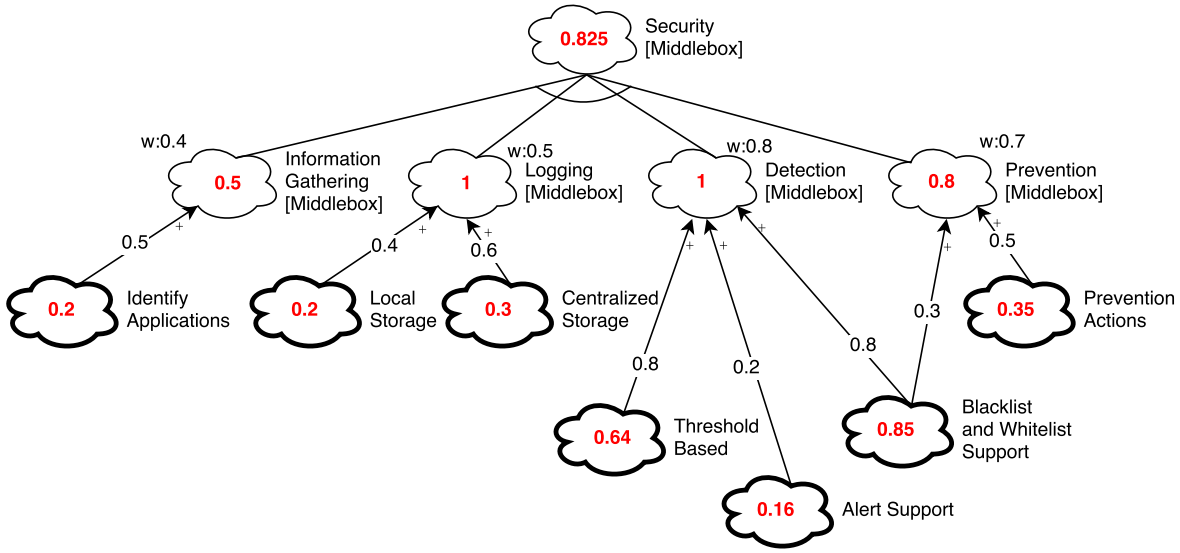
Source: (SCHEID et al., 2017)

is then refined into four leaf-*softgoals*: *Information Gathering*, *Logging*, *Detection*, and *Prevention*. These refined *softgoals* are common security capabilities that, accordingly to Scarfone and Mell, most IDPS technologies provide. For each refined *softgoal*, we attribute a weight corresponding to the importance of this *softgoal* in satisficing the initial *softgoal*. These weights are arbitrary and not set in stone, thus they can be altered by the network operator according to his/her needs. As operationalizations are techniques that contribute to satisfice *softgoals*, a single *operationalization* can have an impact on one or more *softgoals*. For example, the *operationalization Blacklist and Whitelist Support* contributes to both *Detection* and *Prevention softgoals*, while the *operationalization Identify Applications* contributes to only one *softgoal* (*Information Gathering*). These contributions have numerical values attributed to them (similar to the *softgoal* weight) which reflects the impact to satisfice *softgoals*. The bold red values inside the clouds are calculated by `INSpIRE` following the steps presented in the next section.

5.3 Quantitative Calculation of NFR

To accurately quantify the non-functional requirements of a VNF, we leverage the extension of the NFR Framework proposed by Affleck and Krishna (AFFLECK; KRISHNA, 2012). This

Figure 5.2: Pre-defined SIG for Middlebox Security



Source: (SCHEID et al., 2017)

extension provides a lightweight quantitative support for the NFR Framework, defining a mathematical base for the calculation of scores and weights for *softgoals* and *operationalizations*. Given the formalization of the NFRs presented in Section 5.1 and the SIG modeled in Section 5.2, we adapt this extension to our objectives.

Leaf-*Softgoal* weights are defined as:

$$\forall LSG \in \mathcal{V}, (0.0 \leq LSG_{weight} \leq 1.0)$$

where lower values (closer to 0.0) denote a non-critical *softgoal*, while higher values (closer to 1.0) represent critical *softgoals*. The relationships between *softgoals* and *operationalizations* are defined following the contributions depicted in Table 5.1 and are referred to as $impact_{LSGXOP}$. For example, the HELP relationship between the "*Alert Support*" *operationalization* towards the leaf-*softgoal Detection* has an $impact_{LSGXOP}$ of 0.2, which is in $[0, 1]$ range.

Operationalization scores are calculated from top to bottom following Equation 5.1. Therefore, if the network operator decides to add *operationalizations* and *softgoals* to the graph, he/she only includes in the SIG the values of LSG_{weight} and $impact_{LSG \times OP}$.

$$OP_{score} = \sum_{LSG} LSG_{weight} \times impact_{LSG \times OP} \tag{5.1}$$

Given the SIG depicted in Figure 5.2, let us calculate the "*Blacklist and Whitelist Support*" *operationalization*'s score. This *operationalization* contributes positively to two leaf-*softgoals* (*Detection* and *Prevention*). Therefore, we have as the result from Equation 5.1, the score of

Table 5.1: Quantitative Contribuitions from Affleck and Krishna (AFFLECK; KRISHNA, 2012)

Symbol	Name	Contribution
$\uparrow^{++}$	MAKE	1
$\uparrow^{+}$	HELP	[0,1]
$\uparrow^{--}$	BREAK	-1
$\uparrow^{-}$	HURT	[-1,0]
$\wedge$	AND	$\frac{1}{numChilds}$

0.85, which means a positive contribution to the system. The steps are shown in Equation 5.2.

$$\begin{aligned} OP_{score} &= (Detection_{weight} \times impact_{Detection \times BlacklistandWhitelistSupport}) \\ &\quad + (Prevention_{weight} \times impact_{Prevention \times BlacklistandWhitelistSupport}) \\ OP_{score} &= (0.8 \times 0.8) + (0.7 \times 0.3) \\ &= 0.64 + 0.21 \\ &= 0.85 \end{aligned} \tag{5.2}$$

The next step proposed by Affleck and Krishna is the selection of *operationalizations* based on the scores previously calculated (AFFLECK; KRISHNA, 2012). However, in our approach, this step occurs when a network operator inserts a VNF or middlebox in the system. The operator must select which operations the VNF is capable of performing (*e.g.* identify which flows are harmful or detect attacks). Considering the SIG in Figure 5.2, if a network operator specifies that a VNF does not store logs on a centralized server (*Logging [Middlebox]*), the $impact_{LSG \times OP}$ of that *operationalization* is going to be zero. Consequently, the OP_{score} is going to be also zero ($OP_{score} = 0.5 \times 0$) and score of the leaf-*softgoal* will decrease.

To calculate leaf-*softgoal* scores, we employ the same equation as Affleck and Krishna (AFFLECK; KRISHNA, 2012). The only difference is that we consider all *operationalizations* and not only the accepted ones. Equation 5.3 shows that the LSG_{score} is the sum of the impact (even zero impact) of every *operationalization* that contributes to the leaf-*softgoal*. This score is limited to $[-1.0, 1.0]$ by max and min functions, where -1.0 means that the *softgoal* was not satisficed and 1.0 means that the *softgoal* was 100% satisficed. Equation 5.4 reproduces the steps for the calculation of the *Detection[Middlebox]* score.

$$LSG_{score} = max(min(\sum_{OP} impact_{LSG \times OP}, 1), -1) \tag{5.3}$$

$$
\begin{aligned}
LSG_{score} &= max(min(impact_{Detection \times ThresoldBased} + impact_{Detection \times AlertSupport} \\
&\quad + impact_{Detection \times BlacklistandWhitelistSupport}, 1), -1) \\
LSG_{score} &= max(min(0.8 + 0.2 + 0.8, 1), -1) \\
&= max(min(1.8, 1), -1) \\
&= 1
\end{aligned}
\tag{5.4}
$$

Once the *operationalizations* and leaf-*softgoals* scores are computed, the initial *softgoal* (*Security[Middlebox]*) score can be calculated. This score ultimately represents how much (in terms of percentage) the *softgoal* has been satisficed. To simplify our system, we only address AND ($\wedge$) contributions from the initial *softgoal* towards leaf-*softgoals*. Thus, Equation 5.5 considers the sum of leaf-*softgoal* scores divided by the number of children of that *softgoal*, so that every leaf-*softgoal* contributes with a percentage of its score to satisfice the initial *softgoal*.

$$
SG_{score} = max(min(\frac{\sum_{LSG} LSG_{score}}{SG_{numChilds}}, 1), -1) \tag{5.5}
$$

Finally, as our intention is not to calculate how secure a middlebox is, but rather how much security a middlebox can provide to a specific flow passing through it, this score of 0.825 (calculated in Equation 5.6) means that the middlebox or VNF can provide 82.5% of security to this flow. This value is relative to the SIG that was modeled by network operators and may vary from organization to organization. Therefore, as now we have a numerical value for the security *softgoal* of the VNF, we can use this value to cluster the available VNFs into groups with different levels of security. We emphasize that it is important that network operators, administrators and, business partners discuss and model this SIG exhaustively so that the defined weights can faithfully reflect the domain. This is due to the influence of the weights in future service chaining decisions, which directly impact in the clustering process.

$$
\begin{aligned}
SG_{score} &= \frac{InformationGathering_{score} + Logging_{score} + Detection_{score} + Prevention_{score}}{4} \\
SG_{score} &= \frac{0.5 + 1 + 1 + 0.8}{4} \\
&= 0.825
\end{aligned}
\tag{5.6}
$$

5.4 Intent Refinement

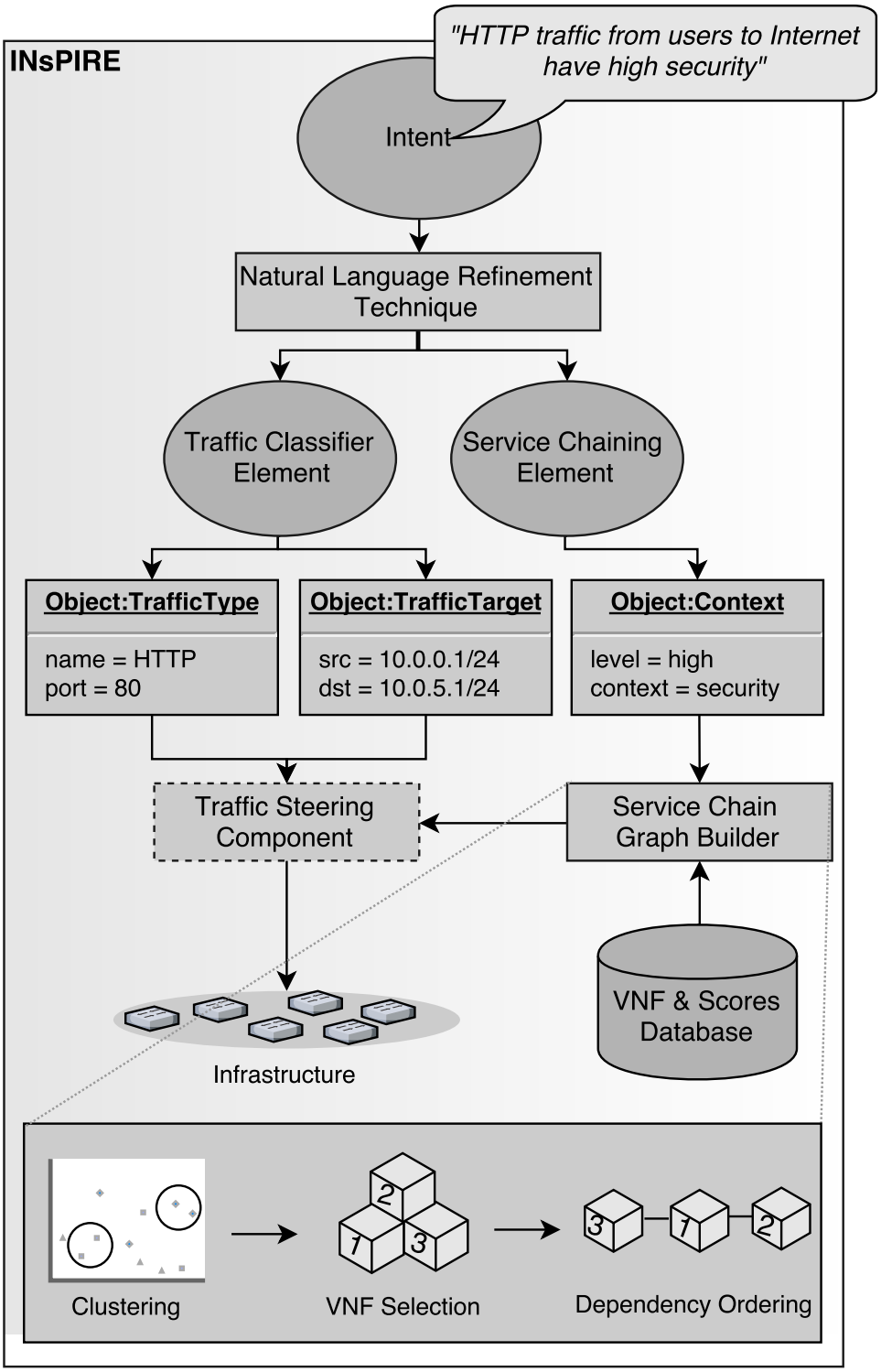
INSpIRE needs to posses knowledge about the environment to properly refine *intents*. Therefore, a network operator has to insert the middleboxes or VNFs present in the infrastructure into a database for later selection. This insertion process consists of uploading in the system a descriptor (*vnfd* in the case of a VNF), filling in information about the middlebox (*e.g.* IP address, switch port, and type of network function), and selecting the operations performed by this network function. These operations are the *operationalizations* defined in the SIG and are necessary for the calculation of *softgoals*. Once this data is informed, INSpIRE computes the *softgoal* score of the target VNF using the Equations described in Section 5.3 and stores it in the database.

After the *intent* is validated, does not conflict with another saved *intent*, and is translated into the regexes described in Section 4.3, INSpIRE can start the refinement process. The refinement process is depicted in Figure 5.3 and includes elements such as a traffic classifier and a service chaining identifier. The former translates part of the *intent*, the *services-regex* and *user-defined-regex* into traffic objects with low-level information (*e.g.* traffic type, port, source IP, and destination IP), that will be used for posterior traffic steering and retrieval by an external component (illustrated by a dashed line box). The latter fetches the *contextLevel-regex* (high) and context-regex (security) of the *intent* and forwards this regexes to the component responsible for constructing the related chaining, named *Service Chain Graph Builder*. To construct the output service chain, the Service Chain Graph Builder carry out three tasks that are described in the next Sections – the Clustering, VNF Selection, and VNF Dependency Ordering tasks.

5.4.1 Clustering

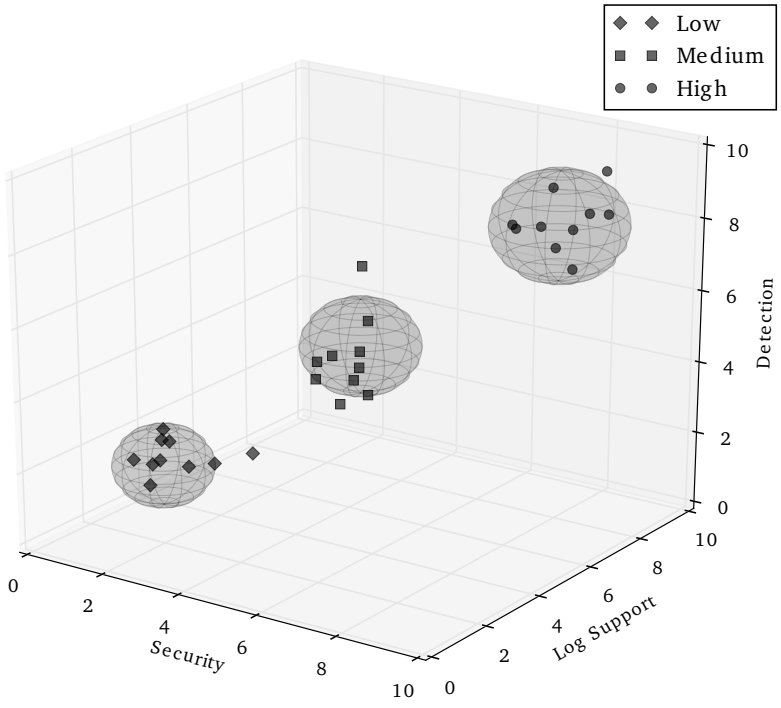
To cluster the VNFs in sets with similar scores, we employ the *k*-means clustering algorithm (MACQUEEN, 1967). This algorithm was proposed to classify n values into a defined k number of clusters sharing similar scores. In our case, we set $k = 3$, representing the levels of the contexts supported by the CNL (low, medium, and high), and n is the number of available VNFs. The dimensionality (*i.e.* number of features) of the plot depends on the number of *softgoals* specified in the *intent*. For example, one can write an *intent* addressing more than one *softgoal*, *e.g.* "*FTP traffic from teachers to teachers have medium* ***security***, ***detection***, *and* ***log support***". Thus, the resulting graph (Figure 5.4) will have three axes (security, detection, and log support) and so forth for more *softgoals*.

Figure 5.3: INSpIRE Intent Refinement Flow



Source: (SCHEID et al., 2017)

Figure 5.4: VNF Clustering Example



Source: (SCHEID et al., 2017)

5.4.2 VNF Selection

After the *k*-means algorithm is executed, and the estimation of the clusters is completed, we must select the VNFs that will compose the service chain. To select from which cluster we will retrieve the VNFs, we leverage the level of the context that was defined by the network operator in the *intent*. In the past *intent* example, one defined as the level being "medium", therefore, we only select the VNFs that are inside the middle cluster, represented as squares in Figure 5.4. We select x random VNFs from the cluster, where $x = 3$ in a first moment. However, this number can be adjusted if the network operator desires. To utilize the full capacity of the VNFs in the infrastructure and not to impact on the overall chaining performance, we employed a selection algorithm. This algorithm prioritizes the selection of VNFs that are already deployed and are not under CPU stress ($VNF_{CPUload} \leq \alpha$, where $\alpha = 0.8$, defined by empirical observation but customizable). If all the deployed VNFs are under CPU stress, then the algorithm selects the undeployed VNFs and the NFV orchestrator takes care of the placement of those VNFs. Bear in mind that is not the scope of this dissertation the placement of VNFs in the network infrastructure.

