AN EDGE ABSTRACTION LAYER ENABLING FEDERATED AND HIERARCHICAL ORCHESTRATION OF CCAM SERVICES IN 5G AND BEYOND NETWORKS

Mauro Femminella[1,2], Gianluca Reali[1,2]
[1]Dept of Engingeering, University of Perugia, Italy
[2]Consorzio Nazionale Interuniversitario per le Telecomunicazioni (CNIT), Parma, Italy

NOTE: Corresponding author: Mauro Femminella, mauro.femminella@unipg.it

Abstract *– This paper shows a flexible orchestration solution for deploying Cooperative, Connected, and Automated Mobility (CCAM) services in 5G and beyond networks. This solution is based on the concepts of federation and hierarchy of orchestration functions. The federated approach is leveraged to cope with the differentiated complexity operation when multiple network operators are considered, whereas the hierarchical approach addresses the issue of jointly orchestrating multiple edge platforms in the network of a single operator. In this complex orchestration architecture, the main contribution of this paper consists of the design and implementation of an Abstraction and Adaptation Layer (AAL) for edge clouds, a new component enabling a truly cooperative and coordinated orchestration between different edge systems, characterized by appreciable experimental performance in terms of latency.*

Keywords – Beyond 5G networks, CCAM services, delegation of orchestration decisions, federated orchestration, hierarchical orchestration

1. INTRODUCTION

In recent years, the development of hardware and software technologies that can be used in the design, implementation and management of mobile communication systems has produced new models for service implementation and deployment. Among these, the general model that is currently characterized by a great momentum is that linked to the use of Multi-access Edge Computing (MEC). Essentially, it consists in making use of the outermost portions of cloud networks to host services that can benefit from the proximity between their deployment site and the users who use them. This concept can be applied to a multiplicity of services, both fixed and mobile, as witnessed by an intense research activity, illustrated in a dedicated section of this paper.

This paper refers to the vehicular environment, which promises truly disruptive innovations compared to the services we commonly use [1][2]. In particular, we consider the so-called Cooperative, Connected, and Automated Mobility (CCAM) services. This restriction of the focus does not correspond to a simplification of the issues to be addressed. On the contrary, it significantly complicates them, as appears from the different research directions that have recently been identified, as illustrated in the background section.

In order to offer CCAM services to vehicles, we assume to rely on 5G and beyond (B5G) technologies. In a B5G network, we can identify the Radio Access Network (RAN), the core network (5GC), including both control plane and User Plane Functions (UPFs), and a transport network interconnecting them. In this architecture, an edge cloud offering MEC services is typically positioned close to the RAN, in order to offer latency bounded services to vehicles. To achieve the desired quality from the deployed (vehicular) MEC services, edge resources need to be managed by orchestration functions [4], which are the main focus of this paper. In fact, both network and computing resources in 5G/B5G are typically virtualized, and network functions are offered as Virtual Network Functions (VNFs) [39] running in both edge and core environments, since this improves flexibility and deployment agility. The entity managing the lifecycle of such VNFs is standardized by ETSI under the Network Function Virtualization (NFV) Management and Orchestration (MANO) initiative [38], and called NFV orchestrator (NFVO).

Since the main characteristic of the vehicular environment is mobility, this calls for general scenarios in which multiple network operators can be involved. In particular, this is unavoidable in cross-border scenarios, as in the so-called "corridors" defined in the framework of the trans-European transport network [34]. In particular, our work is framed in the project 5G CARMEN, which is pursuing the experimental validation of CCAM services through 5G networks over the European highway corridor connecting Bologna (Italy) to Innsbruck (Austria) to Munich (Germany), thus crossing three countries with one bordering the other two. The features that best describe the added value of our orchestration solution include federation in a multi-domain environment and hierarchical orchestration inside the domain of each Mobile Network Operator (MNO). In fact, since edge computing/MEC is characterized by the proximity of network and computing resources to the access network (i.e., the 5G RAN in this scenario), it is necessary to deploy multiple edge/MEC clusters inside the network of an MNO, each one close to a group of gNodeBs. ETSI defined a standard architecture for managing MEC resources, including an orchestration function [37].

In a B5G network, a further cloud deployment, with the relevant orchestration function, is typically required to manage private cloud resources used to run the 5GC, as well as those application services not having critical latency requirements and thus not executed at the network edge. This additional orchestration function may not only manage centralized cloud resources, but also orchestrate service execution in different edge clouds by interacting with MEC orchestrators, through a hierarchical orchestration architecture. Indeed, ETSI provided an architectural solution for orchestrating Network Services (NSs) composed of VNFs running in different domain [36], including also the relevant interface specifications [35]. It defines two roles: the composite orchestrator (NFVO-C), in charge of issuing requests, and the nested peer orchestrator (NFVO-N), offering services. In short, the NFVO-C invokes NS lifecycle management operations towards the NFVO-N. The resulting NS is a composite instance, including a number of nested ones. In principle, this solution can be used for both federated and hierarchical orchestration use cases. However, it makes use of a continuous RESTful request/response pattern for requesting/granting any (even trivial) operation, creating a really significant communication burden on peer NFVOs. In addition, current orchestrators, such as the well-known ETSI Open Source MANO (OSM) [41], do not still implement these functions and relevant interfaces. Finally, the ETSI solution may be critical for services needing latency bounded orchestration operations, such as the horizontal scaling of a VNF instance in another MEC node, possibly in another domain, due to the need of obtaining a grant from the NFVO-C for implementing local actions.

In order to address these challenges, the 5G CARMEN project proposes a flexible orchestration solution, based on the concepts of federation and hierarchy, and leveraging a suitable delegation of the orchestration decisions [30], so as to involve a high-level orchestration function only in critical tasks. In order to deliver CCAM services associated with the pilot of selected use cases, 5G CARMEN goes beyond the validation of functional and operational integrity for the orchestrated edges. It also assesses the contribution of the enabling components for cross-border and multi-domain edge service orchestration to reduce CCAM service interruption, latency and packet loss during automotive mobility.

Paper contribution: The contribution of this paper, framed in the overall 5G CARMEN architecture, is the design and performance evaluation of a component named Abstraction and Adaptation Layer (AAL) for edge clouds, implementing the functions to empower real world MEC orchestrators with federated and hierarchical capabilities. The AAL is an enabler for the 5G CARMEN orchestration architecture, allowing a truly cooperative and coordinated orchestration between different MNOs' edge systems for performance improvement in terms of latency and packet loss reduction. We present the guiding principles that have driven the design of the AAL, its detailed behavior and the exposed interfaces, as well as an experimental validation of its performance, executed on real edge nodes in different MNOs participating in the 5G CARMEN experimentation campaign.

This paper is organized as follows. Section 2 includes a global picture of the background research on the considered subject and related work in the field, especially those related to multi-domain and/or hierarchical orchestration. Section 3 illustrates the overall system architecture, focusing on the design of the AAL. Section 4 presents the results of an experimental campaign carried out on real nodes, and finally Section 5 reports our concluding remarks.

2. BACKGROUND AND RELATED WORK

2.1 Background material

The background research related to introducing MEC-based solutions in vehicular networks is huge. Although this section may not include all existing proposals as in a comprehensive review document, we identify the main research directions and mention significant contributions for all of them, in order to give readers a global picture of the state of the art on the subject.

Regarding the user plane, a considerable research effort has been devoted to ensure low latency and continuous delivery, through the introduction of compute and storage resources at the MEC of mobile networks, and proactive service migration. This is the key aspect for providing latency-sensitive vehicular applications. The paper [3] shows a proposal for reducing the latency observed by users accessing services deployed in MEC. It is based on migrating the service instances to the edge node that are located nearest to users. The proposed solution makes use of Convolutional Neural Networks (CNN) and genetic algorithms for mobility prediction. Similarly, in [6] the authors propose an orchestration algorithm for orchestrating a virtualized infrastructure accessible by a large number of users. Also in this case, prediction of the user movement is used to proactively place user instances in MEC. More sophisticated prediction, based on heterogeneous information including cartograpy and inference from a high volume of past measurements is proposed in [8] and [9]. The use of Recurrent Neural Networks (RNNs) is proposed in [11] to predict network-wide vehicle patterns in urban environments, using claimed similarity between trajectory prediction and language modeling. Again, [12] shows an algorithmic proposal based on CNN, modeling vehicle trajectories as images. They achieve interesting prediction results, showing also the capability of managing two-dimensional trajectories at different spatial scales.

In addition to properly estimating user trajectories to guide content migration, a lot of research has been done on the policies and algorithms for server migration and selection. An interesting comparison is shown in [27]. In addition to assessing the need for service migration in a vehicular environment, different metrics used for server selection are compared, namely distance-based MEC, load-based MEC and their combination.

Although the migration of services, or their status, is essential for their usage in a vehicular environment, the related functions may have a significant cost, as highlighted in [13]. The authors propose a mobility-aware online service placement, which aims at achieving a balance between latency and migration cost. The importance of this balance increases with the frequency of service migrations, the burden of which could degrade the perceived Quality of Experience (QoE).

A further aspect to be considered in the usage of MEC resources is energy efficiency. This issue is addressed in [16], where the authors propose a solution for offloading traffic from resource-constrained vehicles to a MEC platform and integrate MEC-enhanced vehicular networks with Non-Orthogonal Multiple Access (NOMA) technology for improving the efficiency of the available spectrum. Computation offloading combined with resource allocation has recently stimulated many research contributions aiming to optimize offloading decisions [20][21][22][23].

In operation, the situation is further complicated by the heterogeneity of the MEC environment [17][18][14]. Combining heterogenous infrastructures, resources, and technologies available in different MEC sites could determine significant challenges for defining orchestration policies in order to achieve and preserve the desired QoE over time.

The scope of the orchestration functions is another aspect determining the exploitability of research results in a complex real environment. This is indeed the aspect on which this paper provides the most significant research contribution. Some recent papers focus on providing orchestration services throughout multiple domains. The different approaches are characterized by the organization of the individual orchestrators that cooperate in delivering the overall service. The investigated solutions include hierarchical organization, cascade organization, or a distributed one. A comprehensive review of such proposals can be found in [19]. Our contribution to this problem includes service federation and possible delegation, that are shown to improve efficiency and flexibility of inter-domain orchestration functions.

In addition to contributing to the identification of technically valid solutions, it is necessary to make the proposals operational by making them compliant with orchestration reference standards.

In this regard, the strategic importance of resorting to 5G services for implementing advanced vehicular services is addressed by [15]. The authors highlight the superiority of 5G features in supporting Vehicle-to-Everything (V2X) networks in comparison with previous standards, such as IEEE 802.11p. They also observe that the suitable network support to vehicular applications, based on roadside unit connectivity, can significantly benefit from an automated discovery of MEC-based virtualized service components. Again with regard to widely recognized standards, the ETSI standardization activities identify the MEC features needed to support V2X applications and identify the requirements for including new features and functions [2]. In this regard, it is worth mentioning the technologies and related standards that are typically put in place in conjunction with MEC orchestrators to make them effective. In particular, SDN and NFV have been shown to be very effective to deploy management automation fuctions when dealing with delay-sensitive vehicular applications [4]. In [14], the authors depict a closed-loop lifecycle management of network services, and map their proposal over the ETSI NFV MANO architecture. This proposal includes MEC-oriented key features for network service and resource orchestration in vehicular networks. Similarly, the papers [25][26] include a contribution related to introducing monitoring functions for control and/or data planes into the ETSI NFV architecture, designed to enhance 5G services. Additional benefits deriving from SDN and NVF in operating vehicular networks include service customization. For this purpose, the paper [24] shows a proposal of introducing the so-called function-specific managers and service-specific managers, configured by VNF descriptors and NS descriptors. They are used to integrate MANO functions for orchestrating services and resources in a custom manner. As for the implementation of the VNFs, they can be realized not only using the classic approach based on Virtual Machines (VMs), but also in a containerized environment, managed by Kubernetes [52]. A computational efficient scheme to run applications, running on top of containerized settings, is serverless computing [53]. It could be an interesting option for deploying network functions and vehicular services in MEC nodes.

Finally, it is worth mentioning a comparison of two popular orchestration tools shown in [14]. An extensive performance analysis of Open Baton and Open Source MANO is shown. The comparison is based on instantiation delay, responsiveness, and isolation features.

