Self-Modeling based Diagnosis of Services over Programmable Networks

Jose Manuel Sanchez Vilchez, Imen Grida Ben Yahia, Noel Crespi

Self-Modeling based Diagnosis of Services over Programmable Networks

José Manuel Sánchez, Imen Grida Ben Yahia
Orange Labs
Paris, France

Noël Crespi
Télécom SudParis, CNRS UMR5157
Evry, France

Abstract— **In this paper, we propose a multi-layer self-diagnosis framework for networking services within SDN and NFV environments. The framework encompasses three main contributions: 1) the definition of multi-layered templates to identify what to supervise while taking into account the physical, logical, virtual and service layers. These templates are also finer-granular, extendable and machine-readable; 2) a self-modeling module that takes as input these templates, instantiates them and generates on-the-fly the diagnosis model that includes the physical, logical, and the virtual dependencies of networking services; 3) a service-aware root-cause analysis module that takes into account the networking services' views and their underlying network resources observations within the aforementioned layers. We also present extensive simulations to prove the fully automated, finer granularity and reduced uncertainty of the root cause of networking services failures and their underlying network resources.**

Keywords— self-modeling; self-diagnosis; Bayesian networks; SDN; NFV; SDI; fault management; alarm correlation; fault-isolation; fault-localization

I. INTRODUCTION

The advent of programmable networks, with SDN (Software-Defined Networking) and NFV (Network Functions Virtualization) is accelerating faster and faster the transformation of current network leading to rethink network and service management and operations.

SDN proposes to transition from network configurability to network programmability through network abstractions, open interfaces and the separation of control and data plane. Meanwhile, NFV proposes to virtualize network functions. It mainly aims to remove the vendor lock-in barrier and allows networking services to be flexibly instantiated and scaled according to network traffic demands at run-time.

SDN and NFV are thought to be "better together" by the IT and telecommunication industry. Nevertheless, the introduction of the SDN controller within the NFV is still under discussion, evidenced by the lack of consensus on the position of the SDN controller within the NFV framework [1].

Fault management operations particularly, emerge as cornerstone to provide the SDN and NFV promises: In fact, the SDN controller whether it is centralized or distributed is a point of failure and thus its underlying network is impacted.

Moreover, networking services will rely on a dynamic placement and migration of the virtual network functions as well as an elastic usage of the compute, storage and networking resources.

Therefore, in SDN and NFV, the high network dynamicity provided by SDN becomes even higher when combined with NFV, since the VNF can be scaled, instantiated, deleted, and migrated. Thus, service dependencies from the underlying resources are in a continuous change and need dynamic management. In response to these challenges, we focus here on the diagnosis as a key operation among others to ensure the smooth functioning of networking services relying on SDN and NFV principles.

The paper contribution is a self-diagnosis framework that is relying on multi-layered and finer granular templates for diagnosing the dynamic networking services within an SDN and NFV environment. The Self-diagnosis framework encompasses 1) the definition of multi-layered templates to identify what to supervise while taking into account the physical, logical, virtual and service layers. These templates are also finer-granular, extendable and machine-readable; 2) a self-modeling module that takes as input these templates, instantiates them and generates on-the-fly the diagnosis model that includes the physical, logical, and the virtual dependencies of networking services; 3) a service-aware root-cause analysis module that takes into account the networking services' views and their underlying network resources observations within the aforementioned layers. The paper is organized as follows. Section II summarizes the related work. Section III overviews the self-diagnosis framework of networking services and its principles modules. Section IV details the self-modeling module. Section V details the root cause analysis module. Section VI evaluates the self-diagnosis framework and its performance. Section VII concludes the paper and highlights the future work.

II. RELATED WORK

In this section, we first present the related work on model-based self-diagnosis approaches in general and then we focus on the related work of fault management defined for SDN and NFV so far. We finally position our contributions with respect to them.

A. Model-based self-diagnosis

Model-based self-diagnosis relies on a dependency graph which indicates how faults propagate through the network and eventually lead to service failures. In model-based self-diagnosis approaches, a Root Cause Analysis algorithm (RCA) exploits this dependency graph to calculate the root

978-1-4673-9486-4/16/$31.00 ©2016 IEEE

cause. However, in the state of the art, much more attention is paid to the RCA algorithms in use, to the detriment of the generation of the dependency graph, which in most cases, is manually built from operational team's knowledge. This manual generation is valid for static network topologies and for statically predefined services, but not for dynamic and elastic networks such as those expected with SDN and NFV, where the dependency graph must be continuously updated. This mechanism is coined by Honkonnou et. al. in [6] as *self-modeling*.

We propose to classify the model-based self-diagnosis approaches into topology-aware and service-aware, which we define hereafter.

Topology-aware approaches: Topology-aware approaches build the dependency graph from the network topology and this graph only considers how faults in network resources could impact other network resources. The impact of faults in network resources on services or clients is not then considered. As examples, Steinder and Sethi [3] propose a self-diagnosis approach for end-to-end services over bridged networks, based on a self-modeling approach to build the diagnosis model from the network topology. However, the authors do not address the impact of the network faults on the service layer. Bennacer et al. in [4], propose a self-diagnosis approach for VPN-based services and utilize a self-modeling approach that computes the dependencies among the physical symptoms of each network node through statistical tests. They proposed a hybrid RCA module based on the combination of Bayesian Networks (BN) with Case-Based reasoning to reduce the high diagnosis time and increase the low accuracy provided by the BN algorithm. The impact of faulty nodes on the VPN service was not addressed. In our previous work [2], we proposed a self-diagnosis framework that generated on-the-fly the dependency graph from the network topology and the logical resources running on top of nodes. The impact of faulty physical and logical nodes on services was not studied. As a result, the diagnosis is focused on the entire network topology and logical resources so the uncertainty on the root cause is higher when the diagnosed network topology becomes large.

Service-aware approaches: We define service-aware approaches as an extension of topology-aware approaches in order to take into account the impact of faulty network resources on services as well as the clients using them. The diagnosis can then focus on faults leading to service failures and impacting on client experience. As examples, Bahl et. al. propose a self-diagnosis algorithm for IT infrastructures [5], based on a self-modeling approach that computes the service dependencies in a given network topology when clients access to some database in the IT infrastructure. The diagnosis focuses on faults impacting the clients of the IT infrastructure. Hounkonnou et al. propose a self-diagnosis approach for IMS networks [6] based on a self-modeling approach that utilizes a multi-layered model based on the four IMS layers. First, the self-modeling approach builds the dependency graph from the affected service and its underlying resources over a fixed network topology, and then extends this graph with those clients which service observations reduce the uncertainty on the root cause.