Algorithm 1 guide the selection of the VNFs that will compose the output service chain. The algorithm starts with an empty list, which will contain the selected VNFs, then it iterates over all the VNFs present in the cluster, if the $vnfList$ size is equal to n, it stops, if not, it selects

one VNF from the input cluster and starts the verification process. The verification process checks if the VNF is deployed (we prioritize already deployed VNFs to diminish the need for new VNF instances in the network) and what is the CPU load of the VNF, if the CPU load is above a certain threshold the VNF is not selected. After this part of the selection algorithm is finished, if the $vnfList$ is not full, it begins the verification process all over again, but this time selecting only not deployed VNFs to fill the $vnfList$.

Algoritmo 1 VNF Selection Algorithm

```
1:  procedure SELECTVNFS(n, cluster[])
2:      vnfList = []
3:      shuffle(cluster)          ▷ We shuffle the cluster to randomize the selected VNFs
4:      for each vnf in cluster
5:          if len(vnf) == n then                                   ▷ VNF list is full
6:              break
7:          end if
8:          if isDeployed(vnf) then
9:              if getCPULoad(vnf) ≤ α then                         ▷ VNF is not under stress
10:                 insert vnf in vnfList
11:             end if
12:         end if
13:     shuffle(cluster)          ▷ We shuffle again the cluster to randomize the selected VNFs
14:     for each vnf in cluster
15:         if len(vnf) == n then
16:             break
17:         end if
18:         if isDeployed(vnf) then
19:             continue
20:         end if
21:         insert vnf in vnfList
22:     return vnfList
23: end procedure
```

5.4.3 VNF Dependency Ordering

Many VNFs often depend on other deployment units (*i.e.* VMs) to be deployed/initiated before an orchestrator can deploy them. The information about this dependency is stored, accordingly to the ETSI NFV MANO, in the element `dependency` inside the *vnfd* of the VNF. Therefore, if a VNF is selected and is not deployed, `INSpIRE` attempts to resolve this dependency by searching for a VNFD Virtual Deployment Unit (*vnfd:vdu*) that fulfills this dependency. First, it seeks if one of the selected VNFs produced by the selection procedure contains the *vnfd:vdu* specified in the `dependency` element. If there is no match, then it searches all the deployed VNFs for a match and then informs the orchestrator to address this issue. Note that this dependency process relates to the deployment and placement of VNFs, which is out of

the scope of this dissertation, and thus we do not delve into details.

Moreover, when designing service chains, it is important to consider the logical order of the network functions in the chain. This ordering process is not a trivial process, mainly because of the hidden and subjective dependencies across the many network functions. For example, there is no explicit definition that a firewall must come first in the chaining order and then be followed by a DPI. This ordering is based on empirical observation and the logic behind the operations performed by the network functions. It is illogical to put a DPI in front of a firewall, because the DPI will have to inspect every single incoming packet, causing performance degradation. Whereas if the firewall is set in front of the DPI, the firewall will only forward filtered packets, reducing the number of packets to be processed by the DPI, and thus, not impacting on the overall service chain performance.

To address this ordering issue, we propose to order the chain based on the complexity of the selected network functions.[2] We advocate that the complexity of a network function is based on the number of *operationalizations* that were previously selected by the network operator during the insertion of the network function in the database. Thus, if we have a DPI performing ten *operationalizations* and a firewall performing only three *operationalizations*. Then, as the DPI is performing more actions, it will be included after the firewall in the chain, *i.e.* the flow will pass through the firewall first, then go through the DPI and so on to more complex network functions.

[2]Note that after `INSpIRE` completes all of its stages (clustering, selection, and ordering), the chain can be ordered by a network operator to fit his/her needs directly in the database.

6 PROTOTYPE AND EVALUATION

This chapter provides an overview of the prototype and the experiments carried out in order to validate our *intent* refinement solution. In Section 6.1 we conduct a case study of the complete `INSpIRE` flow, from SIG modeling to *intent* authoring. Then, in Section 6.2 the refinement output chains of three different case studies are described. Finally, in Section 6.3 we present and discuss the experimental evaluation of two `INSpIRE` components.

6.1 INSpIRE Prototype Case Study

Let us describe the interaction of users with `INSpIRE`. This interaction is composed of three distinct stages: (*i*) SIG modeling, (*ii*) VNF insertion, and (*iii*) *Intent* authoring. These stages are described in Section 6.1.1, Section 6.1.2, and Section 6.1.3, respectively.

6.1.1 SIG Modeling

In a first moment, without the intervention of `INSpIRE`, network operators along with business partners assemble a team to identify the non-functional requirements (initial *softgoal* and leaf-*softgoals*) that are important for the proper operation of the company. Suppose that the company is concerned about the privacy of its data. Therefore, the team identifies the initial *softgoal* as *Privacy* and two requirements (leaf-*softgoals*) that must contribute to it, such as *Encryption*, and *Traffic Anonymization*.

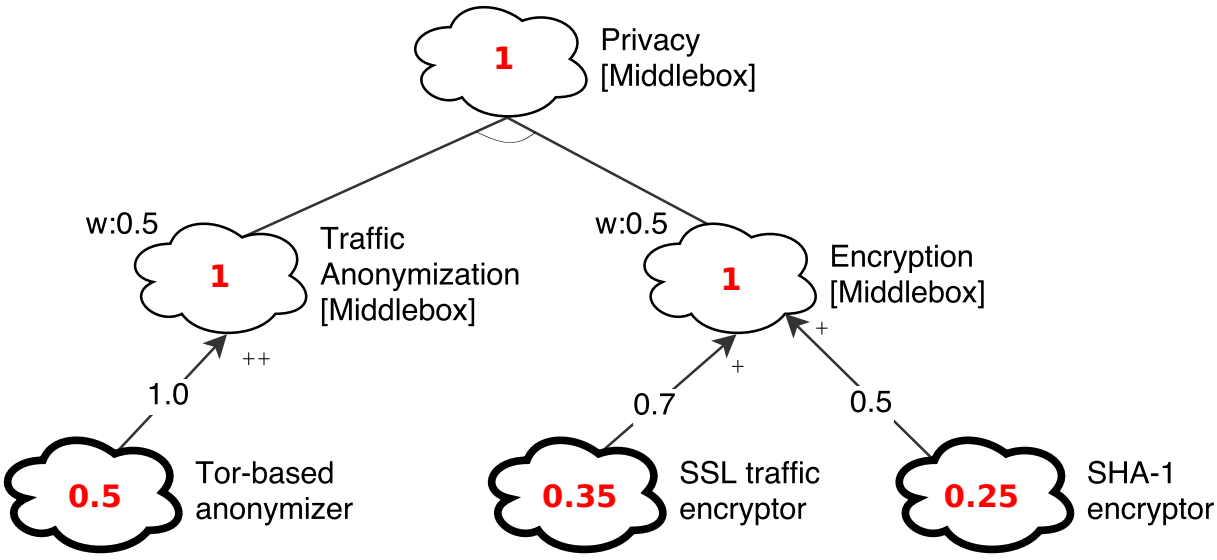
These two requirements are satisficed by the *operationalizations*, which must also be identified and outlined by the team, either by reviewing common functions realized by middleboxes or by empirical knowledge. In the privacy context, the team outlines functions such as *Tor-based anonymizer*, *SSL traffic encryptor*, and *SHA-1 encryptor*. Next, the team attributes the weights of all leaf-*softgoals* and the impact of each *operationalizations* in these leaf-*softgoals*. The attributed impacts of the mentioned *operationalizations* are described in Table 6.1. As the team identified two leaf-*softgoals* they attributed equal weights to both of them, *e.g.* 0.5.

Table 6.1: Impact of *Operationalizations* towards Leaf-*Softgoals*

Operationalization	Contribution	Leaf-*softgoal*	Impact
Tor-based anonymizer	MAKE	*Traffic Anonymization*	1
SSL traffic encryptor	HELP	*Encryption*	0.7
SHA-1 encryptor	HELP	*Encryption*	0.5

The modeled SIG is shown in Figure 6.1 with its scores already calculated. This SIG can be represented in YAML (YAML Ain't Markup Language), XML (eXtensible Markup Language), or JSON (JavaScript Object Notation) format and then imported to `INSpIRE` as well as being edited inside `INSpIRE`. An example of imported SIG in YAML format is depicted in Figure 6.2. We adopted the YAML format due to its readability. In the YAML file, nodes of the SIG are described after the `node:` tag (line 27) in the `node:{atributes:value}` format. For example, the node *Traffic Anonymization* (line 36) is represented as `Traffic Anonymization:{lsg: true, w: 0.5}`, which means it is a leaf-*softgoal* (`lsg: true`) with a weight of 0.5 (`w: 0.5`). Edges are described after the `adj:` tag (line 2) in the `sourceNode:` format. New lines with indentation describe the destination nodes and attributes in the `destinationNode:{attribute:value}` format. For example, there is an edge from node `SSL Traffic Encryptor:` (line 15) to node `Encryption: {impact: 0.7}` (line 16), meaning that the node *SSL Traffic Encryptor* impacts on 70% to satisfice the node *Encryption*.

Figure 6.1: Modeled SIG for Middlebox Privacy



Source: the Author, 2017

Figure 6.2: Example of a SIG in YAML Format

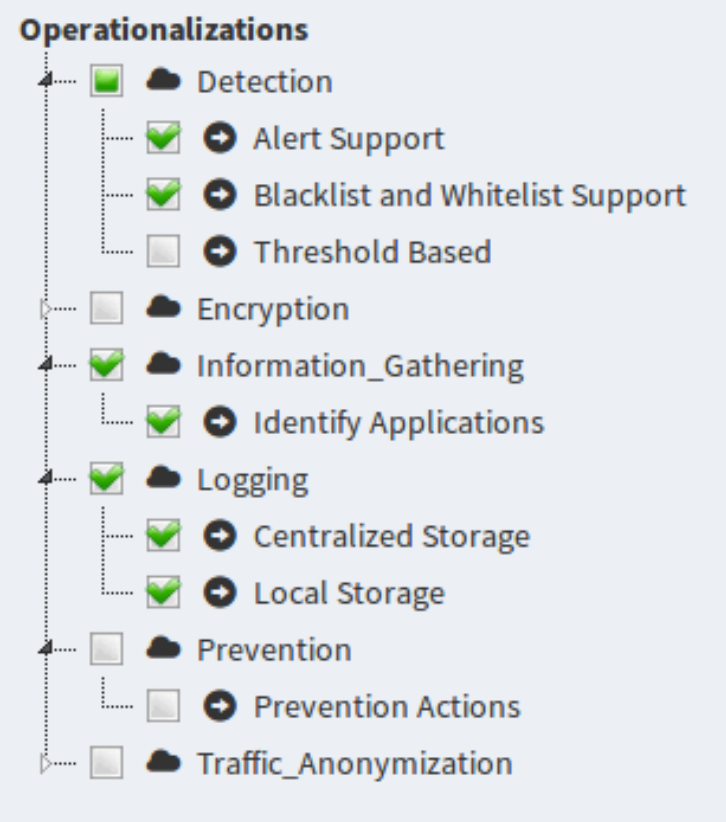
```
1 !!python/object:networkx.classes.graph.Graph
2 adj: &id013
3    Alert Support:
4      Detection: &id001 {impact: 0.2}
5    Blacklist and Whitelist Support:
6      Detection: &id002 {impact: 0.8}
7    Centralized Storage:
8      Logging: &id020 {impact: 0.6}
9    ...
10   Privacy:
11     Traffic Anonymization: *id006
12     Encryption: *id007
13   Tor-based Anonymizer:
14     Traffic Anonymization: &id015 {impact: 1.0}
15   SSL Traffic Encryptor:
16     Encryption: &id011 {impact: 0.7}
17   SHA-1 Encryptor:
18     Encryption: &id017 {impact: 0.5}
19   Security:
20     Detection: *id008
21     Logging: *id010
22   ...
23 adjlist_dict_factory: &id014 !!python/name:__builtin__.dict ''
24 edge: *id013
25 edge_attr_dict_factory: *id014
26 graph: {}
27 node:
28   Alert Support: {op: true}
29   Blacklist and Whitelist Support: {op: true}
30   Centralized Storage: {op: true}
31   Tor-based Anonymizer: {op: true}
32   SSL Traffic Encryptor: {op: true}
33   SHA-1 Encryptor: {op: true}
34   Identify Applications: {op: true}
35   ...
36   Traffic Anonymization: {lsg: true, w:0.5}
37   Encryption: {lsg: true, w:0.5}
38   Detection: {lsg: true, w: 0.8}
39   Logging: {lsg: true, w: 0.5}
40   ...
41   Privacy: {sg: true}
42   Security: {sg: true}
43 node_dict_factory: *id014
```

Source: (SCHEID et al., 2017)

6.1.2 VNF Insertion

After the SIG is modeled and imported to INSpIRE, the network operator can start to add the VNFs and middleboxes that are available in the infrastructure. To perform this addition, he/she utilizes a GUI provided by INSpIRE, which contains a form with the necessary fields to be filled by the network operator with information about the middlebox. Such fields include name, description, IP address, switch, switch port, VNFD file (VNF only), and dependencies. Within this GUI, a list of *operationalizations* in a tree view format, separated by leaf-*softgoal*, is presented for the network operator to select the functions that the VNF carry out (Figure 6.3). For example, if the VNF is a firewall, the network operator will select operations such as *Blacklist and Whitelist Support* and *Alert Support* and submit the form. The *operationalizations* list is composed of nodes of the SIG and may change if the SIG is altered. Once the form is submitted, INSpIRE automatically calculates the scores following the equations described in Section 5.3 and stores the scores in a database along with the information about the middlebox previously informed. One example of entry in the scores database is the tuple `<vnfId,[Security:0.825,...,Detection:1]>`.

Figure 6.3: Operationalizations Tree View



Source: the Author, 2017

6.1.3 *Intent* Authoring

The next step is to write *intents*. This process in based on the language described Section 4.2. We utilize this language for the composition of *intents* in `INSpIRE`. The *intents* that are going to be written in `INSpIRE` are in the format "`trafficType` *from* `source` *to* `destination` *have* `contextLevel contextsList`". For example, let us assume that a board of directors described an SLA specifying that all email traffic from the Finance Department must have high security and privacy. Therefore, a network operator would translate this SLA into two *intents*: *(i)* "*SMTP traffic from finance_department to * (any) have high security and privacy*", and *(ii)* "*IMAP traffic from finance_department to * have high security and privacy*". Then, `INSpIRE` refines these two *intents* (one at a time) into objects to be used for traffic classification and service chain construction purposes. As `INSpIRE` only address service chain construction, the refined elements utilized are: `contextLevel: high` and `contextList:[security,privacy]`. Therefore, `INSpIRE` retrieves all the entries of the score database in order to utilize these scores to cluster the VNFs. In the example above, the cluster will have two dimensions (security and privacy) and `INSpIRE` will plot every VNF based on its score of these two *softgoals*.

Once the entries are plotted, `INSpIRE` discover the three clusters and selects only the entries that belong to the context level defined in the *intent*, *e.g.* (*high*). Next, `INSpIRE` orders the selects VNFs following the process described in Section 5.4.3 and informs the traffic steering element of the service chain related to the traffic classification objects and the *intent*. Finally, packets originated from IPs in the 192.168.1.5\24 range (*i.e.* finance_department) and classified as SMTP (port 25) are steered through the chain related to these objects by an external solution.

6.2 Intent Refinenemt Case Studies

To provide an evaluation of the feasibility of `INSpIRE` , we describe its behavior in three different scenarios that serve as case studies. The choice of these scenarios was based on two premises: the various levels of hierarchy present in the infrastructure and the presence of heterogeneous traffic in the network. As an example of the level hierarchy, we can state that in the case of VNF Service Chain as a Service, *"gold users"* have a higher priority than *"silver users"* and so on. Heterogeneous traffic is described as the presence of different protocols competing for a network share.

6.2.1 Case Study 1 - Generic Academic Network

Typically in universities, schools or colleges the network is shared with different users, such as *teachers, students, staff and guests*. These users also have different levels of freedom in

Table 6.2: User Domains and Respective IP Ranges

Scenario 1		**Scenario 2**		**Scenario 3**	
User Domain	IP Range	User Domain	IP Range	User Domain	IP Range
teachers	192.168.0.0/24	*human_resources*	10.0.2.1/24	*platinum*	172.16.1.1/22
students	192.168.3.0/24	*accounting*	10.0.3.1/24	*diamond*	172.16.4.1/22
staff	192.168.5.0/24	*development*	10.0.4.1/24	*gold*	172.16.8.1/22
msc_students	192.168.7.0/24	*directory*	10.0.5.1/24	*silver*	172.16.12.1/22
phd_students	192.168.9.0/24	*inventory*	10.0.6.1/24	*bronze*	172.16.16.1/22
library	192.168.11.0/24	*marketing*	10.0.7.1/24	*standard*	172.16.20.1/22

the network depending on their occupation. Thus, the set of services that a packet travels in the network changes accordingly to its source and destination. In order to guarantee this forwarding in a dynamic form, infrastructure-level operators have to define, in our system, the user domains that are present in the network. In column *Scenario 1* in Table 6.2 are represented some common user domains that can be present in an academic network, followed by their respective IP range. This information is used to steer the traffic correctly when a packet matches with an user-defined domain, and a traffic type.

