# An ETSI NFV Implementation for Automatic Deployment and Configuration of a Virtualized Mobile Core Network

Francesco Asquini*, Armir Bujari†, Daniele Munaretto*, Claudio E. Palazzi‡, Daniele Ronzani‡
*Athonet S.R.L., Italy
†Department of Computer Science and Engingeering, University of Bologna, Italy
‡Department of Mathematics "Tullio Levi-Civita", University of Padua, Italy
*{name.surname}@athonet.com
†{name.surname}@unibo.it
‡{name.surname}@unipd.it

***Abstract*—The recent trend of network disaggregation and community push towards open source network management and orchestration projects has given rise to a rich ecosystem of frameworks and tools. In particular, ETSI NFV proposes a conceptual, reference architecture standardizing management and control interfaces in softwarized mobile broadband networks. These frameworks present highly customizable features, embodying varying degrees of complexity, communed by a steep learning curve. In this article, we propose a step-by-step description of a virtualized testbed environment setup, adopting an NFV-compliant ecosystem, describing an implementation of the NFV SOL002 reference point which aims to standardize the lifecycle management of virtualized functions. In addition, we present an experimental assessment measuring the mobile core deployment times.**

***Index Terms*—Virtualization, VNF, OSM, mobile core**

## I. Introduction

Cellular networks have become the de facto means for communication. In particular, 5G connectivity represents the next generation of ultra-broadband mobile technology, paving the road to many application areas by leveraging on its performance in terms of high throughput and very low latency [1]–[5]. Industry 4.0, C-V2X, etc. represent some use-cases where 5G is seen as a key technological block [6]–[10]. In this context, the increasing demand of mobile connectivity and traffic increase rates, pose new and critical requirements to mobile networks, in terms of delay, throughput and loss tolerance. Furthermore, the increasing Capital and Operating Expenditures (Capex and Opex) for deployment and maintenance of network assets, are becoming unsustainable for network operators. The evolution of information technology has encouraged the advent of promising research directions, like that of Network Function Virtualization (NFV) and cloud/edge-centric computing. NFV is the paradigm where network functions, defined as functional building blocks of a network infrastructure, with external interfaces and functional behavior (e.g., router, gateway, mobile core, load balancer, etc.), may be implemented as pure software. This innovation is opening new opportunities for network operators and infrastructure providers, aiming at removing the strict dependency of network services on the underlying hardware infrastructure and enhancing network deployment flexibility, scaling and costs. In particular, technologies such as the ETSI NFV [11] and Multi-access Edge Computing (MEC) are considered as key enablers, bringing network automation capability and supporting innovative use-cases [12]–[19].

In particular, since the release of the first ETSI NFV *MANagement and Orchestrator* (MANO) draft, different projects have emerged proposing highly customizable software frameworks aimed at addressing network automation issues [11], [20], [21]. These software frameworks embody a multitude of highly customizable features, posing a higher barrier of entry for practitioners and users.

To this end, in this article we propose a step-by-step description of a virtualized testbed environment setup, adopting an NFV-compliant ecosystem, consisting of OpenStack [22] and Open Source MANO [20], to instrument the deployment of a fully virtualized mobile network core [23]. As an addon, we propose

This work has been partially funded by the FSE project cod. 2105-0601463-2019 "Design and Development of 5G Mobile Network Systems for the Smart Industry 4.0" and by the Department of Mathematics of the University of Padua through the BIRD191227project.

an implementation of the ETSI NFV SOL002 reference point [24], standardizing a set of RETSful APIs aimed at lifecycle management of virtualized network functions. An experimental assessment is presented, measuring the mobile core deployment times, decomposing the delay contributions of each deployment phase.

This paper is organized as follows: Sec. II provides some background information on the ETSI NFV reference architecture, with particular focus on SOL002 interface, while Sec. III discusses some related work in the context of ETSI NFV. Sec. IV provides an overview on the adopted software ecosystem. Sec. V provides some insights into our implementation of the SOL002 RESTful APIs. Sec. VI illustrates the testbed, along with a measurement study of the deployment process. Finally, Sec. VII concludes the paper.

## II. Network Function Virtualization

Traditionally, network functions are realized through a variety of specific combinations of dedicated hardware and proprietary software. Although hardware and software are optimized for network functions, they have the drawback of being designed only for a subset of functions and not being scalable, as well as being subject to deterioration, requiring continuous maintenance. For this reason, in 2012, the European Telecommunications Standards Institute (ETSI) defined the Network Function Virtualization (NFV) reference architecture, where network functions are implemented as purely software entities, called Virtual Network Functions (VNF), which can run on appropriate virtualization infrastructures. The main idea behind NFV is to break the strict binding between software and hardware in current systems, by designing software-only network elements.