B. Fault manangement in SDN and NFV

The existing fault management solutions for SDN are mostly OpenFlow-based and only handle faults in the data plane. Only few solutions focus on the control plane and especially on the SDN controller in itself [7]. For instance, Fonseca et. al. in [8] propose an Openflow-based approach to detect failures on the SDN controller and use replication techniques to transition to a back-up controller. However, it only considers when the SDN controller is compromised, leaving out faults in the rest of components in the network. There is a lack in multi-layer diagnosis approaches covering control and data planes in SDN, although some troubleshooting mechanisms exist such as NICE [9] for testing OpenFlow applications, NDB [10] for tracing packets, OFRewind [11] for finding invalid controller actions and packet parsing errors, STS for analyzing software bugs, or NetSight [12] for detecting forwarding loops. Turchetti and Duaerte in [13] propose a failure detector for SDN. However, the authors assume that the SDN controller does not crash and only focus on faults in the data links. Gheorghe et. al. [14] propose SDN-RADAR, a multi-agent distributed network troubleshooting mechanism for SDN that identifies faulty network links impacting user experience. However, this approach only focuses on links in the data plane and does not deal with large and dynamic network topologies.

In NFV, there are some management platforms [15] such as Cloud4NFV or NetFATE. Miyazawa et. al. in [22] proposed a fault detection mechanism based on Self-Organized Maps (SOM) to detect failures in NFV-based services. The authors propose a failure model to explain degradations in VNFs such as network congestion and memory leaks. However, SOM parameters are tuned manually and in advance in accordance with the type of failure to detect. Another important aspect is fault isolation, as identified by Esteves et. al. in [16] and Chowdhury et. al. in [17], as an open research field, where virtual resources are dynamically mapped over one common physical infrastructure and faults may propagate among networking services. With this concern, Schöller et. al. in [21] propose an information model to ensure a resilient deployment of VNF composing complex services in NFV where redundant components are strategically placed to avoid cascade effects. However, this approach does not ensure resilience in the operational phase. In addition, none of these approaches for NFV considers SDN as underlying architecture, so those do not address the specific fault management and dynamicity challenges of combined SDN and NFV infrastructures.

C. Positioning of our contributions

Our main contributions are presented and positioned hereafter:

Multi-layer self-diagnosis: We propose a multi-layer self-diagnosis framework to diagnose networking services over combined NFV and SDN environments, which to the best of our knowledge has not been tackled before. In this paper, we

extend our previous work [2], by diagnosing and correlating more layers i.e. virtual and services layers, while considering their dynamic dependencies with the underlying logical and physical resources. Our approach utilizes a probabilistic and multi-layered dependency graph, like Bahl in [5] and Hounkonnou in [6]. In our approach, this graph is adapted to SDN and NFV specificities, and it covers the diagnosis of physical, logical, virtual resources and the corresponding networking services. Contrarily to Gheorghe in [14], this multi-layer approach does not only diagnose faults in the data plane links, but also the control links, the controller and its inner components (ports, CPU, applications, VNFs, etc.).

Finer diagnosis granularity: Contrarily to the approaches from Steinder in [3], Bennacer in [4], Bahl in [5] and Hounkonnou in [6], which diagnose up to node level, we propose a self-modeling approach that builds the diagnosis model from a set of finer-grained templates, inspired by the diagnosis approach of Kandula et. al. in [18] for enterprise networks, based on templates. Nevertheless, our templates are extendable and describe the inner dependencies of a given network resource with respect to its inner components, which it has not been done yet in SDN and NFV.

Reduced diagnosis uncertainty: we tackle this problem with the service-aware approach we defined. This approach correlates the service view with its underlying network states in order to reduce the uncertainty in the root cause as it will be described in section IV.

On-the-fly self-modeling: Contrarily to the approaches of Bennacer in [4], Bahl in [5], Hounkonnou in [6], and Gheorghe in [14] that diagnose static network topologies, we propose a self-modeling approach to diagnose dynamic network topologies and deployed services. To our knowledge, this is the first time a self-modeling approach is applied to SDN and NFV. Our self-modelling approach discovers the dependencies in a deterministic manner and regenerates the model on-the-fly with changes, unlike the self-modeling approaches proposed by Bennacer in [4] and Bahl in [5], which may have false positives as a result of an inappropriate 'significance level' parameter when calculating the dependencies.

III. SELF-DIAGNOSIS FRAMEWORK FOR NETWORKING SERVICES

In this section, we propose an overall view of our proposal: a multi-layer self-diagnosis framework to diagnose faults in programmable networks. The self-diagnosis framework ensures a multi-layer diagnosis through a multi-layered model that includes the supervised resources within the following layers: 1) physical, 2) logical, virtual and 3) networking services. Examples of the supervised resources are given in Table 1.

Table 1. Types of resources considered per layer

Layer	*Resources*
physical	links, switches, hosts, controllers, ports, NICs, CPU
logical	Flows, controller application, OpenFlow application, VNFIs
virtual	virtual links, VNFs
service	VPN, NAT, firewall, streaming, etc.

The multi-layer self-diagnosis framework we proposed in Fig. 1 is part of the management and orchestration plane of NFV. It is important to notice that our diagnosis framework is independent from the type of SDN controller, southbound protocol agnostic and it performs diagnosis based on a global view and multi-layered view of the network.

