VNF-based network service consistent reconfiguration in multi-domain federations: A distributed approach

Josué Castañeda Cisneros, Saúl Eduardo Pomares Hernández, Sami Yangui, Julio Pérez Sansalvador, Lil Rodríguez Henríquez, Khalil Drira

VNF-based network service consistent reconfiguration in multi-domain federations: A distributed approach☆

Josué Castañeda [a,*], Saul E. Pomares Hernandez [a,b], Sami Yangui [a], Julio C. Pérez Sansalvador [c], Lil M. Rodríguez Henríquez [c], Khalil Drira [a]

[a] LAAS-CNRS, Université de Toulouse, France. INSA, F31400, Toulouse, France
[b] INAOE, 72840, Santa María Tonantzintla, Puebla, Mexico
[c] INAOE-Cátedra CONACyT, 72840, Santa María Tonantzintla, Puebla, Mexico

Keywords:
Consistent reconfiguration
Shared network services
Multi-domain orchestration
Network function virtualization (NFV)
Virtual network functions (VNF)

A B S T R A C T

The Network Function Virtualization (NFV) virtualizes the network appliances — such as routers and firewalls — with software running on commercial off-the-shelf servers. In NFV specification, Network Services (NS) are composed of multiple Virtual Network Functions (VNF) enabling elastic and finer lifecycle management operations such as scaling. For adapting resources to the VNF-based network services evolving context, these operations require to execute, on the fly, a reconfiguration plan supervised by a central orchestrator. This is the single-domain reconfiguration, where the orchestrator has global up-to-date information, ensuring the correct execution of the lifecycle management operations. Moreover, in practice, NS can be implemented by composing VNFs in a cross-domain schema called a multi-domain federation. In this case, the reconfiguration is morechallenging since there are multiple orchestrators, one by domain, that manage collaboratively shared network services. Sharing network services creates functional and non-functional dependencies among these services that must be considered for ensuring the consistency of the lifecycle operations. The consistent reconfiguration of shared network services in distributed multi-domain federations is called the NFV Dependent Reconfiguration problem. However, despite being identified as an important challenge by the NFV community, no related so- lution has yet been proposed. In this paper, we focus on the NFV Dependent Reconfiguration problem. We introduce a distributed approach to guarantee consistency during dependent reconfiguration. The approach is composed of a distributed multi-domain model that establishes the interactions among the federation's entities, and a causally-consistent distributed orchestration algorithm based on such a model. We verify the algorithm viability using, as a case study, the scaling of shared VNF-based network services as defined by the current NFV standard architecture. To the best of our knowledge, the proposed distributed orchestration approach is the first that supports the consistent execution of dependent reconfiguration operations for VNF-based network services.

1. Introduction

Network Function Virtualization (NFV) is a concept that decouples network services from underlying hardware (Mijumbi et al., 2016). Such decoupling enhances flexibility during the deployment of network services, by creating them with multiple Virtual Network Functions (VNFs) which offer functionalities such as routing or deep packet inspecting (Mijumbi et al., 2016). Moreover, the services' life cycle management, which changes dynamically over time, becomes elastic as VNFs can be replaced, moved, or scaled on the fly. Traditionally, this reconfiguration is done by a single central orchestrator.

Single-domain orchestration ensures the correct execution of life cycle management operations, such as scaling, as the orchestrator has the global up-to-date information of all the domain's components (Etsi, 2019a). However, because of the rising data traffic consumption (Forecast, 2019), the high cost to update services (Antonopoulos, 2020), and the flattened revenue for operators (Cano et al., 2017), service providers share services. Sharing services, in a cross-domain schema,

☆ This document is the results of the research project funded by the Consejo Nacional de Ciencia y Tecnologia (CONACYT) and LAAS-CNRS.
* Corresponding author.
E-mail addresses: jcastane@laas.fr (J. Castañeda), spomares@inaoep.mx (S.E. Pomares Hernández), yangui@laas.fr (S. Yangui), jcp.sansalvador@inaoep.mx (J.C. Pérez Sansalvador), lmrodriguez@inaoep.mx (L.M. Rodríguez Henríquez), drira@laas.fr (K. Drira).

allows providers to increase their revenue and lower expenditures for both capital and operational expenses (Yousaf et al., 2019). Thus, the operators are willing to share resources, for example, by sharing VNFs (Malandrino et al., 2019). To enable such joint deployments, providers transition from single to multi-domain environments. In a multi-domain environment, each domain has its orchestrator that manages all resources in its domain (Etsi, 2014). By exchanging messages, orchestrators coordinate among them and create a distributed multi-domain federation (Etsi, 2019b).

Distributed multi-domain federations improve the NFV architecture, introduced by the European Telecommunications Standards Institute (ETSI), by sharing resources among service providers. They extend the providers' capacities (i.e. market share) despite the limited resources of each service provider in the federation. Orchestrators deploy services in multiple domains. The services adapt to new users' requirements (Taleb et al., 2019; Saraiva de Sousa et al., 2019). Federations also offer resiliency properties, such as scalability, increased performance, and robustness against failures (Katsalis et al., 2016a). Thus, in distributed multi-domain federations, VNF-based network services are shared among many orchestrators and their life cycle is no longer managed by a single orchestrator.

In a Federation, unlike single-domain orchestration, many orchestrators manage lifecycle tasks of VNFs and network services. These tasks include instantiating, reconfiguring, and monitoring the network services. Yet, federations bring new challenges for decentralized and asynchronous operations (Katsalis et al., 2016a). Thus, the orchestrators coordinate preventing unwanted side-effects (e.g. partial service failures). The orchestrators try to guarantee functional and non-functional properties of the services by reconfiguring shared services (Cisneros et al., 2020). This reconfiguration must be consistent among all orchestrators.

Ensuring consistency in shared network services entails having the same up-to-date information of each orchestrator before and after they execute a life cycle management operation. Before executing any operation, orchestrators must coordinate themselves to prevent unwanted behavior of network services. The orchestrators generate grants to notify and validate a reconfiguration operation when a network service has external dependencies. Each grant's recipient verifies the internal consistency in its domain by checking if the management operation affects other network services.

The current specification of NFV federations claims to ensure no undesired effect occurs while reconfiguring takes place, but no order or timing constraints exist between two consecutive grants (Etsi, 2018a). Thus, grants can be executed non-deterministically, in any order, without satisfying the service's external dependencies. Such execution could bring inconsistencies and lead to greater costs to the provider. For example, during the scaling of a shared service managed by many or- chestrators, the service could be redundantly scaled and could also have deprecated connections. Thus, it is necessary to enforce a correct grant ordering for shared service reconfiguration to prevent inconsistencies induced by unsatisfied shared services' dependencies. Such correct grantordering is the founding principle of the distributed approach of man- aging the consistent dependent reconfiguration problem for VNF-based network services in distributed multi-domain federations.

The state-of-the-art for service reconfiguration in NFV focuses on migrating VNFs while optimizing resources such as energy, time, and latency in a single domain (Eramo et al., 2017; Yang et al., 2018; Wang et al., 2018). Other works focus on the network update problem (Shin et al., 2015). These works enforce consistent updates by satisfying invariants; yet, they also focus on single domains where no coordination is required and concentrate at the network level. Some Service Oriented Architecture solutions try to address the reconfiguration challenges for VNF-based network services; however, they are not fully compatible since, unlike web services that are remotely called, VNFs need to be instantiated. As far as we know, solutions at the service-level, that consider coordination among orchestrators address only the problems of placement and chaining tasks in NFV (Pham and Chu, 2019; Li et al., 2018; Ghaznavi et al., 2017; Sun et al., 2018). All previous solutions are unsuitable for the dependent reconfiguration because they assume a static configuration for VNF-based network services. For dynamic tasks in multi-domain environment, only the current ETSI NFV standard specifies an algorithm to scale shared network services (Etsi, 2019b). However, it does not consider non-deterministic network conditions and consistency issues that arise when dealing with dependent services.

In this work, we consider on the fly dynamic dependent reconfiguration of network services. We consider non-deterministic networks and dependencies among shared services. Based on such dependencies, we coordinate reconfiguration by ordering grants sent by orchestrators. Our contributions are as follows:

- A Distributed NFV Multi-Domain Orchestration model is introduced that establishes the interactions among ETSI defined components in a federation to support the life-cycle management of VNF-based shared network services (Section 6.1).
- An inconsistent pattern for the NFV dependent reconfiguration is identified and formally defined from a temporal and logical perspective (Section 6.3, 6.4)
- A Causally-consistent orchestration algorithm is presented based on the proposed orchestration model to prevent inconsistencies that may occur when VNF-based network services have external de- pendencies (Section 7).

The viability of the approach is shown via simulations using the scaling of VNF-based shared network services as the target operation.For this, we measured the inconsistencies, time to reconfigure, and message overhead and compared them to the current reconfiguration algorithm (Section 8).

The paper is organized as follows: The preliminaries are presented in Section. The related work is discussed in Section 3. The problem and a use case are shown in Section 4. Section 5 contains the system model. We formalize the inconsistencies while reconfiguring network services in Section 6. Our solution is presented in Section 7. Evaluation and results are presented and discussed in Section 8. Finally, our conclusions and insights for future work are presented in Section 9.

2. Preliminaries

In this section, we present the concepts required for the dependent reconfiguration of VNF-based network services in distributed multi-domain federations. For the rest of the paper, we use the terms VNF- based network services and network services interchangeably, accord- ing to ETSI.

2.1. *VNF-based network services*

Network services are the building units for next-generation network applications. Under the NFV concept, multiple VNFs compose a network service according to one or more forwarding graphs (Etsi, 2018b). Network services belong to distinct classes according to their users via the service's access points. *Dedicated* network services belong to a single domain and only have VNFs as internal dependencies. *Composite* ones belong to at least two different domains and have external dependencies as network services (Etsi, 2018c). The external dependencies are also called *nested services* as the provider combines them to create a composite service (Etsi, 2018d). External dependencies are managed by multiple administrative domains, unlike internal domains that belong to a single domain.

Fig. 1 shows a *composite* service (C) that has two dependencies (A, B). In this example, the two *dedicated* network services that belong to the *composite* network service are offered by different domains. The service C has internal and external connection points to deliver features to consumers. Detailed information about the *dedicated* network services A

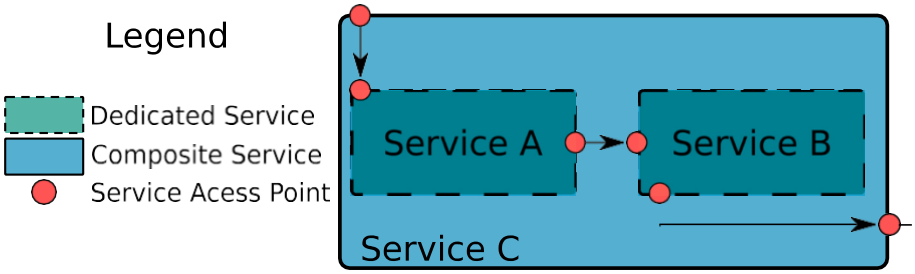

Fig. 1. Example of a *composite* and two *dedicated* network services.

and *B* is unavailable to the orchestrator that manages of service *C*, such as topology, lifecycle management policies (e.g. scaling rules) and communication endpoints. This is because the services *A* and *B* are external dependencies managed by other administrative domains. The limited knowledge of orchestrators outside their administrative domain enables providers to share their resources without compromising the orchestrators' privacy or autonomy. For example, orchestrators can setup complex service function chains without access to detailed infor- mation of other orchestrators (Liu et al., 2020).

2.2. Distributed multi-domain orchestration

Harnessing the benefits of network services requires control to support their lifecycle. Under the NFV paradigm, the Network Functions Virtualization Orchestrator − simply orchestrator − manages the network service's lifecycle tasks (i.e. building, instantiating, executing, reconfiguring, and monitoring) (Etsi, 2018c). The orchestrator also administers all the systems and networks operated by a single organization (Etsi, 2019a), it also handles the Virtual Infrastructure Manager and VNF Manager to support all the VNFs that compose the services. Multi-domain orchestration extends the capacities of the single orches- trator by offering network services within the same organization and facilitating these services to another network operator (Etsi, 2019b). Three types of architectures support the multi-domain orchestration as defined by the ETSI Standard: Centralized, distributed, and hybrid (Rosa et al., 2015; Katsalis et al., 2016a). Centralized solutions establish a global orchestrator that coordinates other orchestrators via vertical calls. Distributed architectures lack a central coordinator and orches- trators communicate to support the network services' lifecycle. Hybrid ones create hierarchies where orchestrators coordinate both horizon-tally and vertically. All the previous architectures enable a federation.

2.3. Distributed multi-domain federations

A Federation is a collective group of service providers who share resources to support complex network services (Baranda et al., 2020). This reduces the costs of each individual provider and extends the capacities despite the limited resources of each provider. Thus, orchestrators access the network services of different providers by negotiating the limited resources among them. This creates *shared* network services that can be used by multiple services in the federation. *Composite* services in such federations can be *shared* services and also have these type of services as external dependencies. Thus, for the rest of the paper, we use the terms *composite* and *shared* services interchangeably. Due to shared services in a distributed multi-domain federation, the orchestrators coordinate to overcome the limited knowledge of each participant. On the one hand, these federations offer flexible, scalable, and extendable network services; on the other, they add overhead to the service's life-cycle management. We consider closed federations that reject new participants to enter; thus, lowering the overhead, but keeping the benefits of a federation.

2.4. Life-cycle management of network services in multi-domain federations

Network services must meet their required Service Level Agreement despite changes in the network. Such a service reconfiguration triggers, by energy consumption, fault tolerance, higher revenues, or improve-ment of the QoS (Kim et al., 2016; Eramo et al., 2017, 2019; Yang et al., 2018; Wang et al., 2018; Liu et al., 2017). The ETSI standard identifies different tasks at the service-level, such as scaling, migrating, and restoring a network service to meet the service level agreement re- quirements of a *composite* service (Etsi, 2019c). In multi-domain feder- ations, ETSI defines a special communication reference point between orchestrators, called the orchestrator to orchestrator *or-or* (ETSI, 2020). This reference point is shown in Fig. 2 where the orchestrators can setup a network service for a content delivery network by chaining four VNFs:
(1) A translator (TRA), (2) streamer (ST), (3) encoder (ENC), and finally a (4) decoder (DEC). This four VNFs are managed by different orchestrators as shown in Fig. 2. For example, the *NFVO−C* manages the DEC VNF; while the *NFVO-A* manages both TRA and ST VNFs, respectively.
The orchestrators coordinate over the *or-or* point, used for the exchanges between orchestrators in different administrative domains. This reference point enables interfaces to support complex multi-domain tasks via grant messages (Etsi, 2018a). According to the ETSI standard, the following tasks in multi-domain environments require coordination by sending grants over the *or-or* reference point:

- **Scale Network Service**: Increase/decrease internal and/or external dependencies. This work focuses on this task for reconfiguration.
- Terminate Network Service.
- Heal Network Service: Recover a service after an error.
- Subscription/Notification.

In this work, we consider closed federations where a fixed number of trusted orchestrators expose connections points of their shared network services. Orchestrators communicate with each other via messages since distributed multi-domain federations lack global references. Thus, there is a flexible hierarchy according to each service that enforces the use of coordination among the orchestrators because of limited knowledge. Fig. 2 shows an example. The orchestrator *NFVO−C* plays the role of a consumer and provider of network services. The VNF Managers in each administrative domain interact with their orchestrator; yet, the orchestrators are not aware of the constituent VNF instances of the *shared* service instance and do not interact with the VNF Managers of other administrative domains. This is the case for orchestrator *NFVO−C* and the VNF managers of *NFVO-A* and *NFVO−B*, respectively.

