Computer Networks 178 (2020) 107337

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Network service topology: Formalization, taxonomy and the CUSTOM specification model

Vinicius Fulber-Garcia[a,*], Elias P. Duarte Jr.[a], Alexandre Huff[b], Carlos R.P. dos Santos[c]

[a] Department of Informatics, Federal University of Paraná, Curitiba, Brazil
[b] Federal Technological University of Paraná, Toledo, Brazil
[c] Department of Applied Computing, Federal University of Santa Maria, Santa Maria, Brazil

ARTICLE INFO

Keywords:
NFV
Service
Topology
Definitions
Taxonomy
Specification
Model

ABSTRACT

Network Function Virtualization (NFV) relies on virtualization technologies to allow the implementation of middleboxes in software that is executed on commercial off-the-shelf hardware. Multiple Virtual Network Functions (VNFs) can be combined to form arbitrary network services. The term *service topology* has been freely employed by both major NFV recommendation and standardization bodies (the Internet Engineering Task Force (IETF) and European Telecommunications Standards Institute (ETSI)) and in the literature. The objective of this work is to present a formal specification that unifies these different views of service topologies. A taxonomy is proposed which allows the classification of topologies according to multiple criteria, including structure, size, heterogeneity, function sharing, among others. We also propose the CUstom Service TOpology Model (CUSTOM), a specification model that allows the design of network service topologies that feature the different categories proposed in the taxonomy. Finally, we demonstrate the specification capabilities of CUSTOM through a series of case studies.

1. Introduction

The Internet infrastructure relies on *middleboxes* for running network protocols (e.g., Internet Protocol (IP), Multi Protocol Label Switching (MPLS), Border Gateway Protocol (BGP)) and services (e.g., packet filtering, naming). Although middleboxes have been traditionally implemented in hardware, virtualization technologies have opened up the possibility for implementing those services in software. The advantages are manyfold, not only in terms of the flexibility for operating and managing the network, since it is so much easier to create, use and manage virtual services, but also in terms of cost which is substantially lower. One might even argue that virtualization has solved – or at least has the potential to solve – the problem known as "ossification of the Internet" [1].

Network Function Virtualization (NFV) [2] relies on virtualization technologies to implement middleboxes as Virtual Network Functions (VNF) which are executed on Virtual Machines (VM) or containers that run on commercial off-the-shelf hardware platforms. The use of NFV improves the flexibility of services traditionally implemented as hardware-based middleboxes [3] and promotes the reduction of both CAPital EXpenditure (CAPEX) and OPerational EXpenditure (OPEX) [4]. Network services and their enablers can also be provided by modern NFV business environments (e.g., FENDE [5]) in the context of the Network-as-a-Service model, thus creating a value chain and financial flows that fit in the context of the Internet of Services (IoS) paradigm [6,7]. Note that, Multiple types of NFV enablers are already available, including VNF platforms [8–12] which have been widely adopted to support both implementation and management of virtualized services.

Most important, NFV allows the creation of services by composing multiple Network Functions (NF) on Service Function Chains (SFC) [13] or Network Services (NS) [14]. These services are created by specifying service topologies and service descriptors. Services topologies, in turn, specify through which network functions, end points (ingress and egress nodes) and internal connections the network traffic is steered and processed. Service descriptors are also used to specify each component of a service topology (e.g., Virtual Deployment Units of a VNF), in addition to other service features. The management and maintenance of a service involve processes which can be considered complex, not only those related to the composition of the service topology itself, but also those related to lifecycle management tasks, including scaling, migrating, and placement of individual functions and chains. Currently, it is not trivial to ensure the successful execution of those service management tasks. There are actually multiple approaches for running the myriad of tasks for managing the lifecycle of network services, some of which are widely adopted and several of which are not compatible with each other.

* Corresponding author.
E-mail addresses: vfgarcia@inf.ufpr.br (V. Fulber-Garcia), elias@inf.ufpr.br (E.P. Duarte Jr.), alexandrehuff@utfpr.edu.br (A. Huff), csantos@inf.ufsm.br (C.R.P. dos Santos).

https://doi.org/10.1016/j.comnet.2020.107337
Received 23 September 2019; Received in revised form 4 May 2020; Accepted 24 May 2020
Available online 29 May 2020
1389-1286/

The problem starts with the very specification of service topologies. There are currently multiple and different ways to define a topology, such as using formal description languages [15], context-free grammars [16], by directly specifying a traffic forwarding graph [17], or by using structured markup languages, such as Yet Another Next Generation (YANG) [18]. Each strategy has features that make it easier or more difficult to be used in particular scenarios (e.g., to run specific service management tasks). But none provides a comprehensive collection of features to solve the major problems related to the service topology specification.

It is undeniable that although plenty of service topology characteristics have been recently addressed in the literature (e.g., dependencies, sharing, and heterogeneity), most if not all require complex service configurations or very low-level descriptors. We claim in the present work that several of these characteristics can be natively coded in the service topology specification itself.

In this work, we give a formal definition of a network service topology as a graph, and propose a taxonomy that allows the classification of topologies according to multiple criteria, which are also formally specified. These categories include: structure (linear and branched); length and size; function sharing; function dependency; and heterogeneity (physical/virtual). We also introduce the CUstom Service TOpology Model (CUSTOM), a service topology specification model based on a context-free grammar that is fully compliant with the taxonomy. Finally, we demonstrate the specification capabilities of CUSTOM through a series of case studies.

The rest of the paper is organized as follows. Section 2 presents the background and preliminary definitions. Section 3 gives the definition of a service topology as a graph and presents the proposed taxonomy. The CUSTOM service topology specification model is described in Section 4. In Section 5 several case studies are presented and discussed. Section 6 presents relevant related work. Finally, Section 7 concludes the paper.

2. NFV: definitions & standards

This section presents definitions and an overview of standardization efforts of both Network Function Virtualization and Service Function Chaining.

2.1. Network function virtualization

Network Function Virtualization (NFV) decouples network functions from hardware by using virtualization technologies [2]. Despite the fact that there are advantages of using dedicated hardware (e.g., performance), it is undeniable that those solutions imply higher CAPEX and OPEX. Furthermore, the challenges keep increasing for the implementation of new sophisticated network services in hardware [4]. In this context, NFV technologies represent an effective solution, in terms of flexibility and reduced costs.

In order to define standards to foster interoperable NFV adoption, the European Telecommunication Standards Institute (ETSI) has specified a comprehensive architecture, which is organized in three functional blocks. The NFV Infrastructure (NFVI) comprises the physical resources (i.e., computing, storage, and network), and how they are virtualized to support the execution of network functions; the Virtualized Network Functions (VNF) block represents the implementation of network functions running on the NFVI; and the NFV Management and Orchestration (NFV-MANO) block is in charge of the overall activities regarding the virtualized network functions.

NFV-MANO is composed of three elements: the Orchestrator, responsible for the management of complex network services; the VNF Manager, which provides VNF lifecycle operations (e.g., instantiation, scaling, update, and termination); and the Virtualized Infrastructure Manager - VIM, which controls the physical/software infrastructure that supports the virtualized resources (e.g., computing, storage, and network). In addition, each VNF has an associated Element Management System (EMS), which enables the communication between the NFV-MANO and the VNF itself providing FCAPS (*Fault, Configuration, Accounting, Performance, and Security*) management functionalities.

Besides the definition of the architecture, the ETSI has also identified a set of key aspects that must be taken into account by NFV platforms (i.e., portability, performance, integration, scalability, management, and orchestration) [19]. These aspects are essential to support the widespread adoption of the NFV paradigm, thus enabling the creation of network services that can be widely adopted. In this context, it is important to simplify the management of virtualized infrastructures. Particularly important is the definition of high-level models and practices to create dynamic chains of VNFs, which are presented in the next sections.

2.2. Virtualized network services

Complex network services consist of multiple network functions working together according to a specification. Network functions can be organized in arbitrary ways to offer different network services. The instantiation of such services involves deploying VNFs on demand and chaining them together. Both the Internet Engineering Task Force (IETF) and the ETSI have led efforts to provide standards for such network services in the context of the NFV. However, there is still no consensus regarding not only terms but also some of the core functionalities, as we describe below.

The ETSI defines the concept of Network Service (NS) [14,20], which consists of chained network functions providing a service and service information (e.g., performance requirements). An NS is defined by its functionalities and requirements specification (e.g., Service Level Agreement (SLA) and network policies) in addition to the corresponding VNF Forwarding Graph (VNF-FG) [21]. The VNF-FG is a graph formed by vertices representing the network functions (at least with one VNF), and edges representing the connections between pairs of VNF-FG elements (i.e., VNFs, middleboxes, and endpoints). An NS can be defined by using high-level network service descriptors to represent the VNF-FG as well as other related information.

The IETF has also made efforts for defining NFV standards in the context of the Internet by its working groups (e.g., NFV Research Group (NFVRG) and SFC). The IETF has also specified a Service Function Chaining (SFC) architecture in order to standardize the composition of complex services. An SFC can be defined as a set of service functions connected in an ordered way and through which the traffic is steered [22]. Essentially, an SFC is created by first defining each single service function and the corresponding endpoints, the links connecting those endpoints and service functions, and finally deploying the SFC on the virtualized infrastructure (i.e., by allocating the required resources and setting up the environment as necessary).

The architecture proposed by the IETF allows the specification, creation, and maintenance of SFCs [22]. This architecture has management components (i.e., the Classifier and the Service Function Forwarder (SFF)) and operational components (i.e., the SFC Proxy and the Service Function). The Service Function (SF) is the element that processes network traffic and is the basic building block of SFCs, (i.e., it corresponds to the network function which can be virtualized or not). The Service Function Chaining Proxy (SFC-P) is employed to support legacy network functions by adding/removing the SFC encapsulation to/from the original network traffic on behalf of a given SF. The classifier element is placed at the start of a service chaining path (i.e., the ingress node) and is responsible for traffic classification as well as encapsulating selected frames/packets to indicate the SFC Path (i.e., the SF sequence of the SFC) through which the traffic must be forwarded. The SFF is employed to check the SFC encapsulation and forwarding the network traffic to the specific SFs.

Although the ETSI and the IETF specifications overlap at several points, several of the terms each defines/employs are not the same. For

Table 1
Comparison Between ETSI NS and IETF SFC.

		IETF			
		Boundary node	Service function	Service function path	Service function chain
ETSI	End Point	●			
	Virtualized / Physical Network Function		●		
	Path			●	
	End-to-End Service				●
	VNF Forwarding Graph				●
	Network Service				◑

example, ETSI defines endpoints [20] which correspond to the IETF boundary nodes [22], both terms refer to the bounds of End-to-End Services (ETSI) or for SFCs (IETF) and define the ingress and egress data nodes. The ETSI Virtualized and Physical Network Functions (VNF and PNF) [21] process network traffic as IETF Service Functions (SF) do [22]. The ETSI Paths [21] are similar to the IETF Service Function Paths [22] indicating the service paths of a VNF-FG (ETSI) or SFC (IETF). An SFC can be considered similar to the ETSI's VNF-FG [23], as they specify network elements, such as VNFs or PNFs and the connections between them. However, an SFC also considers boundary nodes which are only considered by ETSI End-to-End Services (consisting of both a VNF-FG and endpoints [20]). Furthermore, the SFC specification defines aspects related to dependencies, sharing, and policies as described in Quinn and Nadeau [13]. These aspects are used in the context of an ETSI NS [21], but an NS also includes operational information and even scripts (e.g., lifecycle scripts and deployment flavors) that are never mentioned in the IETF SFC specification. Table 1 summarizes this comparison between these related ETSI and IETF specifications.

Despite the differences and similarities of IETF SFC and ETSI NS concepts, a common element, called service topology, is defined and used by both organizations. For the ETSI, a service topology is closely related to the VNF-FG and expresses the relationships between network functions using virtual connections [21]. The IETF describes service topology as the element that defines the connections between service functions which are instantiated "on the top" of a physical network [13]. Consequently, the IETF service topology represents one or more paths that steer the network traffic through the functions of an SFC [24]. Thus, the service topology is implemented by the routing decisions taken by the SFFs during the execution of a service function chain [22]. Fundamentally, a service topology represents the network functions, virtual connections, data ingress and egress nodes, and other information which is typically used in the service deployment (e.g., partially ordered segments, connection topologies, and dependencies among network functions or infrastructures).

