University of Padova
Department of Information Engineering

Master Degree in
ICT for Internet and Multimedia

A framework and tools for mobile network automation in a cloud-based environment featuring NFV and MEC

Supervisor:
Prof. Michele Rossi

Assistant Supervisor:
Dr. Fabio Giust

Candidate:
Francesco Asquini

Academic Year 2019-2020
Dissertation Date 20 July 2020

Abstract

The present work investigates the potential offered by Network Function Virtualization (NFV) in the automation of deployment and configuration operations for mobile networks. This thesis provides the description of an experimental framework realized with open-source tools conceived for integration of a 4G network into a virtual infrastructure, making use of the standardized architecture and interfaces provided by the NFV model. Thereafter, the management mechanisms made available by the framework and the implemented solutions for network automation are outlined.

Subsequently, integration of Multi-access Edge Computing (MEC) services into the aforementioned framework has been considered, carrying out a study on efficient content delivery through an application located at the edge of the network. Procedures have been conceived for automatic activation and configuration of the edge services, responding to a user performance improvement demand, and a proof of concept of the system is provided, demonstrating the NFV and MEC integration feasibility and the advantages of leveraging the edge computing model for data communications.

Acknowledgements

First of all, I am deeply grateful to the all the people who with their ideas and researches forged and nurtured the telecommunications world. Through their contributions this field has sprung to life, and in a great enthusiasm that continues nowadays is changing people's lives. I strongly believe interconnections and information mobility can make a better world where people could connect, know and evolve. I hope I would be able to add a tile myself to this ever-growing mosaic.

Gratitude goes to my supervisor at university of Padova, professor Michele Rossi, for being inspirational and donating a portion of its time to support my thesis work.

Deepest thanks to Athonet, for accepting me as an intern and giving me the possibility to develop my master thesis. I was daily immersed in a stimulating and vivacious environment which taught me a lot, in many different ways. Especially, I desire to thank my company tutor, dr. Fabio Giust, his guidance and precious advices led me to the conclusion of the work without getting lost; it has been a pleasure to work with him and I could learn much, both in terms of technical knowledge and creative expression. Special thanks to Dario, Marcello, and Alberto, for their Long Term Support. And to the coffee machine, invaluable friend.

Non-trivial thanks to my family, that has always believed in me and even endorsed my departure from the Friulian lands... To Lorenzo, who was waiting to end up in the acknowledgements, and to the beloved uncle Paolo, ceaseless echo.

Flowery thanks to Marta, for the warmth of her presence and the affectionate support, not only in the tense months. Many hugs!

Optimized thanks to my engineering friends, Alberto and Emilio, for their vivacity, Alessandro and Beniamino, for their tranquillity.

Vibrating thanks to an angelic teacher, whose steps I follow, and can bring enlightenment.

Contents

List of Figures

Figures 2.3, 2.4, 2.6, 2.7, 3.4, are courtesy of ETSI.

Acronyms

3GPP Third Generation Partnership Project

API Application Programming Interface

DASH Dynamic Adaptive Streaming over HTTP

E-UTRAN Evolved UMTS Terrestrial Radio Access Network

EM Element Manager

EPC Evolved Packet Core

ETSI European Telecommunications Standards Institute

HSS Home Subscriber Server

LBO Local Breakout

LTE Long Term Evolution

MANO Management and Orchestration

MEC Multi-access Edge Computing

MME Mobility Management Entity

NFV Network Function Virtualization

NFVI NFV Infrastructure

NFVO NFV Orchestrator

NS Network Service

OSM Open Source MANO

PGW Packet Data Network Gateway

SGW Serving Gateway

UE User Equipment

URL Uniform Resource Locator

VIM Virtualized Infrastructure Manager

VM Virtual Machine

VNF Virtual Network Function

VNFC VNF Component

VNFM VNF Manager

Chapter 1

Introduction

The worldwide increasing demand for mobile connectivity, along with the massive amount of data exchanged daily over the Internet and the arise of new, challenging, communication scenarios, is putting great pressure on the existing telecommunication technologies. Long Term Evolution (LTE) mobile networks have proven themselves a valuable standard in the last decade, able to meet the demands of an increasingly data-oriented world, where the exchange of information has become an essential driver for progress and one of the main sources of revenue. 4G is today regarded as a mature technology, and even if the transition to 5G networks is in full swing, involving a great engagement of industry and academia, 4G networks are still undergoing an evolution and optimization process. One of the most promising research direction is that of network virtualization, aiming at removing the strict dependency of network services on the underlying hardware infrastructure. In fact, the increasing Capital and Operating Expenditures (Capex + Opex) for deployment, maintenance, and activity of the network assets, are tending to become unsustainable for the operators, the revenues not covering the costs; moreover, the scaling procedures to accomodate for increasing connectivity demand and greater volumes of traffic are made complex by the hardware dependency. Network virtualization, taking advantage of the evolution of IT field and the advent of cloud computing, emerged as a promising paradigm to enhance network deployment flexibility and scaling easiness, while reducing hardware costs and need for human assistance. Another emerging connectivity model is the edge computing paradigm, aiming at providing mobile services at the edge of the network, closer to the users: this allows to reduce the network access time and to provide custom contents based on specific geographical areas; it has a strict relation with network virtualization, being based on cloud computing infrastructures. Both the network function virtualization (NFV) and the multi-access edge computing (MEC) paradigms are considered an essential component of 5G technologies; nonetheless, opening to their sperimentation in 4G networks can provide valuable insights about

their integration in 5G technologies, apart from providing an intrinsic improvement to LTE installations.

The present work has been developed during an internship carried out at Athonet Srl. Its objective is the integration of the NFV and MEC paradigms in a 4G mobile network complying with the related standards and making use of open-source tools, to ensure interoperability between implementations of different operators. The commercial software realizing the 4G network has been provided by Athonet. Especially, the potential of a unified NFV and MEC system for network automation has been explored, using software solutions to deploy, configure, and maintain network components and respective functionalities requiring the least possible human intervention, for a more efficient and cost-effective network management.

The thesis is organized as follows:

- Chapter 2 presents an overview of the LTE mobile networks structure and functionalities, along with a description of NFV and MEC functional architecture and applications. A review on recent works in the field is also proposed;
- Chapter 3 proposes an high level architectural solution for the system implementation, a detailed description of its composing network functions, and the considered use cases;
- Chapter 4 provides an overview of the open-source tools employed for the system realization and the way they have been used;
- Chapter 5 describes in full detail the implementation of the system, focusing on the solutions devised for integration of the components and provision of the expected functionalities;
- Chapter 6 illustrates the experimental simulations carried out to evaluate the system capabilities and provides observations on the obtained results;
- Chapter 7 finalizes the thesis commenting the work and the results, and outlining some possible extensions of interest.

NOTE: Due to the Covid-19 outbreak in the early months of 2020, the internship at the base of the present work was subject to strong limitations and the programmed work plan could not be carried to completion. A part of the evaluation tests is missing; a general proof of concept of the system was however produced.

Chapter 2

Background

2.1 4G Mobile Networks

Fourth generation mobile networks mark a difference respect to the previous cellular systems being the first technology to support only packet-switched services, aiming to provide seamless IP connectivity between a user equipment (UE, a device demanding data services) and a packet data network (PDN, an external network providing the data services). The first 4G standardization traces back to 2008, with the 3GPP Release 8 defining its premises, new radio interfaces and core network. Since then LTE has become a mature and flexible technology used worldwide and continuously evolving, with the current latest release mainly concerning 4G being number 14 [1]. Release 15 outlines instead the first full set of standards for the forthcoming 5G networks, together with some enhancements to LTE.

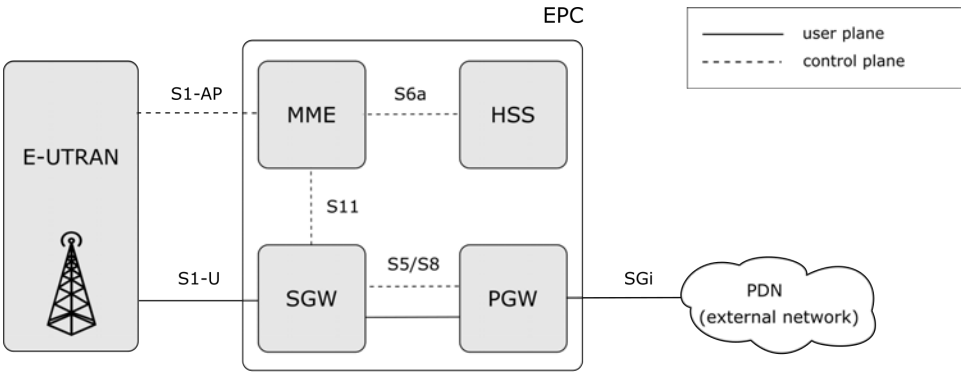


Figure 2.1: 4G network architecture.

2.1.1 Architecture

As depicted in figure 2.1, the network consists of two separate elements: the E-UTRAN (Evolved UMTS Terrestrial Radio Access Network) and the EPC (Evolved Packet Core). Together with the PDNs they form the EPS (Evolved Packet System).

The E-UTRAN consists of a network of base stations (eNB) whose task is to enable and handle communications between the UEs and the EPC. It is responsible for all the radio-related functions, such as the radio resources management (e.g. transmission scheduling and dynamic allocation of resources to UEs in both uplink and downlink), and for handovers, namely the transfer of ongoing data sessions whenever the UE attaches to a diverse eNB, without dropping the connection.

The EPC is the core of the network and manages all data flows to provide connectivity to users. Two main distinct flows can be identified: control plane communications and data plane communications, identified respectively by dashed and solid lines in figure 2.1. The former regards all the signalling performed in order to set up, manage and maintain connections, while the latter carries the user traffic.

The core network consists of four interoperating components:

- **Mobility Management Entity (MME)**: it is the main control node, processes the signalling between the UE and the network. It authenticates users as soon as they request an attach to the network and selects the appropriate data gateway for their traffic; it retains information about the active users and their connections. MME exchanges information with the HSS through the S6a interface, with the SGW through the S11 interface, with the E-UTRAN through the S1-AP interface (also found as S1-MME or S1-C);

- **Home Subscriber Server (HSS)**: a database containing information about the subscribers. It is consulted by the MME to authenticate users trying to connect to the network: only if its identity is present in the database a user is authorized to consume network services. HSS communicates with the MME through the S6a interface;

- **Serving Gateway (SGW)**: its main function is the routing and forwarding of user data packets. It is the termination point of the packet data interface towards E-UTRAN and is responsible for inter-eNB handovers, serving as a mobility anchor as the user moves across eNBs. It also provides interconnection with other 3GPP technologies (such as 2G/3G) and performs lawful interception. SGW exchanges information with the MME through the S11 interface, with the PGW through the S5/S8 interface, with the E-UTRAN through the S1-U interface;

- **Packet Data Network Gateway (PGW)**: it is the connecting node between UEs and external networks, and responsible for IP address allocation for the UEs. In order to access multiple PDNs, UEs can connect to several PGWs at the same time. PGW communicates with the SGW through the S5/S8 interface and with the PDN through the SGi interface.

An actual 4G network can comprise different instances of the previous components (MME/SGW pools), to support mobility of users and to better handle high traffic loads.

2.1.2 Bearers and connectivity setup

A bearer is a virtual tunnel between a PDN gateway and the UE providing an IP packet flow with a defined quality of service (QoS); the E-UTRAN and EPC together set up and release bearers as required by applications. Multiple bearers can be established for a single user, in order to provide different QoS streams or connectivity to different PDNs: as an example, a VoIP call performed while also browsing the web. The MME is the entity that keeps all the information about the established bearers for every user and handles their creation, teardown, or updates.

The UE attachment procedure to the network includes the establishment of a default EPS bearer which will be maintained until the termination of the connection, to guarantee an always-on best-effort service. The setup of any bearer requires a number of similar subsequent steps [2]:

- as the UE requests services to the network, the EPC selects an appropriate PGW to provide connectivity to the user
- the setup procedure is initiated by selected PGW, which assigns an IP address to the UE and sends the SGW a request to create a bearer with the required QoS
- all the information about the request is forwarded by the SGW to the MME
- MME creates a Bearer Setup Request including session management and configuration information along with the bearer identity and sends it to the eNB serving the UE. The Setup Request also provides the connection requirements of the bearer to the eNodeB; this information is used to ensure the necessary QoS by appropriate scheduling of the user's IP packets
- the eNB, which has already a radio bearer active toward the UE, establishes a bearer toward the SGW
- the bearer is now operative and a positive response is sent back to notify the correct establishment, completing the process.

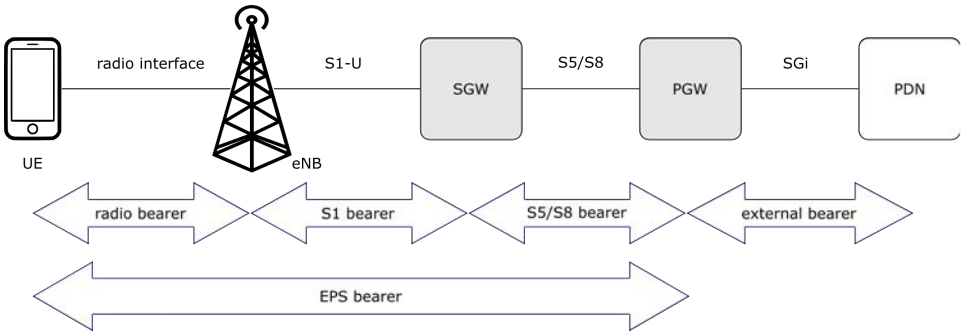


Figure 2.2: Bearers established to enable connectivity and EPS bearer composition.

As summarized in figure 2.2 , an EPS bearer is then composed by three minor bearers, each of which established on a single interface: a radio bearer between the UE and the eNB, a S1 bearer on interface S1-U, and a S5/S8 bearer, on the homonymous interface. The existence, linkage, and mutual identification of all these is necessary for the connection to work. An ulterior external bearer is created between the PGW and the PDN, to enable data flow toward where services are located.

To support UE mobility between eNBs, the bearer changes along with the user position: if the UE attaches to the network via a different eNB the radio and S1 bearers are recreated respect to the new radio link; the S5/S8 bearer remains fixed. If instead an event triggers the selection of another SGW to serve the user, all bearers are updated, since the data flow needs to follow a diverse path from/to the PGW. The full EPS bearer needs to always remain active, even if its composing bearers may be reconfigured to meet the mobility needs.

2.2 Network Function Virtualization (NFV)

Several functions chained together are necessary for a network to work properly and provide the required services to the users. Nowadays, network functions are largely realized through a wide variety of proprietary and highly-specific combinations of software and hardware, known as network elements. Despite being the most adopted solution, it presents some important drawbacks due to the low flexibility a specific hardware introduces into the system: first of all, hardware lifecycles are becoming shorter as technology and services evolution accelerates, forcing the frequent design and deployment of new equipment, limiting therefore the innovation rate of network technologies and lowering the overall revenues. Moreover, every hardware instance can provide only a specific set of functionalities, defined in the design phase of the product, forbidding the opportunity to easily respond with flexibil-

ity to different requirements. Trained personnel would also be necessary to repair or substitute these deployments. As a consequence, in the following years a network architecture excessively relying on specific hardware will constitute the bottleneck of the whole system, both in terms of innovation rate and offered performances.

To overcome this matter, the European Telecommunications Standards Institute (ETSI) has been developing the Network Function Virtualization (NFV) paradigm, since 2012 when the first Industry Specification Group was created in order to lay the groundwork for the new standard. Its premises can be found in [3] along with relevant use cases which make it appealing for the application in modern networks. The idea behind NFV is to break the strict binding between software and hardware present in current proprietary systems, by making network functions software-only applications (named Virtual Network Functions, VNF) running on top of generic hardware by the use of an appropriate virtualization infrastructure. Most of the complexity and functionalities are then transferred to the software side of the system, leaving the hardware as a mere support.

Virtualization is defined as the process of creating a simulated computing environment on top of a physical host computer, in a layer abstracted from the actual hardware; virtual machines (VM) created this way act as an independent physical computer respect to the host machine, able to run their own operative system and software. The software that creates and monitors virtual machines on the host hardware is called a hypervisor: it is in charge of retrieving required hardware resources and making them available for guest system applications.

NFV paradigm aims at including virtualization in the network context, with the opportunity to install network functions on generic commercial off-the-shelf hardware, e.g. multi-purpose servers, whose role is only to provide computational, storage, and (physical) networking resources. The advantage in doing this is the reduction of the number of different hardware devices in a network and the complexity of deploying, configuring, and maintaining them. Among the benefits such approach offers:

- rapid service innovation through software-based deployment, given that a software update alone would be sufficient to perform the update of the function, leaving underlying hardware unaffected;
- standardized and open interfaces between virtual network functions, infrastructure and associated management entities, so that hardware and software separate elements can be provided by different vendors;
- great flexibility in deploying VNFs on the hardware. This both aids scalability and decouples network functionalities from location, by allowing software to be located at the most appropriate places, anywhere an infrastructure is available. Moreover, the infrastructure resources can be reassigned to the VNFs according to the requirements;

- improved capital efficiencies compared to dedicated hardware implementations.

Among the network functions which are suitable for virtualization, the LTE core network components offer great opportunities in this perspective. In addition to the advantages of removing dependency on proprietary hardware, we mention the higher service availability and resiliency provided by dynamic network reconfiguration, along with the ease of network topology changes to optimize performances and support agile introduction of new services, both potentials inherent to virtualization technology with a distributed infrastructure. In this cloud native environment, the user can request services to the network as if they were offered by a traditional deployment: the end-to-end services should be independent of whether the network functions are virtualized or not.

Another crucial application regards the 5G Network Slicing architecture. It consists in running multiple independent logical networks on a common physical infrastructure, each of them supporting different use cases and QoS requirements (e.g. massive IoT connections, or low latency communications). Devices can choose the appropriate network slice conforming to their needs, they connect to the same service provider infrastructure but use a different set of virtual or physical functions, accordingly.

2.2.1 Architectural framework

ETSI standard [4] defines the high-level functional architecture to support VNF operations across different hypervisors and computing resources. To enable interoperability it is important to elaborate a standardized architectural model that specifies how the elements necessary to realize NFV can be implemented and connected, allowing different VNFs to be deployed over the virtualized infrastructure with minimal integration effort and maximum reuse.

Three main working domains (shown in figure 2.3) are identified in NFV context :

- NFV Infrastructure (NFVI), consisting of the underlying physical resources and the hypervisor to virtualize them. NFVI supports the deployment and execution of the VNFs;

- Virtual Network Function(s), as the software implementation of a network function which is capable of running over the NFVI;

- Management and Orchestration (MANO), which covers the orchestration of physical and software resources that support the infrastructure virtualization, and the lifecycle management of VNFs. MANO focuses on all virtualization-specific management tasks necessary in the NFV framework.

