Verifying the configuration of Virtualized Network Functions in Software Defined Networks

Johan Pelay
b<>com
Email: johan.pelay@b-com.com

Fabrice Guillemin
Orange Lab. (France)
Email: fabrice.guillemin@orange.com

Olivier Barais
INRIA, University of Rennes 1
Email: olivier.barais@inria.fr

***Abstract*—The deployment of modular virtual network functions (VNFs) in software defined infrastructures (SDI) enables cloud and network providers to deploy integrated network services across different resource domains. It leads to a large interleaving between network configuration through software defined network controllers and VNF deployment within this network. Most of the configuration management tools and network orchestrators used to deploy VNF lack of an abstraction to express Assume-Guarantee contracts between the VNF and the SDN configuration. Consequently, VNF deployment can be inconsistent with network configurations. To tackle this challenge, in this paper, we present an approach to check the consistency between the VNF description described from a set of structural models and flow-chart models and a proposed deployment on a real SDN infrastructure with its own configuration manager. We illustrate our approach on virtualized Evolved Packet Core function.**

I. Introduction

The emergence of virtualization techniques is revolutionizing the architecture of telecommunications networks. In particular, network functions, which were so far designed and deployed on dedicated hardware, are progressively migrating onto virtual infrastructures (see for instance [1]). The dissociation between functions and hosting hardware allows network operators to be more agile in the deployment of new services. The goal for a network operator is eventually to be able to deploy on demand network functions according to customer's needs and thus create new businesses.

Several functions are currently redesigned in order to be virtualized and flexibly instantiated on common hardware. This is notably the case of Radio Access Network (RAN) and Evolved Packet Core (EPC) functions for mobile networks. The virtualization of these two sets of functions enables a network operator to instantiate a mobile network on demand according to the needs of a company or a (virtual) mobile operator.

The virtualization of network functions urges network operators to change their business models. Instead of offering connectivity and information transport, they become IT technology providers and have to operate distributed storage and compute facilities in addition to their traditional role of connectivity provider. It is very likely that VNFs will be hosted and maybe split over several data centers disseminated throughout the network.

The decomposition of a global VNF into several components (or micro-services [2] [3]), which can be instantiated on distant servers, raises the problem of interconnecting them. This is deeply related to the method of programming the network. With the emergence of Software Defined Networking (SDN), it becomes possible to program a network by means of external controllers and thus to completely configure how different entities communicate between each other. In some sense, for a network operator, SDN and Network Function Virtualization (NFV) will rapidly become intimately interleaved and will raise the problem of consistency between the service which has to be offered by a VNF and the way the VNF is deployed within the network.

Beyond performance issues, which pose the problem of the placement of the various components of a VNF in the network so as to meet the associated grade of service objectives, a problem that network operators will have to solve in the next future is the consistency between the exchange of information between the components of a VNF and the configuration of the network. In this paper, we precisely address the problem of consistency between call flows and network configuration. We propose an approach, where the VNF call flow is attached to the VNF definition as a behavioral contract and we define the SDN behavior using the NetKAT formalism [4] [5] [6]. Then, on the basis of a deployment model of VNF micro-services on a real network infrastructure, we check the consistency between the VNF network assumption and the SDN guarantees [7].

NetKAT relies on the fact that network procedures acting on packets can be viewed as regular expressions on a certain alphabet (namely, that formed by the fields of packets). Then, by introducing the concept of history capable of tracking the progression of a generic packet through the network, notably depending on the forwarding decisions in each switch along the data path, and using the theory developed by Kozen in 90's [8] [9], it is possible to show that the system is decidable and provable. While the NetKAT framework is capable of proving that any given property is satisfied or not by the network, when configured by means of SDN and in particular by using OpenFlow, this formalism however does not account of VNFs. *To remedy this situation, we introduce in this paper an artifact, which allows us to describe the micro-services of a VNF as switches and to verify that the history of a generic packet in this augmented network topology, is compliant with the targeted call flow of a VNF.*

This paper is organized as follows: In Section II, we motivate our proposal by using a detailed example of EPC VNF. Through this example, we illustrate that a VNF provider cannot easily attach to its VNF modular implementation a contract that defines its assumptions regarding the network configuration. Section III provides an overview of our approach based on NetKAT and we illustrate its use on the virtual EPC (vEPC) use case. Section IV discusses related works. Section V presents some conclusions and future work.