Based on the specifications presented by ETSI and IETF, in the next section we define a service topology as a directed connected graph. A service topology can consist of one or more "Service Function Paths" (as defined by the ETSI) or to "Paths" (defined by IETF) originating from a common ingress node. Note that a single virtualized network service can be provided by different service topologies [25], the decision of which topology will be used is usually taken during the composition phase of the deployment process.

A few definitions are given now. A service topology "A" is said to be *contained* in topology "B" if and only if "B" contains every path of "A" (i.e., the same network functions, ingress and egress nodes, and internal connections). Even if service topology "A" is not contained in topology "B", it can *share* common components with "B".

A complete service can consist of more than one topology. This happens, for example, with *symmetric* services which process traffic in both directions, called direct and inverse [13,22]. In each way the egress and ingress nodes are switched – thus the inverse traffic is processed by the same network functions used for the direct traffic but in the opposite direction. Note that the direct and inverse directions can or use not all but a subset of each other's functions. Furthermore, topologies can have cycles as described in Halpern and Pignataro [22] in which some NF instance appears more than once. This work as well as all NFV enablers that we are aware of do not allow cycles.

In summary, both the ETSI and the IETF are working on NFV standards. Both have defined concepts, architectures, and elements required to promote the deployment of network services. However, there are several differences in the two approaches. While the ETSI is concerned with a broad discussion about services and their specifications (i.e., in terms of service topology, functionalities, performance and policy descriptions), the IETF focuses on architectural aspects to support the deployment and maintenance of those services. Finally, it is important to highlight that both views can be seen as complementary and both should be taken into account for the adoption of NFV-based solutions.

3. Service topology: a taxonomy

Multiple aspects must be taken into consideration when dealing with virtualized network service orchestration and lifecycle management (e.g., placement, migration, and scaling). Wrong decisions taken by network operators may ultimately lead to performance degradation and violation of policies, affecting not only the service itself but the underlying infrastructure as a whole. To properly deal with this challenge, it is fundamental to understand service topology concepts and features, for which we define a taxonomy in this section. We argue that NFV enablers may rely on the proposed taxonomy to define how to deal with the multiple scenarios that may arise in the NFV management processes.

Next, we first formally define a service topology as a directed graph and then present the proposed taxonomy, depicted in Fig. 1. The taxonomy is based on six categories: Structure, Length, Size, Sharing, Function Dependency, and Heterogeneity. For each category, we present a discussion based on recent efforts available in the literature, including concepts and application cases.

3.1. Formal definition

A network service topology is represented by graph $G = (V, E)$, where V is a set of network functions, the data ingress node, plus data egress nodes. E corresponds to the set of virtual connections between network functions. A service topology has a single data ingress node that is connected to a network function that processes all the incoming traffic. The ingress node is depicted as a triangle plus an virtual connec-

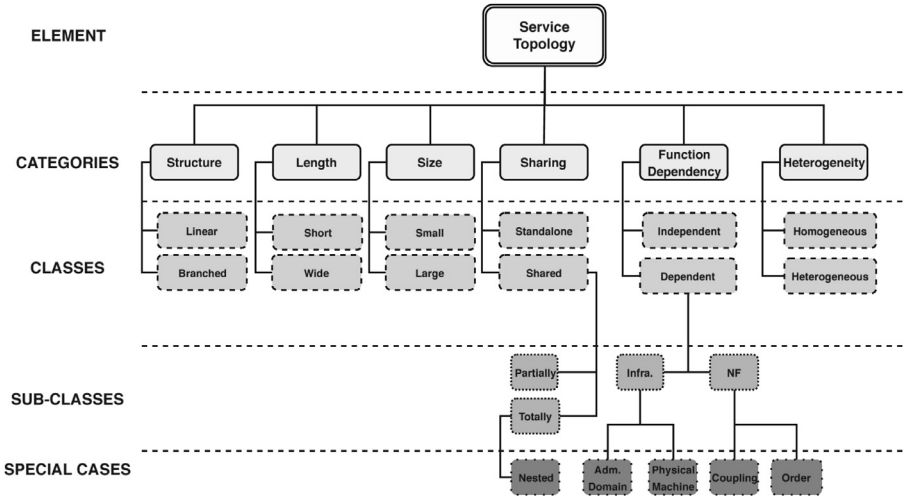

Fig. 1. Service topology taxonomy.

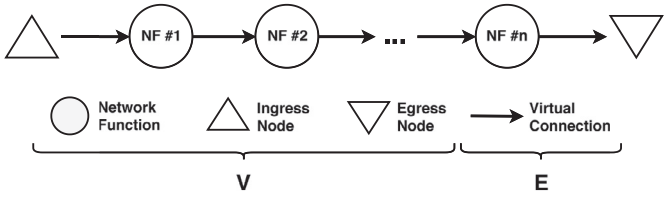

Fig. 2. Service topology elements.

tion to the first function of the chain. A service topology may have one or more egress nodes, which correspond to the entities that receive the traffic from the last function of the chain. An egress node is depicted as an inverted triangle receiving data from a virtual connection leaving the last function of the chain. Service topology elements are depicted in Fig. 2.

3.2. Structure

Service topologies can vary from simple linear chains to arbitrary graphs. The structure is directly related to the complexity of managing virtualized network services. For example, monitoring and estimating the performance of a linear chain of functions can be easily done using traditional methods (e.g., evaluating the incoming traffic at a single ingress node and the outgoing traffic at a single egress node). However, network services provided by such chains are more likely to be susceptible to system failures, either because of misconfigurations or unexpected high network loads in terms of traffic. On the other hand, although services topologies with more complex structures can be more resilient and scalable, other problems may arise, such those related to the optimal placement of network functions with restrictions in terms of reducing resource consumption.

In [26], the authors assume two types of service topology structures: linear and bifurcated. The former refers to a set of network functions connected one by one forming a single chain, while the latter describes topologies that split the network traffic into two different paths but share a single ingress node. These structures can also have up to two different destinations (egress nodes); this is convenient for specifying that different portions of the network traffic should be processed differently (e.g., in a load balancing scenario). The authors of [27] also adopt the linear and bifurcated classification and include the possibility of bifurcated chains having a common destination (single egress node). Linear topologies are also discussed in other works [28], but sometimes with different names (e.g., cascading chains in Ding et al. [29]).

However, bifurcated topologies may not be enough to fulfill the needs of all possible scenarios. For example, when a single service receives traffic from different application protocols (e.g., Dynamic Host Configuration Protocol (DHCP), File Transfer Protocol (FTP), Post Office Protocol (POP), Hypertext Transfer Protocol (HTTP)), it may use a packet classifier to split the network traffic through more than two distinct paths. In this context, Ye et al. [28] presents the concept of irregular mesh chains (which can be seen as structurally the same as the branching chains defined in Ding et al. [29]). In this case, a single network function that composes the service topology can forward packets through multiple paths depending on traffic characteristics. This kind of topology structure provides flexibility for network operators to design diverse network services.

In this work we propose to classify service topology structure in two types: linear and branched. Linear topologies, as discussed above and in Luizelli et al. [26], Ye et al. [28], and Ding et al. [29], are the simplest type of structure and present a linear sequence of network functions. In this way $\forall v \in V$ there is either exactly one arc (v, t) from function v to function t in E, plus the ingress and a single egress node. The analysis of such service topology is straightforward since all the network functions are processing all the traffic in the same order.

Branched topologies, in turn, not only have functions with a single outgoing arc as in linear topologies, but also $\exists v \in V | (v, t_1), \ldots, (v, t_i) \in E, i \geq 2$. Branched topologies correspond to the irregular mesh chains of [28] and branching chains of [29]. Basically, branched topologies are not only defined by allowing multiple paths between network functions, but also by allowing multiple boundary nodes connected to the same function. In some cases, boundary nodes are chosen according to the characteristics of the infrastructure (e.g., processing capacity, power consumption, and security concerns). This is the case when load balancers are employed to split the network traffic to distinct nodes that basically provide the same service but present different performance profiles according to the metrics being considered. We note that bifurcated chains [26] can be considered as a subset of branched chains.

Some network services can be deployed with linear topologies, for example, in home networks the Customer Premise Equipment (CPE) can be replaced by three linearly connected VNFs [3]: a router, a Set Top Box (STB), and the Residential Gateway (RG) (e.g., firewall, Network Address Translation (NAT) service, DHCP server). However, more sophisticated services (involving decisions based on the incoming traffic or current state of the environment) may be required in several scenarios, and in this case, branched topologies must be employed. Security applications are good examples: the network traffic must be processed at multiple levels and involving possibly elaborate decisions. In [30], for example, the authors present an NFV-based multi-layer security ar-

chitecture which can split the incoming traffic according to its type (i.e., malicious or not), to different layers according to the kind of threat it poses (e.g., network layer, application layer).

3.3. Length and size

A path of a service topology is a sequence of vertices $v_1, v_2, \ldots, v_n$ such that each $v_i \in V$ and $(v_i, v_{i+1}) \in E$, furthermore there is an ingress node connected to v_1 and v_n is connected to a egress node. The length of a path n corresponds to the number of network functions along the path. The service topology length corresponds to the length of the largest path of the topology [31].

The service topology size, in turn, can be measured in two ways: it can either correspond to the total number of network functions and boundary nodes in the topology (i.e., $|V|$) or the total number of connections between functions (i.e., $|E|$). The length and size of a service topology are of course interrelated.

The number of network functions composing a service topology and their organization have an influence on the network performance and behavior as noted by Riera et al. [32]. In that work, topologies are classified into simple (which consist of three or less functions) and complex. The authors evaluate topologies in different network environments and applications (i.e., edge computing, data centers, and heterogeneous environments). The authors conclude that not only the deployment but also the number of network functions that compose a network service (i.e., the service topology size) must be taken into account to guarantee bounds on network service execution time.

In [33] the authors evaluated the impact of service topology length on network performance, and show that latency does not always increase linearly as the length increases. The authors also identified that large topologies require more sophisticated processes for managing, provisioning, and placing the network functions - especially when heterogeneous environments are employed. Another work [34] discusses NF dependency aiming at the parallel execution of the network functions of a given service. The authors concluded that reducing the service topology length through parallelization also reduces the overall processing time.

There is no consensus about specific criteria to determine whether a service topology is wide or short (length) and large or small (size). These often depend on the environment, service, and operator needs. In order to formally define this concept there must be some constant Θ which is a threshold above which the size of the topology is considered large, otherwise it is small. Analogously, a constant Ψ must be employed as a threshold to the length in order to classify a topology as wide or short. Note that it is not possible to predict network performance based only on length/size. For example, both short and small topologies may lead to poor performance (e.g., high latency, low throughput) due for instance to limited computational resources or any other reason. At the same time, wide and large service topologies executing lightweight functions deployed on the same physical machine may reach results close to barebone middleboxes [35].

3.4. Sharing

Resource utilization is a major concern for the development of NFV enablers. There are some techniques (e.g., scaling down and scaling in) that aim to avoid resource idleness, and adapt services to increase or decrease their load as required. Another approach that can be used is to allow the same network function to be used by two distinct network services which thus share one or more function instances [36]. Thus a shared function receives traffic addressed to two or more services, and after processing the traffic accordingly forwards the results to the next function or endpoint according to the corresponding service topology. In this way two services represented by graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are such that $V_1 \cap V_2 \neq \emptyset$; $\forall v_i \in V_1 \cap V_2$ we say v_i is a shared

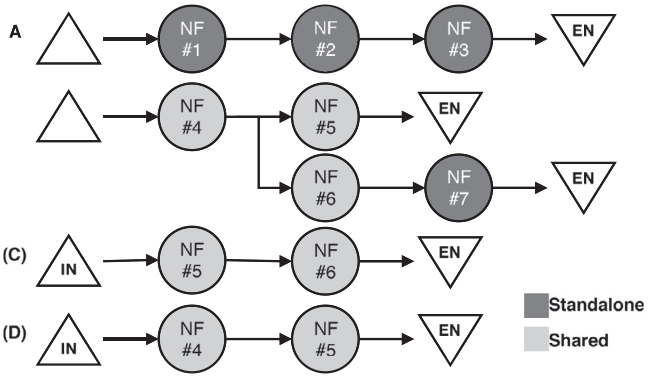

Fig. 3. Sharing NFs by multiple service topologies.

function. As a side effect, network function sharing may impact the management process, as there are no standardized techniques to determine how much computing resources are being used by the shared networks functions to support each virtualized network service.