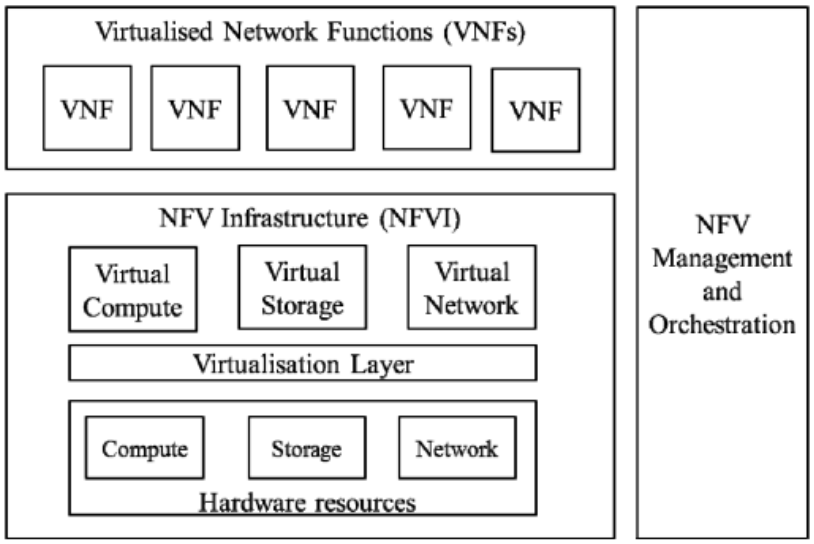


Figure 2.3: High-level NFV framework. From [4].

At a greater level of detail, referring to figure 2.4, the working domains can be decomposed in functional blocks and management interfaces, to be briefly presented in the following.

OSS/BSS

OSS/BSS is the combination of the operator's other operation and business support functions that are not otherwise captured in the considered architectural framework, but require information exchanges with entities belonging to the NFV system. It communicates with the MANO via the Os-Ma interface.

NFVI

NFV infrastructure, it represents the totality of all hardware and software components which build up the environment in which VNFs are deployed, managed and executed. It is composed by the hardware layer, consisting of physical hardware resources that provide the processing, storage and connectivity capabilities, and the virtualization layer, which abstracts the hardware resources and makes them usable by the software implementing the VNFs. It is controlled by the MANO via the Nf-Vi interface, and communicates with deployed VNFs via a set of Vn-Nf interfaces.

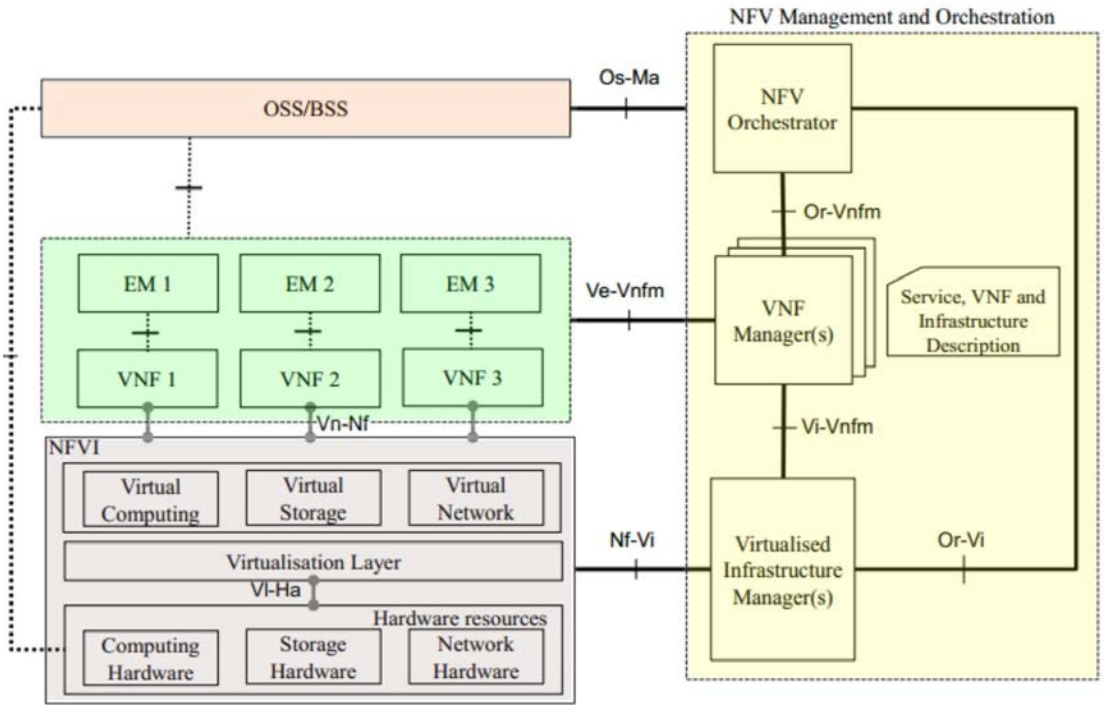


Figure 2.4: NFV reference architectural framework. From [4].

Virtual Network Functions and Network Services

A VNF is a virtualized instance of a network function in a legacy non-virtualised network (the OSS), expected to have the same behaviour of its non-virtualized counterpart. Each VNF can be composed by one or more modules, called VNF components (VNFC), each of which is an independent entity able to communicate with its peers and implementing specific functionalities of the VNF. A VNFC is generally a software component running on a single VM, this allowing an easier scalability to meet performance and distribution requirements: a VNFC can be scaled vertically by increasing the amount of its allocated resources, improving performance, or scaled horizontally by triggering the activation of multiple instances of such VNFC over multiple platforms, improving spatial distribution. Scaling operations do not affect the overall VNF logical behaviour.

To each VNF instance is associated an Element Manager (EM), that can be part of the VNF itself or a different VNF with management capabilities. It is responsible for the configuration (at deployment or run-time), fault detection, security management, and data collection for the involved VNF. MANO contacts the element managers via the Ve-Vnfm interface.

A Network Service (NS) is the composition of a set of VNFs, whose end-to-end functionality is a combination of the behaviour of its constituent

functions. To define the NS behaviour starting from its components, a VNF Forwarding Graph (VNF-FG) can be used: it is a graph whose nodes are the network functions and the edges are the links between them, useful to describe relations and data flows between the constituents; an example is given in figure 2.5. The graph only defines logical links and entities, which is separated from the physical implementation of the system: this emphasizes the fact that the exact physical deployment of a VNF instance on the infrastructure is not visible from the end-to-end service perspective, and neither is an interest of the NS.

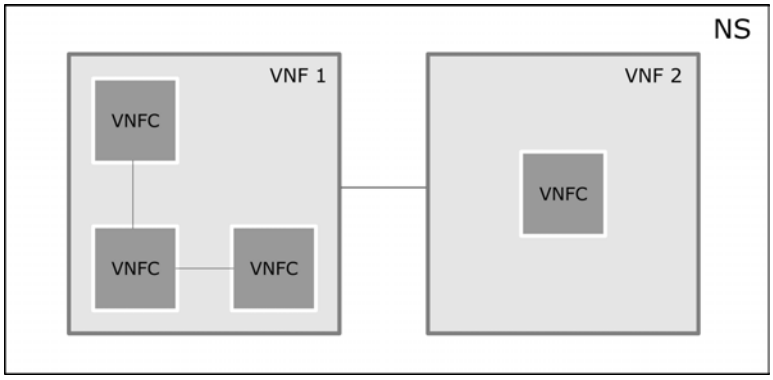


Figure 2.5: Example of the structure of a Network Service. It is composed of two distinct and interconnected VNFs; the first VNF presents three internal components, the second one is a function with a single internal component. The NS will be deployed on the virtualized resources of the NFV infrastructure.

Management and Orchestration (MANO)

The NFV Management and Orchestration entity is the core control node of the NFV environment. Its role is to manage the NFVI, orchestrate the allocation/release of resources required by the VNFs, and deal with all the operations regarding lifecycle management of VNF/NS instances, such as istantiation, configuration, scaling, termination.
MANO is made up of three functional blocks and of internal data repositories containing necessary information to manage instances. These components and their interactions are outlined in detail in a dedicated ETSI standard[5].

- NFV Orchestrator (NFVO)

The NFVO is the highest-level control entity; it is responsible for the orchestration of NFVI resources among multiple infrastructure managers, and for the life-cycle management of Network Services. It coordinates the other MANO components.

- VNF Manager (VNFM)

The VNFM is responsible for the life-cycle management of VNF instances. A single VNFM may manage many VNFs or a VNFM may be deployed for every existing VNF; on the contrary, a single VNF responds to only one VNFM. It controls the network functions under its authority via the Ve-Vnfm interface, and communicates with the other MANO components.

- Virtualised Infrastructure Manager (VIM)

The VIM is directly responsible for controlling and managing the NFVI compute, storage and network resources. It orchestrates the allocation, scaling and release of physical resources and their interaction with the VNFs, actuating the necessary operations on the infrastructure. The VIM can be controlled eiher by the NFVO or the VNFM, and exchanges control information with the NFVI through the Nf-Vi interface.

- Service, VNF and Infrastructure Description

This dataset, stored and accessible inside the MANO, provides all the required information to deploy, configure and monitor network functions and services. In particular, it stores the information models (i.e. a specification of data semantics in a certain context) which define the consented structure and parameters of NSs, VNFs, and virtual links between them.

- VNF and NS Catalogs

Catalogs contain information about all the VNFs and NSs on-boarded in the system, in the form of descriptors. The VNF catalog can be accessed by both the NFVO and the VNFM, while the NS catalog can be accessed by the NFVO only.

- NFV Instances Repository

As a result of an instantiation operation (NFV, NS, or virtual links), records are created to represent the newly created entities: they are based on descriptors representing the instances and additional runtime information.

2.2.2 Deployment Operations

Two distinct and subsequent operations are to be performed by MANO in order to deploy a VNF or NS on the virtualization infrastructure [5]:

- **On-boarding.** It refers to the process of submitting a VNF or NS package to the NFVO to be included in the catalog.
  A VNF package consists of a software image to be deployed, a VNF

descriptor (VNFD), and possibly ulterior specific functionality for lifecycle management. The VNFD is a deployment template specifying resource requirements (CPU, RAM, disk space) and network connection points for the VMs to be instantiated; the connection points can be both internal (connecting components inside the VNF) or external (connecting the VNF with other VNFs or physical networks). VNFD must be validated against the appropriate information model.
A NS package consists of a NS descriptor (NSD), containing information about connections between its composing VNFs and end-to-end functionalities; it may reference to a VNFFG. NSD must be validated against the appropriate information model.
Inside the VNFD and NSD it is necessary to provide a description of every virtual link to be created (being it among VNF components or different VNFs) to enable connectivity. A Virtual Link Descriptor (VLD) specifies the topology and connection points of the network to be created in the virtualized environment.
When a NFV package is submitted, the NFVO processes the VNFD to check the presence of mandatory elements and validate integrity and authenticity, then notifies and updates the VNF catalog and makes software images available to each applicable VIM. When a NS package is submitted the NFVO primarily checks the presence of mandatory elements, and validates integrity and authenticity. Then it checks the presence of VNF packages for the VNFs that are part of the Network Service and in those packages the presence of the external connection points defined in the NS package. At this stage NS catalog is updated.

- **Instantiation.** In this phase the required resources are identified and reserved at the NFVI, and the VMs constituting the VNFs are started. After the instantiation request is done and validated by the VNFM, either the VNFM or the NFVO contact the VIM and request for resources allocation. VIM allocates the internal connectivity network and instantiates the VMs on the infrastructure, according to the parameters specified in the descriptors. VNF Manager configures the newly created VNF with any specific lifecycle parameters over the Ve-Vnfm configuration interface, eventually notifying the EM (if present) of the procedure completion.
  In the case of a NS instantiation request, at first the NFVO checks if any of the required NFVs are already deployed in the system; if some are missing they are instantiated according to the previous procedure. To complete the procedure virtual links between VNFs are created according to the NS descriptor, and where required a connection between the VNFs external interfaces and the physical network is created.
  After deployment the VNFs start providing their services and become available for run-time reconfiguration or scaling if required.

2.3 Multi-access Edge Computing (MEC)

Multi-access edge computing provides IT and cloud-computing capabilities within the Radio Access Network (RAN) in close proximity to mobile subscribers, making feasible to provide services, store and process content, at the network's edge. The RAN edge offers a service environment with ultra low latency and high bandwidth as well as direct access to real-time radio network information, increasing the responsiveness of the system. This paradigm, first proposed in 2014 by ETSI [6], is made possible by a highly distributed deployment of cloud servers running at the edge of the mobile network and performing specific tasks that could not be achieved with traditional network infrastructure. This framework allows the network to move a lot of the computational burden and data exchanges at the edge, which can reduce congestion while also improving users' quality of experience and being able to provide customized services based on location. Aside from the closeness to user devices, which considerably reduces communication latency, other important characteristics of MEC servers are the location awareness and the possibility to take advantage of context information about users connectivity, making possible to react upon radio link quality variations.

MEC is considered to be a key enabler for 5G technologies, aiming at increasing the pervasiveness and responsiveness of the network; two major use cases are the URLLC and mMTC network slices, the first requiring ultra-reliable and low-latency communications and the second aiming to provide connectivity to a great number of IoT devices in a defined area. Nonetheless, it is of interest the realization of the MEC paradigm in 4G networks too, also in a perspective of reusing the acquired technical knowledge during the transition to 5G networks. Use cases of interest include high bandwidth or computationally heavy services which could take advantage of the closeness between consumer and provider, such as augmented reality content delivery or video analytics for public safety.

2.3.1 Architectural framework

ETSI provides an high-level functional architecture for MEC framework [7] with the same premises of open standards and interoperability between multi-vendor services to which also NFV adhered, in order to encourage the development and integration of new cutting-edge applications to exploit MEC potential at its best.

The involved general entities can be grouped into system level, host level and network level components. Figure 2.6 shows the MEC framework, consisting of the MEC hosts and the management infrastructure necessary to run edge applications within an operator network; its components will be presented in the following.

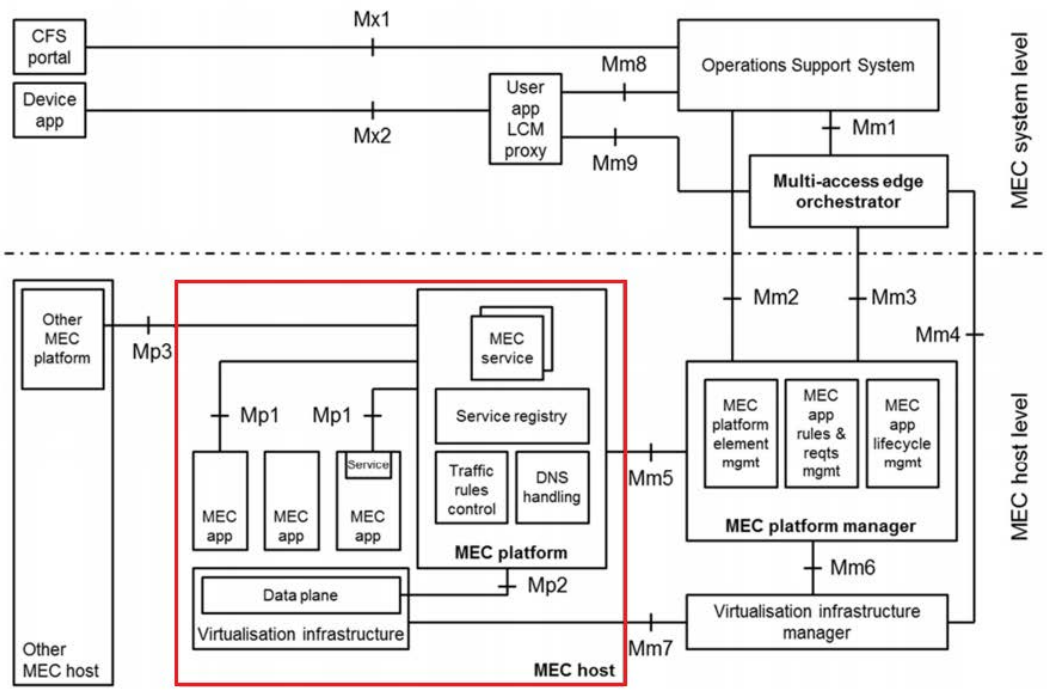


Figure 2.6: Multi-access edge system reference architecture. The MEC host, main executive environment, is highlighted. From [7].

MEC host

The MEC host contains a MEC platform and a virtualization infrastructure which provides compute, storage and network resources, for the purpose of running MEC applications. The virtualization infrastructure includes a data plane that executes the traffic rules received by the MEC platform, routing the traffic among applications, services and external networks. The host is the main application server to store, process, and provide content in MEC framework.

MEC platform

The MEC platform is the collection of essential functionalities required to run applications on a particular virtualization infrastructure and enable them to provide, discover and consume MEC services. It is also responsible for receiving traffic rules from the MEC platform manager, applications or services, and instructing the data plane accordingly to differentiate the traffic remaining at the edge from that directed toward the core network. It communicates with the MEC applications, the virtualization infrastructure, and other MEC platforms, via interfaces belonging to the "Mp" group.

MEC applications

They run as virtual machines on top of the virtualization infrastructure provided by the MEC host, and can interact with the MEC platform to consume and provide MEC services. They are connected to the platform via the Mp1 interface, which enables service registration and service discovery, along with traffic rules and DNS rules activation, access to persistent storage and time of day information.

MEC system level management

It contains the multi-access edge orchestrator, the core control node which has an overview of the complete MEC system: it possesses a knowledge of deployed MEC hosts, system topology, available resources and MEC services; it supervises the on-boarding of application packages, and triggers application instantiation, relocation, termination. The other actors involved are the OSS of the operator and the user application lifecycle management proxy, allowing the instantiation of user-required applications in the system (when permitted). This entity exposes interfaces of the "Mx" group to communicate with external entities and interfaces of the "Mm" group that provide management connections with other MEC elements.

MEC host level management

It handles the management of a particular MEC host and the applications running on it. It comprises the MEC platform manager which deals with the lifecycle of applications, their rules and requirements (e.g. service authorization and traffic rules configuration), and the virtualization infrastructure manger (VIM), which allocates, manages, releases virtualized resources enabling the deployment of software images, as for NFV systems. This entity exposes interfaces of the "Mm" group for management flows.