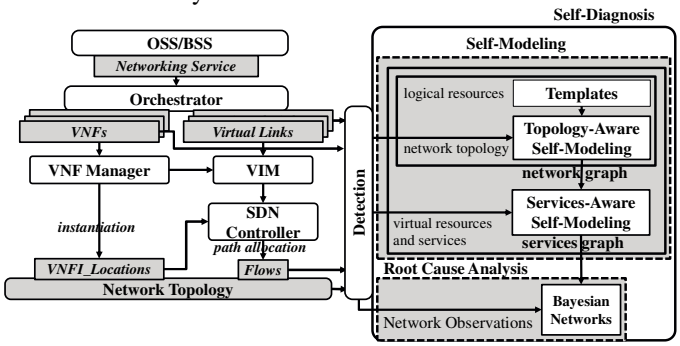

Fig. 1. Multi-layer Self-diagnosis framework for SDN and NFV

We propose three modules within our framework: a detection module, a self-modeling module, and a RCA module. Self-modeling and RCA modules include a methodology and associated algorithms as well as extensive validation. The detection module is a set of scripts to feed the other modules as the observability techniques are out of the scope of this paper. In Fig. 1 we sketch how those modules are related.

1) The *detection module* builds a view on the networking services and their underlying resources at instant t. It receives the following data:

-The network topology and the logical resources running on networked nodes (Openflow client applications running on switches and the instantiated VNF (VNFIs) running on hosts)

-The deployed networking services and their respective VNFs and Virtual links

-The flows sent by the SDN controller to establish the physical path to connect the VNFIs.

It also keeps the dependency graph updated, by ordering the self-modeling module to regenerate the dependency graph to prevent that the root cause had been calculated based on an outdated model.

2) The *self-modeling module* builds the multi-layered dependency graph. It relies on two algorithms:

Topology-aware self-modeling algorithm: it generates a first dependency graph from the network topology (physical nodes and links) and logical applications running on the network nodes, hereafter named *network dependency graph.*

Service-aware self-modeling algorithm: it generates a second dependency graph, hereafter named *services dependency graph,* by extending the *network dependency graph* with the dependencies of the networking services. The *services dependency graph* contains the dependencies between networking services and virtual resources, and the dependencies of virtual resources from logical and physical resources underneath.

3) The *RCA module* finds the root cause explaining the service failures by propagating the retrieved network observations through the *services dependency graph* given as input.

IV. THE SELF-MODELING MODULE

The self-modeling module is composed of a topology-aware algorithm, which generates the *network dependency graph*, and the service-aware algorithm which generates the *services dependency graph.* The *services dependency graph* includes the *network dependency graph*. We formalize the problem to model as it follows:

- The network topology is composed of P links and Q nodes.
- There are N networking services deployed over the network topology
- Each *networking service*$_i$ $\forall\, i = [1, N]$ is composed of M_i virtual links connecting N_i VNFs.
- In the VNF forwarding graph, VNFs connect to each other through virtual links via their CP (connection points)
- Each virtual link connects two CPs through a physical path between two hosts
- VNFIs are embedded in hosts and use their corresponding hosts' NICs as CP
- The SDN controller allocates each *physical path*$_j$ $\forall\, j = [1, M_i]$ by installing $n_i^{(j)}$ *flows* on $n_i^{(j)}$ intermediate switches that will connect both CPs and eventually the VNFIs.

Fig. 2 shows a *physical path*$_j$ composed of the SDN controller, several switches and links to connect two VNFs embedded in $host_1$ and $host_2$. All the switches of this path will receive flows from the SDN controller of the following type: {"in_port":x,"out_port":y,"src":"CP_2","dst":"CP_3"}.

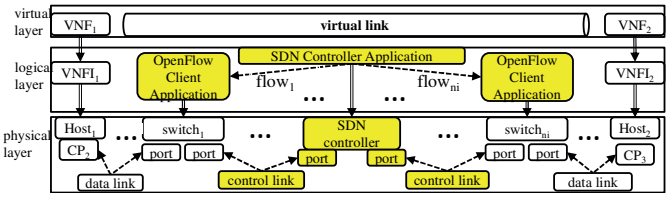

Fig. 2. Resources involved in a virtual link connecting two VNFs

Topology-aware self-modeling algorithm

This algorithm generates the *network dependency graph* from the current network topology and logical resources running on physical resources. It builds the *network dependency graph* by assembling the instantiated dependency graphs of network resources which are based on a set of finer-grained templates. This finer granularity allows diagnosing until inner component level. The dependency graphs of network resources (Fig. 3) describe their internal components and their dependencies among them. Network nodes are composed of physical and logical components, so their dependency graphs contain a physical layer with physical components–CPU, NICs, ports–and a logical layer with logical components–Openflow client applications or VNFIs–. VNFIs have three states: instantiated ($VNFI_I$), configured ($VNFI_C$), and active ($VNFI_A$), following the ETSI NFV GS specification [19].

-The K VNFIs embedded in hosts are given by the *VNFI_Locations* variable, which is used by the topology-aware self-modeling algorithm to update the dependency graph of hosts with their corresponding embedded VNFIs (Fig. 3).

-Switches run an OpenFlow client application to communicate with the SDN controller

-The SDN controller runs a SDN controller application such as OpenDaylight or Floodlight.

The number of VNFIs, ports and NICs of those dependency graphs are extendable with the VNFIs embedded in each host and the connections found in the topology.

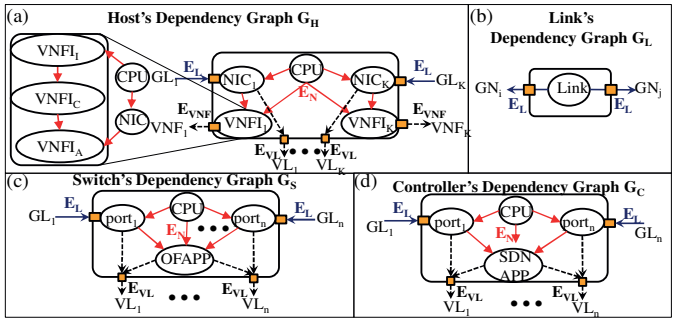

Fig. 3. Dependency graphs of network resources: (a) host, (b) link, (c) switch, and (d) SDN controller.

The *network dependency graph* (NDG) is generated through a three-step algorithm:

1) It classifies each network resource in nodes and links and it instantiates a different dependency graph $G_N := \{G_S, G_C, G_H\}$ for nodes (switches, controllers and hosts) and $G_L := \{G_{DL}, G_{CL}\}$ for links (data links (DL) and control links (CL)). The dependency graph of nodes and links (G_N and G_L) contain vertices V_N and V_L modelling their internal components. The edges E_N and E_L model the dependencies among those internal components in the network nodes (in red in Fig. 4).