II. Context and motivating example

A. Decomposition of a VNF into micro-services

A VNF is a complete software suite composed of several modules and accomplishing a number of tasks. The current trend is to decompose a complete VNF in the form of micro-services [2], [10], [3], [11], [12] interacting between each other, each micro-service executing a set of elementary tasks. Once a VNF is decomposed into micro-services, the subsequent task is to instantiate them onto a virtualized architecture.

Before proceeding to the instantiation phase, let us stress the fact that a micro-service is a software package that is developed by independent entities (companies specialized in software development or open-source communities) and used as plug-and-play by a network operator. Engineers, who are in charge of software development or who design services, do a job different from that of network administrators and have different skills.

This gap between these two worlds can cause misunderstandings or configuration errors between the wanted VNF logical architecture [13] and the real set up. The services are often designed without taking into account the use of the network by others, which can lead to problems such as security of communications, latency, congestion, etc.

There exists several configuration management tools and network orchestrators, that allow operators to quickly describe VNF architectures, i.e., the micro-services using VMs or containers, micro-service configuration and assembly. We can cite Ansible[1], Chef[2], Puppet[3], Docker Compose[4], etc. Currently to the best of our knowledge, only openMANO and of course the SONATA NFV service platform propose a way to declare network configuration assumptions, that has to be declared before the deployment[5].

In the following, we pay special attention to the way the micro-services of a VNF are interconnected. When dealing with the implementation correctness of a VNF, we can definitely identify two aspects:

- The semantic correctness: The various micro-services exchange messages between them according to a given protocol. The semantic checking of the VNF amounts to verify the correctness of the implementation of the protocol. This can be done off-line when coding the VNF in the form of micro-services. Usual model checking tools [14] can be used to check behavioral consistency between services [15].
- The correctness of the exchange of information: When micro-services are implemented on various servers, they have to communicate between them across the network. Micro-services are hosted by a server attached to the network. The key point is to check that the messages are correctly exchanged between micro-services interconnected by a network configured by means of OpenFlow.

In this paper, we focus on the second issue.

[1]https://www.ansible.com/
[2]https://www.chef.io/
[3]https://puppet.com/
[4]https://docs.docker.com/compose/
[5]In OpenMANO, *connection_point* can be used to declare that a service must communicate with another service.

B. Motivating example: A modular implementation of vEPC

1) The various functions of vEPC: An EPC is composed of data and control plane functions as depicted in Figure 1, which displays the various functions for cellular and WiFi radio access. In the following, we focus on 4G access composed of HSS (Home Subscriber Server), S/PGW (Serving/Packet data network Gateway), MME (Mobility Management Entity). Note when considering non cellular access, additional modules are necessary (e.g., ePDG, WiFi controllers, etc.).

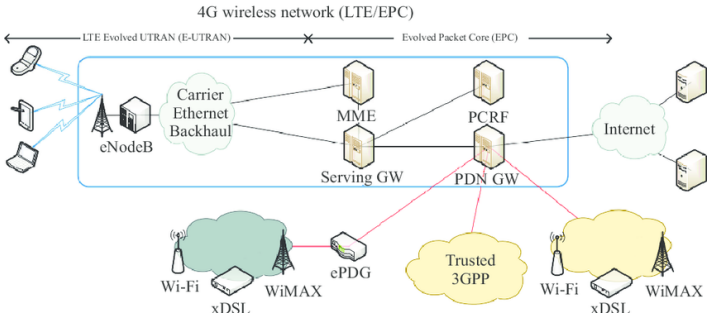

Fig. 1. Data and control plane modules for radio access [16].

2) Attach and Authentication: To get attached to the radio network, a UE (User Equipment) identified by its SIM card connects to an eNodeB base station. Before authorizing a UE to access the network its identity is verified by the MME thanks to a database that describes the entities of the network and contains the list of users and the associated rights and permissions as well as the current sessions (HSS). For the sake of conciseness in our next examples, we will focus on those first steps.

3) Default Radio Bearer Setup : UE's mobility is managed by the 4G control plane. A packet is routed by using an address that is generally linked to a fixed location. The solution chosen in 4G is to pass traffic through a Packet Gateway (PGW). When the UE moves and changes the base station, the PGW is informed of its new location by the MME.