3. Related work

We discuss the relevant work for the dependent reconfiguration task. First, we present single-domain reconfiguration algorithms, focusing on the scale of VNFs. Then, we present reconfiguration algorithms for multi-domain environments and highlight the drawbacks of the current state-of-the art solutions. Finally, we briefly describe how our proposed

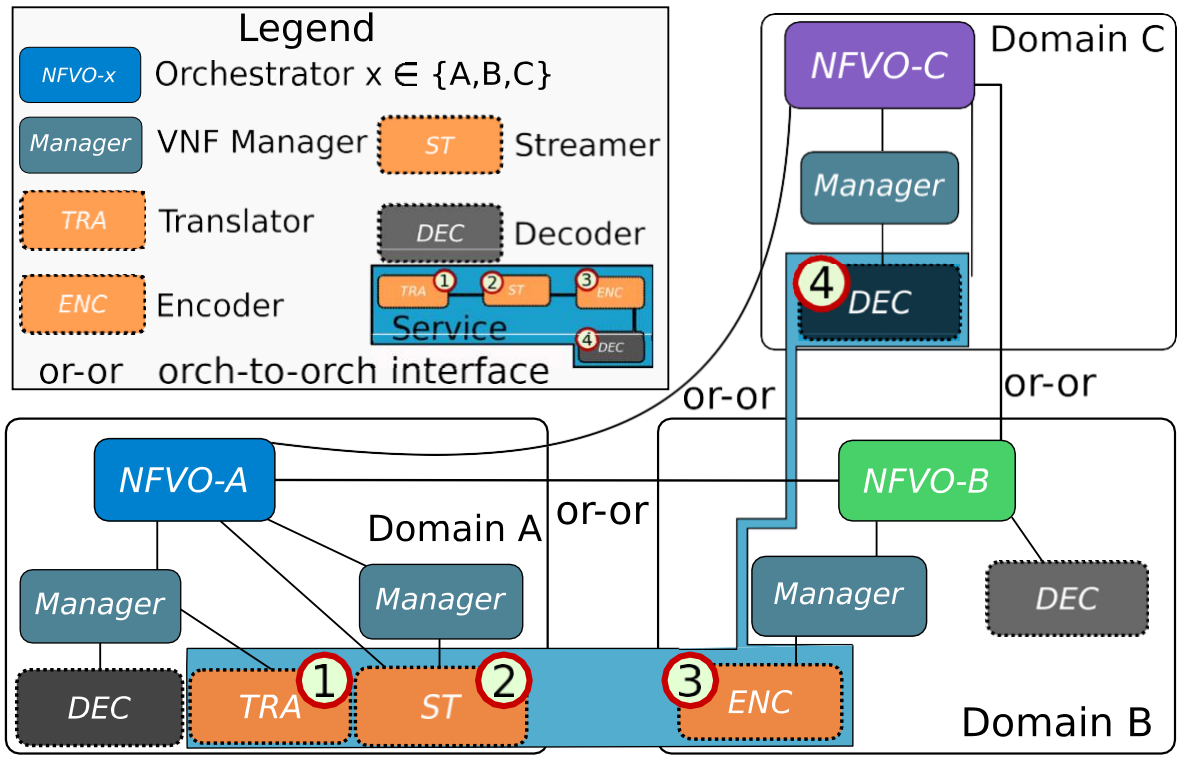

Fig. 2. Network Services federation provided using multiple administrative domains. An orchestrator can be both a provider and a consumer of other services. A VNF-based network service is created by chaining the VNFs in the order specified by the numbers.

model and algorithm extend the state-of-the art for VNF-based service reconfiguration in multi-domain environments. Fig. 3 shows how we organized the related work. Our work is positioned in the colored branch with the bold font for multi-domain VNF-based shared network services.

3.1. *Reconfiguration of VNF-based network services in single-domain environments*

The reconfiguration of network resources focuses on three tasks of

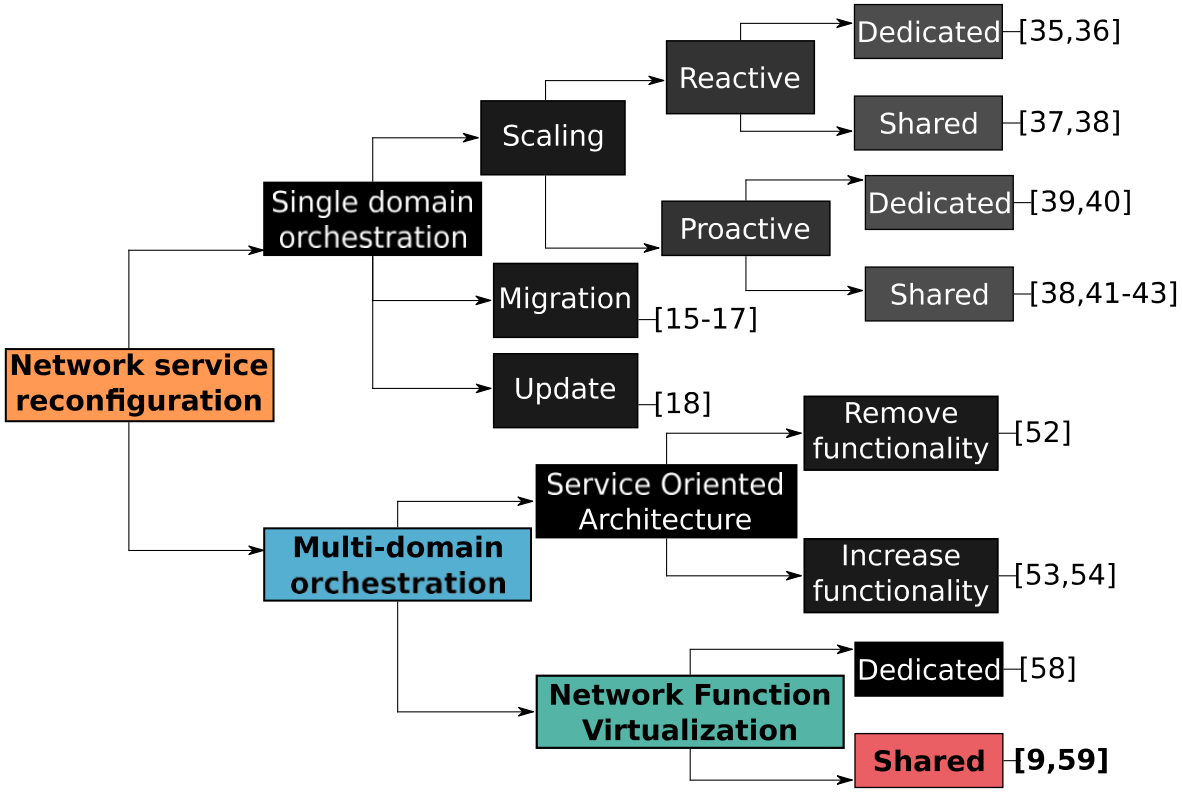

Fig. 3. Taxonomy for network service reconfiguration. Our work is positioned in the lower branch with NFV multi-domain shared network services.

the life cycle management of network services. The orchestratorsexecute tasks such as migrating, updating, and scaling VNFs. The problem of migration focuses on the new placement of a VNF while optimizing resources such as energy, time, and latency in a single domain (Eramo et al., 2017; Yang et al., 2018; Wang et al., 2018). Other works focus on the network update problem that changes a VNF descriptor to include more functionalities (Shin et al., 2015). Scaling with NFV allows operators to resize network services at runtime to handle load surges with performance guarantees (Adamuz-Hinojosaet al., 2018). In this work, we focus on the VNF scaling as it is the closest related to our work. We classified the related work for scaling shared resources in a single domain as either reactive (i.e. monitor the traffic) or proactive (i.e. predict future traffic).

3.1.1. *Single-domain reactive works*

Most of the works in the literature consider dedicated services that belong to a single network service despite having VNFs in distributed data centers. Auto-scaling orchestration mechanisms have been proposed to minimize the cost of scaling a service while meeting end-to-end delay (Nadjaran Toosi et al., 2019; Duan et al., 2017). These works propose heuristic algorithms with monitoring algorithms to scale the VNFs. As far as we know, only two works in the literature consider shared resources for a single domain. The first work proposed a latency-aware mechanism (Sarrigiannis et al., 2020). It offers a scheduling algorithm for the initial placement and reallocation of VNFs. The second work proposed a VNF scaling on-line algorithm that considers all the costs associated with provisioning network resources (Tong et al., 2020). It achieves an upper-bounded competitive ratio. The major drawback of reactive works is the negative impact of the reconfiguration. Since they only reconfigure services when they capture a problem, the service must be halted or temporally degraded while the changes take place. Proactive scaling mechanisms were proposed to mitigate the negative impact of reconfiguration.

3.1.2. *Single-domain proactive works*

Proactive works predict future traffic and try to scale network services or VNFs to address these changes. While some works were proposed for dedicated services (Jia et al., 2018; Xu, 2020), we focus on shared services. For shared VNF-based network services, several works have been published. The first work proposed a traffic model based on Gated Recurrent Units (Tong et al., 2020). After the prediction, many independent agents explore the network to get optimal placement. Another work proposed a log-linear Poisson auto-regression model to forecast the traffic (Hu et al., 2020). Based on the model's output, an evolution-based algorithm scales automatically the VNFs. Similarly, an adaptive scaling mechanism based on Q-learning and Gaussian Pro- cesses to train a single agent was proposed (Arteaga et al., 2017). The agent learns the scaling policy despite traffic variations. Another inter- esting work that extends previous works by allowing tenants to refuse scaling was proposed (Rahman et al., 2020). The method offers a negotiation phase where, based on the predicted traffic and goals for each tenant, under the same domain, the VNFs are scaled.

All the previous approaches rely on a single administrative domain under a global orchestrator. The advantages of such deployment are ease of use and simple life cycle management. However, this approach has drawbacks, such as scalability, security, and limited flexibility. In practice, multiple administrative domains want to keep autonomy froma single orchestrator. Decentralized solutions face the shortcomings ofthe global deployments (Chen et al., 2010).

3.2. *Reconfiguration of VNF-based network services in multi-domain environments*

Decentralized approaches achieve better performance since the orchestrators distribute traffic among participants (Nanda et al., 2004). Since NFV falls into the Definition of IT services at large (Katsalis et al., 2016b), VNFs can be provisioned as any other type of services. Service-Oriented Architecture (SOA) principles (e.g., service abstrac- tion, discoverability, and composability) ensure the viability of an ecosystem of network services, such in multi-domain environments, concerning the NFV paradigm (Yi et al., 2018). Thus, first, we present the SOA reconfiguration solutions. Then, we describe why these solu-tions do not fully align with the NFV paradigm. Finally, we describe the NFV reconfiguration solutions for VNF-based network services.

3.2.1. *Service-oriented architecture reconfiguration for network services*

Before NFV, in the domain of web services, choreographies have been proposed to handle the reconfiguration of a service. A service choreography achieves service composition without centralized control through a protocol via observable events (Leite et al., 2013). The collaborative protocol, encoded in the choreography, ensures correct-ness properties such as deadlock prevention, conformance to message specification, and realizability (Kattepur et al., 2013). Some works propose a coordination protocol to reconfigure services where a shared global state is maintained without a central orchestrator (Kazhamiakin et al., 2006; Salaün and Roohi, 2009). Works can either remove faulty components by choosing the optimal and correct candidates from a limited pool of options (Boudries et al., 2019), or bring new function- alities on the fly and add them to the existing chaining (Moo-Mena and Drira, 2007; Hnětynka et al., 2006). Previous works adapt network services to changes in the environment; however, they exclude consis- tency issues brought by dependencies among the services that arise while reconfiguring services like in NFV.

The VNF life-cycle task was inspired by SOA (Yangui et al., 2016); however, discrepancies between web services and VNFs make SOA solutions inappropriate to address VNF-based network services tasks (el houda Nouar et al., 2021). For instance, unlike web services, VNFs are not remotely invoked but must be downloaded and executed locally in different administrative domains. The VNFs, and by extension VNF-based network services, include technical details such as the sup- ported technologies, the configuration settings, and their operation management for each task. Moreover, the service choreography in SOA lacks elements present in NFV such as the orchestrator who has a well-defined workflow for the life cycle of network services (Etsi, 2019a). Another challenge present in the NFV context is the heteroge- neity of VNFs (Bouras et al., 2017), unlike web services that only consider input and output parameters. Finally, since administrative domains have different capabilities (e.g. CPU, RAM, bandwidth) the reconfiguration operation for VNF-based network services mustconsider such resources to ensure functional and non-functional re- quirements (Xu, 2020). Thus, solutions for reconfiguring VNF-based network services with a focus on consistency need to be explored in the NFV context, considering both internal (hidden) and external (observable) events.

3.2.2. *Network Function Virtualization reconfiguration for VNF-based services*

NFV reconfiguration for network services under multi-domain considers federations that share resources by negotiating among many participants (Pham and Chu, 2019). Some works consider multiple administrative domains but the services are still dedicated. Deep learning was proposed in a proactive orchestration algorithm to predict traffic and scale VNF instances (Subramanya and Riggio, 2021). Unlike in single domain orchestration that only considers dedicated network services, multi-domain federations generally create composite servicesby sharing resources (Etsi, 2018c). As far as we know, only a few works have considered the scaling of composite network services undermulti-domain orchestration. The ETSI NFV standard specifies an algo- rithm to scale composite network services (Etsi, 2019b). The algorithm proposes a workflow based on grants to coordinate orchestrators. A custom platform was deployed using the ETSI standard to scale com- posite services (Baranda et al., 2020). The previous works handle the

composite scaling of services in ideal conditions (e.g. messages are ordered, no latency in transmission, zero messages lost). However, in real conditions, the non-deterministic conditions of the network and limited information of each orchestration bring new challenges, such as preventing inconsistencies (Vaquero et al., 2019). Preventing inconsistencies is a desired property when reconfiguring VNF-based network services as it prevents unwanted effects from rippling across the federation. However, currently, there are no formal models to identify and prevent inconsistencies while reconfiguring composite VNF-based services.

3.3. Synthesis

The review of the relevant literature shows that many works consider a centralized solution. Others, non-centralized, consider approaches not completely compatible for NFV. Some solutions do not consider sharing services. These limitations reduce the applicability of solutions as providers want to: have autonomy and privacy for their domains, achieve local interoperability, and share resources to extend their market share. The ETSI standard orchestration algorithm addresses some of these limitations; however, it considers ideal conditions to reconfigure network services without timing constraints. Moreover, currently no grant message exchange pattern has been identified to prevent in- consistencies. In this paper, we extend the state-of-the art by proposing a distributed multi-domain orchestration model that, unlike the state-of- the art, considers non-deterministic network conditions where services can have multiple dependencies. The model allows to formally identifyan inconsistency pattern for dependent reconfiguration of VNF-based network services, that is missing today in the literature. To prevent this pattern, we propose a causally consistent orchestration algorithm to prevent inconsistencies while doing dependent reconfiguration.

4. The NFV dependent reconfiguration problem

Dependent reconfiguration of VNF-based network ser-vices in multi-domain federations considers the internal VNFs, the services, and the external dependencies. A service's reconfiguration can be simple if the service has zero external dependencies (i.e. *dedicated* service); other- wise, it is dependent (i.e. *composite* service). Consider the *composite* service C shown in Fig. 4, the service has two external dependencies in service A, B. Shared external dependencies introduce new challenges on the service's reconfiguration tasks, such as scaling. For example, Fig. 5 shows a composite scaling with a single dependency (i.e. one de- pendency is shared by two services). The *NFVO−C* orchestrator manages a *composite* service s_o with two external dependencies s_1 managed by *NFVO−B*, and $s_3 \equiv s_2$ managed by *NFVO-A*, respectively. It is important to note that the other orchestrators, namely *NFVO−C* and *NFVO−B*, ignore that s_3 s_2 is a shared external dependency of both s_0 and s_1 because of their limited knowledge constrained to the local domain of each orchestrator and must send grants to prevent service disruption when scaling a service with external dependencies/nested services. These grants allow the orchestrators to coordinate the compositescaling. The composite scaling is as follows:

1. Initially, the *NFVO−C* orchestrator sends a *Scale Nested* (event c_1) operation to orchestrators *NFVO-A* and *NFVO−B* via a multi-cast message m_1 to scale services s_1 managed by *NFVO−B*, and s_3 managed by *NFVO-A*, respectively. A *Scale Nested* instruction denotes the petition to scale a *nested* network service that is an external dependency.
2. The orchestrator *NFVO−B* receives message m_1 with the *scale Nested* instruction (event a_1). Since the service s_2 is an external dependency of service s_1 managed by *NFVO-A*, the *NFVO−B* sends a grant G_1 so scale service s_2 to *NFVO-A*.
3. Assume that the *scale Nested* instruction, sent by message m_1, arrives first to *NFVO-A* (event b_1). Since s_1 is an external dependency of service s_3, *NFVO-A* sends a second grant G_2 to *NFVO−B*.
4. The orchestrator *NFVO−B* validates, *sc*ales the service s_1 (event a_2), and sends a positive acknowledgment to *NFVO-A*. The orchestrator *NFVO-A* receives the positive reply via message m_2 and triggers a scale event for service s_3 (event b_2).
5. After scaling the service s_3, *NFVO-A* sends an acknowledgment to *NFVO−C* via message m_3. *NFVO−C* stores the positive answer (event c_2).
6. NFVO-*A* gets the first grant G_1 from *NFVO−B* (event b_3); but, it will not scale the service s_2 since the scaling already took place by executing event b_2 since service s_2 and service s_3 are the same (i.e. $s_3 \equiv s_2$). Thus, *NFVO-A* sends a positive reply to *NFVO−B* via message m_4.
7. NFVO−*B* receives the positive reply from *NFVO−B* (event a_3); however, it will also not scale network service s_1 since it has already scaled it by executing event a_2, and sends a positive reply to *NFVO−C* via message m_5.
8. Finally, *NFVO−C* scales the *composite* service s_0 after receiving two positives replies from *NFVO-A* and *NFVO−B*. Because of the dependent reconfiguration, there were three scale operations (event c_3).