2.2 Related work

When more than one MNO is involved, it is possible to envisage two main approaches.

In the first one, multiple MNOs federate together by using a top-layer orchestrator. It does not really orchestrate any resource, but only redirects service requests to one of the MNO networks, or builds virtualized connections between them in order to create a composite NS with VNFs running in the private clouds of multiple MNOs. Thus, the federation is built throught an additional, inter-MNO, vertical orchestration layer. In this way, it is possible to avoid any direct interaction between the NFVOs of the different MNOs, which communicate through standardized interfaces like ETSI GS NFV-SOL005 [40] only with the top layer NFVO [32]. For instance, this is the approach adopted in the 5G-EVE project [29]. In order to mask this complexity to (vertical) user of operators' cloud services, abstraction layers are commonly used to provide a simplified view of a service, the interconnections of its components, as well as the relevant descriptors [28].

The other approach is based on a horizontal federation [33], in which the NFVOs of different MNOs set up distributed NSs across their private clouds using the standardized ETSI solution, thus communicating on the so-called Or-Or reference point [35]. However, as already mentioned in the Introduction, this approach suffers from increased latency and excessive communication burden on peer NFVOs, and thus is difficult to adopt in practical settings.

In addition, when the supported services have strict requirements in terms of latency and packet losses, often MNOs resort to the MEC approach, implying a number of edge clouds distributed in the MNO's network. In turn, also this deployment raises a number of issues. A first and simplistic solution could be to deploy just a centralized NFVO, which is able to manage not only a central private cloud, but also multiple remote edge clouds. They are handled as multiple Virtualized Infrastructure Managers (VIMs), each one deployed in an edge cloud. A VIM is an entity responsible for controlling and managing the NFV infrastructure (NFVI). It consists of physical compute, storage, and network resources. However, this would increase the complexity of operations and management burden of the NFVO. As an alternative solution, the central private cloud and all the distributed edge clouds could be managed by a single VIM, in turn orchestrated by the central NFVO. Again, the communication and complexity burden required to the VIM to handle a set of distributed computing cluster would be not acceptable. This led ETSI to the definition of a self contained solution for MEC deployments, including its own orchestrator [37]. However, the ETSI solution does not solve the issue of deploying services made of components distributed on multiple edge nodes and a central cloud, which is instead a quite popular setting, especially in dynamic environments as those offering CCAM services.

The 5G CARMEN architecture addresses this issue, by proposing an architecture that adopts hierarchical orchestration within the infrastructure of an MNO, and a horizontal federation between different MNOs, as proposed in [31]. In addition, in order to solve the issue of horizontal federation realized through the Or-Or reference point, it adopts a form of delegation of orchestration decisions [30]. This delegation principle can be implemented both intra-domain, that is between the high level NFVO and the MEC NFVOs, and inter-domain, that is between the MEC NFVOs operated by different MNOs, through previously agreed policies at the Or-Or level.

3. ORCHESTRATION ARCHITECTURE

Since the access network is implemented through B5G technology, edge clouds have to be connected to 5G user plane and control plane functions. The design principles of the 5G CARMEN system architecture can be summarized as follows.

- *Optimized lifecycle management of distributed CCAM service instances*. The orchestrated 5G platform supports an automated Lifecycle Management (LCM) of NS instances, including operations such as dynamic service instantiation, scaling, migration, update/reconfiguration, and termination.

- *Hierarchical and distributed edge orchestration*. The orchestrated edge platform is capable of performing a flexible and agile service orchestration in a hierarchical and distributed manner, by deploying top-level service orchestrators in different administrative domains, and edge-level orchestrators in multiple edge domains within each MNO network. With such a setting, services and associated resources can be managed locally (i.e., in edge domains), but different orchestration layers collaborate to optimize the outcome of the orchestration operations. A design according to such objective is in line with existing solutions for end-to-end orchestration under the control of top-level orchestrators, while enabling direct edge-to-edge orchestration, which is considered of particular value for the automotive industry for its ability to reduce latency of orchestration.

- *Delegation of MANO operations in a federated environment*. In order to optimize the performance of the MANO operations, a key element is the introduction of the concept of Management Level Agreement (MLA) [30]. It enables the delegation of MANO tasks/operations between the top-level and edge-level orchestration systems (intra-domain), and between the peering edge platforms in the same and/or different domains (inter-domain). MLA enables the offloading of LCM operations from the top level to edge level orchestrators. Such a negotiated agreement determines the operations and functions that the edge-level entities are allowed to perform within their edge boundaries, thereby executing LCM operations on the relevant service applications and their respective resources *without asking for permission from the top-level entity*. The prerequisite for establishing cross-domain federation is an MLA negotiated between administrative domains, i.e., relevant top-level NFVOs. Developing federation also at the edge level enables the interworking of MEC platforms, to provide a cross-edge on-demand management and orchestration in a collaborative manner, while enabling and maintaining low-latency edge-to-edge CCAM service/session continuity.

- *Coupling of 5G UPF and MEC data plane.* Cooperation between the edge platform with the overall 5G architecture is mandatory for complete end-to-end system management and control. This allows enforcing data plane traffic rules aligned with policies and configurations associated with mobile subscribers in both the UPF and the MEC platform. This allows enabling (i) traffic steering within the MEC programmable data plane for execution of CCAM services, (ii) traffic forwarding towards a different MEC platform, and (iii) relocation of the UPF of a mobile subscriber due to vehicle mobility.

- *Slicing*. The orchestrated edges platform can host and manage services from different vertical players acting as tenants for the platform, and isolating them by adopting slicing techniques enforced by edge orchestration systems.

3.1 Overall system architecture

Fig. 1 shows the key components of the 5G CARMEN architecture. It extends and interfaces the MNOs' centralized service orchestration system (NFV-SO), which is in charge of implementing top-level orchestration functions, with the edge orchestration system. The latter is composed of the NFV Local Orchestrator (NFV-LO) and the MEC Application Orchestrator (MEAO), which are connected through the Mv1 reference point in alignment with ETSI GS MEC 003 specification [37]. The MEAO is responsible for the LCM of CCAM services running on the MEC hosts of the edge cluster, while the NFV-LO is responsible for the management of the VNFs hosted on the NFVI of the MEC platform. This edge orchestration system interfaces with the NFV-SO via the newly defined Mv1' reference point, which is an extension of the standard Mv1 reference point. In fact, as the MEAO takes an orchestration decision on CCAM services and communicates them to the NFV-LO via Mv1, in the same way the NFV-SO takes a high-level orchestration decision on deployed services and communicates them via Mv1' to the NFV-LO. The NFV-SO supports the Or-Or reference point for interconnecting and federating with other MNOs' NFV-SOs in alignment with the ETSI NFV-IFA 028 [36] and ETSI NFV-IFA 030 [35]. The NFV-LO executes local orchestration tasks following the NFV-SO's directives, which are defined in MLA [6] and exposed to the NFV-LO via the Mv1' reference point. The MLA is a concept that enables granting operational autonomy from a top-level management entity to an edge-level one. In the 5G CARMEN architecture, this concept of delegation is applied to resource orchestration. The MLA allows the NFV-SO granting some degree of autonomy in orchestration operations to the underlying NFV-LO. This means that some orchestration operations, such as scaling, can be carried out by the NFV-LO/MEAO without asking any permission from the controlling NFV-SO.

The way to implement the MLA concept consists of onboarding an MLA descriptor (a simplified example can be found in [30]) from the NFV-SO to the NFV-LO for each service type before instantiating any instance of the service itself. This descriptor is uploaded to the NFV-LO via a suitable interface over the Mv1' reference point. In this way, the NFV-LO is aware of the operations that it can execute autonomously and of those that need to be authorized by the NFV-SO on request, thus speeding up and easing orchestration tasks at the edge. This autonomy may extend also to include LCM operations carried out in cooperation with other edge nodes, both intra-domain and inter-domain, specifying for each service the permitted operations and the potential peer edges. In any case, the NFV-SO must be kept aligned with what happens in edge nodes. This means that a notification mechanism has to be implemented at the Mv1' reference point.

Finally, a new Lo-Lo reference point was introduced to enable the peering between NFV-LO instances, in order to enable direct and low-latency management of multi-domain and multi-site services in the edge domains, bypassing the top-level NFV-SO orchestrators, as specified by MLA. The design of the Lo-Lo reference point [31] inherits from the Or-Or reference point between the NFV-SOs [35].

The overall system architecture defined within the 5G CARMEN project is shortly described in what follows and illustrated in Fig. 1, focusing on orchestration aspects:

- NFV-SO: Represents the top-level orchestrator of the multi-tier orchestration system of the platform running CCAM services. The operational scope of this orchestrator includes the management of the entire virtualized infrastructure of an operator domain. In particular, it is responsible for the management and orchestration of application services from multiple tenants. It has the additional task of enabling federation with the NFV-SOs of other administrative domains. It maintains a global repository of the application packages and software images received through the northbound interface on the Os-Ma-nfvo reference point upon onboarding requests.
- NFV-LO/MEAO: The combination of the NFV-LO and the MEAO realizes the local edge orchestrator, which represents the second tier of the multi-tier orchestration system. The operational scope of this orchestrator includes the designated clusters of MEC sites. A 1:N relationship exists between the NFV-SO and the local orchestrator.

- The Edge Controller (EC) acts as a VNF manager (VNFM) for the management of the NFVI resources, on top of VIM, as per the ETSI MEC specification [37]. The EC represents a binding element between the technology specific edge platform (realized through Kubernetes clusters [52] in the 5G CARMEN) and the relevant orchestration layer, realized by NFV-LO/MEAO. The EC combines the features for edge platform management, VNFM, as well as additional control enablers, such as connectivity management of both edge platform applications (CCAM services) and Value Added Services (VAS), slice management, and coupling with the 5G system via 3GPP Naf reference point.
- CCAM services: service functions or micro-service instances running on the MEC Platform. Those services can be persistent, such as maneuvering services, or on demand, with situation-aware or dynamic mission-critical requirements.
- MEC Value-Added Services (VAS): services running on the MEC platform to provide value added functions to other CCAM services. Examples include the Radio Network Information Service (RNIS) [48], location services, as well as publish/subscribe AMQP broker, extensively used in 5G CARMEN experiments to distribute vehicular data to services.

3.2 Abstraction and Adaptation Layer (AAL): functional view

The NFV-SO is the orchestrator that runs in the core of the MNO in order to manage and orchestrate the application services for the MNO's tenants at a global infrastructure level. The NFV-SO manages and coordinates the LCM of the CCAM services deployed in the edge system by interacting with the edge level orchestration system (NFV-LO/MEAO) over interfaces defined over the Mv1' reference point. With the information retrieved through this reference point, the NFV-SO has an abstract view of the orchestrated 5G edge platforms running below.

The Mv1' reference point on the MEC side is managed by the Abstraction and Adaptation Layer (AAL) depicted in Fig. 2. It is the intermediate layer that enables the communication among orchestrators implementing incompatible APIs. It is deployed on the top of the NFV-LO, abstracting its features and making it compliant to the standard ETSI GS NFV-SOL005 [40], to simplify the operations at the NFV-SO level. Thus, it is the enabler of the orchestration functional split, allowing the NFV-SO to be unaware of the underlying NFV-LO proprietary APIs. This abstraction is provided by means of an Adaptation and Abstraction Module (AAM).