2) It adds the instantiated dependency graphs G_N and G_L to the *network dependency graph*.

3) It connects those instantiated dependency graphs through E_L edges (in blue in Fig. 4). An E_L edge connects two dependency graphs of nodes (G_N): $E_L^{(k)} = (G_N^{(i)}, G_N^{(j)})$. A E_L edge represents the impact of faults in links on the interfaces (ports and NICs) inside nodes. The *network dependency graph* is built by assembling the instantiated dependency graphs G_N and G_L belonging to the P links and Q nodes of the network topology.

$$NDG = \bigcup_{k=1}^{P} G_L^{(k)}(V_L^{(k)}, E_L^{(k)}) \cup \bigcup_{k=1}^{Q} G_N^{(k)}(V_N^{(k)}, E_N^{(k)})$$

Fig. 4 shows a network dependency graph built from a simple topology connecting two hosts. NDG is composed of Q=3 G_N

and P=2 G_L dependency graphs. The network dependency graph includes also the SDN controller and the control links, not shown here due to space constraints.

Service-aware self-modeling algorithm

This algorithm takes as input the NFV records, which contain runtime information of the deployed instances of the VNFs, virtual links and networking services (further detailed in [19]). It contains also the VNF forwarding graph, which depicts how the traffic among VNFs is forwarded through the virtual links.

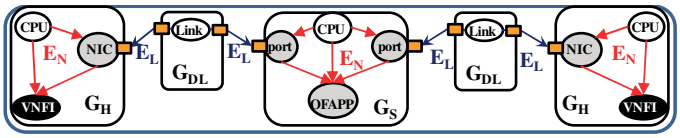

Fig. 4. Example of Network dependency graph (Q=3 nodes and P=2 links)

Virtual resources dependency graph generation algorithm

Input: NSR(Network Service record)
Output: VRG (Virtual Resources Dependency Graph)
nsr←NSR[i] ∀i={1,…,N} //retrieval of NSR of that network service
V(VRG)← V(VRG) ∪ nsr:id
VL←nsr:vlr[j]* ∀j={1,…, M_i} //retrieval of virtual links
V(VRG)← V(VRG) ∪ VL //adds virtual link vertex to virtual layer
E(VRG)← E (VRG) ∪ E_S:=(orig:[VL:id*],dest:[VL:parent_ns*]) // adds E_S
VNFR←nsr:vnfr[k]* ∀k={1,…,N_i} //retrieval of VNFs
V(VRG)← V(VRG) ∪ VNF //adds VNF vertex to virtual layer
E(VRG)← E (VRG) ∪ E_S:=(orig:[VNF:id*],dest:[VNF:parent_ns*])//adds E_S

* the access to nsr parameters is given in the NFV record defined by ETSI NFV [19]

The service-aware algorithm generates the *services dependency graph* in two steps.

Step 1: This algorithm creates an auxiliary graph, called *virtual resources dependency graph (VRG)*, containing the discovered networking services and their virtual resources.

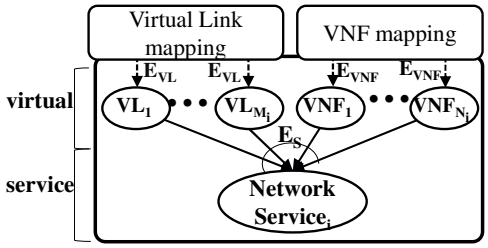

Fig. 5. Virtual Resources Dependency Graph of the i-th networking service

The *VRG* is composed of a virtual layer and a service layer (Fig. 5). For each discovered networking service, both layers are filled as follows:

Service layer: the algorithm adds a *networking service$_i$* vertex to the *VRG*.

Virtual layer: the algorithm adds M_i virtual links vertices and N_i VNFs vertices. It then adds M_i+N_i E_S edges (in black in Fig. 5) from those virtual resources vertices to the *networking service$_i$* vertex. E_S edges represent the impact of faults in virtual resources on that networking service. If the SDN controller is enabled with the SFC module such as in OpenDaylight, the VRG could be directly generated from the VNF FG information.

Step 2: It connects the *network dependency graph* to the *VRG* and builds the *service dependency graph.* For each networking service, two mappings are done:

-**VNF mapping:** The VNFI vertices of the hosts in the *network dependency graph* are connected to the VNFs vertices in the *VRG* through edges E_{VNF} (in dash black in Fig. 6). E_{VNF} edges represent the impact of faults in the VNFI embedded in hosts on VNFs composing a networking service.

-**Virtual Links mapping:** The physical network resources involved in each virtual link (hosts NICs, switches ports, and OpenFlow client applications inside switches) are connected to their respective virtual links vertices through edges E_{VL} (in dash black in Fig. 6). E_{VL} edges represent the impact of faults in physical and logical resources on a virtual link. These network resources are extracted from the *flows*, defined in section IV.

Fig. 6 shows an example of *services dependency graph* sent to the RCA. This *services dependency graph* belongs to one networking service (N=1) composed of one virtual link (M_i=1) connecting two VNFs (N_i =2) deployed over a physical path. The *services graph* includes the *network graph* shown in Fig. 5.

Service dependency graph generation algorithm

Input: NDG, VRG, Flows, VNFI_Locations, NSR(Network Service Record)
Output: SDG (Service Dependency Graph)
SDG←NDG ∪ VRG //initialization
nsr←NSR[i] ∀i={1,…,N} //retrieval of networking services
VL←nsr:vlr[j]* ∀j={1,…, M_i}//retrieval of virtual links
VNF←nsr:vnfr[k]* ∀k={1,…,N_i} //retrieval of VNFs
flowsPerVL←Flows[VL] // retrieval of flows composing virtual links
flow←flowsPerVL[l] ∀l={1,…, $n_i^{(j)}$}
[switch,ports,OFAPP] ←ExtractSwitchInfo(flow) //extracts switch storing flow
[hosts,NICs] ←ExtractHostsInfo(flow) //extracts hosts connected by that flow
VNFID←VNFI_Locations[hosts] //finds VNFID of VNFI embedded in host
E (SDG)← E(SDG) ∪ E_{VNF}:=(orig:[VNFID],dest:[VNF:id*]) //adds edge
E(SDG)← E(SDG) ∪ E_{VL}:= (orig:[hosts:NIC],dest:[VL:id*]) //adds edge
E(SDG)← E(SDG) ∪ E_{VL}:= (orig:[switch:ports],dest:[VL:id*]) //adds edge
E(SDG)← E(SDG) ∪ E_{VL}:= (orig:[switch:OFAPP],dest:[VL:id*]) //adds edge