The exchange of messages, as defined by the ETSI NFV standard, suffices to achieve consistent reconfigurations in an ideal scenario where no messages are lost and the orchestrators synchronize via global references. However, in real scenarios, orchestrators lose, send, and deliver messages asynchronously and out of order. **Such network properties lead to inconsistencies during the network services reconfiguring operations**. For example, Fig. 6 shows an inconsistency during another

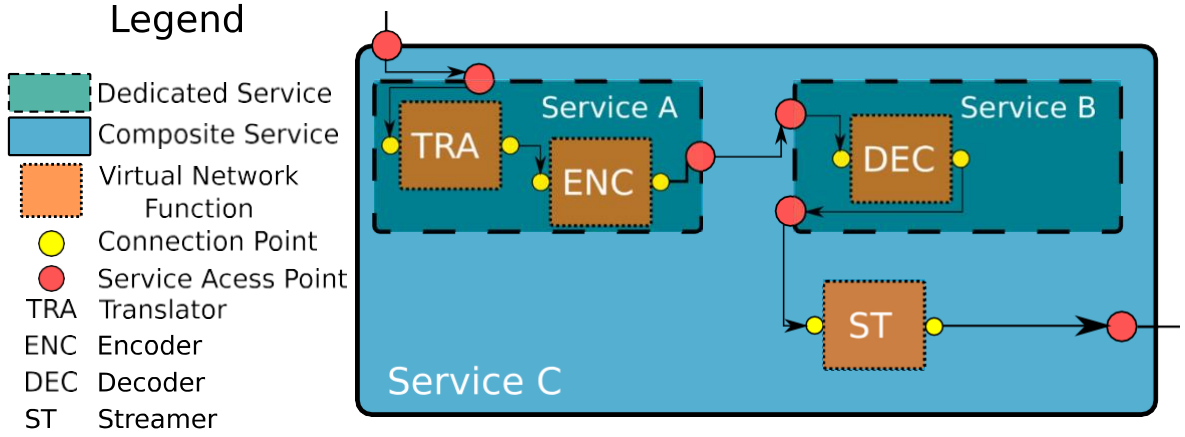

Fig. 4. Complete *composite* service C with two *external* dependencies (i.e. services A, B) as shared services. Both *external* dependencies have *internal* dependencies as VNFs (i.e. TRA, ENC, DEC, and ST) linked by connection points.

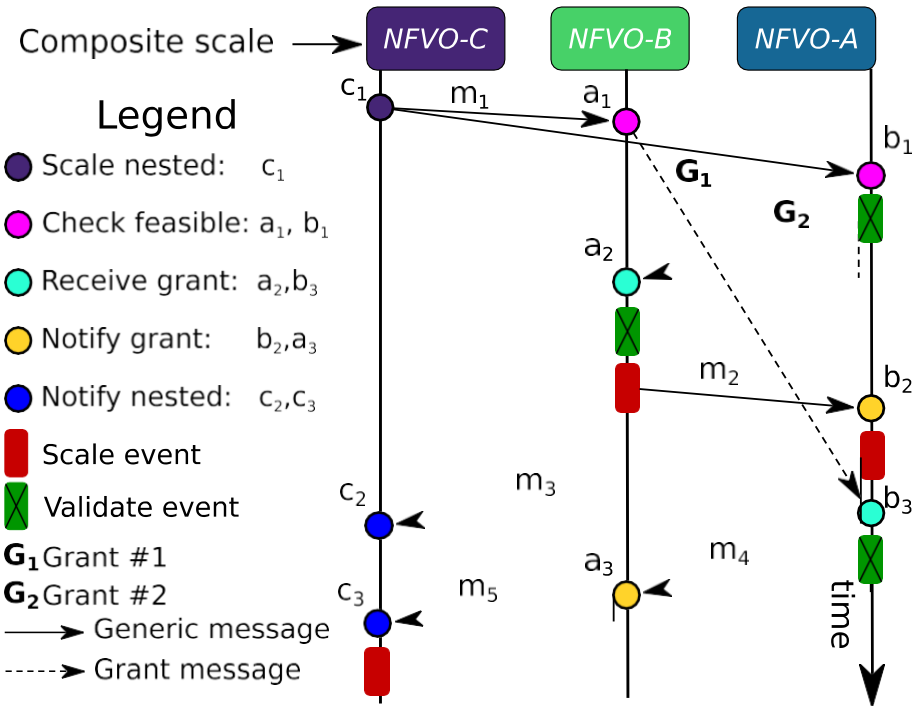

Fig. 5. The orchestrators consistently scale a *composite* service that is shared among them. In this execution of the *composite* scale operation, only three scale operations are done.

possible execution of a dependent.

reconfiguration (the order of the operations is different, here the grant G_1 is executed before the *scale nested* operation). The example shows four scale operations, where the fourth is redundant as only three operations suffice to reconfigure the *shared* network service. The extra scale operation happens because the asynchronous message delivery leads to an execution that does not satisfy the dependent relations of the *shared* network services. Figs. 5 and 6 illustrate a single dependency between a pair of services; however, in general, network services can have multiple dependencies that have the possibility of one or more inconsistencies. An inconsistency increases costs for the provider; even worse, with a long chain of network services, the cost compounds along all the chains which could violate the service level agreement. Not only does cost increase, but the network services can also be left on partial or total failure. Thus, is necessary to impose an execution order to prevent inconsistencies while doing dependent reconfigurations with shared external dependencies for VNF-based network services. To introduce our proposed execution order, first, we present the system model for the rest of the manuscript.

5. System model

The orchestrator handles the life cycle management of network services. This includes internal dependencies such as VNFs and network services. Thus, we define a management relation as follows:

Definition 1. **(Relation is-managed by)** The relation ∼ identifies the management of domain *d* for either a Virtual Network Function (VNF), v, or a service, s, according to:

1. If v ∼ d ≡ True, means the VNF v is instantiated at domain d.
2. If s ∼ d ≡ True, means the service s has either: (i)

only internal dependencies managed by the domain d (ii) if there are external dependencies, there is at least one VNF dependency v such that v ∼ d *and all other external dependencies are managed by other domains.*

Fig. 2 (see Section 2.4) shows an example of the *is-managed by* relation. In this example, the orchestrator *NFVO−C* manages the DEC VNF (i.e. *DEC* ∼ *NFVO−C True*). For the VNF-based service, the *is- managed by*, as stated by Definition 1, relation holds for all orchestrators since all the orchestrator manage external dependencies, exposed as services.

5.1. Basic concepts in distributed systems

In a distributed system, entities communicate with each other by exchanging messages. It is assumed that there is no global reference and transmission delay is bounded but arbitrary. A distributed system is composed of the sets *P*, *M*, *E* which belong to the set of processes, messages, and events, respectively.

- Processes: Programs and instances running simultaneously that communicate with other programs. Each process belongs to the set of processes of *P*. A process $p \in P$ communicates with another process $p' \in P$ by message passing over an asynchronous, non-deterministic, and reliable network.
- Messages: Abstraction of any type of message which contains data structures. Each message in the system belongs to the set *M*.
- Events: An event *e* is an action performed by a process $p \in P$. All events in the system belong to the set of events *E*. We consider two types: internal, external. An internal event occurs at a process locally hidden from other processes. An external also happens in the process but can be seen by other processes affecting the global system state. For external events, *send* and *delivery* events are considered. A *send* event emits a message $m \in M$ executed by a process. *Delivery* events identify the execution performed of received messages by a process.

5.2. Causal order

Distributed systems need shared references, such as the physical time, to decide correctly how to execute transactions. But, because of the lack of global references, the difficulty arises to find if an event takes place another. Thus, the distributed systems need another reference to

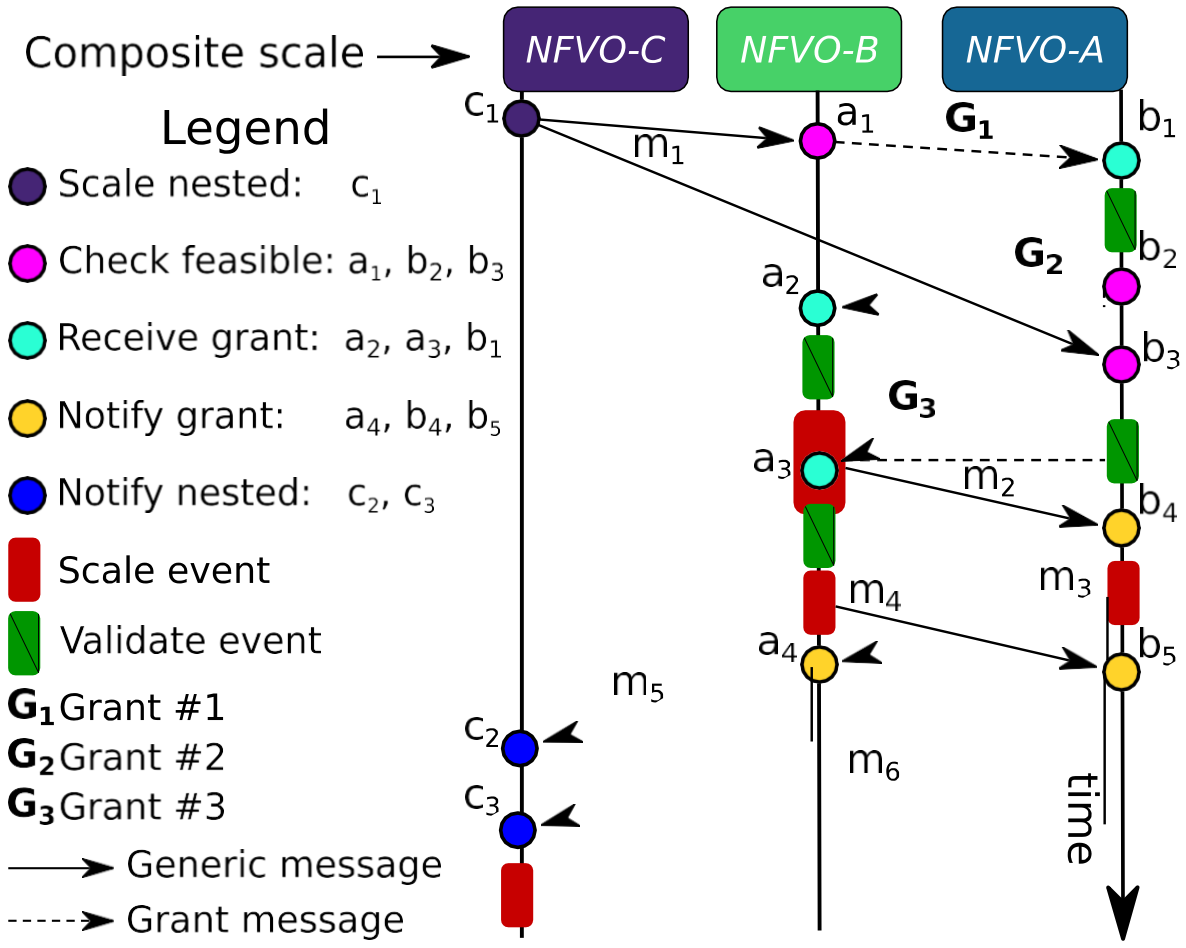

Fig. 6. Redundant dependent reconfiguration during a scaling-out operation when the *Grant* messages arrive in a different order. In this execution, four scale operations are done. This entails costs to the service provider while three only are necessary.

circumvent the absence of synchronized clocks.

Logical time introduces an execution order between events based on a partial order known as the Happened-Before Relation that establishes a precedence order between two events in the following way (Lamport, 1978): let e and e' be two events causally related. According to the happened-before relation, *e happened before* e' if there is a transference of information from e to e'. Thus, according to the relation, e must be processed before e'. Formally, the Happened-Before Relation denoted "→", is defined as follows:

Definition 2. **(Happened-Before Relation)** The relation → is the smallest relation on a set of events E satisfying:

1. If e and e' are events belonging to the same process and e originated before e', then $e \rightarrow e'$.
2. If e is the sending of a message by one process and e' is the receipt of the same message by another process, then $e \rightarrow e'$.
3. If $e \rightarrow e'$ and $e' \rightarrow e''$, then $e \rightarrow e''$.

Formally, the message causal delivery based on the happened-before relation is defined for the distributed model as follows:

Definition 3. **(Causal order delivery in distributed models)** $\forall((send(e), send(e')) \in E, send(e) \rightarrow send(e') \Rightarrow delivery(e) \rightarrow delivery(e')$ for any two internal events e, e' E. We denote $\hat{E}$ $\{E, \rightarrow\}$ as the set of events causally ordered.

6. Modeling dependent reconfigurations in distributed multi-domain orchestration

In this section, we extend the distributed system model presented in Section 5. We evaluate the NFV dependent reconfiguration problem from a temporal/event point of view and identify key information tosupport the consistent dependent reconfiguration of VNF-based network services.

6.1. *Distributed multi-domain orchestration model for Network Function Virtualization*

We present the system model. Fig. 7 shows the relation between all the entities of the distributed multi-domain orchestration system model. The federations have two or more domains. Each domain manages both network services and VNFs. Depending on their dependencies, services can be external or internal. Internal dependencies have only VNFs or other services managed by a single administrative domain. External dependencies are managed by different administrative domains. All these domains' orchestrators coordinate through messages. For internal dependencies, a scale message suffices. For external dependencies, the orchestrator must acquire a grant from all the other domains.

We develop the distributed multi-domain orchestration model by adapting the sets of processes P, messages M, and events E (defined in Section 5.1) to the NFV context by adding and defining the sets of events and messages which are specific for the model specification. Before the Definition, we introduce a set of entities used during the reconfiguration operation.

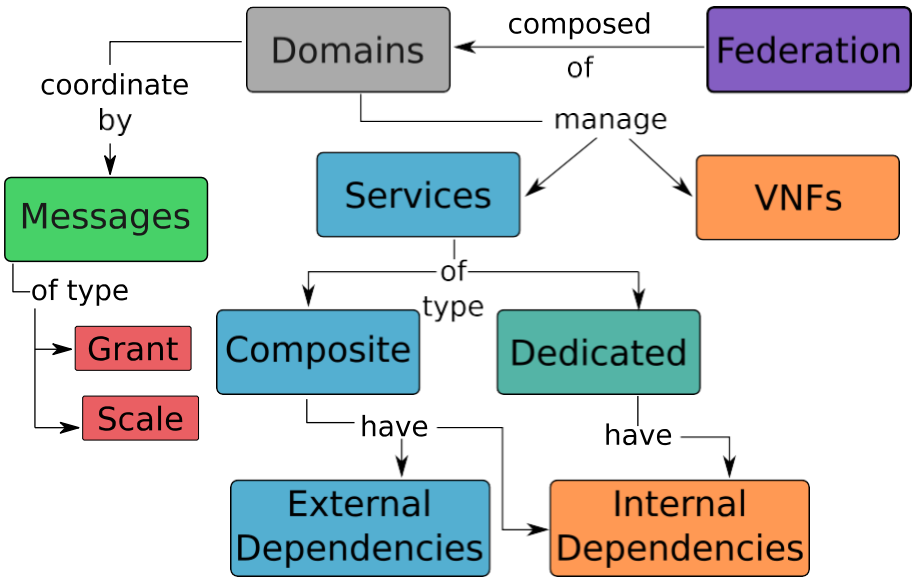

Fig. 7. Relations between the entities of the distributed multi-domain orchestration system model. The Federation on top composed of domains that manage services and VNFs.

- Domains: The collection of systems and networks operated by a single organization. In the case of multi-domain federations we denote this set as $D = \{d_1, d_2, \ldots, d_p\}$. The number of domains p is known beforehand since we considered a close federation.
- Virtual Network Functions (VNFs): The basic components to instantiate complex network services. We follow the ETSI standard by using the abstraction of Virtual Network Function Componentsthat enable the VNF to operate. The super-set of VNFs V is composed of disjoint sets $V_1, V_2, \ldots, V_p$ where $\forall v \in V_i$, $v \sim d_i$ where v is a VNF and d_i is the i-th domain in the federation. Each v ∈ V is defined as a function $f : x \Rightarrow y$ where x, y are input and output traffic flows, respectively
- Network services: The entities which offer complex solutions by aggregating VNFs with unspecified connectivity between them or according to forwarding graphs. A forwarding graph describes a topology of network services by referencing a pool of connection points and Services Access Points. The network service set is composed by a set of processes $S = \{s_1, s_2, \ldots, s_k\}$. Each service s is associated with a domain $d \in D$ denoted as $s \sim d$. Internal dependencies I_s of a network service s are either VNFs $v \in V$ or network services $s' \in S$ such that $\forall v$, $s' \in I_s$, $v \sim d$, $s' \sim d$ where $s \sim d$. Similarly, external dependencies Γ_s are only network services such that $\forall s' \in \Gamma_s$, $s' \nsim d$ where $s \sim d$. The set of total dependencies of service s is denoted as $\Delta_s = I_s \cup \Gamma_s$. Let Ω_s be the set of orchestrators that manage the external dependencies of service s such that $\forall o \in \Omega_s$, $\nexists s' \in \Gamma_s | s' \sim o$. Network services in the set S can be either of type *dedicated* or *shared* according to their dependencies. We formalize this with the following:

Definition 4. **(Shared and dedicated network services)** The service, s, managed by orchestrator o, belongs to the type shared if it is an external dependency of another service s' unmanaged by the same orchestrator o: $s \in \Gamma_{s'}, s' \nsim o, o \in o', s \nsim o'$; otherwise, it is dedicated.

Fig. 2 (see Section 2.4) show an example of *shared* and *dedicated* services. In this example, the *DEC* VNF is shared among all the domains as shown in Fig. 2. Moreover, this VNF is exposed as service B which makes it a *shared* service as shown in Fig. 4. All other VNFs can be exposed as *dedicated* services.