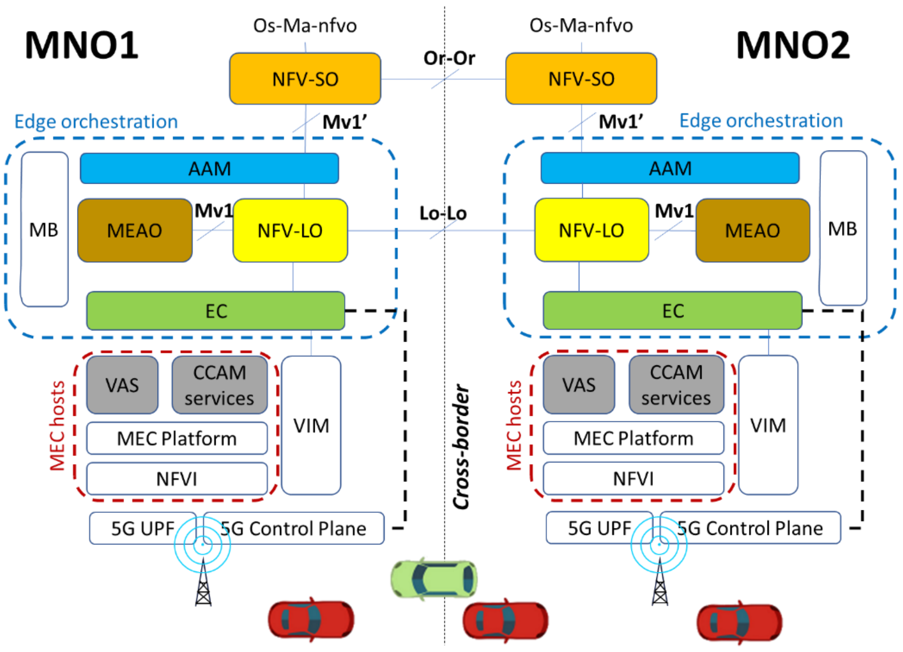


Fig. 1 – 5G CARMEN system orchestration architecture

It masks the features and semantics of the orchestrators running in the edge orchestration layer, i.e., the NFV-LOs, and runs in the edge sites together with it. This enables the NFV-SO to carry out orchestration operations that are independent and transparent to those specific of the MANO platform running in edge nodes. Thus, the AAM performs the necessary adaptation of the interfaces over the Mv1' reference point between the different types of the orchestration system employed at the NFV-SO and the NFV-LO/MEAO. As mentioned above, the north-bound interface of the AAM over the Mv1' reference point is compliant with the ETSI GS NFV-SOL005 interface specification [40]. The abstraction layer was developed to support not only OSM, which (partially) implements the ETSI NFV-SOL005 interface as northbound API [41], but also other, possibly cloud-native, NFV-LO APIs. In more detail, in the AAM we developed an adaptor also for lightMANO [49], a lightweight, standard compliant ETSI NFVO designed to nativelly run in a Kubernetes environment [52].

Adopting a standard interface is a clear advantage for decoupling the development of NFV-SO from that of NFV-LOs, enabling to deploy different NFV-LOs in different MEC nodes. In order to do so, the AAM exposes a REST Application Programming Interface (API) as Northbound API (NBI) on the Mv1'. On this API, the AAM accepts and manages the mandatory fields defined in the ETSI NFV-SOL005 APIs towards the NFV-SO. On the other end, on the AAM SBI, it is compliant with the NBI exposed by the underlying NFV-LO, which can be OSM or lightMANO. Thus, the AAL is a functional entity accomplishing multiple tasks. First, it provides an adaptation between the REST call received on its NBI and those available on the NBI on the underlying NFV-LO, mapping different API URLs between the two interfaces and adapting the format of messages, when needed. For instance, in the case of OSM, some NFV-SO REST calls received on the Mv1' reference point will be relayed almost directly to the OSM, whereas for lightMANO there is the need of REST calls translation to completely new calls, with a mapping between different parameters of message body. In addition, in some cases the AAM also hides to the NFV-SO the details of how specific operations are implemented, providing an abstract view of the NFV-LO operations, and implementing some functions when they are not provided at all by the underlying NFV-LO. For instance, both OSM and lightMANO do not support REST notifications, which is instead defined in the ETSI GS NFV-SOL005 standard and necessary to support orchestration operations in multi-domain scenarios. Thus, when the underlying NFV-LO is OSM, the AAM is in charge of retrieving NS-related information through repeated polling on the OSM NBI. When dealing with lightMANO, the AAM has to subscribe to an AMQP Message Broker (MB) to retrieve notifications about instantiated services on the lower layer. In turn, a subset of this information, retrieved via polling or subscription to an MB, has to be forwarded to the NFV-SO according to the agreed notification filters, acting as a client node contacting the NFV-SO on its callback URI, as specifed on ETSI GS NFV-SOL005 [40]. Thus, the presence of the AAL allows decoupling the design of the orchestrator used as the NFV-LO from its adoption in the overall orchestration architecture used in 5G CARMEN.

Finally, the AAL enables the enforcement of the MLA descriptor from the NFV-SO to the underlying NFV-LOs, enabling the formalization of autonomous operations that can be executed by the NFV-LO without affecting upper layers for any further permission requests. The delegation allows also the NFV-LO to execute NS instantiation/termination operations on cross-border domains, establishing the horizontal Lo-Lo interface with relevant peer NFV-LOs. This descriptor includes also the definition of the scaling policies of each NS instance.

To sum up, the usage of OSM as the underlying NFV-LO is intended for orchestrating a core MNO cloud, whereas a simpler and lightweight NFV-LO, such as lightMANO, are designed for usage on edge nodes. The combination of NFV-SO and AAM allows managing a hierarchy of generic NFVOs, each one with its target usage environment. If some functions are not provided by the underlying NFV-LO, it is the AAL that takes care of them.

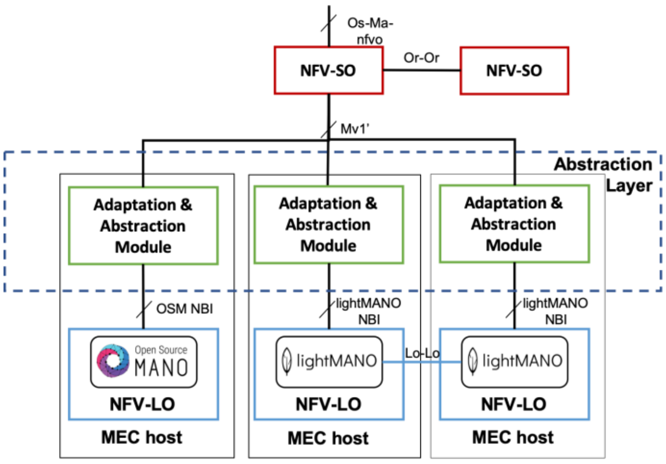


Fig. 2 – The AAL role in intra-domain operations

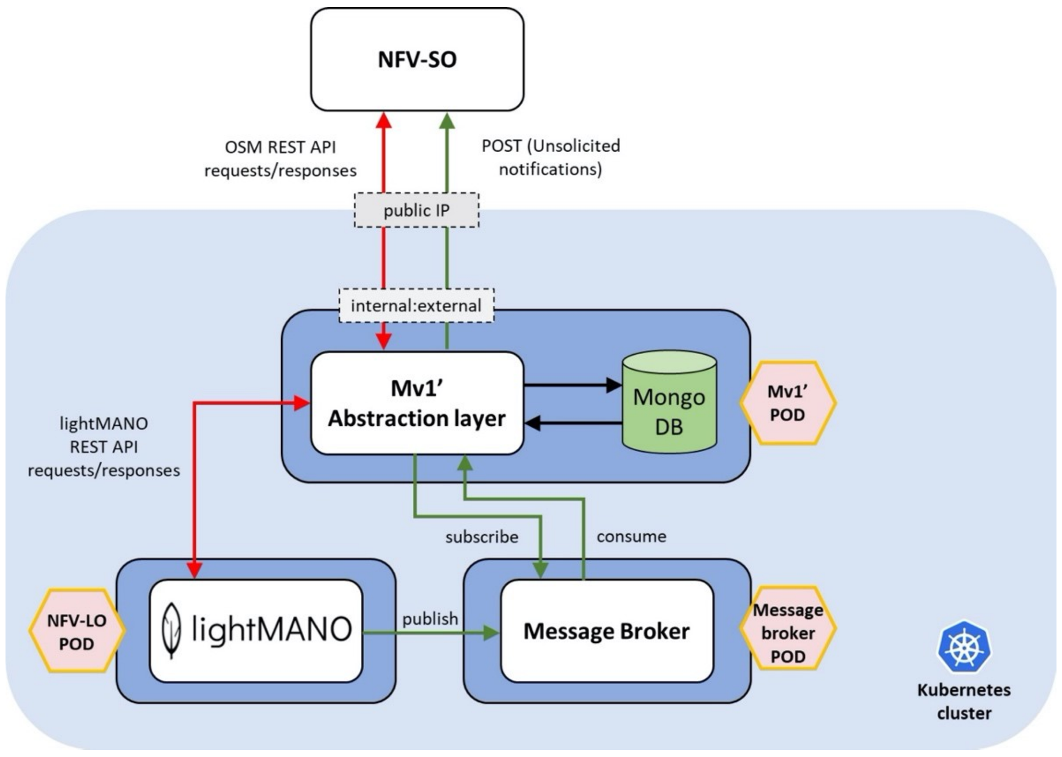


Fig. 3 – Scheme of the edge entities composing the orchestration function split running on Kubernetes cluster.

From a functional point of view, the abstraction layer relies on a *persistence layer* to abstract the functionalities specified by the ETSI standard and not implemented in the underlying NFV-LO. The persistence layer is provided by deploying a dedicated database along with the Mv1' adaptation layer (Fig. 3). Below we report the main abstract operations that rely on the persistence layer:

- ETSI VNF/NS descriptors onboarding and querying, a feature not necessarily implemented in the underlying NFV-LO. For instance, lightMANO is not compliant with ETSI/OSM descriptors and instead it uses a helm-chart to describe orchestrated services. Thus, onboarded descriptors are stored in the Mv1' internal DB and the parameters defined therein are processed to complete some preliminary tests, such as the availability check of the relevant helm-chart on the NFV-LO side and for configuring the NS instantiation. In the end, the descriptors onboarding is a feature exposed by the abstraction layer on behalf of the NFV-LO. Since the latest versions of OSM support the usage of Kubernetes cluster to implement VNFs (KNFs), we used this option in our descriptors to ease the mapping towards helm-charts used by lightMANO.
- NS creation/deletion operations are exposed towards the NFV-SO, which is expecting at first to create the NS, and then to instantiate it, as defined by OSM/ETSI NFV-SOL005 APIs. This operation is not supported by lightMANO, which instead instantiates directly its applications (Apps). Thus, the creation operation consists in the creation of the NS entry in the internal database, according to the body parameters enclosed in an OSM compliant request. A unique ID is then returned to the calling NFV-SO, enabling the lifecycle management of the created resource. Similarly, the NS deletion is an abstract feature that involves the Mv1' database only, avoiding the interaction with the lower layers. The removal of the entity from the DB is allowed on the NOT_INSTANTIATED NS only, which means that the resource is not running anymore on the lower layers. Thus, the AAM has to implement a state machine for handled NS instances. In addition, since lightMANO does not support services made of multiple apps, it has to keep the mapping between the VNFs composing a service and the relevant NS.
- MLA onboarding: The AAM stores locally and then forwards the MLA descriptor to the NFV-LO. In case of an NFV-LO that is not compliant

with the MLA, such as OSM, the AAM will manage the relevant permissions towards the NFV-LO underneath.

3.3 AAL supported operations

The Mv1' reference point is exposed by the AAM and instantiated between each NFV-LO and its "parent" NFV-SO. Its scope is limited within the same operator's domain. However, in case the underlying NFV-LO cannot handle the Lo-Lo interface, the extension to support it and the relevant MLA permissions on behalf of the edge orchestration underneath would be straightforward. This reference point is primarily designed to:

- Enable the interaction between multiple decentralized orchestration functions. This implies North-South (and potentially East-West) interactions in case of orchestrators' hierarchy.
- Support the provisioning of management autonomy to lower orchestration domains of an orchestration hierarchy.
- Support the LCM of services in a multi-domain environment.

The Mv1' reference point is unique to the orchestrated platform for CCAM, thus an appropriate initialization is required to implement the orchestration functional split. This procedure is called bootstrap in the following list.

The Mv1' reference point is primarily used for the NS/VNF package management between the global package repositories and the local package repositories, as well as for carrying out the main NS LCM operations in the 5G edge platform:

- *Create/Instantiate NS*: ETSI NFV-SOL005 compliant operation, leading to the creation of an NS instance.
- *Scaling NS*: request to scale an already instantiated NS, likely on a different edge site.
- *Terminate/Delete NS*: ETSI NFV-SOL005 compliant operation, leading to the deletion of an NS instance.
- *Notification of NS instance updates*: ETSI NFV-SOL005 compliant notifications towards the NFV-SO to update it about some parameters related to the running instances.