In the case of a university, if the dean, which can be characterized as a business-level operator, wants to determine that the traffic from Master students must experience a high level of security before achieving the Internet, he/she will write the following *intent* in the Intent Editor: *"HTTP traffic from msc_students to Internet have high security"*. This *intent* will then be refined to a service chaining graph containing the highest level of security that the available VNFs in the university's infrastructure can provide. This chaining can be dynamically changed if another *intent* is written, *e.g., "FTP traffic from msc_students to phd_students have low security"*. There is now a different *intent* to traffic originated from Master students, so `INSpIRE` creates a new service chaining graph with the desired level of security. The output generated by `INSpIRE` of these *intents* is shown in Table 6.3.

Table 6.3: Results Obtained for the Case Study 1

context	*contextLevel*	`INSpIRE` Output Chain
security	*high*	WAF → Deep Packet Inspection → Complete IDS
	medium	Logger → Simple vFirewall → Firewall
	low	Alert VNF → Tor-based Anonymizer

We notice that the requirement of dynamically steering the traffic through a set of network functions is achieved. Packets from the same user with different destinations in the network do not necessarily traverse the same network functions as occurs in traditional networks. In addition, the path that a packet will traverse is ruled by the previously defined *intents* and the context level specified in each one.

6.2.2 Case Study 2 - Common Company Network

Consider the case of a generic company, with different departments, such as Marketing, Human Resources, Directory, and among others. These departments have different network requisites due to the diversity of applications in each of them. For example, the need for a high level of privacy on financial transactions originated from the Marketing department or a strong security between the communication of two branches. Thus, the organization's network business-level operator and infrastructure-level operators must guarantee that these requirements are fulfilled. In order to comply with these requirements, infrastructure-level operators can define different IP ranges for the departments. Moreover, business-level operators can define sets of *intents* for the traffic traveling on the organization's network.

As an example, we will define an company with the user domains presented in Table 6.2, column *Scenario 2*. The board of directors may hold weekly meetings with the human resources department. In order to eliminate the need of physical presence in the conference room, the participants can use VoIP calls to attend the meeting. This possibility introduces the VoIP privacy requirement. To address this requirement, one could write an *intent* of the type *"VoIP traffic from accounting to directory have high privacy"* and only activate this *intent* once a week, during the meeting. The traffic matching the *intent* will then be steered through the set of middleboxes imposed by `INSpIRE`, guaranteeing a private communication, and dynamically using network resources. Furthermore, if we consider an organization with more than one branch, infrastructure-level operators can detail branch domains and user domains for these branches and a business-level operator can write *intents* accordingly, *e.g., "HTTP traffic from* $branch_1$ *to* $branch_3$ *have medium privacy"* and so on.

Table 6.4: Results Obtained for the Case Study 2

context	*contextLevel*	`INSpIRE` Output Chain
privacy	*high*	Encryption VNF
	medium	Tor-based Anonymizer
	low	Alert VNF $\rightarrow$ Logger $\rightarrow$ Firewall

This example aims to provide a picture of how can our approach be used in organizations with multiple departments. As we can see, the lowest level of privacy was composed of three VNFs, and the highest level was composed of just one, however, the three VNFs of the lowest level (Alert VNF, Logger, and Firewall) do not provide any privacy to the traffic, whereas just the one (Encryption VNF) alone provide the privacy of the traffic. The CNL proposed by our solution is generic enough to include the requirements of different types of departments and occasions present in companies. In addition, there is support for companies with multiple branches, as user domains can be assumed to be edge routers inside a branch.

6.2.3 Case Study 3 - VNF Service Chain as a Service (VNF-SCaaS)

The price of current Internet middleboxes represents a significant percentage of a company's expenses. With this premise, there is the possibility to monetize service chaining graphs specially designed for exclusive corporations, reducing its CAPEX and OPEX. Our approach facilitates this monetization of service chains by allowing the network operator to set different users domains in the infrastructure with different classes, such as represented in column *Scenario 3* in Table 6.2. If a company does not require a high level of inspection or traffic performance it does not need to buy an expensive middlebox just to use some of it features, it pays to have a low degree of inspection in a private NFV environment and redirect the traffic to them. Some examples of *intents* that can be applied to a VNF-SCaaS environment are: *(i) "VoIP traffic from diamond to Internet have high logging, and detection"*, *(ii) "VoIP traffic from gold to Internet have medium logging, and detection"*; and *(iii) "VoIP traffic from silver to Internet have low logging, and detection"*.

As the steering of traffic is dynamic, business-level operators can define not only classes but generic traffic from different tenants. Let us consider the case where a company provides VNF as a Service. In this case, there is more than one subscriber to this service, so a business-level operator can define *user domains* for different subscribers (tenants). Once these domains are defined, the operator can write *intents* such as *"HTTP traffic from* $tenant_X$ *to* $tenant_Y$ *have high inspection"* or *"Video traffic from* $tenant_X$ *to* $tenant_Z$ *have medium performance"*. The first *intent* is translated, and our solution constructs a service chain for this flow containing an IDS, a Firewall and a DPI. Having the service chain defined, the SCaaS provider can charge the $tenant_X$ accordingly. The second *intent* specifies that *video service* from $tenant_X$ to $tenant_Z$ must have some level of performance. However, the SCaaS provider only owns a license for a single video caching virtualized function; therefore, the resultant graph will be composed of only the video cache function. This service chain (a video caching function) may improve the overall video quality in streaming, meeting some of $tenant_Z$'s expectation.

Table 6.5: Results Obtained for the Case Study 3

context	*contextLevel*	`INSpIRE` Output Chain
logging, and detection	*high*	Logger → Firewall Raspberry → Complete IDS
	medium	Firewall → WAF → nDPI
	low	Alert VNF → Tor-based Anonymizer → Encryption VNF

This case study is focused on applying NFV as a Service alongside with Service Chaining as a Service. We observe that both service chaining and NFV can be monetized as a business-level operator can add value to different service chains and VNFs, thus charging accordingly. Also, we can notice that *intents* can comprise more than one leaf-*softgoals*, and/or *softgoals*, such as logging and detection.

6.3 Evaluation

We conducted two evaluations. The first, in Section 6.3.1, evaluates the clustering process. While the second, in Section 6.3.2, presents experiments with the score calculation process. The algorithms, equations, and simulations were implemented in Python utilizing well-know graph (NetworkX (HAGBERG; SCHULT; SWART, 2008)) and clustering (SciPy (JONES et al., 2001–)) libraries.

This section describes the scalability of the clustering phase of INSpIRE. Also, this section aims to evaluate the time that is necessary to calculate all the scores of a network function. The simulations were performed in a Dell XPS 8900 with an Intel Core i7-6700 CPU at 4GHz processor and 16GiB of RAM memory.

6.3.1 Clustering Evaluation

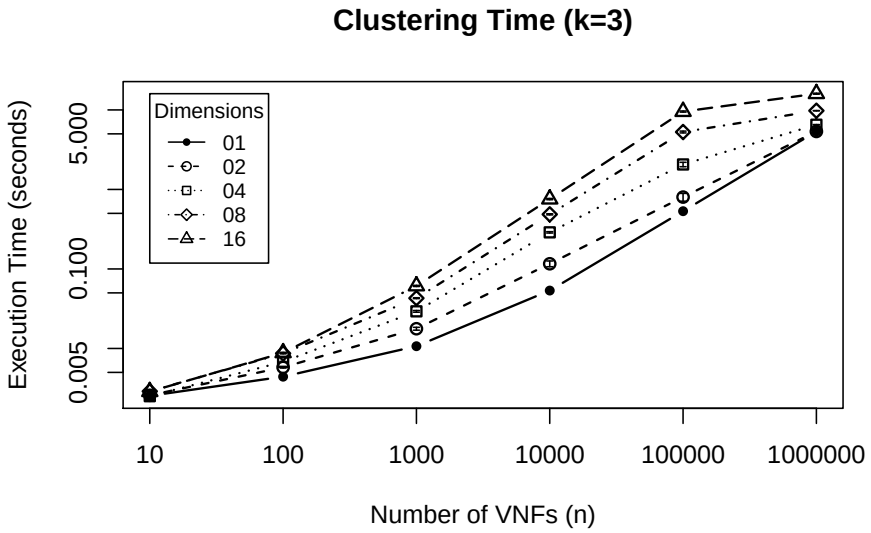
This component, the clustering phase, of `INSpIRE` is implemented utilizing the k-means algorithm. The simulations were ran thirty times to provide enough significance level. The error bars display the standard deviation. We simulated different types of environments varying the number of elements (n) to be clustered and the number of dimensions of the elements. The number of dimensions represented the number of *requirements* specified in the *intent* and the number of elements represented the number of VNFs and middleboxes in the database. The score of each requirement was randomly attributed varying from 0 to 1. Also, the number of clusters (k) for the algorithm to estimate was set to 3 due to the different context levels (low, medium, and high).

Figure 6.4 shows the Execution Time (in seconds) for each set of VNFs varying the number of Dimensions. The number of VNFs varied from 10 VNFs up to 1 million VNFs, and the number of dimensions ranged from 1 to 16. We notice that the k-means execution time is relatively insignificant up to 10.000 VNFs with 16 dimensions. Even for a million of VNFs with four (4) dimensions (square dashed line) the algorithm took 5 seconds to execute. Therefore, `INSpIRE` can cluster up to 10000 (ten thousand) VNFs and middleboxes with 16 dimensions in less than 1 second.

6.3.2 Score Calculation Evaluation

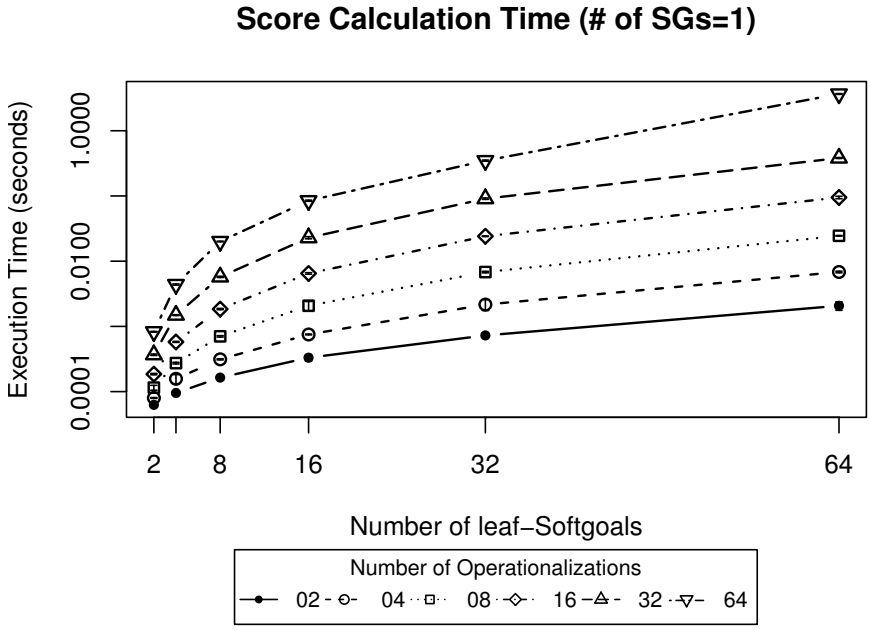
To evaluate this component, the score calculation phase, we simulated different SIGs with the number of leaf-*softgoals* and *operationalizations* varying from 2 to 64 in a logarithmic scale. For example, we created a SIG with 2 leaf-*softgoals* with 2 *operationalizations* each, then a SIG with the same number of leaf-*softgoals* (2) but with 4 *operationalizations* each and so son until we reached a SIG with 64 leaf-*softgoals* with 64 *operationalizations* each. Due to the simplicity of the equation to calculate the score of the initial-*softgoals*, which is only a

Figure 6.4: Time to discover 3 clusters in different scenarios



Source: (SCHEID et al., 2017)

Figure 6.5: Time to calculate the scores in different SIGs



Source: (SCHEID et al., 2017)

division, the number of initial-*softgoals* in the experiments was fixed to 1.

For every SIG configuration, we ran the calculations of the scores thirty times. The results of this simulation are depicted in Figure 6.5. The error bars display the standard deviation. The

x-axis characterizes the number of leaf-*softgoals* and the different lines characterize the number of *operationalizations* of each leaf-*softgoal*. The time to calculate all the equations (in seconds) related to *softgoals* scores, leaf-*softgoals* scores and *operationalizations* scores are depicted in the y-axis in a logarithmic scale. As we can notice, the execution time increases as we increase the number of *operationalizations* for each leaf-*softgoal*. This execution time stays below 1 second until the number of leaf-*softgoals* reaches 64 and the number of *operationalizations* attached to them reaches 64 as well. With this SIG configuration, the time to calculate the scores of the total amount of nodes ($64 \times 64 = 4096$) reaches approximately 3 seconds. In `INSpIRE`, we consider an acceptable execution time of less than 1 second. Therefore, the number of nodes in a SIG scale up to 2048 nodes without affecting the overall `INSpIRE` performance.

7 CONCLUDING REMARKS

This dissertation has presented INSpIRE, which is an Intent-based Networking (IBN) solution that refines *intents* into service chains of VNFs by employing a technique based on *Softgoal* Interdependency Graphs (SIGs) and clustering. INSpIRE is able to automatically calculate and attribute scores to meet the non-functional requirements of a VNF. In addition, it draws on these scores to cluster and select the appropriate VNFs to fulfill a defined *intent*. Thus, INSpIRE adopts an approach that involves solving the refinement of *intents* into service chains.

7.1 Summary of Contributions

We have defined a CNL to allow business-level operators to write *intents* in a Web-based interface. These *intents* guide the construction and the enforcement of service chains in the network. In our approach, the employment of IBN with the aid of a CNL to dynamically write service chaining graphs, reduces the need from network operators to have previous knowledge, and thus reduces OPEX. Additionally, combining NFV and SDN to build the infrastructure and virtualize the related elements, adds elasticity and at the same time, lowers CAPEX. Furthermore, when selecting the network functions that will form the chain, they must be characterized on the basis of their non-functional requirements. We have proposed the use of the NFR Framework to provide this characterization. The NFR Framework employs *Softgoal* Interdependency Graphs to express non-functional requirements while the software is being developed; each non-functional requirement is influenced (either positively or negatively) by different softgoals, and operationalizations. We have adopted this approach and quantified each non-functional requirement, on the basis of previous work in the literature, so that the network functions can be grouped with similar non-functional requirements values. It is worth mentioning that the graphs are designed by drawing on empirical knowledge from the network operators, and business-level operators, and thus vary from scenario to scenario.

In addition, we provided a case study and simulations of the components of INSpIRE to validate its feasibility. The case study examines the interaction of the users (stakeholders and network operators) during the different stages of INSpIRE (SIG modeling, insertion of VNFs, *intent* writing, and the refinement process). We also included three case studies to validate the feasibility of our solution in translating *intents*. The case studies describe different *intents* and the outputs provided by INSpIRE within various network management situations. The first case study describes an academic network shared by students, teachers and staff. The second examines a company network that is divided into departments and branches. The third sets out a scenario where our solution helps operators to monetize their deployment of middleboxes. The monetization of service chains is achieved by implementing VNF service chaining as a service, by allowing a business-level operator to set priority classes for different customers, and thus

setting different prices to each class.

A set of simulations of the clustering and score calculation components have been carried out as a means of evaluating INSpIRE. These are generally time consuming components that are carried out during the whole refinement process due to the complexity of the algorithms implemented in them. However, the results showed that INSpIRE is able to work in small scenarios (with hundred of VNFs and SIGs containing 128 nodes) and large scenarios (tens of thousands of VNFs and SIGs containing 2048 nodes) as well. It can thus be concluded that INSpIRE is versatile and can be employed in different scenarios.

On the basis of our findings, it has been demonstrated that INSpIRE achieves the three main objectives outlined in the introduction of this dissertation:

(i) **Determining the specific VNFs required to fulfill an *intent*:** INSpIRE determines the specific VNFs required to fulfill an *intent* by employing the use of *Softgoal* Interdependency Graphs to calculate the values needed to select the correct VNFs by means of clustering techniques.

(ii) **Pre-chaining the required VNFs according to their dependencies:** INSpIRE employs the number of *operationalizations* (*i.e.* the functions that a network function is capable of performing) to determine where the network function will be placed in the chain.

(iii) **Providing enough low-level information to network devices for posterior traffic steering:** INSpIRE refines *intents* into JSON objects with the maximum low-level information available. In addition, INSpIRE publishes a REST API that contains functions to help external traffic steering solutions in the retrieval of the chains.

7.2 Final Remarks and Future Work

It should be noted that the act of defining a service chain is a subjective process – one service chain might be optimal for a specific infrastructure but poor for another. In view of this, INSpIRE is far from providing a perfect and well-tailored service chaining creation solution that can be applied to every situation. Nevertheless, we hope that our work can shed light on the complexity of the question and enable future research to characterize network functions, VNFs, and middleboxes on the basis of the functions that they carry out. Moreover, as computer network management solutions are moving towards needing the least human intervention as possible, the research on IBN, and the refinement of *intents* is indispensable to provide more autonomous solutions.

In future work, we plan to provide a complete solution, by extending INSpIRE so that it can be integrated with a consolidated NFV framework, such as the ones outlined in Section 3.1. We also intend to model a comprehensive SIG to represent other non-functional requirements, such as Integrity, Availability, and Performance. In addition, we seek to conduct a qualitative

evaluation of INSpIRE, *i.e.* to assess its degree of accuracy when providing the service chain and enforcing it in the network. With regard to the refinement process, INSpIRE does not take account of the feedback from the service chain when the graphs are already saved in the database. In light of this, there is a need to design a tool to monitor the service chains and provide the related feedback, *e.g.* high latency, slow VNF performance, and incorrect chaining. In addition, INSpIRE only selects network functions and pre-orders them. However, there can be cases when only one network function is chosen on the basis of the contexts of the intent. In this case, a network operator can choose to add more network functions to the chain. Thus, it is important to allow the network operator to decide whether to adjust the chain, and add more functions to it.

Finally, to extend INSpIRE further and provide a consolidated prototype, it is recommended that future work should include: (*i*) integrating with a traffic steering solution, (*ii*) testing our solution in production scenarios, (*iii*) improving the VNF ordering process; and (*iv*) implementing additional features in the Web-based interface.