* the access to nsr parameters is given in the NFV record defined by ETSI NFV [19]

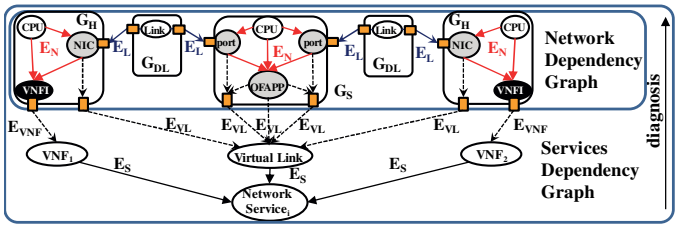

Fig. 6. Services dependency graph of one networking service

V. THE ROOT CAUSE ANALYSIS MODULE

The RCA is based on BN. A BN is a probabilistic dependency graph BN(*V, E, π)* where Vertices *V* are the variables modelled, in this case are physical, logical, and virtual network components characterized by binary random variables which indicate their state (‘*down*’ or ‘*up*’) and the edges represent the dependencies among network components. *π* is the set of CPT (Conditional Probability Tables) to describe the conditional probability distribution of each random variable modelled in the BN. We assume that all network components

can always fail by themselves, with a *probability of fault p*, despite their parents work as expected. Also, one fault in one network component immediately propagates to those network components depending on it. The CPT that describes this behavior is given in Table 3, and it is justified by the works led by Hounkonnou et. al. for IMS [6] and our self-diagnosis framework for SDN [2]. The RCA reasons over the *services dependency graph.* In Fig. 6, the RCA starts propagating evidence on the network service vertex through the graph based on the CPTs until it reaches the root vertices, yielding a posteriori probability distribution *P*. The *services dependency graph* modelling allows the RCA to diagnose dynamic networking services on dynamic network topologies.

Table 2. CPT of a network resource (Y)

CPT(Y)	*Pr(Y='down')*	*Pr (Y='up')*
at least one parent 'down'	1	0
if all parents of Y 'up'	*p*	1- *p*

P is the probability of root cause for each network component that explains the network observations given as input. The uncertainty of this distribution is quantified via the entropy (in bits) with the following equation:

$$H(X) = -\sum P \log_2 P$$

Entropy depends on the number and quality of the network observations added. The lower entropy, the lower uncertainty and the better the BN engine discriminates among different root causes. However, in our previous work, the finer granularity of our templates and the consideration of all the network topology in the diagnosis resulted in a high number of vertices in the graph, what led to high uncertainty in the root-cause. Nevertheless, if additional observations *Y* are added in the graph, entropy *H(X)* is reduced by $G(Y) = H(X) - H(X|Y)$. We propose two RCA strategies to reduce this uncertainty, which effectiveness is proved in section VI.

Extension of the *services dependency graph*: we define an RCA strategy that extends the *services dependency graph* to include the dependencies of the healthy networking services that are sharing resources with the affected service. This RCA strategy allows discarding those network resources involved in healthy services.

Reduction of the *network dependency graph*: we define an RCA strategy that reduces the *network dependency graph* to only consider the dependencies of network resources that are involved in the identified faulty networking services, thereby reducing the uncertainty and the diagnosis time.

VI. PROOF OF CONCEPT

In this section, we diagnose several networking services delivered with two different network topologies (Fig. 7 and 8) to which we apply our RCA strategies. Each networking service is composed of two VNFs, whose instances $VNFI_{Ai}$ and $VNFI_{Bi}$, are embedded in different hosts. Both VNFIs are connected through a virtual link $VL_{Ai,Bi}$, which is established at run-time by the SDN controller.

First, the self-modeling module generates on-the-fly the *services dependency graph* (that includes the *network dependency graph).* The *services dependency graph* is generated in the following situations:

Changes on the network topology: The *network dependency graph* is generated for a tree topology (D=2, F=3), in Fig. 7, and a linear topology with $L \in [5,10]$, in Fig. 8. Also, the self-modeling module models changing topologies by discovering new network resources and regenerating the *network dependency graph*, as shown in this video [20]. The *network graph* includes the connections from the SDN controller to the switches, not shown here.

Table 3. Affected networking services and underlying physical paths

Topology	*Service*	*Virtual Link*	*Host:VNFI*	*Physical path*
Tree D=2, F=3 (Fig.7)	NS_1	$VL_{A1,B1}$	**H_1:**$VNFI_{A1}$ H_9:$VNFI_{B1}$	DP:[**AL_1,S_2,IL_1,S_1, IL_3,S_4,**AL_9], CP:[**C_0,CL_1,CL_2,CL_3, CL_4,C_1**]
Linear L=5 (Fig. 8)	NS_4	$VL_{A4,B4}$	**H_4:**$VNFI_{A4}$ H_5:$VNFI_{B4}$	DP:[**AL_4,S_4,**IL_4, S_5,AL_5], CP:[**C_0,CL_4,**CL_5,**C_1**]

*S: switch, IL: inter switch link, CL: control link, AL: access link, C: controller, H:host
DP: Data plane (hosts,switches,datalinks), CP: Control plane (controllers,control links)

VNF migrations: In Fig. 7, the host H_1 embeds four VNFIs while the rest of hosts embed one VNFI, but in Fig. 8, VNFI are differently distributed and the self-modeling algorithm generates *the services dependency graph* taking into account both distributions of VNFIs in both cases. If VNFIs migrate, the self-modeling regenerates the *services dependency graph* with the new distribution of VNFIs.

Changes on the virtual links: VNF migrations and topological changes lead to changes on the virtual links connecting them. Table 4 presents the underlying physical resources involved in the networking services that will be diagnosed hereafter. In both topologies, networking services share some physical network resources such as physical links, switches and part of the control plane. The shared network resources are depicted in bold. This information will be exploited by the RCA to reduce the uncertainty.