In addition, the Mv1' is the reference point over which the MLA parameters are negotiated to determine the scope of the management autonomy that the NFV-SO can delegate to the NFV-LO. Moreover, this reference point also exposes interfaces that enable the NFV-SO not only to monitor the performance and fault events of the resources within the NFV-LO domain, but also to monitor the compliance of the MLA agreement. It should be noted that the NFV-SO has full administrative access of the entire management domain and can support and/or overrule the management decisions of the NFV-LO.

3.3.1 Bootstrap

The Bootstrap operation is the very first operation to be executed after the AAM initialization (see Fig. 4). At this time (message 1), the AAM knows only the public endpoint of the NFV-SO and the endpoint of the local MB, both defined in its configuration file.

At the beginning of the bootstrap phase, the AAM forwards a subscription request to some specific topics on the local MB. Altough this operation is part of the bootstrap phase, it is not strictly related to the message chain described below. This means that it can be initialized at any time without affecting the vertical registration sketched in Fig. 4. The underlying NFV-LO, performing a periodic polling, detects the AAM availability and then transmits the registration message (message 2) in order to provide relevant information about its identity, its network location (i.e., the endpoint in the Kubernetes cluster), and the topological coordinates of the radio coverage it manages in GeoJSON format.

The AAM forwards this information towards the NFV-SO (message 3), providing its public endpoint in order to allow the NFV-SO to orchestrate the underlying cluster below the Mv1' interface. The NFV-SO's response (message 4) consists of its unique ID. Provided information will be processed by the NFV-SO to compile the MLA descriptor that will be uploaded in a subsequent phase. Finally, the AAM pushes the NFV-SO ID towards the NFV-LO (message 5) in order to complete the vertical registration phase.

As a final operation, which is not strictly related to the registration phase, the NFV-SO transmits a subscription on the Mv1' reference point in order to receive unsolicited notifications about instantiated NSs. The request body contains subscription filters, in order to define only a subset of messages, and the callback URI that should be used by the AAM for the transmission of relevant notifications. This functionality is essential to enable low latency orchestration operation via Lo-Lo reference point.

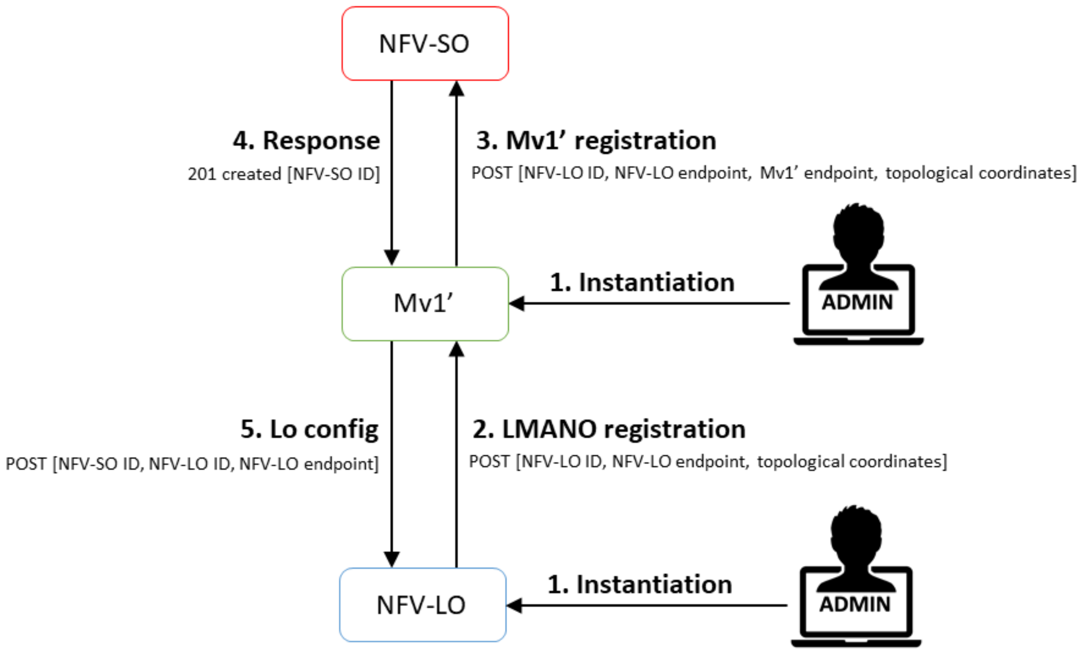

NFV-SO
4. Response
201 created [NFV-SO ID]
3. Mv1' registration
POST [NFV-LO ID, NFV-LO endpoint, Mv1' endpoint, topological coordinates]
Mv1'
1. Instantiation
ADMIN
5. Lo config
POST [NFV-SO ID, NFV-LO ID, NFV-LO endpoint]
2. LMANO registration
POST [NFV-LO ID, NFV-LO endpoint, topological coordinates]
NFV-LO
1. Instantiation
ADMIN

Fig. 4 – Bootstrap and registration of vertical layers

3.3.2 MLA onboarding/query/deletion

Since the MLA is not defined in the ETSI standard, the AAM API was extended to support it. The MLA descriptor, formatted by using the JSON encoding, is pushed by the NFV-SO on the Mv1' towards the AMM, which forwards the descriptor to the NFV-LO. The MLA is also stored in the local AAM DB for handling the other NS-related operations. Any further upload will result in an update of the MLA on both AAM (internal DB) and NFV-LO.

In the descriptor retrieval operation, the NFV-SO sends the information request to the AAM, which forwards the request towards the NFV-LO, upon proper translation of the request URI. The response body returned by the NFV-LO is forwarded via Mv1' towards the NFV-SO. The response body consists of an array of MLA descriptors in JSON format.

As for deletion, the NFV-SO triggers this operation on the Mv1' reference point. In turn, the AAM forwards the request to the NFV-LO. If no errors occur, the AAM assumes that the NFV-LO has successfully removed the content, and removes the corresponding entry from its internal DB, providing proper confirmation to the NFV-SO.

3.3.3 Descriptors onboarding/query/deletion

The onboarding operation of both VNF/NS descriptors (VNFD/NSD) is an abstracted feature exposed by the AAM, as it is not implemented in the NFV-LO used in 5G CARMEN edges (i.e., lightMANO). It basically consists of a file upload, as defined in the ETSI NFV-SOL005 APIs, by using standard endpoints. Onboarding is a two-step operation. First, the NFV-SO uploads the VNFDs and then the related NSD. A preliminary check is performed in order to verify whether the helm-chart defined in the VNFD under the KDU section (Kubernetes Deployment Unit) is available on the NFV-LO side. Thus, an OSM-compliant onboarding request triggers a GET request on the NFV-LO NBI. If no errors occur, the VNFD can be stored as JSON object in the local DB.

As mentioned above, the second step consists of the NSD file upload, done in a similar manner as described in the first step. The AAM searches among the onboarded VNFDs for those defined in the NSD body. This is an internal operation that does not involve the NFV-LO. If a match for all VNFs included in the NS is found, the NSD can be stored in the local DB. For each successful NS onboarding the AAM returns a unique ID to the NFV-SO in order to properly address each descriptor on the Mv1' side. This ID is required for the NS creation and instantiation phases and for any other operation involving the descriptors (query or deletion).

The descriptor retrieval is an abstracted feature implemented on the Mv1' side only through HTTP GET messages compliant with ETSI NFV-SOL005 for both VNF and NS descriptors. In both cases, the information returned via Mv1' are collected from the internal DB and are compliant to the ETSI standard format.

The descriptors cancellation is an abstract feature implemented on the Mv1' side only. For a complete removal of a VNFD descriptor, the NFV-SO has to trigger the removal of all the NSDs that include such a descriptor, specifying in both cases their IDs. The AAM then removes the descriptor matching the provided ID from the local DB.

3.3.4 NS creation

The NS creation operation enables the creation of an NS instance towards the AAM providing mandatory parameters for the deployment phase on the NFV-LO. Note that the creation operation is not implemented in lightMANO. It is an abstract feature exposed by the Mv1' only and consists of validating the OSM mandatory parameters provided in the request body:

- *nsName*: the NS instance name should be unique in the local DB. Any duplication will cause the rejection of the current request;
- *nsdId*: the ID provided should match an already onboarded NSD in the local DB;
- *vimAccountId*: this parameter, coming from the OSM APIs' definition, was overwritten to represent the ID of the target NFV-LO;
- *additionalParamsForNs*: it is a data structure containing some instantiation parameters used by the underlying layers to properly deploy the applications on the Kubernetes cluster.

In case no errors occur, the NS instance can be stored in the local DB, setting its internal state to NOT_INSTANTIATED. Thus, the resource was created and it is ready for the instantiation at any time, as specified by ETSI NFV-SOL005. An NS instance ID (*nsInstanceId*) is then generated and returned to the NFV-SO, in order to enable subsequent LCM operations.

3.3.5 NS instantiation

The NS instantiation is a two-step operation, according to the ETSI NFV-SOL005 APIs. Initially, the NS instance is created (as shown in the previous section) and then it can be instantiated. Both operations are triggered by the NFV-SO and can be also executed at different times maintaining the message sequence defined above.

The ETSI NFV-SOL005 standard instantiation request, including in the request body the NS ID (*nsInstanceId*), the *nsdId*, the *vimAccountId*, and *additionalParamsForNS*, is mapped into the equivalent one on the lightMANO NBI. The request body extends the parameters provided in the creation request with some additional fields required for the app deployment in the underlying edge platform. Any errors will be detected and reported by the underlying layers towards the Mv1' and then up to the NFV-SO. The internal state of the NS instance is updated according to the result of the instantiation request.

3.3.6 NS scaling

The NS scaling operation is already present in the ETSI NFV-SOL005. In the proposed architecture, the scaling operation is one of those typically handled autonomously by the NFV-LO, as per the MLA definition. However, we implemented the possibility to scale an existing NS by scaling one of its constituents VNFs on a different edge node, assuming to use the Lo-Lo reference point for the instantiation over a peer NFV-LO. Clearly, this is possible only if an MLA with such a permission is defined. This operation is analogous to the NS instantiation one, using the same set of parameters but with the *vimAccountId* set to the target remote NFV-LO.

3.3.7 Get NS information

The AAM exposes over the Mv1' reference point the operation for the NS instance information retrieval, defined by the ETSI standard. The query operation returns details about the overall instantiated NSs, although it is possible to filter the results for any specific NS instance by providing its unique ID in the request URI. Hence, the NFV-SO invokes the request towards the AAM, which in turn forwards the request to the NFV-LO, making proper mappings on both request URI and inline parameters. The response provided by the NFV-LO contains two parameters only: the name of the app and its status. To this aim, the AAM matches the returned app name(s) to those stored in the local DB in order to provide all details according to the ETSI NFV-SOL005 response format. Note that NOT_INSTANTIATED NS instances (i.e., those created but not running) are not returned by the NFV-LO, that manages running apps only. This means that the number of stored NS instances in the local DB can be equal or higher than those provided by the NFV-LO, but only the NS instances provided by the NFV-LO can match with the INSTANTIATED state in the local DB.