The IETF SFC architecture allows network functions sharing among multiple services [22]. In this architecture, the functions are completely agnostic to sharing, and all the tasks required to handle the traffic flow are provided by the SFC Controller, the SFC Forwarder, and the Classifier. The use of Software Defined Networking (SDN) also makes it feasible to support network function sharing [37,38].

A service topology can be classified into two classes according to network function sharing: standalone or shared. Standalone topologies employ network functions dedicated to handling data addressed to a unique service ID, in other words, this corresponds to a service topology that has no intersections with any other service topology and does not share any particular resource with other services. Thus if the service topology represented by graph $G_1 = (V_1, E_1)$ is standalone, then $\forall v \in V_1, \nexists G_2 = (V_2, E_2) | v \in V_2$. Fig. 3 presents a scenario with three different service topologies. Each network function is depicted with a unique ID. Fig. 3A exemplifies a standalone service topology with dedicated NFs #1, #2 and #3.

Shared service topologies, on the other hand, have at least one network function present in two or more topologies. Shared service topologies can be divided into two sub-classes: partially and the totally shared topologies. Partially shared service topologies contain at least one dedicated network function, but not all. Consider two services represented by topologies $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$; we say that topology G_1 is partially shared if $V_1 \cap V_2 \neq \emptyset$ but $V_1 \not\subset V_2$. Fig. 3B is an example of a partially shared topology, where NF #7 is dedicated and NFs #4, #5 and #6 are shared.

Totally shared service topologies, as depicted in Fig. 3C, contain only network functions that are shared by two or more services, so all changes affect multiple topologies. Consider two services represented by topologies $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$; we say that topology G_1 is totally shared if $V_1 \subseteq V_2$. A special case of a totally shared service is the so-called nested service topology (Fig. 3D) and occurs when all the network functions and connections of an service topology are within another topology (called composite topology). Thus in this case we say that G_1 is nested in G_2 if $V_1 \subseteq V_2$ and also $E_1 \subseteq E_2$. Those service topologies are analog to Composite Network Services and Nested Network Services [21], where one service topology is nested within another topology (Fig. 3B). It is possible to merge the composite and nested service topologies just by adding boundary nodes in the composite service path, thus they can be treated as a unique element.

The decision for sharing network functions can be taken according to the service characteristics and goals. Standalone topologies may be adequate, for example, for services where changes must be performed individually for each client [39]. On the other hand, shared topologies

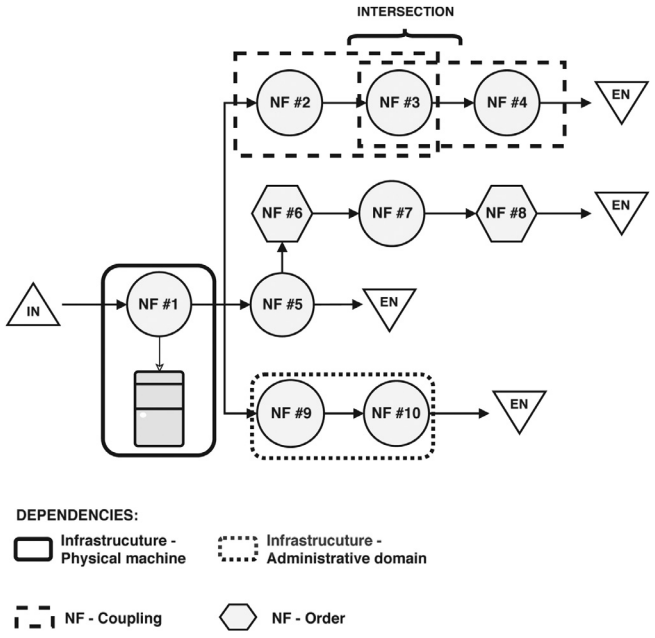

Fig. 4. Service topology dependency.

can be used when the service is provided "as-a-Service", with multiple service levels provided according to the client agreements [40].

3.5. Function dependency

Network services are highly dependent on the environment in which the network functions are virtualized and executed [13]. Disparate restrictions may appear when chaining single network functions into sophisticated services, thus impacting the performance, limiting placement options, and reducing orchestration flexibility. Those aspects, however, need to be properly tackled by the operators, leading to additional overhead in fulfilling network policies and managing the network.

In this work, we consider that dependency leads to a classification of service topologies in two main groups: independent and dependent. An independent service topology does not have any coupling requirements/restrictions and is the most flexible as it allows automated optimization algorithms to be executed without any limitations, as well as migrations to occur at any moment. On the other hand, dependent service topologies present restrictions that can interfere on the optimal placement of its network functions. Dependency can be either of infrastructure or network functions, as discussed next.

The network functions of a service topology with infrastructure dependency are coupled to an administrative domain, such as NF #9 and #10 in Fig. 4, or to physical machines, as exemplified by NF #1 in Fig. 4. In such topologies, the network functions must execute in predefined locations due to some policy adopted or specific restriction (e.g., data-residency requirements that confine data within a domains borders [41]). Placing network functions at specific locations may however remove the freedom of the network administrator from applying arbitrary criteria in activities such as traffic engineering; function placement restrictions can lead to performance problems [13].

Service topologies with NF dependency, on the other hand, can have an impact on the organization and segmentation of structures [42–45] and can be divided into two types: *(i)* service topologies that consist of a group of strongly coupled network functions that must execute together. Thus for service topology $G = (V, E)$ if function v_j is coupled to v_i, then $\exists(v_i, v_j) \in E$. In Fig. 4 NFs #2, #3 and #4 are coupled; and *(ii)* service topologies consisting of network functions that have to obey a specific order because of their functionalities. Thus functions v_i and v_j present order dependency if in every directed path $v_1, \ldots, v_i, \ldots, v_j, \ldots, v_n \in G$ in which v_i and v_j appear there is always a subpath from v_i to v_j. In Fig. 4, NFs #6 and #8 present order dependency.

The dependency in a service topology can both be explicit, determined by the service creator, or be implicit, defined by the intrinsic characteristics of network functions being used, the environment on which it is executed, or the network service being provided. For example, order dependency is usually implicit (e.g., a packet compressor must always be placed before a packet decompressor), while domain dependency is typically explicitly defined.

Dependencies can also intersect as shown in Fig. 4 in which NF #3 which is subject to two NF dependencies. If the intersection of dependencies does not create a conflict, the service can be executed respecting all dependencies. However, if it is not possible to satisfy all the dependencies in the intersection, extra care must be taken to avoid and solve conflicts. A priority based scheme can be adopted to guarantee the execution of services topologies with multiple dependencies to avoid/solve conflicts. The ability to specify service topology dependencies is important. Note that this includes, for example, privacy restrictions, which may prevent data from being stored/processed/transferred outside certain regions.

3.6. Heterogeneity

While the NFV paradigm is being exhaustively investigated, middleboxes are still essential elements in current networks. As late as 2015, about 50% of all network elements were implemented in dedicated hardware [46], and it is not feasible to replace these middleboxes in the short term, especially due to financial and operational reasons. In this context, it is adequate to deploy services composed by both physical and virtualized network functions. The ETSI has already explored the existence of mixed scenarios where NFs work together with middleboxes through a hybrid vCPE [14]. In this scenario, it is important to be able to use both paradigms concurrently.

In this work, we define two types of service topologies: homogeneous and heterogeneous. Homogeneous service topologies use only virtualized network functions. Heterogeneous service topologies, on the other hand, allow the use of legacy and dedicated hardware-based network functions – called Physical Network Functions (PNF) [21] – together with VNFs to compose a complete service. In the formal model in which a service topology is represented by graph $G = (V, E)$ it is possible to define a label which can be either *physical* or *virtual* to each $v \in V$. If $forall v \in V$ the label is *virtual* then the topology is homogeous, otherwise if $\exists v \in V$ with label *physical*, then the topology is heterogeneous. The support for such integration is important to enable the practical use of the NFV.

Many challenges are still open related to heterogeneous services topologies, including function scaling and migration. Ongoing efforts on heterogeneous services aim at maximizing the amount of traffic processed by reducing the cost of routing as well as VNF overhead [47]. Moreover, management systems and placement methods are being developed to make possible the creation of network services that are aware of heterogeneity - in particular, so that they can deal with PNF limitations [48,49]. Security Service Chains [50] is a technology that explicitly considers the use of heterogeneous service topologies employing PNFs because of the higher level of maturity of this technology leading to higher stability and performance levels; they demonstrate the ability to process higher volumes of traffic for applications such as for attack identification and mitigation [51].

3.7. Synthesis

In Table 2 we show for each service topology category the references that present related work.

Table 2
Taxonomy categories main references.

	Category				
	Structure	Length and size	Sharing	Function dependency	Heterogeneity
Ref.	[26–29]	[31–34]	[22,36–38]	[13,42–44]	[14,21,46–49,51]

4. CUstom service TOpology model

In this section, we present a new service topology specification model. This model, called CUstom Service TOpology Model (CUSTOM), aims at covering all the categories defined in the taxonomy presented in Section 3, thus providing for service developers a comprehensive set of specification capabilities to design, create and validate customized topologies.

CUSTOM is a formal Context-Free Grammar (CFG) model. CUSTOM foresees a variety of structures, from straightforward linear topologies to topologies with terminal or non-terminal branches. For terminal branchings, every branch segment ends in an egress node. On the other hand, in non-terminal branchings there is a common crossing point between all the branch segments, and they do not have egress nodes. The position of specific network functions (i.e., VNFs and PNFs) along a topology can be predefined. Furthermore, it is also possible to specify segments that consist of a sequence of functions that appear in a partial order [25]. In partial ordering segments, specific order restrictions can be defined through network function dependencies (e.g., packet compressing function must appear before a packet decompressing function). Infrastructure dependencies can also be specified in network functions. VNFs could have a single infrastructure dependency of any type (administrative domain or physical machine), while PNFs must have an administrative domain dependency. Finally, functions can be shareable or not. Shareable functions can be adopted by different service topologies at the same time.

It is important to notice that CUSTOM is focused on a very particular type of specification, i.e. it is used to specify service topologies. Thus, the CUSTOM model does not provide any information about, for example, computational resource requirements, minimal bandwidth required between network functions, or any other metric that is considered for service deployment and management. Actually, this kind of information (i.e., service topology) is often present in high-level documents such as service descriptors (e.g., TOSCA NFV YAML[52]) and service requests (*e.g*, the ones used in Mehraghdam et al. [16], Drxler and Karl [18], Vizarreta et al. [53]). Note that in most cases these high-level documents do include a service topology. For example, in TOSCA NFV YAML, the service topology is specified through multiple objects (key:value) that create connections between previously defined network functions. This ultimately leads to a very intricate model that, to be specified, requires of network operators high expertise on the descriptor structure, including a myriad of details. Thus we claim that the CUSTOM model can be used as an abstraction of these indirect models, being directly employed by deployment request documents (e.g., we refer the reader to the reference in Mehraghdam et al. [16] for a related work that exemplifies how this is done) or acting as an auxiliary model for descriptor documents, such as TOSCA NFV YAML.