According to the LTE standard, the S/PGW must provide an IP address to the client. This step is divided into two distinct parts: on the one hand, the request followed by the transmission of the IP address and on the other hand, the choice of the IP address. This choice can be made directly by the S/PGW if it has a database with the pool of IP addresses or through the query from a DHCP server. The IP address can be returned to the customer via the MME, or only after the opening of the tunnel (bearer) by the S/PGW when the client requests it through a tunnel.

The call flow of the attachment of a UE is depicted in Figure 2. The various elements of the EPC has to exchange information, which has to be forwarded through the network. In current networks, all servers (MME, HSS) and the data plane network elements (eNodeB, S/PGW) have fixed IP addresses and routing is static. The challenge of NFV is to dynamically implement these functions on data centers.

C. Implementation issues

We tested the deployment of an open source EPC (as illustrated in Figure 3) in the Network Architecture Lab at b<>com, namely the Open Air Interface (OAI) EPC[6] [17]. For deploying this

[6]http://www.openairinterface.org/?page_id=864

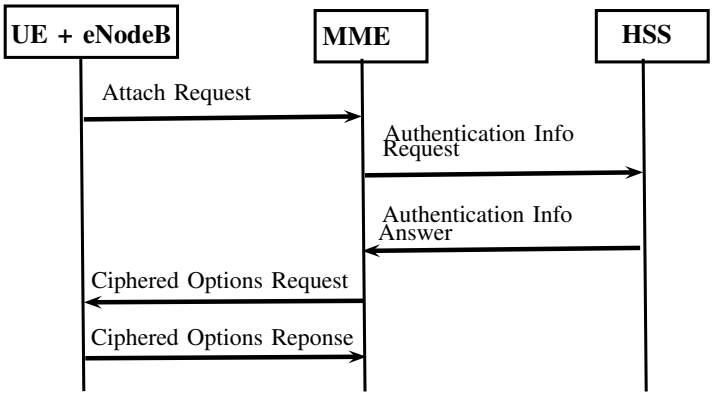

Fig. 2. UE attachment call flow.

vEPC with a simple configuration manager, we used Ansible scripts to define roles and a Vagrant file to mount and connect all of the resources. A role is defined by a list of files to be installed or imported, a list of values that will change for each resource associated with the role (name, IP ...). The same type of equipment usually plays several roles, a playbook lists them for all types and passes to the associated roles the variables to apply for each resource. The configuration of the various services as well as the routing tables are fixed and written in files that are copied by Ansible after the launch of the VMs.

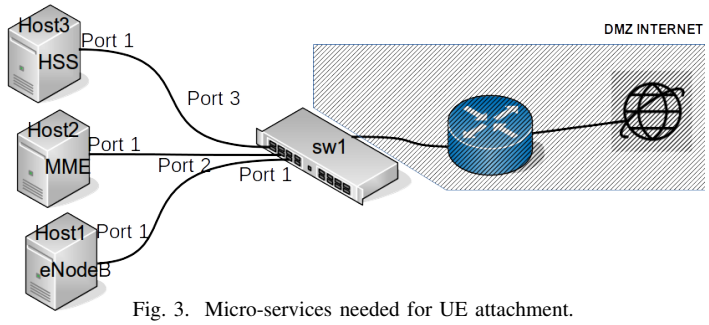

Fig. 3. Micro-services needed for UE attachment.

A total of 1047 lines in Ansible code and 344 of configuration files was necessary to deploy the six micro-services that form the vEPC as well as the network equipment and their controller. No verifications could be made to ensure that the network configuration allows for the necessary exchanges. When the deployment was done on a different architecture than the development lab (namely, when implementing the vEPC on the platform of another project), network problems appeared (loops). Tests performed when writing configurations are valid only on the network tested at a given time.

This simple example incites us to develop a methodology for testing the consistency between the decomposition of a VNF in terms of micro-services and its deployment in the multi-cloud environment.

III. Proposed solution

The NFV approach promises to be able to encapsulate network functions into reusable boxes that operates predictably without requiring network operators know the details of how it does so. To detect inconsistencies between NFV assumptions and real SDN configurations, this section introduces an extended model of VNF with software contracts. This section shows how these contracts could be used in combination with existing approach such as NetKat, to check the overall consistencies of the VNF deployment. We define below the four main steps of our approach and illustrate each of these steps on the EPC case study:

1) Define a reusable VNF model (subsection III-A);
2) Define the *augmented topology* of the network and IT infrastructure (subsection III-B);
3) VNF deployment model (subsection III-C);
4) Check the consistency of the VNF deployment model (subsection III-D).