We now extend the model presented in Section 5.1 to meet the ETSI standard for multi-domain orchestration as follows:

- Messages: We extended the concept of abstract messages in distributed systems, described in Section 5.1, with the specific types required for reconfiguration of network services. All messages have the following parameters: m {(*sender*, *receiver*, *data*, *type*)}. We consider the following type of messages:
  - Scale: Increase or decrease the number of instances that belong to either a VNF or Network Service.
  - Response: Acknowledgment of a complete scaling operation of an external dependency.
  - Notification: To signal the sender of a Scale message there has been an update in the lifecycle management.
  - Grant: Permission to scale external dependencies.
- Events: As mentioned in Section 5.1, there are two types of events: internal and external ones. The set of internal events $E_{internal}$ is the following:
  - VnfMScaleRequest (v, data) denotes the orchestrator's request to the VNF Manager to initiate the scaling of VNF v specifying data.
  - VimChangeResource (v, data) is the event that the orchestrator sends to the Virtual Infrastructure Manager to change either processor, storage, or network information of the VNF v according to the data.
  - VimModifyConnectivity(v, data) refers to the changing of connection points of VNF v by the Virtual Infrastructure Manager.
  - VimInstantiate(v, data) denotes the instantiation or shutdown of VNFs v ∈ V for a particular service according to the data.
  - CheckCompositeNSConsistency(s) denotes the verification before the scaling of service s. Since we consider the complete ETSI orchestrator's architecture as a single process for ease of understanding, all the details of this event are abstracted in a single execution; however, in reality, multiple entities play a role in this message.
  - ScaleNS(s) denotes the scaling of the dedicated network service s whose only dependencies are internal and belong to the set Vp managed by orchestrator op.

The external events considered are send, receive, de - livery and the set is denoted as $E_{external} = E_{send} \cup E_{receive} \cup E_{delivery}$. Since we consider a single orchestration per domain, we re-write the *is-managed by* relation (see Definition 1, Section 5) as $s \sim o$ for any service $s \in S$ and orchestrator $o \in O$.

The set of send events E_{send} is the following:

- $ScaleCompositeNS(s, data)$ denotes the petition to scale the *composite* network service s by using parameters in *data*, this could trigger multiple *ScaleNestedNS* or *SdNSLCMGrant* events by the external dependencies of s.
- $ScaleNestedNS(s, data)$ denotes the petition to scale a *nested* network service s by using parameters in *data*, this will trigger a grant *SdNSLCMGrant(s')* request event for the external dependency $s' \in \Gamma_s$.
- $SdNSLCMGrant(s')$ refers to the request coming from orchestrator o to verify and scale service $s' | s' \sim o', s' \Gamma \in o o'$. This event can trigger multiple *ScaleCompositeNS*, *ScaleNestedNS SdNSLCMGrant* events, respectively.

The set $E_{receive}$ is composed of the following events:

- *RecResponseNestedNsScaling(s').* This message is received by orchestrator $o \in O$ that manages service s which has service s' as an external dependency. It denotes the answer (positive or negative) to a previous *SdNSLCMGrant* related to nested service $s' | s' \in \Gamma s$.
- *RecResponseCompositeNsScaling(s').* This message is received by orchestrator $o \in O$ that manages service s which has service s' as an external dependency. It denotes the answer (positive or negative) to a previous *SdNSLCMGrant* related to *composite* service $s' | s' \in \Gamma s$.
- *RecNSLifecycleChangeNotification({start, result}, s')* refers to the acknowledgment or result from orchestrator o' to o that has sent an event of type *ScaleNestedNS(s')* or *ScaleCompositeNS(s')* such that $s' \sim o'$ and $s' | s' \in \Gamma_s$ for any service $s \in S$.

The set $E_{delivery}$ has a unique event:

- *DlvNSLCMGrant(s')* denotes that an orchestrator $o' \in O$ delivered a Grant message sent by a *SdNSLCMGrant(s')* event. It identifies the execution performed by the orchestrator o' associated to service s' managed by orchestrator o'. After delivery, o' will send a notification to the sender of the grant.

The set E for events (see Section 5.1) is augmented. We consider internal and external events as defined by the sets *Einternal*, *Eexternal*, respectively such that $\hat{E} = \{E \cup Einternal \cup Eexternal, \rightarrow\}$. For simplicity, we represent the send events of set E_{send}: *ScaleCompositeNS*, *ScaleNestedNS*, *SdNSLCMGrant* as (*s*). The delivery event of set $E_{delivery}$ *DlvNSLCMGrant* as *delivery(s)* for any service $s \in S$. This set is causally ordered by the $\rightarrow$ relation (see Definition 2).

6.2. Dependent reconfiguration of network services in distributed multi-domain federations

A dependent reconfiguration happens when an orchestrator requests a reconfiguration for service s and has external dependencies such that the set $\Gamma_s \neq \varnothing$. In this case, multiple grant requests are sent to orchestrators bounded by $|\Gamma_s|$. According to the ETSI standard, the following steps are required for a dependent reconfiguration (we don't consider notifications or answers) (Etsi, 2019b):

1. Scale a *composite* network service.
2. Scale nested network services.
3. Request grants to scale external dependencies.
4. Validate requests, check feasibility, and consistency of grants.

We formally define the dependent reconfiguration as follows:

Definition 5. (Dependent reconfiguration) Let $s \in S$ be a composite service that has at least one external dependency such that $\Gamma_s \varnothing$ managed by orchestrator $o \quad O$. Let event $e \quad E$ be of type ScaleCompositeNS executed by orchestrator o. A dependent reconfiguration happens during the composite scaling of e if and only service s has an external dependency $s' \in \Gamma_s$ such that $s'' \in \Gamma_{s'}, s'' \in \Gamma_s$. In other words, both network services share an external dependency; thus, the joint set $\Gamma_s \Gamma_{s'} \varnothing$.

We illustrate an example of the conditions required for a dependent reconfiguration as described in Definition 5 for a *composite* service by extending the example shown in Fig. 4. Fig. 8 shows an example of the *composite* service G. This service has three nested services in C, E, and F, respectively. Each nested service has its own internal and external dependencies, as shown by the differences between service A and B. Since the two nested services C and E share an external dependency (i.e. *shared* service) they will trigger a dependent reconfiguration in case one orchestrator decides to reconfigure the *shared* service B. The orchestrators exchange grants by executing events in the set $E_{external}$ (i.e. *SdNSLCMGrant* and *DlvNSLCMGrant*). In the case of dependent reconfiguration executed during the scaling for *shared* network services in multi-domain federations, the delivery of *DlvNSLCMGrant* events mustbe respected to consistently execute the reconfiguration.

6.3. Modeling inconsistency in NFV dependent reconfiguration of network services from a temporal perspective

Consider the diagram of Fig. 9. It shows the relevant events during a dependent reconfiguration (see Definition 5) according to the ETSI standard. In this scenario, the topology is composed of four orchestrators. According to our model (see subsection 6.1), the set of processors for this scenario is $O = \{o_{-1}, o_0, o_1, o_2\}$. The update begins when orchestrator o_{-1} sends a message m_0 to o_0 where $m_0 = (o_{-1}, o_0, e_{x0} = \{(s_0, data)\})$ and $s_0 \sim o_0$ and e_{xx0} is a *ScaleCompositeNS*. Validation and feasibility are checked on event e_{00}; this entails validating the parameters sent, the authority of the sender, and checking the feasibility for the VNF Manager to scale all the VNFs. In the case the service has external dependencies, the orchestrator sends a multi-cast message $m_1 = (o_0, \{o_1, o_2), e_{01} \{(s_1, s_2), data)\})$ where e_{01} is of type *Scale nested*. If an orchestrator receives a *Scale nested* event, it will also validate the request as shown in event e_{11}. If the service has an external event, it will send a grant through a message as shown in m_2 and will wait for a positive acknowledgment before scaling takes place; if there are no external dependencies, a scaling will take place. In the event of receiving a grant, the orchestrator checks the consistency of all network services affected by the grant as shown in event e_{21} and sends a notification through message m_3. If network services are affected, the orchestrator can request a *scale* composite instruction and begin another dependent reconfiguration.

According to the ETSI standard, the execution of events shown in Fig. 9 is valid; however, it introduces an inconsistency for network services being scaled. The inconsistency is created because service s_1 managed by orchestrator o_1 has outdated information after the scaling operation of service s' triggered by event e_{23} if s' is an external dependency of s_1. More precisely, the inconsistency is brought by the execution of e_{20} before e_{22}.

One way to see this out-of-order execution is the delay brought by asynchronous communication and lack of global references; in turn, this creates a non-deterministic execution of reconfiguration operations. This is the temporal perspective encoded in the message sent by or-chestrators to signal the relevant steps of dependent reconfiguration.Upon analyzing the communication diagram corresponding to the execution diagram shown in Fig. 9, we can observe the inconsistency is created when the transmission time interval of m_1 is greater than the transmission of time interval m_2 plus the message forwarding time of all the *grant* of dependencies. We generalize and formalize the inconsis- tency pattern from a temporal perspective in the Definition 6.

Definition 6. (Temporally inconsistent dependent reconfiguration)

Let s be a composite service managed by orchestrator o such that $\Gamma_s \neq \varnothing$. *Let* $m = \{o, \Omega_s, ScaleNe(\Gamma_s)\}$ *be a message to scale the nested external*

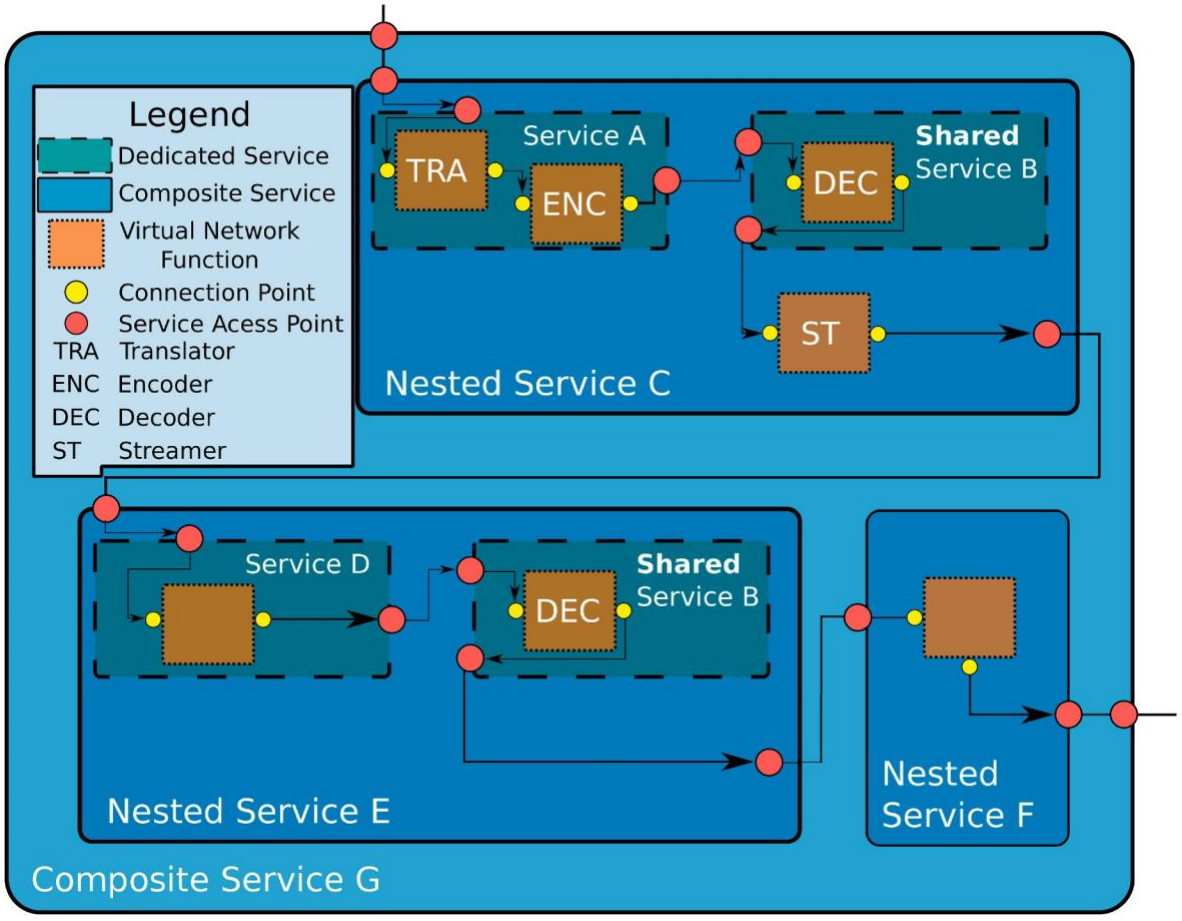

Fig. 8. Example of the conditions for a dependent reconfiguration. The *composite* service G has two *nested* services that share service B as an *external* dependency. To reconfigure this *shared* service, the orchestrators must coordinate by grants.

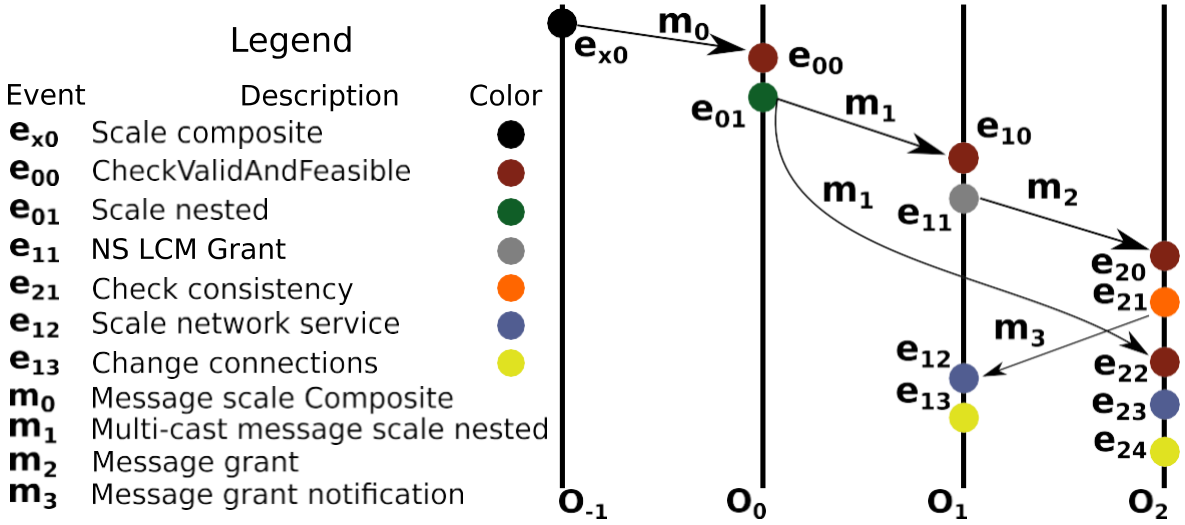

Fig. 9. Inconsistency during dependent reconfiguration for a *composite* service. All events concerning acknowledgments and notifications are hidden and abstracted. Messages arrive arbitrarily and events can be executed out of order.

dependencies of s. Let $m' = \{o, \Omega_s \,/\, o', ScaleNe(\Gamma_s \,/\, s')\}$ *be a message to scale all external network services except a single service* $s' \in \Gamma_s$ *managed by* $o' \in \Omega_s$. *Let* $m'' \{o', \Omega_{s'}, SdNSLCMGrant(\Gamma_{s'})\}$ *be a message to* grant *the operation of service s'. Let* (x) *be a function that measures the time taken to send, receive and deliver a message* $x \in M$. *A temporal inconsistency during dependent reconfiguration is created if:*

$$(m') > (m) + T(m'') \mid \Gamma_s \cap \Gamma_{s'} \neq \varnothing, s' \in \Gamma_s$$

A possible solution to the problem of inconsistent dependent reconfiguration posed by Definition 6 is to establish common temporal references for all orchestrators and perform the execution of operations

according to this reference. However, as discussed in Subsection 2.3 such a solution is unpractical since each orchestrator has limited knowledge. Furthermore, even clock synchronization algorithms such as the Network Time Protocol (Mills, 1991) make it difficult to perfectly synchronize clocks across all the network entities.

6.4. Modeling the NFV dependent inconsistent reconfiguration of network services from an event perspective

In this section, we discuss the inconsistency during the dependent reconfiguration (see Definition 5) of network services from an event perspective. The focus centers on the relevant events during the scaling operation. Similarly to Section 6 we use the execution flow presented in Fig. 9 as an inconsistent example. We define the conditions of inconsistency as follows:

Definition 7. (Event related inconsistency of dependent reconfiguration) Let s_1, s_2 be two network services managed by o, o' respectively. Let $\Gamma_{s1} \neq \varnothing$, $\Gamma_{s2} \neq \varnothing$ be the sets of external services' dependencies of s_1, s_2 respectively. Let Ω_{s1}, Ω_{s2} be the set of orchestrators that manage the external dependencies of service s_1, s_2 respectively. Let e_{sc} be a **ScaleComposite** event of network service s_1 requested by an orchestrator $\hat{o} \notin \Omega_s1 \cup \Omega_s2$. Let e_{sn} be a **ScaleNested** event of network services Γ_s1 such that $e_{sc} \rightarrow e_{sn}$ and the scale instruction is sent to Ω_s1 using the message m. Let e be the execution of the **ScaleNested** e_{sn} instruction at an orchestrator $o \in \Omega_s1$. Let e_g be a **Grant** of network services Γ_s2 such that $e_{sn} \rightarrow e_g$ and the grant instruction is sent to Ω_s2 using the message m'. Let e' be the execution of the **Grant** e_g at an orchestrator $o \in \Omega_s2$. There is an inconsistency if the following conditions hold:

1. $\exists(\bar{s}, \hat{s})$, $\bar{s} = \hat{s} \in \Gamma_s1 \cap \Gamma_s2$ such that $\bar{s} \in \Gamma_s1, \hat{s} \in \Gamma_s2$, and
2. $e' \rightarrow e$

Fig. 10 shows a graphic representation of Definition 7. In Fig. 10 (I) we observe the composite scaling of service s_1 with event e_{sc}, given that this service has external dependencies it sends a scale nested instruction to all orchestrators that manage its dependencies by e_{sn}. Fig. 10 (II) shows the delivery of the nested instruction at orchestrator o'. Since the external dependency s_2 also has dependencies, it sends a grant instruc- tion to all the managers of its dependencies with event e_g. Fig. 10 (III, IV) show the case when both s_1 and s_2 have common external dependencies.