MEC and NFV can be considered complementary concepts, having as a common ground the deployment of services and applications on a virtualized infrastructure and the presence in the system of a central entity carrying out management and orchestration tasks. A variant on the MEC architectural design considers therefore the opportunity to deploy the MEC system in a NFV environment, allowing to instantiate MEC applications and NFV functions on the same virtualization infrastructure, and to reuse NFV MANO components to fulfill a part of the MEC management and orchestration tasks. Figure 2.7 represents the variant architecture integrating NFV and MEC components. In this framework the MEC platform and the MEC applications are deployed as VNFs controlled by the MANO, the virtualization infrastructure is deployed as a NFVI, and the MEC control entities are

designed to interoperate with NFV ones. The interfaces are a mingling of the ones present in MEC and NFV frameworks, together with some new connections ("Mv" group) to grant operational compatibility between the two systems. A simplified version of this particular configuration will be adopted in the present work.

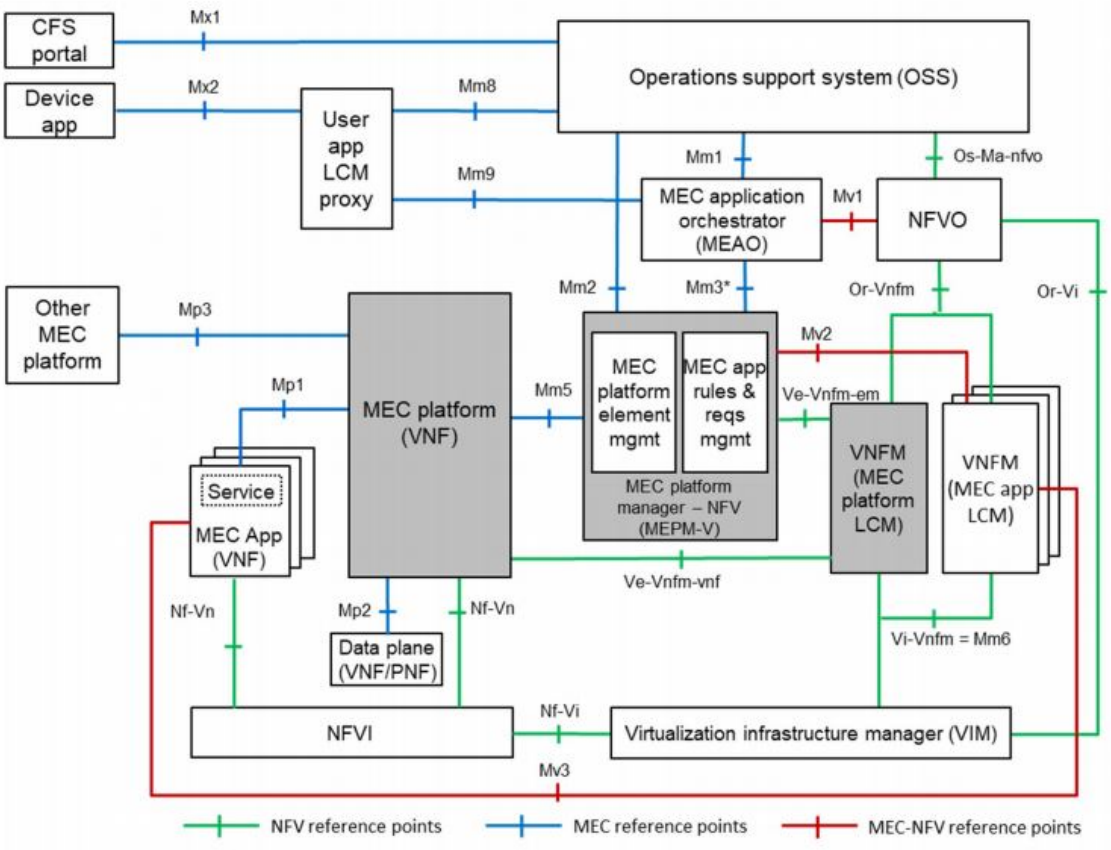


Figure 2.7: Multi-access edge system reference architecture variant for MEC in NFV. From [7].

2.4 Related Work

The present work pushes forward the study carried out in [8], which analyzed the deployment of a multi-site NFV system including MEC services. Procedures for creation and configuration of a MEC system were investigated and the suggested workflow partly served as reference for the implementation to be outlined in the next sections, though our work further explores automation processes and integration with external services.

Another experimentation about a multi-site NFV deployment can be

found in [9]. In this work the authors studied the feasibility of deploying multiple network services at different points in the network under the control of a centralized management entity. The claim is that the presented implementation for the MANO entity is effective in instantiating and orchestrating spatially distant functions, while also giving the chance to separate control and data traffic flows between VNFs.

A recent work [10] analyzes the current state-of-the-art on MEC and NFV integration, presenting the strong points of this approach, which grants a very suitable environment for dynamic deployment and scaling of applications, and highlighting the gaps that still exist for a complete integration between the two paradigms. A full description of the workflow to implement an edge robotics scenario is also provided, describing the different role the management entities for MEC and NFV have and how they interact.

Application of the MEC environment to latency reduction for a high-quality video streaming is explored in [11], with a dynamic adaptive streaming service provided at the edge. The computational capacity and the context-awareness available at the edge are leveraged to support a seamless streaming service robust to bandwidth fluctuations, trying to avoid stalls during reproduction. The network function management framework was however not fully specified and it is not clear to which extent the interoperability requirements are fulfilled.

Chapter 3

Design of a solution for NFV and MEC integration in a 4G mobile network

The objective of this chapter is to present a possible architectural design for the deployment of a 4G core network in a NFV environment, and successively integrate MEC services in the aforesaid framework. The first experimental study is carried out on the NFV system to explore the potential this virtual environment offers for network automation; the second study is carried out on the MEC system to examine the procedures for automated activation of a local service with traffic steering at the edge. For both of the studied scenarios, at first an high-level system architecture is outlined, followed by a detailed description of deployed network functions, their structure, role and connections with other functions; eventually the devised mechanisms to start providing the expected system functionalities are illustrated.

3.1 4G core network in NFV framework

In the considered environment, the core network (EPC) has been modeled as a single network function containing all the processes and functionalities of its non-virtualized counterpart, exposing SGi and S1 interfaces for external connectivity. To simplify the configuration of connection points, the EPC presents a unique S1 interface toward the E-UTRAN, performing both control plane and user plane tasks. To integrate the virtual EPC into the NFV framework, the architecture must include a NFV infrastructure providing hardware on which to deploy the virtual core network, and the MANO entity. For implementation purposes, in this work the VIM has been conceived as a separate entity respect to the other management components: from now on, we refer to MANO as the combination of NFVO and VNFM, not including VIM; however, the overall management functionality is assured

due to data exchanges between the two entities. More details to be given in chapter 4. NFV MANO is responsible for initial deployment and lifecycle management of the core VNF; configuration of the instance is performed via the Ve-Vnfm interface. Figure 3.1 depicts the conceived scenario.

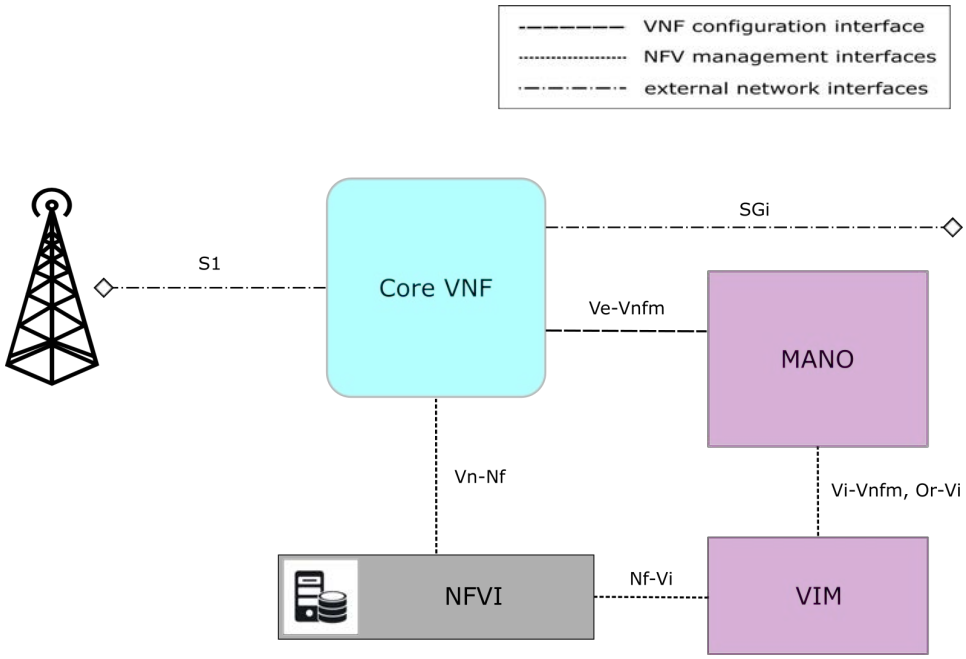


Figure 3.1: Functional architecture of a mobile network deployed in a NFV environment.

The Core VNF implements an EPC and is internally made up of the corresponding four components: MME, HSS, SGW, PGW; in addition, an element manager (EM) is embedded into the function, for control purposes. The internal components are deployed starting from proprietary software images developed and provided by Athonet. The internal connection points refer to the S6a, S11, S5/S8 interfaces that the EPC does not expose, and to the management interface between the EM and the other components. The external connection points refer to the S1 and SGi interface for EPC connectivity and to the Ve-Vnfm interface, coupled with an additional control interface, to exchange management information with the EM from outside the VNF. The internal structure along with the connection points is shown in figure 3.2.

The element manager plays the main configuration role for the VNF, allowing to instantiate the function with the desired parameters and functionalities or providing a run-time reconfiguration facility on demand. The EM receives configuration specifications from the MANO entity via the Ve-Vnfm interface and takes care of their application to the VNF; also it is possible to communicate directly with the EM via the control interface it

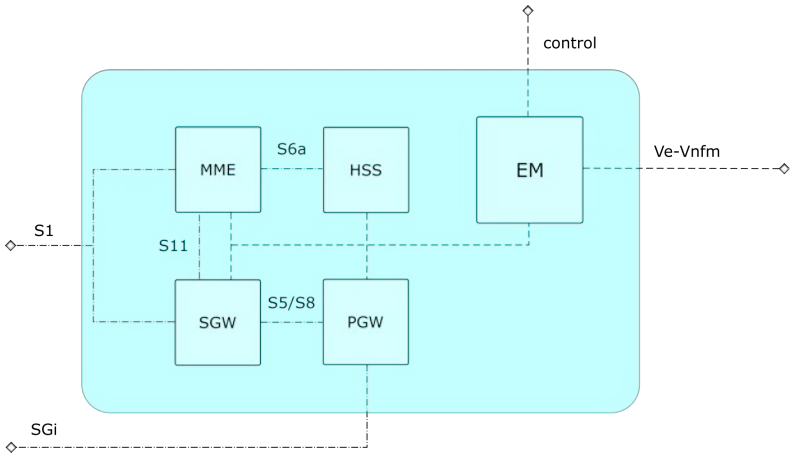


Figure 3.2: Core VNF internal strucure.

exposes on the external network, to facilitate the user interaction with this entity and ease the function reconfiguration processes. The communication modes differ according to the used interface, and will be further detailed in chapter 5. Both modalities are designed to allow the correct configuration of the LTE network exploiting the functionalities provided by the NFV framework.

As suggested in [8], it has been chosen to integrate in the VNF structure a virtual router (as a VNF component) to collect the external connection points and avoid them to be exposed on the physical network, as depicted in Figure 3.3. This approach is intended for isolation improvement: exposing all interfaces on a public network may cause security issues, and the introduction of a router (which may implement a firewall) can enhance security policies with packet based filtering. By doing so, the only exposed interface on the network is the router external interface, and all traffic is exchanged and filtered over it.

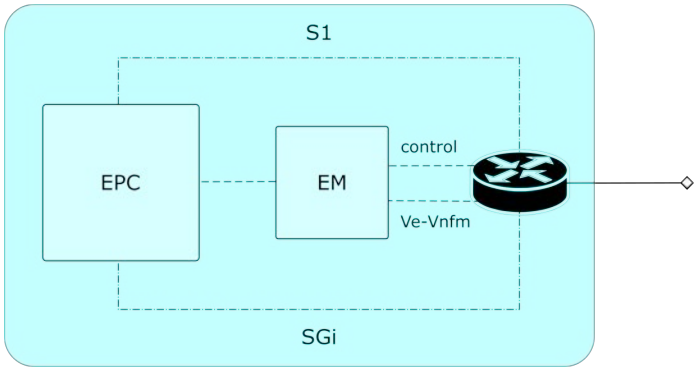


Figure 3.3: Core VNF internal strucure including the virtual router. For drawing clearness the components are condensed in a single EPC entity.

3.2 MEC services for 4G mobile network

A key functionality of the MEC platform is to route IP packets to MEC applications that are meant to provide a service, which are either hosted locally on the platform or on a distinct server. The traffic steering to/from MEC applications is achieved by configuring appropriately the MEC host's data plane. The way to implement the traffic redirection is however highly dependent upon the location of the MEC host respect to the 4G architecture; referring to [12], four different deployment scenarios for a MEC system can be considered:

- **bump in the wire** scenario: the MEC host can be located in any position between the base station and the core network. This deployment is convenient for its flexibility but presents some issues related to the processing of data packets, which are encapsulated using tunnelling protocols on the S1 interface.

- **distributed EPC** scenario: in this deployment the MEC host logically includes all or part of the EPC components. It is less impacting on the operator's network, and a local copy of the EPC provides more responsiveness for applications that demand high performances.

- **distributed SGW/PGW** scenario: similar to the previous scenario, except that only SGW and PGW entities are deployed at the edge site, whereas the control plane functions remain at the core. This architecture allows selective offloading of the traffic at the edge based on the PGW choice, and permits the operator to retain full control over the MME.

- **distributed SGW with Local Breakout (SGW-LBO)** scenario: in this deployment the MEC host at the edge is co-located with a SGW providing traffic steering capabilities. The SGW is provided with an additional local-breakout SGi interface which supports traffic separation, allowing the users to reach both the MEC applications (via the breakout SGi interface) and the operator's core site applications (via the conventional S5/S8 interface) in a selective manner through a unique gateway. The decision about whether to route the traffic to local applications or forward it to the core PGW can be performed on the basis of any operator-based combination of policies, such as user identifier or IP parameters. The distributed SGW scenario is depicted in figure 3.4.

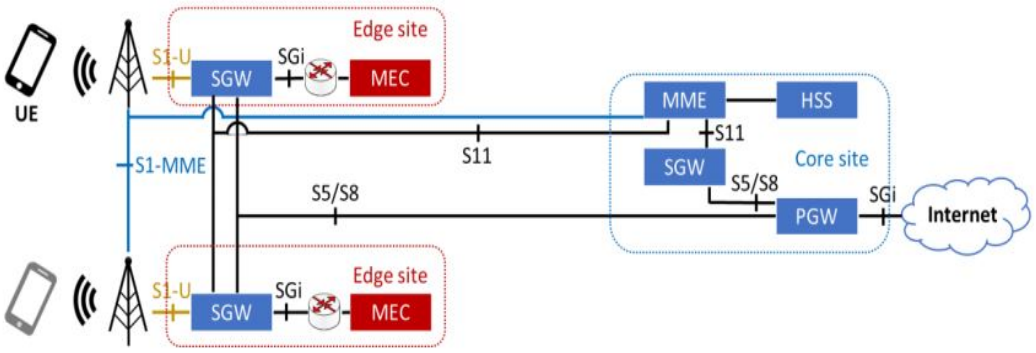


Figure 3.4: Distributed SGW in local breakout scenario. From [12].

3.2.1 MEC system architecture in NFV framework

A multi-site NFV framework has been designed to deploy a MEC system in accordance with the SGW-LBO scenario. The experimental framework (outlined in figure 3.5) is composed by three different sites: the management site, the core site, the edge site; each of them is installed in a different physical location but logically connected to the others. This flexible positioning option is made possible by the multi-site capacity of MANO, which enables the deployment and control of network services across multiple NFVIs, as long as they are under the control of compliant VIMs.

The management site consists of the MANO component, which has the overview of the entire NFV system and exchanges control information with all the other entities. In particular, it communicates with the VIMs located at the edge and core sites to manage virtual resources and deploy network services, and directly with the deployed functions for lifecycle management (through the Ve-Vnfm interface). To secure inter-site control communications the use of a VPN on top of the underlying network is suggested.

The core site contain a NFVI to instantiate VNFs and a VIM to orchestrate its resources. It hosts a Core VNF and an App VNF, to be described in the next section.

The edge site is located at the E-UTRAN and an eNodeB is part of the connectivity environment. As the core site, it comprises a NFVI and a VIM, together with an Edge VNF and an App VNF, to be described in the next section.

This multi-site deployment realizes a distributed EPC, with the core site accomodating MME, HSS, PGW components, and the edge site providing an SGW that may work in local-breakout mode. For experimental study, two application servers have been included into the system, providing access to an arbitrary kind of content; they are deployed as App VNFs and integrated in the aforementioned environment. In this study, the App VNF residing at

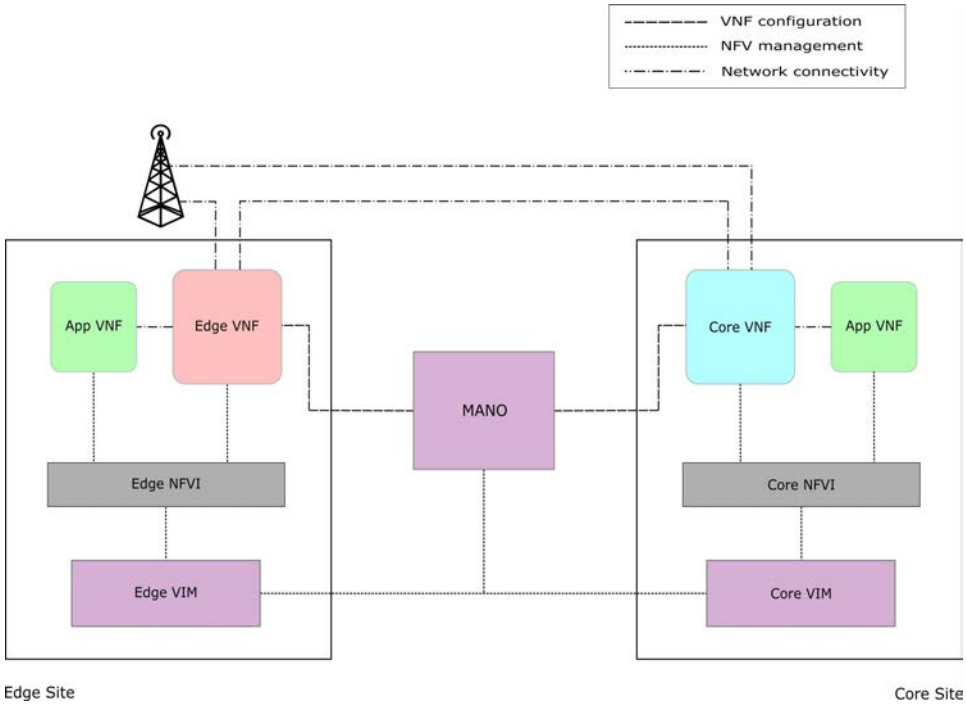


Figure 3.5: MEC system architecture in NFV multi-site framework.

the core site implements a remote server hosting a video streaming service and accessible via the core network, whereas the App VNF residing at the edge site is a local cache for the remote server, providing a copy of its content as a MEC service. A user can request content either to the remote server, to be retrieved via the SGi interface of the core PGW, or to the local copy of the server, to be retrieved via the SGi interface of the SGW-LBO exploiting its traffic steering capabilities.