REFERENCES

AFFLECK, A.; KRISHNA, A. Supporting Quantitative Reasoning of Non-functional Requirements: A Process-oriented Approach. In: **Proceedings of the International Conference on Software and System Process**. [S.l.: s.n.], 2012. (ICSSP '12), p. 88–92.

BANDARA, A. K. et al. A Goal-based Approach to Policy Refinement. In: **Proceedings of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks**. Washington, DC, USA: IEEE Computer Society, 2004. (POLICY '04), p. 229–.

BATISTA, B.; CAMPOS, G. de; FERNANDEZ, M. A proposal of policy based OpenFlow network management. In: **Telecommunications (ICT), 2013 20th International Conference on**. [S.l.: s.n.], 2013. p. 1–5.

BEHRINGER, M. et al. **Autonomic Networking: Definitions and Design Goals**. [S.l.]: IETF, 2015. RFC 7575 (Informational). (Request for Comments, 7575). Available at <https://tools.ietf.org/html/rfc7575> Accessed April 12, 2017.

BLENDIN, J. et al. Position Paper: Software-Defined Network Service Chaining. In: **Software Defined Networks (EWSDN), 2014 Third European Workshop on**. [S.l.: s.n.], 2014. p. 109–114.

CASE, J. D. et al. **Simple Network Management Protocol (SNMP)**. [S.l.], 1990. Available at <https://www.ietf.org/rfc/rfc1157.txt> Accessed April 12, 2017.

CHUNG, L.; LEITE, J. C. P. In: ____. **Conceptual Modeling: Foundations and Applications**. 1. ed. Berlin, Heidelberg: Springer-Verlag, 2009. chp. On Non-Functional Requirements in Software Engineering, p. 363–379.

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

CRAVEN, R. et al. Policy Refinement: Decomposition and Operationalization for Dynamic Domains. In: **Network and Service Management (CNSM), 2011 7th International Conference on**. [S.l.: s.n.], 2011. p. 1–9. Available at <http://ieeexplore.ieee.org/document/6103981/> Accessed April 12, 2017.

CSOMA, A. et al. ESCAPE: Extensible Service Chain Prototyping Environment Using Mininet, Click, NETCONF and POX. In: **Proceedings of the 2014 ACM Conference on SIGCOMM**. New York, NY, USA: ACM, 2014. (SIGCOMM '14), p. 125–126.

DAMIANOU, N. et al. The Ponder Policy Specification Language. In: **Proceedings of the International Workshop on Policies for Distributed Systems and Networks**. London, UK, UK: Springer-Verlag, 2001. (POLICY '01), p. 18–38.

DING, W. et al. OpenSCaaS: an open service chain as a service platform toward the integration of SDN and NFV. **Network, IEEE**, v. 29, n. 3, p. 30–35, May 2015.

ETSI. **Network Functions Virtualisation (NFV)**. 2012. White Paper. Available at <https://portal.etsi.org/nfv/nfv_white_paper.pdf> Accessed 15 October, 2015.

European Telecommunications Standards Institute (ETSI). **ETSI Group Specification Network Functions Virtualisation (NFV); Management and Orchestration**. 2014. Deliverable 1.1.1. Available at <http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_NFV-MAN001v010101p.pdf> Accessed December 17, 2015.

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.

FOSTER, N. et al. Frenetic: A Network Programming Language. In: **Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming**. New York, NY, USA: ACM, 2011. (ICFP '11), p. 279–291.

FRANCO, M. F. et al. VISION – Interactive and Selective Visualization for Management of NFV-Enabled Networks. In: **2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA)**. [S.l.: s.n.], 2016. p. 274–281.

GE, X. et al. OpenANFV: Accelerating Network Function Virtualization with a Consolidated Framework in Openstack. In: **Proceedings of the 2014 ACM Conference on SIGCOMM**. New York, NY, USA: ACM, 2014. (SIGCOMM '14), p. 353–354.

GEMBER-JACOBSON, A. et al. OpenNF: Enabling Innovation in Network Function Control. In: **Proceedings of the 2014 ACM Conference on SIGCOMM**. New York, NY, USA: ACM, 2014. (SIGCOMM '14), p. 163–174.

HAGBERG, A. A.; SCHULT, D. A.; SWART, P. J. Exploring Network Structure, Dynamics, and Function using NetworkX. In: VAROQUAUX, G.; VAUGHT, T.; MILLMAN, J. (Ed.). **Proceedings of the 7th Python in Science Conference**. Pasadena, CA USA: [s.n.], 2008. p. 11 – 15. Available at <http://math.lanl.gov/~hagberg/Papers/hagberg-2008-exploring.pdf> Accessed April 12, 2017.

HALPERN, J.; PIGNATARO, C. **Service Function Chaining (SFC) Architecture**. [S.l.]: IETF, 2015. RFC 7665 (Informational). (Request for Comments, 7665). Available at <https://tools.ietf.org/html/rfc7665> Accessed April 12, 2017.

JOHN, W. et al. Research Directions in Network Service Chaining. In: **Future Networks and Services (SDN4FNS), 2013 IEEE SDN for**. [S.l.: s.n.], 2013. p. 1–7.

JONES, E. et al. **SciPy: Open source scientific tools for Python**. 2001–. Available at <http://www.scipy.org/> Accessed April 12, 2017.

KUHN, T. A Survey and Classification of Controlled Natural Languages. **Comput. Linguist.**, MIT Press, Cambridge, MA, USA, v. 40, n. 1, p. 121–170, mar. 2014.

LANTZ, B.; HELLER, B.; MCKEOWN, N. A Network in a Laptop: Rapid Prototyping for Software-defined Networks. In: **Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks**. New York, NY, USA: ACM, 2010. (Hotnets-IX), p. 19:1–19:6.

Linux Foundation. **Open Platform for NFV**. 2015. Available at <https://www.opnfv.org> Accessed October 18, 2017.

MACHADO, C. C. et al. An EC-based Formalism for Policy Refinement in Software-Defined Networking. In: **2015 IEEE Symposium on Computers and Communication (ISCC)**. [S.l.: s.n.], 2015. p. 496–501.

MACHADO, C. C. et al. Policy authoring for software-defined networking management. In: **Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on**. [S.l.: s.n.], 2015. p. 216–224.

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. Available at <http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.308.8619> Accessed April 12, 2017.

MARTINS, J. et al. ClickOS and the Art of Network Function Virtualization. In: **Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation**. Berkeley, CA, USA: USENIX Association, 2014. (NSDI'14), p. 459–473.

MCKEOWN, N. et al. OpenFlow: Enabling Innovation in Campus Networks. In: . New York, NY, USA: ACM, 2008. v. 38, n. 2, p. 69–74.

MOFFETT, J. D.; SLOMAN, M. S. Policy Hierarchies for Distributed Systems Management. In: . [S.l.: s.n.], 1993. v. 11, n. 9, p. 1404–1414.

MOORE, B. et al. **Policy Core Information Model**. [S.l.], 2001. 1-100 p. Available at <https://www.rfc-editor.org/rfc/rfc3060.txt> Accessed April 12, 2017.

NeMo Project Team. **An Application's Interface to Intent Based Networks**. 2015. Available at <http://nemo-project.net/> Accessed March 08, 2017.

ONOS Project. **ONOS Intent Framework**. 2014. Available at <https://wiki.onosproject.org/display/ONOS/Intent+Framework> Accessed March 08, 2017.

PFITSCHER, R. J. et al. DReAM - a Distributed Result-Aware Monitor for Network Functions Virtualization. In: **2016 IEEE Symposium on Computers and Communication (ISCC)**. [S.l.: s.n.], 2016. p. 663–668.

QAZI, Z. A. et al. SIMPLE-fying Middlebox Policy Enforcement Using SDN. In: **Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM**. New York, NY, USA: ACM, 2013. (SIGCOMM '13), p. 27–38.

QUINN, P.; ELZUR, U. **Network Service Header**. [S.l.], 2015. Available at <https://tools.ietf.org/html/draft-ietf-sfc-nsh-12> Accessed April 12, 2017.

QUINN, P.; NADEAU, T. **Problem Statement for Service Function Chaining**. [S.l.]: IETF, 2015. RFC 7498 (Informational). (Request for Comments, 7498).

RUBIO-LOYOLA, J. et al. A Functional Solution for Goal-Oriented Policy Refinement. In: IEEE. **Policies for Distributed Systems and Networks, 2006. Policy 2006. Seventh IEEE International Workshop on**. [S.l.], 2006. p. 133–144.

SCARFONE, K. A.; MELL, P. M. **SP 800-94. Guide to Intrusion Detection and Prevention Systems (IDPS)**. Gaithersburg, MD, United States, 2007. Available at <http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-94.pdf> Accessed April 12, 2017.

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.

SCHEID, E. J. et al. Policy-based dynamic service chaining in Network Functions Virtualization. In: **2016 IEEE Symposium on Computers and Communication (ISCC)**. [S.l.: s.n.], 2016. p. 340–345.

SEFRAOUI, O.; AISSAOUI, M.; ELEULDJ, M. OpenStack: Toward an Open-source Solution for Cloud Computing. **International Journal of Computer Applications**, v. 55, n. 3, p. 38–42, October 2012. Available at <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.245.229&rep=rep1&type=pdf>, Accessed at October 15, 2015.

STRASSNER, J. **Policy-Based Network Management: Solutions for the Next Generation (The Morgan Kaufmann Series in Networking)**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.

Unify Project. **UNIFY Deliverable 2.2 Final Architecture**. 2014. Deliverable 2.2. Available at <https://www.fp7-unify.eu/files/fp7-unify-eu-docs/Results/Deliverables/UNIFY%20Deliverable%202.2%20Final%20Architecture.pdf> Accessed September 11, 2015.

USZOK, A. et al. KAoS Policy and Domain Services: Toward a Description-Logic Approach to Policy Representation, Deconfliction, and Enforcement. In: **Policies for Distributed Systems and Networks, 2003. Proceedings. POLICY 2003. IEEE 4th International Workshop on**. [S.l.: s.n.], 2003. p. 93–96.

VERMA, D. C. Simplifying Network Administration Using Policy-based Management. **IEEE Network: The Magazine of Global Internetworking**, IEEE Press, Piscataway, NJ, USA, v. 16, n. 2, p. 20–26, mar. 2002.

VERMA, D. C. **Principles of Computer Systems and Network Management**. 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2009.

ZHANG, Y. et al. StEERING: A Software-Defined Networking for Inline Service Chaining. In: **Network Protocols (ICNP), 2013 21st IEEE International Conference on**. [S.l.: s.n.], 2013. p. 1–10.

AppendixA PUBLISHED PAPER – ISCC 2016

Eder John Scheid, Cristian Cleder Machado, Ricaro Luis dos Santos, Alberto Egon Schaeffer-Filho and Lisandro Zambenedetti Granville. **Policy-based dynamic service chaining in Network Functions Virtualization**. 2016 IEEE Symposium on Computers and Communication (ISCC), Messina, 2016, pp. 340-345. DOI: 10.1109/ISCC.2016.7543763

- **Title:** *Policy-based Dynamic Service Chaining in Network Functions Virtualization*.
- **Contribution:** An architecture to translate high-level policies into dynamic service chains in NFV.
- **Abstract:** Network Functions Virtualization (NFV) enables the rapid development, flexible management, and the dynamic placement of new, innovative Virtualized Network Functions (VNFs), such as load balancers, firewalls, and Intrusion Detection Systems (IDSes). Furthermore, NFV along with Software-Defined Networking (SDN) allows VNFs and physical middleboxes to be dynamically composed into service chaining graphs. Despite these benefits, service chaining graphs can be further improved through the use of techniques that have not been satisfactorily explored yet, such as Policy-Based Network Management (PBNM). In PBNM, policies can be written and triggered during runtime, thus supporting the dynamic (re)configuration of service graphs with minimal disruption. In this paper, we propose an approach to automatically design NFV service chaining graphs based on policies. These policies rule the forwarding of traffic and the construction of service chaining graphs. In our approach, service chaining graphs are enforced dynamically in the network during runtime. Finally, to assess its feasibility and generality, we create two different scenarios to demonstrate and discuss how our solution can be employed and its expected results.
- **Status:** Published.
- **Qualis:** A2.
- **Conference:** 21st IEEE Symposium on Computers and Communication (ISCC).
- **Date:** June 27 - June 30, 2016.
- **Local:** Messina, Italy.
- **URL:** <http://iscc2016.unime.it/>.
- **Digital Object Identifier (DOI):** <http://dx.doi.org/10.1109/ISCC.2016.7543763>.

Policy-Based Dynamic Service Chaining in Network Functions Virtualization

Eder J. Scheid, Cristian C. Machado, Ricardo L. dos Santos, Alberto E. Schaeffer-Filho, Lisandro Z. Granville
Institute of Informatics - Federal University of Rio Grande do Sul
Porto Alegre, RS, Brazil
Email: {ejscheid, ccmachado, rlsantos, alberto, granville}@inf.ufrgs.br

***Abstract*—Network Functions Virtualization (NFV) enables the rapid development, flexible management, and the dynamic placement of new, innovative Virtualized Network Functions (VNFs), such as load balancers, firewalls, and Intrusion Detection Systems (IDSes). Furthermore, NFV along with Software-Defined Networking (SDN) allows VNFs and physical middleboxes to be dynamically composed into service chaining graphs. Despite these benefits, service chaining graphs can be further improved through the use of techniques that have not been satisfactorily explored yet, such as Policy-Based Network Management (PBNM). In PBNM, policies can be written and triggered during runtime, thus supporting the dynamic (re)configuration of service graphs with minimal disruption. In this paper, we propose an approach to automatically design NFV service chaining graphs based on policies. These policies rule the forwarding of traffic and the construction of service chaining graphs. In our approach, service chaining graphs are enforced dynamically in the network during runtime. Finally, to assess its feasibility and generality, we create two different scenarios to demonstrate and discuss how our solution can be employed and its expected results.**

***Index Terms*—Policy-based Network Management; Network Functions Virtualization; Service Chaining.**

I. Introduction

Network functions, such as load balancing, firewalls, and Intrusion Detection Systems (IDSes) are traditionally realized in physical devices often refereed to as middleboxes. Middleboxes tend to be proprietary and vendor-specific, and thus force network operators to learn about the peculiarities of middleboxes from different vendors, which is counterproductive. Also, physical middleboxes are not flexible enough to accommodate bursts of demand, which intrinsically hinders their scalability. Network Functions Virtualization (NFV) [1] is a novel technology that addresses the lack of flexibility of physical middleboxes. NFV proposes the use of Commercial Off-The-Shelf (COTS) hardware to host virtualized network services. With this approach, the capital expenditure (CAPEX) and operational expenditure (OPEX) can be significantly reduced. Also, with NFV, service provisioning can be easily scaled up and down according to network demands.

NFV allows the chaining of multiple Virtualized Network Functions (VNFs). Such VNF chaining enables network operators to dictate to which sequence of VNFs a packet should go through. The act of specifying the sequence of VNFs is called *network service chaining* [2]. Service chaining on current network infrastructures is statically defined and dependent on the network's topology. This imposes a challenge to the operator when adding or removing services, considering that earlier technologies are difficult to redeploy [3]. With NFV and Software-Defined Networking (SDN) [4], this chaining can be performed dynamically. SDN decouples the control plane from the data plane, providing a global view of the network and a controller that performs traffic forwarding decisions [4]. With this separation, a controller can be implemented to steer the traffic dynamically during runtime. Therefore, service chaining can be easily adapted to the administrator's need. This chaining is created from existent VNFs and middleboxes, thus using network resources efficiently [5].

Network operators have different needs according to the traffic of the networks that they manage. Also, network users do not necessarily need the same services (*e.g.,* packets exchanged inside the enterprise's network can pass through a simple firewall instead of a more sophisticated one). From these premises, a question emerges: how can the operator dynamically compose a set of VNFs to handle customized traffic flows? An approach to solving this problem is to use policies to govern the service chain of a flow. Policy-Based Network Management (PBNM) in computer networks is a concept widely applied and well-defined [6], but as long as the authors of this paper are aware of, its use in NFV service chaining has not been exploited.

In this paper, we present a PBNM solution to design and manage service chaining, where business-level operators can write Service Level Agreements (SLAs) to guide the building of service chaining graphs. We also introduce a Controlled Natural Language (CNL) to establish requirements and constraints for the writing of policies. Further, we discuss our solution's feasibility and generality in two different scenarios. Our proposed solution can be employed in both homogeneous environments (VNFs only) and heterogeneous environments (composed of both VNFs and physical middleboxes).

This paper is structured as follows. In Section II, we review work related to this approach. Then, in Section III, the solution and associated architecture are described. In Section IV, two case studies are outlined and discussed. Finally, in Section V, we finish this paper with conclusions and future work.

II. Related Work

PBNM is a concept already widely employed and studied for years. However, with the recent scientific and industrial interest in both NFV and SDN, this concept is emerging back.

Moreover, some high-level languages for programming OpenFlow networks (*e.g.,* Frenetic [7]), simplify the steering and the classification of traffic by abstracting packet-forwarding policies and modularizing components. Thus, promoting the use of PBNM approaches in OpenFlow networks.

Machado *et al.* [8] propose to manage an SDN environment with minimal changes in the controller implementation. To achieve this, the authors introduce a framework that translates, in the work's scope, Quality-of-Service (QoS) policies into a set of OpenFlow rules. The work aims to reduce the complexity of management tasks and to enable the writing of high-level policies by using a CNL. Nevertheless, the authors do not address traffic steering policies nor NFV.

In the service chaining area, the Internet Engineering Task Force (IETF) has described an architecture for the development of Service Function Chains (SFCs) [9]. In addition, the Network Service Headers (NSH) [10] is an approach that introduces a new header in packets traveling through services instances. This header is intended to help the separation of traffic. However, the addition of this header increases the traffic processing time, which may cause delays or even packet loss.