The adoption of a CFG is motivated owing to the formalization level achieved by using grammar production rules. These rules strongly define the set of service topology features supported by the model. Also, many existing libraries developed for different programming languages, are capable to execute the lexical and syntactic validation of context-free grammars. (e.g., Lex [54], Yacc [55], and Natural Language Toolkit (NLTK) [56]). At last, the grammar production rules can be defined to improve readability. So, service developers can easily understand these rules, thus reducing time necessary for the development of service topologies.

The CUSTOM CFG is defined by the quadruple CUSTOM = (ν, τ, ρ, ς). The identifier ν represents a set of non-terminal symbols that correspond the production rules as follows: `START`, `MAIN`, `NTMAIN`, `OPERATIONAL`, `PORDER`, `FDEPENDENCY`, `FORDERING`, `FCOUPLING`, `TBRANCHING`, `TBRANCH`, `NTBRANCHING`, `NTBRANCH`, `INTBRA- NCHING`, `FUNCTION`, `NFUNCTION`, `ADDEPENDENCY`, `PMDEPENDENCY`, `SHARABLEVNF`, `SHARABLEPNF`, `VNF`, `PNF`, `ADMDOMAIN`, `PHYMACHINE`, and `EN`.

The identifier τ, in turn, indicates the terminal symbols of the service topology model. These symbols are divided in two subsets: static and variable. The static subset is composed by 13 symbols and contains mandatory elements (`'IN'`), delimiters (`'['`, `']'`, `'('`, `')'`, `'<'`, `'>'`, `'{'`, `'}'`, `'/'`, `'|'`), and modifiers (`'*'`, `'!'`). On the other hand, the variable subset includes customized service topology symbols and can vary in type and number according to each different specification. In particular, variable symbols represent the available VNFs, PNFs, Administrative Domains, Physical Machines, and Egress Nodes. The terminal symbols and their main assignments are summarized in Table 3.

The terminal (ν) and non-terminal (τ) symbols are employed in the definition of grammar production rules (ρ). The CUSTOM production rules ρ consists of 24 rules as shown in Fig. 5. It is important to highlight that motivated on providing compatibility with any available lexer and parser libraries, no empty symbol was used in the production rules. Finally, ς is the production rule that triggers the grammar evaluation. In the CUSTOM CFG, ς is the rule indicated by the non-terminal symbol `START`.

Rule 1 (`START`) indicates the beginning of a service topology. It includes a mandatory ingress node (`IN`) and specifies the transition to the second rule. `START` is the only rule that sets up an ingress node, and this rule is reached just one time per specification. Thus a single ingress node can be defined in each service topology in a CUSTOM specification. This ensures that, at least, one network function (i.e., the very first VNF or PNF allocated after the ingress node) will receive all the incoming network traffic. Thus, every specification made with this grammar is compliant with the service topology requirements outlined in Section 2.

Rule 2 (`MAIN`) defines the service topology main structures. These structures consist of terminal branchings (Rule 9), non-terminal branchings (Rule 11), and operational segments (Rule 4) followed by an egress node (Rule 24) or another operational segment. Other structures can be defined as shown in Rule 3 (`NTBMAIN`) in the context of a non-terminal branching. The `NTBMAIN` rule ensures that no egress node will be set up until the end of the previously started non-terminal branching. In order to do so, this rule removes every transition which, directly or indirectly, leads to Rule 24 (`EN`). Rule 13 switches the last main block to a non-terminal main block, allowing the specification of recursive non-terminal branchings. Moreover, the specification of different branches within a branching structure is in Rules 10 (`TBRANCH`) and 12 (`NTBRANCH`), for terminal and non-terminal branching, respectively.

Operational segments presented in Rule 4 correspond to a function with fixed position (Rule 14) or a partially ordered function segment (Rule 5). In the first case, functions can be defined as virtual (Rule 20) or physical (Rule 21) and both types can be marked as shareable or non shareable (Rule 18 for VNFs and Rule 19 for PNFs). In the case of partially ordered segments, they consist of a set of functions with flexible ordering. For these segments, NF dependencies can be specified using Rule 6 (`EDEPENDENCY`) to guarantee an internal ordering (Rule 7) or cou-

```
START → 'IN' MAIN
MAIN → TBRANCHING | NTBRANCHING | OPERATIONAL MAIN |
    OPERATIONAL EN
NTBMAIN → INTBRANCHING | OPERATIONAL NTBMAIN | OPERATIONAL
OPERATIONAL → PORDER | FUNCTION
PORDER → '[' FUNCTION NFUNCTION ']' EDEPENDENCY | '['
    FUNCTION NFUNCTION ']'
EDEPENDENCY → EORDERING | ECOUPLING
EORDERING → '(' FUNCTION FUNCTION ')' EDEPENDENCY | '('
    FUNCTION FUNCTION ')'
ECOUPLING → '(' FUNCTION FUNCTION '*' ')' EDEPENDENCY | '('
     FUNCTION FUNCTION '*' ')'
TBRANCHING → OPERATIONAL '{' MAIN TBRANCH '}'
TBRANCH → '/' MAIN TBRANCH | '/' MAIN
NTBRANCHING → OPERATIONAL '{' NTBMAIN NTBRANCH '}' MAIN
NTBRANCH → '/' NTBMAIN NTBRANCH | '/' NTBMAIN
INTBRANCHING → OPERATIONAL '{' NTBMAIN NTBRANCH '}' NTBMAIN
FUNCTION → SHARABLEVNF | SHARABLEVNF ADDEPENDENCY |
    SHARABLEVNF PMDEPENDENCY | SHARABLEPNF ADDEPENDECY
NFUNCTION → FUNCTION NFUNCTION | FUNCTION
ADDEPENDENCY → '<' ADMDOMAIN '>'
PMDEPENDENCY → '<' PHYMACHINE'|'ADMDOMAIN '>'
SHAREABLEVNF → '!' VNF | VNF
SHAREABLEPNF → '!' PNF | PNF
VNF → 'VNF#1', 'VNF#2', ..., 'VNF#n'
PNF → 'PNF#1', 'PNF#2', ..., 'PNF#n'
ADMDOMAIN → 'AD#1', 'AD#2', ..., 'AD#n'
PHYMACHINE → 'PM#1', 'PM#2', ..., 'PM#n'
EN → 'EN#1', 'EN#2', ..., 'EN#n'
```

Fig. 5. CUSTOM production rules.

pling (Rule 8). Furthermore, VNFs can have infrastructure dependency of any type (machine or domain), while PNFs have a mandatory administrative domain dependency. Administrative domain dependencies are specified according to Rule 16 (ADDEPENDENCY) using the available domains as indicated in Rule 22 (ADMDOMAIN). Physical machine dependencies, in turn, are supported by Rule 17 (PMDEPENDENCY) and assign a VNF to specific hardware (Rule 23) placed in a previously specified administrative domain (Rule 22).

A validator for the CUSTOM grammar was implemented using the Python 3 programming language and the NLTK library.[1] This validator receives a service topology specification string and follows the previously presented production rules for the execution of its lexical and syntactical analysis. The algorithm also performs the semantic analysis of the specifications, avoiding inconsistencies such as conflicting dependencies (e.g., inverse dependencies applied over the same pair of functions) or auto-dependencies (i.e., a function depending on itself). Although the validator was implemented to evaluate string specifications, it is possible to represent the grammar production rules in markup languages and structured data models by using, for example, eXtensible Stylesheet Language (XSL) [57] or Yet Another Markup Language (YAML) [58].

A service topology specified with the CUSTOM grammar corresponds to a directed graph as defined in previous sections. The vertices are represented by network functions, while edges are implicitly indicated by the specified function order. The resulting graph has its scope delimited by edge nodes (ingress and egress), and the network traffic traverses the service topology from the ingress node to an egress node. Finally, the service topology model can be used in any stage of the deployment process (i.e., composition, embedding, and scheduling). To this end, a service topology can be specified to be used by a deployment system with a service descriptor (Service Function Chain Descriptor (SFCD) [59]) or request (Service Function Chain Request (SFCR) [25]).

[1] Source code is available at https://github.com/ViniGarcia/NFV-FLERAS.

Table 3
CUSTOM terminal symbols.

Terminal symbols	Description
'IN'	Ingress node - a data incoming point to the service topology. Provides frames/packets/flows to be processed by the service.
'[','] '	Partial ordering segment delimiters. These elements embrace a partially ordered set of VNFs and PNFs.
'(',')'	NF dependency delimiters. Used to define an order or coupling dependency between two VNFs/PNFs.
'<', '>'	Infrastructure dependency delimiters. Elements intended to describe an administrative domain or physical machine dependency.
'{','}'	Service topology branching delimiters. Mark the beginning and the end of a terminal/non-terminal topology branching.
'/'	Service topology branch separator. Elements that are used to separate branches.
'|'	Physical machine dependency separator. This symbol is used to separate the physical machine ID and the administrative domain ID in a dependency specification.
'*'	NF dependency modifier. This element indicates that the dependency corresponds to a coupling dependency between two VNFs/PNFs. The absence of this symbol implies on an ordering dependency.
'!'	NF sharing modifier. This symbol may occur before VNFs/PNFs. Indicates that the network function is shareable by different service topologies.
VNFs	List of virtualized network function identifiers. The non-terminal symbol `VNF` corresponds to this list.
PNFs	Physical network functions identifiers. Supported by the non-terminal symbol `PNF`.
Administrative Domains	Administrative domain identifiers used in the specifications of infrastructure dependencies. The non-terminal symbol `ADMDOMAIN` is used to keep these identifiers.
Physical Machines	List of physical machine identifiers. Employed to specify physical machine dependency and kept in the non-terminal symbol `PHYMACHINE`.
Egress Nodes	Egress node identifiers - service topology data outgoing points. The non-terminal symbol `EN` keeps the egress node list.

5. Case studies

In this section, we demonstrate and discuss the specification capabilities of CUSTOM through a series of case studies. For the sake of clarity most case studies are dedicated to a single category, however CUSTOM can easily deal with more than one category – even with all categories – in a single topology, as we show in the last case study.

5.1. Linear service topology

Network functions in a linear service topology have a single network traffic incoming point (network function or ingress node) and outgoing point (network function or egress node). This simplest type of structure is widely adopted to build a myriad of virtualized network services, such as multimedia caches [29], video live transcoding [60], mobile core networking [31], and Virtual Private Network (VPN) services [61]. In Fig. 6A, the service topology that implements the multimedia cache presented in Ding et al. [29] is depicted. The network functions that compose this service topology are Cache (C), Firewall (FW), and Network Address Translator (NAT).

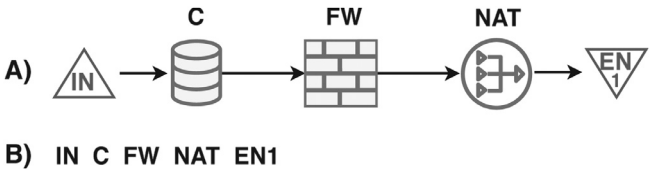

Fig. 6. Multimedia cache service topology.

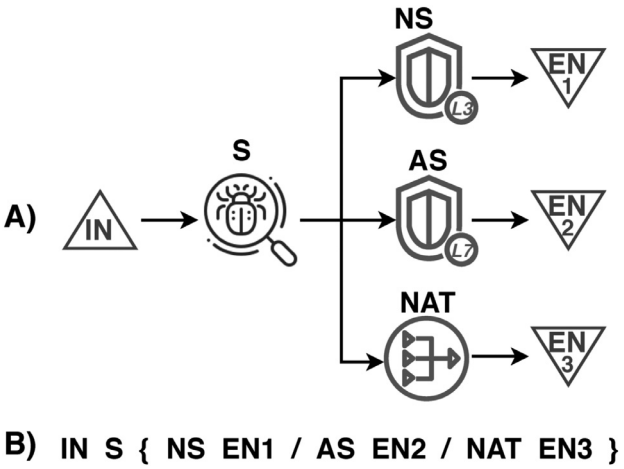

Fig. 7. DDoS mitigation service.