To provide integration between MEC and NFV technologies at the edge site, the SGW-LBO is conceived as a MEC platform through which the App VNF (MEC application) can provide its services. Both are equipped with the Mp1 control interface, to enable service registration and traffic rules activation. This work does not consider a totally integrated NFV-MEC platform, but relies on a simpler virtualization framework, which could be extended to become a full NFV-MEC platform. In fact, all the main management tasks are performed by the MANO, lacking the presence of a MEC platform manager and a MEC application orchestrator.

3.2.2 Network Functions description

The multi-site network service consists of four components simulating a real-world connectivity scenario. The VNFs deployed at core and edge site together make up a full EPC with spatial separation of the internal constituent

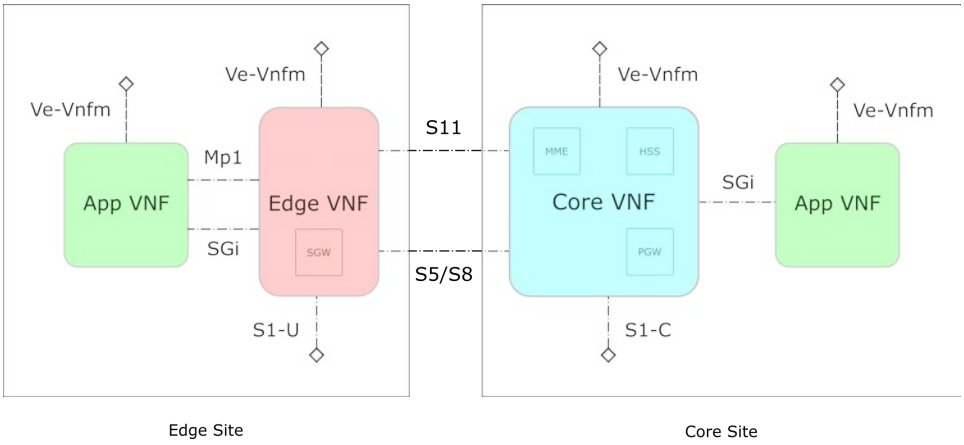


Figure 3.6: Network functions composing the MEC system and associated external connection points.

elements; the S5/S8 and S11 interfaces are no longer included in the VNF context as it was for the previous study, due to the different position of the SGW respect to the rest of the EPC: it is then mandatory the exposure of two connection points connecting the sites and implementing the said interfaces. Two similar Application VNFs are instantiated at both sites, used as external hosts providing services. Figure 3.6 shows the four network functions along with the exposed connection points and their mapping to logical interfaces. Follows a detailed description of the deployed functions focusing on their role, internal components, connection points.

Core VNF

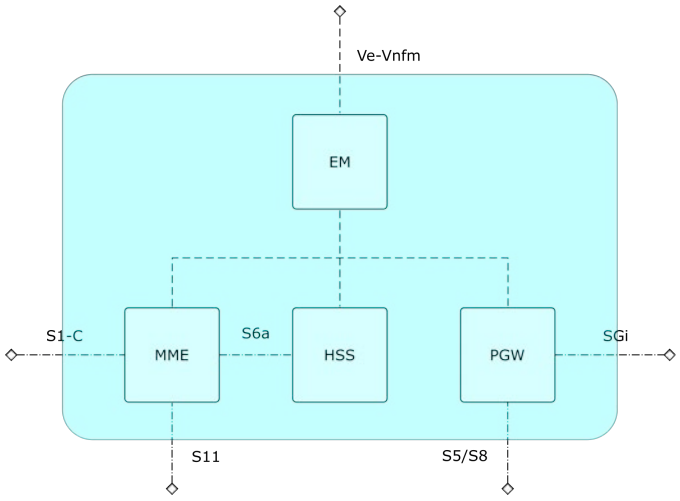


Figure 3.7: Core VNF internal structure and connection points.

It implements the core network providing the MME, HSS, PGW components, together with an element manager for configuration purposes. It exposes the SGi interface to enable data plane connectivity toward external applications, and the S5/S8 interface to exchange user data with the SGW located at the edge site. It also provides connection points for control plane traffic: the S11 interface for MME-SGW communications, and the S1-C interface for communications toward the E-UTRAN. The Ve-Vnfm interface is exposed from the EM towards the VNFM, allowing for initial and run-time configuration of the VNF. Its internal connection points correspond to the S6a interface for MME-HSS communications, and management interfaces between the EM and the various components.

For isolation improvement, the approach already described of integrating a router collecting the external connection points and exposing its own interface on the public network has been adopted.

Edge VNF

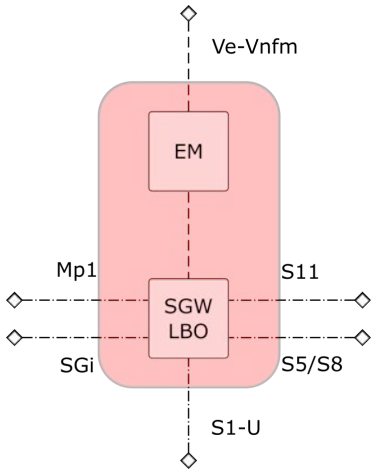


Figure 3.8: Edge VNF internal structure and connection points.

It implements the edge network providing the SGW-LBO and an element manager; it presents a unique internal interface, between these two components. Regarding the user plane traffic, it exposes three interfaces: the S1-U interface, to communicate with the E-UTRAN and provide content to the users, the SGi interface, to access content from the edge application if the traffic switch is enabled, the S5/S8 interface, to communicate with the core PGW and access other remote networks not offering edge services. With respect to the control plane, the S11 interface is exposed toward the MME, and the Mp1 interface is exposed toward the local MEC application to manage offering modalities for the services. The SGW-LBO component is deployed

starting from an Athonet proprietary software image.

For isolation improvement, the approach already described of integrating a router collecting the external connection points and exposing its own interface on the public network has been adopted.

Application VNFs

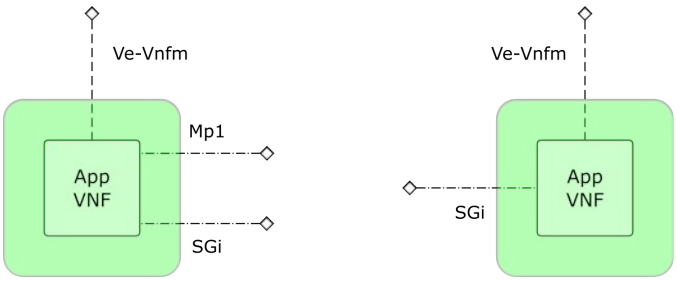


Figure 3.9: Application VNFs, deployed at edge site (on the left) and at the core site (on the right).

The App VNF deployed at the core site (shown on the right in figure 3.9) implements a simple server hosting an arbitrary content; it exposes the SGi interface toward the core VNF to make its content accessible to mobile users, and the Ve-Vnfm toward the VNFM for configuration purpose and monitoring of performance indicators for the VNF.

The App VNF residing at the edge site (shown on the left in figure 3.9) is a clone of the core application, providing the same service though in a more advantageous location. It exposes the Mp1 and SGi interfaces toward the edge VNF for control and user plane connections with the SGW-LBO, and the Ve-Vnfm interface toward the VNFM.

3.2.3 Use cases and MEC service activation

The proposed framework is designed to enquire the performance improvements offered by traffic redirection from a remote server to a server placed in a location closer to the final user. This scenario could account for any application running in a cloud environment respecting the architecture outlined in the previous sections; without loss of generality, a video streaming service has been used in the present work. The choice fell on this application for the simplicity also for a human user to appreciate the differences in experienced quality when enjoying a multimedia content, giving a clear proof of concept of the devised system.

The decision to deploy a MEC system arises from the performance degradation inherent in content retrieval from a remote server residing in a generic network location: above all, the link between the E-UTRAN and the core network (which may need to cover great distances) introduces a non-negligible

delay and presents an available bandwidth which is typically lower respect to the bandwidth offered by the link between the user and the edge network. Moreover, traffic redirection can mitigate network congestion problems.

As depicted in figure 3.10, in a conventional (non-breakout) communication scenario the user plane traffic flow coming from the S1-U interface is forwarded to the PGW in the core site by the SGW-LBO working as a conventional SGW, via the S5/S8 interface. The PGW then reaches the application server via the standard SGi interface. When instead a local copy of the server is present at the edge, the breakout mode can be activated and the traffic redirected toward the edge application: in this case the traffic flow remains within the access network, leveraging the SGi interface exposed by the SGW-LBO to reach the server. Control plane communications keep occurring between edge and core sites: they do not affect performances of the user plane flows.

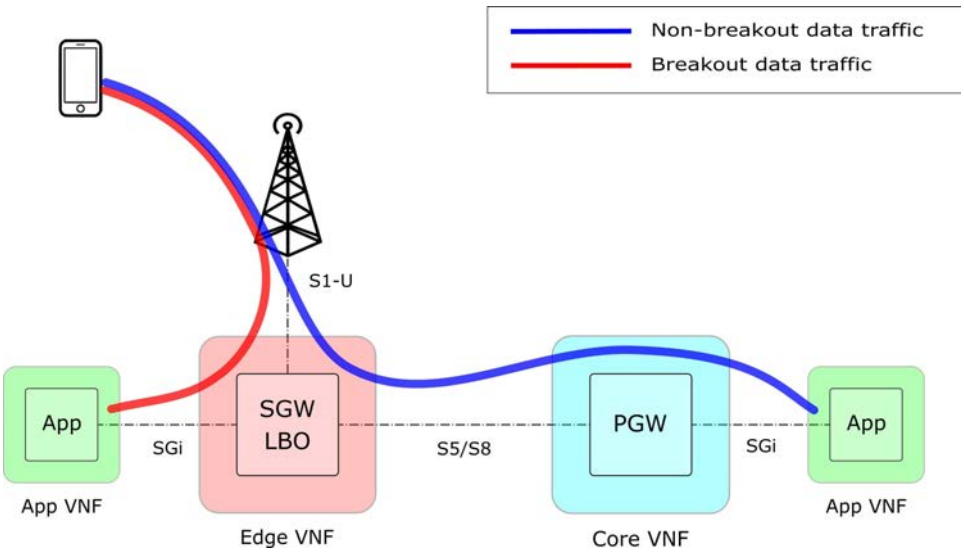


Figure 3.10: User plane traffic flows in non-breakout (blue route) and breakout (red route) mode.

To simulate a real-case scenario of MEC services activation and access, the application at the core is provided with a performance monitor able to signal whether the offered service is experiencing a quality lower than required (according to either bandwidth or latency metric); if that happens, the application triggers the cloning of itself and the deployment of a SGW-LBO at the edge site to start providing the same service closer to the user.

The initial deployment procedure includes the instantiation of the core VNF (comprising a conventional SGW component) and the core application; the edge VNF and the edge application will be instantiated at a later time, on demand. After the core network service is deployed, MANO subscribes to

a monitoring service provided by the core app, by which the app cloning is requested: as soon as the experienced service quality drops and remains below a given threshold for a certain amount of time, MANO is notified by the application. MANO is in charge of the executive operations for the cloning procedure: the request to spawn a new edge NS is validated and the availability of necessary resources at NFVI is verified; the SGW-LBO and the application are deployed at edge site and properly configured. In addition, the core EPC is notified about the deployment of the external SGW and a network reconfiguration is performed to establish the connectivity toward the edge site. Thereafter, through the management interface MANO will add a traffic rule to the MEC platform (the SGW-LBO) in order to enable the traffic switch toward the edge application. In the meantime the cloned application starts the registration procedure at the MEC platform, to enable the new traffic rule and begin delivering its service. All the devised procedures are meant to be entirely automatic, not requiring the intervention of a human operator but being the result of the network reacting to its current conditions, in an effort to provide a better user experience.

As the MEC functionality is fully set up, any authorized user requesting content to the application will be served by the local copy of the server, making it easier to comply with the requested quality of service. The traffic rules registered in the MEC platform database specify which kind of traffic must be redirected to the edge and can take advantage of the local breakout capabilities; traffic matching any of the steering rules will enjoy local breakout, while all the other connections will exchange data in the conventional way, traversing the usual path toward the core. In the present study the traffic rules perform a selection based only on IP parameters, but also a selection on the allowed users or other parameters could be chosen, with the MEC platform carrying out customizable authentication and authorization procedures.

Chapter 4

Description of the tools

To deploy the systems outlined in the previous sections, it has been necessary to identify suitable implementations for the NFV management and orchestration entities. The main features that have been discriminant in making the choice are the alignment with ETSI specifications, the maturity and stability of the solution, and the type of license. Also taking as reference the work in [8] [9], the choice fell on two software implementations which combined together provide a full NFV MANO functionality.

Open Source MANO has been chosen as the baseline technology to build the NFV system, implementing a full MANO stack comprising the VNFM and NFVO entities. It supports the integration with many compliant VIMs, among which we chose the OpenStack solution. Both of them are open source software distributions, offering a favourable framework for experimentation and research, and taking advantage from the constant community contributions. The following sections describe these tools, focusing on the relevant functionalities for the creation of an experimental NFV scenario.

4.1 OpenStack

OpenStack[1] is a cloud computing platform, providing an Infrastructure-as-a-Service (IaaS) solution for both public and private clouds. Its aim is to manage hardware pools and provide them as a virtual infrastructure on which users can deploy their own services or networks. It can operate as a VIM in the NFV MANO framework, orchestrating NFVI resources and on top of them instantiating VMs and virtual networks; instructions about the actions to perform on the resources are conveyed by the other high-level management entities. OpenStack version *Stein* was used for this work.

OpenStack project features a collection of microservices, each delivering a specific functionality, which can be combined together to provide the needed

[1] https://openstack.org/

capabilities. A custom OpenStack installation may be composed by an arbitrary number of microservices chosen from the more than thirty available, which exploiting the APIs they provide can be easily integrated. The OpenStack installation used for the present work features five modules (shown in figure 4.1), to be described in the following.

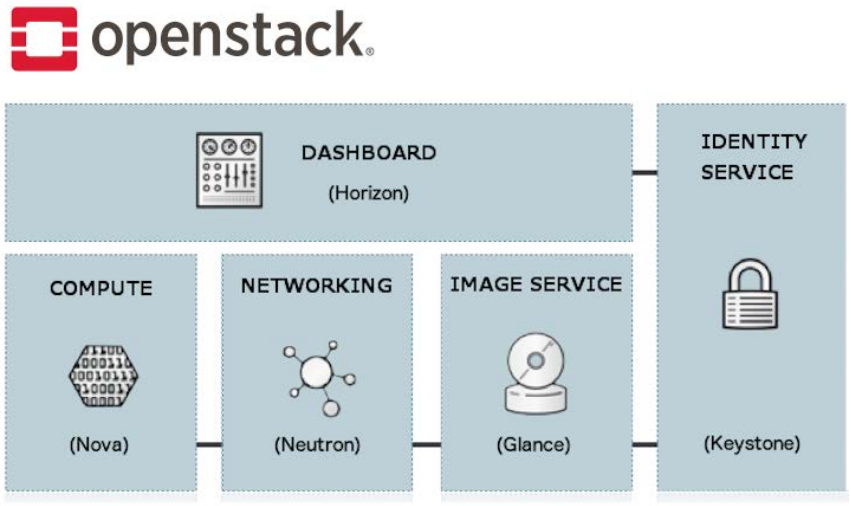


Figure 4.1: Service architecture for the considered OpenStack installation.

Keystone (Identity service)
Keystone service implements the OpenStack's Identity API and provides client authentication and authorization. Upon validation of inserted credentials the user is given access to the other available services.

Glance (Image service)
Glance service allows discovering, registering and retrieving virtual machine images. A VM image is a virtual hard disk file that is used as a baseline template for creating virtual machine instances, providing all the system functionalities included in the image available for the VM after the deployment procedure. Glance has a RESTful API that allows querying of VM image metadata as well as retrieval of the actual stored image. VM images made available through Glance can be stored in a variety of locations, from simple filesystems to object-storage systems.

Nova (Compute service)
Nova is the service which enables the provisioning of compute instances. It permits the creation on demand of virtual machines on top of the virtual

infrastructure; each instance is launched starting from a software image retrieved through the glance service. This service supports integration with the most common virtualization technologies and hypervisors to manage the resources in the hardware pool.

To deploy a new virtual machine from a software image it is required to provide a 'flavor', namely parameters defining the compute, memory, and storage capacity for the instance to be launched, together with a network to locate the VM and information about its connection points.

Neutron (Networking service)
Neutron service provides network connectivity between devices managed by other OpenStack services, such as Nova. It handles all facets about the creation and management of a virtual networking infrastructure, realizing arbitrarily complex network topologies and implementing services such as firewalls and DHCP. Neutron provides networks, subnets, and routers as virtual abstractions, each of them having functionalities that mimic the physical counterpart: networks contain subnets, and routers route traffic between different subnets and networks.

Any Neutron setup presents one or more internal networks, to which the deployed VMs are attached through connection points called ports; to access a VM connected to an internal network it is necessary to lie on the same network or rely on a virtual router providing a port on the given network. Besides, it is mandatory in the setup the presence of at least one external network, which unlike the other ones is not virtually defined but a physical network accessible outside the OpenStack installation. Internal and external networks are mutually isolated. For the outside network to access VMs located in an internal network, and vice versa, a router between the external and internal networks is required, exposing interfaces on both of them: this gateway can allocate floating IP addresses on external networks and associate them to ports on the internal network. This way, entities on the outside network can access deployed VMs.

Neutron also supports security groups, which enable administrators to define applicable firewall rules. A VM can belong to one or more security groups, whose rules serve to block or unblock ports, or traffic types, for that VM.