Qazi *et al.* [11] proposed SIMPLE, a solution that relies on SDN to provide middlebox traffic steering. SIMPLE introduces a policy enforcement layer which translates user-defined policies into OpenFlow rules and track packets that had their headers modified by service instances. This solution addresses the service chaining problem with SDN but does not take into consideration if the middleboxes are deployed as VNFs, consequently, not leveraging NFV's flexibility and scalability.

Likewise, Csoma *et al.* [12] introduced ESCAPE, a prototyping framework that allows the developer to test and chain customized VNFs in an SDN environment. ESCAPE employs some consolidated tools, such as Mininet and ClickOS, providing a strong basis to develop and evaluate different types of NFV and SDN solutions. Although allowing the developer to compose any VNF chain, the prototype does not support the modification of this chain during runtime.

Several solutions have been proposed in the NFV and SDN area [13] [14] [15] [16] (due to space constraints they are not detailed in this paper). Despite the efforts employed by the authors to provide these solutions, such solutions have some important shortcomings. For example, the synergy between SDN and NFV that introduces the possibility to write high-level rules to guide the composition of service chains is not considered. In addition, the mechanisms to analyze low-level rules in order to provide richer data to guide this composition are not covered. Therefore, we pursued to cover all these aspects during the development of our solution.

III. Policy-Based Dynamic Service Chaining

We present an approach to enable network operators to write management policies that govern the chaining of VNFs. Based on the set of available VNFs in the infrastructure, our proposed framework creates a graph that represents the Service Chaining (SC). This graph is then used by an SDN controller to perform the traffic steering. The primary objective of our solution is to ease the tasks of network operators when specifying service chains and also decouple the need of expressing how low-level configurations must be implemented in the infrastructure. Therefore, is not under the scope of our solution how the controller will perform the steering of traffic.

A. Controlled Natural Language

The syntax of many policy languages often resembles the syntax of traditional programming languages, which is the case of Ponder [17]. This approach requires the network operator to have a prior knowledge of the language and to translate SLAs into a particular format. On the other hand, with the employment of CNLs [18] to write policy languages, network operators can write SLAs in (a subset of) English, which diminish the need for prior specific knowledge. Machado *et al.* [8] proves the feasibility of using a CNL to write SLAs that are translated to QoS rules and then enforced in the network elements. Given this premise, we present a CNL to write rules for the creation of service chaining graphs. The grammar of the proposed CNL is presented in the Listing 1.

Listing 1: Proposed CNL grammar

```
1 Language: →<Service><Flow><Preposition><Expression>
2 Service: →service-regexes
3 Flow: →<Direction><Target><Direction><Target>
4 Direction: → From|To
5 Target: →user-defined-regexes
6 Preposition: →Have
7 Expression: →<Term>|<Term><Connective><Expression>
8 Term: →<Adjective><Context>
9 Adjective: →adjective-regexes
10 Context: →context-regexes
11 Connective: →And|Or
```

As the policy language is defined as a CNL, in order to identify strings that compose a policy in a solid way we defined a set of regular expressions. We have classified these regular expressions into four main types according their purposes: *(i) service-regexes* to identify the type of service; *(ii) user-defined-regexes* that are set by the user; *(iii) adjective-regexes* to identify the level of requirement; and *(iv) context-regexes* to identify the context of the policy. Some examples of regular expressions are presented in Table I.

TABLE I: Regular Expressions Examples

Type	*Expression*
service-regexes	HTTP, SMTP, FTP, VoIP...
user-defined-regexes	teachers, staff, Internet...
adjective-regexes	none, low, medium, high...
context-regexes	inspection, perfomance, resiliency...

B. Architectural Model

In this section, we propose a solution that permits the writing of SLAs which are an abstraction of a service chain. This means that the network operator only specifies the level of context enforcement a flow must have. Thus, a flow has to pass through the respective service chain that complies with the stored policies.

We present in Figure 1 a high-level perspective of the conceptual model of our solution. The components characterized with a dashed line are existing solutions and were not implemented, given that our solution is generic enough and can be applied on top of them. In the next sections, we present a more in-depth description of the layers and its components.

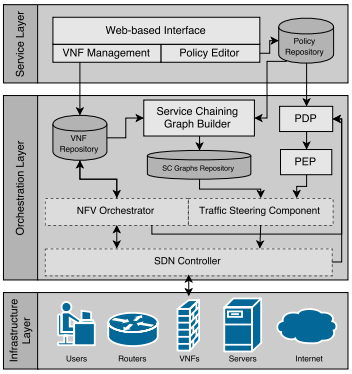


Fig. 1: Proposed System Architecture

1) Service Layer: The Service Layer comprises: (*i*) traditional lifecycle management functions, such as Operation Support Systems (OSS) and Business Support Systems (BSS), (*ii*) virtualization-related management functions, such as lifecycle management for VNFs; and (*iii*) adaptation functions toward lower layer. As the management functions of the service layer must not depend on the infrastructure of the network, their implementation is generic. They are described below.

- *Web-based Interface*: This component acts as a frontend that allows the operator to interact with the VNF Management and the Policy Editor using a user-friendly interface. With the utilization of a Graphical User Interface (GUI), the interaction with the system is simplified.
- *VNF Management*: In order for the system to recognize the available VNFs in the infrastructure, infrastructure-level operators must inform their description and details. To manage this information (create, remove and update VNFs) a set of functions are accessible using the operator's account to login in the web-based interface. The NFV ETSI Industry Specification Group (ISG) formalizes the information that a VNF should contain [19]. This information is stored in the VNF Descriptor (`vnfd`), which contains elements regarding requirements of the deployment and operation of VNFs, such as the number of virtual CPUs (`computation_requirement`), the amount of virtual network bandwidth needed (`virtual_network_bandwith_resource`) and the version of the VNF software.
- *Policy Editor*: This component allows business-level operators to create, retrieve, update and delete policies. Operators can also enable or disable policies accordingly to their needs. The aforementioned CNL is used to input the policies. Also, the Policy Editor has to parse and match the given CNL with low-level rules.
- *Policy Repository*: Policies wrote by business-level operators are stored in this component. This repository is accessed by the Policy Decision Point (PDP) and the Service Chaining Graph Builder to design the graphs.

2) Orchestration Layer: This layer comprises two components, the resource orchestrator, and the controller adapter, which are embedded in this layer and are not depict in architecture. The first is composed of virtualizers, policy enforcement and orchestration with underlying resources. The latter comprises resource abstraction functions and virtualization for different technologies. This layer is in charge of all the infrastructure management and networking control, being the main layer of our solution. Within these two components, we place a set of elements; they are detailed below.

- *VNF Repository*: This component holds information about the VNFs informed by the infrastructure-level operator in the GUI. The ETSI defines that once an instance of a VNF is deployed, a VNF Record (`vnfr`) is created. This record (*e.g.,* IP Address (`vnf_address`), type of service, and status) is updated during the lifecycle of the respective VNF by the NFV Orchestrator.
- *Service Chaining Graph Builder*: This component access the policies wrote by business-level operators and automatically creates service chaining graphs based on those policies. In addition, to have sufficient information to create these graphs, this element retrieves the available VNFs from the VNF Repository. Once it retrieved the detailed information about the policies and VNFs, the graphs are created and stored or updated in a repository.
- *SC Graphs Repository*: This repository stores the Service Chaining graphs created by the builder. The stored tuple consists of a policy and its respective service chain.
- *Policy Decision Point (PDP)*: This component determines which policy is going to be enforced based on the information given by the SDN Controller and the NFV orchestrator. It decides which policy matches with a flow informed by the SDN Controller given the stored policies in the Policy Repository. Information about the network and VNFs are constantly fed to this component.
- *Policy Enforcement Point (PEP)*: This component informs the Traffic Steering Component to access the respective graph in the SC Graphs Repository and install the rules.
- *Traffic Steering Component*: This component communicates with the SDN controller and steers the flows through the desired set of VNFs based on the defined service chaining graph. This steering is ruled by the policies enforced by the PEP. Thus, when a policy is enforced, this component access the SC Graphs Repository to retrieve the graph and informs the SDN controller of the rules that must be installed in the switches.

3) Infrastructure Layer: Within this layer are compromised all the physical resources, and controllers. Resources are composed of machines containing compute, storage and network resources and their respective managers. In our model users, VNFs, routers, and servers compose this layer.

C. Policy Translation

The process of translating SLAs into service chains comprises three phases, performed by the *Policy Editor* (Phase 1 and 2) and *Service Chaining Graph Builder* (Phase 3).

1) Phase 1 - Policy Validation: To validate an SLA written by an operator, the *Policy Editor* has to parse this SLA into a set of defined regular expressions, as depict in Figure 2. The Policy Editor iterates over the string to find *service-regexes* first, then it moves to *user-defined-regexes* that specifies the source and destination of the flow, and finally, it searches for the *adjective-regexes* followed by the *context-regexes*. If the parser encounters an error in any part of the parsing process, the SLA is marked as invalid and is not stored in the database. In the web-based interface, the operator receives an error message. In addition, there is an option that allows the operator to validate the SLA before committing it to the database.

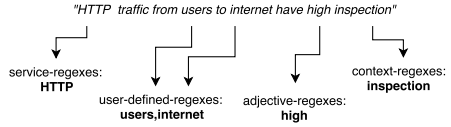


Fig. 2: Valid SLA Policy Parsing Example

2) Phase 2 - Conflict Detection: The primary focus of this paper is not to resolve conflicts among policies. However, our system estimates some conflicting policies at the insertion. Then, the GUI displays this information to the current operator, who must write a new nonconflicting policy. Conflicts can vary from already defined policies to priority conflicts, such as the inclusion of two equal policies but with different *adjective-regex* will trigger a priority conflict. For example, if an operator writes an SLA *"HTTP traffic from teachers to students have high inspection"* and later tries to insert another SLA informing *"HTTP traffic from teachers to students have none inspection"*, the system will notify the operator of the conflict, which in this case is the same *service-regex* (HTTP), same *user-defined-regexes* (teachers and students) and different *adjective-regexes* (high and none) for the same *context-regex* (inspection). Next, after the notification, the operator must resolve the conflicting SLAs.

3) Phase 3 - Service Chain Graph Construction: Once policies are defined and inserted into the database, the *Service Chaining Graph Builder* component has to create the desired service chains. To construct these service chains, the builder must have knowledge of what VNFs are available for composing the graphs. It does that by accessing the VNF Repository and retrieving the information stored at the VNF descriptors or records. This process of retrieval is guided by the *context-regexes* (*e.g.,* if the *context-regex="inspection"*) the builder only retrieves VNFs related with security or inspection, which minimizes the amount of VNF information to process.

After retrieving the VNFs, the SC builder sets a chaining threshold for the highest level related to the context. This threshold is based on the available VNFs. For instance, a graph composed of a firewall, an intrusion detection system, and a deep packet inspection may represent the highest "inspection level" for an infrastructure with those three VNFs available. In a first moment, this threshold is a suggestion based on predefined thresholds; the infrastructure-level operator can add or remove VNFs as required. After setting this threshold, the *adjective-regex* is examined to determine the desired context level of the to-be-constructed service chain. If the desired context level is *high*, the service chain is set to the threshold, which is the case of *adjective-regex* depict in Figure 2. Otherwise, the builder conducts the removal of virtualized functions until it reaches the desired level.

In Table II some examples of context levels and its equivalent service chain are represented. It also represents the different chaining possibilities given a set of VNFs. The "$\rightarrow$" character represents an edge in the graph, and the functions are those defined as present in the infrastructure by infrastructure-level operators. The resultant graph is composed of network functions (nodes) and links (edges); the sequence of the network functions is determined based on the "`vnf_depedency`" element present in the ETSI Network Service Descriptor (`nsd`). This element describes the dependency among VNFs and informs which source VNF must exist before a target VNF is deployed. The information stored in the node is a pointer to the desired VNF in the repository and in the edges are stored only linking information, such as the source's and destination's port.

Lastly, the *Service Chaining Graph Builder* inserts the tuple `<policy,graph>` in the *SC Graph Repository* for posterior access by the *Traffic Steering Component*.

TABLE II: Service Chains and Context Levels Examples

context-regex	*adjective-regex*	*Graph Builder Output*
inspection	*high*	Firewall $\rightarrow$ DPI $\rightarrow$ IPS
	medium	Firewall $\rightarrow$ DPI
	low	Firewall

D. Traffic Classification and Steering

To properly steer the traffic through the set of desired VNFs, the incoming flow of packets must be classified. This classification is performed by the PDP, in which the *services-regexes* and the *user-defined-regexes* are employed to match with the policies stored at the *Policy Repository*. The former is used to classify the type of service of the current flow. The latter defines the source and destination of the graph (*e.g., "from **teachers** to **Internet**"*). This information is then utilized to retrieve the matching policy from the *Policy Repository*.

Having retrieved the policy, the PDP forwards the information and the policy to the PEP to enforce it. This act of enforcing is performed by informing this policy to the *Traffic Steering Component* so that it can retrieve the matching SC graph from the *SC Graph Repository*.

Considering that SDN helps to address the problem of dynamic steering of traffic, with the use of OpenFlow [20] enabled switches and routers; and that OpenFlow rules are

installed and expired during runtime, SDN was elected as the networking paradigm to compose the solution. With the use of this approach, the network becomes more flexible and more manageable, as service graphs can be modified during runtime.

TABLE III: User Domains and Respective IP Ranges

Scenario 1		Scenario 2	
User Domain	IP Range	User Domain	IP Range
human-resources	154.15.2.0/24	*platinum*	135.98.1.0/24
accounting	154.15.3.0/24	*diamond*	135.98.5.0/24
development	154.15.4.0/24	*gold*	135.98.10.0/24
directory	154.15.5.0/24	*silver*	135.98.15.0/24
marketing	154.15.7.0/24	*bronze*	135.98.25.0/24

IV. Case Studies

To provide an evaluation of the feasibility of our solution, we describe its implementation in two different scenarios that serve as case studies. The choice of these scenarios was based on two premises: the various levels of hierarchy present in the organization and the presence of heterogeneous traffic in the network. As an example of the level hierarchy, we can state that in the case of VNF Service Chain as a Service, *"gold users"* have a higher priority than *"silver users"* and so on. Heterogenous traffic is described as the presence of traffic such as different internet protocols competing for a network share.

A. Scenario 1 - Common Enterprise Network

Let us consider the case of a generic enterprise, with different departments, such as Marketing, Human Resources, Directory, and among others. These departments have different network requisites due to the diversity of applications in each of them. For example, the need for a high level of inspection on financial transactions originated from the Marketing department or a strong security between the communication of two branches. Thus, the organization's network business-level operator and infrastructure-level operators must guarantee that these requirements are fulfilled. In order to comply with these requirements, infrastructure-level operators can define different IP ranges for the departments, an example of user domains can be found in column *Scenario 1* in Table III. Moreover, business-level operators can define sets of policies for the traffic traveling on the organization's network.

As an example, we will define an enterprise with the user domains presented in Table III, column *Scenario 1*. This scenario is depicted in Figure 3. The board of directors may hold weekly meetings with the human resources department. In order to eliminate the need of physical presence in the conference room, the participants can use VoIP calls to attend the meeting. This possibility introduces the VoIP QoS requirement. To address this requirement, one could write an SLA of the type *"VoIP traffic from human-resources to directory have high performance"* and only activate this SLA once a week, during the meeting. This traffic will then be steered to the set of middleboxes imposed by our solution, guaranteeing a good VoIP user experience for the participants, dynamically using network resources (solid red line). If we consider an organization with more than one branch, infrastructure-level operators can detail branch domains and user domains for these branches and a business-level operator can write SLAs accordingly, *e.g., "HTTP traffic from* $branch_1$ *to* $branch_3$ *have low security"* (blue dashed line) and so on.

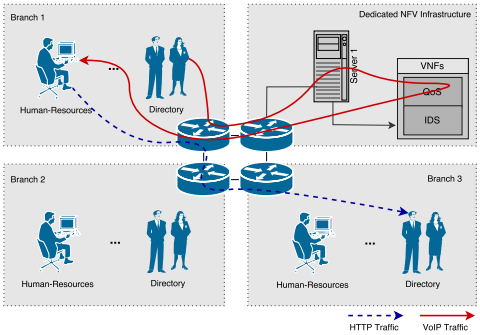


Fig. 3: Traffic Flows in a Common Enterprise Network

This example aims to provide a picture of how can our approach be used in organizations with multiple departments. The CNL proposed by our solution is generic enough to englobe the requirements of different types of departments and occasions present in enterprises. In addition, there is support for enterprises with multiple branches, as user domains can be assumed to be edge routers inside a branch's.

B. Scenario 2 - VNF Service Chain as a Service (VNF-SCaaS)

The price of current Internet middleboxes represents a significant percentage of a company's expenses. With this premise, there is the possibility to monetize service chaining graphs specially designed for exclusive corporations, reducing its CAPEX and OPEX. Our approach facilitates this monetization of service chains by allowing the network operator to set different users domains in the infrastructure with different classes, such as represented in column Scenario 2 in Table III. If a company does not require a high level of inspection or traffic performance it does not need to buy an expensive middlebox just to use some of it features, it pays to have a low degree of inspection in a private NFV environment and redirect the traffic to them. Some examples of SLAs that can be applied to a VNF-SCaaS environment are: *(i) "VoIP traffic from diamond to Internet have high performance"*, *(ii) "FTP traffic from Internet to bronze have none inspection"*; and *(iii) "SMTP traffic from silver to Internet have low inspection"*.

As the steering of traffic is dynamic, business-level operators can define not only classes but generic traffic from different tenants. Let us consider the case where a company provides VNF as a Service, portraited in Figure 4. In this case, there is more than one subscriber to this service, so a business-level operator can define *user domains* for different subscribers (tenants). Once these domains are defined, the operator can write policies such as *"HTTP traffic from* $tenant_X$ *to* $tenant_Y$ *have high inspection"* or *"Video traffic from* $tenant_X$ *to* $tenant_Z$ *have medium performance"*. The first policy is translated, and our solution constructs a service chain for this flow containing an IDS, a Firewall and a DPI (solid red line).