In Fig. 10 (III) the reconfiguration is consistent in the purple rectangle as the scale precedes the grant instruction. Fig. 10 (IV) shows an inconsistent dependent reconfiguration as the grant operation precedes the scale instruction. Even if some reconfigurations are consistent, a single inconsistent reconfiguration is sufficient to bring the whole service down. More, precisely, inconsistencies bring both partial and total failures for network services reflected on greater cost for the providers. A key property for any reconfiguration is to be consistent and stop the conditions of Definition 7. To prevent them at least one condition from Definition 7 must remain unsatisfied. A federated environment makes preventing Condition 1 challenging as the core of these environments is sharing resources plus is difficult to ensure due to the limited information each orchestrator has. Thus, the goal is to prevent Condition 2 from happening; that is, a nested scaling should always precede a grant operation. Our proposed algorithm identifies and prevents the second condition from happening via causal consistency.

7. Consistency management in dependent reconfigurations

Based on the formalization presented in Section 6, we show how the presented algorithm allows us to capture the causality and avoid inconsistency of network services during dependent reconfigurations, such as scaling in NFV. First, an overview of the algorithm is presented in Subsection 7.1. Then, a simplified workflow of the algorithm is shown in subsection 7.2. We showed all details of functions in Appendix A.

7.1. Algorithm overview

Our algorithm complies with the ETSI standard procedure to provision network services (i.e. we consider the same definitions for VNFs, services, messages, events); however, we propose a new orchestration algorithm to reconfigure *composite/shared* VNF-based network services not considered in the standard. The federation's orchestrators execute the algorithm when they send and receive standard instructions as defined by ETSI. The algorithm bifurcates with multiple function calls because of asynchronous calls while reconfiguring *shared* services. The recursive nature of our solution handles the dependencies of network services by emitting grants as messages among orchestrators. The or- chestrators deliver the messages in causal order preventing in- consistencies (see Definition 7). Each orchestrator in the federation

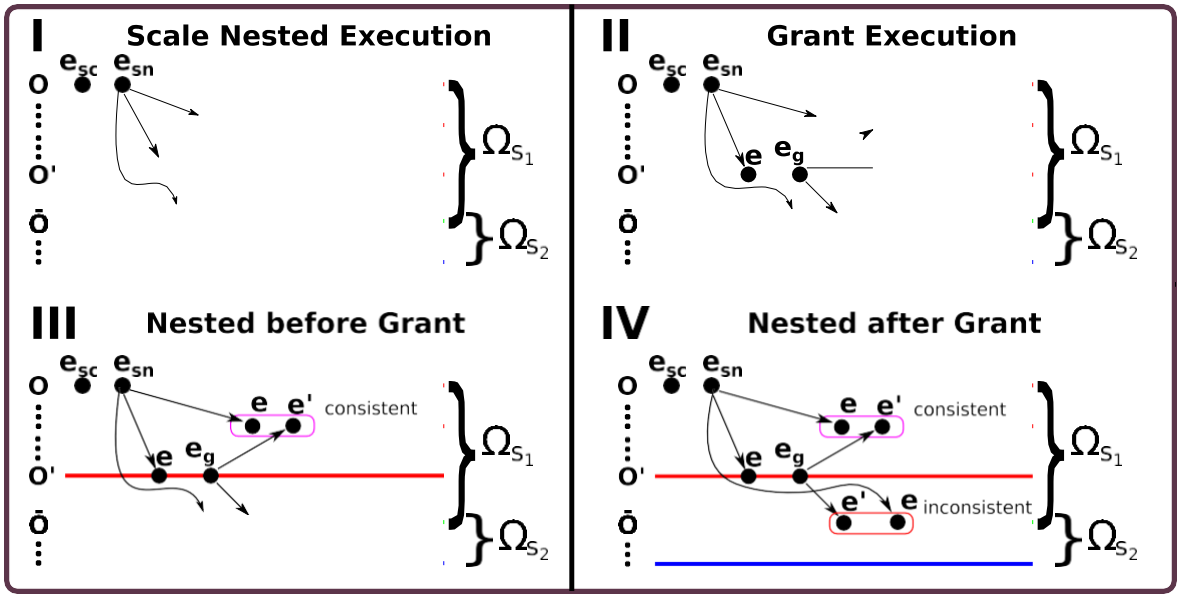

Fig. 10. Inconsistency during dependent reconfiguration for a *composite* service. An orchestrator executes scale and grant events of a shared external dependency out-of-order.

executes the following algorithm to support the causal delivery of messages according to the dependencies in the service's descriptors. We summarize it as follows:

- Input: The set of messages presented in Section 6.1 and a vector clock create a tuple that communicates to other orchestrators the changes seen so far from the sender.
- Execution: All reconfiguration messages are asynchronously disseminated and no upper bound on delay is considered. Whenever an orchestrator *o* sends a *LifeCycleManagement* message *m* to the orchestrator *o'*, it never blocks and waits for an acknowledgment message of the delivery of *m*. The clock of each orchestrator is independent of each other such that there is no synchronization with another orchestrator, thus, the execution is fully asynchronous.
- Data Structures: We consider two data structures: orchestrators and network services.
  - Orchestrator:
    * ID: The unique identifier of the orchestrator.
    * vectorClock: Control information that stores the dependency information between messages being exchanged. The size of the vector is equal to the number of orchestrators in the federation. Each element of the vector clock of orchestrator *o* is a tuple of the form (oid, *logical_clock*) which records the last messages seen by *o*.
    * pendingOperations: The external dependencies that wait for the confirmation of the scaling operations.
    * externalOrchestrators: The list of all orchestrators in the federation.
    * internalDependenciesToScale: The list of all VNFs and network services waiting to be scaled. They are stored to prevent scaling them and then receiving a failure for an external dependency.
  - Network Service:
    * ID: The service unique identifier.
    * dependencies: The list of dependencies of the service. In case of scaling, all must confirm the scaling otherwise the operation is aborted.
    * orchestratorID:The identifier of the service's orchestrator.
    * originalService: The service who originally sent the scaling operation in case of a dependent reconfiguration.
    * type: They type of the component. It could either be a Service or VNF.
- Messages:
  - *NotificationLCM*: This notifies the other orchestrators of a change in their vector clock.
  - *NotificationLCMFailure*: Indicates the failure in scaling an external service. This will abort the original scaling operation.
  - *GrantLCM*: Ask permission to scale an external dependency in the form of a *composite* service.
  - *ScaleConfirmation*: Acknowledgment of the successful scaling of a dependency.

7.2. Algorithm details

Fig. 11 shows the flowchart to scale a VNF-based network service. First, the orchestrator increments its vector clock by one. Then, it adds the scaling to pending operations since the network service could have

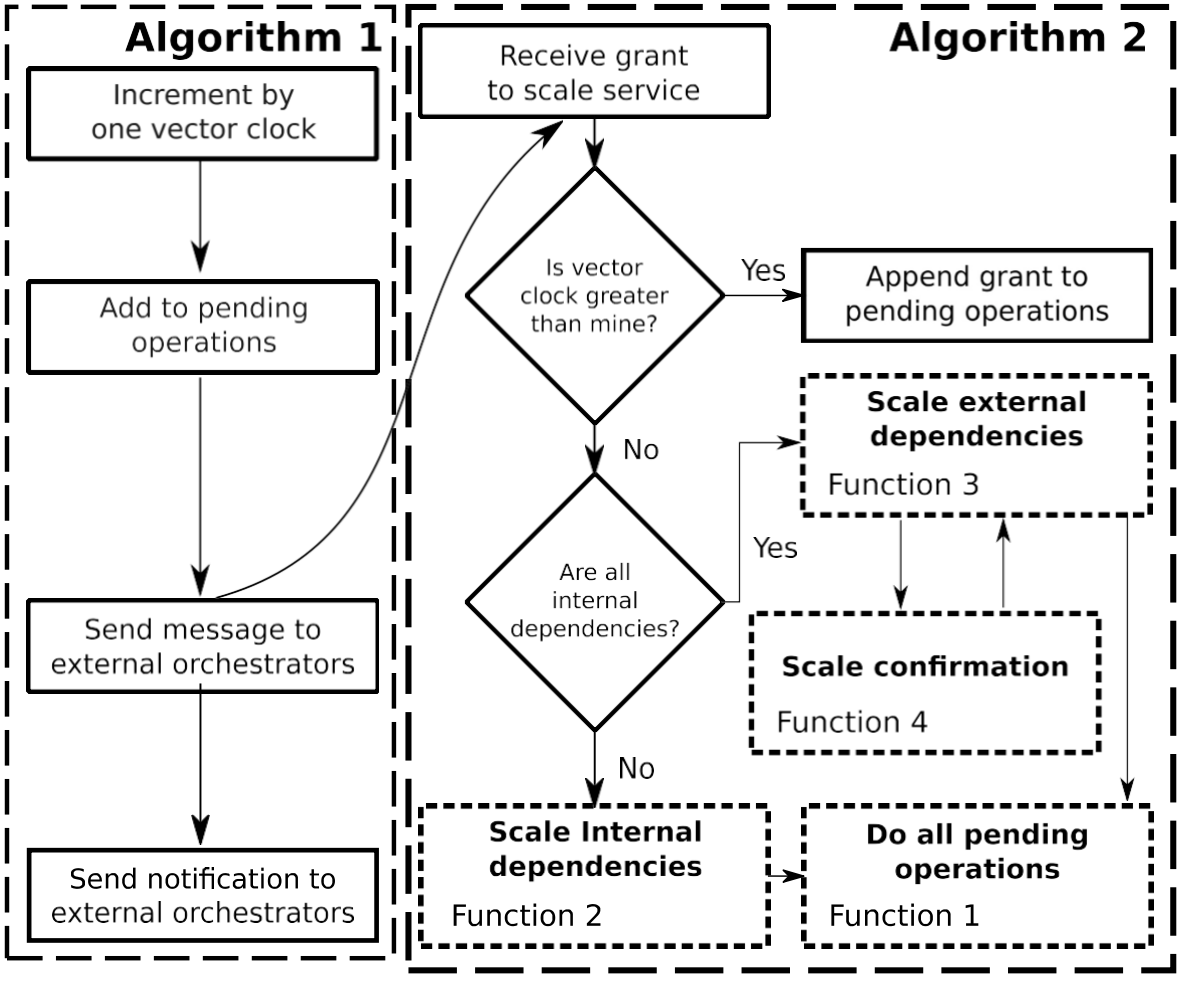

Fig. 11. Simplified workflow to consistently scale a shared VNF-based network service. All algorithms and functions in bold are defined in Appendix A.

external dependencies. After, it sends a grant to all the orchestrators who manage the service's external dependencies. Finally, to ensure the causal delivery of messages, the orchestrator notifies the others. Algo- rithm 1 (see Appendix A) shows the function to scale a VNF-based network service.

Once an orchestrator receives a grant request to scale a VNF-based network service, it executes the workflow, as shown in Fig. 11. First,the orchestrator receives the grant to scale the service. Then, it compares the received clock with its own. If it is greater by more than one value, it stores the grant in the list of pending operations. After, if the clock's difference is one, the orchestrator checks if the service to be scaled has any external dependencies, — scaling them in the positive case or justthe internal ones. In the end, the orchestrator tries to execute pending operations, thus ensuring events are delivered according to the causal order. Algorithm 2 (see Appendix A) shows the function to apply the grant to scale a VNF-based shared network service.

8. Implementation and validation

We implemented our proposed algorithm to measure both performance and correctness criteria (i.e. zero inconsistencies while reconfiguring). The following sections comprise the distributed setup (Section 8.1), metrics evaluation (Section 8.2), experiments (Section 8.3), and discussions (Section 8.4).

8.1. Distributed federation setup

We tested our solution using Azure's cloud infrastructure. We chose multiple domains from the cloud provider from the following locations: North Europe, West US, South Korea, East US, and the UK. For each domain, we instantiated a virtual machine to host the orchestrator software. All virtual machines have the same configuration: 2 CPUs, 30GB of hard drive, 4 GB of RAM, and Linux 18.04-LTS. Each domain hasits policies, topology, and is managed by a single orchestrator. Nowa- days, multiple open-source orchestrators follow the ETSI standard like OSM (Israel et al., 2019), however, none implements the required interfaces to support a federation. Thus, we implemented an orchestration platform in Python. The source code can be found in[1].

Network Services are created by chaining VNFs, internal, and external services. The VNFs considered for the network services are part of Content Delivery Networks functions that process video such as En- coders. The specification for the service is stored in JSON files that contain parameters for network services and their corresponding VNFs. Table 1 shows the parameters used for all experiments. We created multiple experiments by randomly assigning VNFs, network services, and their constraints to the orchestrators in each of our domains. We alsogenerated a random set of scale requests to test our solution. To simulate asynchrony in the network, all messages have random waiting times.

Table 1
Experiment's parameters and their range.

Variable	Range
Number of services	3000
Number of reconfigurations	10, 20, ···, 100
Repetitions per experiments	30
Number of dependencies	1−6
VNFs per orchestrator	600
Random delay range	[1−100]ms

[1] https://doi.org/10.5281/zenodo.3989957.

8.2. Metrics to evaluate

To measure the benefits and trade-offs of our algorithm we evaluated the following metrics:

- Inconsistencies: The number of differences when two or more orchestrators have different configuration for a shared network service. They should be minimized or prevented while reconfiguring dependent network services.
- Message overhead: The number of messages sent to coordinate orchestrators. Messages induce a waiting time until the appropriate one is received.
- Reconfiguration time: The time taken to achieve the reconfiguration. Ideally, this would be short; otherwise, the user of the network ser- vice suffers an interruption.
- Memory overhead: The amount of information stored in memory to coordinate the orchestrators.

Ideally an orchestration algorithm would have zero inconsistencies while achieving a fast reconfiguration with few messages to coordinate the orchestrators. However, preventing inconsistencies has an associ- ated cost. Next, we measure the performance of our algorithm compared to the ETSI standard (Etsi, 2019b) as it is the closest work in the liter- ature as shown in Fig. 3 (see Section 3).

8.3. Experiments

We evaluated the performance of our algorithm and the current ETSI standard (Etsi, 2019b) for *composite/shared* services. This work is the closest work to ours as it includes: (i) multiple administrative domains, (ii) *composite* VNF-based network services, and (iii) dependent reconfiguration. It is important to note that to support inconsistency detection we implemented and added vector clocks (Fidge, 1988) to the ETSI standard. The original implementation does not contemplate this. Weconsider two experiments. For the first experiment, we deploy multiple *dedicated* and *composite* VNF-based network services and reconfigure them. For the second experiment, we deploy only a single service reconfiguration and measure the effects of dependencies for each metric considered. Both experiments had a threshold of 60 s. If a reconfigura- tion takes longer than the threshold, we consider it invalid.

8.3.1. Experiment 1. single service reconfiguration

This experiment aims to measure the overhead of our proposed algorithm compared to the ETSI standard for a dependent reconfiguration of network services. In this scenario, each reconfiguration is done one at a time. We tested over 5500 random reconfigurations for all the services we generated, as shown in Table 1. We consider intervals with increments of ten, up to 100, to measure the performance of both algorithms in terms of the metrics considered (see Section 8.2). A time-out of 60 s was set. If the time to reconfigure exceeded the time-out, we consider the reconfiguration as invalid. Figs. 12−16 show the results.

8.3.2. Experiment 2. performance with respect to the number of dependencies

The aim of this experiment is to measure how the overhead metrics increase concerning the total external dependencies. In this experiment, we count all the external dependencies, not only the immediate dependency. For example, if a service has 3 external dependencies and one of these external dependencies has also 2 external dependencies, the service will have 3 immediate dependencies but 5 in total. Thus, this experiment reveals more about the relationship between the external dependencies and the solution overhead. Table 2 shows the parameters for the experiment. Similarly to Experiment 8.3.1 a time-out of 60 s was set for invalid services. Figs. 17−20 show the results for each metric considered.