In the next part we show how the RCA can adapt and exploit the *services dependency graph* and the *network dependency graph* to efficiently diagnose networking services failures in two different cases. The diagnosis is automated by the on-the-fly generation of both dependency graphs. The RCA calculates the root cause. i.e. the RCA identifies physical, logical and virtual network resources presumed to be the root cause of a given networking service failure. We consider that all the network resources and their internal components have the same probability of fault (p=0.1) in the conducted experiments. Hereafter, we evaluate the two RCA uncertainty reduction strategies:

Case 1: Extension of the *services dependency graph*

We consider the tree topology (Fig. 7), where the services are deployed sequentially i.e. at $t_i=t_0 + (i-1)T$, i=1…N. A failure is injected in service NS_1 and the self-modeling algorithm is launched at $t_1 = t_0$, generating the *services dependency graph*

from the affected service NS_1 and the RCA gives a posteriori distribution probability (Fig. 7 dark blue bar on the bottom) so spread over all the network resources that no root cause can be clearly identified (entropy: 4.1 bits). The uncertainty can be reduced by adding the healthy services sharing resources with the affected service. Indeed, if the graph is regenerated at $t_2=t_0+T$, when a healthy service NS_2 is deployed (N=2), the root cause becomes less uncertain (entropy: 3.6 bits). Adding the new healthy service NS_2 allows the RCA to discard those shared resources between the affected service NS_1 and NS_2 (S_2, AL_1 and the control plane resources). The RCA module extends the *services dependency graph* to reduce the entropy four times more by including the healthy services as those appear: NS_3 at $t_3=t_0+2T$, NS_4 at $t_4=t_0+3T$, NS_5 at $t_5=t_0+4T$, and NS_6 at $t_6=t_0+5T$. Fig. 7 shows how the entropy is reduced from 4.1 (dark blue bar on the bottom), with the only affected service added, to 0.9 bits (brown bar on top), with the affected service and 5 healthy services added.

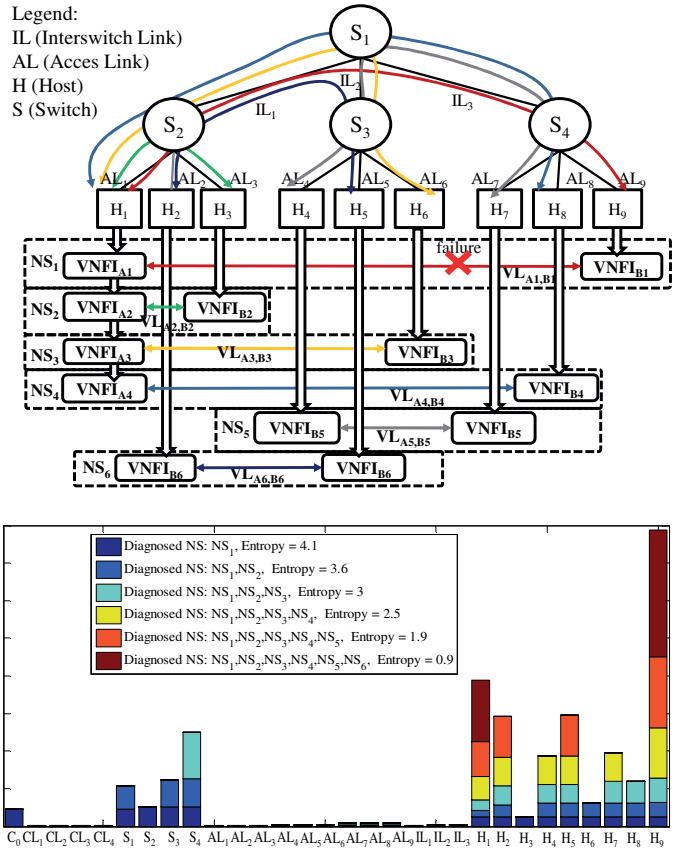

Fig. 7. Case 1: RCA strategy on extending the services dependency graph

In the brown bar probability distribution (Fig. 7 on the bottom), the root cause list consists of the hosts H_1 (33%) and H_9 (67%). The rest of hosts are discarded, as those are not involved in the affected service. Our finer granular templates enable a deeper analysis. Not only links and switches are shared among services, but also the CPU and NIC inside hosts. Host H_1 embeds four VNFIs, as a result, those VNFIs share NIC and CPU. Nevertheless, NIC and CPU are immediately discarded when at least one of these VNFIs is involved in a healthy service as it means that NIC and CPU is working fine. Indeed, we see that the most probable explanations (Table 4, Case 1) are that $VNFI_{A1}$ and $VNFI_{B1}$ are not initiated, configured, or activated (adding up all VNFI states: $VNFI_{A1}$ (33%) and $VNFI_{B1}$ (51%)), which is coherent with the injected failure in NS_1, composed of those VNFs. Also, the RCA can discard those VNFIs embedded in H_1 ($VNFI_{A2}$, $VNFI_{A3}$, and $VNFI_{A4}$), because they are not involved in the affected service NS_1.

Case 2: Reduction of the *network dependency graph*

We first inject a failure in service NS_4 and generate the *network dependency graph* from the network topology of the blue region in Fig. 8 and we incrementally reduce the diagnosis region until the optimal one (brown region in Fig. 8) that gives the lowest uncertainty.

Blue region: the *network dependency graph* is built from a linear topology L=10 and it includes the following services to build the *services dependency graph*:

- (i) the affected service NS_1: the a posteriori distribution probability has as entropy 4.7 bits.
- (ii) the affected service and the healthy services NS_2, NS_3, and NS_4: the a posteriori distribution probability (Fig. 8, dark blue bar on the bottom) has lower entropy (3.9 bits), because the added networking services help discard those network resources involved in them.