Having the service chain defined, the SCaaS provider can charge the $tenant_X$ accordingly. The second policy specifies that *video service* from $tenant_X$ to $tenant_Z$ must have some level of performance. However, the SCaaS provider only owns a license for a single video caching virtualized function; therefore, the constructed graph will comprise only the video cache function (blue dashed line). This service chain (a video caching function) may improve the overall video quality in streaming, meeting some of $tenant_Z$'s expectation.

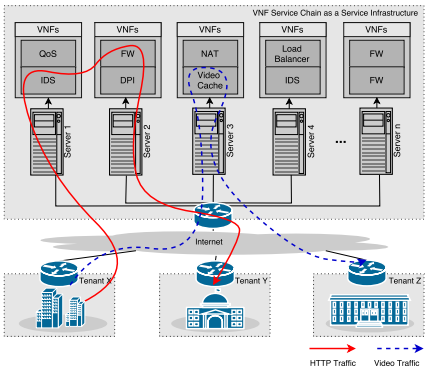


Fig. 4: Traffic Flows in a VNF-SCaaS Infrastructure

This case study is focused on applying NFV as a Service alongside with Service Chaining as a Service. We observe that both service chaining and NFV can be monetized as a business-level operator can add value to different service chains and VNFs, thus charging accordingly.

V. Conclusion and Future Work

In this paper, we presented a policy-based approach to chain multiple VNFs. We also have defined a CNL to allow business-level operators to write SLAs in a GUI. These SLAs guide the construction and the enforcement of service chains in the network. Our approach takes in consideration some elements from the ETSI specification to guide the construction of service chaining graphs. In our approach, PBNM is applied to support the implementation of the components. With the employment of PBNM and a CNL to dynamically compose service chaining graphs, the need for previous knowledge from network operators is lowered, thus reducing OPEX. Additionally, combining NFV and SDN to compose the infrastructure adds elasticity and lowers CAPEX.

We provided two case studies that have validated the feasibility of our solution. The first details a common enterprise network divided by departments and branches. The second presents a scenario where our solution helps operators to monetize their middleboxes. This is achieved by implementing means to provide VNF service chaining as a service, allowing a business-level operator to set classes of priority to different customers, thus charging accordingly. In these two cases, the PBNM approach proved to simplify the management and creation of customized service chains to different flows.

To extend our solution, future work proposals include: (*i*) integration with an implemented NFV framework, such as the ESCAPE framework described in Section II, (*ii*) extension of our solution to address currently not supported requirements, such as QoS and performance; and (*iii*) propose a more sophisticated approach to the suggestion of thresholds and VNF order in the service chaining graph.

References

[1] "Network Functions Virtualisation (NFV)," White Paper, European Telecommunications Standards Institute (ETSI), 2014.

[2] P. Quinn and T. Nadeau, "Problem Statement for Service Function Chaining," (IETF), RFC 7498, 2015.

[3] W. John, K. Pentikousis, G. Agapiou, E. Jacob, M. Kind, A. Manzalini, F. Risso, D. Staessens, R. Steinert, and C. Meirosu, "Research Directions in Network Service Chaining," in *2013 IEEE SDN4FNS*, 2013.

[4] N. Feamster, J. Rexford, and E. Zegura, "The Road to SDN: An Intellectual History of Programmable Networks," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, 2014.

[5] J. Blendin, J. Ruckert, N. Leymann, G. Schyguda, and D. Hausheer, "Position Paper: Software-Defined Network Service Chaining," in *2014 Third European Workshop on Software Defined Networks (EWSDN)*.

[6] J. Strassner, *Policy-Based Network Management: Solutions for the Next Generation (The Morgan Kaufmann Series in Networking)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.

[7] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: A Network Programming Language," in *Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming*, ser. ICFP '11. ACM, 2011.

[8] C. Cleder Machado, J. Araujo Wickboldt, L. Zambenedetti Granville, and A. Schaeffer-Filho, "Policy authoring for software-defined networking management," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, May 2015, pp. 216–224.

[9] J. Halpern and C. Pignataro, "Service Function Chaining (SFC) Architecture," (IETF), RFC 7665, 2015.

[10] P. Quinn and U. Elzur, "Network Service Header," Working Draft, (IETF), Internet-Draft draft-ietf-sfc-nsh-01, 2015.

[11] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "SIMPLE-fying Middlebox Policy Enforcement Using SDN," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13. New York, NY, USA: ACM, 2013, pp. 27–38.

[12] A. Csoma, B. Sonkoly, L. Csikor, F. Németh, A. Gulyas, W. Tavernier, and S. Sahhaf, "ESCAPE: Extensible Service Chain Prototyping Environment Using Mininet, Click, NETCONF and POX," in *Proceedings of the 2014 ACM Conference on SIGCOMM*. ACM, 2014.

[13] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella, "OpenNF: Enabling Innovation in Network Function Control," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM '14. New York, NY, USA: ACM, 2014.

[14] X. Ge, Y. Liu, D. H. Du, L. Zhang, H. Guan, J. Chen, Y. Zhao, and X. Hu, "OpenANFV: Accelerating Network Function Virtualization with a Consolidated Framework in Openstack," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM '14. ACM, 2014.

[15] Y. Zhang, N. Beheshti, L. Beliveau, G. Lefebvre, R. Manghirmalani, R. Mishra, R. Patneyt, M. Shirazipour, R. Subrahmaniam, C. Truchan, and M. Tatipamula, "StEERING: A software-defined networking for inline service chaining," in *Network Protocols (ICNP), 2013 21st IEEE International Conference on*, Oct 2013, pp. 1–10.

[16] W. Ding, W. Qi, J. Wang, and B. Chen, "OpenSCaaS: an open service chain as a service platform toward the integration of SDN and NFV," *Network, IEEE*, vol. 29, no. 3, pp. 30–35, May 2015.

[17] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, "The Ponder Policy Specification Language," in *Proceedings of the International Workshop on Policies for Distributed Systems and Networks*, ser. POLICY, 2001.

[18] T. Kuhn, "A Survey and Classification of Controlled Natural Languages," *Comput. Linguist.*, vol. 40, no. 1, pp. 121–170, Mar. 2014.

[19] European Telecommunications Standards Institute (ETSI), "ETSI Group Specification Network Functions Virtualisation (NFV); Management and Orchestration," 2014.

[20] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, 2008.

AppendixB PUBLISHED PAPER – IM 2017

Eder John Scheid, Cristian Cleder Machado, Muriel Figueredo Franco, Ricaro Luis dos Santos, Ricardo Jose Pfitscher, Alberto Egon Schaeffer-Filho and Lisandro Zambenedetti Granville. **INSpIRE: Integrated Intent-baSed Intent Refinement Environment**. 15th IFIP/IEEE International Symposium on Integrated Network Management (IM 2017), Lisbon, Portugal.

- **Title:** *INSpIRE: Integrated Intent-baSed Intent Refinement Environment*.
- **Contribution:** An *intent* refinement technique based on non-functional requirements.
- **Abstract:** Many aspects of the management of computer networks, such as quality of service and security, must be taken into consideration to ensure that the network meets theusers and clients demands. Fortunately, management solutions were developed to address these aspects, such as Intent-Based Networking (IBN). IBN is a novel networking paradigm that abstracts network configurations by allowing administrators to specify how the network should behave and not what it should do. In this paper, we introduce an IBN solution called INSpIRE (Integrated NFV-based Intent Refinement Environment). INSpIRE implements a refinement technique to translate intents into a set of configurations to perform a desired service chain in both homogeneous environments (VNFs only) and heterogeneous environments (VNFs and physical middleboxes). Our solution is capable of (i) determining the specific VNFs required to fulfill an intent, (ii) chaining these VNFs according to their dependencies, and (iii) presenting enough low-level information to network devices for posterior traffic steering. Finally, to assess the feasibility of our solution we detail a case study that reflects real-world management situations and evaluate the scalability of the refinement process.
- **Status:** Accepted - to appear.
- **Qualis:** B1.
- **Conference:** 15th IFIP/IEEE International Symposium on Integrated Network Management (IM 2017).
- **Date:** May 08 - May 12, 2017.
- **Local:** Lisbon, Portugal.
- **URL:** <http://im2017.ieee-im.org/>.
- **Digital Object Identifier (DOI):** –.

INSpIRE: Integrated NFV-baSed Intent Refinement Environment

Eder J. Scheid, Cristian C. Machado, Muriel F. Franco, Ricardo L. dos Santos, Ricardo P. Pfitscher,
Alberto E. Schaeffer-Filho, Lisandro Z. Granville
Institute of Informatics - Federal University of Rio Grande do Sul
Porto Alegre, RS, Brazil
{ejscheid, ccmachado, mffranco, rlsantos, rjpfitscher, alberto, granville}@inf.ufrgs.br

***Abstract*—Many aspects of the management of computer networks, such as quality of service and security, must be taken into consideration to ensure that the network meets the users and clients demands. Fortunately, management solutions were developed to address these aspects, such as Intent-Based Networking (IBN). IBN is a novel networking paradigm that abstracts network configurations by allowing administrators to specify how the network should behave and not what it should do. In this paper, we introduce an IBN solution called `INSpIRE` (Integrated NFV-based Intent Refinement Environment). `INSpIRE` implements a refinement technique to translate intents into a set of configurations to perform a desired service chain in both homogeneous environments (VNFs only) and heterogeneous environments (VNFs and physical middleboxes). Our solution is capable of (*i*) determining the specific VNFs required to fulfill an intent, (*ii*) chaining these VNFs according to their dependencies, and (*iii*) presenting enough low-level information to network devices for posterior traffic steering. Finally, to assess the feasibility of our solution we detail a case study that reflects real-world management situations and evaluate the scalability of the refinement process.**

I. Introduction

Many aspects of the management of computer networks, such as quality of service and security, must be taken into consideration to ensure that the network meets the users and clients demands. In order to achieve the desired behavior, the devices composing the network must be individually configured, requiring time and effort from network administrators [1]. This activity typically involves the interruption of the network and results on rules that highly depend on the physical network topology. Fortunately, many approaches have been developed to tackle these issues, including well-known and widely employed solutions such as Simple Network Management Protocol (SNMP) [2] and Policy-Based Networking (PBNM) [3] [4], as well as more recent techniques based on Intent-Based Networking (IBN) [5].

IBN is a novel networking paradigm that abstracts network configurations by allowing administrators to specify how the network should behave and not what it should do. For example, in IBN solutions, one must write an *intent* "All outgoing network traffic is encrypted" and not an instruction "If a packet is destined to outside the network then encrypt it using SHA3". *Intent* brings context and is not vendor-specific, which means that the underlying mechanisms must be capable of translating this intent to low-level configurations and maintaining the desired state through the entire network operation. Given the dynamicity involved in IBN, the underlying technologies must be flexible enough to cope with an ever-changing network environment, scaling and moving accordingly. Novel technologies arise as alternatives to provide this flexibility, such as Software-Defined Networking (SDN) [6] and Network Functions Virtualization (NFV) [7].

NFV introduces the possibility to instantiate and terminate network functions dynamically in the infrastructure. Additionally, NFV combined with SDN provides a paradigm shift from earlier technologies, as now different traffic flows can be dynamically steered through any sequence of network functions to provide specialized network services. This act of specifying the sequence of network functions is known as *service function chaining* [8]. However, the management of network functions, service chains, and other network resources becomes challenging as the dynamicity of the network increases. Therefore, the employment of *intents* and IBN in the service chaining context is appropriate. Moreover, *intents* can be used to decouple management strategies from implementation details, reducing the amount of specific knowledge from system-level administrators when configuring low-level settings, *e.g.,* the chaining of Virtual Network Functions (VNFs).

Even though IBN is a novel networking paradigm, *intents* can still be considered high-level abstract policies. In addition, *intents* do not hold specific requirements nor configurations, *e.g.,* OpenFlow rules [9]. Thus, IBN solutions must be able to translate these high-level policies into lower-level specific configurations *e.g.,* IPTables rules or routing tables. This translation process is referred to as *policy refinement* and it has been investigated for several years [10] [11] in the PBNM context. However, to the best of our knowledge, refinement techniques alongside with IBN and NFV were not exploited in any other solution. Therefore, there is an opportunity to investigate refinement techniques in the IBN context.

In this paper, we introduce an IBN solution called `INSpIRE` (Integrated NFV-based Intent Refinement Environment). `INSpIRE` implements a refinement technique to translate *intents* (constrained by a Controlled Natural Language - CNL [12]) into a set of configurations to perform a desired service chain in both homogeneous environments (VNFs only) and heterogeneous environments (VNFs and physical middleboxes). Our solution applies a refinement process, based

on non-functional requirements and *softgoals*, to decompose *intents* and to calculate values that are utilized as selection criteria for the choice of middleboxes that will compose the service chain; ultimately, satisfying the desired *intent*. INSpIRE is capable of *(i)* determining the specific VNFs required to fulfill an *intent*, *(ii)* chaining these VNFs according to their dependencies, and *(iii)* presenting enough low-level information to network devices for posterior traffic steering. Further, we extend the descriptor of VNFs defined in the ETSI Management and Orchestration (MANO) framework [13] to contain meta-data to aid in the ordering of VNFs. Finally, we describe a case study and conduct experiments to demonstrate the scalability of the refinement elements and to evaluate the feasibility of our solution in real-world scenarios.

This paper is structured as follows. In Section II, we review related work. Section III provides a brief description of the main concepts used in our solution, presents INSpIRE, and describes a case study. In Section IV, the experiments, and results are described. Finally, in Section V, we finish this paper with conclusions and future work.

II. Related Work

Early work on policy refinement in the context of PBNM has achieved promising results. Bandara *et al.* [14] proposed to decompose high-level policies into low-level concrete policies based on goal mapping. Given a formal representation of a system, based on Event Calculus (EC), the proposed solution can derive the sequence of operations that will allow a system to achieve the desired goal. The goals can be accomplished by reaching one or more of the underlying goals that were previously derived. Rubio-Loyola *et al.* [15] used linear temporal logic and reactive systems analysis to provide a solution for goal-based policy refinement. Leveraging the KaOS methodology to goal elaboration, the solution can derive goals into low-level policies in the Ponder specification language [16]. Also, the authors present their solution in a DiffServ QoS management scenario. Craven *et al.* [11] described a method for the refinement of authorization and obligation policies. In the work, the domains are represented in UML diagrams, which are used as inputs to the refinement process, alongside with a policy and decomposition rules. After the decomposition, operationalization and re-refinement stages the policy is ready for deployment.

More recently, the use of high-level abstractions for configuring and managing SDN-based infrastructures has been investigated. A novel language for programming OpenFlow networks was proposed by Foster *et al.* called Frenetic [17]. Frenetic provides a high-level language for handling network traffic and a reactive library to compose packet-forwarding policies. This language aims to ease the work of network operators when developing SDN controllers. Machado *et al.* [18] presented a formalism based on EC to represent high-level Service-Level Agreements (SLA) policies and then apply logical reasoning to refine these SLAs into low-level rules to manage an SDN. The authors advocate that some aspects of SDN, such as the ease to gather information about the network (*e.g.,* jitter and delay) by OpenFlow controllers, enhances the policy refinement process in such environments.

All these works address the refinement of high-level into low-level policies. However, *intents* are abstract and subjective, changing from domain to domain. Therefore, we need to consider this subjectiveness when refining *intents*. Furthermore, the scope of most refinement techniques is restricted to specific domains, such as QoS management in conventional IP networks or access control systems, which limits their employment in the refinement of *intents*. Moreover, a single *intent* can alter the configuration of many elements in the network. Thus, the modeled domains used for the refinement process must accurately reflect the elements and configurations of the whole network and not just a single scope.

III. Integrated NFV-baSed Intent Refinement Environment

In this section, we present INSpIRE to address the refinement of *intents*, which are defined as high-level abstract policies, into a set of network functions and information that satisfy a specific *intent*. Our refinement technique is composed of three steps. The first step relates to the modeling of the domain in which the *intent* is being applied, including operations performed by network functions (*e.g.,* L2 inspection and packet filtering) and non-functional requirements (*e.g.,* security and performance). We detail this model and requirements in Section III-A and Section III-B. The second step includes the quantitative calculation of the non-functional requirements of a VNF based on the modeled domain, resulting in numerical values for those requirements. Finally, the third step includes the parsing of the *intent* and the clustering of VNFs based on the resultant values from the quantitative calculation.

A. Non-Functional Requirements (NFR) Framework

When designing a software, it is crucial that software engineers consider both functional requirements and non-functional requirements. The former dictates what the software is expected to do (*e.g.,* store employees data and exchange email), the latter defines the qualities of the software (*e.g.,* store data securely and exchange email quickly). Non-Functional Requirements (NFRs) are, usually, informally specified during the software development process, being based on empirical observation from stakeholders, and thus are hard to model. Therefore, one of the main challenges is to define how one can model the qualities of a system [19].

To model NFRs, we rely on the NFR Framework [20]. In which *softgoals* and *operationalizations* represent the requirements in a *Softgoal* Interdependency Graph (SIG). An example of a SIG is depicted in Figure 1. One *softgoal* (cloud shape) can have different types of contributions and relationships towards other *softgoals*, such as BREAK (−−), HELP (+), HURT (−), and MAKE (++). While MAKE and HELP contribute positively to *satisfice*[1] an upper *softgoal*,

[1]According to the Oxford Dictionary *satisfice* is defined as "Accept an available option as satisfactory". Therefore we utilize this word instead of *satisfy* in this context.

BREAK and HURT contribute negatively. To satisfice these *softgoals*, one must first identify possible techniques that must be implemented in the system, named *operationalizations* (bold cloud shape). These *operationalizations* are the external nodes of the SIG.