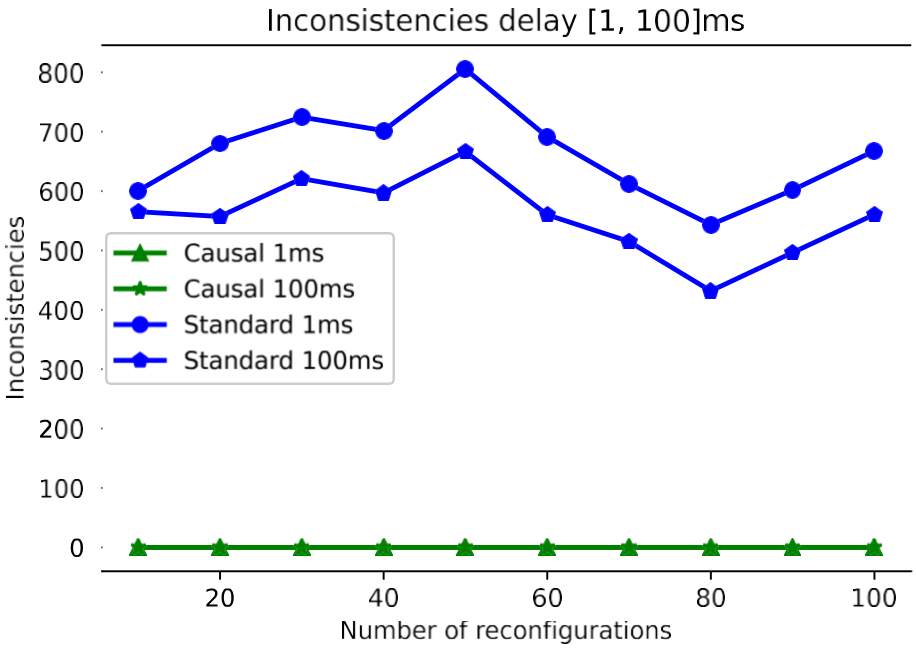

Fig. 12. Inconsistencies per number of reconfigurations. Our algorithm obtains zero inconsistencies, unlike the standard.

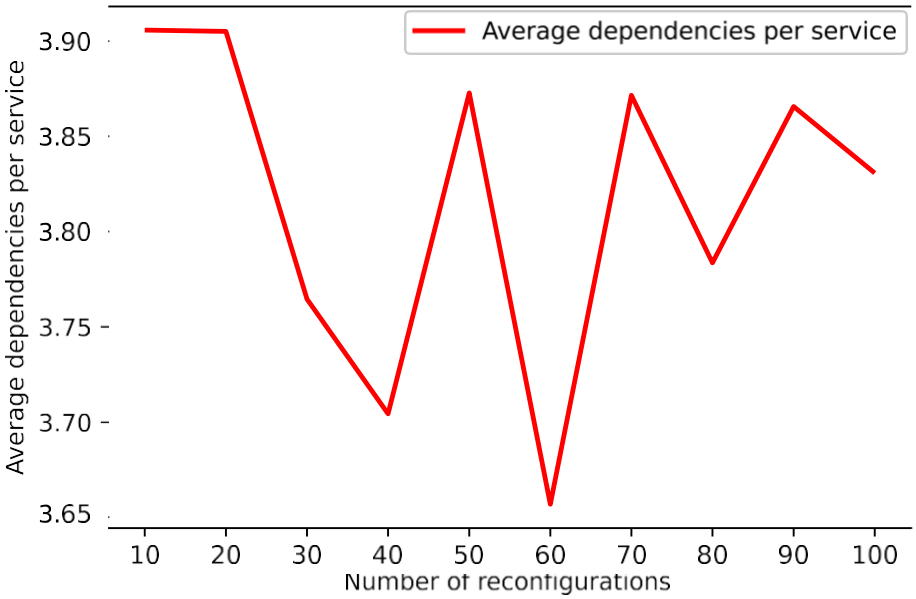

Fig. 13. Average dependencies per VNF-based network service. On average, services have between 3,4 dependencies.

8.4. Discussion

Our solution gets the expected performance of zero inconsistencies; this is not the case for the ETSI standard. EXperiments validate the algorithm and show that the current standard gets inconsistencies while reconfiguring the network services. Even if network services have few external dependencies, the standard still has inconsistencies, as shown in Fig. 17. It can be seen how, despite reconfiguring a single *composite* service, when the number of dependencies is greater than two, the standard already has inconsistencies.

For multiple reconfigurations, our proposed algorithm prevents inconsistencies, unlike the standard as shown in Fig. 12. For the standard, the number of inconsistencies changes over the number of reconfigurations. This variation happens as the services, for each step, were created and selected at random using the range of parameters (see Table 1). From the first sight, it appears there is no relation between the number of dependencies and the inconsistencies as shown in Fig. 13. However, the more detailed analysis of the second experiment, where we fiXed the dependencies instead of having services different dependencies, reveals that there is a relation between the them as shown in Fig. 17 where the number of inconsistencies grows as a function of the number of dependencies. Moreover, since we considered both *dedicated* and *composite* VNF-based network services, it is likely that for larger experiments a higher number of *dedicated* were chosen. Thus, we see the downtrend between steps 60−80 in Fig. 12. Nevertheless, our algorithm prevents inconsistencies irrespective of the number of reconfigurations, unlike the ETSI standard.

Preventing such inconsistencies comes with a cost associated with it. First, we evaluated the complexity of our proposed algorithm in terms of the number of dependencies n and orchestrators m. Then, we measure

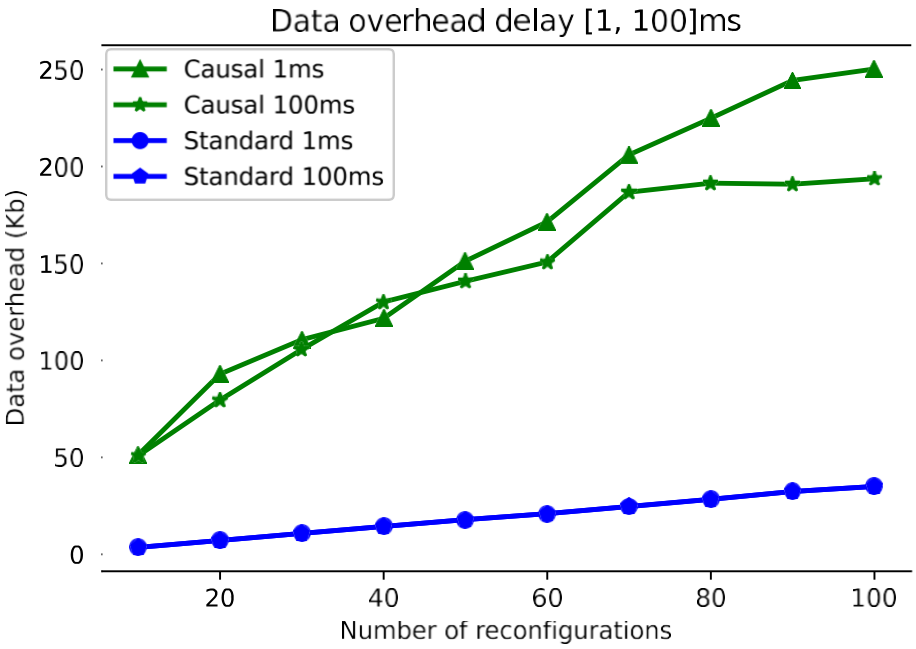

Fig. 14. Memory overhead per number of reconfigurations. Our proposed algorithm has a greater cost.

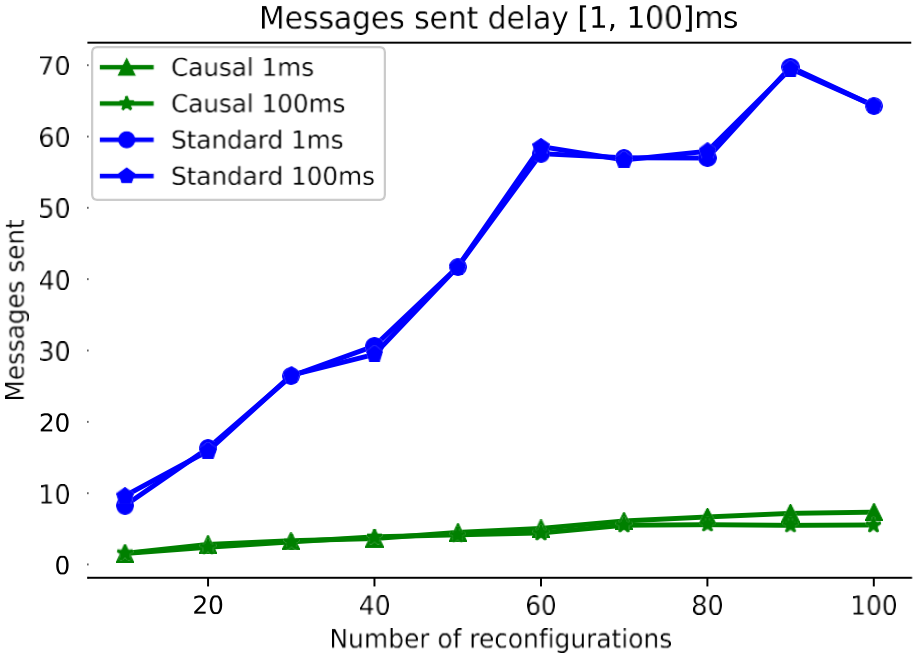

Fig. 15. Messages sent to request grants and notify orchestrators. Our proposed algorithm obtains better performance as it prevent inconsistencies that create redundant messages.

the performance of our proposed algorithm and compared it to the ETSI standard.

The time complexity of our proposed algorithm is $O(n^2)$ where n is the number of dependencies. The space complexity is $O(m)$, where is the number of orchestrators. It is linear since our algorithm keeps track of the affected orchestrators using vector clocks and stores out-of-order instructions as pending operations for each orchestrator. Our algorithm has greater time complexity than the ETSI standard (Etsi, 2019b) who has a time complexity of $O(n)$ and a space complexity of $O(m)$ in ideal conditions (i.e. one reconfiguration at a time, deterministic network). However, the added cost of our algorithm, in terms of time and space, prevents inconsistencies for dependent reconfigurations.

Performance-wise we see how our algorithm requires storing more information to coordinate the orchestrators compared to the ETSI standard as shown in Figs. 14 and 18. Delay has an impact on the amount of information stored, as shown by the gap between the twolines of our proposed algorithm. In general, for smaller waiting times more messages arrive out of order and the orchestrators must store causal information to deliver them in the correct order to prevent the inconsistency pattern identified of Definitions 6,7 (see Sections 6.3 and 6.4). The ETSI standard is unaffected by the delay as it not keeps any information to coordinate the orchestrators outside the grants. Delay also impacts to a lesser extent the other metrics when there is more than one service reconfiguration. This can be seen when comparing Figs. 12

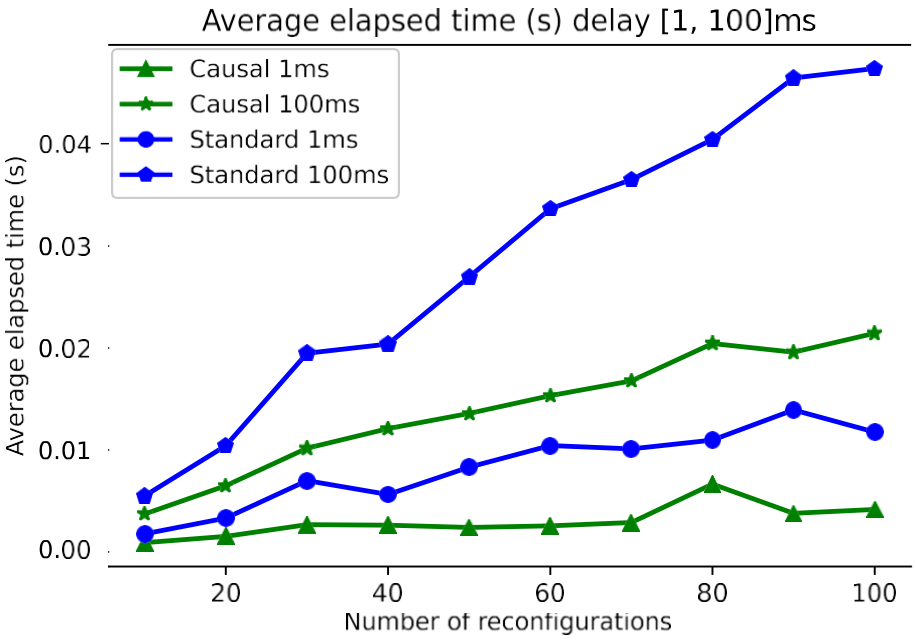

Fig. 16. Time spent reconfiguring the VNF-based services. Our proposed solution reconfigures faster than the standard. This is in part because of the number of messages the standard must process, unlike ours.

Table 2
Parameters for the second experiment.

Variable	Range
Number of network services	215
VNF Components per service	1–13
VNFs per orchestrators	30
Random Delay	[1, 100]ms

and 17; in the first one, there are more inconsistencies when the delay is higher, unlike the latter. This would suggest that concurrent updates have a greater impact. However, we leave this for future work as pre- venting inconsistencies when concurrent updates take place means there must be a way to establish precedence, not currently captured by any algorithm.

The inconsistencies increase the number of redundant messages, as shown in Fig. 15. The standard sends about 7 times more messages than our proposed algorithm when multiple services as considered. For a single service reconfiguration, this factor is only 2 as shown in Fig. 19. Moreover, it can be seen that for services with few dependencies, our proposed algorithm sends almost the same amount of messages. For services with more than 9 dependencies, the standard sends more redundant messages due to inconsistencies. The amount of messages sent by both algorithms reflects on the time spent on the reconfiguration.

Based on the complexity analysis of our algorithm, we expect the

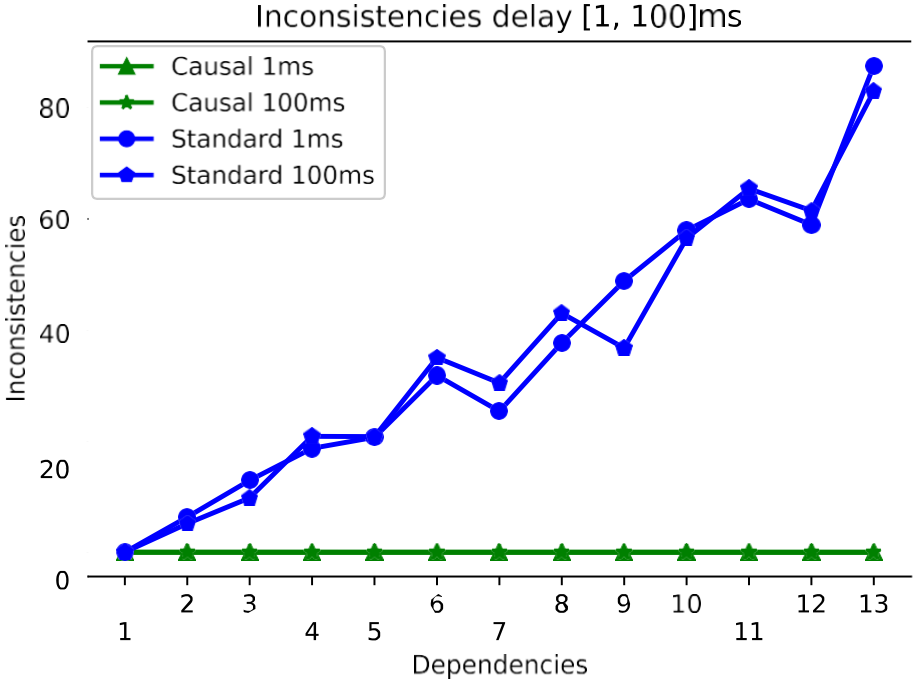

Fig. 17. Inconsistencies per dependencies. Our algorithm obtains zero inconsistencies. For the standard, the inconsistencies increase with more dependencies.

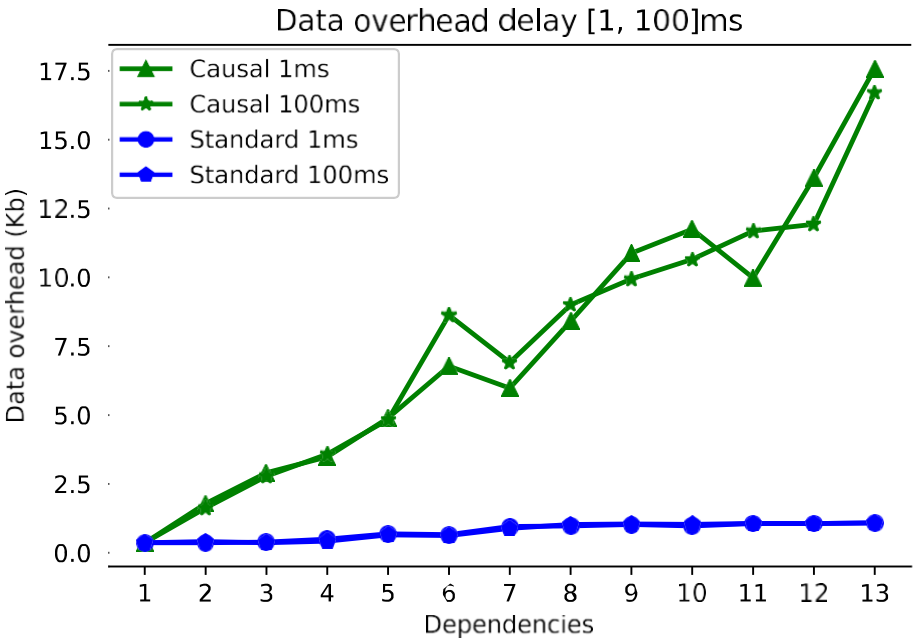

Fig. 18. Memory overhead per dependencies. For our algorithm the overhead increases dependencies. For the standard, the growth is below ours.