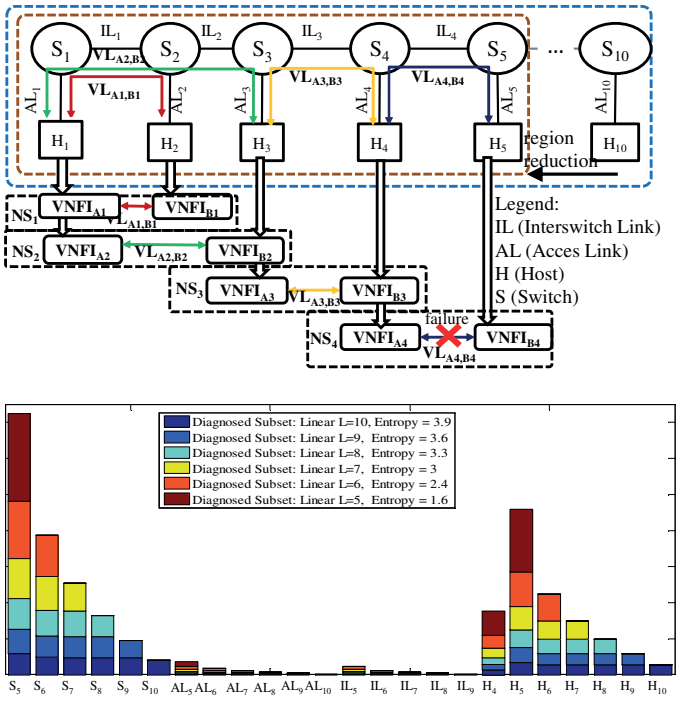

Fig. 8. Case 2: RCA strategy on reducing the network dependency graph

Table 4. Zoom on the root cause inside host in cases 1 and 2

Host	*CPU*	*NIC*		*VNFI Not Instantiated*	*VNFI Not Configured*	*VNFI Not Active*
Case 1						
H_1	0	0	$VNFI_{A1}$	**6**	**11**	**16**
			$VNFI_{A2}$	0	0	0
			$VNFI_{A3}$	0	0	0

			$VNFI_{A4}$	0	0	0
H_9	**6**	**11**	$VNFI_{B1}$	**11**	**16**	**24**
Case 2						
H_4	0	0	$VNFI_{B3}$	0	0	0
			$VNFI_{A4}$	**3**	**5**	**7**
H_5	**3**	**7**	$VNFI_{B4}$	**5**	**7**	**13**

In both situations (i) and (ii), the a posteriori distribution is so spread over the existing network resources that no root cause can be identified.

Brown region: the *network dependency graph* is built from a linear topology L=5 and it includes the following services to build the *services dependency graph*:

- (i) the affected service NS_1: the a posterior distribution probability has as entropy 2.2 bits.
- (ii) the affected service and the healthy services NS_2, NS_3, and NS_4: the a posteriori distribution probability (brown bar on top) has lower entropy (1.6 bits) because the added services help discard those network resources involved in them.

Fig. 8 shows that the uncertainty on the root cause is reduced when the diagnosis region gets closer to the brown diagnosis region: In situation (i) there is a reduction from 4.7 to 2.2 bits with one service added. In situation (ii) there is a reduction from 3.9 to 1.6 bits with 4 services added (Fig. 8 on the bottom). We focus on the situation (ii), where a clear subset of the network–S_5 (48%), AL_5 (3%), H_4 (15%), and H_5 (35%)–is presumed to be the root cause. This result is coherent with the injected failure in NS_4 as its underlying virtual resources, the VNFI embedded in hosts H_4 ($VNFI_{A4}$) and H_5 (VNF_{B4}), are pinpointed as possible root causes. Analogously as in previous section, we can zoom on hosts H_4 and H_5 (Table 4, Case 2) to obtain the probability of fault in the VNFIs running inside those hosts. The most probable explanation is that those VNFIs embedded on H_4 and H_5 are not initiated, configured, or active (adding up all VNF states: $VNFI_{A4}$ (17%), and $VNFI_{B4}$ (25%)). Contrarily, the hosts embedding VNFIs which are not involved in the affected service (i.e. H_1, H_2, and H_3) are discarded. Furthermore, other VNFIs (e.g. $VNFI_{B3}$) embedded in the hosts presumed to be the root cause (H_4) but not involved in the affected service are discarded. In all regions, those network resources not involved in the affected service NS_4 are discarded (e.g. S_1, S_2, S_3, H_1, H_2, H_3 among others).

We evaluate the performance of both RCA strategies that reduce the uncertainty on the diagnosis of networking services, measured in terms of generated vertices and edges in the dependency graph and diagnosis time.

Case 1, extension of the *services dependency graph*: The RCA strategy that extends the *services dependency graph* in Fig. 7, adds a lower number of vertices per service added compared to the number of edges added, as seen in Table 5. This difference is due to the high number of dependencies (E_{VL} edges) of each virtual link from the physical resources (NICs, switches ports, and OpenFlow client applications inside switches). For instance, Fig. 6 shows 7 edges (5 E_{VL} and 2 E_{VNF}) and 4 vertices added. These added edges and vertices increase the diagnosis time $\mathbf{t_D = t_{SM} + t_{RCA}}$, where $\mathbf{t_{SM}}$ is the self-modeling time and $\mathbf{t_{RCA}}$ is the RCA time, both averaged 20 times. $\mathbf{t_{SM}}$ represents at least 51% of the diagnosis time $\mathbf{t_D}$. When six services are added to the graph, $\mathbf{t_{SM}}$ is increased a 57% of $\mathbf{t_D}$ with respect to one service added, whilst the $\mathbf{t_{RCA}}$ is increased by 72% of $\mathbf{t_D}$, proving that the BN engine inside the RCA scales worse than the self-modeling algorithm in itself.

Table 5. Cost of extending the services dependency graph

Services added	*#Vertices*	*#Edges*	t_{RCA}	t_{SM}
NS_1	108	306	1.1	1.4
NS_1, NS_2	115	334	1.2	1.6
NS_1, NS_2, NS_3	122	372	1.6	1.7
NS_1, NS_2, NS_3, NS_4	129	410	1.7	2
NS_1, NS_2, NS_3, NS_4, NS_5	133	440	1.7	2.1
NS_1, NS_2, NS_3, NS_4, NS_5, NS_6	137	470	1.9	2.2

Case 2, reduction of the *network dependency graph*: The RCA strategy that reduces the diagnosis region reduces also the diagnosis time, as the diagnosed network topology is smaller. As example of this reduction, we compare the size of the *services dependency graph* when it is generated from the blue region in Fig. 8 (linear topology L=10) to the *services dependency graph* generated from the brown region in Fig. 8 (linear topology L=5) resulting from reducing the diagnosis region. The graph includes the 4 networking services (NS_1 … NS_4). The number of vertices is reduced from 196 to 111 vertices while the number of edges is reduced from 592 to 350 edges, and the diagnosis time is almost divided in half, transitioning from 4 to 2.1 seconds (averaged 20 times).