Horizon (Dashboard)
Horizon is Openstack's dashboard, providing a web based user interface to interact with available services. It allows the user to manage the active projects, performing operations such as launching or destroying instances, creating and configuring virtual networks and connectivity for the instances.

4.2 Open Source MANO

Open Source MANO (OSM)[2] is an ETSI-hosted project which delivers a functional implementation of a MANO software stack aligned with ETSI NFV specifications. It realizes the NFVO and VNFM functionalities and can relate with compliant VIMs (e.g. OpenStack, OpenVIM, VMware, Amazon Web Services) to deploy the requested network services on the underlying virtualization infrastructure. It also supports multi-site deployments, as long as every infrastructure is controlled by a VIM. The latest stable version, *OSM Release SEVEN*, was used.

4.2.1 Architecture

OSM features a modular design [13], each component providing specific functionalities and connections toward other components, while keeping its operational independence. The three core modules that realize the MANO capabilites are described in the following; their connections and relations with external entities such as the VIMs and the VNFs are shown in figure 4.2.

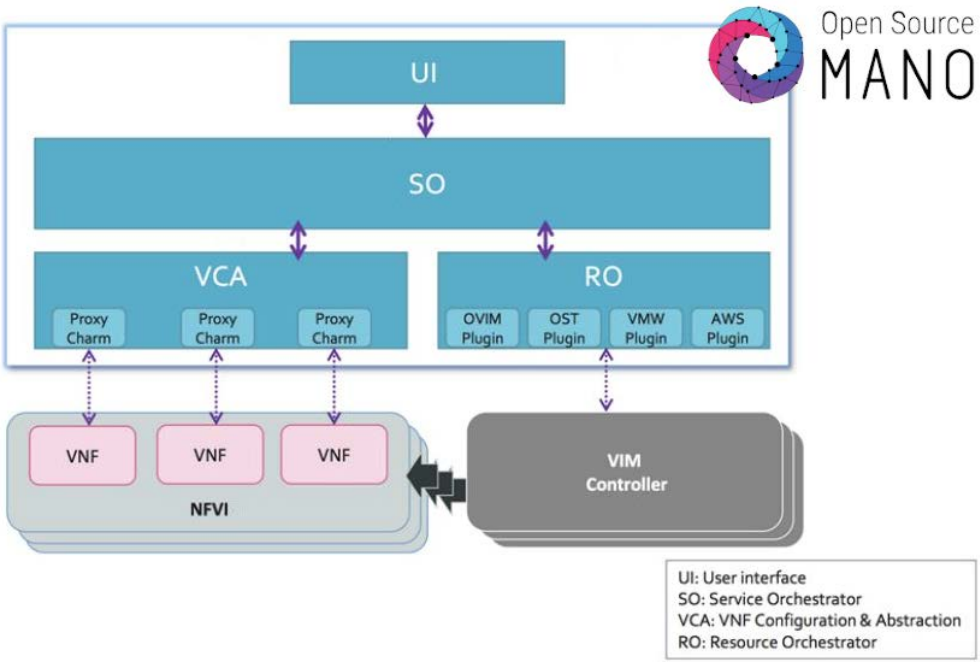


Figure 4.2: OSM core architecture and external management connections.

[2]https://osm.etsi.org/

Service Orchestrator (SO)
The service orchestrator (in the latest releases replaced by a lighter version, the Lifecycle Management component, LCM) is the highest-level control node, realizing the NFVO functions. It supports the lifecycle management of network services, coordinating the creation, configuration and deletion of services composed of multiple VNFs, interacting with the RO and the VCA modules. Additionally, it provides other enabling functionalities, such as the management of NS/VNF descriptors and packages.

VNF Configuration and Abstraction (VCA)
The VCA module implements the VNF Manager defined by ETSI, supporting the configuration of VNFs after deployment and enabling the notifications from VNFs or Element Managers to the OSM platform. To support the configuration services VCA makes use of an external application modeling tool, Juju[3]. This tool allows the configuration and scaling of cloud applications through the execution of software scripts, called proxy charms, which can be included in the on-boarded VNF packages.

Resource Orchestrator (RO)
The RO module requests and supervises the allocation and configuration of computing, storage and network resources under the control of one or multiple VIMs, in order to support the execution and interconnection of VNFs. It can orchestrate resources across different supported VIMs utilizing a set of plug-ins to interact with the specific interfaces they expose.

OSM, in addition to a command line client, also provides a handy graphical web user interface, through which it is possible to interact with the MANO services. Among the available options, the user can deploy, scale, or destroy VNFs or network services, perform VNF run-time configuration or monitoring, manage the registered VIMs.

4.2.2 Instantiation procedures

In order to deploy network functions and network services as cloud applications in the VIM environment (from now on assumed to be OpenStack), OSM follows the on-boarding and instantiation procedures outlined in chapter 2.

The first step regards the on-boarding of the VNF and NS packages describing the desired structure and functionalities. Every VNF package consists of a software image (stored at VIM), which serves as foundation for the virtual machines to deploy, and a descriptor adding all the information for correct initial setup of the new instance; optionally, a configuration agent

[3]https://jaas.ai/

such as a Juju charm may be included. The NS packages are instead made up of a descriptor only, referencing to already on-boarded VNF packages and specifying NS composition and end-to-end functionality. The packages, compressed in .tar.gz archives, can be on-boarded through the web user interface. Once validated, they are inserted in OSM catalog and become ready for deployment.

Open Source MANO supports the instantiation of network services only, thus it is necessary to embed all the required VNFs in a NS - by referencing the corresponding packages in the NS descriptor. The instantiation can be requested via the web interface; however, for advanced functionalities the OSM client is preferred. Following the instantiation request, via the RO module OSM notifies the VIM, which puts into effect the request by handling the resources of the underlying NFVI. Instantiating the network service launches all the VNFs it references, creates the necessary virtual networks and attaches connections points. Successively, the initial configuration operations specified in the VNF packages are performed. At this stage the network functions are ready to deliver their services and open to successive reconfigurations via the Juju charms.

4.2.3 Descriptors

Descriptors are used to model and automate the full lifecycle of network functions and network services; as such, they are a necessary component to enable their deployment in OSM environment. The descriptors are expressed in YAML format (a human-readable data-serialization language in key/value pairs) and must be written and validated according to an ETSI-defined information model[4], which specifies all acceptable configuration parameters together with their role and value type.

VNF descriptors
Virtual network function descriptors (VNFD) are deployment templates used to describe the attributes of a single VNF. The information contained is used by the RO and the VCA modules for resources allocation and initial configuration. An overview of the main fields of the descriptor is now presented; an example of VNFD can be found in Appendix A for greater clarity.

- Identifiers: the parameters used to univocally define the VNF. They include id, name, vendor, and an optional description.

- Connection points: a list of connection points (virtual interfaces) exposed by the VNF on the networks. It is also used to enable or disable their port security in the VIM.

[4] Accessible at: https://osm.etsi.org/wikipub/index.php/OSM_Information_Model in tree representation.

- Management interface: it defines the connection point through which the VNF is managed by OSM (Ve-Vnfm interface). It can be specified referring to an already defined connection point or by indicating a reachable IP address.

- Internal virtual links descriptors (VLD): used to describe the connectivity between internal VNF components. A list of virtual links connecting the components, describing the respective topology and the attached connection points together with their IP addresses.

- IP profiles: describe the IP characteristics for the internal virtual links (IP version, subnet associated to the link, default gateway, DNS server, DHCP parameters, security groups).

- Virtual deployment units (VDUs): a list of virtual machines implementing the VNF; they correspond to the internal VNF components. For each VDU it is mandatory to specify the respective identifiers, the software image from which to launch the VM (to be found in the VIM), the deployment flavor (the amount of CPUs, RAM and disk space to be allocated for the VM), and the exposed interfaces, corresponding to internal or external connection points.

- VNF configuration: this section is used to specify configuration options for the VNF. It includes the name and parameters of the executable configuration primitives (namely, actions to be performed) and must reference a juju charm or a script which define their implementation.

NS descriptors

Network service descriptors (NSD) are deployment templates describing the high-level functionality of the system, the connectivity between its composing entities (the VNFs), and information concerning the connection of virtual functions to the external networks. The user may instantiate a NS starting from the corresponding descriptor; the SO module is in charge of verifying the existence of the VNF packages referenced in the descriptor and requesting the creation of instances, virtual links and connections on the external network, engaging the RO module. An overview of the main fields of the descriptor is now presented; an example of NSD can be found in Appendix A for greater clarity.

- Identifiers: the parameters used to univocally define the NS. They include id, name, vendor, and an optional description.

- Constituent VNFDs: reference to the descriptors of VNFs constituting the NS.

- Connection points: characterization of the external connection points exposed by the constituent VNFs, outlined in the respective descriptors. For each point it can be requested the assignment of a floating IP (chosen among those allocated to the VIM) to provide attachment of the NS to a physical external network.
- External virtual link descriptors (VLD): used to define the virtual links exposed by the NS on the external networks or created for VNF connections inside the NS. It is possible to specify an existing network or to create a new one in the VIM where to attach the VNFs interfaces. For every VNF external connection point it is allowed to choose an IP address on an existing network.
- IP profiles: describe the IP parameters for the external virtual links (IP version, subnet, default gateway, DNS and DHCP specifications, security groups).

Ulterior advanced functionalities not employed in the present work, such as VNF forwarding graphs and monitoring of performance indicators, may be referenced in the NS descriptor.

4.2.4 Configuration options

OSM provides three different solutions for configuration of the deployed VNFs, classified according to the time of execution and the available operations. The different sets of operations are known as day-0, day-1, day-2 procedures [14].

Day-0 configuration

Day-0 configuration procedures regard the basic instantiation stages, and their aim is to make the VNFs operational and manageable. These procedures are executed during bootstrap of the instances. The deployed VNFs are configured according to the parameters specified in the YAML descriptor, in particular the management interface is set up to enable successive communications toward the MANO entity.
OSM permits an ulterior level of initial configuration, by injection of scripts inside the VDUs at bootstrap through an external software, *cloud-init*[5]. This procedure applies in case a VDU is deployed starting from a generic disk image (as example, cloud images - disk images of an operating system providing only basic functionality and ready for deployment in a cloud infrastructure): in this eventuality, the image contains only the minimal configuration to guarantee operability and no hardcoded parameters that are relevant to the

[5]https://cloudinit.readthedocs.io/

service (e.g. username and password for access to the machine) are present. Cloud-init scripts are defined at VDU level and may be included in the corresponding section in the descriptor; the scripts are executed at bootstrap and inject the specified configuration. Some options made available by this procedure are the addition of users in the system, injection of keys to enable SSH access, configuration of network devices, execution of arbitrary commands.
In case the used disk image is already customized (e.g. being a snapshot of an already existing instance), this procedure needs not be applied.

Day-1 configuration

Day-1 procedures regard service initialization for VNFs; they are automatically executed right after instantiation, provided that the user specified the operations to be performed referencing them in the descriptor.
To perform day-1 operations, OSM makes use of the Juju plugin, which enables the execution of lifecycle configuration routines through the use of proxy charms. A proxy charm is a collection of YAML descriptors, support files and executable code, written either in Python or Bash language; its main components and structure are shown in the following schema.

```
example_charm/
├── actions/
│   ├── action_A
│   ├── action_B
│   └── ...
├── reactive/
│   └── example_charm.py
├── actions.yaml
├── layer.yaml
└── metadata.yaml
```

The `metadata.yaml` file contains descriptive information about the identity of the charm, the maintainer, and the provided functionality.
The `layer.yaml` file can be used to leverage other existing charms including them as base layers to build upon, and may specify other dependencies to use in the executable code.
The `actions.yaml` file contains an high-level list of the actions implemented by the charm and executable on the VNFs, with an optional description. For every defined action it is possible to enumerate a set of arguments accepted in input, together with their types (string, integer, boolean, object) and default assumed values.

The `actions` folder contains a file for each defined action, with its same name. Every file includes boilerplate code whose only aim is to make the actions executable.
The `reactive` folder contains a Python script which is the actual implementation of all the declared actions. The script is built according to the reactive programming paradigm, that makes it possible to monitor data streams and perform actions reacting upon particular events or calls; this paradigm provides great potential for automation processes.

After the charm is composed, it is included in a specific VNF package to be on-boarded, and referenced in the related descriptor. The charm, however, is not part of the VNF itself, but resides in a container inside the OSM installation managed by a Juju controller, and enabled to communicate with the associated VNF. In day-1 configuration its execution is automatically launched after the VNF bootstrap is completed; the parameters to be used for the actions are specified directly in the descriptor. At the end of day-1 configuration, the VNF starts providing the expected service.

Day-2 configuration

Day-2 procedures allow reconfiguration of the VNFs so their behaviour can be modified during runtime, as well as the possibility to monitor the key performance indicators (KPI) and run scaling operations. All these procedures are executed on-demand, being it a request from the user or a triggering state.
To enable day-2 operations it is mandatory to create a proxy charm implementing the actions considered of interest for the VNF to be deployed, and to include it into the package as for the day-1 configuration procedures. After the VNF is instantiated and fully configured, the user may request at any time the execution of a day-2 operation stated in the charm, via the OSM user interface, specifying all the necessary input parameters. This kind of reconfiguration procedures offer large potential in interaction with the deployed VNF, although the impossibility to change the available actions once the VNF is created (since the charm cannot be modified runtime, then all the operations to be enabled must be designed before instantiation) is a current limitation for this approach.

Chapter 5

Implementation of the system

The present chapter outlines the software solutions devised to integrate Athonet commercial products with the other tools, with the purpose of implementing the NFV and MEC joint environment. The first part describes the setup of the physical and virtualization infrastructure. The second part is dedicated to the realization of the system outlined in section 3.1, examining the network automation procedures in the NFV scenario. The third part is dedicated to the realization of the system outlined in section 3.2, including MEC services in the NFV environment and considering the traffic steering activation.

5.1 Environment setup

To support the deployment of the NFV systems, the experimental environment should be composed by two separate virtualization infrastructures, one management entity, and a base station. To provide the virtualization infrastructures we made use of two physical hosts, each of them provisioning an OpenStack installation to manage the available hardware resources as a VIM and instantiate the VNFs on top. The characteristics of the hosts are reported in the following table.

	OpenStack Host 1 (Core site)	**OpenStack Host 2** (Edge site)
CPU	Intel Core i7-2600, 3.40 GHz	Intel Core i5-7200U, 2.50 GHz
RAM	16 GB DDR3	16 GB DDR4
Storage	SSD, 480 GB	SSD, 250 GB
OS	Ubuntu 18.04 LTS	Ubuntu 18.04 LTS

Each OpenStack installation provides connectivity on the physical network to the deployed VMs through its internal gateway. All instances requiring external connectivity attach a port on the `private` internal OpenStack

network, created beforehand; the gateway performs the association of internal ports with floating IP addresses on the `public` network, a subnet of the physical network, as shown in figure 5.1. This allows the VNFs to communicate with other machines or the Internet.

For the first implemented system only the cloud infrastructure located at the core site has been employed; for the second system both infrastructures have been employed.

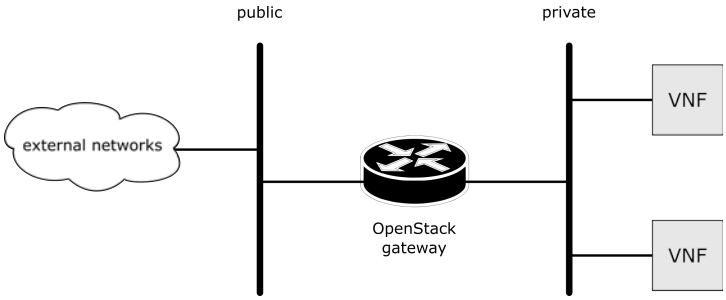


Figure 5.1: OpenStack gateway performing connection of virtual machines to external networks.

Open Source MANO is deployed on a laptop featuring a Ubuntu 18.04 LTS operative system. The management communications between the OSM host and the two cloud infrastructure hosts are performed within a secured private network.

To provide LTE access to the user equipments, a portable base station has been included in the experimental environment. Connectivity between the base station and the edge site is ensured through a private network and required for data exchanges with the EPC components deployed in both cloud infrastructures. It is required to configure the MME and SGW components to establish a link toward the eNB, providing information on its network location.

5.2 Automation of deployment and configuration procedures

5.2.1 SOL002 configuration API

Configuration requests for VNF instances are performed over the standardized Ve-Vnfm interface, exposed by MANO toward the Element Managers. ETSI group specification SOL002 [15] outlines the set of RESTful APIs and data models supported over the Ve-Vnfm logical interface, which by extension is referred also as SOL002 interface.

Application Programming Interfaces (API) are protocols used for inter-

action between different entities, providing a shared set of procedures and a common language for the access to resources and services. The use of APIs is meant to facilitate the information exchange between heterogeneous services and applications, allowing them to support the same access modality, adherent to agreed specifications. RESTful APIs are the most used type of web APIs, based on REpresentational State Transfer (REST), an architectural style and approach to communications often used in web services development, typically over HTTP. A RESTful service satisfies a number of structural conditions, among which:

- Use of a uniform interface. Resources should be uniquely identified with a single URL (Uniform Resource Locator, stating the address where the web resource is placed), and only by using methods responding to the CRUD functions (Create, Retrieve, Update, Delete) should it be possible to manipulate a resource. For HTTP protocol, the methods GET, POST, PUT, PATCH, DELETE are used.

- Client-Server relations. There should be a clear distinction between the client and the server, the first performing requests on the resources and the latter providing access to the hosted resources. This separation of roles between the client and server enables each to be developed and enhanced independent of the other.

- Stateless operations. No information about sessions and client context is retained on the server between subsequent requests, and every request is independent respect to the previous occurred. Each request from any client contains then all the information necessary to provide the desired service, and the session state is held at the client side only.

The SOL002 interface supports different administration procedures on the VNFs, such as performance management, fault management, and network configuration, each accessible via HTTP protocol through a corresponding URL. All the URL prefixes of the API are of the form:

```
{apiRoot}/{apiName}/{apiVersion}/
```

where `apiRoot` indicates the protocol ("http" or "https"), the hostname (IP address) of the machine where the resource is located, and an optional port, `apiName` indicates the interface name for the requested service, `apiVersion` indicates the current version of the API. For the configuration section of the SOL002 interface `apiName` is set to "vnfconfig", `apiVersion` is currently version 1, and the only resource made available by the VNF to the external management entities is the `configuration` resource.