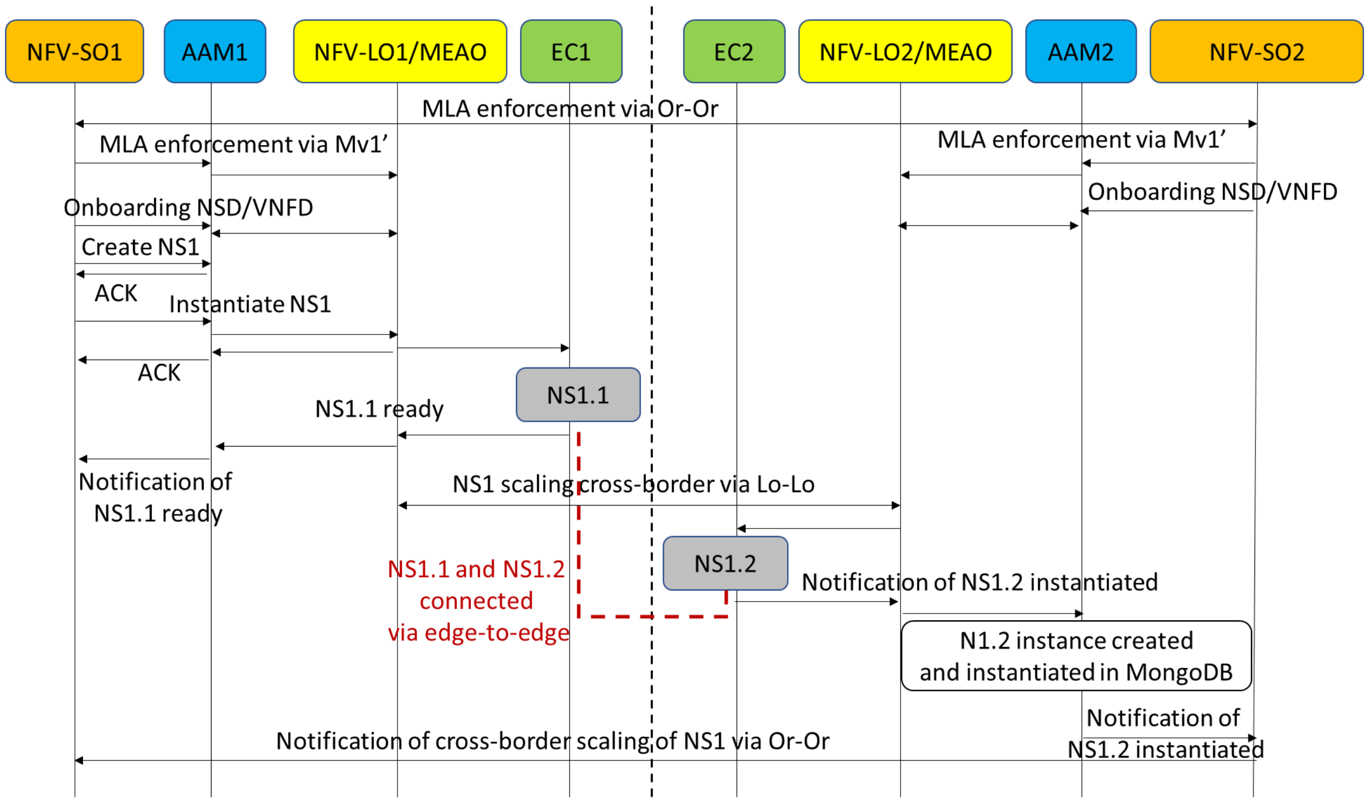

NFV-SO1
AAM1
NFV-LO1/MEAO
EC1
EC2
NFV-LO2/MEAO
AAM2
NFV-SO2
MLA enforcement via Or-Or
MLA enforcement via Mv1'
MLA enforcement via Mv1'
Onboarding NSD/VNFD
Onboarding NSD/VNFD
Create NS1
ACK
Instantiate NS1
ACK
NS1.1
NS1.1 ready
Notification of NS1.1 ready
NS1 scaling cross-border via Lo-Lo
NS1.1 and NS1.2 connected via edge-to-edge
NS1.2
Notification of NS1.2 instantiated
N1.2 instance created and instantiated in MongoDB
Notification of NS1.2 instantiated
Notification of cross-border scaling of NS1 via Or-Or

Fig. 5 – NS instantiation sequence diagram, including vertical instantiation via Mv1', horizontal low latency scaling operation via Lo-Lo, vertical remote notification via Mv1', and final horizontal final notification via Or-Or. Most details about operations occurring between NFV-LO, MEAO, EC and Kubernetes cluster below, including messages distributed via MB, are omitted to ease the reading.

This constraint comes from the abstraction of the NS creation operation, as described in the operation above (Section 3.3.4).

3.3.8 *NS notification operations*

As previously described in the bootstrap phase, the NFV-SO subscribes to the AAM over the Mv1' reference point at the very beginning. This allows NFV-SO to receive notifications coming from the orchestrated edge platform. The notifications of interest here are those relevant to the app instantiation/deletion on cross-border domains. Fig. 5 shows a sequence diagram of NS instantiation and subsequent scaling of an NS in a cross-border setting, this second event triggered by the MEAO. Notifications messages, as well as other AAM operations, are clearly identified. In addition, the AAM, through the subscription to the MB, is able to monitor the performance as well as the resource usage of computing nodes in the edge platform. Thus, it can use this operation to periodically notify the NFV-SO about the edge resource usage in an abstract and aggregated way.

In general, any app instantiation or deletion on the NFV-LO side produces a notification message that it is published on the MB on a specific topic. The AAM, which has subscribed to the same topic on the MB during the bootstrap phase, receives the relevant notification, which in turn triggers internal controls to detect any matching to the NFV-SO notification filters. For instance, some notifications provide updates on some parameters related to the running instances (e.g., the IP address of a running app), whereas others include the detection of new NS instantiations occurred via Lo-Lo as per MLA permissions, and thus transport the notification of new instances created.

For any positive matching, the AAM prepares and transmits an unsolicited notification message towards the NFV-SO, by using the callback URI defined in the subscription request. Both NFV-SO notification filters and callback URI are stored in the local DB.

3.3.9 *NS termination*

The termination operation is the reverse of the instantiation operation described above. It triggers the termination of an NS that has already been instantiated (i.e., its internal status is INSTANTIATED). The first step consists of the termination request from the NFV-SO to the NFV-LO via the Mv1'. The NS instance ID (*nsInstanceId*) used by the NFV-SO to address the instance to be terminated is included in the POST message URI. This is the same ID provided by the AAM after the creation of this NS instance and it is required to retrieve the NS instance information from the local DB. This enables obtaining the unique app name used by the NFV-LO to identify the running app (*appName*). A HTTP DELETE message towards the NFV-LO is then generated by using the *appName* and the relevant parameters retrieved from the local DB, such as the *namespace* of the relevant tenant. In case of success, the app instance is stopped and then removed from the NFV-LO.

The response provided by the NFV-LO causes an update of the NS instance internal status on the local DB. If no error occurs, the NS state changes returning to NOT_INSTANTIATED. If needed, the same NS instance can be instantiated again by the NFV-SO always using the NS instantiation operation.

3.3.10 NS deletion

This operation is the reverse of Create NS. Following the ETSI NFV-SOL005 API specifications, the Mv1' exposes the abstract operation for the complete NS instance deletion by providing its unique ID (*nsInstanceId*). This operation is executed by the AAM only and consists of the removal of the NS entry from the local DB. Any deletion request on NS instances whose internal state is different from NOT_INSTANTIATED will be rejected.

4. IMPLEMENTATION AND PERFORMANCE ASSESSMENT

4.1 AAM implementation and testbed setup

The AAM used to execute the experiments was implemented by using the Java Spring Framework [50]. It allowed us to take advantage of some of its out of the box solutions, such as the RESTful web services based on Apache Tomcat® [45] and the native multi-threaded handling of incoming requests, thus accepting concurrent requests and avoiding waiting times for the resolution of previous ones. In order to implement the persistence layer of the AAL, we used MongoDB [42]. This choice is motivated by the fact that it is natively compliant to the JSON format [43], making it the ideal solution to handle SOL005 API payloads.

The AAM is composed by three main software modules, sketched in Fig. 6:

1. The MongoDB controller, which is in charge of both marshalling operations (i.e., transforming the in-memory software object into a data format suitable for DB storage and retrieval) and for executing DB queries. The payload of the REST calls is encoded by using the JSON format, which enables direct storage in the local DB.
2. The MB client based on Apache Qpid™ component [46], which is responsible for a subscription to AMQP broker [47] topics and for consuming published messages.
3. The adaptation interface based on the well-known adapter design pattern that allows incompatible interfaces to work together. This is the core architecture implementing the business logic of the abstraction layer implemented in the AAM.

Basically, the adapter design pattern is composed of three main elements, highlighted by dashed red boxes in Fig. 6:

- The target interface defines the operations exposed to the client (i.e., the calling NFV-SO). These operations are defined by the OSM/ETSI NFV-SOL005 API and are identified by their unique endpoints. They consist of a request method (i.e., GET, POST, DELETE) and at least one header field (i.e., Content-Type) defining the media type of the body content for both requests and responses (i.e., application/JSON).
- The adapter class implements the target interface invoking relevant functions on the adaptee in order to expose the expected OSM/ETSI NFV-SOL005 operations. It is in charge also of managing the MongoDB controller and the interactions with the AMQP client.
- The adaptee class is the interface that must be adapted (i.e., the abstract NFV-LO class in Fig. 6). This abstraction level enables the definition of concrete adaptee objects able to manage incoming requests according to their own internal business logic, providing standard OSM responses to the caller. This means that the adaptees implement the adaptation functions that transform OSM/ETSI NFV-SOL005 request/response bodies to custom ones, according to the interface to be adapted. Moreover, each concrete adaptee maps OSM endpoints to custom ones when an equivalent function exists on the contacted

NFV-LO (as defined by the REST API implemented in the underlying NFV-LO), otherwise the required function is emulated by the adaptee in order to provide the expected OSM response. This means that the equivalent OSM/ETSI NFV-SOL005 feature can be obtained by forwarding multiple requests to the underlying NFV-LO or by processing data stored in the internal database (or by a proper combination of both techniques). In the case that the underlying NFV-LO implements the same functionality as defined by the OSM/ETSI NFV-SOL005 APIs, the adaptee object acts as a relay node avoiding any further manipulation on both endpoint and message body. Note that this is the standard implementation of the OSM_adaptee concrete class shown in Fig. 6.

A brief example of adapted call is shown in Fig. 6, for the “query information about multiple NS instances” operation, defined in the ETSI compliant target interface. Any request on the endpoint defined therein (i.e., “/nslcm/v1/ns_instances”) is handled by the method “get_NS_interfaces()”, highlighted by the blue color. The adapter class implements this method and invokes the equivalent operation (defined here using the same method name) on the abstract adaptee class. According to the running concrete adaptee object, selected during the AAM bootstrap phase, the algorithm defined in that method may differ significantly, as shown by the pseudocode associated with OSM_adaptee and lightMANO_adaptee classes, respectively, shown in the gray boxes.

The AAM runs in a single pod, including both the MongoDB database, storing requests metadata and descriptors, and the module exposing the Mv1’ endpoint and managing the communications with NFV-LO and the MB. As for the underlying modules running in the edge node, that is NFV-LO, MB, and MEAO, they are executed in different pods running in the same orchestration VM where the AAM is executed (see Fig. 3). As for the EC, it runs in another VM in the same edge cluster. This means that the latency associated with their communications does not suffer from long propagation delays through the MNO network.

We run tests on two MEC platforms, by using edge nodes deployed in the network of two MNOs participating in the project. The relevant configuration is reported in Table 1. These tests focus on the AAM module, thus we did not use the relevant NFV-SO to trigger the orchestration requests, but emulated it by using a Postman client running in our the lab at University of Perugia. Consequently, we used the public Internet to establish the connectivity between the (emulated) NFV-SO and relevant AAM. The latency associated with these communications may be considered representative of that obtainable between a remote NFV-SO and the AAM on an edge node.

For some tests requiring a simple response from NFV-SO, we used an instance of NFV-SO running in AWS able to accept requests coming from the AAM in edge platform (e.g., during the bootstrap) and subsequently to answer them in a predefined way.