A. Step 1 - Reusable VNF model

We first introduce an abstract definition of a VNF.

Definition 1: A VNF is defined as :

- A set of micro-services $S = \{s_1, s_2, ..., s_i\}$.
- A set of Hubs $H = \{h_1, h_2, h_i\}$, where H represents a logical links between a set of services and meta-data. For example, if $h1 = \{services : \{s_1, s_2\}, secured : false\}$ and $h_2 = \{services : \{s_2, s_3\}, secured : false\}$ means that s_1 can send message to s_2, s_2 can send message to s_3 but s_1 can not exchange message with s_3 and cannot view the messages between s_2 and s_3. $h_2 = \{services : \{s_3, s_4, s_5\}, secured : true\}$ means that communication between s_3, s_4, s_5 must be encrypted.
- A behavioral contract (see definition 2).
- A set of micro-service implementation.

In the above definition, we have used the concept of behavioral contract defined as follows.

Definition 2: A behavioral contract is defined through a set of messages. Each message is a tuple $msg = (s_1, s_2, order, \{a_1, a_2, ..., a_n\})$ where:

- s_1 is the micro-service source of the message,
- s_2 is the target micro-service of the message,
- $order$ is the order number of the message in the corresponding sequence diagram structure tree,
- $a_1, a_2, ..., a_n$ is a set of routing rule actions (*addition*, *remove*, *modification*) that can be triggered before the message reception or after its emission (see Definition 3).

All the order numbers of messages in the sequence diagram construct a partial order. The order number of a message is given according to its position in the tree. The root node corresponds to the starting micro-service of a sequence diagram.

Definition 3: A routing rule action a is defined using NetKAT syntax, it has a name and it handles virtual nodes and virtual ports of a VNF. A virtual node ($VN(s_i)$) maps the nodes where the micro-service s_i is deployed. A virtual port ($VP_k(s_i)$) maps the port k of the node where the micro-service s_i is deployed.

1) vEPC attachment illustration: To illustrate this formalism, a vEPC can be characterized as follows.

Proposition 1: A vEPC can be described as a set $S = \{eNodeB, MME, HSS\}$ with

- a set of hubs $H = \{h_1, h_2\}$ with
  $h_1 = \{services : \{eNodeB, MME\}, secured : false\}$
  $h_2 = \{services : \{MME, HSS\}, secured : false\}$

- a behavioral contract

$$BC = \{Attach\_Request, Auth\_info\_Req, Auth\_info\_Ans, Ciphered\_Req, Ciphered\_Rep\},$$

with: $Attach\_Request = \{eNodeB, MME, 1\}$,
$Auth\_info\_Req = \{MME, HSS, 2\}$,
$Auth\_info\_Ans = \{HSS, MME, 3\}$,
$Ciphered\_Req = \{MME, eNodeB, 4\}$,
$Ciphered\_Rep = \{eNodeB, MME, 5\}$.

We could show that the vEPC has three micro-services connected through two different unsecured hubs (h_1, h_2). The behavioral contract contains five messages: $Attach\_Request$, $Auth\_info\_Req$, $Auth\_info\_Ans$, $Ciphered\_Req$ and $Ciphered\_Rep$.

B. Step 2 - Augmented topology based on NetKAT

To describe the network configuration, we directly propose to use NetKAT. In the following subsection, we recall the basic elements of NetKAT.

1) Basic elements of NetKAT: The basic approach of NetKAT is to suppose that functions implemented in a network can be viewed as regular expressions acting on packets. In this framework, a packet pk is a series of fields, namely

$$pk ::= \{f_1 = v_1, \ldots, f_k = v_k\}$$

where f_i, $i = 1, \ldots, k$ are fields expressed as series of bits. The most common fields used in the context of IP networks are source and destination IP addresses, source and destination port numbers, protocol types, DiffServ Code Point (DSCP), etc. Moreover, to track the position of a packet in the network, NetKAT introduces two additional fields: the switch label and the port number at which the packet appears when arriving at the switch. The key observation is that these two fields are changed at each switch while other fields are relevant end-to-end (except the DSCP field which may be updated inside the network in the case of untrusted marking but which should have in theory an end-to-end value). With the above definition, packets form an alphabet of 2^N elements if $N = |f_1| + \ldots + |f_k|$. This number is potentially very high but most network procedures act only on a restricted number of fields (IP addresses in case of rerouting inside the network, switch and port labels).