We formally define a SIG with the set below:

$$\mathcal{SIG} = (\mathcal{V}, \mathcal{E})$$
$$\mathcal{V} \in \{SG, LSG, OP\}$$

where

- SG: represents the primary set of *softgoals*, which are the root-node of the graph.
- LSG: is the set of refined leaf-*softgoals* from the primary *softgoal*.
- OP: contains the set of *operationalizations* that contributes to satisfice the LSG or the SG.

and

$$\mathcal{E} \in \{\uparrow^{++}, \uparrow^{+}, \uparrow^{--}, \uparrow^{-}, \wedge\}$$

where

- $\uparrow^{++}$ (MAKE): Denotes a strong positive contribution towards a *softgoal*. One single MAKE contribution fully satisfices a parent *softgoal* if the offspring is satisficed.
- $\uparrow^{+}$ (HELP): Denotes a positive contribution. Which means that a child *softgoal* partially contributes to satisfice a parent *softgoal*.
- $\uparrow^{--}$ (BREAK): Denotes a strong negative contribution. If a *softgoal* is satisficed then the parent *softgoal* is automatically denied.
- $\uparrow^{-}$ (HURT): Denotes a negative contribution. Which means that a child *softgoal* partially contributes negatively to satisfice a parent *softgoal*.
- $\wedge$ (AND): This contribution relates to a group of *softgoals* to their parent. If all child *softgoals* are satisficed then the parent is also satisficed.

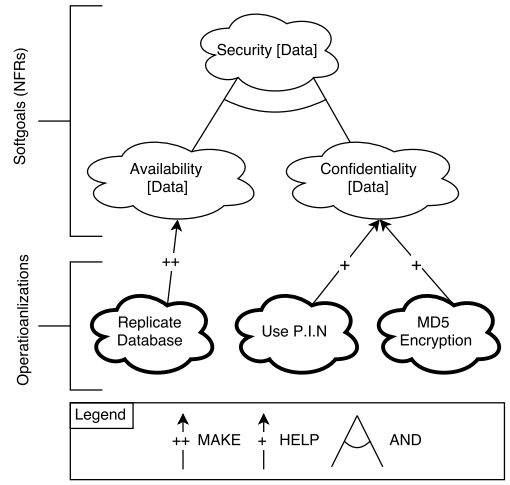


Fig. 1: SIG Example

B. SIG Modeling

In software engineering, the modeling of the SIG follows a top-down approach, starting with a high-level *softgoal* being refined into other *softgoals* until the *operationalizations* are defined and selected. However, in our solution, we assume the SIG is already pre-defined, and each VNF is submitted through a bottom-up process in the SIG to quantify its initial *softgoal* score, *i.e.,* attributing a numerical value for the primary non-functional requirement. In order to model this pre-defined SIG, we first identify the domain in which the SIG is going to be applied, in our case, middleboxes. Then, we select the non-functional requirements that we want to measure, such as security or performance.

Let us consider the SIG depicted in Figure 2. We have extracted the NFRs to compose this SIG from the work Guide to Intrusion Detection and Prevention Systems (IDPS) by Scarfone and Mell [21]. In the work, the authors provide an overview of Intrusion Detection Systems (IDSes) and Intrusion Prevention Systems (IPSes) to help organizations understand such systems. We use this work as a guideline to model a pre-defined SIG, which is then used for evaluating VNFs. To cope with the subjectiveness of the *intents* and requirements, one can alter the SIG at any time to reflect its domain, middleboxes, and network.

To simplify and provide a more straightforward example, we only address the refinement and modeling of one non-functional requirement, which is *Security*. Therefore, following the traditional SIG modeling approach, we start with an initial *softgoal* (*Security*). This *softgoal* is then refined into four leaf-*softgoals*: *Information Gathering*, *Logging*, *Detection*, and *Prevention*. These refined *softgoals* are common security capabilities that, accordingly to Scarfone and Mell, most IDPS technologies provide. For each refined *softgoal*, we attribute a weight corresponding to the importance of this *softgoal* in satisficing the initial *softgoal*. These weights are arbitrary and can be altered by the network operator according to his needs. As operationalizations are techniques that contribute to satisfice *softgoals*, a single *operationalization* can have an impact on one or more *softgoals*. For example, the *operationalization Blacklist and Whitelist Support* contributes to both *Detection* and *Prevention softgoals*, while the *operationalization Identify Applications* contributes to only one *softgoal* (*Information Gathering*). These contributions have numerical values attributed to them (similar to the *softgoal* weight) which reflects the impact to satisfice *softgoals*. The bold red values inside the clouds are calculated by `INSpIRE` following the steps presented in the next section.

C. Quantitative Calculation of NFRs

To accurately quantify the non-functional requirements of a VNF, we leverage the extension of the NFR Framework proposed by Affleck and Krishna [22]. This extension provides a lightweight quantitative support for the NFR Framework, defining a mathematical base for the calculation of scores and weights for *softgoals* and *operationalizations*. Given the formalization of the SIG presented in Section III-A and the

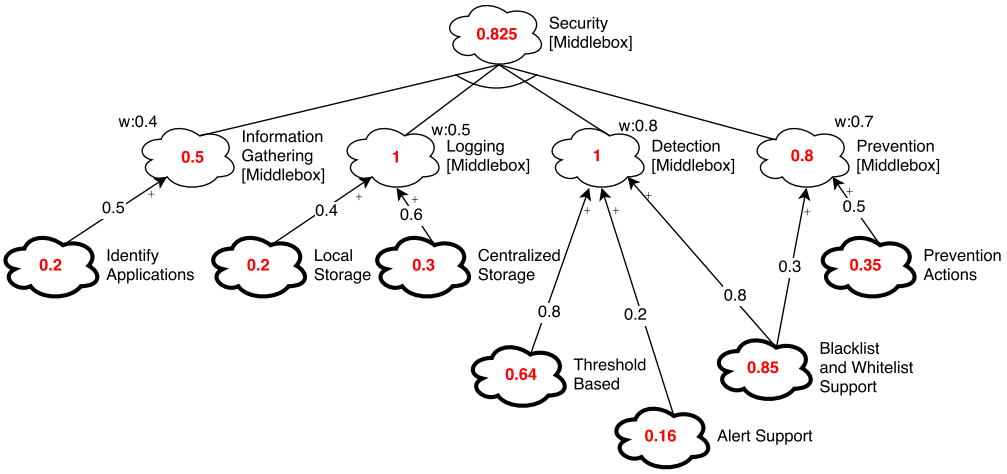


Fig. 2: Pre-defined SIG for Middlebox Security

TABLE I: Quantitative Contribuitions from Affleck and Krishna [22]

Symbol	Name	Contribution
$\uparrow^{++}$	MAKE	1
$\uparrow^{+}$	HELP	[0,1]
$\uparrow^{--}$	BREAK	-1
$\uparrow^{-}$	HURT	[-1,0]
$\wedge$	AND	$\frac{1}{numChilds}$

SIG modeled in Section III-B, we adapt this extension to our objectives.

Leaf-*Softgoal* weights are defined as

$$\forall LSG \in \mathcal{V}, (0.0 \leq LSG_{weight} \leq 1.0)$$

where lower values (closer to 0.0) denote a non-critical *softgoal*, while higher values (closer to 1.0) represent critical *softgoals*. The relationships between *softgoals* and *operationalizations* are defined following the contributions depicted in Table I and are referred to as $impact_{LSGXOP}$.

Operationalization scores are calculated from top to bottom following Equation 1. Therefore, if the network operator decides to add *operationalizations* and *softgoals* to the graph, he only includes in the SIG the values of LSG_{weight} and $impact_{LSG\times OP}$.

$$OP_{score} = \sum_{LSG} LSG_{weight} \times impact_{LSG\times OP} \quad (1)$$

Given the SIG depicted in Figure 2, let us calculate the "*Blacklist and Whitelist Support*" *operationalization*'s score. This *operationalization* contributes positively to two leaf-*softgoals* (*Detection* and *Prevention*). Therefore, we have as the result from Equation 1, the score of 0.85, which means a positive contribution to the system. The steps are shown below.

$$\begin{aligned} OP_{score} &= (0.8 \times 0.8) + (0.3 \times 0.7) \\ &= 0.64 + 0.21 \\ &= 0.85 \end{aligned}$$

The next step proposed by Affleck and Krishna [22] is the selection of *operationalizations* based on the scores previously calculated. However, in our approach, this step occurs when a network operator inserts a VNF or middlebox in the system. The operator must select which operations the VNF is capable of performing (*e.g.,* identify which flows are harmful or detect attacks). Considering the SIG in Figure 2, if a network operator specifies that a VNF does not store logs on a centralized server (*Logging [Middlebox]*), the $impact_{LSG\times OP}$ of that *operationalization* is going to be zero. Consequently, the OP_{score} is going to be also zero ($OP_{score} = 0.5 \times 0$) and score of the leaf-*softgoal* will decrease.

To calculate leaf-*softgoal* scores, we employ the same equation as Affleck and Krishna [22]. The only difference is that we consider all *operationalizations* and not only the accepted ones. Equation 2 shows that the LSG_{score} is the sum of the impact (even zero impact) of every *operationalization* that contributes to the leaf-*softgoal*. This score is limited to $[-1.0, 1.0]$ by max and min functions, where -1.0 means that the *softgoal* was not satisficed and 1.0 means that the *softgoal* was 100% satisficed. Below Equation 2 is depicted the steps for the calculation of the *Detection[Middlebox]* score.

$$LSG_{score} = max(min(\sum_{OP} impact_{LSG\times OP}, 1), -1) \quad (2)$$

$$\begin{aligned} LSG_{score} &= max(min(0.8 + 0.2 + 0.8, 1), -1) \\ &= max(min(1.8, 1), -1) \\ &= 1 \end{aligned}$$

Once the *operationalizations* and leaf-*softgoals* scores are computed, the initial *softgoal* (*Security[Middlebox]*) score can

be calculated. This score ultimately represents how much (percentage) the *softgoal* has been satisficed. To simplify our system, we only address AND ($\wedge$) contributions from the initial *softgoal* towards leaf-*softgoals*. Thus, Equation 3 considers the sum of leaf-*softgoal* scores divided by the number of children of that *softgoal*, so every leaf-*softgoal* contributes with a percentage of its score to satisfice the initial *softgoal*.

$$SG_{score} = max(min(\frac{\sum_{LSG} LSG_{score}}{SG_{numChilds}}, 1), -1) \quad (3)$$

Finally, as our intention is not to calculate how secure a middlebox is, but rather how much security a middlebox can provide to a specific flow passing through it, this score of 0.825 (calculated below) means that the middlebox or VNF can provide 82.5% of security. This value is relative to the SIG that was modeled by network operators and may vary from organization to organization. Therefore, as now we have a numerical value for the security *softgoal* of the VNF, we can use this value to cluster the available VNFs into groups with different levels of security. It is important that network operators, administrators and, business partners discuss and model this SIG exhaustively so that the defined weights can faithfully reflect the domain. This is due to the weights influence in future service chaining decisions.

$$SG_{score} = \frac{0.5 + 1 + 1 + 0.8}{4}$$
$$= 0.825$$

D. Refinement of Intents and Clustering of VNFs

`INSpIRE` needs to posses knowledge about the environment to properly refine *intents*. Therefore, a network operator has to insert the middleboxes or VNFs present in the infrastructure into a database for later selection. This insertion process consists of uploading in the system a descriptor (*vnfd* in the case of a VNF), filling in information about the middlebox (*e.g.,* IP address, switch port, and type of network function), and selecting the operations performed by this network function. These operations are the *operationalizations* defined in the SIG and are necessary for the calculation of *softgoals*. Once this data is informed, `INSpIRE` computes the *softgoal* score of the target VNF using the Equations in Section III-C and stores it.

The refinement process is depicted in Figure 3 and includes elements such as a traffic classifier and a service chaining identifier. The former translates part of the *intent* into traffic objects (*e.g.,* traffic type, source, and destination) that will be used for posterior traffic steering and matching by an external component (illustrated by a dashed line box). The latter retrieves the context (security) and level (high) of the written *intent* and forwards this context to the component responsible for constructing the related chaining. The Service Chain Graph Builder component retrieves the VNFs that were inserted by the network operator and cluster the VNFs in different sets of VNFs with similar *softgoals* scores.

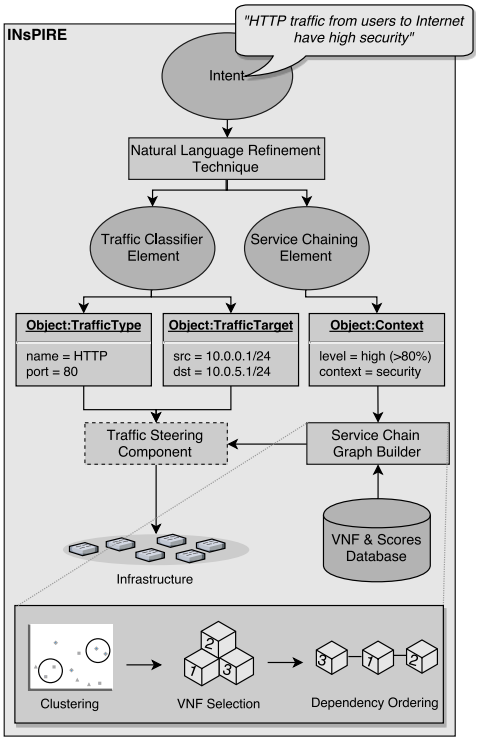


Fig. 3: INSpIRE Intent Refinement Flow

1) Clustering: To cluster the VNFs in sets with similar scores, we employ the *k*-means clustering algorithm [23]. This algorithm was proposed to classify n values into a defined k number of clusters sharing similar scores. In our case, we set $k = 3$, representing the levels of the contexts supported by the CNL (low, medium, and high), and n is the number of available VNFs. The dimensionality (*i.e.,* number of features) of the plot depends on the number of *softgoals* specified in the *intent*. For example, one can write an *intent* addressing more than one *softgoal*, *e.g.,* "*FTP traffic from teachers to teachers have medium* ***security***, ***detection***, *and* ***log support***". Thus, the resulting graph (Figure 4) will have three axes (security, detection, and log support) and so forth for more *softgoals*.

2) VNF Selection: After the *k*-means algorithm is executed, and the estimation of the clusters is completed, we must select the VNFs that will compose the service chain. To select from which cluster we will retrieve the VNFs, we leverage the level of the context that was defined by the network operator in the *intent*. In the past *intent* example, one defined as the level being "medium", therefore, we only select the VNFs that are inside the middle cluster, represented as squares in Figure 4. We select x random VNFs from the cluster, where $x = 3$ in a first moment. However, this number can be adjusted if the network operator desires. To attempt to utilize the full capacity of the VNFs in the infrastructure and not to impact on the overall chaining performance, we employed a selection algorithm. This algorithm prioritizes the selection of VNFs that are already deployed and are not under CPU stress ($VNF_{CPUload} \leq \alpha$, where $\alpha = 0.8$, defined by empirical

observation but customizable). If all the deployed VNFs are under CPU stress, then the algorithm selects the undeployed VNFs and the NFV orchestrator takes care of the placement of those VNFs. Bear in mind that is not the scope of this work the placement of VNFs.

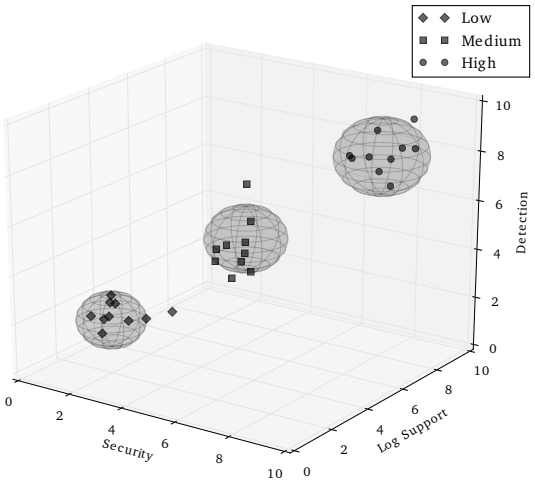


Fig. 4: VNF Clustering Example

3) VNF Dependency Ordering: Many VNFs often depend on other deployment units (*i.e.,* virtual machines) to be deployed/initiated before an orchestrator can deploy them. The information about this dependency is stored, accordingly to the ETSI NFV-MANO [13], in the element `dependency` inside the *vnfd* of the VNF. Therefore, if a VNF is selected and is not deployed, `INSpIRE` attempts to resolve this dependency by searching for a VNFD Virtual Deployment Unit *vnfd:vdu* that fulfills this dependency. First, it seeks if one of the selected VNFs produced by the selection procedure contains the *vnfd:vdu* specified in the `dependency` element. If there is no match, then it searches all the deployed VNFs for a match and then informs the orchestrator to address this issue. Note that this dependency process relates to the deployment and placement of VNFs, and thus we do not delve into details.

Moreover, when designing service chains, it is important to consider the logical order of the network functions. This ordering process is not a trivial process, mainly because of the hidden and subjective dependencies across the many network functions. For example, there is no explicit definition that a firewall must come first in the chaining order and then be followed by a DPI. This ordering is based on empirical observation and the logic behind the operations performed by the functions. It is illogical to put a DPI in front of a firewall, because the DPI will have to inspect every single incoming packet, causing performance degradation. Whereas if the firewall is set in front of the DPI, then the firewall will only forward filtered packets, reducing the number of packets to be processed by the DPI, and thus, not impacting on the overall service chain performance.

To address this ordering issue, we propose to include in the VNF Descriptor two new elements: `vnf_type` and `preference_list`. The former describes what is the type of the VNF, *e.g.,* DPI, IPS, Firewall, Video-cache, NAT, and so on. The latter describes the list of VNF ordering preferences. For example, a DPI, in theory, is more complex than a firewall. Therefore, the DPI has a preference to be included after the firewall in the chain, *i.e.,* the flow will pass through the firewall first, then go through the DPI. If there is no preference, the order is not important for the VNF. This preference will be specified in the element `preference_list` with a list of `types`. Thus, we utilize this preference list to order the chain. The network operator can re-order the service chain after the chain is saved into the database. If a loop is encountered in the chain, the network operator is notified to address this issue.

E. Case Study

Let us describe the interaction of users with `INSpIRE`. This interaction is composed of three distinct stages: *(i)* SIG modeling, *(ii)* VNF insertion, and *(iii) intent* creation.

In a first moment, without the intervention of `INSpIRE`, network operators along with business partners assemble a team to identify the non-functional requirements (initial *softgoal* and leaf-*softgoals*) that are important for the proper operation of the enterprise. If the enterprise is concerned about the privacy of their data, the team identifies the initial *softgoal* as *Privacy* and two requirements (leaf-*softgoals*) that must contribute to it, such as *Encryption*, and *Traffic Anonymization*. These requirements are satisficed by the *operationalizations*, which must also be identified and outlined by the team, either by reviewing the common functions realized by middleboxes or by empirical knowledge. In the privacy context, the team outlines functions such as *Tor-based anonymizer*, *SSL traffic encryptor*, and *SHA-1 encryptor*. Next, the team attributes the weights of all leaf-*softgoals* and the impact of each *operationalizations* in these leaf-*softgoals*. The attributed impacts of the mentioned *operationalizations* are described in Table II. As the team identified two leaf-*softgoals* they attributed equal weights to both of them, *e.g.,* 0.5.

TABLE II: Impact of *Operationalizations* towards Leaf-*Softgoals*

Operationalization	Contribution	Leaf-*softgoal*	Impact
Tor-based anonymizer	MAKE	*Traffic Anonymization*	1
SSL traffic encryptor	HELP	*Encryption*	0.7
SHA-1 encryptor	HELP	*Encryption*	0.5

The modeled SIG can be represented in YAML (YAML Ain't Markup Language), XML (eXtensible Markup Language), or JSON format and then imported to `INSpIRE` as well as exported from it for posterior edition. An example of exported SIG in YAML format is depicted in Figure 5. We adopted the YAML format due to its readability. In the YAML file, nodes of the SIG are described after the `node:` tag (line 27) in the `node:{atributes:value}` format. For example, the node *Traffic Anonymization* (line 36) is represented as `Traffic Anonymization:{lsg: true, w: 0.5}`, which means it is a leaf-*softgoal* (`lsg: true`)

with a weight of 0.5 (`w: 0.5`). Edges are described after the `adj:` tag (line 2) in the `sourceNode:` format. New lines with indentation describe the destination nodes and attributes in the `destinationNode:{attribute:value}` format. For example, there is an edge from node `SSL Traffic Encryptor:` (line 15) to node `Encryption: {impact: 0.7}` (line 16), meaning that the node *SSL Traffic Encryptor* impacts on 70% to satisfice the node *Encryption*.

```
1 !!python/object:networkx.classes.graph.Graph
2 adj: &id013
3    Alert Support:
4      Detection: &id001 {impact: 0.2}
5    Blacklist and Whitelist Support:
6      Detection: &id002 {impact: 0.8}
7    Centralized Storage:
8      Logging: &id020 {impact: 0.6}
9    ...
10   Privacy:
11     Traffic Anonymization: *id006
12     Encryption: *id007
13   Tor-based Anonymizer:
14     Traffic Anonymization: &id015 {impact: 1.0}
15   SSL Traffic Encryptor:
16     Encryption: &id011 {impact: 0.7}
17   SHA-1 Encryptor:
18     Encryption: &id017 {impact: 0.5}
19   Security:
20     Detection: *id008
21     Logging: *id010
22   ...
23 adjlist_dict_factory: &id014 !!python/name:__builtin__.dict ''
24 edge: *id013
25 edge_attr_dict_factory: *id014
26 graph: {}
27 node:
28   Alert Support: {op: true}
29   Blacklist and Whitelist Support: {op: true}
30   Centralized Storage: {op: true}
31   Tor-based Anonymizer: {op: true}
32   SSL Traffic Encryptor: {op: true}
33   SHA-1 Encryptor: {op: true}
34   Identify Applications: {op: true}
35   ...
36   Traffic Anonymization: {lsg: true, w:0.5}
37   Encryption: {lsg: true, w:0.5}
38   Detection: {lsg: true, w: 0.8}
39   Logging: {lsg: true, w: 0.5}
40   ...
41   Privacy: {sg: true}
42   Security: {sg: true}
43 node_dict_factory: *id014
```

Fig. 5: Example of a SIG in YAML Format

After the SIG is modeled and imported to INSpIRE, the network operator can start to add the VNFs and middleboxes that are available in the infrastructure. To perform this addition, he/she utilizes a Graphical User Interface (GUI) provided by INSpIRE, which contains a form with the necessary fields to be filled by the network operator with information about the middlebox. Such fields include IP address, switch port, type of network function, VNFD file (VNF only), description and name. Within this GUI, a list of *operationalizations* in a tree view format, separated by leaf-*softgoal* and *softgoal*, is presented for the network operator to select the functions that the VNF support. For example, if the VNF is a firewall, the network operator will select operations such as *Blacklist and Whitelist Support* and *Alert Support* and submit the form. The *operationalizations* list is composed of nodes of the SIG and may change if the SIG is altered. Once the form is submitted, INSpIRE automatically calculates the scores following the equations described in Section III-C and stores the scores in a database along with the information about the middlebox previously informed. One example of entry in the scores database is the tuple `<vnfId,[Security:0.825,...,Detection:1]>`.

Next step is to write *intents*. This process in based on previously work [12], where we have defined a controlled language for the writing of high-level policies. We utilize this language for the composition of *intents* in INSpIRE. The *intents* that are going be written in INSpIRE are in the format "`trafficType` *from* `source` *to* `destination` *have* `contextLevel contextsList`". For example, let us assume that a board of directors described an SLA specifying that all email traffic from the Finance Department must have high security and privacy. Therefore, a network operator would translate this SLA into two *intents*: *(i)* "*SMTP traffic from finance_department to * (any) have high security and privacy*", and *(ii)* "*IMAP traffic from finance_department to * have high security and privacy*". Then, INSpIRE refines these two *intents* (one at a time) into objects to be used for traffic classification and service chain construction purposes. As INSpIRE only address service chain construction, the refined elements utilized are: `contextLevel: high` and `contextList:[security,privacy]`. Therefore, INSpIRE retrieves all the entries of the score database in order to utilize these scores to cluster the VNFs. In the example above, the cluster will have two dimensions (security and privacy) and INSpIRE will plot every VNF based on its score of these two *softgoals*.

Once the entries are plotted, INSpIRE discover the three clusters and retrieves only the entries that belong to the context level defined in the *intent*, *e.g.,* (*high*). Next, INSpIRE orders the retrieved VNFs following the process described in Section III-D3 and informs the traffic steering element of the service chain related to the traffic classification objects and the *intent*. Finally, packets originated from IPs in the 192.168.1.5\24 range (*i.e.,* finance_department) and classified as STMP (port 25) are steered through the chain related to these objects.

IV. Evaluation

As INSpIRE is composed of different stages, such as clustering and NFR scores calculation, we performed a series of scalability simulations for these stages. The simulations were designed to stress the components (isolated) in order to discover the behavior of INSpIRE in different types of scenarios, varying from small (a couple of elements) to huge scenarios (thousand of elements). The tests were performed in a Dell XPS 8900 with an Intel Core i7-6700 CPU at 4GHz processor and 16GiB of RAM. The algorithms, equations, and simulations were implemented in Python utilizing well-know graph (NetworkX [24]) and clustering (SciPy [25]) libraries.

A. Score Calculation Evaluation

To evaluate this component, we simulated different SIGs with the number of leaf-*softgoals* and *operationalizations* varying from 2 to 64 in a logarithmic scale. For example, we

created a SIG with 2 leaf-*softgoals* with 2 *operationalizations* each, then a SIG with the same number of leaf-*softgoals* (2) but with 4 *operationalizations* each and so son until we reached a SIG with 64 leaf-*softgoals* with 64 *operationalizations* each. Due to the simplicity of the equation to calculate the score of the initial-*softgoals*, which is only a division, the number of initial-*softgoals* in the experiments was fixed to 1.

For every SIG configuration, we ran the calculations of the scores 30 times. The results of this simulation are depicted in Figure 6. The error bars expose the standard deviation. The x-axis characterizes the number of leaf-*softgoals* and the different lines characterize the number of *operationalizations* of each leaf-*softgoal*. The time to calculate all the equations (in seconds) related to *softgoals* scores, leaf-*softgoals* scores and *operationalizations* scores are depicted in the y-axis in a logarithmic scale. As we can notice, the execution time increases as we increase the number of *operationalizations* for each leaf-*softgoal*. This execution time stays below 1 second until the number of leaf-*softgoals* reaches 64 and the number of *operationalizations* attached to them reaches 64 as well. With this SIG configuration, the time to calculate the scores of the total amount of nodes ($64 \times 64 = 4096$) reaches approximately 3 seconds. In INSpIRE, we consider an acceptable execution time of less than 1 second. Therefore, the number of nodes in a SIG scale up to 2048 nodes without affecting the overall INSpIRE performance.

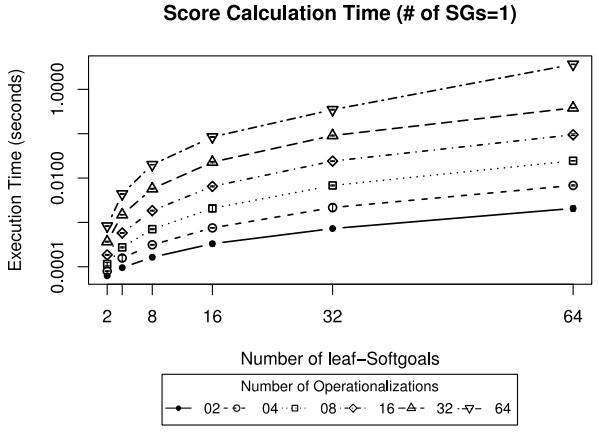


Fig. 6: Time to calculate the scores in different SIGs

B. Clustering Evaluation

This component of INSpIRE is implemented utilizing the k-means algorithm. The simulations were ran 30 times to provide enough significance level. We simulated different types of environments varying the number of elements (n) to be clustered and the number of dimensions of the elements. The number of dimensions represented the number of *requirements* specified in the *intent* and the number of elements represented the number of VNFs and middleboxes in the database. The score of each requirement was randomly attributed varying from 0 to 1. The number of clusters (k) for the algorithm to estimate was set to 3. Figure 7 exposes the Execution Time (in seconds) for each set of VNFs varying the number of Dimensions. The number of VNFs varied from 10 VNFs up to 1 million VNFs, and the number of dimensions ranged from 1 to 16. We notice that the k-means execution time is relatively insignificant up to 10.000 VNFs with 16 dimensions. Even for a million of VNFs with four (4) dimensions (square dashed line) the algorithm took 5 seconds to execute. Therefore, INSpIRE can cluster up to 10000 (ten thousand) VNFs and middleboxes with 16 dimensions in less than 1 second.

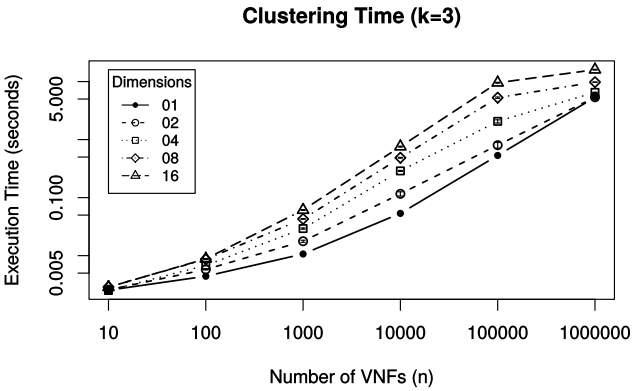


Fig. 7: Time to discover 3 clusters in different scenarios

V. Conclusion and Future Work

In this paper, we presented INSpIRE. INSpIRE is an IBN solution that refines *intents* into service chains of VNFs by employing a technique based on *Softgoal* Interdependency Graphs (SIGs) and clustering. INSpIRE is able to automatically calculate and attribute scores for the non-functional requirements of a VNF. Also, it utilizes these scores to cluster and select the appropriate VNFs to fulfill a defined *intent*. In addition, we proposed the insertion of two elements in the descriptor of VNFs specified in the ETSI NFV MANO. These elements aid INSpIRE to order the VNFs in the service chain based on the VNF's ordering preferences. Thus, INSpIRE provides an approach to solve the refinement of *intents* into service chains issue.

We provided a case study and simulations of the components of INSpIRE to validate its feasibility. The case study details the interaction of users (stakeholders and network operators) with the different stages of INSpIRE (SIG modeling, insertion of VNFs, *intent* writing, and refinement). The implemented simulations of the components showed that INSpIRE is able to work in small scenarios (hundred of VNFs and SIGs with 128 nodes) and large scenarios (ten thousands of VNFs and SIGs with 2048 nodes) as well.

To extend INSpIRE, future work proposals include: (*i*) integrate INSpIRE with a consolidated NFV framework, (*ii*) model a complete SIG to represent other non-functional requirements such as Integrity and Availability, and (*iii*) conduct a qualitative evaluation of INSpIRE.

REFERENCES

[1] D. C. Verma, *Principles of Computer Systems and Network Management*, 1st ed. Springer Publishing Company, Incorporated, 2009.

[2] J. D. Case, M. Fedor, M. L. Schoffstall, and J. R. Davin, "Simple Network Management Protocol (SNMP)," Internet Requests for Comments, RFC Editor, STD 15, May 1990.

[3] J. Strassner, *Policy-Based Network Management: Solutions for the Next Generation (The Morgan Kaufmann Series in Networking)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.

[4] D. C. Verma, "Simplifying Network Administration Using Policy-based Management," *Netwrk. Mag. of Global Internetwkg.*, vol. 16, no. 2, pp. 20–26, Mar. 2002.

[5] M. Behringer, M. Pritikin, S. Bjarnason, A. Clemm, B. Carpenter, S. Jiang, and L. Ciavaglia, "Autonomic Networking: Definitions and Design Goals," RFC 7575 (Informational), 2015.

[6] N. Feamster, J. Rexford, and E. Zegura, "The Road to SDN: An Intellectual History of Programmable Networks," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 87–98, Apr. 2014.

[7] "Network Functions Virtualisation (NFV)," White Paper, European Telecommunications Standards Institute (ETSI), october 2014.

[8] P. Quinn and T. Nadeau, "Problem Statement for Service Function Chaining," RFC 7498 (Informational), Internet Engineering Task Force, 2015.

[9] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[10] J. D. Moffett and M. S. Sloman, "Policy Hierarchies for Distributed Systems Management," *IEEE Journal on Selected Areas in Communications*, vol. 11, no. 9, pp. 1404–1414, Dec 1993.

[11] R. Craven, J. Lobo, E. Lupu, A. Russo, and M. Sloman, "Policy Refinement: Decomposition and Operationalization for Dynamic Domains," in *Network and Service Management (CNSM), 2011 7th International Conference on*, Oct 2011, pp. 1–9.

[12] E. J. Scheid, C. C. Machado, R. L. dos Santos, A. E. Schaeffer-Filho, and L. Z. Granville, "Policy-based dynamic service chaining in Network Functions Virtualization," in *2016 IEEE Symposium on Computers and Communication (ISCC)*, June 2016, pp. 340–345.

[13] European Telecommunications Standards Institute (ETSI), "ETSI Group Specification Network Functions Virtualisation (NFV); Management and Orchestration," 2014, deliverable 1.1.1. Available from http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_NFV-MAN001v010101p.pdf [accessed 17 Dez 2015].

[14] A. K. Bandara, E. C. Lupu, J. Moffett, and A. Russo, "A Goal-based Approach to Policy Refinement," in *Proceedings of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks*, ser. POLICY '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 229–.

[15] J. Rubio-Loyola, J. Serrat, M. Charalambides, P. Flegkas, and G. Pavlou, "A Functional Solution for Goal-Oriented Policy Refinement," in *Policies for Distributed Systems and Networks, 2006. Policy 2006. Seventh IEEE International Workshop on*. IEEE, 2006, pp. 133–144.

[16] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, "The Ponder Policy Specification Language," in *Proceedings of the International Workshop on Policies for Distributed Systems and Networks*, ser. POLICY '01. London, UK, UK: Springer-Verlag, 2001, pp. 18–38.

[17] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: A Network Programming Language," in *Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming*, ser. ICFP '11. New York, NY, USA: ACM, 2011, pp. 279–291.

[18] C. C. Machado, J. A. Wickboldt, L. Z. Granville, and A. Schaeffer-Filho, "An EC-based Formalism for Policy Refinement in Software-Defined Networking," in *2015 IEEE Symposium on Computers and Communication (ISCC)*, July 2015, pp. 496–501.

[19] L. Chung and J. C. Prado Leite, "Conceptual Modeling: Foundations and Applications," A. T. Borgida, V. K. Chaudhri, P. Giorgini, and E. S. Yu, Eds. Springer-Verlag, 2009, ch. On Non-Functional Requirements in Software Engineering, pp. 363–379.

[20] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*, 1st ed. Springer Publishing Company, Incorporated, 2000.

[21] K. A. Scarfone and P. M. Mell, "SP 800-94. Guide to Intrusion Detection and Prevention Systems (IDPS)," Gaithersburg, MD, United States, Tech. Rep., 2007.

[22] A. Affleck and A. Krishna, "Supporting Quantitative Reasoning of Non-functional Requirements: A Process-oriented Approach," in *Proceedings of the International Conference on Software and System Process*, ser. ICSSP '12, 2012, pp. 88–92.

[23] J. Macqueen, "Some Methods for Classification and Analysis of Multivariate Observations," in *In 5-th Berkeley Symposium on Mathematical Statistics and Probability*, 1967, pp. 281–297.

[24] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring Network Structure, Dynamics, and Function using NetworkX," in *Proceedings of the 7th Python in Science Conference*, G. Varoquaux, T. Vaught, and J. Millman, Eds., Pasadena, CA USA, 2008, pp. 11 – 15.

[25] E. Jones, T. Oliphant, P. Peterson *et al.*, "SciPy: Open source scientific tools for Python," 2001–. [Online]. Available: http://www.scipy.org/