Fig. 6 B, in turn, shows the CUSTOM specification that corresponds to the graphical presentation of the service topology depicted in Fig. 6A. In this case, we consider that rule `VNF` of the CUSTOM grammar includes network functions "C", "FW", and "NAT". Furthermore, the egress node "EN1" must be set up with rule `EN`. Finally, the spaces between each component represent a virtual link. Note that linear service topologies, such as the one presented in our example, are identified in a CUSTOM specification by the absence of branching elements (i.e., { and }). Thus, a single end-to-end path is formed from the ingress node to the egress node in a linear structure.

5.2. Service topologies with terminal branches

Service topologies with terminal branching present multiple egress nodes in their structure – at least two, or more if nested terminal branches are defined. This type of structure is commonly used when traffic classifiers operate on a service. Thus, the network traffic is split according to some criteria and forwarded to the proper branch to be processed. Terminal branches are usually employed in NFV security services, such as the architectures presented in Alharbi et al. [30], 62] where a screener function analyzes the network traffic and classifies the flows according to the threat they pose in terms of Distributed Denial of Service (DDoS). Also, the screener classifies malicious flows according to the network layer they target and forwards each type to the corresponding mitigation branch. We illustrate the described DDoS mitigation service in Fig. 7A. The service is composed by the following network functions: Screener (S), Net-layer Security (NS), App-layer Security (AS), and Network Address Translator (NAT).

In Fig. 7B the CUSTOM specification of the DDoS mitigation service is presented. In this particular case, we set the `VNF` rule containing the network functions "S", "NS", "AS", and "NAT", while the `EN` rule has the egress nodes "EN1", "EN2", and "EN3". The branching resulting from the traffic screener classification is represented by the service segments between symbols { and }, and the specific branch segments are limited by the symbol /. We classify this branching as terminal since every branch segment ends in an egress node. Therefore, multiple end-to-end paths with different purposes can be created in the service topology.

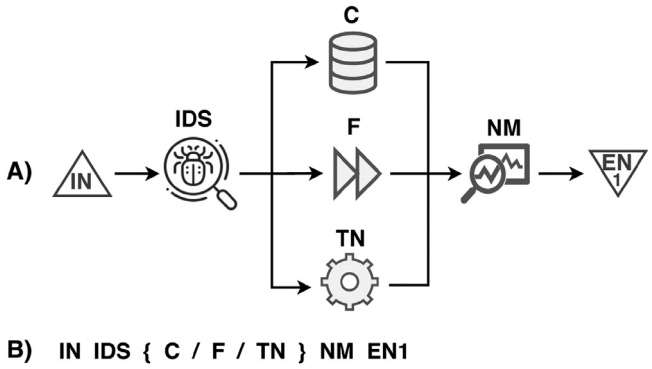

Fig. 8. Traffic analyzer service.

5.3. Service topologies with non-terminal branches

Non-terminal branching enables service topologies to define an intersection point (i.e., a VNF or a PNF) for every branch segment. In addition to being required by many types of services, such as security services and traffic classifiers [63], this structure is also employed to improve the performance of a network service through its parallel on multiple network functions [64–66]. In the case study presented in Palkar et al. [63], an Intrusion Detection System (IDS) is used to analyze and split the network traffic into three different branch segments: the first segment contains a Cache (C) function that processes safe HTTP traffic; the second segment uses a simple Forwarder (F) to bypass any other type of safe traffic; finally, the third segment executes a Traffic Normalizer (TN) function for the unsafe traffic. The outputs of these branches intersect on a common VNF, the Network Monitor (NM), that processes all the traffic. Fig. 8A depicts this service.

The CUSTOM specification of the traffic analyzer service is shown in Fig. 8B. The `VNF` contains functions “IDS”, “C”, “F”, “TN”, and “NM”, and rule `EN` consists of a single egress node “EN1”. Non-terminal branching, like terminal branching, is specified with symbols `{` and `}` with branch segments separated with an slash `/`. Observe that there is no egress node at any point of the branch segments and the service topology does not end on the branch outputs. In this service, multiple end-to-end paths are necessary, however the they all start and end in the same points. This is a typical scenario that requires non-terminal branching.

5.4. Network function dependencies

A single network service can be provided by different service topologies. In the CUSTOM model, this flexibility is achieved by using partially ordered segments in the service specification. Although the functions can have flexible ordering, some function permutations may not be allowed within a partially ordered segment. These particular cases are specified with constraints that are called network function dependencies that define mandatory ordering and coupling relationships among network functions. These dependencies are processed during the service deployment, they are processed in the composition phase. Currently, many service composing solutions, such as [44,67–69], can process network function dependencies during their execution.

We present a case study for the specification of network function dependencies in an HTTP security service. This case study, shown in Fig. 9A, involves five different network functions: Firewall (FW), Deep Packet Inspector (DPI), Markup Filter (MF), Intrusion Prevention System (IPS), and Load Balancer (LB). The FW is responsible to eliminate non-HTTP traffic. The packets of HTTP flows are inspected by the DPI function and receive a mark in case they contain forbidden content; marked packets are then discarded by the MF. The IPS discards flows classified as malicious and forwards the non-malicious flows to the next function. Finally, the LB receives the processed traffic and distributes the HTTP requests among the available servers.

In this case study, two NF dependencies are defined: (i) the ordering between FW and DPI functions; and (ii) the coupling between the DPI and MF functions. The first dependency ensures that the DPI will only process the network traffic for which it was designed for (i.e., HTTP), thus the FW must process the traffic earlier than the DPI. The second dependency ensures that illegal HTTP traffic will not be processed by the other functions, as it is discarded by the coupling of DPI and MF functions. The CUSTOM specification of the HTTP security service is shown in Fig. 9B. The `VNF` rule contains functions “FW”, “DPI”, “MF”, “IPS”, and “LB”, and the `EN` rule contains the egress node “EN1”. The symbols `[` and `]` establish the limits of the partially ordered segment. Furthermore, the network function dependencies within the partially ordered segment are specified with symbols `(` and `)`. It is important to notice that the specification of the ordering dependencies does not include any modifier symbol, while the specification of coupling dependencies does (`*`).

5.5. Infrastructure dependencies

It is possible to specify network services in which some network functions have constraints of where they should be executed, both in terms of hardware and domain. These constraints can be specified by the network operator for many reasons, including for example security and accountability requirements, the use of hybrid environments with legacy physical appliances (PNF), and particular resource requirements that are not available on every machine. The CUSTOM model allows the specification of infrastructure constraints with two types of dependencies: administrative domains and physical machines. Administrative domain dependencies are typically employed when a service mapping is done on multiple domains. On the other hand, physical machine dependencies are used as to map the service topology to the physical resources of a data center.

5.5.1. Administrative domain

A DeMilitarized Zone (DMZ) service (e.g., [70]), illustrated in Fig. 10A, is used to demonstrate the specification of administrative domain dependencies. This service contains five network functions: External Firewall (EFW), Cache (C), Internal Firewall (IFW), Antivirus (AV), and Network Address Translator (NAT). The EFW function filters all the traffic that does not represent a request to a known service provided neither by the C function, nor by other services provided in the internal network. The first two functions (EFW and C) are in a DMZ, note that the C function only provides public services and information. If a request requires a private service/information, the C function forwards the request to the IFW function that represents a gateway between the DMZ and the internal network. This second firewall protects the internal network granting access to authorized sources only. Furthermore, the AV function improves the security by checking if the incoming traffic is not malicious. Finally, the address translation is done by the NAT function.

For security reasons, the functions that access the internal network must be deployed in a private domain called PD. Therefore, they can be monitored by network operators in an accountable and auditable environment. The CUSTOM specification of the DeMilitarized Zone service is presented in Fig. 10B. Rule `VNF` contains network functions “EFW”, “C”, “IFW”, “AV”, and “NAT”, and the rule `EN` has the egress node “EN1”. Also, the rule `ADMDOMAIN` specifies the previously mentioned private domain “PD”. In the service topology, the last three functions present the same administrative domain dependency (PD). These dependencies are specified with “PD” between the symbols `<` and `>`. As the other functions do not have infrastructure dependencies, they can be allocated to any available domain during the mapping process, which can include outsourced Network-as-a-Service providers (e.g., FENDE [5]).

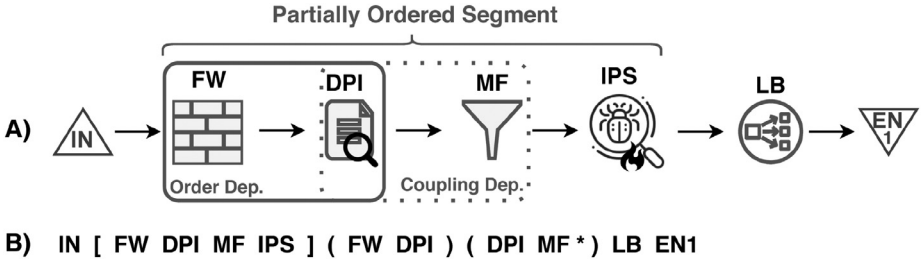

Fig. 9. HTTP security service.

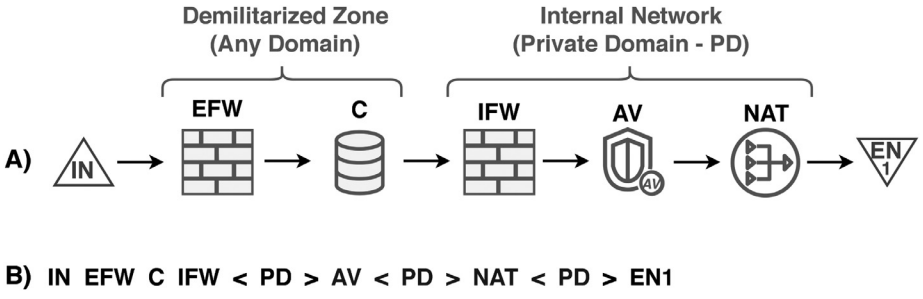

Fig. 10. Demilitarized zone service.

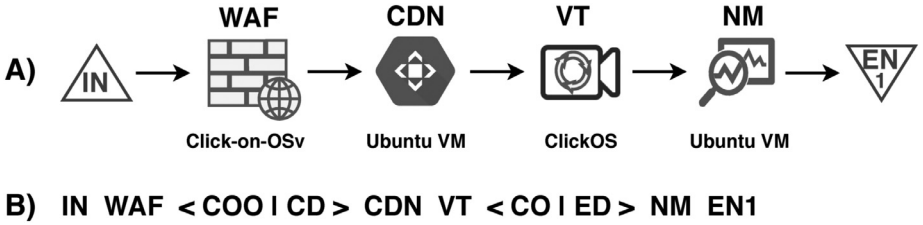

Fig. 11. Video optimizer service.

5.5.2. Physical machine

The case study to demonstrate physical machine dependencies is a video optimizer service. This service, shown in Fig. 11A, consists of four network functions: Web Application Firewall (WAF), Content Delivery Network (CDN), Video Transcoding (VT), and Network Monitor (NM). WAF acts as a layer 7 firewall and reverse proxy that intermediates the commutation between customers and media servers, which is triggered when some particular content is not available from a specific CDN point of presence. The CDN is responsible to cache content from the media service and provide the content as a stream to the customers. VT adjusts content streaming (e.g., downgrades or upgrades video resolution) according to the customer requirements and network metrics such as available bandwidth, latency, among others. The NM function monitors content streaming, QoS metrics, and QoE reports to help the VT to properly configure the streaming quality as well as the service itself (e.g., by allocating new CDN points of presence if the demand for some content grows in a certain region).

In order to guarantee fast packet processing for complex functions (WAF and VT), we should employ platforms tailored to NF execution (Click-on-OSv [10] and ClickOS [8]). However, Click-on-OSv uses the DPDK[2] packet accelerator that requires an Intel processor. Moreover, ClickOS depends on the Xen hypervisor.[3] Thus, these network functions cannot be deployed in an arbitrary physical machine, but in one that supports the required technologies. We solve this problem by pinning both WAF and VT onto specific physical machines, respectively COO and CO. The first machine (COO) is located at the Cloud Domain (CD) near the media server. The second machine (CO), in turn, is in the Edge Domain (EG) close to end-users. The CUSTOM specification of this case study service is shown in Fig. 11B. In addition to rule `VNF` that includes "“WAF”, “CDN”, “VT”, and “NM”, rule `EN` corresponds to the egress node “EN1”. The infrastructure rules `ADMDOMAIN` and `PHYMACHINE` are configured with “CD” and “ED” domains and “COO” and “CO” machines. Other functions do not have any dependencies and can be mapped to and placed on any available infrastructure.