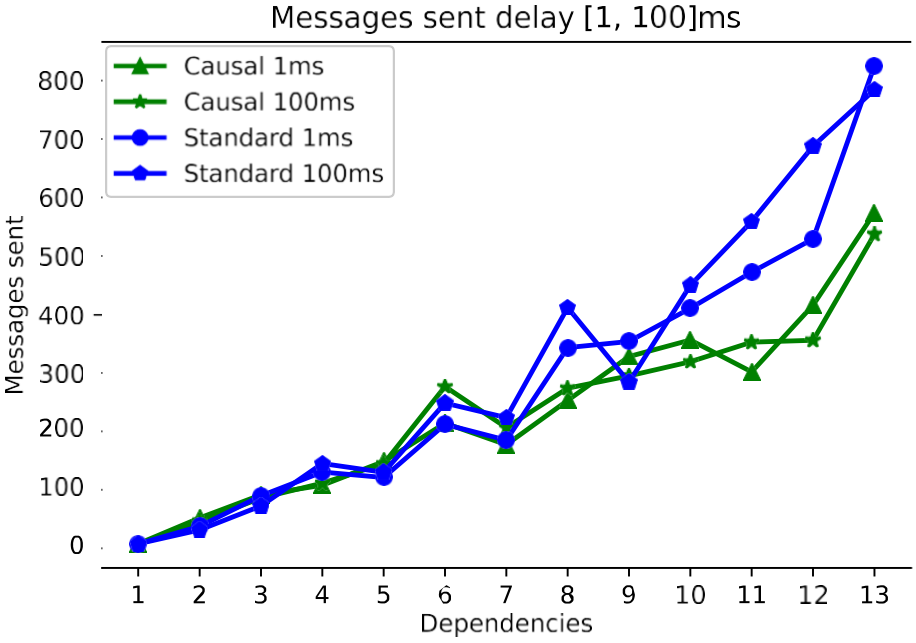

Fig. 19. Message overhead increases as a function of dependencies. For services with a higher number of dependencies, the standard behaves worst due to inconsistencies.

time for reconfiguration of our algorithm to be greater compared to the standard. However, Figs. 16 and 20 show that standard behaves worst as it takes about double the time compared to our proposed algorithm. This could be explained by the number of messages that need to be processed by the standard compared to our proposed algorithm. As previously mentioned, one effect of inconsistencies is that orchestrators send more messages to reconfigure a network service. This can be seen by analyzing Figs. 17, 19 and 20. With one dependency, the standard has no inconsistencies; consequently, the messages sent are the same as our proposed algorithm. The time is also the same. As the number of inconsistencies becomes greater, the disparity between our algorithmand the standard is also greater. Our algorithm by preventing inconsistencies reduces the time it takes for a reconfiguration. For ideal conditions (i.e. deterministic network conditions, one reconfiguration at a time), the ETSI standard would reconfigure faster than our proposed algorithm.

Our proposed algorithm prevents the inconsistency pattern for dependent reconfiguration by ordering and executing grants in the correct order; unlike the standard. Nevertheless, our algorithm has limitations. First, we assume a known set of orchestrators. This means

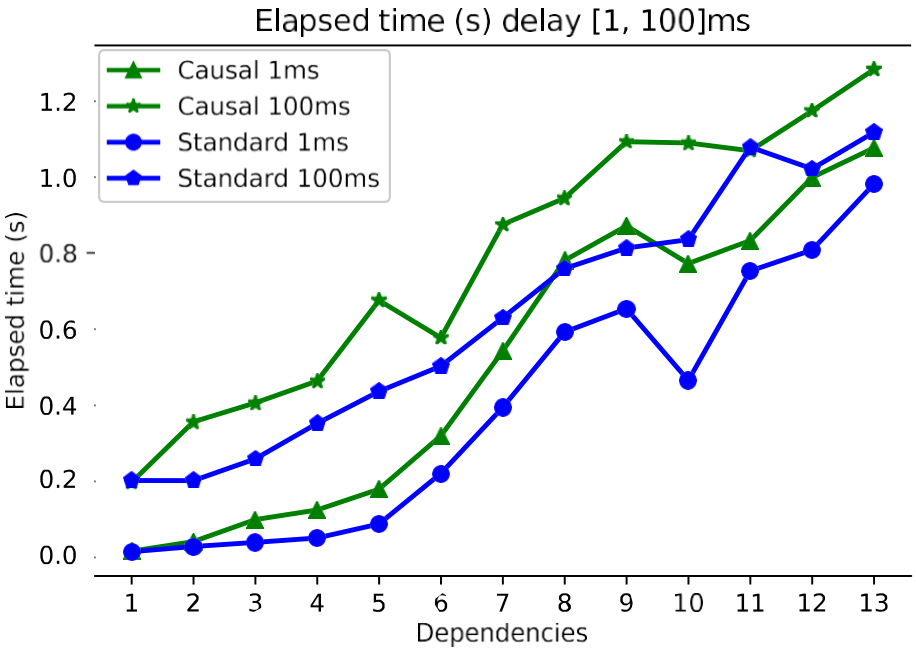

Fig. 20. Time for a dependent reconfiguration increases with more external dependencies.

that our algorithm only works in cooperative environments where the providers will share some information to coordinate with other orchestrators. Second, currently our algorithm stores causality informa-tion for both *dedicated* and *composite/shared* services. To reduce some of the redundant information we consider as future work optimizations such as detecting immediate causal relations to store less information and reduce the number of messages sent. Third, our algorithm supports only sequential reconfiguration, as the HBR relation does not capture concurrent events. In NFV, is possible to have concurrent reconfigurations for shared services. We leave for future work the management of such type of reconfiguration. Our proposed model and algorithm can apply to other lifecycle management operations of shared VNF-based network services such as healing, terminating, and monitoring. More-over, since we followed many of the ETSI standard guidelines to implement the orchestration algorithm, our work can be integrated to open source solutions that are ETSI compliant.

9. Conclusion

Reconfiguration of shared VNF-based network services must satisfy functional and non-functional dependencies to ensure the consistency of these services. This reconfiguration problem, known as NFV Dependent Reconfiguration, was addressed in this paper, following a distributed approach to guarantee consistency in NFV Dependent Reconfiguration. We defined, implemented, and evaluated a multi-domain model that identifies inconsistency patterns for dependent reconfiguration and a causally consistent distributed orchestration algorithm based on this model. The model identifies and prevents inconsistencies, which reduces the cost for service providers by coordinating orchestrator' s activity through multicast messages. The algorithm enforces a causal order for reconfiguration operations to satisfy dependencies in-network services. We compared our approach to the current ETSI-NFV reconfiguration standard. Both algorithms were applied and evaluated using a case study for the scaling of network services in a distributed multi-domain federation. We showed that our approach prevents inconsistencies while reconfiguring services by capturing only the relevant events to ensure a causal order. Hence, service providers can set up complex and shared network services using distributed orchestrators. However, preventing inconsistencies comes at a price reflected in the greater overhead of our solution compared to the ETSI/NFV standard by a linear factor. Moreover, our solution focuses on closed environments assuming trustful participants. In future work, we will research more refined approaches to capture only immediate causal relations to reduce the time and message overhead; also, we will explore open federations where orchestrators can join or leave on the fly. Our approach can be extended to other operations in the reconfiguration, such as healing, terminating, and updating shared network services.

Author statement

Josué Castañeda: Conceptualization. Investigation, Implementation, Writing, Proving, Writing − original draft preparation. Saul E. Pomares Hernández: Reviewing, Writing, Proving. Sami Yangui: Supervision, Reviewing and Editing. Julio C. Pérez Sansalvador: Reviewing, Editing. Lil M. Rodríguez Henríquez: Reviewing, Editing. Khalil Drira: Supervision, Reviewing and Editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The research presented in this paper is supported by the Mexican Council for Science and Technology CONACYT (Grant 708000) and LAAS-CNRS.

A. Appendix.

Algorithm 1
Request service scale

```
   Input: Composite network service service
1          Orchestrator myOrchestrator
2  myOrchestrator[vectorClock].increment();
3  myOrchestrator[pendingOperations].append(service);
4  myClock ← myOrchestrator[vectorClock];
5  myID ← myOrchestrator[id];
6  foreach dependency in service[dependencies] do
7  |  newOrch ← dependency[orchestrator];
8  |  newID ← dependency[id];
9  |  send(newOrch,GrantLCM(newID, myClock));
10 end
11 foreach externalOrchestrator in orchestrator[externalOrchestrators] do
12 |  send(externalOrchestrator, NotificationLCM(myClock,myID));
13 end
```

Algorithm 2
GrantLCM

```
   Input: Composite network service service
1          Sender's vector clock senderVC
2          Orchestrator myOrchestrator
3  myClock ← myOrchestrator[vectorClock];
4  myPendingOperations ← myOrchestrator[pendingOperations];
5  myID ← myOrchestrator[ID];
6  if compare(myClock, senderVC) ≤ 1 then
7  |  otherOrchestrator ← service[orchID];
8  |  myClock[otherOrchestrator] += 1;
9  |  if validateScaling(service) then
10 |  |  serviceID ← service[ID];
11 |  |  myPendingOperations.append(serviceID);
12 |  |  myOrchestrator[vectorClock][myID] += 1;
13 |  |  if allDependenciesInternal(service[dependencies]) then
14 |  |  |  scaleAllInternalDependencies(service);
15 |  |  end
16 |  |  scaleExternalDependencies(service);
17 |  end
18 |  send(otherOrchestrator, NotifyFail(myClock, myID, serviceID));
19 |  doPendingOperations();
20 end
21 myPendingOperations.append(service, senderVC);
```

The orchestrator scales a component (e.g. VNF, network service) as follows: First, it validates the internal logic and policies of the request scale while ensuring the scaling will not violate the service's SLA. Then, the orchestrator updates his vector clock, stores the request as a pending operation if the service has external dependencies, and sends the respective grant or scale request to other orchestrators as shown in Function 1; otherwise, all internal dependencies are scaled as shown in Function 2. Finally, the orchestrator notifies all other orchestrators in the federation to enforce the causal delivery of messages. A chain of scaling is created when dependencies of service have external dependencies themselves.

The dependency sends a *ScaleConfirmation* message to the orchestrator once scaling has finished. Once the message is delivered, the orchestrator executes Function 3 as follows: First, the orchestrator checks if the scaling confirmation relates to a pending operation and waits to receive all external confirmations. Then, after waiting for all internal dependencies scale as this ensures all-or-nothing scaling. Finally, the orchestrator acknowledges the sender of the scaling request by confirming everything went fine. However, if the pending operation is local, only the scaling takes place.

Function 4 is the most complex of all functions. First, it evaluates if there is at least a single operation that can be executed when the difference of vector clocks of the operation and the current clock is one. If it is the case, it validates the operation by checking if the request has the correct permissions and resources. In case of a valid operation, it checks the dependency type of the service or VNF referenced by the request. In case all dependencies are internals (usually only VNFs) it calls Function 2. For external dependencies Function 1 is called.

.

Function 1
scaleExternalDependencies

```
   Input: Composite network service service
1          Orchestrator myOrchestrator
2  myID ← myOrchestrator[ID];
3  serviceID ← service[ID];
4  myClock ← myOrchestrator[vectorClock];
5  inDependencies ← myOrchestrator[internalDependenciesToScale];
6  originalServiceID ← service[originalService];
7  myOrchestrator[pendingOperations][serviceID][originalServiceID] ← list();
8  foreach dependency in service[dependencies] do
9      dependencyID ← dependency[ID];
10     if dependency[type] == VNF then
11         inDependencies.append(dependency)
12     end
13     else
14         otherOrchestrator ← dependency[orchestratorID ];
15         originalDependencyServiceID ← dependency[originalServiceID] ];
16         send(otherOrchestrator, GrantLCM(dependencyID, myClock, originalDependencyServiceID))
17     end
18 end
```

Function 2
scaleInternalDependencies

```
   Input: Composite network service service
1          Orchestrator myOrchestrator
2  myID ← myOrchestrator[ID];
3  serviceID ← service[ID];
4  originalService ← service[originalService];
5  foreach dependency in service[dependencies] do
6      send(dependency, Scale(myID, serviceID, originalService ));
7  end
```

Function 3
scaleConfirmation

```
   Input: VNF Component ID vnfcID
1          Composite Network Service service
2          Original Orchestrator originalOrchestrator
3          My Orchestrator myOrchestrator
4  currentOperation ← orchestrator[pendingOperations][service[id];
5  pendingOperations ← currentOperation[pendingOperations];
6  myPendingOperations ← myOrchestrator[pendingOperations];
7  if pendingOperations.isNotEmpty() then
8      pendingOperations.remove(vnfcID);
9      if pendingOperations.isEmpty() then
10         myPendingOperations.remove(currentOperation);
11         if myOrchestrator.isExternalDependency(currentOperation) then
12             send(originalOrchestrator, ScaleConfirmation(currentOperation);
13             return
14         end
15         if myPendingOperations.isEmpty() then
16             scaleAllInternalDependencies(currentOperation)
17         end
18     end
19 end
```

Function 4
doPendingOperations

```
Input: Orchestrator myOrchestrator
clockUpdated ← True;
myClock ← myOrchestrator[vectorClock];
myID ← myOrchestrator[ID];
myPendingOperations ← myOrchestrator[pendingOperations];
myIndex ← myOrchestrator[orchestratorID];
otherOrchestrator ← service[orchestratorID];
while clockUpdated do
    atLeastOneClockChanged ← False;
    for operation in orch[pendingOperations] do
        opClock ← operation[vectorClock];
        opIndex ← operation[orchestratorID];
        if compare(myClock, opClock) ≤ / then
            myClock[opIndex] += 1;
            service ← operation[service];
            serviceID ← service[ID];
            if validateScaling(service then
                myPendingOperations.append(service[ID]);
                myClock[myIndex] += 1;
                if areAllDependenciesInternal
                (service[dependencies]) then
                    scaleInternalDependencies(service);
                end
                scaleExternalDependencies(service);
            end
            send(otherOrchestrator, NotifyFail(myClock, myID, serviceID));
            myPendingOperations.remove(operation);
            atLeastOneClockChanged ← True;
        end
    end
    clockUpdated ← atLeastOneClockChanged ;
end
```

References

Adamuz-Hinojosa, O., Ordonez-Lucena, J., Ameigeiras, P., RamosMunoz, J.J., Lopez, D., Folgueira, J., 2018. Automated network service scaling in NFV: concepts, mechanisms and scaling workflow. IEEE Commun. Mag. 56, 162–169.

Antonopoulos, A., 2020. Bankruptcy problem in network sharing: fundamentals, applications and challenges. IEEE Wireless Communications 27, 81–87.

Arteaga, C.H.T., Risso, F., Rendon, O.M.C., 2017. An adaptive scaling mechanism for managing performance variations in network functions virtualization: a case study in an NFV-based EPC. In: 2017 13th International Conference on Network and Service Management (CNSM), Volume 2018-Janua. IEEE, pp. 1–7. https://doi.org/10.23919/CNSM.2017.8255982. https://ieeexplore.ieee.org/document/8255982/.

Baranda, J., Mangues-Bafalluy, J., Vettori, L., Martínez, R., 2020. Scaling composite NFV-network services. In: Proceedings of the International Symposium on Mobile Ad Hoc Networking and Computing. MobiHoc), pp. 307–308. https://doi.org/10.1145/3397166.3415277.

Baranda Hortiguela, J., Mangues-Bafalluy, J., Martinez, R., Vettori, L., Antevski, K., Bernardos, C.J., Li, X., 2020. Realizing the network service federation vision: enabling automated multidomain orchestration of network services. IEEE Veh. Technol. Mag. 15, 48–57.

Boudries, F., Sadouki, S., Tari, A., 2019. A bio-inspired algorithm for dynamic reconfiguration with end-to-end constraints in web services composition. Service Oriented Computing and Applications 13, 251–260.

Bouras, C., Kollia, A., Papazois, A., 2017. SDN & NFV in 5G: advancements and challenges. In: 2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN). IEEE, pp. 107–111. https://doi.org/10.1109/ICIN.2017.7899398. http://ieeexplore.ieee.org/document/7899398/.

Cano, L., Capone, A., Carello, G., Cesana, M., Passacantando, M., 2017. On optimal infrastructure sharing strategies in mobile radio networks. IEEE Trans. Wireless Commun. 16, 3003–3016.

Chen, X., Zeng, H., Wu, T., 2010. Decentralized orchestration with local centralized orchestration for composite web services. In: 2010 International Conference on Parallel and Distributed Computing, Applications and Technologies. IEEE, pp. 255–260. https://doi.org/10.1109/PDCAT.2010.16. http://ieeexplore.ieee.org/document/5704427/.

Cisneros, J.C., Yangui, S., Pomares Hernandez, S.E., Perez Sansalvador, J.C., Drira, K., 2020. Coordination algorithm for migration of shared VNFs in federated environments. In: 2020 6th IEEE Conference on Network Softwarization (NetSoft). IEEE, pp. 252–256. https://doi.org/10.1109/NetSoft48620.2020.9165333. https://ieeexplore.ieee.org/document/9165333/.