The fundamental idea of NetKAT is recognize that usual network functions such as forwarding, rerouting, firewall, etc. can be viewed as regular expressions which can be:

- either predicates ($f = v$, where f is a field and v a given value);
- or else policies, for instance updating a field ($f \leftarrow v$).

The basic procedures (policies and predicates) of NetKAT are given in Table I. The Kleene star

$$p^* = \sum_{n \geq 0} \underbrace{p \ldots p}_{n \text{ times}}$$

is the sum of the finite iterates of procedure p. The dup policy is introduced to duplicate packets in the construction of histories

TABLE I
PREDICATES AND POLICIES IN NETKAT.

Predicates			**Policies**		
$a, b ::=$	1	Identity	$p, q ::=$	a	Filter
	0	Drop		$f \leftarrow v$	Modification
	$f = v$	Test		$p + q$	Union
	$a + b$	Disjunction		$p \cdot q$	Sequential composition
	$a.b$	Conjunction		p^* &	Kleene star
	$\neg a$	Negation		dup	Duplication

(namely prepends the same packet at an history); histories are introduced below and are instrumental in the capability of NetKAT of proving properties of the network.

The set of policies with identity 1 (no actions) and null 0 (packet drop) procedures and equipped with $+$, $\cdot$, the star * operations constitutes a Kleene algebra. The set of predicates with the identity 1 and the drop 0 equipped with the operations $+$, $\cdot$ and $\neg$ is a Boolean algebra. Predicates with $+$ and $\cdot$ operations is a subalgebra of policies. The set of policies and predicates with the above operations is a Kleene Algebra with Tests (KAT).

With the above notation, a firewall which drops all packets towards a given IP address (say, A_0) can be written as

$$(@IP_d = A_0) \cdot 0,$$

where as stated above, 0 is the filtering policy that drops all packets. Similarly, forwarding a packet from port pt_1 on switch A to port pt_2 on switch B reads

$$(\text{SW} = A, pt = pt_1) \cdot (\text{SW} \leftarrow B, pt \leftarrow pt_2)$$

To record the path taken by a packet through the network, NetKAT introduces the concept of history that is the list of states occupied by a generic packet when traversing the network. A history h has the form $< pk_1, \ldots, pk_n >$ where pk_i is the state of packet pk at the $n - i$ switch (the list reads from the right to the left). All NetKAT predicates and policies act on the packet-history to create a new history (possibly empty if the packet is dropped). Policies and predicates act on histories as detailed in Section 3 of [4].

More precisely, a predicate on a history h returns a singleton $\{h\}$ or the empty set $\{\}$. A field modification $(f \leftarrow v)$ returns a singleton history in which the field f of the current packet has been set equal to v. Thus, usual predicates and policies induce functions on histories. The function induced by policy p is denoted by $[[p]]$. A function $[[p]]$ is from the set of histories H to $\mathcal{P}(H)$, the set of parts of H.

Using the same formalism, NetKAT describes the network topology in order to check if rules are not trying to make links that do not physically exist (as described in the example provided in Proposition 2).

The key properties of NetKAT is that this language is sound and complete. This means that with KAT axioms and the NetKAT axioms (see Section 2 of [5]), every equivalence provable by using NetKAT axioms also holds in the equational model (soundness) and conversely, every equivalence in the equational model is provable with NetKAT axioms (completeness); these two statements are proved in Section 4 of [4].

In particular, it is possible to prove that a packet follows a given route (namely a sequence of switches in the network). This

simple remark motivates us to introduce an augmented version of the network.

2) Augmented network topology: The network is configured by means of SDN, for instance OpenFlow. By acquiring the configuration of the network (namely, the OpenFlow rules pushed in the network elements via the controllers), it is possible to completely translate the network configuration in NetKAT. We can then abstract the network topology in terms of nodes, ports and links.

To build the augmented topology of the virtualized infrastructure, we consider micro-services exchanging messages as (virtual) switches exchanging packets. Indeed, from a transport of view, micro-services receive messages, process them and transmit them to other micro-services or to the end users. Everything happens as if micro-services were switches and messages were (virtual) packets.

On the basis of the (physical) network topology and the (virtual) topology of micro-services, we are able to build the augmented topology of the network combined with VMFs. With this artifact and the power of the NetKAT formalism, we are able to verify that virtual packets are routed through the augmented topology so that the call flows of a VNF can be implemented.