5.6. Shared network functions

The Network-as-a-Service (NaaS) model [71] was developed inspired by the widely adopted Software-as-a-Service (SaaS) model [72]. A relevant feature of these “as-a-Service models” is the ability to share resources. Among other features, NaaS allows providers and the customers to decide, usually with the help of Service Level Agreements (SLA), which computational resources (including software) can or cannot be shared. NFV technology allows network functions to be shared during the embedding phase of the deployment process. In order to do that, the selection technique is employed so that the network service is embedded using shareable network functions that have been already deployed on the underlying infrastructure [73]. Of course, in a typical scenario, not all network functions can be shared, due to several different reasons. However, many types of NFs, such as antiviruses and virtualized customer premises equipment, can be provided in a shareable manner.

We illustrate how CUSTOM deals with shareable VNFs through the specification of a virtualized Customer Premises Equipment (vCPE) [74], as depicted in Fig. 12A. This service consists of five network functions: Set-Top Box (STB), Parental Control (PC), Residencial Gateway (RGW), Antivirus (AV), and Broadband Network Gateway (BNG). The STB is used to authenticate users and provide media content, such as IPTV and movie streams. Requests for content are validated by the PC function and then forwarded to the BNG with the RGW. Before arriving at the BNG function, all the network traffic must be verified by an AV.

[2] https://www.dpdk.org.
[3] https://xenproject.org.

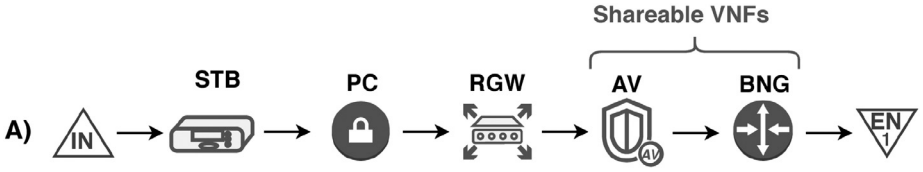

Fig. 12. Virtualized customer premises equipment.

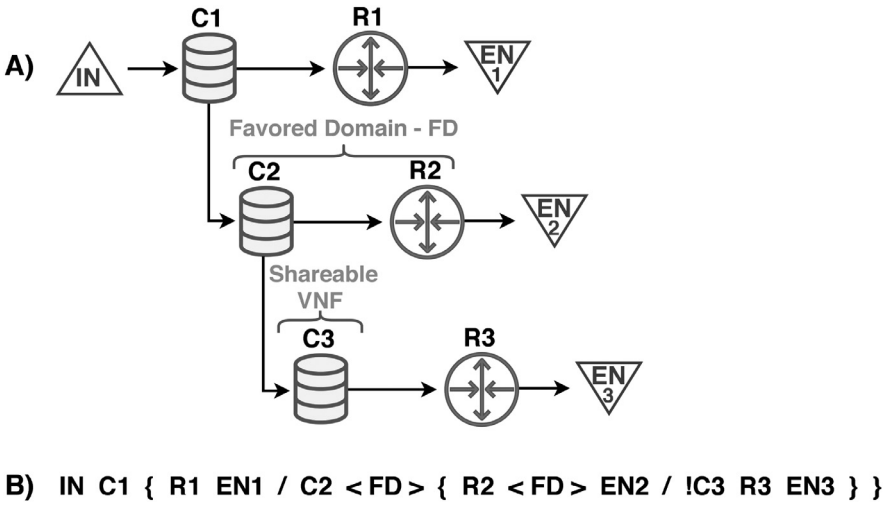

Fig. 13. Hierarchical content delivery networks.

Finally, the BNG forwards the requests to the server that provides the proper media service.

Note that the first three functions (i.e., STB, PC, and RGW) are customized to the end-user contract agreements and requirements. Furthermore, once any contracted service change occurs, these network functions have to be modified to reflect the new set of functionalities and configurations. Therefore, these functions should not be shared among different customers. The last two functions (i.e., AV and BNG) in turn have the same operational behavior, regardless of the customer from which the traffic is coming, making them good options to be shared. The CUSTOM specification of the vCPE service is shown in Fig. 12B. The network functions "STB", "PC", "RGW", "AV", and "BNG" are included in rule `VNF`, and the egress node "EN1" is in rule `EN`. Shareable network functions are preceded by the modifier symbol `!`. We highlight that the presence of shareable functions does imply any other change to any logical connection in the service topology specification, but they do have an impact on how the service is deployed.

5.7. Multiple category service topologies

CUSTOM can deal with the specification of multi-category service topologies complex service topologies belong to multiple categories. To demonstrate an example, consider the specification of a hierarchical Content Delivery Network (CDN) service composed of chained caches and routers. These functions are mapped in a multi-domain environment and provide multimedia content for customers. A specific cache can both receive (push) and require (pull) multimedia content from the previous cache in the chain and provide, if necessary, content to the next cache (caches are identified by ascending numeric IDs); the first cache is an exception that communicates directly with the main server to receive/require new content. Caches provide content to their customers through the respective attached routers. Fig. 13A depicts the described network service topology. Caches and routers are named, respectively, "C" and "R" plus a numeric ID.

In this service, the second cache function (i.e. C2) is a legacy physical machine located at a Favored Domain (FD) with priority customers, this machine can not be moved. Thus, the virtual router connected to C2 (i.e. R2) is pinned to the FD through a domain dependency. Other network functions do not present any dependency and can be mapped to arbitrary domains that improve QoS and QoE of service and customers. Finally, the third cache (i.e. C3) is a multi-provider function that can be shared with partner content providers. The CUSTOM specification of the presented hierarchical content delivery network service is shown in Fig. 13B. The network functions "C1", "C3", "R1", "R2", and "R3" are included in rule `VNF`, and the cache function "C2" is in `PNF`. Furthermore, rule `ADMDOMAIN` contains "FD" and rule `EN` has "EN1", "EN2", and "EN3". The shareable network function "C3" is preceded by the modifier symbol `!`. At last, we highlight that the specification of this network service has features all service topology categories: branched (structure); four functions in the largest path (length); ten elements – vertices – in the service topology (size); partially shared (sharing); administrative domain dependent (function dependency); and heterogeneous (heterogeneity).

6. Related work

The NFV paradigm is quite recent and is still going through a standardization process. Models that define VNFs and all that constitute and surround VNFs, including network service topologies are being continuously proposed and discussed. In terms of service topology, the creation of network services with the NFV paradigm should take into account the different features presented in Section 3. We claim that a useful service topology model should encompass all of those features and provide enough power of expression so that they can be effectively employed by service developers to design, validate, implement and maintain their specific services.

Different strategies have been proposed to model a service topology, from models based on graphs and formal languages to others based on

Table 4
Service topology models specification capabilities.

		Service topology					
		Structure	Length	Size	Sharing	Dependency	Heterogeneity
Models	Grammar [16]	●	●	●	○	○	○
	pACSR [15]	◑	◑	◑	○	○	○
	SG [17]	●	●	●	○	◑	○
	YANG [18]	●	●	●	○	○	○
	CUSTOM	●	●	●	●	●	●

high level markup languages. Project decisions related to the creation of topology models determine how much effort must be spent by services developers on the deployment process. While strategies lead to models that are executable (e.g., MiniNet emulator topology description - in Python [75]) others aim at improving the usability. The gap between the two can involve intermediate meta-models, that are easy to build and can be translated to verifiable formal models or even executable code.

In [16] authors define models for both the network substrate and service topology - the models are used to allow function placement which optimizes resource usage and latency. A context-free language is proposed to specify service topology features and restrictions. The elements of the context-free language are: individual network functions, a start point, and end points; optional order segments (i.e., set of functions that can be applied to the flows in any order); split segments (i.e., functions capable to split the incoming flows and send them through topology branches with different execution modules); and parallel modules (i.e., functions that split the incoming flows and send them on outgoing branches which have the same execution modules - such as load balancers).

In this context-free language a service topology as a quadruple $G = (\nu, \tau, \rho, \varsigma)$. Each language element represents a single part of the service topology. The ν element represents the group of non-terminal symbols with ten possibilities, of which $< modules >$ is the key non-terminal symbol after the start symbol. τ is the group of terminal symbols representing the functional elements of the service topology. In ρ the production rules are described, they consist of eleven rules that guide the specification of service topologies. Finally, ς represents the mandatory initial symbol $< start >$ of the grammar. The grammar can be easily be validated with traditional parsers, but it is also necessary to employ a high-level tool to ensure the semantic correctness of the resulting derivation tree.

With this model it is possible to identify categories such as structure, length, and size. On the other hand, the authors do not provide a way to declare explicit dependencies (neither of NFs nor of infrastructure), and only consider implicit dependencies related to the parallel execution branches. Furthermore, there is no means to describe function sharing or to declare that a specific function instance is physical, thus heterogeneous chains are not even mentioned.

The adaptation of existing languages with strong formal syntax and semantics is another option to define virtual services. Shin and others [15] expand the Algebra of Communicating Shared Resources (ACSR) - a formal description language that encompasses concepts such as processes, resources, events and priorities [76] - to define the so called pACSR (packet ACSR) language. The pACSR language allows the specification in a rich and detailed way of the packet forwarding process through a set of network functions using an operations syntax based on predefined predicates and functions.

ACSR allows process description by defining a set of operators, semantic rules and mappers (to construct and assign a meaning to a process) and applies algebraic laws for process manipulation. pACSR extends the ACSR by allowing packets to be forwarded along ports - which are used to describe VNFs. Two general operations are defined in pACSR: packet forwarding, which basically consist of values passing through ports, and packet processing, which correspond to values being used as process parameters. Some operations that are used in a pACSR specification include the “no action” (NIL), parallel execution (//), conditional process execution ($\rightarrow$), process restriction (/) and non-deterministic choices between two processes (+).

The pACSR language is capable of defining how packets are processed by a network service with fine granularity. However, other categories defined in the taxonomy such as structure and length cannot explicitly defined, but can be implicitly understood. On the other hand, service sharing features cannot be specified using this model as it does not employ VNF identifiers – IDs are used for the processes that run within VNFs. Dependencies are addressed using parallel processes, which allow the specification of implicit order dependencies. On the other hand, there is no way to specify some service policies or management decisions (e.g., either related to the infrastructure or the NFs themselves) directly in the model. Heterogeneity could be addressed using this model, as a process can be defined not only to execute VNFs, but also could also correspond to traditional middleboxes, but tags need to added to allow this differentiation.

Garay and others [17] propose a low-level straw-man model that relies on a graph-based system called Service Graph (SG) to specify a service topology. This model main concern is on embedding the SG in the physical infrastructure. The physical infrastructure is also defined through a graph, called Resource Graph (RG). The authors emphasize that the network service and the computational resources are closely related and both models should be compatible.

The SG straw-man model presents three main components: Network Functions (NF) which include deployment information and, if necessary, life cycle management scripts; Service Access Points (SAP) representing an element that attaches the service with other elements outside the scope of the model. Besides forwarding data to NFs, SAPs are also processing nodes capable of doing traffic analysis and classification; and Service Links (SL) which represents logical connections between the other elements. The SG straw-man model can be easily used to describe characteristics such as structure, size and length. Although data attached to the elements can specify infrastructure dependency, the model does not present any way to specify NF dependencies neither sharing.