Thus, it is possible to access the configuration of the VNF via the URL:

`{apiRoot}/vnfconfig/v1/configuration`

with `apiRoot` containing the IP address of the Element Manager which is in charge of applying the settings to the VNF. The allowed HTTP methods on the considered resource are the GET method, to read the configuration data of a VNF instance and its VNFC instances, and the PATCH method, to set (add or modify) the configuration data. The data model included in the specification states the data structures accepted by the interface, and consequently the configuration fields it is possible to edit via a SOL002 HTTP request. Our interest is in the network configuration of the various VNF components (namely, the EPC components), to associate each VNFC connection point to a physical interface of the VM and to a LTE interface identifier, along with inclusion of network routes, IP addresses, and other parameters specific to the EPC deployment; however, part of these configuration data is not supported by the NFV standard, which has no knowledge of physical interfaces and specific LTE settings. To solve the limitation, we exploited the optional field `vnfcSpecificData` made available by the API, whose scope is to allow vendors to perform specific configuration that cannot be done through the other SOL002 parameters: it is a generic field which supports data in key/value pairs, such as provided by YAML or JSON files, for additional settings.

The content of the SOL002 request is a JSON file with the following structure:

```
{ "vnfcConfigurationData": [
  { "vnfcInstanceId": "<vnfc_id>",
    "CpConfig": [
    { "cpId": "<interface_name>",
      "cpdId": "<phy_interface_type>",
      "addresses": [
      { "address": {"ipAddress": "<address>/<netmask>"},
        "useDynamicAddress": "<boolean>"
      }]
    }],
    "vnfcSpecificData": "<additional_conf>"
  }]
}
```

A single request may contain configuration parameters for an arbitrary number of VNF components, each identified by the `vnfcInstanceId` field. The `CpConfig` section regards the configuration of the connection points exposed by the selected component; in our implementation the meaning of some fields has been modified respect to the NFV specifications to meet the

demands of physical interfaces setup, which NFV architecture does not natively support. The field `cpId` is used to associate the interfaces of the VNF component with the correspondent connection point defined in the OSM descriptor, the field `cpdId` is used to declare the physical interface to be configured on the component, associating it to the logical connection point. The `addresses` section is used to define a set of IP addresses for the selected connection points, and choose whether they are to be statically or dynamically allocated. The optional field `vnfcSpecificData` serves to define all the ulterior parameters for component management and EPC connectivity not otherwise specifiable.

To load a new configuration, or modifying an existing one, it is possible to build a charm performing requests at the configuration URL of the SOL002 interface, as part of day-1 or day-2 procedures. The data, sent by OSM via the Ve-Vnfm connection, is received and handled by the Element Manager, which applies the submitted configuration to the chosen VNF components.

5.2.2 Element Manager

The communications between the Element Manager and the network elements are performed over proprietary interfaces providing their own APIs. The Element Manager receives from OSM configuration requests compliant with the SOL002 API, and is in charge of translating them into requests consistent with the proprietary API, by which the configuration can effectively be applied to the network elements.

To access the configuration interface of any of the VNF components, the EM must have knowledge of the endpoint to which the component binds: this correspond to the IP address of the management interface exposed by the machine and a specific port. This information must be included in the body of the SOL002 request together with the other parameters.

A single configuration request to the EM is validated according to SOL002 specifications and then split in a sequence of instructions sent to the various network elements. The first configuration segment to be processed is related to connection points: the EM parses the `CpConfig` field and for each element associates the physical interface of the component with the logical identifier specified in the OSM descriptor and an IP address, if not already set in the instantiation phase. Lastly, it parses the information contained in the `vnfcSpecificData` section; here, it is possible to retrieve the endpoints at which the different components can be reached and configured, together with information about routing, binding of physical interfaces to LTE interfaces, and network location of other EPC components.

The EM retrieves the current network configuration of the machines and updates it according to the received data, through HTTP requests adhering to the proprietary API.

An additional configuration modality has been implemented for the network elements, directly accessible to the user over a control interface exposed by the EM on the external network. The Element Manager provides a command line user interface (CLI) for direct interaction with the EPC components network settings, without the necessity to perform requests through OSM and the SOL002 interface. This alternate mode can be efficiently used for day-2 procedures. Figure 5.2 graphically represents the diverse configuration mechanisms provided to the user via the Element Manager.

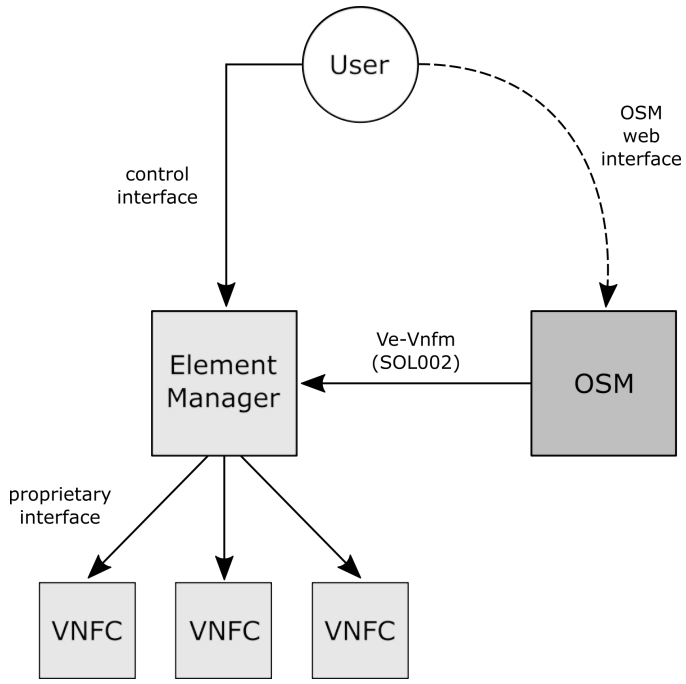
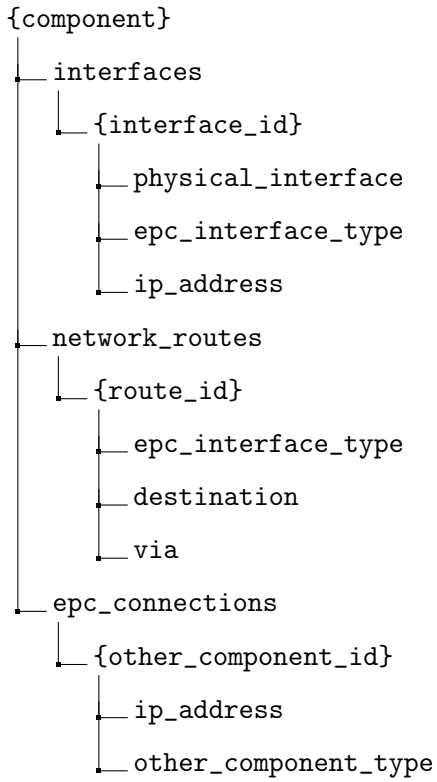


Figure 5.2:
Configuration options offered by the Element Manager. The user may demand OSM to perform requests over the Ve-Vnfm interface, or directly access the EM configuration environment.

The EM CLI supports a basic authentication scheme (user/password): the user may access the configuration of VNF components upon insertion of valid credentials. EM makes available three configuration sections for each network element, supporting all necessary network settings for proper connectivity setup and covering all options provided by SOL002 requests, though in a simpler and interactive way. The diagram representing all accessible configuration fields is reported hereinafter:



The EM records the endpoints and stores all configuration data about the existing components as part of day-1 procedures, after application of SOL002 setup information, in order to be able to access the elements at a later time. The user, as part of the day-2 operations, can select the `component` of interest and enter in its context, where all the settings can be retrieved or modified navigating a tree structure.

- The `interfaces` section is related to all settings about the physical interfaces of the network element. It is possible to select an interface descriptor specifying its `interface_id` if already present. Through this section it is possible to associate a VM physical interface with the logical LTE interface it implements and an IP address. Each logical interface can be associated to only one physical interface.
- The `network_routes` section is related to routing information for the component. It is possible to create or modify routes specifying a `route_id` value. Each LTE interface may have its own routing table, used to specify the path to reach the other endpoint(s) of the LTE connection; editing the `destination` and `via` fields allows to set the gateway to be used to reach the destination host.

- The `epc_connections` section is used to specify connectivity information for LTE interfaces. It is possible to access an existing item or create a new one by specifying an `other_component_id` value. Each item informs on how to reach a peer component over a LTE interfaces the network element exposes. Values for `other_component_type` are to be chosen among the set `{mme, sgw, pgw, hss, enb}` and the EM accepts only a subset of these, according to which LTE interfaces are provided by the considered component (e.g. a MME component must expose the S11 interface to accept a SGW peer). The field `ip_address` specifies the IP address at which the selected peer component can be found; routes to reach the various IP addresses can be indicated in the `network_routes` section. The peer entities may be other internal components of the same VNF or reside in an external network location.

The sections can be edited navigating the configuration tree and modifying the fields with fine granularity, or by loading a compliant JSON file specifying the entire new configuration. After each modification the EM validates the inserted data, and in case of incorrect insertion it returns an error message. After the new configuration is validated, the EM pushes the modifications into the components performing requests over the proprietary interface.

5.2.3 Instantiation and lifecycle configuration

It is feasible to automate the deployment and full configuration of a LTE core network integrated in the NFV environment, by leveraging the interplay of OSM descriptors, Juju charms, and the Element Manager. The procedures to be outlined in the following are devised for instantiation of a full EPC in a single site, but can be used to implement any other LTE over NFV scenario, with appropriate modifications.

OSM packages

The VNF package includes the software images of the network elements (MME, HSS, SGW, PGW), each running on a proprietary Linux distribution, and of the Element Manager, running on a standard CentOS 7 distribution equipped with the software implementing EM functionalities. In addition, a juju charm to perform SOL002 requests is included, and the VNF descriptor outlining the structure of the VNF, its internal components and connections.

The NS package includes the NS descriptor, referencing the contained VNFs and characterizing attachment of VNF connection points to the external network.

Juju charm

The Juju charm is designed to send the configuration file to the Element Manager as part of day-1 procedures; the EM handles all the successive EPC configuration operations. It is possible to extend the charm to provide day-2 functionalities; however, for runtime reconfiguration the CLI is preferred for its ease of use.

The charm implements the primitive `initial-patch-configuration`, which accepts in input the IP address to reach the Element Manager and a VNF configuration file compliant with SOL002 specification. After the NS instantiation the charm performs a request over the Ve-Vnfm interface to submit the specified configuration. It is necessary to include the charm and declare its primitives and input parameters in the correspondent section of the VNF descriptor, structured as the given example:

```
vnf-configuration:
  initial-config-primitive:
  - name: initial-patch-configuration
    parameter:
    - name: em-hostname
      value: <EM_IP>
    - name: vnfc-configuration-data
      value: <json_conf_file>
  juju:
    charm: athonet-sol002-client
```

The values enclosed by angle brackets, stating the EM address and the SOL002 configuration, are placeholders and their value is defined by the user inside the NS instantiation request.

NS deployment

When performing a NS instantiation request, it is necessary to provide OSM with information about the NS descriptor to employ, the NS name, and the VIM to be used for deployment. Additional information can be supplied by including a YAML configuration file in the request. In particular, it is possible to define the set of parameters which will be passed to Juju during instantiation and are left unspecified in the VNF descriptor, referring them in the `additionalParamsForVnf` section of the file:

```
additionalParamsForVnf:
- member-vnf-index: 1
  additionalParams:
    EM_IP: '<element_manager_ip>'
    json_conf_file: '<sol002_configuration>'
```

The instantiation procedure can be launched from OSM client executing a single command including all the necessary information:

```
osm ns-create --ns_name athonet_full_epc --nsd_name full_epc_nsd \
--vim_account openstack-01 --config_file epc_configuration.yaml
```

Following the request, OSM retrieves the instantiation parameters from the additional configuration file and provides them for the Juju script execution. The required VNFs are deployed at OpenStack site and the internal and external connectivity is set up, following the information contained in the descriptors. The charm performs a configuration request over the Ve-Vnfm interface toward the Element Manager, which subsequently validates and applies the network settings to the VNF components.

The virtual EPC, deployed and correctly configured, is at this stage ready to provide the expected LTE connectivity service. We verified the system functionality performing an attachment procedure to the network using a cellular phone provided with an authorized SIM card: through the Wireshark network analyzer we monitored the activity over the LTE interfaces and registered the signalling relative to the bearer setup procedure, which was successful. The device was enabled to exchange traffic data through the deployed mobile network, the actual EPC implementation (physical or virtual) being transparent respect to the offered services.

With the design of the OSM descriptors and network configuration as the only elements requiring user expertise and intervention, once they are defined the NS instantiation qualifies itself as a one-click procedure, therefore easily scriptable. After deployment and day-1 configuration the NS is ready to provide 4G connectivity to the user devices. Network reconfiguration is made availble as a day-2 operation, taking advantage of the Element Manager CLI interface.

From inspection of the event logs of the OSM LCM module, we are able to detect the notification relative to deployment procedure start and the notification relative to deployment and configuration completion, with respective timestamps. Exploiting the integration work carried out so far, it is possible to implement a sequence of automatic deployment and deletion cycles for the NS, to verify the robustness of the realized system and quantify the average amount of time the NS takes to be instantiated and become fully operative. We designed a script which analyzes the OSM logs and cyclically performs the following operations:

- when the previous NS is fully destroyed (or there is none, at the beginning), a new NS is launched, saving relative timestamp;
- when the NS is fully deployed and configured, it is destroyed, saving relative timestamp.

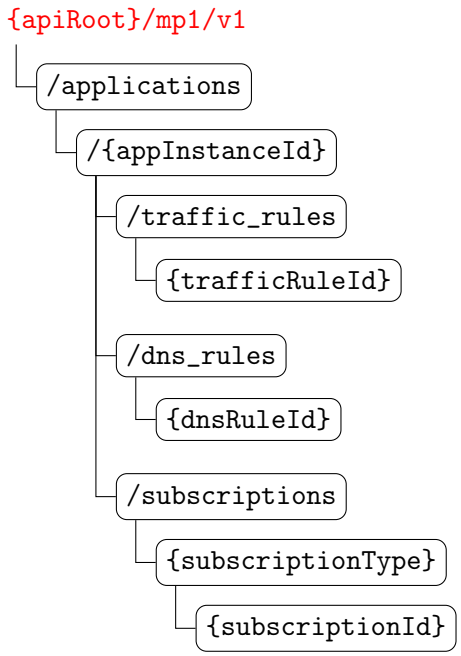
Running the process for 100 cycles we noticed that the automation procedure is robust: the process did not get stuck and at every cycle the EPC installation was complete and functional. The average time required for deployment/configuration procedures was 258.95 seconds; this, however, is highly dependent on the infrastructure hardware, and on more performing cloud infrastructures it is expected to be significantly lower.

5.3 Activation and fruition of MEC services

5.3.1 Mp1 configuration API

The Mp1 API specification [16] scope is to describe the reference point between mobile edge applications and mobile edge platform, specifying protocols and data models for management communications over the considered interface. The Mp1 interface supports configuration operations for MEC applications registered at MEC platform, such as information exchange about service availability, traffic rules and DNS rules setting, subscription to notifications, and MEC applications lifecycle management.

The URL prefixes of the API are of the form `{apiRoot}/{apiName}/{apiVersion}/`, with the parameters assuming the same meaning of the corresponding SOL002 prefix. The value `apiName` is set to `mp1` for the considered interface. Below is a tree representation of the main relevant resources made available by the considered API:



Through the provided resources it is possible to access the configuration of an application by specifying the respective `appInstanceId`. In the context of an application the accessible resources regard the associated traffic and DNS rules and the active subscriptions for notifications from the MEC platform.

A limitation of the standard Mp1 API is the impossibility to register new applications or load new traffic rules at the MEC platform: the allowed methods enable only the modification of resources pre-provisioned in the machine. To overcome the restriction, we performed an integration of the API, including additional resources supporting the POST method that enable the registration of applications and traffic rules. Using the extended API, OSM may be exploited to perform Mp1 requests toward the MEC platform, provisioning information about applications and traffic rules, playing a management role also for the MEC system.

The VNF containing the SGW-LBO, implementing the MEC platform, is complemented by an Element Manager whose scope is to accept and validate the Mp1 requests and forward the content according to the proprietary API provided by Athonet SGW-LBO for its MEC configuration. In addition, it provides the same functionality regarding the Ve-Vnfm reference point, as outlined in the previous sections.

5.3.2 MEC system setup

The multi-site scenario considers the deployment of a distributed EPC, with the SGW component residing at the edge node and the remaining components residing at the core node. The base station is able to reach exclusively the edge host through a separate network, to provide LTE connectivity: to simulate a MEC scenario all EPC communications are to be conveyed by the edge site toward the eNB. The overall system is shown in figure 5.3.

The two sets of EPC components are separately instantiated and configured by OSM, following the procedure described in the previous sections. Each set is embedded in a network service including the EPC components to be deployed in each OpenStack environment, together with the relative application VM. Network configuration for both EPC instances is performed over the SOL002 interface: of particular importance is the correct setting of LTE interfaces and IP routes, allowing the components located at one site to reach the components at the other site.

To simulate a physical separation between the core site and edge site, despite being them located in a restricted simulation environment, we made use of the `tc` (traffic control) Linux utility. `tc` is used to configure the Linux kernel packet scheduler, a node that manages the sequence of network packets in the transmit and receive queues of the network interface controller (NIC).

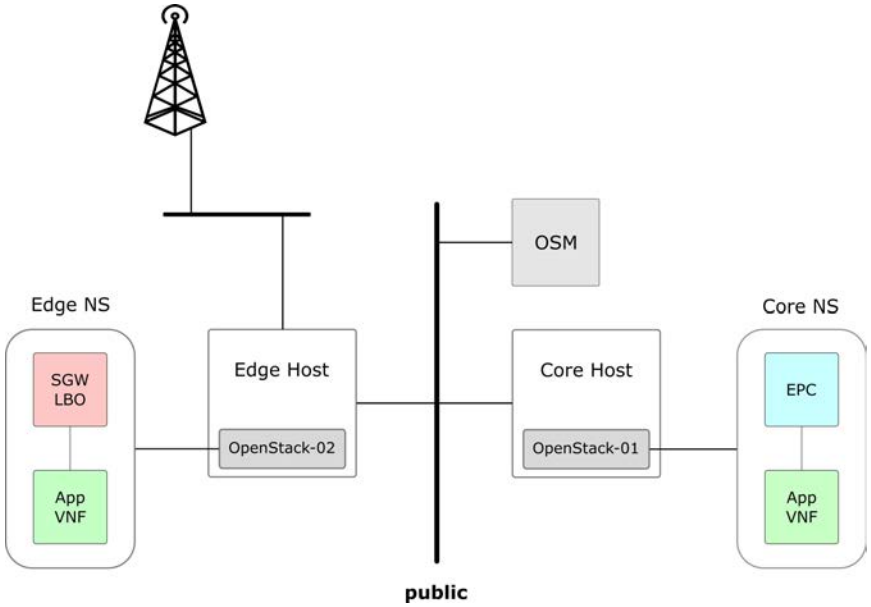


Figure 5.3: Multi-site deployment network architecture.