3) Illustration with the UE attachment: The micro-services are illustrated in Figure 3 and the associated call flow in Figure 2. We introduced an augmented topology with four nodes ($host1$, $host2$, $host3$, $sw1$), each $host_i$ has one port, sw_1 has three ports. There is a link between each $host_i$ and sw_1. Considering micro-services as switches, the augmented topology of our example can be described as follows.

Proposition 2: The augmented topology of the UE attachment procedure for the example depicted in Figure 3 is

$$\begin{aligned} &t = (sw = host1 \cdot pt = 1 \cdot sw \leftarrow sw1 \cdot pt \leftarrow 1) \\ &+(sw = sw1 \cdot pt = 1 \cdot sw \leftarrow host1 \cdot pt \leftarrow 1) \\ &+(sw = host3 \cdot pt = 1 \cdot sw \leftarrow sw1 \cdot pt \leftarrow 2) \\ &+(sw = sw1 \cdot pt = 2 \cdot sw \leftarrow host3 \cdot pt \leftarrow 1) \\ &+(sw = host2 \cdot pt = 1 \cdot sw \leftarrow sw1 \cdot pt \leftarrow 3) \\ &+(sw = sw1 \cdot pt = 3 \cdot sw \leftarrow host2 \cdot pt \leftarrow 1) \end{aligned}$$

We could combine this topology with a deployment model of a new VNF to check the consistency between the VNF model and the current network topology before acting the real VNF deployment.

C. Step 3 - VNF deployment model

To deploy a reusable VNF on a real network, the deployment model creates a mapping between each micro-services belonging to a VNF model and a node belonging to the network configuration model.

Definition 4: A deployment model D is defined as $D = \{m_1, m_2, ..., m_i\}$ and m is tuple such as $m = \{n_i, s_j\}$.

In our illustrative use case (UE attachment), the proposed mapping is as follows:

Proposition 3: The VNF deployment model for the UE attachment procedure of the example depicted in Figure 3 is $D = \{m_1, m_2, m_3\}$, $m_1 = \{host1, eNodeB\}$, $m_2 = \{host2, HSS\}$, $m_3 = \{host3, MME\}$.

The micro-service $eNodeB$ is deployed on the host $host1$. The micro-service HSS is deployed on the host $host2$. The micro-service MME is deployed on the host $host3$.

Once the topology and the VNF description are combined we can write the desired networks rules in NetKAT which will then be translated in OpenFlow.

Proposition 4: The NetKAT policies for the UE attachment procedure in the example in Figure 3:

$$\begin{aligned} &p = (sw = sw1 \cdot pt = 1 \cdot sw \leftarrow MME \cdot pt \leftarrow 1) \\ &+(sw = sw1 \cdot pt = 2 \cdot dst = HSS \cdot sw \leftarrow HSS \cdot pt \leftarrow 1) \\ &+(sw = sw1 \cdot pt = 2 \cdot dst = eNodeB \\ &\cdot sw \leftarrow eNodeB \cdot pt \leftarrow 1) \\ &+(sw = sw1 \cdot pt = 3 \cdot sw \leftarrow MME \cdot pt \leftarrow 1) \\ &+(dst = HSS \cdot src = eNodeB \cdot 0) \end{aligned}$$

D. Step 4 - Checking consistency of a VNF deployment model

1) Running call flow on augmented topology: For a call flow of a VNF, a virtual packet is introduced and a history is created to record the journey of this virtual packet in the augmented network. NetKAT can then be used to prove that the exchange in the call flows are achieved by the virtual packet. This method can notably be used to prevent from loops, undue packet discard, missing forwarding rules, etc.

To check the compliance between histories and call flow, we basically built two message sequences traces and we check that it exists a weak bisimulation [18] relation between histories and call flows to check trace equivalence.