Table 1 – Testbed configuration

MNO	VM	Configuration
MNO1	Orchestration VM (AAM, NFV-LO, MEAO)	4 vCPU (shared), 8 GB RAM, 70 disk
	Edge controller + apps	32 vCPU (shared), 16 GB RAM, 250 disk
MNO2	Orchestration VM (AAM, NFV-LO, MEAO)	8 vCPU (shared), 20 GB RAM, 70 disk
	Edge controller + apps	16 vCPU (shared), 16 GB RAM, 250 disk, fast I/O data plane

4.2 Test cases

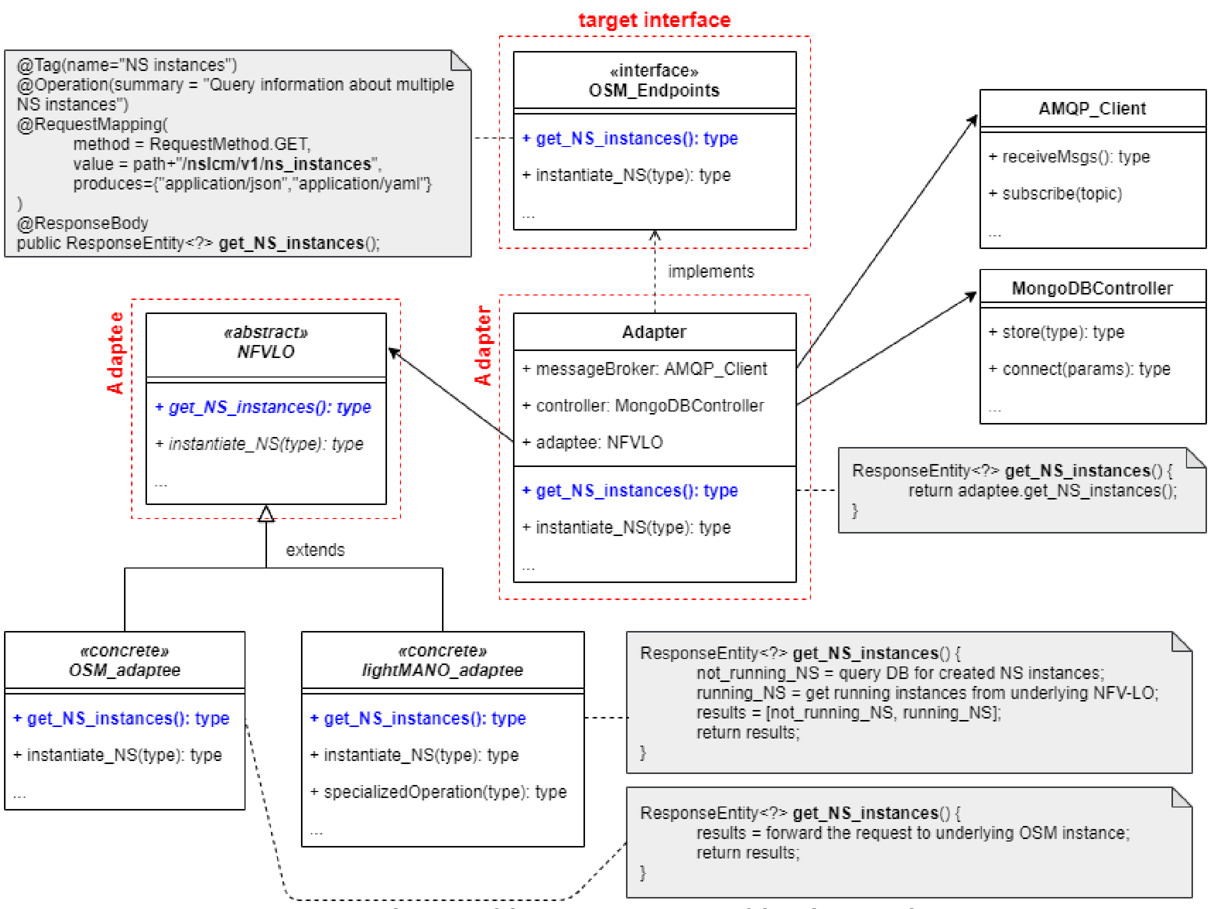
Experiments were performed for operations characterized by different complexity of the orchestration requests. Each request is generated, received, and processed as a REST API request.

The following types of request were performed:

1. Overall edge-NFV-SO registration procedure;
2. Simple GET requests that involve certain transactions and checkups in MongoDB co-located with the AAM itself;
3. POST requests able to trigger changes in the service deployments, such as instantiation and termination requests.

Table 2 provides, for each operation, a brief description and the methods tested with such an operation. The tests have been executed in the following conditions:

- System completely clean, with the MongoDB empty (*default test condition*, used for MNO1 and MNO2 edge nodes);
- System loaded, with the MongoDB containing 10,000 NSD, 10,000 VNFD, 10,000 MLA descriptors, and 10,000 NS instances, used for MNO1 edge node only.


target interface
«interface»
OSM_Endpoints
+ get_NS_instances(): type
+ instantiate_NS(type): type
...
implements
AMQP_Client
+ receiveMsgs(): type
+ subscribe(topic)
...
MongoDBController
+ store(type): type
+ connect(params): type
...
Adaptee
«abstract»
NFVLO
+ get_NS_instances(): type
+ instantiate_NS(type): type
...
Adapter
Adapter
+ messageBroker: AMQP_Client
+ controller: MongoDBController
+ adaptee: NFVLO
+ get_NS_instances(): type
+ instantiate_NS(type): type
...
extends
«concrete»
OSM_adaptee
+ get_NS_instances(): type
+ instantiate_NS(type): type
...
«concrete»
lightMANO_adaptee
+ get_NS_instances(): type
+ instantiate_NS(type): type
+ specializedOperation(type): type
...

```
@Tag(name="NS instances")
@Operation(summary = "Query information about multiple
NS instances")
@RequestMapping(
        method = RequestMethod.GET,
        value = path+"/nslcm/v1/ns_instances",
        produces={"application/json","application/yaml"}
)
@ResponseBody
public ResponseEntity<?> get_NS_instances();
```

```
ResponseEntity<?> get_NS_instances() {
        return adaptee.get_NS_instances();
}
```

```
ResponseEntity<?> get_NS_instances() {
        not_running_NS = query DB for created NS instances;
        running_NS = get running instances from underlying NFV-LO;
        results = [not_running_NS, running_NS];
        return results;
}
```

```
ResponseEntity<?> get_NS_instances() {
        results = forward the request to underlying OSM instance;
        return results;
}
```

Fig. 6 – UML diagram of the main components of the adaptation layer

In this way, we evaluated the impact of the database operation, which is the component in charge of ensuring the consistency of data and operations across the AAM, with respect to the other components, on the overall delay budget. The number of items stored in the database is so high for the considered service scenario, that the obtained results can be considered a sort of upper bound to the average performance. In fact, for each edge node, we expect some tens of services contemporarily deployed in it, and for sure no more than a few hundred.

In all executed tests, the main measured Key Performance Indicator (KPI) is the average response time per orchestration request. This KPI is highly relevant since it reflects the capability of an orchestrator to perform orchestration operations efficiently. In the case under consideration, this is even more important, since the orchestration operations are split both vertically (NFV-SO and underlying NFV-LO) and horizontally (peer NFV-SOs and peer NFV-LOs). For all operations, the latency KPI is further evalauted in two ways: (i) processing time in the AAM, and (ii) the overall communication latency seen by the NFV-SO (emulated by the Postman client).

In order to collect measurements of latency, we defined a further endpoint on the AAM, which allows retrieving the latency associated with each atomic operation carried out by it. This data is available in JSON format via REST call. Each element of the JSON file reports the *timestamp* in which the request was received or issued, the *duration* of the operation, and the *uri* of the request itself, including also a further string specifying also the method, and optionally an index and some extra parameters, to identify different transactions occurring to complete the same operation.

In addition to parsing this JSON file with latency measurements, we also parsed the AAM logs, in order to identify errors or anomalous situations causing abnormal latencies. When the measurements were taken from the Postman client to emulate a remote NFV-SO, we used the delay measurement tool provided by that software.

Table 2 – Description of requests done to the AAM

Operation	Description	Tested methods
Bootstrap	Overall process of registration of NFV-LO to NFV-SO	Multiple POST/PUT requests and DB transactions
Onboarding VNFD	Descriptors onboarding to AAM	Single POST request + DB transaction
Onboarding NSD	Descriptors onboarding to AAM	Single POST request + DB transaction
Onboard MLA	Descriptors onboarding to AAM	Single POST request + DB transaction
Create NS	Creation NS request to AAM, for compliance with ETSI NFV-SOL 005	Single POST request + DB transaction
Instantiate NS	Instantiation of an NS, it involves also NFV-LO and MB	POST request triggering a further POST request towards the NFV-LO and message from Broker + DB transactions
Terminate NS	Terminate an NS, it involves also NFV-LO	POST request triggering a further POST request towards the NFV-LO + DB transactions
get NS list	Request of NS list	Single GET request + DB transactions
get NSD	Request of a specific NS descriptor	Single GET request + DB transactions
get VNFD	Request of a specific VNF descriptor	Single GET request + DB transactions
get MLA	Request of a specific MLA descriptor	Single GET request + DB transactions
delete MLA	Delete MLA descriptor	DELETE request triggering a further DELETE request towards the NFV-LO + DB transactions
delete NS	Delete NS descriptor	Single DELETE request + DB transactions
delete VNFD	Delete VNF descriptor	Single DELETE request + DB transactions
delete NSD	Delete NS descriptor	Single DELETE request + DB transactions
Notification NS	Processes an NFV-LO notification via MB and prepares the one for NFV-SO	AMQP message + DB transaction + POST message towards NFV-SO

Finally, a slightly different procedure was adopted to measure the notification operation. In this case, we evaluated the latency by considering the following components separately. The first one is the contribution associated with the delay introduced by the MB. The other one is that associated with the processing required by the AAM to create/update the NS in the local MongoDB and to prepare the notification to be sent to the NFV-SO.

The delay of the notification, delivered by using a broker, was evaluated by the difference of the timestamp values collected in the AAM and NFV-LO. This is feasible since they see the same system time, as they run in different pods in the same VM.

4.3 Performance evaluation

Fig. 7 shows the measurement results per each operation, measured from the Postman client for the MNO1 and MNO2 edge nodes, so emulating operations triggered by a remote NFV-SO. As for the edge node of MNO1, we tested the “default” configuration, with the MongoDB internal to the AAM empty, and the “loaded” configuration, as explained in Section 4.2.

For each operation, we include the 95% confidence interval obtained by multiple experiments. This figure shows not only the measurement associated with the Registration phase (from reception of message 2 to reception of message 4 in Fig. 4), but also those associated with simple GET operations as well as those associated with more complex orchestration operations, such as instantiation and termination of NSs. We recall that NS creation and deletion are very simple operations, implying only a local database transaction on the MongoDB, and implemented just for compliance with ETSI NFV-SOL005 [40]. It is evident that most of operations, when observed from an external entity, requires nearly the same time (approximately 100 ms) but NS instantiation and NS termination, which involve more complex interactions with the platform below, and namely with the EC to launch/stop real Apps running in the Kubernetes cluster. For these operations, the required time increases, and can be even larger than 1s. When the loaded case is considered, we can see that any increase of the number of instances and descriptors loaded in the AAM, which has to track their status by means of database persistence, produces larger mean values of the latency profile. However, variability is still low, as can be seen by the value of the confidence intervals. In any case, most operations, when observed from the NFV-SO, and thus including also the delay contribution due to the interaction with the underlying entities beyond the AAM, are always lower than 600 ms. In particular, we can see that latency associated with operations carried out on the edge node of MNO2 is generally larger than both that of MNO1 in standard configuration, and often of that of the loaded configuration. Fig. 8 presents the same set of measurements, but related to the AAM processing only.