VII. CONCLUSIONS AND FUTURE WORK

This paper specifies, implements, and evaluates a multi-layer self-diagnosis framework capable of diagnosing faults in programmable networks with SDN and NFV, while taking into account the networking service, virtual, logical, and physical layers. In this regard, this paper extends our previous work, a topology-aware self-diagnosis approach, by diagnosing and correlating two additional layer, virtual and services layer, while considering their dynamic dependencies with the underlying logical and physical resources. The core of the self-diagnosis framework is a self-modeling module that relies on two algorithms to generate on-the-fly and update the diagnosis model from the network topology, logical resources and networking services with a set of adaptable templates. Service-aware diagnosis reduces uncertainty by automatically extending or reducing the dependency graph according to the faulty networking service. In addition, the finer granularity of the proposed templates details the states of the components inside the networked nodes and details the state of the VNFIs embedded in the hosts. This framework could operate in preventive and reactive modes, i.e. respectively triggered by notifications indicating thresholds crossings or by alarms indicating failures or faults. As future work, we will build a detection module to evaluate the impact of degradations in network resources such as CPU load and throughput on the VNFs and networking services to predict future failures.

ACKNOWLEDGMENTS

This work is partly funded by the French ANR under the ANR-14-CE28-0019 REFLEXION project

REFERENCES

[1] ETSI NFV Group Specification Draft: "Network Functions Virtualisation (NFV); Ecosystem; Report on SDN Usage in NFV Architectural Framework", Sept. 2015.

[2] J. Sanchez, I. Grida Ben Yahia, N. Crespi, "Self-Modeling Based Diagnosis of Software-Defined Networks," Workshop MISSION 2015 at 1st IEEE Conference on Network Softwarization, London, 13-17 April 2015.

[3] M. Steinder and A. S. Sethi. "End-to-end Service Failure Diagnosis Using Belief Networks". In Network Operations and Management Symposium, NOMS 2002, pages 375-390, 2002

[4] L. Bennacer, L. Ciavaglia, et.al., "Optimization of fault diagnosis based on the combination of Bayesian Networks and Case-Based Reasoning," in NOMS, 2012 IEEE , vol., no., pp.619,622, 16-20 April 2012.

[5] P. Bahl, R. Chandra, et. al., "Towards highly reliable enterprise networking services via inference of multi-level dependencies," in SIGCOMM, 2007.

[6] C. Hounkonnou, "Active Self-Diagnosis in Telecommunication Networks". PhD thesis. Université de Rennes 1. July 2013.

[7] D. Kreutz, F.M.V. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," Proceedings of the IEEE , vol.103, no.1, pp.14,76, Jan. 2015.

[8] P. Fonseca, R. Bennesby, E. Mota and A. Passito, "A replication component for resilient OpenFlow-based networking," Network Operations and Management Symposium (NOMS), 2012 IEEE , vol., no., pp.933,939, 16-20 April 2012

[9] M. Canini, D. Venzano, et. al., "A NICE way to test OpenFlow applications," in Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, ser. NSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 10–10.

[10] N. Handigol, B. Heller, V. Jeyakumar, D. Mazieres, and N. McKeown,"Where is the debugger for my software-defined network?" in Proceedings of the First Workshop on Hot Topics in Software Defined Networks, ser. HotSDN '12. New York, NY, USA: ACM, 2012, pp. 55–60.

[11] A. Wundsam, D. Levin, S. Seetharaman, and A. Feldmann, "OFRewind: Enabling Record and Replay Troubleshooting for Networks," in Proc. 2011 USENIX Conference on USENIX Annual Technical Conference, ser. USENIXATC'11. USENIX Association, 2011, pp. 29–29.

[12] N. Handigol, B. Heller, V. Jeyakumar, D. Mazieres, and N. McKeown,"I know what your packet did last hop: Using packet histories to troubleshoot networks," in 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14). Seattle, WA: USENIX Association, Apr. 2014, pp. 71–85.

[13] R.C. Turchetti, E. P. Duarte, "Implementation of Failure Detector Based on Network Function Virtualization," in Dependable Systems and Networks Workshops (DSN-W), 2015 IEEE International Conference on , vol., no., pp.19-25, 22-25 June 2015.

[14] G. Georghe; T. Avanesov, M.-R. Palattella, T. Engel, Popoviciu, C., "SDN-RADAR: Network troubleshooting combining user experience and SDN capabilities," in Network Softwarization (NetSoft), 2015 1st IEEE Conference on, vol., no., pp.1-5, 13-17 April 2015.

[15] R. Mijumbi, et.al., "Network Function Virtualization: State-of-the-art and Research Challenges," in *Communications Surveys & Tutorials, IEEE* , vol.PP, no.99, pp.1-1.

[16] R.P. Esteves, L.Z. Granville, R. Boutaba, "On the management of virtual networks," in Communications Magazine, IEEE , vol.51, no.7, pp.80,88, July 2013.

[17] N.M.M.K. Chowdhury, R. Boutaba, "Network virtualization: state of the art and research challenges," in *Communications Magazine, IEEE*, vol.47, no.7, pp.20-26, July 2009.

[18] S. Kandula, R. Mahajan, et. al, "Detailed diagnosis in enterprise networks," in SIGCOMM, 2010.

[19] ETSI NFV Group Specification: "Network Functions Virtualisation (NFV); Management And Orchestration", Dec. 2014

[20] Topology-Aware Self-Diagnosis framework, presented in Orange Labs Research exhibition 2015, Paris, France. Video available at: https://www.youtube.com/watch?v=xNudu48quRM

[21] M. Scholler et. al., "Resilient deployment of virtual network functions," in UltraModern Telecommunications and Control Systems and Workshops (ICUMT), 2013 5th International Congress on , vol., no., pp.208-214, 10-13 Sept. 2013

[22] M. Miyazawa et.al., "vNMF: Distributed fault detection using clustering approach for network function virtualization," in Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on , vol., no., pp.640-645, 11-15 May 2015