The NFV reference architecture exploits 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 networking resources. The advantage in doing this is the reduction of the number of different devices in a network and the complexity of deploying, configuring, and maintaining them. ETSI standard defined the NFV architecture [24] that illustrates a high-level functional view to support VNF operations. With reference to Fig. 1, three main working domains are identified:

- **NFV Infrastructure (NFVI)**, the underlying physical resources and hypervisor of virtualized elements; the NFVI provides a virtualized environment for all the VNFs and communicates with the MANO through the *Nf-Vi* connection point.
- **Virtual Network Functions (VNFs)**, the software implementation of network functions capable of running over the NFVI; the MANO can interface to VNFs through the *Ve-Vnfm* connection point, applying SOL002 RESTful API commands (evidenced in red in the figure). Another module on top of the VNF, the Element Manager (EM), is responsible of FCAPS (fault, configuration, accounting, performance, and security) management of the actual network application residing in the VNF itself.
- **Management and Orchestrator (MANO)**, covers the orchestration layer of physical and virtualized resources and manages the entire VNF lifecycle. The MANO can receive or send information to the vendor/operator-specific *Operations Support System and Business Support System* (OSS/BSS) through the *Os-Ma* connection point.

In the following, we synthetically discuss some literature material about the NFV domain, outlining relevant features of the NFV reference architecture.

## III. Related Work

There is a vast body of literature related to NFV and orchestration, spanning different areas of networking. However, to the best of our knowledge, there is a lack of studies devoted to the implementation and evaluation of the ETSI NFV APIs for configuration and automation of deployments. The authors in [26] provide a general overview of the APIs exposed by the components of the NFV system. They focus on RESTful characteristics of the APIs, providing usage examples regarding the lifecycle of a VNF.

In [27], a novel testbed, called 5GIIK is introduced, proposing a framework for network slicing across dif-

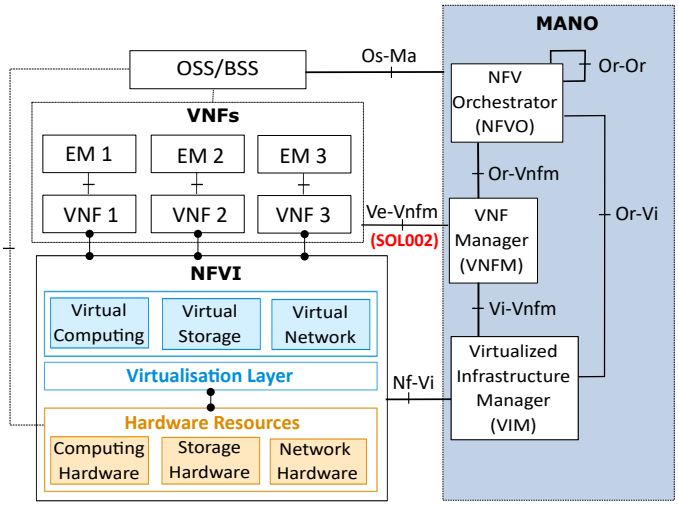


Fig. 1. NFV reference architectural framework, highlighting in red the SOL002 API context [24].

ferent domains and technologies. This is achieved by making use of open-source tools and additional features and capabilities to include dynamicity and real-time monitoring to the VNFs.

A comprehensive state-of-the-art analysis of open-source software and frameworks for 5G networks has been conducted in [28]. The authors describe in detail and compare capabilities and functionalities of each of them, concluding with a discussion of hardware and testbeds where these frameworks can run and their limitations.

In [29], a performance comparison of the OSM (release 4) and the ONAP softeare solution has been undertaken measuring some specific MANO reference KPIs. The two solutions have been also compared in terms of required virtual CPU, memory and storage.

In the following section, we illustrate the virtualization and orchestration tools that we will use in our experimental framework, aligned with ETSI NFV specifications [21].

## IV. Key Enabling Technology and Tools

In this section, we identify and briefly describe our software choice for the NFV management and orchestration implementation. The main aspect that have been considered are the alignment with ETSI specifications, as well as the maturity and stability of the solutions. The adopted software framework and its alignment to ETSI NFV standard architecture is illustrated in Fig. 2. Open Source MANO implements a full MANO stack including the VNFM and NFVO entities; it adheres to the ETSI standards and supports the integration with many compliant VIMs, among which we chose OpenStack. Both the applications are open source and offer a large and continuous community contribution.

### *A. OpenStack*

OpenStack [22] provides an Infrastructure-as-a-Service (IaaS) solution for public and private clouds. It manages hardware pools and provides a virtual infrastructure for service or network deployments. OpenStack operates as a VIM in the NFV MANO framework and orchestrates NFVI resources, instantiating VMs and virtual networks.

The OpenStack project includes a set of independent microservices, each deliverying a specific functionality. Some of these services are not strictly needed and a typical minimal installation comprises the following five modules:

- Keystone (Identity service), which implements client authentication and authorization, managing user credentials.
- Glance (Image service), which provides the management of virtual machines' images.
- Nova (Compute service), which enable the provisioning of compute instances, allowing the creation/destruction of virtual machines on-demand.
- Neutron (Networking service), which provides network connectivity between the services.
- Horizon (Dashboard), which provides a web based interface to interact with the services.

### *B. Open Source MANO*

Open Source MANO (OSM) [20] is an ETSI project that implements the MANO layer, in particular the functionalities of NFV Orchestration (NFVO) and VNF Manager (VNFM) and can relate to a VIM (e.g., Openstack, AWS, etc.). Its main purpose is to deploy, orchestrate and automate a network service (NS) under a virtualization infrastructure. A NS can be deployed as a single VNF or can be composed by multiple chained VNFs realizing a certain end-to-end service. OSM adheres to the ETSI NFV set of standards; for that reason, it has been widely considered for telco services, inasmuch it removes uncertainties and captures the complexity of real systems. OSM approach aims to minimize the integration effort thanks to a well-known information model (IM), which allows to model and automatize the complete life-cycle of network functions (virtual, physic, or hybrid) and network services [25]. The IM is completely infrastructure agnostic, meaning that the same model can be instantiated on different VIMs.

OSM provides three different phases for VNF deployment, 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. Day-0 procedures regard the basic VM instantiation stages and initial configuration, aiming to make the VNF operational and manageable. The basic instantiation is configured by VNF descriptors (VNFD), templates used to describe every VNF attribute. The VNFD is defined by a yaml file loaded into OSM via GUI or CLI. Day-1 procedures apply the service initialization for VNFs; they are automatically executed right after instantiation. To perform day-1 operations, OSM makes use of Juju, a framework that enables the execution of life-cycle configuration routines through the use of proxy charm. Day-2 procedures allows to perform actions to the VNF at any time after the VNF is installed and configured; we exploit this option to make the VNF reconfiguration. In the next section, a more in-depth description of Juju and proxy charms is provided.

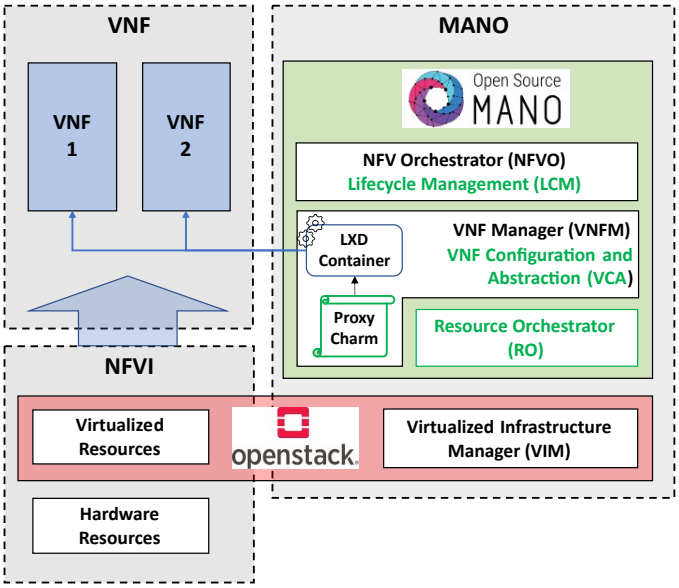


Fig. 2. ETSI NFV framework architecture, related to the chosen software. Note the OSM modules (green) implementing MANO entities.

### C. Juju and proxy charms

Juju is an application modelling tool that supports the configuration and scaling of cloud applications through the executions of software scripts, called proxy charms. Juju is adopted by the VNF Configuration and Abstraction (VCA) module, the OSM implementation of VNFM. A proxy charm is a collection of YAML descriptors, support files and executable code, written either in Python or Bash language, which provide the actual executive actions onto the VNF, via SSH, RESTful API, etc.). From the point of view of ETSI NFV architecture, a proxy charm may represent the EM) of a VNF. Inside a proxy charm a set of actions, also called *primitives*, are implemented, to be executable on the VNF.

## V. SOL002 API implementation

In this section we expose our implementation of the SOL002 configuration API, which OSM adopts to configure the instantiated VNF. First, we provide some details about the API, then we illustrate the specific configuration field, and finally we illustrate our actual implementation.

### A. SOL002 configuration API

ETSI group specification SOL002 [24] outlines the set of RESTful APIs and data models supported over Ve-Vnfm interface. The SOL002 APIs define different administration procedures on the VNFs, such as performance management, fault management, and network configuration. Each of them is accessible via HTTP protocol, trough a corresponding URL of the form:

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

In our work, we implemented the configuration part, where `apiName` is set to "vnfconfig" and the only resource made available by the VNF is the `configuration` resource. In the specific, the configuration access is made via the URL:

```
{apiRoot}/vnfconfiguration/v1/configuration
```

with `apiRoot` containing the IP address of the entity in charge of applying the configuring (which can be the VNF itself).

### B. SOL002 configuration field

The scope and focus of the SOL002 interface is that of configuring a deployment of the virtual core. This entails the association of each VNFC connection point to a physical interface of the VM, the inclusion and configuration of network routes, IP addresses, and other parameters specific to the virtual core deployment. Since part of these configuration data is not natively supported by SOL002, we exploited the optional field `vnfSpecificData` made available by the API. It is a general field providing key/value pairs, whose purpose is to allow vendors and operators to perform their specific configuration. Hence, the SOL002 request is accompanied by a JSON payload with the following structure:

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

A single request may contain the configuration of an arbitrary number of VNF components, each identified by `vnfInstanceId` and with a connection point defined in `CpConfig`. Finally, the optional field `vnfcSpecificData` serves to define all the specific parameters for the virtual core configuration, not otherwise specifiable.

### C. SOL002 client/server implementation

The VNF representing the virtual core receives from OSM the SOL002 configuration request and then translates it into a request consistent with the proprietary API exposed by the virtual core, by which the configuration is effectively applied. Hence, a *server* agent has been implemented as a separate VNF component, with

the task of exposing the SOL002 configuration API and performing such translation.

The *client* side has been made through the Juju charm, which sends the VNF configuration as part of day-1 procedure. The charm implements the primitive of configuration request for the VNF instance and is developed in Python. More in detail, the action implements the RESTful API that manages the `configuration` resource.

The implemented primitive is called `initial-patch-configuration`, and accepts in input the IP address to reach VNF and a json file containing the VNF configuration 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: vnf-hostname
      value: <vnf_IP>
    - name: vnfc-configuration-data
      value: <json_conf_file>
  juju:
    charm: sol002-client
```

The primitives defined within the `initial-config-primitive` section are called right after the NS instantiation. The values enclosed by angle brackets represent the VNF address and the SOL002 configuration, respectively; their value is defined by the user in the NS instantiation request.

When performing the NS instantiation request, it is possible to define the set of parameters to pass to Juju during instantiation, referring them in the `additionalParamsForVnf` section of the yaml file attached to the instantiation command:

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

## VI. Testbed Implementation and Evaluation

This section outlines the experimental environment.

### A. Environment Setup

The testbed is implemented in a virtualization infrastructure based on an OpenStack instance. For practical reasons OSM is installed and deployed into a VM instantiated on OpenStack itself. The hardware and software characteristics of the host are reported in Table I. Openstack provides connectivity on the physical network to the VMs through its internal gateway. All the instances have a port attached to the `private` internal Openstack network for external connectivity.

TABLE I
Host characteristics of the testbed.

| **HW Component** | **Characteristic** |
|---|---|
| CPU | Intel Core i7-2600, 3.4 GHz |
| RAM | 16 GB DDR3 |
| Storage | SSD, 480 GB |
| **SW Component** | **Characteristic** |
| OS | Ubuntu 18.04 LTS (bionic) |
| OpenStack | Stein |
| OSM | 8.0.4 (2020-07-01) |
| Juju | 2.8.8-bionic-amd64 |
| Charm-tools | 2.8.2 |

### B. VNF instantiation and configuration

The instantiation procedure can be launched from OSM command line client:

```
osm ns-create --ns_name vCore --nsd_name full_vCore_ns \
--vim_account openstack --config_file vCore_config.yaml
```

The required VNF is deployed at OpenStack site and the internal and external connectivity is set up, following the information contained in the descriptors. Then, the charm performs the configuration request over the Ve-VNFM interface toward the VNF, which subsequently validates and applies the network settings to the VNF components.

### C. Results

The testbed described above has been successfully tested with a 4G core network. