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.

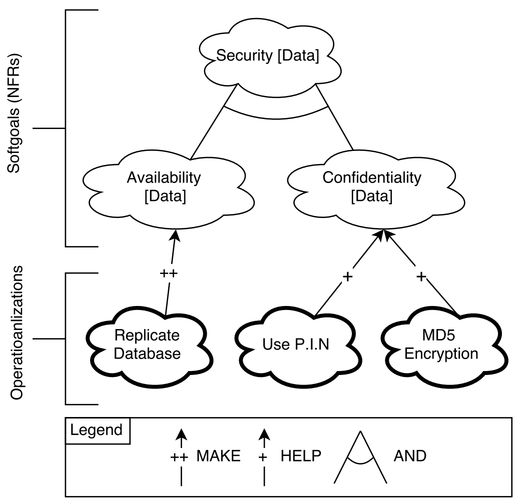

Fig. 1: SIG Example

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

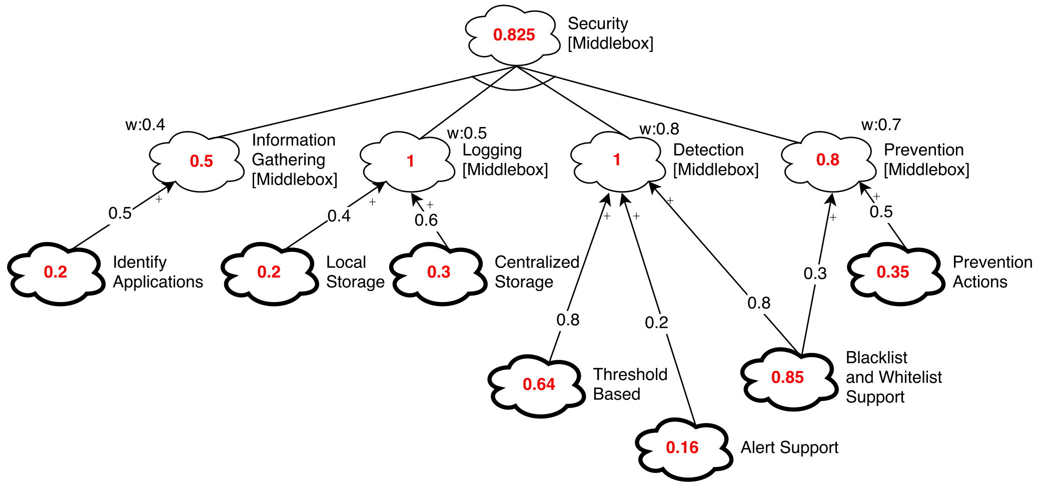

Fig. 2: Pre-defined SIG for Middlebox Security

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.

$$\begin{aligned} SG_{score} &= \frac{0.5 + 1 + 1 + 0.8}{4} \\ &= 0.825 \end{aligned}$$

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.

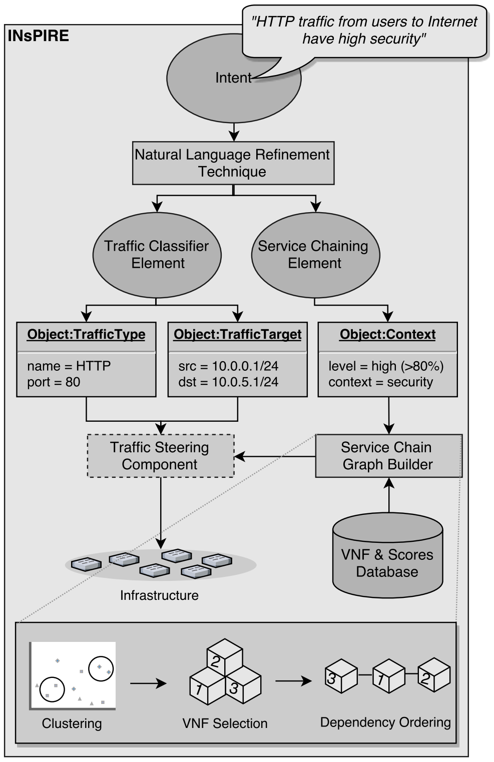

Fig. 3: INSpIRE Intent Refinement Flow

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.

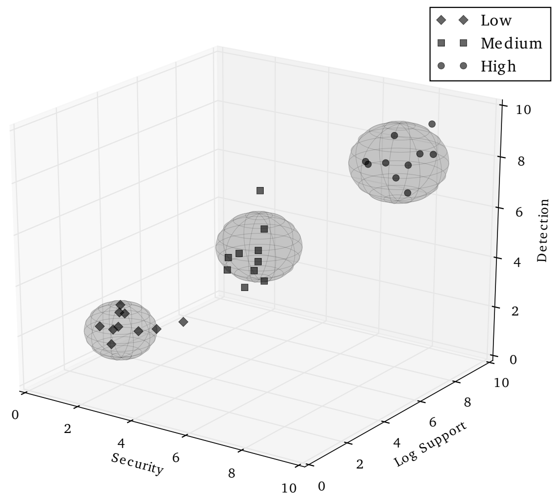

Fig. 4: VNF Clustering Example

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.

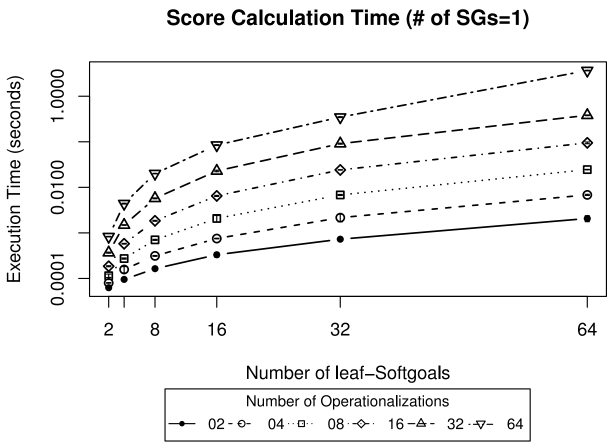

Fig. 6: Time to calculate the scores in different SIGs

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.

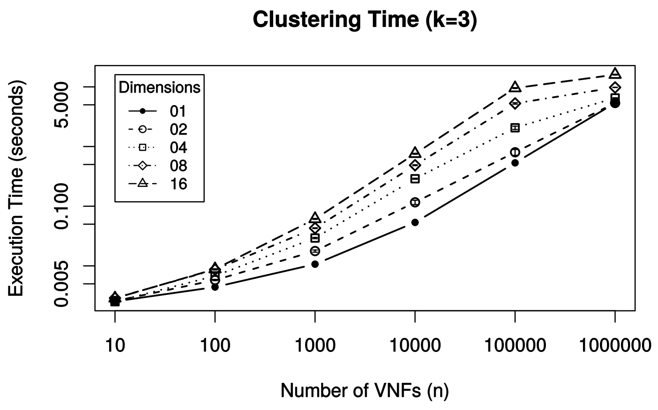

Fig. 7: Time to discover 3 clusters in different scenarios

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.

Acknowledgment

This work has been partially supported by the project "Um Ecossistema para Funes Virtualizadas de Rede" (#0409/2016), from CAPES Brazil, program FCT 039/2014.

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 Multi-variate 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/