The scheduler logic decides which packet to forward next, among those temporarily stored in the associated queuing system waiting to be transmitted. Many scheduling algorithms (also called queuing disciplines, or qdisc) exist to implement the scheduler logic, each providing specific reordering, forwarding, or dropping policies for network packets stored in the buffer, leading to different performances. As the default queuing discipline, Linux kernel uses a FIFO (first in, first out) policy, which however can be substituted with another logic using the `tc` tool.

To limit the bandwidth of the link between core and edge site, it is possible to rely on the token bucket filter (tbf) logic: it consists in generating tokens at a desired rate, and only dequeue packets from the buffer if a correspondent number of tokens is available; tokens can be generated up to a number defined by the capacity of a bucket: if ulterior tokens are generated and not used, they are discarded. The tbf qdisc is an example of traffic shaping and rate limiting algorithm, used to reduce the speed at which packets are dequeued from a particular interface, by setting a proper token generation rate. To set a bandwidth limitation on the output flow of a specific interface, it is necessary to change the queuing policy issuing a command with the following structure:

```
tc qdisc add dev <interface_name> tbf rate <desired_rate>
```

An additional control over the interfaces behaviour is provided by `netem` facility: it is used to emulate the properties of wide area networks, such as

delay, packet loss, packet duplication and re-ordering. Appending a netem rule to a specific interface allows in particular the addition of a variable delay to output packets, determined according to a chosen statistical distribution. It is also allowed to introduce correlation between the delay experienced by two consecutive packet dispatches, better simulating a realistic network behaviour. To set an output delay following a normal distribution, given its mean, jitter and delay correlation, a command with the following structure is issued:

```
tc qdisc add dev <interface_name> netem delay <mean> <jitter> \
distribution normal <correlation_percentage>
```

Since the queuing discipline is applied to the packets in output only, to keep the link symmetric it is necessary to apply the same rules at both sites on the relevant interfaces, setting the same limiting rate and half of the overall desired latency.

The link degradation simulated with the `tc` utility is the triggering condition for the traffic switch activation. It is meant to set the content retrieval from the remote server as the main performance limiter for the system, as perceived by the user.

5.3.3 Traffic switch

The initial state for the deployment consists in the core (full) EPC activated and providing users with connectivity toward Internet and the Application VNF, deployed as a virtual machine in an OpenStack internal network reachable by the PGW. The EPC is able to communicate with the eNB located at the edge site, representing the access network. When OSM is notified by the Application about a service relocation requirement, because of insufficient QoS, it responds starting the instantiation procedures for the NS comprising SGW-LBO and Application at the edge. When the VNFs are instantiated, the network configuration is initiated: via the SOL002 interface the Core EPC is notified about SGW relocation (selecting the edge SGW instead of the internal one in the S5/S8 and S11 settings), and the Edge SGW is notified about the presence of S5/S8 and S11 peers at the core site. The EPC data and control plane communications in the two subsequent stages are portrayed in figure 5.4.

Successively, OSM registers the new application and associated traffic rules at the SGW-LBO via the extended Mp1 interface. To load a new traffic rule at the MEC platform, a HTTP POST request is performed, including the rule to be applied as its body.

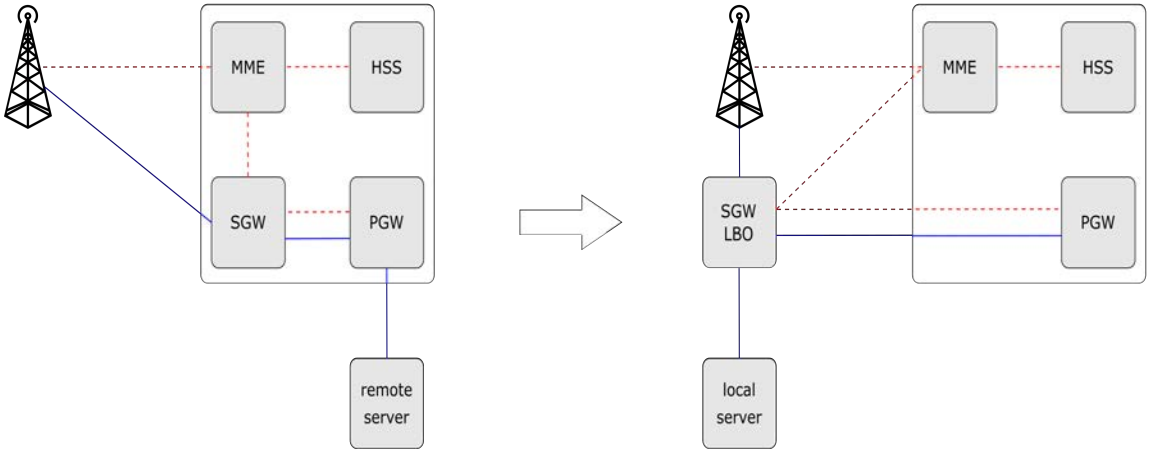


Figure 5.4: Reconfiguration of EPC traffic flows following the SGW relocation and traffic steering procedure. Red dashed lines represent the control plane, blue solid lines represent the user plane.

The traffic rule is a JSON object with the following structure:

```
{ "trafficRuleId": "<id>",
  "filterType": "<type>",
  "priority": "<int>",
  "trafficFilter": "<filter>",
  "action": "<action>",
  "state": "<state>"
}
```

Each rule identifies the specific packets that are to be handled differently by the MEC host through the field `trafficFilter`. It may contain various selection criteria, and for our implementation we performed a selection based exclusively on the destination IP address of the packets: if packets match the destination address of the traffic filter, they experience local breakout. It is permitted to specify a single IP address or an IP range (a network). The `action` field is used to specify the data plane action to be performed when a packet matches the `trafficFilter` parameters: as examples, packet dropping, forwarding, or redirection. The `state` field, assuming value ACTIVE or INACTIVE, can enable or disable the application of the traffic rule. This attribute may be updated using HTTP PUT method: it is then possible to pre-provision a disabled traffic rule in the platform and successively enable it on demand. The attribute `filterType` states whether the filtering is to be performed on a packet basis or flow basis.

The Element Manager receives the traffic rules on the Mp1 interface and forwards the content to the SGW-LBO, according to the proprietary configuration API. The traffic steering rule is successively activated by the

edge Application at the end of its instantiation procedures. Consequently to the MEC system setup, the traffic flow for the considered service is not forwarded toward the PGW along the S5/S8 interface but redirected on the SGi interface toward the local server (edge App).

5.3.4 Video streaming service

To provide the streaming service we chose the MPEG-DASH solution, valuable for its interoperabililty and effective response to the link quality variations. MPEG-DASH (Dynamic Adaptive Streaming over HTTP) is an adaptive bitrate streaming technique that enables provision of multimedia content over the Internet, delivered from conventional HTTP web servers [17]. The aim of DASH is to provide a seamless streaming service with the video adapting its quality according to the network conditions and compensating for the bandwidth fluctuations of the link.

A DASH stream is split in a sequence of HTTP-based data segments, each containing a portion of fixed time duration of the media, progressively downloaded by the user device. The media segments are made available at a variety of different bitrates (defining the video quality) for every time interval, to allow the DASH client to choose among the segments encoded at different rates the one best matching with the current link condition. The client uses a bitrate adaptation algorithm (ABR) to estimate the available bandwidth and selects the next segment to download as the one with the highest bitrate supported by the link, to avoid reproduction stalls and maintain a continuous playback.

The multimedia content is stored on a web server and delivered over HTTP, benefiting from the vast diffusion and support of this protocol. The content exists on the server in two parts: the Media Presentation Description (MPD), a XML manifest providing information on the available content, its alternative encoding bitrates, the URL addresses of the segments - and the data segments, which contain the actual multimedia bitstreams in the form of chunks. The DASH client at first obtains and parses the MPD file, acquiring information about the segments; subsequently it starts retrieving the content by fetching the segments as specified in the MPD using HTTP GET requests. After an appropriate initial buffering to allow for network throughput variations, the client starts playing the stream and continues retrieving the subsequent segments; at any time, based on the bandwidth measurements, the client may decide to adapt the streaming by fetching segments of different alternatives (with lower or higher bitrates) to maintain an adequate buffer.

A video input can be prepared for DASH streaming by transcoding it at different qualities and successively splitting it up over several time points, creating the segments. Tools such as `ffmpeg` and `mp4box` can be used for the purpose. The test stream used for this work has been retrieved from a web

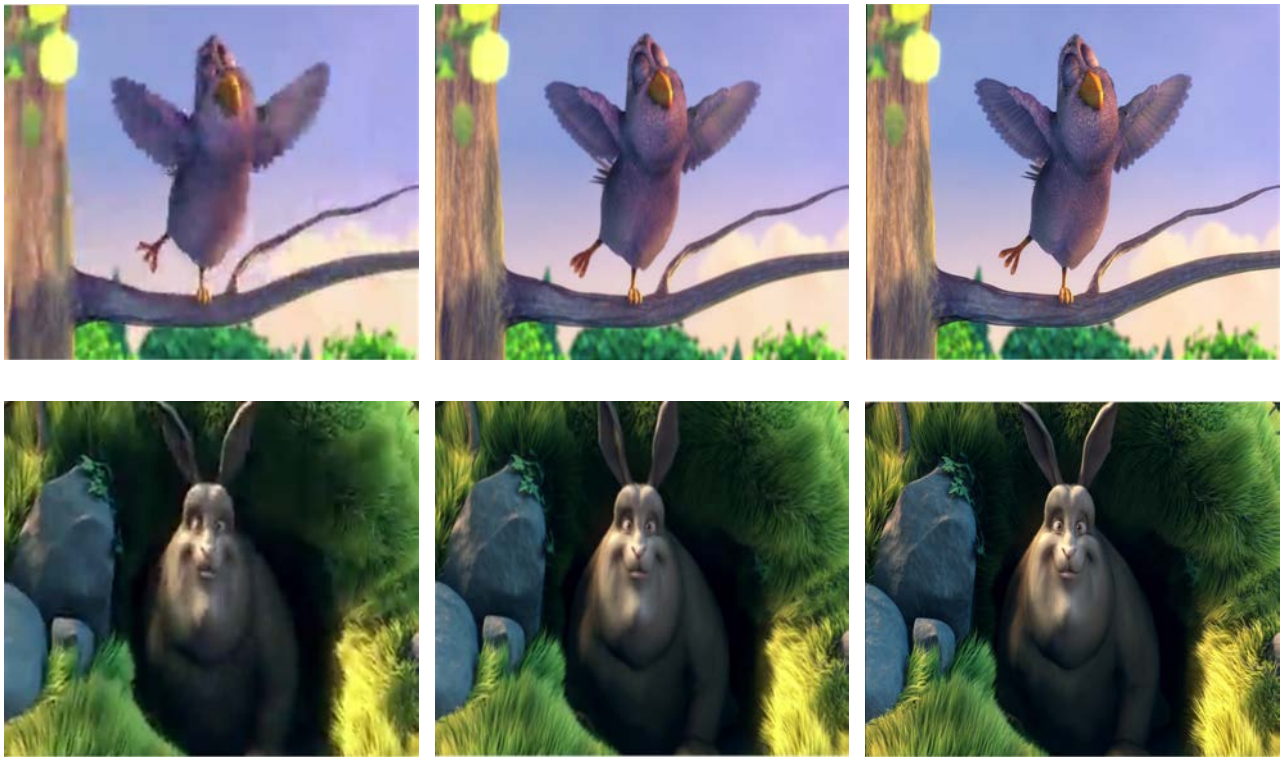


Figure 5.5: Video frames pertaining to segments encoded at 45 Kbps, 566 Kbps, 3.85 Mbps, respectively.

repository providing various DASH datasets ready for use[1].The considered stream features segment lengths of 6 seconds each, and segment bitrates ranging from 45 Kbps (may correspond to 144p resolution) to 3.85 Mbps (may correspond to 1080p resolution), covering various intermediate steps. Figure 5.5 presents two frames of the video encoded at different bitrate (low, medium, high resolution), showing the DASH capability to deliver segments of varying quality.

To host the video streaming service a web server has been deployed on the application VNFs; the server process is started after the VNF is instantiated and operative, and has access to the directories where the DASH files are stored. For the purpose we employed a nginx web server, lightweight, open-source, and providing an easy configuration procedure. Editing the `nginx.conf` file is required to set the service functionalities; a basic configuration necessary to activate the HTTP web server has the following structure:

[1]https://dash.itec.aau.at/dash-dataset/

```
http
    {
    server
        {
        server_name dash_server;
        listen 8080;
        location /dash
            {
            root <PATH_TO_FOLDER>
            }
        }
    }
```

The server listens for incoming connections on port 8080 at the IP address related to the SGi interface. Performing GET requests at the URL `http://<IP>:8080/dash/` gives web access to the folder located at `<PATH_TO_FOLDER>` in the machine, where the MPD file and DASH segments are stored. To enable video playback on a web browser, it is possible to embed the DASH stream into a web page. This has been obtained through an html script located in the same folder: accessing the html file via the browser retrieves automatically the MPD and starts the video reproduction.

After the setup procedures, the user is able to access the stream through the LTE network, retrieving content from the currently active application VNF. The experienced video quality allows the user a qualitative estimation of the bandwith available between the client and the server. For a quantitative evaluation, it is possible to resort to network analyzers such as Wireshark: by inspection of the exchanged packets, it has been possible to notice that the bitrate of the downloaded segments is contained in the GET request itself, due to the way DASH mechanism is configured. In fact, the requests present the syntax:

```
GET /dash/<XXX>bps/BigBuckBunny_6snonSeg.mp4 HTTP/1.1
```

with the chunk `<XXX>bps` specifying the bitrate of the segments, providing an easy to obtain yet fairly accurate estimate on the link bandwidth (up to the maximum available bitrate of the stream). This information can be used by the system to register the connection degradation and user experience decline, and request the traffic switch at the edge.

Chapter 6

Evaluation tests and results

The planned simulation campaign was originally devised to test the capabilities of the complete MEC deployment, evaluating the automatic procedure of SGW relocation and traffic steering activation in response to the worsening of service quality. However, due to the circumstances[1] the part concerning the switch automation could not be fully implemented and tested. A proof of concept of the system considering the two stages (before and after the traffic redirection from core to edge) as separate experiments is provided.

The first stage of the experiment consists of the mobile user requesting the video stream from the remote server; the data plane communications are carried out exclusively through the core EPC, as depicted in figure 5.4 a). The link between the edge site where the access eNB is located and the core network suffers an additional latency of 100 ms and a reduced bandwidth of 1 Mbps, artificially set to simulate the link limitations between edge and core network, impacting on the service. The strong bandwidth limitation has been chosen to provide a visually noticeable service deterioration that the user could easily perceive. In real case scenarios this condition is hardly met, it has been set to be representational of the remote content retrieval as the bottleneck of the communication system for simulation purposes. It may be the consequence of a network congestion or server overload, in which cases a traffic redirection at the edge can prove beneficial.

In the second stage the data traffic has been redirected through the SGW-LBO located at the edge to retrieve content from the local server, by manually triggering the application cloning and inserting the traffic steering rules in the MEC platform. The data plane traffic follows the path shown in figure 5.4 b). The communications are not limited and can reach a maximum nominal bandwidth of 150 Mbps.

In both stages the device attempts to access content contacting the same IP address, of the remote server: it is the duty of the MEC system to apply

[1] See NOTE in the Introduction.

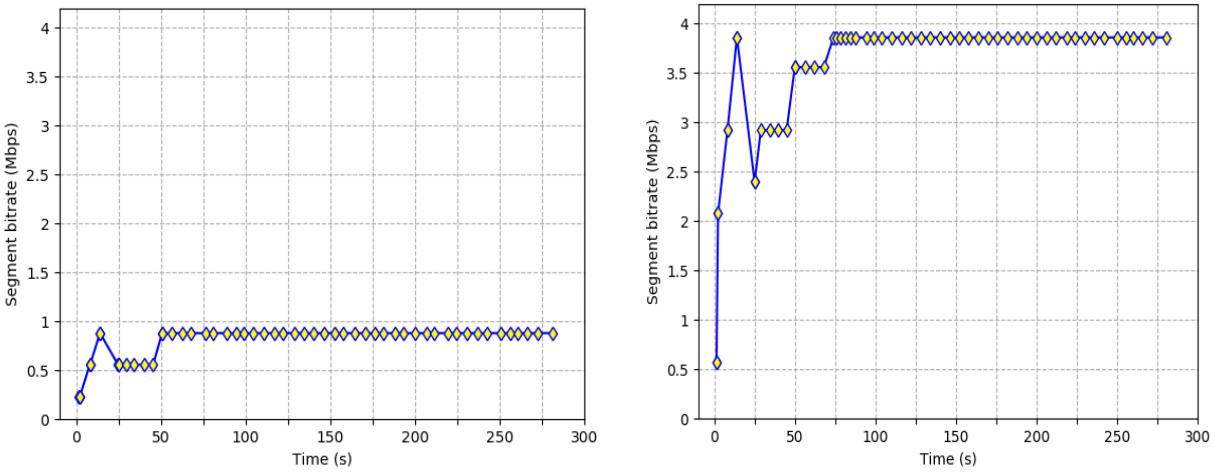


Figure 6.1: Encoding bitrate of the DASH segments requested to the remote server (left panel) and the local server (right panel), versus time. Every request is represented by a yellow marker.

or not the traffic rules and redirect the data flow toward the different servers. This allows the user not to perceive the server change or being compelled to perform an additional web search to continue the video playback.

The metric used to measure the performances of the system has been the encoding bitrate of the DASH segments retrieved by the user device when reproducing the stream; it is representative of the available bandwidth of the connection between the client and the server, up to a certain value, and at the same time it provides a clear insight about the quality of the content delivered to the user by the system. Figure 6.1 reports two graphs showing the bitrate of the segments fetched respectively from the remote server and the local server, over a time span of 280 seconds. The DASH segments are requested approximately every 6 seconds, with the interval variations likely caused by the variable connection delay and the filling level of the playback buffer at client side.

It can be noticed a transition period at the beginning, with the DASH stream behaving conservatively and providing segments of lower quality in the starting phase. The DASH algorithm attempts to counteract to bandwidth fluctuations increasing progressively the video quality when the link proves itself stable and able to support retrieval of segments encoded at the current bitrate over a certain time span. After the transitory, the service reaches the highest quality supported by the employed connection. It is shown how the bitrate in the non-breakout scenario caps at around 0.9 Mbps due to the link restrictions, while in the breakout scenario the link

capacity is greater and the bitrate caps at 3.85 Mbps, the highest available DASH bitrate for the considered stream.