Structured data modeling languages can also be employed for service topology specification. The authors of [18] propose a complete service specification model based on Yet Another Next Generation (YANG) language. This model follows the IETF SFC YANG Descriptors [77], it has the power to describe SFCs in detail, from standalone VNFs to so-

phisticated SFCs. The model is composed of an ordered list of service components. Each component, in turn, is a list of service function compositions. The compositions can represent different structure classes as follows. Best-binding compositions define partially ordered segments of VNFs. The all-binds composition indicates that a set of VNFs can be traversed in any order at any time (i.e., full mesh paths). The split composition defines branches in the service topology. Also, the function composition represents a single network function in the topology. Finally, links connect compositions to create a complete service topology. In this model there is no way to specify important characteristics such as function sharing, partial ordering dependencies, and network service heterogeneity.

Table 4 summarizes how the different service topology models allow the specification of the categories of the proposed taxonomy. A black filled circle indicates full support for the category; a half filled circle denotes an implicit or partial support of the category (e.g., some classes, sub-classes or special cases are not recognized). Last, an unfilled circle means that a category is not supported by the model.

Essentially, current service topology models are mostly concerned with network functions organization, thus easily supporting categories such as structure, length, and size. But, other information categories (e.g., sharing, dependencies, and heterogeneity) are not supported. The proposed CUSTOM model supports all categories, thereby avoiding the need of meta-models, descriptors and large-scale requests for service deployment processes (e.g., composition, split and mapping, placement, and scheduling).

7. Conclusion

Network Function Virtualization is a new paradigm that has already changed the network infrastructure landscape. By allowing the replacement of hardware middleboxes with Virtual Network Functions, NFV has unleashed a virtually unlimited number of novel possibilities and opportunities to parts of the network that were previously beyond the limits that most could reach. With the growing demand of network services to support the massive traffic generated by the network users and entities (from humans to machines and “things” - driven by the Internet of Things), the NFV paradigm encompasses methods and mechanisms to create, deploy and manage from individual network functions to complex services built up by composing multiple functions into virtualized network services.

We claim service topologies are an important aspect of services that have been neglected, even treated in conflicting ways by different entities. We hope our contribution represents a step on the way to fill this gap. Considering the various characteristics necessary to create specialized services for different scenarios, there are several different ways to view, evaluate and classify a virtualized network service topology. In this work, we proposed a unifying view of the concept of service topologies, which is formally defined relying on Graph Theory. A taxonomy is presented based on six categories which are also formally defined: Structure, Length, Size, Sharing, Function Dependency, and Heterogeneity. Some categories are decomposed in classes, sub-classes and special cases which we hope will provide the means for a richer definition and better understanding of service topologies. We also proposed a new service topology specification model, called CUSTOM. This model employs a context-free grammar for the creation of specification rules and provides the necessary capabilities to express all service topology categories defined in our taxonomy. Several case studies were presented to demonstrate the capabilities of the proposed model. We highlight that CUSTOM allows the specification of service topologies with features that collectively are not supported by any other model.

Finally, future work includes the definition of strategies to specify dynamic service topologies that take into account environment changes to readapt themselves. Furthermore, we note that the CUSTOM specification model can be easily extended to deal with the needs of specific systems and environments, such as Internet of Things (IoT) and Vehicular Ad-hoc Networks (VANET). Finally, integrating CUSTOM to actual NFV managers such as OpenStack Tacker [78] and NFV emulators such as NIEP [79] is also left as future work.

Declaration of Competing Interest

Authors declare that they have no conflict of interest.

CRediT authorship contribution statement

Vinicius Fulber-Garcia: Conceptualization, Funding acquisition, Formal analysis, Writing - original draft, Writing - review & editing. **Elias P. Duarte Jr.:** Conceptualization, Funding acquisition, Formal analysis, Writing - original draft, Writing - review & editing. **Alexandre Huff:** Conceptualization, Funding acquisition, Formal analysis, Writing - original draft, Writing - review & editing. **Carlos R.P. dos Santos:** Conceptualization, Funding acquisition, Formal analysis, Writing - original draft, Writing - review & editing.

References

[1] M. Handley, Why the internet only just works, BT Technol. J. 24 (3) (2006) 119–129.
[2] E. NFVISG, Network Functions Virtualization: White Paper, Technical Report, European Telecommunications Standards Institute, 2012.
[3] B. Han, V. Gopalakrishnan, L. Ji, S. Lee, Network function virtualization: challenges and opportunities for innovations, Commun. Mag. 53 (2) (2015) 90–97.
[4] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, V. Sekar, Making middleboxes someone else’s problem: network processing as a cloud service, Comput. Commun. Rev. 42 (4) (2012) 13–24.
[5] L. Bondan, M.F. Franco, L. Marcuzzo, G. Venancio, R.L. Santos, R.J. Pfitscher, E.J. Scheid, B. Stiller, F. De Turck, E.P. Duarte, A.E. Schaeffer-Filho, C.R.P. dos Santos, L.Z. Granville, FENDE: marketplace-based distribution, execution, and life cycle management of VNFs, Commun. Mag. 57 (1) (2019) 13–19.
[6] C. Schroth, T. Janner, Web 2.0 and SOA: converging concepts enabling the internet of services, IT Prof. 9 (3) (2007) 36–41.
[7] J. Cardoso, K. Voigt, M. Winkler, Service engineering for the internet of services, in: International Conference on Enterprise Information Systems, Springer International Publishing, 2008, pp. 15–27.
[8] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, F. Huici, ClickOS and the art of network function virtualization, in: Conference on Networked Systems Design and Implementation, USENIX Association, 2014, pp. 459–473.
[9] J. Hwang, K.K. Ramakrishnan, T. Wood, NetVM: high performance and flexible networking using virtualization on commodity platforms, Trans. Netw. Serv. Manag. 12 (1) (2015) 34–47.
[10] L. d. C. Marcuzzo, V.F. Garcia, V. Cunha, D. Corujo, J.P. Barraca, R.L. Aguiar, A.E. Schaeffer-Filho, L.Z. Granville, C.R.P. dos Santos, Click-on-OSv: a platform for running click-based middleboxes, in: Symposium on Integrated Network and Service Management, Institute of Electrical and Electronics Engineers, 2017, pp. 885–886.
[11] V.F. Garcia, L.C. Marcuzzo, G.V. Souza, L. Bondan, J.C. Nobre, A.E. Schaeffer-Filho, C.R.P. dos Santos, L.Z. Granville, E.P. Duarte Jr., An NSH-enabled architecture for virtualized network function platforms, in: International Conference on Advanced Information Networking and Applications, Springer International Publishing, 2019, pp. 376–387.
[12] V.F. Garcia, L. Marcuzzo, C. da, A. Huff, L.Z. Granville, A. Schaeffer-Filho, C. dos Santos, C.R.P. dos Santos, L.Z. Granville, E.P. Duarte, On the design of a flexible architecture for virtualized network function platforms, in: Global Communications Conference, Institute of Electrical and Electronics Engineers, 2019, pp. 1–6.
[13] P. Quinn, T. Nadeau, Problem Statement for Service Function Chaining - RFC 7498, Technical Report, Internet Engineering Task Force, 2015.
[14] E. NFVISG, Network Functions Virtualisation (NFV): Use Cases, Technical Report, European Telecommunications Standards Institute, 2013.
[15] M.K. Shin, Y. Choi, H.H. Kwak, S. Pack, M. Kang, J.Y. Choi, Verification for NFV-enabled network services, in: International Conference on Information and Communication Technology Convergence, Institute of Electrical and Electronics Engineers, 2015, pp. 810–815.
[16] S. Mehraghdam, M. Keller, H. Karl, Specifying and placing chains of virtual network functions, in: International Conference on Cloud Networking, Institute of Electrical and Electronics Engineers, 2014, pp. 7–13.
[17] J. Garay, J. Matias, J. Unzilla, E. Jacob, Service description in the NFV revolution: trends, challenges and a way forward, Commun. Mag. 54 (3) (2016) 68–74.
[18] S. Drxler, H. Karl, Specification, composition, and placement of network services with flexible structures, Int. J. Netw. Manag. 27 (2) (2017) e1963.
[19] E. ISG, Network Functions Virtualisation (NFV): Virtualisation Requirements, 2013,
[20] G. ETSI, Network Functions Virtualization (NFV); Architectural Framework, Technical Report, European Telecommunications Standards Institute, 2013.
[21] N. ETSI, Network Function Virtualisation (NFV): Terminology for Main Concepts in NFV, Technical Report, European Telecommunications Standards Institute, 2014.
[22] J. Halpern, C. Pignataro, Service Function Chaining (SFC) Architecture - RFC 7665, Technical Report, Internet Engineering Task Force, 2015.

[23] C.J. Bernardos, A. Rahman, J. Ziga, L.M. Contreras, P.A. Aranda, P. Lynch, Network Virtualization Research Challenges, Technical Report, Internet Engineering Task Force, 2018.

[24] R. Fernando, S. Mackie, D. Rao, B. Rijsman, M. Napierala, T. Morin, Service Function Chaining Using Virtual Networks with BGP VPNs, Technical Report, Internet Engineering Task Force, 2018.

[25] J.G. Herrera, J.F. Botero, Resource allocation in NFV: a comprehensive survey, Trans. Netw. Serv. Manag. 13 (3) (2016) 518–532.

[26] M.C. Luizelli, L.R. Bays, L.S. Buriol, M.P. Barcellos, L.P. Gaspary, Piecing together the NFV provisioning puzzle: efficient placement and chaining of virtual network functions, in: International Symposium on Integrated Network Management, Institute of Electrical and Electronics Engineers, 2015, pp. 98–106.

[27] M. Jalalitabar, Service Function Graph Design and Embedding in Next Generation Internet, Georgia State University, 2018 Ph.D. thesis.

[28] Z. Ye, X. Cao, J. Wang, H. Yu, C. Qiao, Joint topology design and mapping of service function chains for efficient, scalable, and reliable network functions virtualization, Network 30 (3) (2016) 81–87.

[29] W. Ding, W. Qi, J. Wang, B. Chen, OpenSCaaS: an open service chain as a service platform toward the integration of SDN and NFV, Network 29 (3) (2015) 30–35.

[30] T. Alharbi, A. Aljuhani, H. Liu, Holistic DDoS mitigation using NFV, in: Computing and Communication Workshop and Conference, Institute of Electrical and Electronics Engineers, 2017, pp. 1–4.

[31] F. Carpio, S. Dhahri, A. Jukan, VNF placement with replication for load balancing in NFV networks, in: International Conference on Communications, Institute of Electrical and Electronics Engineers, 2017, pp. 1–6.

[32] J.F. Riera, X. Hesselbach, M. Zotkiewicz, M. Szostak, J.F. Botero, Modelling the NFV forwarding graph for an optimal network service deployment, in: International Conference on Transparent Optical Networks, Institute of Electrical and Electronics Engineers, 2015, pp. 1–4.

[33] A. Vu, N. Dinh, Y. Kim, Modeling of service function chaining in network function virtualization environment, in: Korean Institute of Communication Sciences Conference, Institute of Electrical and Electronics Engineers, 2016, pp. 1100–1101.

[34] H. Baek, I. Jang, H. Ko, S. Pack, Order dependency-aware service function placement in service function chaining, in: International Conference on Information and Communication Technology Convergence, Institute of Electrical and Electronics Engineers, 2017, pp. 193–195.

[35] Z. Meng, J. Bi, H. Wang, C. Sun, H. Hu, CoCo: compact and optimized consolidation of modularized service function chains in NFV, in: International Conference on Communications, Institute of Electrical and Electronics Engineers, 2018, pp. 1–7.

[36] Q. Sun, P. Lu, W. Lu, Z. Zhu, Forecast-assisted NFV service chain deployment based on affiliation-aware vNF placement, in: Global Communications Conference, Institute of Electrical and Electronics Engineers, 2016, pp. 1–6.

[37] J. Matias, J. Garay, N. Toledo, J. Unzilla, E. Jacob, Toward an SDN-enabled NFV architecture, Commun. Mag. 53 (4) (2015) 187–193.

[38] V. Antonenko, R. Smeliansky, A. Plakunov, P. Mikheev, Cube: multi-user virtual function life-cycle orchestration technique, in: International Scientific and Technical Conference Modern Computer Network Technologies, Institute of Electrical and Electronics Engineers, 2018, pp. 1–8.