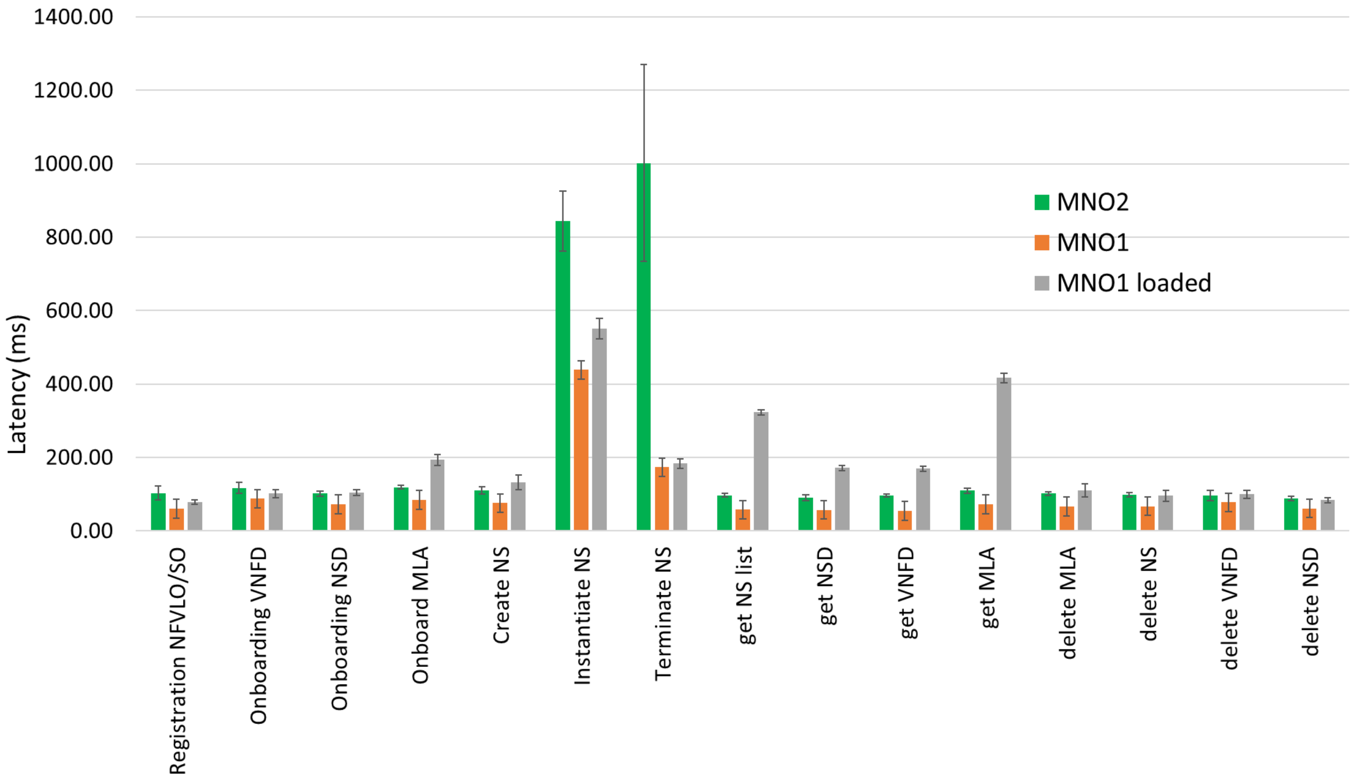


Fig. 7 – Latency measurement taken from the remote Postman client, emulating an NFV-SO, for edge nodes of MNO1 and MNO2 in default conditions, and for edge node of MNO1 in loaded condition.

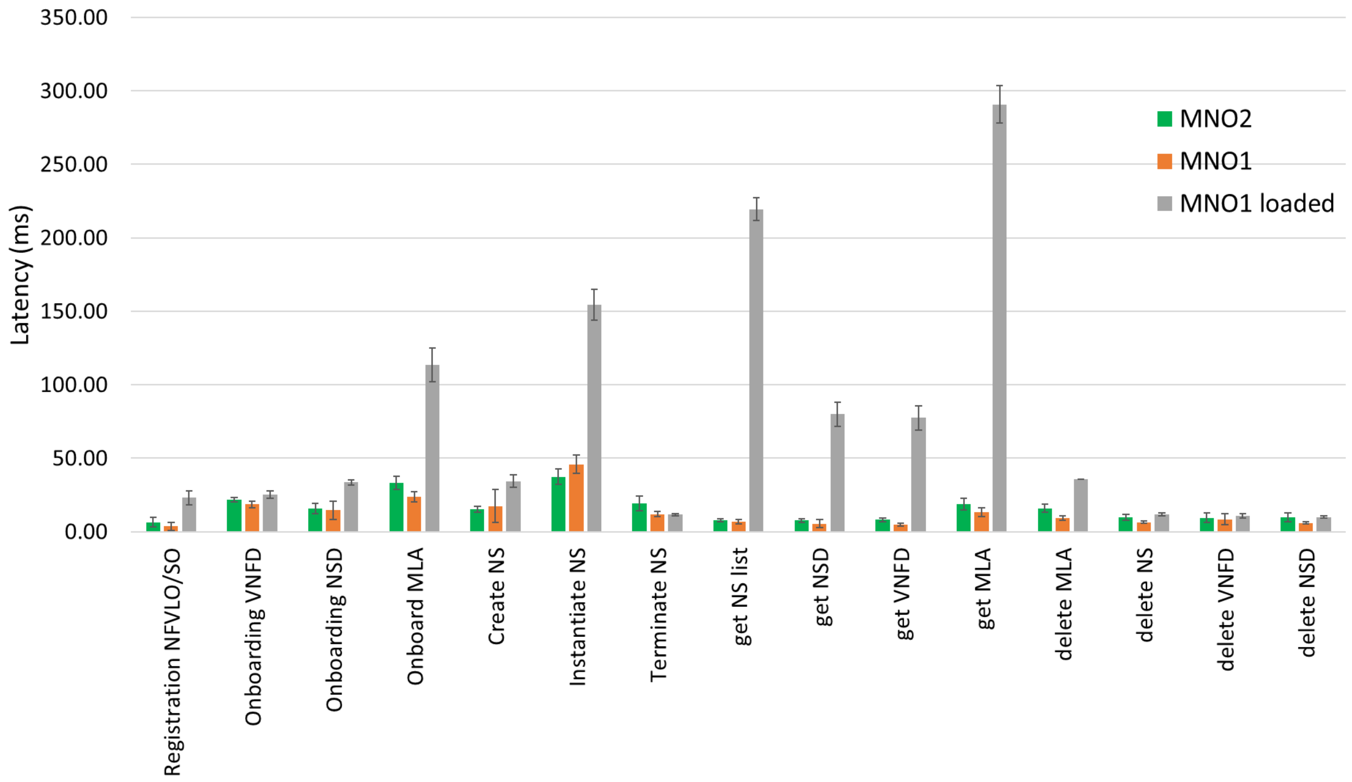


Fig. 8 – Latency measurement taken from the AAM for edge nodes of MNO1 and MNO2 in default conditions, and for edge node of MNO1 in loaded condition.

In this case, it is evident a variability of the processing times. Although most of them are below 20 ms, MLA onboarding and NS instantiation require higher processing times. In fact, for those operations the processing time is higher since it implies a significant interaction with entities below, and namely with the NFV-LO. In this case, not only a local transaction for state persistence is necessary, but also the preparation of a new message to feed into the NFV-LO and the validation of some parameters with the onboarded descriptors (NSD, VNFD and MLA). This is the case of the MLA onboarding and of NS instantiation. The NS termination, instead, requires a much simpler interaction with the NFV-LO, without requiring the preparation of a complex message. In any case, the maximum contribution to the delay budget is less than 50 ms on average, and of 60 ms in 95% of cases. Again, the average performance of edge node of MNO1 is slightly better than those of MNO2 for the standard configuration. When considering the AAM contribution only and the loaded case, we can see a significant difference with respect to the situation shown to Fig. 7.

Beyond NS instantiation, the operations that are typically more affected by the database content are those implying getting the list of NSs or descriptors (MLA, NSD, VNFD), especially when compared with the values with empty system. The GET operations are those with a larger increase, well beyond 10 times. However, they are operations with a very small latency/processing time in the empty system condition, thus the effect of system load is, in the end, not so dramatic. The further consideration is that, for most of the other operations, the AAM processing time increases by less than 5 times, especially for onboarding operations, although the number of content items increased by a 1000x factor. In any case, considering the perspective of NFV-SO, which is at the end the most significant, most operations have only a modest increase. It is a positive result, considering the very high number of items included in the database.

Finally, we show a separate figure for evaluating the time needed to receive and process a notification coming from the NFV-LO via the MB. Fig. 9 presents the results of measurement done on both MNO1 and MNO2 in default test conditions. Since results were quite different, we used the logarithmic scale for the ordinate axis. The notification delay via the internal MB results is significantly lower than the processing delay of the AAM to prepare the notification, as expected. This is due to the fact that the AAM has to parse the notification received via AMQP, extract meaningful information, map that on the content of the MongoDB database in order to identify the NS for which the notification is relevant, update that entry with the content of the received notification, and finally prepare the message for the NFV-SO. For the cross-border case, the AAM does not find any correspondence in the MongoDB database beyond the MLA. Thus, it has to create a new NS, push it in the database, and prepare the instantiation notification message for the NFV-SO, specifying the instantiation has occurred via Lo-Lo. It is quite evident that the edge node MNO1 is significantly more performant than the MNO2 one.

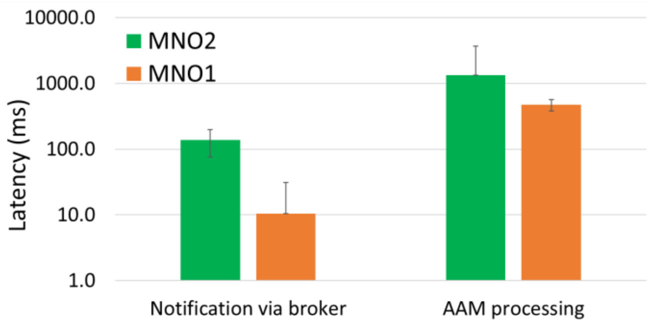


Fig. 9 – Latency associated to notifications from NFV-LO received by AAM via AMQP MB.

Finally, we collected the measurement of average CPU time and memory footprint of the pod during instantiation operations, which is the more demanding one. The result is that the CPU time is, on average, only 5m (milliunits of CPU time, see [51]), whereas the memory footprint is 1081 Mi, that is about 1 GB.

From the results presented in the previous figures, we can derive the following considerations:

- The overall contribution to the delay budget from the AAM in complex operations (NS instantiation and termination) is quite limited, when considering the end-to-end delay seen by the NFV-SO. There are some operations that terminate directly on the AAM itself, either for compliance with ETSI-NFV-SOL 005 or for overall system architecture (e.g., the deletion of some descriptors does not propagate in the entities below), but these are characterized by very low values of the processing time. This is a quite important result due to the function splitting adopted for the orchestration architecture.

- From the analysis of latency values, measured by using Postman, and the processing time captured directly by the AAM when considering only simple operations that involve just the AAM and do not propagate to the entities below, it is possible to estimate the round-trip delay between the client and the edge node. This value results to be 87.5 ms with a 95% confidence interval equal to 3 ms for the edge node of MNO2, and equal to 58 ms with a 95% confidence interval equal to 4 ms for the edge node of MNO1. These values are realistic, especially considering also the geographical distance from the measurement point.

- The impact of the system load on the overall operation latency is really modest. This is due to the fact that we selected MongoDB to implement data persistence, which is a document-based NoSQL database particularly suitable to manage JSON files.
- We tested extensively the two edge nodes. It is quite evident that the edge node of MNO2, although it has computing characteristics similar to those of MNO1 (as shown in Table 2), is sometimes significantly less performant, with a higher processing delay, not only in the AAM layer (Fig. 8 and Fig. 9), but also when the other underlying entities are considered (Fig. 7). This analysis is limited to the default test condition. The main difference, which cannot be captured by the analysis of experimental results, is that the MNO2 host runs several other applications contemporarily. They share the CPU cores, while the system of MNO1 is quite unloaded.
- From the analysis of the resource footprint, it is quite evident that the AAM component requires a significant amount of memory and a very low share of CPU time. This is reasonable, since all operations imply a DB transaction in the MongoDB database. However, the presented values are fully manageable by using the resources typically available in modern servers, without the risk of becoming the system bottleneck.

5. CONCLUSION

Orchestration of CCAM services is known to be a challenging task in a multi-operator envirinment. The main contribution of this paper consists of a flexible and effective solution for orchestrating the deployment and the operation of CCAM services in the MEC nodes of future cellular systems. The core of the proposed solution is based on the the federation and hierarchical organization of the orchestration function, adapted to a multi-operator scenario. The implementation of the proposed architecture, which is compliant with the ETSI recommendations, required the design and implementation of a suitable abstraction and adaptation layer for edge clouds. The resulting system implements a truly cooperative and coordinated orchestration between different edge systems. The performance obtained through an extensive experimentation campaign shows significant benefits in terms of latency, which demonstrate the effectiveness of the proposed solution.