The full system implementation, including the automatic traffic redirection, would provide an expected behaviour close to that outlined in figure 6.2. In the first stage, the user device retrieves content from the remote server: after a transition period the stream will likely reach the best quality allowed by the worsened link. After the Application demands the traffic switch and the MEC service is available at the edge, a new bearer setup procedure must be performed, to account for SGW relocation and allow the user device to connect to the new SGW-LBO. This requires a narrow amount of time, during which the user is detached from the network and the throughput drops to zero. After reattachment the user device is connected to the local server, and as the transition period is over the stream will likely reach its best quality, which the new scenario permits.

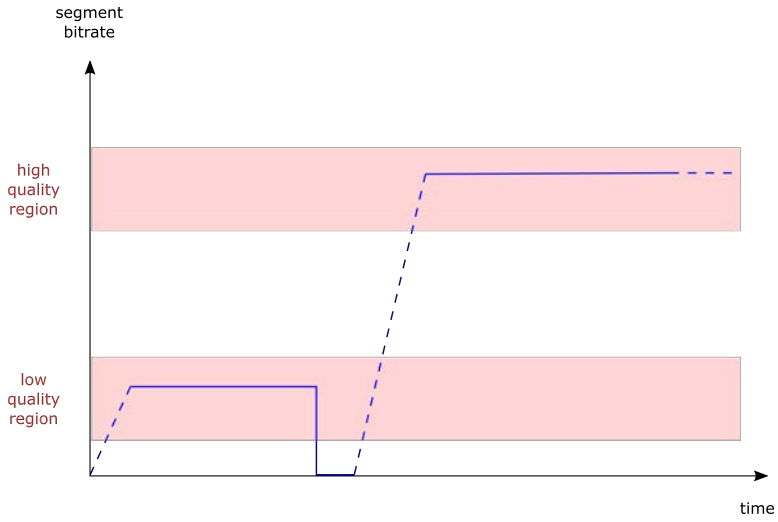


Figure 6.2: Expected behaviour of the streaming service in a scenario including automatic traffic switch.

For each stage screenshots of the stream reproduction have been captured on the device, to appreciate the difference in the quality of the content collected through the different routes. Comparison of two frames taken from the diverse streams is reported in figure 6.3. On the left, images of the remote streaming service, on the right, images of the local streaming service; though at 0.9 Mbps the video quality is fairly good, a greater amount of details can be appreciated in the content served at 3.85 Mbps, uppermost capacity. It is proven that the user may benefit from the traffic redirection at the edge if the connectivity toward the core site and remote server undergoes a drop in quality.

Figure 6.3: Comparison of two frames taken from the DASH stream at different qualities. The frames on the left belong to the stream from the remote server, the frames on the right to the stream from the local server. The remote stream reaches a top bitrate of 0.88 Mbps, whereas the local stream a top bitrate of 3.85 Mbps.

Chapter 7

Conclusions

This thesis presents the implementation of a system providing integration of ETSI NFV and MEC technologies into a 4G mobile network, inquiring the procedures for correct deployment and configuration of a virtualized network and subsequently the inclusion and delivery of edge services.

The unified NFV and MEC system is realized through the integration of both proprietary and open-source software, elaborating on the Ve-Vnfm and Mp1 standardized interfaces to harmonize the various components and enable the configuration of the LTE network embedded in the system. In particular, an Element Manager has been developed to interact with the network components and apply the settings communicated through the NFV and MEC standard interfaces.

The first target has been the automation of the deployment and configuration procedures for the mobile network in the NFV framework. Upon provision of previously prepared descriptors and configuration files, the procedures can be made fully automatic, requiring only the user to issue the instantiation command and wait for the network to be deployed. We demonstrated the feasibility of the process and verified the correct functionality of the LTE network in enabling user devices to connect to the Internet. We can conclude that the NFV paradigm provides a great flexibility in mobile network management, allowing to instantiate and scale LTE networks removing many of the limitations caused by a strong dependency on the hardware. Moreover, it can be stated that from the point of view of the user it is irrelevant whether the LTE installation is on a physical or virtualized infrastructure, not affecting the service delivery.

The second objective has been the inclusion in the system of a platform for provision of MEC services and the redirection of the user plane traffic from a remote server, to be reached through the LTE core network, to a local server, providing the same content, to be reached through the MEC platform. To test the functionality of the MEC implementation and evaluate the advantages of accessing the content at the edge, we set up a video

streaming service with quality adaptation according to the connection bandwidth. We also devised a procedure to automate the traffic switch procedure. A proof of concept of the system is produced, considering as performance metric the quality of the stream retrieved by the user, in which we set the connection toward the remote server as the bottleneck of the connectivity scenario. From the experiment we observed the service improvement when exploiting the MEC capabilities and performing a local breakout toward the local server. The conclusion is that edge computing can prove beneficial in increasing the user experience or solving possible congestion issues, and by automatizing the traffic switch procedure it would be possible to realize a network architecture adapting itself in an attempt to improve its performances. This network reconfiguration potential is assisted by NFV systems, when used in interplay with LTE and MEC technologies.

Future work

In the presented work, the main interest has been put upon metrics relative to the bandwidth of the system; the other metric of interest, the communication delay, has been just mentioned. As a complementary study it can be of interest the analysis of the impact on the communications of an high-latency link between the edge site and the remote server, together with the traffic redirection as a solution to mitigate the problem. Moreover, more complex connectivity scenarios (e.g., introducing network congestion or server overloads) could be implemented, to provide a better simulation of real network conditions.

Another limitation of the approach followed in the system implementation regards the absence of a central MEC management site, with all the orchestration tasks performed by the NFV MANO entity. For a deeper understanding of the potential offered by the integration of NFV and MEC paradigms, it should be considered a full development of both the models, featuring a close cooperation between the MEC and NFV orchestration entities.

Appendix A

Example of OSM descriptors

This section presents examples of VNF and NS descriptors used by OSM to deploy and configure network functions and services. The considered descriptors are compliant with the ETSI information model and provide a sample configuration sufficient to adequately define the structure, connections, functionality, of a simple NS and its composing VNFs.

A.1 VNF descriptor

The given VNFD defines the VNF structure shown in figure A.1. This VNF is made up of two internal components, outlined in the `vdu` section, connected by an internal virtual link, `simple_internal_link`, a LAN subnet with IP address range `10.0.0.0/24`. The first internal component exposes the connection point `main_cp` which serves as management interface for the whole VNF, and provides a port `internal_endpoint_A` on the internal link for connection with the other component, with IP address `10.0.0.1`. The second internal component exposes the connection point `secondary_cp`, and provides a port `internal_endpoint_B` on the internal link for connection with the other component, with IP address `10.0.0.2`. The first component is deployed starting from the image `disk_image_1` and is allocated 4096 MB of RAM, 30 GB of storage, and 4 VCPUs. The second component is deployed starting from the image `disk_image_2` and is allocated 2048 MB of RAM, 5 GB of storage, and 2 VCPUs. The Juju charm `example_configuration_charm` is used for day-1 and day-2 configuration. In day-1 configuration the primitive `apply-initial-configuration` is executed and applies a pre-defined configuration file to the VNF. In day-2 configuration the primitive `runtime-reconfiguration` can be executed on demand and applies to the VNF a new configuration which must be provided by the user.

```
vnfd:
- id: example_vnf
  name: example_vnf
  description: mock VNF to illustrate descriptor options
  connection-point:
  - name: main_cp
    port-security-enabled: true
    type: VPORT
  - name: secondary_cp
    port-security-enabled: false
    type: VPORT
  mgmt-interface:
    cp: main_cp
  internal-vld:
  - id: simple_internal_link
    type: ELAN
    internal-connection-point:
    - id-ref: internal_endpoint_A
      ip-address: 10.0.0.1
    - id-ref: internal_endpoint_B
      ip-address: 10.0.0.2
    ip-profile-ref: simple_internal_link
  ip-profiles:
  - name: simple_internal_link
    ip-profile-params:
      ip-version: ipv4
      subnet-address: 10.0.0.0/24
  vdu:
  - id: internal_component_1
    name: internal_component_1
    description: example internal VNF component
    count: 1
    image: disk_image_1
    vm-flavor:
      memory-mb: 4096
      storage-gb: 30
      vcpu-count: 4
    interface:
    - external-connection-point-ref: main_cp
      type: EXTERNAL
    - internal-connection-point-ref: internal_endpoint_A
      type: INTERNAL
    internal-connection-point:
    - id: internal_endpoint_A
      port-security-enabled: false
      type: VPORT
  - id: internal_component_2
    name: internal_component_2
```

```
    description: example internal VNF component
    count: 1
    image: disk_image_2
    vm-flavor:
      memory-mb: 2048
      storage-gb: 5
      vcpu-count: 2
    interface:
    - external-connection-point-ref: secondary_cp
      type: EXTERNAL
    - internal-connection-point-ref: internal_endpoint_B
      type: INTERNAL
    internal-connection-point:
    - id: internal_endpoint_B
      port-security-enabled: false
      type: VPORT
  vnf-configuration:
    initial-config-primitive:
    - name: apply-initial-configuration
    config-primitive:
    - name: runtime-reconfiguration
      parameter:
      - name: new_configuration
        data-type: STRING
    juju:
      charm: example_configuration_charm
```

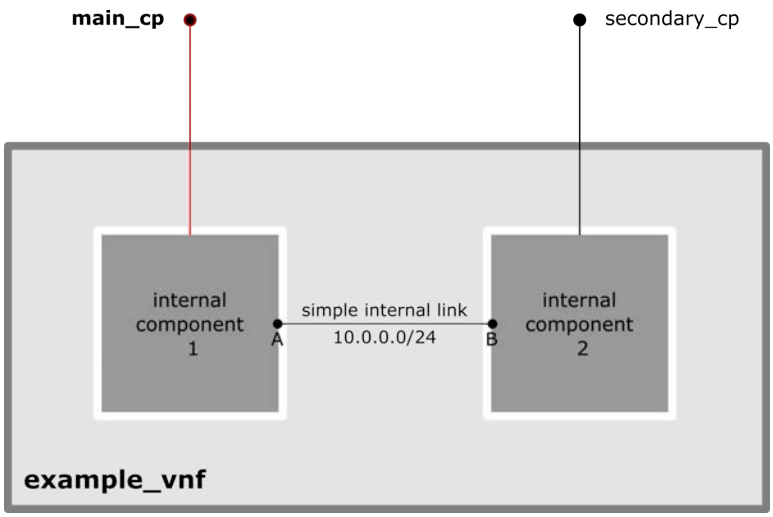


Figure A.1: VNF implemented by the proposed descriptor.

A.2 NS descriptor

The given NSD defines the NS structure shown in figure A.2. This NS is made up of two identical VNFs, both deployed starting from the previous descriptor: only the connection points names are different, to avoid misconceptions. The two VNFs are connected inside the NS through a internal network `internal_vim_net` of IP address range `10.10.10.0/24`, involving their secondary connection points for the attachment. The internal network is assumed to be a preexisting network in the VIM and needs not be further specified, only referenced in the `vld` section. Each VNF exposes its management interface toward an external network `external_vim_net` of IP address range `192.168.1.0/24`, likewise a preexisting VIM network. These connection points exposed by the NS are used to attach it to the physical network supporting the virtual deployment, giving connectivity toward the external networks and systems. The descriptor can define the VNFs composition and NS structure with a high degree of freedom, by specifying the role of the VNF connection points (for internal or external links), the networks to be created, and the static IP addresses the attachment points assume on the considered networks. The ports on the external network are given IP addresses `192.168.1.1` and `192.168.1.2`, respectively for the first and second VNF management interfaces. The ports on the internal network are given IP addresses `10.10.10.1` and `10.10.10.2`, respectively for the first and second VNF internal interfaces.

```
nsd:
- id: example_ns
  name: example_ns
  description: mock NS to illustrate descriptor options
  constituent-vnfd:
  - member-vnf-index: 1
    vnfd-id-ref: example_vnf
  - member-vnf-index: 2
    vnfd-id-ref: second_example_vnf
  connection-point:
  - name: main_cp
    floating-ip-required: true
    member-vnf-index-ref: 1
    vld-id-ref: external_vim_net
    vnfd-connection-point-ref: main_cp
    type: VPORT
  - name: secondary_cp
    floating-ip-required: false
    member-vnf-index-ref: 1
    vld-id-ref: internal_vim_net
    vnfd-connection-point-ref: secondary_cp
    type: VPORT
```

```
    - name: main_cp_vnf2
      floating-ip-required: true
      member-vnf-index-ref: 2
      vld-id-ref: external_vim_net
      vnfd-connection-point-ref: main_cp_vnf2
      type: VPORT
    - name: secondary_cp_vnf2
      floating-ip-required: false
      member-vnf-index-ref: 2
      vld-id-ref: internal_vim_net
      vnfd-connection-point-ref: secondary_cp_vnf2
      type: VPORT
  vld:
  - id: external_vim_net
    vim-network-name: external
    type: ELAN
    vnfd-connection-point-ref:
    - ip-address: 192.168.1.1
      member-vnf-index-ref: 1
      vnfd-connection-point-ref: main_cp
      vnfd-id-ref: example_vnf
    - ip-address: 192.168.1.2
      member-vnf-index-ref: 2
      vnfd-connection-point-ref: main_cp_vnf2
      vnfd-id-ref: second_example_vnf
  - id: internal_vim_net
    vim-network-name: internal
    type: ELAN
    vnfd-connection-point-ref:
    - ip-address: 10.10.10.1
      member-vnf-index-ref: 1
      vnfd-connection-point-ref: secondary_cp
      vnfd-id-ref: example_vnf
    - ip-address: 10.10.10.2
      member-vnf-index-ref: 2
      vnfd-connection-point-ref: secondary_cp_vnf2
      vnfd-id-ref: second_example_vnf
```

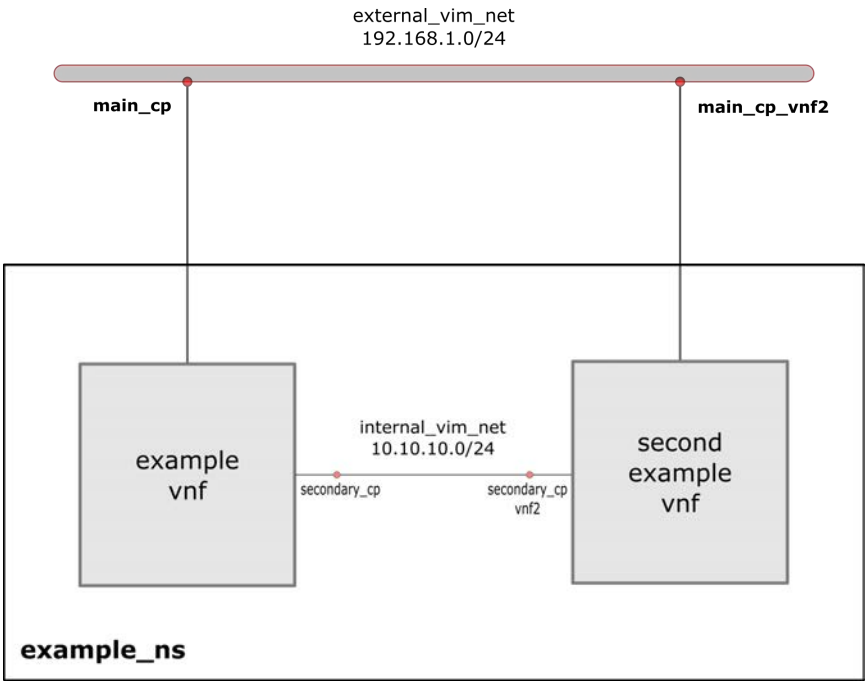


Figure A.2: NS implemented by the proposed descriptor.

Bibliography

[1] 3GPP. Technical Specification 23.401. "General Packet Radio Service (GPRS) enhancements for Evolved Universal Terrestrial Radio Access Network (E-UTRAN) access", May 2017.

[2] S. Palat, P. Godin, *Network Architecture*, in S. Sesia, I. Toufik, M. Baker, "LTE - The UMTS Long Term Evolution: From Theory to Practice", pp. 25-55, Wiley, 2011.

[3] ETSI. Group Report NFV001. "Network Functions Virtualisation (NFV); Use Cases", October 2013.

[4] ETSI. Group Specification NFV002. "Network Functions Virtualisation (NFV); Architectural Framework ", December 2014.

[5] ETSI. Group Specification NFV-MAN 001. "Network Functions Virtualisation (NFV); Management and Orchestration", December 2014.

[6] ETSI. Introductory Technical White Paper. "Mobile-Edge Computing", September 2014.

[7] ETSI. Group Specification MEC003. "Multi-access Edge Computing (MEC); Framework and Reference Architecture", March 2016.

[8] D. Faccin, "A 5G-ready Management and Automation Platform for the Telco and Edge Clouds: Design and Experimentation of a Unified NFV and MEC System" [Unpublished master thesis], University of Padova, 2019.

[9] B. Nogales, I. Vidal, J. Garcia-Reinoso, D. R. Lopez, J. Rodriguez and A. Azcorra, "Design and Deployment of an Open Management and Orchestration Platform for Multi-site NFV Experimentation", in *IEEE Communications Magazine*, vol. 571, pp. 20-27, January 2019.

[10] K. Antevski, C. J. Bernardos, L. Cominardi, A. de la Oliva, and A. Mourad, "On the integration of NFV and MEC technologies: architecture analysis and benefits for edge robotics", in *Computer Networks 175*, April 2020.

[11] L. Van Ma, V. Q. Nguyen, J. Park, and J. Kim, "NFV-Based Mobile Edge Computing for Lowering Latency of 4K Video Streaming", in *Tenth International Conference on Ubiquitous and Future Networks*, pp. 670-673, 2018.

[12] ETSI. White Paper n.24. "MEC Deployments in 4G and Evolution Towards 5G", February 2018.

[13] ETSI OSM Community White Paper. “OSM Release THREE, A Technical Overview”, October 2017.

[14] ETSI OSM Community White Paper. "OSM VNF onboarding guidelines", June 2019.

[15] ETSI. Group Specification NFV-SOL 002. "RESTful protocols specification for the Ve-Vnfm Reference Point", August 2017.

[16] ETSI. Group Specification MEC 011. "Mobile Edge Computing (MEC); Mobile Edge Platform Application Enablement", July 2017.

[17] I. Sodagar, "The MPEG-DASH Standard for Multimedia Streaming Over the Internet," in *IEEE MultiMedia*, vol. 18, pp. 62-67, April 2011.