[39] R. Bruschi, F. Davoli, L. Galluccio, P. Lago, A. Lombardo, C. Lombardo, C. Rametta, G. Schembra, Virtualization of set-top-box devices in next generation SDN-NFV networks: the input project perspective, in: International Conference on Internet of Things, Data and Cloud Computing, Association for Computing Machinery, 2017, pp. 10:1–10:8.

[40] E.J. Scheid, C.C. Machado, R.L. dos Santos, A.E. Schaeffer-Filho, L.Z. Granville, Policy-based dynamic service chaining in network functions virtualization, in: International Symposium on Computers and Communication, Institute of Electrical and Electronics Engineers, 2016, pp. 340–345.

[41] B. Rashidi, C. Fung, CoFence: a collaborative DDoS defence using network function virtualization, in: International Conference on Network and Service Management, Institute of Electrical and Electronics Engineers, 2016, pp. 160–166.

[42] M. Jalalitabar, G. Luo, C. Kong, X. Cao, Service function graph design and mapping for NFV with priority dependence, in: Global Communications Conference, Institute of Electrical and Electronics Engineers, 2016, pp. 1–5.

[43] M. Jalalitabar, E. Guler, G. Luo, L. Tian, X. Cao, Dependence-aware service function chain design and mapping, in: Global Communications Conference, Institute of Electrical and Electronics Engineers, 2017, pp. 1–6.

[44] A.F. Ocampo, J. Gil-Herrera, P.H. Isolani, M.C. Neves, J.F. Botero, S. Latré, L. Zambenedetti, M.P. Barcellos, L.P. Gaspary, Optimal service function chain composition in network functions virtualization, in: International Conference on Autonomous Infrastructure, Management and Security, Springer International Publishing, 2017, pp. 62–76.

[45] X. Li, J. Rao, H. Zhang, A. Callard, Network slicing with elastic SFC, in: Vehicular Technology Conference, Institute of Electrical and Electronics Engineers, 2017, pp. 1–5.

[46] M.F. Bari, S.R. Chowdhury, R. Ahmed, R. Boutaba, On orchestrating virtual network functions, in: International Conference on Network and Service Management, Institute of Electrical and Electronics Engineers, 2015, pp. 50–56.

[47] H. Huang, S. Guo, J. Wu, J. Li, Service Chaining for Hybrid Network Function, Trans. Cloud Comput. 7 (4) (2019) 1082–1094.

[48] H. Moens, F.D. Turck, Customizable function chains: managing service chain variability in hybrid NFV networks, Trans. Netw. Serv. Manag. 13 (4) (2016) 711–724.

[49] C. Sun, J. Bi, Z. Zheng, H. Hu, HYPER: a hybrid high-performance framework for network function virtualization, J. Sel. Areas Commun. 35 (11) (2017) 2490–2500.

[50] W. Lee, Y. Choi, N. Kim, Study on virtual service chain for secure software-defined networking, Adv. Sci. Technol. Lett. 29 (13) (2013) 177–180.

[51] Y. Liu, Z. Guo, G. Shou, Y. Hu, To achieve a security service chain by integration of NFV and SDN, in: International Conference on Instrumentation Measurement, Computer, Communication and Control, Institute of Electrical and Electronics Engineers, 2016, pp. 974–977.

[52] O. TOSCA, TOSCA Simple Profile for Network Functions Virtualization (NFV), Technical Report, OASIS Committee Specification, 2015.

[53] P. Vizarreta, M. Condoluci, C.M. Machuca, T. Mahmoodi, W. Kellerer, QoS-driven function placement reducing expenditures in NFV deployments, in: International Conference on Communications, Institute of Electrical and Electronics Engineers, 2017, pp. 1–7.

[54] M.E. Lesk, E. Schmidt, Lex: A Lexical Analyzer Generator, 2020, http://dinosaur.compilertools.net/lex/index.html.

[55] S.C. Johnson, M. Hill, Yacc: Yet Another Compiler-Compiler, 2020, http://dinosaur.compilertools.net/yacc/index.html.

[56] N.R. Group, Natural Language Toolkit, 2020, http://www.nltk.org.

[57] T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, F. Yergeau, Extensible markup language (XML) 1.0, Technical Report, W3C Recommendation, 2000.

[58] O. Ben-Kiki, C. Evans, B. Ingerson, YAML Ain't Markup Language (YAML) version 1.1, Technical Report, YAML Recommendation, 2005.

[59] N.I. ETSI, NFV Management and Orchestration: VNF Descriptor and Packaging Specification, Technical Report, European Telecommunications Standards Institute, 2019.

[60] H. Koumaras, C. Sakkas, M.A. Kourtis, C. Xilouris, V. Koumaras, G. Gardikis, Enabling agile video transcoding over SDN/NFV-enabled networks, in: International Conference on Telecommunications and Multimedia, Institute of Electrical and Electronics Engineers, 2016, pp. 1–5.

[61] L. Durante, L. Seno, F. Valenza, A. Valenzano, A model for the analysis of security policies in service function chains, in: Conference on Network Softwarization, Institute of Electrical and Electronics Engineers, 2017, pp. 1–6.

[62] T. Alharbi, A. Aljuhani, H. Liu, C. Hu, Smart and lightweight DDoS detection using NFV, in: International Conference on Compute and Data Analysis, Association for Computing Machinery, 2017, pp. 220–227.

[63] S. Palkar, C. Lan, S. Han, K. Jang, A. Panda, S. Ratnasamy, L. Rizzo, S. Shenker, E2: a framework for NFV applications, in: Symposium on Operating Systems Principles, Association for Computing Machinery, 2015, pp. 121–136.

[64] G. Liu, Y. Ren, M. Yurchenko, K.K. Ramakrishnan, T. Wood, Microboxes: high performance NFV with customizable, asynchronous TCP stacks and dynamic subscriptions, in: Conference of the ACM Special Interest Group on Data Communication, Association for Computing Machinery, 2018, pp. 504–517.

[65] C. Sun, J. Bi, Z. Zheng, H. Yu, H. Hu, NFP: enabling network function parallelism in NFV, in: Conference of the ACM Special Interest Group on Data Communication, Association for Computing Machinery, 2017, pp. 43–56.

[66] Y. Zhang, B. Anwer, V. Gopalakrishnan, B. Han, J. Reich, A. Shaikh, Z. Zhang, Parabox: exploiting parallelism for virtual network functions in service chaining, in: Symposium on SDN Research, Association for Computing Machinery, 2017, pp. 143–149.

[67] J. Gil-Herrera, J.F. Botero, A scalable metaheuristic for service function chain composition, in: Latin-American Conference on Communications, Institute of Electrical and Electronics Engineers, 2017, pp. 1–6.

[68] Y. Wang, Z. Li, G. Xie, K. Salamatian, Enabling automatic composition and verification of service function chain, in: International Symposium on Quality of Service, Institute of Electrical and Electronics Engineers, 2017, pp. 1–5.

[69] V. Fulber-Garcia, M.C. Luizelli, C.R.P. dos Santos, E.P. Duarte, CUSCO: a customizable solution for NFV composition, in: International Conference on Advanced Information Networking and Applications, Springer International Publishing, 2020, pp. 204–216.

[70] L. Bondan, C.R.P. dos Santos, L.Z. Granville, Management requirements for ClickOS-based network function virtualization, in: International Conference on Network and Service Management, Institute of Electrical and Electronics Engineers, 2014, pp. 447–450.

[71] R. Yu, G. Xue, V.T. Kilari, X. Zhang, Network function virtualization in the multi–tenant cloud, Network 29 (3) (2015) 42–47.

[72] M.A. Cusumano, Cloud computing and saas as new computing platforms, Communications 53 (4) (2010) 27–29.

[73] A.M. Medhat, G. Carella, C. Luck, M. Corici, T. Magedanz, Near optimal service function path instantiation in a multi-datacenter environment, in: International Conference on Network and Service Management, Institute of Electrical and Electronics Engineers, 2015, pp. 336–341.

[74] Z. Bronstein, E. Shraga, NFV virtualisation of the home environment, in: Consumer Communications and Networking Conference, Institute of Electrical and Electronics Engineers, 2014, pp. 899–904.

[75] B. Lantz, B. Heller, N. McKeown, A network in a laptop: rapid prototyping for software-defined networks, in: SIGCOMM Workshop on Hot Topics in Networks, Association for Computing Machinery, 2010, pp. 19:1–19:6.

[76] P. Brmond-Grgoire, I. Lee, A process algebra of communicating shared resources with dense time and priorities, Theor. Comput. Sci. 189 (1) (1997) 179–219.

[77] R. Penno, P. Quinn, D. Zhou, J. Li, Yang Data Model for Service Function Chaining, Technical Report, Internet Engineering Task Force, 2016.

[78] O. Foundation, Tacker: OpenStack NFV Orchestration, 2020, https://wiki.openstack.org/wiki/Tacker.

[79] T. Tavares, L. Marcuzzo, V.F. Garcia, G. Venncio, M. Franco, L. Bondan, F. De Turk, L. Granville, E. Duarte, C. Santos, A. Schaeffer-filho, NIEP: NFV infrastructure emulation platform, in: International Conference on Advanced Information Networking and Applications, Institute of Electrical and Electronics Engineers, 2018, pp. 173–180.

Vinícius Fülber Garcia is a Ph.D. student in Computer Science at the Departament of Informatics of the Federal University of Paraná (UFPR - Brazil) under the supervision of Prof. Dr. Elias Procópio Duarte Júnior. He holds a Computer Science degree from Federal University of Santa Maria (UFSM - Brazil) and a Master degree in Computer Science from UFSM Post-Graduate Program in Computer Science. His research interests include network functions virtualization, service function chaining, compression algorithms, and information theory.

Elias Procopio Duarte Jr. is a Full Professor at Federal University of Parana, Curitiba, Brazil, where he is the leader of the Computer Networks and Distributed Systems Lab (LaRSis). His research interests include Computer Networks and Distributed Systems, their Dependability, Management, and Algorithms. He has published more than 200 peer-reviewer papers and has supervised more than 130 students both on the graduate and undergraduate levels. Prof. Duarte is currently Associate Editor of the IEEE Transactions on Dependable and Secure Computing, and has served as chair of more than 20 conferences and workshops in his fields of interest. He received a Ph.D. degree in Computer Science from Tokyo Institute of Technology, Japan, 1997, M.Sc. degree in Telecommunications from the Polytechnical University of Madrid, Spain, 1991, and both B.Sc. and M.Sc. degrees in Computer Science from Federal University of Minas Gerais, Brazil, 1987 and 1991, respectively. He chaired the Special Interest Group on Fault Tolerant Computing of the Brazilian Computing Society (2005–2007); the Graduate Program in Computer Science of UFPR (2006–2008); and the Brazilian National Laboratory on Computer Networks (2012–2016). He is a member of the Brazilian Computing Society and a Senior Member of the IEEE.

Alexandre Huff is a Ph.D. student in Informatics at the Federal University of Parana, holds a Master's Degree in Computer Science from the State University of Maringa (2010), and a Degree in Computer Technology from the Universidade Paranaense (2004). He is an Adjunct Professor of the Superior Magisterium at the Federal Technological University of Parana, Toledo, Brazil. His research topics include Distributed and Fault-Tolerant Systems, Network Function Virtualization, and Computer Networks.

Carlos Raniery Paula dos Santos is Adjunct Professor of Computer Science at the Department of Applied Computing of the Federal University of Santa Maria (UFSM), Brazil. He holds Ph.D. (2013) and M.Sc. (2008) degrees in Computer Science, both received from the Federal University of Rio Grande do Sul (UFRGS), where he was also Postdoctoral Research Fellow from October 2013 to September 2014. From May 2010 to April 2011 he was a visiting researcher at the IBM T.J. Watson Research Center - Hawthorne, where he developed projects on IT Service Management and Security Management. His research interests focus on design and management of Future Networks and Technologies, including aspects such as network virtualization, quality of service management, network programmability, and security management.