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.

978-1-5090-0679-3/16/$31.00 ©2016 IEEE

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.

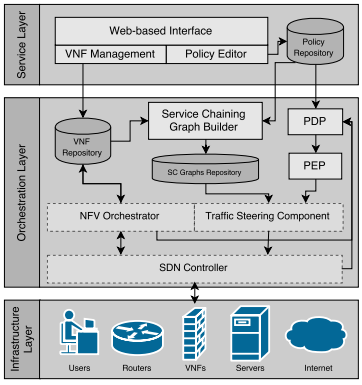

Fig. 1: Proposed System Architecture

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.

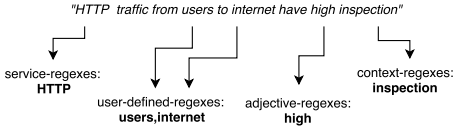

Fig. 2: Valid SLA Policy Parsing Example

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.

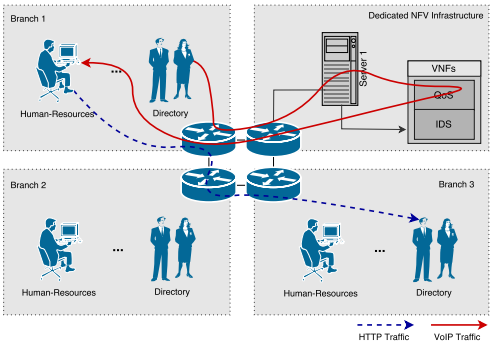

Fig. 3: Traffic Flows in a Common Enterprise Network

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.

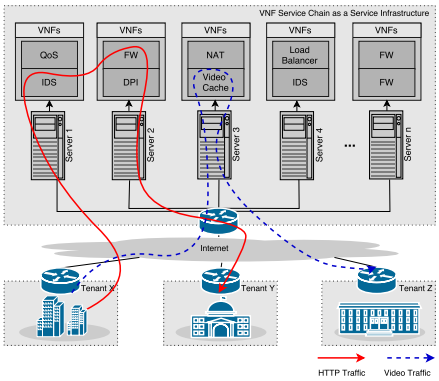

Fig. 4: Traffic Flows in a VNF-SCaaS Infrastructure

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.