2) History applied to our use case: To illustrate the last step, we focus on *Attach and Authentication* behavioral contract defined in Figure 2. In NetKAT formalisms, for a packet from eNodeB virtual switch, we obtain the following history (recall that it must be read from last to first):

$$\begin{aligned} &pk ::= \{ \\ &pk_1[src{:=}eNodeB; dst{:=}MME; sw{:=}sw1; port{:=}2], \\ &pk_2[src{:=}eNodeB; dst{:=}MME; sw{:=}eNodeB; port{:=}1], \\ &pk_3[src{:=}MME; dst{:=}eNodeB; sw{:=}sw1; port{:=}1], \\ &pk_4[src{:=}MME; dst{:=}eNodeB; sw{:=}MME; port{:=}1], \\ &pk_5[src{:=}HSS; dst{:=}MME; sw{:=}sw1; port{:=}3], \\ &pk_6[src{:=}HSS; dst{:=}MME; sw{:=}HSS; port{:=}1], \\ &pk_7[src{:=}MME; dst{:=}HSS; sw{:=}sw1; port{:=}3], \\ &pk_8[src{:=}MME; dst{:=}HSS; sw{:=}MME; port{:=}1], \\ &pk_9[src{:=}eNodeB; dst{:=}MME; sw{:=}sw1; port{:=}2], \\ &pk_{10}[src{:=}eNodeB; dst{:=}MME; sw{:=}eNodeB; port{:=}1]\} \end{aligned}$$

Based on this history, we could build two label transition systems (LTS) in which the label is defined using the source name and the target name. Then we check the trace inclusion between the VNF behavioral contract and the NetKAT history. To check the trace inclusion of LTSs we check a global property of weak bi-simulation between LTS and it is known that weak simulation implies trace inclusion [19]. For tooling the approach, on top of NetKAT, we use the LTSA [20] model checker. LTSA can check the weak simulation by representing the VNF behavioral contract as the safety property process.

IV. Related Work

The interleaving between SDN and NFV is one of the challenge that has recently been investigated in technical literature. In [21],

Medhat *et al* explore the limitations of current service function chaining approaches in next generation networks in terms of architectural and conceptual research work by providing a brief analysis of each solution in the state of the art. This article also proposes some new research directions. With regard to placement of functions, they discus the automatic placement/migration of VNF to ensure their correct execution. In such a scenario, guaranteeing the consistency between the VNF assumptions and the network configuration guarantees could be used as an oracle to accept a placement/migration.

Closer to our approach, in [22], Spinozo *et al.* check that the functionalities implemented in the VNF are not disturbed by the modifications made by the middle-boxes or other VNFs. For this purpose, they use a satisfiability modulo theories (SMT) solver that quickly provides formal proof before a deployment. The topology of the network is not managed but only the network graph/service chaining. With respect to their approach, we focus in this paper on possible errors in forwarding rules that can cause loop, reachability issue or security breach, etc. Our work could easily be extended to packet modification by middle-boxes and VNFs viewed as virtual switches in our augmented topology.

In [23], Shin *et al.* provide formal foundations for supporting the development of reliable network services. The solution proposed uses a packet based Algebra of Communicating Shared Resources to check whether there is any inconsistency between chosen specifications and the implementation. All the above cited studies stress the fact, as we do in this paper, that there is a real need for formally proving the correctness of the implementation of VNFs in an SDN context.

V. Conclusion

We have addressed in this paper the correctness of the deployment of VNFs when distributed on distant data centers. We have advocated for the development of a unified view of all resources involved in the implementation and deployment of VNFs, notably how they are interconnected. On the basis of this view, we have introduced the concept of augmented topology where micro-services appear as switches. By associated virtual packets with call flows of VNFs it is possible to use the NetKAT formalism to verify that histories of virtual packets are compliant with the call flows of a VNF and thus the VNF is correctly implemented.

We furthermore believe that such an approach is utmost relevant in the development of ONAP (Open Network Automation Project) [24], which aims at automating the creation and the instantiation of VNFs. By adopting the solution proposed in this paper, ONAP will be capable of safely instantiating VNFs in a network programmed by means of SDN. ONAP is in the first development phase but has already identified a number of features which are required for the automatic creation and instantiation of VNFs. In particular, ONAP aims at developing a holistic view of network resources, including traditional assets of the network (types of connectivity and bandwidth) managed with OpenDayLight together with IT resources (storage and compute) managed by OpenStack. Hence, ONAP can develop a unified view of all resources so as to optimize and configure resources.

The developed solution is valid for a static environment, where micro-services do not migrate from one data center to another. Building such a model, in particular the network configuration model, that can be highly dynamic, could be error-prone. But we can get it through system introspection.

References

[1] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90–97, 2015.

[2] D. Namiot and M. Sneps-Sneppe, "On micro-services architecture," *International Journal of Open Information Technologies*, vol. 2, no. 9, pp. 24–27, 2014.