Duan, J., Wu, C., Le, F., Liu, A.X., Peng, Y., 2017. Dynamic scaling of virtualized, distributed service chains: a case study of IMS. IEEE J. Sel. Area. Commun. 35, 2501–2511.

el houda Nouar, N., Yangui, S., Faci, N., Drira, K., Tazi, S., 2021. A Semantic virtualized network functions description and discovery model. Comput. Network. 195, 108152.

Eramo, V., Ammar, M., Lavacca, F.G., 2017. Migration energy aware reconfigurations of virtual network function instances in NFV architectures. IEEE Access 5, 4927–4938.

Eramo, V., Cianfrani, A., Catena, T., Polverini, M., Lavacca, F., 2019. Reconfiguration of cloud and bandwidth resources in NFV architectures based on segment routing control/data plane. In: 2019 21st International Conference on Transparent Optical Networks (ICTON). IEEE, pp. 1–5. https://doi.org/10.1109/ICTON.2019.8840406. https://ieeexplore.ieee.org/document/8840406/.

Etsi, N., 2014. Etsi Gs Nfv-Man 001 V1. 1.1 Network Function Virtualisation (NFV); Management and Orchestration. https://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.0101_60/gs_NFV-MAN001v010101p.pdf.

Etsi, N., 2018a. Etsi Gs Nfv-Ifa 030 V3.2.1 Network Functions Virtualisation (NFV) Release 3; Management and Orchestration; Multiple Administrative Domain Aspect Interfaces Specification. https://docbox.etsi.org/isg/nfv/open/Publications_pdf/Specs-Reports/NFV-IFA030v3.2.1-GS-MultiDomain20MANO-spec.pdf.

Etsi, N., 2018b. Etsi Gs Nfv-Ifa 014 V3.4.1 Network Functions Virtualisation (NFV) Release 3; Management and Orchestration. Network Service Templates Specification. https://www.etsi.org/deliver/etsi_gs/NFV-IFA/001_099/014/03.0401_60/gs_NFV-IFA014v030401p.pdf.

Etsi, N., 2018c. ETSI GS NFV 003, V1.4.1 Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV. https://www.etsi.org/deliver/etsi_gs/NFV/001_099/003/01.04.01_60/gs_nfv003v010401p.pdf.

Etsi, N., 2018d. ETSI GR NFV-IFA 012 V3.1.1 Network Functions Virtualisation (NFV) Release 3; Management and Orchestration; Report on Os-Ma-Nfvo Reference Point - Application and Service Management Use Cases and Recommendations. https://docbox.etsi.org/ISG/NFV/Open/Publications_pdf/Specs-Reports/NFV-IFA012v3.1.1-GR-Os-Ma-Nfvo20refpoint-svcmgmtUCsl.pdf.

Etsi, N., 2019a. Etsi Gs Nfv-Ifa 010 V3.2.1 Network Functions Virtualisation (NFV); Management and Orchestration; Functional Requirements Specification. https://www.etsi.org/deliver/etsi_gs/NFV-IFA/001_099/010/03.02.01_60/gs_NFV-IFA010v030201p.pdf.

Etsi, N., 2019b. Etsi Gr Nfv-Ifa 028 V3.1.1 Network Functions Virtualisation (NFV) Release 3; Management and Orchestration; Multiple Administrative Domain Aspect

Interfaces Specification. https://www.etsi.org/deliver/etsi_gr/NFV-IFA/001_099/028/03.01.01_60/gr_NFV-IFA028v030101p.pdf.
Etsi, N., 2019c. ETSI GR NFV-REL 010 V3.1.1 Network Functions Virtualisation (NFV) Release 3; Reliability; Report on NFV Resiliency for the Support of Network Slicing. https://docbox.etsi.org/ISG/NFV/Open/Publications_pdf/Specs-Reports/NFV-REL010v3.1.1 GRResiliency20forNetworkSlicingreport.pdf.
ETSI, N., 2020. ETSI GS NFV-SOL 003 V3.3.1 Network Functions Virtualisation (NFV) Release 3;Protocols and Data Models; RESTful Protocols Specification for the Or-Vnfm Reference Point. https://docbox.etsi.org/ISG/NFV/Open/Publications_pdf/Specs-Reports/NFV-SOL003v3.3.1-GS-Or Vnfm20RESTfulprotocolsspec.pdf.
Fidge, C.J., 1988. Timestamps in message-passing systems that preserve the partial ordering. In: Proc. 11th Austral. Comput. Sci. Conf. ACSC '88), pp. 56–66.
Forecast, G., 2019. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2017–2022, vol. 2017, p. 2022. Update.
Ghaznavi, M., Shahriar, N., Kamali, S., Ahmed, R., Boutaba, R., 2017. Distributed service function chaining. IEEE J. Sel. Area. Commun. 35, 2479–2489.
Hnětynka, P., Plášil, F., 2006. Dynamic reconfiguration and access to services in hierarchical component models. In: Gorton, I., Heineman, G.T., Crnković, I., Schmidt, H.W., Stafford, J.A., Szyperski, C., Wallnau, K. (Eds.), Component-Based Software Engineering. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 352–359. https://doi.org/10.1007/11783565_27.
Hu, G., Li, Q., Ai, S., Chen, T., Duan, J., Wu, Y., 2020. A proactive autoscaling scheme with latency guarantees for multi-tenant NFV cloud. Comput. Network. 181, 107552.
Israel, A., Sepúlveda, A., Reid, A., Vicens, F., Salguero, F., de Blas, G., Lavado, G., Shuttleworth, M., Harper, M., Marchetti, M., V R, A.S., et al., 2019. OSM Release FIVE Technical Overview. https://osm.etsi.org/images/OSM-Whitepaper-TechContent-ReleaseFIVE-FINAL.pdf.
Jia, Y., Wu, C., Li, Z., Le, F., Liu, A., 2018. Online scaling of NFV service chains across geo-distributed datacenters. IEEE/ACM Trans. Netw. 26, 699–710.
Katsalis, K., Nikaein, N., Edmonds, A., 2016a. Multi-domain orchestration for NFV: challenges and research directions. In: 2016 15th International Conference on Ubiquitous Computing and Communications and 2016 International Symposium on Cyberspace and Security (IUCC-CSS). IEEE, pp. 189–195. https://doi.org/10.1109/IUCC-CSS.2016.034. http://ieeexplore.ieee.org/document/7828601/.
Katsalis, K., Nikaein, N., Edmonds, A., 2016b. Multi-domain orchestration for nfv: challenges and research directions. In: 2016 15th International Conference on Ubiquitous Computing and Communications and 2016 International Symposium on Cyberspace and Security. IUCC-CSS), pp. 189–195. https://doi.org/10.1109/IUCC-CSS.2016.034.
Kattepur, A., Georgantas, N., Issarny, V., 2013. QoS composition and analysis in reconfigurable web services choreographies. In: 2013 IEEE 20th International Conference on Web Services. IEEE, pp. 235–242. https://doi.org/10.1109/ICWS.2013.40. http://ieeexplore.ieee.org/document/6649584/.
Kazhamiakin, R., Pistore, M., 2006. Choreography conformance analysis: asynchronous communications and information alignment. In: Bravetti, M., Núñez, M., Zavattaro, G. (Eds.), Web Services and Formal Methods. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 227–241. https://doi.org/10.1007/11841197_15.
Kim, S., Han, Y., Park, S., 2016. An energy-aware service function chaining and reconfiguration algorithm in NFV. In: 2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W). IEEE, pp. 54–59. https://doi.org/10.1109/FAS-W.2016.24. http://ieeexplore.ieee.org/document/7789440/.
Lamport, L., 1978. Time, clocks, and the ordering of events in a distributed system. Commun. ACM 21, 558–565.
Leite, L.A.F., Ansaldi Oliva, G., Nogueira, G.M., Gerosa, M.A., Kon, F., Milojicic, D.S., 2013. A systematic literature review of service choreography adaptation. Service Oriented Computing and Applications 7, 199–216.
Li, G., Zhou, H., Feng, B., Li, G., 2018. Context-aware service function chaining and its cost-effective orchestration in multi-domain networks. IEEE Access 6, 34976–34991.
Liu, J., Lu, W., Zhou, F., Lu, P., Zhu, Z., 2017. On dynamic service function chain deployment and readjustment. IEEE Transactions on Network and Service Management 14, 543–553.
Liu, Y., Zhang, H., Chang, D., Hu, H., 2020. GDM: a general distributed method for cross-domain service function chain embedding. IEEE Transactions on Network and Service Management 17, 1446–1459.
Malandrino, F., Chiasserini, C.F., Einziger, G., Scalosub, G., 2019. Reducing service deployment cost through VNF sharing. IEEE/ACM Trans. Netw. 27, 2363–2376.
Mijumbi, R., Serrat, J., Gorricho, J.-L., Bouten, N., De Turck, F., Boutaba, R., 2016. Network function virtualization: state-of-the-art and research challenges. IEEE Communications Surveys & Tutorials 18, 236–262.
Mills, D., 1991. Internet time synchronization: the network time protocol. IEEE Trans. Commun. 39, 1482–1493.
Moo-Mena, F., Drira, K., 2007. Reconfiguration of web services architectures: a model-based approach. In: 2007 IEEE Symposium on Computers and Communications. IEEE, pp. 357–362. https://doi.org/10.1109/ISCC.2007.4381628 http://ieeexplore.ieee.org/document/4381628/.
Nadjaran Toosi, A., Son, J., Chi, Q., Buyya, R., 2019. ElasticSFC: autoscaling techniques for elastic service function chaining in network functions virtualization-based clouds. J. Syst. Software 152, 108–119.
Nanda, M.G., Chandra, S., Sarkar, V., 2004. Decentralizing execution of composite web services. ACM SIGPLAN Not. 39, 170–187.
Pham, T.-M., Chu, H.-N., 2019. Multi-provider and multi-domain resource orchestration in network functions virtualization. IEEE Access 7, 86920–86931.
Rahman, S., Ahmed, T., Huynh, M., Tornatore, M., Mukherjee, B., 2020. AutoScaling network service chains using machine learning and negotiation game. IEEE Transactions on Network and Service Management 17, 1322–1336.
Rosa, R.V., Silva Santos, M.A., Rothenberg, C.E., 2015. MD2-NFV: the casefor multi-domaindistributed networkfunctions virtualization. In: 2015 InternationalConferenceandWorkshopsonNetworkedSystems (NetSys). IEEE, pp. 1–5. https://doi.org/10.1109/NetSys.2015.7089059. https://ieeexplore.ieee.org/document/7089059/.
Salaün, G., Roohi, N., 2009. On Realizability and Dynamic Reconfiguration of Choreographies. http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=E5FDDFC44F6CEE355820E8A3CD8E1979?doi=10.1.1182.2156&rep=rep1&type=pdf.
Saraiva de Sousa, N.F., Lachos Perez, D.A., Rosa, R.V., Santos, M.A.S., Esteve Rothenberg, C., 2019. Network service orchestration: a survey. Comput. Commun. 142–143, 69–94.
Sarrigiannis, I., Ramantas, K., Kartsakli, E., Mekikis, P.-V., Antonopoulos, A., Verikoukis, C., 2020. Online VNF lifecycle management in an MEC-enabled 5G IoT architecture. IEEE Internet of Things Journal 7, 4183–4194.
Shin, M.-K., Choi, Y., Kwak, H.H., Pack, S., Kang, M., Choi, J.-Y., 2015. Verification for NFV-enabled network services. In: 2015 International Conference on Information and Communication Technology Convergence (ICTC). IEEE, pp. 810–815. https://doi.org/10.1109/ICTC.2015.7354672. http://ieeexplore.ieee.org/document/7354672/.
Subramanya, T., Riggio, R., 2021. Centralized and federated learning for predictive VNF autoscaling in multi-domain 5G networks and beyond. IEEE Transactions on Network and Service Management 18, 63–78.
Sun, G., Li, Y., Liao, D., Chang, V., 2018. Service function chain orchestration across multiple domains: a full mesh aggregation approach. IEEE Transactions on Network and Service Management 15, 1175–1191.
Taleb, T., Afolabi, I., Samdanis, K., Yousaf, F.Z., 2019. On multi-domain network slicing orchestration architecture and federated resource control. IEEE Network 33, 242–252.
Tong, R., Xu, S., Hu, B., Zhao, J., Jin, L., Guo, S., Li, W., 2020. VNF dynamic scaling and deployment algorithm based on traffic prediction. In: 2020 International Wireless Communications and Mobile Computing (IWCMC). IEEE, pp. 789–794. https://doi.org/10.1109/IWCMC48107.2020.9148479. https://ieeexplore.ieee.org/document/9148479/.
Vaquero, L.M., Cuadrado, F., Elkhatib, Y., Bernal-Bernabe, J., Srirama, S.N., Zhani, M.F., 2019. Research challenges in nextgen service orchestration. Future Generat. Comput. Syst. 90, 20–38.
Wang, G., Feng, G., Quek, T.Q.S., Qin, S., 2018. On fast slice reconfiguration. In: 2018 IEEE Global Communications Conference (GLOBECOM). IEEE, pp. 1–7. https://doi.org/10.1109/GLOCOM.2018.8648117. https://ieeexplore.ieee.org/document/8648117/.
Xu, R., 2020. Proactive VNF scaling with heterogeneous cloud resources: fusing long short-term memory prediction and cooperative allocation. Math. Probl Eng. 2020, 1–10.
Yang, B., Xu, Z., Chai, W.K., Liang, W., Tuncer, D., Galis, A., Pavlou, G., 2018. Algorithms for fault-tolerant placement of stateful virtualized network functions. In: 2018 IEEE International Conference on Communications (ICC). IEEE, pp. 1–7. https://doi.org/10.1109/ICC.2018.8422444. https://ieeexplore.ieee.org/document/8422444/.
Yangui, S., Glitho, R.H., Wette, C., 2016. Approaches to end-user applications portability in the cloud: a survey. IEEE Commun. Mag. 54, 138–145.
Yi, B., Wang, X., Li, K., Das, S.k., Huang, M., 2018. A comprehensive survey of network function virtualization. Comput. Network. 133, 212–262.
Yousaf, F.Z., Sciancalepore, V., Liebsch, M., Costa-Perez, X., 2019. MANOaaS: a multi-tenant NFV MANO for 5G network slices. IEEE Commun. Mag. 57, 103–109.

Josue Castañeda Cisneros received a B.S degree in computer engineering from the Universidad Auton' oma de Baja California, an M.S. degree in computer science from the Instituto Nacional de Astrofísica, Óptica y Electrónica. He is currently a Ph.D. student working with Professors Khalil Drira, Sami Yangui, and Saul Pomares at the Laboratory for Analysis and Architecture of Systems (LAAS) in Toulouse, France. His research at LAAS focuses on Distributed Multi-Domain Orchestration under Network Function Virtualization. His topics of interest include coordination of orchestrators in both open and close federations, respectably.

Saul E. Pomares Hernández received the Ph.D. degree in computer science and telecommunications from the Institute National Polytechnique de Toulouse, France. Since 1998, he has been researching in the field of distributed systems and partial order algorithms. He is currently a Professor with the computer science department, Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE), Puebla, Mexico. He is also an Honorary Researcher (Chercheur Affilié) with the Laboratory for Analysis and Architecture of Systems, CNRS, Toulouse, France.

Sami Yangui is an Associate Professor with Institut National des Sciences Appliquées (INSA), Toulouse, France. He is member of the CNRS LAAS research team. His research interests include distributed systems and architectures, service-oriented computing and Internet of Things. He is working on different aspects related to these topics, such as cloud/fog computing, network functions virtualization and content delivery networks. He is involved in different European and International projects, as well as, standardization efforts. He published several scientific papers in high ranked conferences and journals in his field of research. He is IEEE Member and he served on many program and organization committees of International conferences and workshops, as well as, guest editor in several journals such as Future Generation Computer Systems journal edited by Elsevier and IEEE Access.

Julio César Pérez-Sansalvador received the B.S. degree in Computer Science from the Benemérita Universidad Autónoma de Puebla, Mexico, in 2005, the M.S. degree in computer science from the Instituto Nacional de Astrofísica, Óptica y Electrónica, Puebla, in 2007, and the Ph.D. degree in applied mathematics from Manchester University, U.K., in 2016. He is currently a CONACYT Research Fellow with the Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE), Puebla, Mexico.

Lil María Rodríguez Henríquez received the Ph.D. degree from the Centro de Investigación y Estudios Avanzados del IPN, in 2015. She is currently a Research Fellow with the Consejo Nacional de Ciencia y Tecnología (CONACYT) commissioned to the Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE). Her recent work involves partial order algorithms. Her research interests include cryptography and distributed systems.

Khalil Drira received the Master degree in computer science from INP, Toulouse, in 1988, and the Ph.D. and HDR degrees in computer science from Universite Paul Sabatier Toulouse in 1992 and 2005, respectively. Since 1992, he assumes a full-time research position in CNRS, France. His research interests include cooperative network IoT services, platforms and applications. His research activity addresses topics in this field focusing on Software architectures and communication services. He continues to be involved in national and international conferences and journals. He serves as a member of the program journals in the fields of software architecture as well as IoT and Internet networks. He has also been an Editor of several proceedings, books, and journals.