ACKNOWLEDGEMENT

This work was supported by EC H2020 5GPPP project 5G CARMEN under grant agreement No. 825012. The views expressed are those of the authors and do not necessarily represent the project. The Commission is not responsible for any use that may be made of the information it contains. The authors sincerely thank the colleagues participating in Work Package 4 of 5G CARMEN for extensive and fruitful discussions.

REFERENCES

[1] F. Giust et al., "Multi-access edge computing: The driver behind the wheel of 5G-connected cars", IEEE Commununications Standards Magazine, vol. 2, no. 3, pp. 66-73, Sep. 2018.

[2] ETSI, GR MEC 022 v2.1.1, "Multi-Access Edge Computing (MEC); Study on MEC Support for V2X Use Cases", (2018-09).

[3] A. Dalgkitsis et al, "Data Driven Service Orchestration for Vehicular Networks," IEEE Transactions on Intelligent Transportation Systems, vol. 22, no. 7, pp. 4100-4109, July 2021.

[4] F. Bari et al, "Orchestrating virtualized network functions," IEEE Transactions on Network and Service Management, vol. 13, no. 4, pp. 725-739, Dec. 2016.

[5] A. Machen, S. Wang, K. K. Leung, B. J. Ko and T. Salonidis, "Live service migration in mobile edge clouds", IEEE Wireless Commun., vol. 25, no. 1, pp. 140-147, Feb. 2018.

[6] I. Farris, T. Taleb, H. Flinck and A. Iera, "Providing ultra-short latency to user-centric 5G applications at the mobile network edge", Transactions on Emerging Telecommunications Technologies, vol. 29, no. 4, Apr. 2018.

[7] I. Farris, T. Taleb, M. Bagaa and H. Flick, "Optimizing service replication for mobile delay-sensitive applications in 5G edge network", Proceedings of the IEEE International Conference on Communications (ICC), May 2017.

[8] A. Nadembega, T. Taleb and A. Hafid, "A destination prediction model based on historical data contextual knowledge and spatial conceptual maps", Proceedings of the IEEE International Conference on Communications (ICC), Jun. 2012.

[9] G. Xu, S. Gao, M. Daneshmand, C. Wang and Y. Liu, "A survey for mobility big data analytics for geolocation prediction", IEEE Wireless Communications, vol. 24, no. 1, pp. 111-119, Feb. 2017.

[10] R. Cziva, C. Anagnostopoulos and D. P. Pezaros, "Dynamic latency-optimal vNF placement at the network edge", Proceedings of the IEEE Conference on Computer Communications (INFOCOM), pp. 693-701, Apr. 2018.

[11] S. Choi, H. Yeo and J. Kim, "Network-wide vehicle trajectory prediction in urban traffic networks using deep learning," Transportation Research Record: Journal of the Transportation Research Board, vol. 2672, no. 45, pp. 173-184, Sep. 2018.

[12] J. Lv, Q. Li, Q. Sun and X. Wang, "T-CONV: A convolutional neural network for multi-scale taxi trajectory prediction," Proceedings of the IEEE International Conference on Big Data and Smart Computing (BigComp), pp. 82-89, Jan. 2018.

[13] T. Ouyang, Z. Zhou and X. Chen, "Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing," IEEE Journal on Selected Areas in Communications, vol. 36, no. 10, pp. 2333-2345, Oct. 2018.

[14] Slamnik-Kriještorac, N., Silva, E., Municio, E., Resende, H., Hadiwardoyo, S. A., & Marquez-Barja, J. M. "Network Service and Resource Orchestration: A Feature and Performance Analysis within the MEC-Enhanced Vehicular Network Context". Sensors, 20(14), 3852.

[15] Shah S.A.A., Ahmed E., Imran M., Zeadally S. "5G for Vehicular Communications," IEEE Communications Magazine Volume: 56, No: 1, Jan. 2018.

[16] Ning Z., Wang X. Mobile Edge Computing-Enabled "5G Vehicular Networks: Toward the Integration of Communication and Computing," IEEE Vehicular Technology Magazine, Volume: 14, No: 1, March 2019.

[17] Soua R., Turcanu I., Adamsky F., Führer D., Engel T. "Multi-Access Edge Computing for Vehicular Networks: A Position Paper," Proceedings of the 2018 IEEE Globecom Workshops (GC Wkshps); Abu Dhabi, December 9-13, 2018.

[18] Taleb T., Samdanis K., Mada B., Flinck H., Dutta S., Sabella D., "On Multi-Access Edge Computing: A Survey of the Emerging 5G Network Edge Cloud Architecture and Orchestration," IEEE Communications Surveys & Tutorials, Volume: 19, No: 3, 2017.

[19] Saraiva de Sousa et al. "Network Service Orchestration: A Survey". Computer Communications, Volumes 142-143, June 2019, Pages 69-94.

[20] Zhao J., Li Q., Gong Y., Zhang K., "Computation Offloading and Resource Allocation For Cloud Assisted Mobile Edge Computing in Vehicular Networks," IEEE Transactions on Vehicular Technology. Volume: 68, No: 8, Aug. 2019.

[21] Du J., Yu F.R., Chu X., Feng J., Lu G., "Computation Offloading and Resource Allocation in Vehicular Networks Based on Dual-Side Cost Minimization," IEEE Transactions on Vehicular Technology, Volume: 68, No: 2, Feb. 2019.

[22] Hoang V.H., Ho T.M., Le L.B., "Mobility-aware Computation Offloading in MEC based Vehicular Wireless Networks," IEEE Communications Letters, Volume: 24, No: 2, Feb. 2020.

[23] Wang J., Feng D., Zhang S., Tang J., Quek T.Q.S., "Computation Offloading for Mobile Edge Computing Enabled Vehicular Networks," IEEE Access, Volume:7, 2019, pp. 62624-62632.

[24] Soenen T. te al, “Empowering Network Service Developers: Enhanced NFV DevOps and Programmable MANO, ” IEEE Communications Magazine, Volume: 57, No: 5, May 2019.

[25] Celdrán A.H., Clemente G., Pérez G.M., “Automatic Monitoring Management for 5G Mobile Networks,” Proceedings of the 12th International Conference on Future Networks and Communications, Leuven, July 24-26, 2017.

[26] R. Perez et al., "Monitoring Platform Evolution towards Serverless Computing for 5G and Beyond Systems," IEEE Transactions on Network and Service Management, 2022, doi: 10.1109/TNSM.2022.3150586.

[27] Zhdanenko O., Liu J., Torre R., Mudriievskiy S., Salah H., Nguyen G.T., Fitzek F.H.P., “Demonstration of Mobile Edge Cloud for 5G Connected Cars,”, Proceedings of the 2019 16th IEEE Annual Consumer Communications & Networking Conference (CCNC); Las Vegas, January 11-14, 2019.

[28] M.Femminella, M. Pergolesi, G. Reali, “5G experiment design through Blueprint”, Computer Networks, Volume 190, 2021.

[29] W. Nakimuli et al., "Automatic deployment, execution and analysis of 5G experiments using the 5G EVE platform," 2020 IEEE 3rd 5G World Forum (5GWF), 2020, pp. 372-377.

[30] F. Z. Yousaf, V. Sciancalepore, M. Liebsch and X. Costa-Perez, "MANOaaS: A Multi-Tenant NFV MANO for 5G Network Slices," IEEE Communications Magazine, vol. 57, no. 5, pp. 103-109, May 2019.

[31] N. Slamnik-Krijestorac, G. M. Yilma, M. Liebsch, F. Z. Yousaf and J. Marquez-Barja, "Collaborative orchestration of multi-domain edges from a Connected, Cooperative and Automated Mobility (CCAM) perspective," IEEE Transactions on Mobile Computing, doi: 10.1109/TMC.2021.3118058.

[32] T. Taleb, I. Afolabi, K. Samdanis and F. Z. Yousaf, "On Multi-Domain Network Slicing Orchestration Architecture and Federated Resource Control," IEEE Network, vol. 33, no. 5, pp. 242-252, Sept.-Oct. 2019.

[33] O. Bekkouche, F. Z. Yousaf, X. Li and T. Taleb, "Management and Orchestration of Mobile Network Services over Federated Mobile Infrastructures," IEEE Network, vol. 35, no. 6, pp. 178-185, November/December 2021.

[34] Trans-European Transport Network (TEN-T), https://transport.ec.europa.eu/transport-themes/infrastructure-and-investment/trans-european-transport-network-ten-t_en. [Accessed 30 04 2022].

[35] ETSI NFV ISG, “Network Functions Virtualisation (NFV) Release 3; Management and Orchestration; Multiple Administrative Domain Aspect Interfaces Specification”, ETSI GS NFV-IFA 030 V3.3.1 (2019-09).

[36] ETSI NFV ISG, “Network Functions Virtualisation (NFV) Release 3; Management and Orchestration; Report on architecture options to support multiple administrative domains”, ETSI GR NFV-IFA 028 V3.1.1 (2018-01).

[37] ETSI, "Multi-Access Edge Computing (MEC); Framework and Reference Architecture," ETSI ISG MEC, ETSI GS MEC 003 V2.1.1, 2019.

[38] ETSI, "Network Functions Virtualisation (NFV); Management and Orchestration,," ETSI ISG NFV, ETSI GS NFV-MAN 001, V1.1.1, 2014. Online

[39] ETSI, "Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV," ETSI GR NFV 003, 2020.

[40] ETSI, "Network Functions Virtualisation (NFV) Release 3; Protocols and Data Models; RESTful protocols specification for the Os-Ma-nfvo Reference Point," ETSI GS NFV-SOL005 v3.3.1, 2020.

[41] OSM, "OSM North Bound Interface API," [Online]. Available: https://osm.etsi.org/wikipub/index.php/NBI_API_Description. [Accessed 27/04/2022].

[42] MongoDB, "a general purpose document-based database," [Online]. Available: https://www.mongodb.com/. [Accessed 27/04/2022].

[43] JSON, "JavaScript Object Notation data-interchange format," [Online]. Available: https://www.json.org/json-en.html. [Accessed 27/04/2022].

[44] ETSI, "Multi-access Edge Computing (MEC); MEC 5G Integration," ETSI GR MEC 031 V2.1.1, 2020.

[45] Apache, "Tomcat project," [Online]. Available: https://tomcat.apache.org/. [Accessed 27/04/2022].

[46] Apache, "Apache Qpid Project," [Online]. Available: https://qpid.apache.org/. [Accessed 27/04/2022].

[47] AMQP, [Online]. Available: https://www.amqp.org/. [Accessed 27/04/2022].

[48] E. Coronado et al., "ONIX: Open Radio Network Information eXchange," IEEE Communications Magazine, vol. 59, no. 10, pp. 14-20, October 2021.

[49] R. Riggio, S. N. Khan, T. Subramanya, I. G. B. Yahia, and D. Lopez, "Lightmano: Converging nfv and sdn at the edges of the network," IEEE/IFIP NOMS 2018, April 2018.

[50] Spring, Framework, [Online]. Available: https://spring.io/projects/spring-framework. [Accessed 30 04 2022].

[51] Kubectl Top Pod/Node: Collecting Kubernetes Metrics, https://www.containiq.com/post/kubectl-top-pod-node-for-metrics. [Accessed 30 04 2022].

[52] Kubernetes web site: https://kubernetes.io/. [Accessed 30 04 2022].

[53] Benedetti, P.; Femminella, M.; Reali, G.; Steenhaut, K. "Experimental Analysis of the Application of Serverless Computing to IoT Platforms", Sensors, 2021, 21, 928.

AUTHORS

Mauro Femminella received both a Master's degree and a Ph.D. in electronic engineering from University of Perugia in 1999 and 2003, respectively. Since November 2006, he is an assistant professor at the Department of Engineering, University of Perugia. He is co-author of about 100 papers in international journals and refereed international conferences. His current research interests focus on molecular communications, big data systems, and architectures and protocols for 5G networks.

Gianluca Reali has been an associate professor at the University of Perugia, Department of Engineering, Italy, since January 2005. He received a Ph.D. degree in telecommunications from the University of Perugia in 1997. From 1997 to 2004 he was a researcher at the Department of Electronic and Information Engineering of University of Perugia. In 1999 he visited the Computer Science Department at UCLA. His research activities include resource allocation over packet networks, wireless networking, network management, multimedia services, big data management, and nanoscale communications.