[3] D. Jaramillo, D. V. Nguyen, and R. Smart, "Leveraging microservices architecture by using docker technology," in *SoutheastCon, 2016*. IEEE, 2016, pp. 1–5.

[4] C. J. Anderson, N. Foster, A. Guha, J.-B. Jeannin, D. Kozen, C. Schlesinger, and D. Walker, "NetKAT: Semantic foundations for networks," ser. POPL '14. New York, NY, USA: ACM, 2014, pp. 113–126.

[5] N. Foster, D. Kozen, M. Milano, A. Silva, and L. Thompson, "A coalgebraic decision procedure for NetKAT," in *Proc. POPL 2015*, 2015.

[6] N. Foster, D. Kozen, K. Mamouras, M. Reitblatt, and A. Silva, "Probabilistic netkat," in *Proceedings of the 25th European Symposium on Programming Languages and Systems - Volume 9632*. New York, NY, USA: Springer-Verlag New York, Inc., 2016, pp. 282–309.

[7] C. Chilton, B. Jonsson, and M. Z. Kwiatkowska, "Assume-guarantee reasoning for safe component behaviours." in *FACS*, vol. 12. Springer, 2012, pp. 92–109.

[8] D. Kozen, "A completeness theorem for kleene algebras and the algebra of regular events," *Inf. Comput.*, vol. 110, no. 2, pp. 366–390, May 1994.

[9] ——, "Kleene algebra with tests," *ACM Trans. Program. Lang. Syst.*, vol. 19, no. 3, pp. 427–443, May 1997.

[10] K. Katsalis, N. Nikaein, E. Schiller, R. Favraud, and T. I. Braun, "5g architectural design patterns," in *Communications Workshops (ICC), 2016 IEEE International Conference on*. IEEE, 2016, pp. 32–37.

[11] W. John, F. Moradi, B. Pechenot, and P. Sköldström, "Meeting the observability challenges for vnfs in 5g systems."

[12] A. Sheoran, X. Bu, L. Cao, P. Sharma, and S. Fahmy, "An empirical case for container-driven fine-grained vnf resource flexing," in *Network Function Virtualization and Software Defined Networks (NFV-SDN), IEEE Conference on*. IEEE, 2016, pp. 121–127.

[13] P. B. Kruchten, "The 4+ 1 view model of architecture," *IEEE software*, vol. 12, no. 6, pp. 42–50, 1995.

[14] D. Giannakopoulou and J. Magee, "Fluent model checking for event-based systems," in *ACM SIGSOFT Software Engineering Notes*, vol. 28, no. 5. ACM, 2003, pp. 257–266.

[15] A. Beugnard, J.-M. Jézéquel, N. Plouzeau, and D. Watkins, "Making components contract aware," *Computer*, vol. 32, no. 7, pp. 38–45, 1999.

[16] K. Gierlowsky, "Ubiquity of client access in heterogeneous access environment," *Journal of Telecommunications and Information technology*, 2014.

[17] P. Ravali, S. K. Vasudevan, and R. Sundaram, "Open air interface–adaptability perspective," *Indian Journal of Science and Technology*, vol. 9, no. 6, 2016.

[18] R. van Glabbeek and U. Goltz, "Equivalence notions for concurrent systems and refinement of actions," in *Mathematical Foundations of Computer Science 1989*. Springer, 1989, pp. 237–248.

[19] R. Milner, "A calculus of communicating systems," 1980.

[20] J. Magree, "Behavioral analysis of software architectures using ltsa," in *Software Engineering, 1999*. IEEE, 1999, pp. 634–637.

[21] A. M. Medhat, T. Taleb, A. Elmangoush, G. A. Carella, S. Covaci, and T. Magedanz, "Service function chaining in next generation networks: State of the art and research challenges," *IEEE Communications Magazine*, vol. 55, no. 2, pp. 216–223, February 2017.

[22] S. Spinoso, M. Virgilio, W. John, A. Manzalini, G. Marchetto, and R. Sisto, *Formal Verification of Virtual Network Function Graphs in an SP-DevOps Context*. Cham: Springer International Publishing, 2015, pp. 253–262.

[23] M. K. Shin, Y. Choi, H. H. Kwak, S. Pack, M. Kang, and J. Y. Choi, "Verification for nfv-enabled network services," in *2015 International Conference on Information and Communication Technology Convergence (ICTC)*, Oct 2015, pp. 810–815.

[24] "Open network automation project," https://www.onap.org/, accessed: 2017-07-09.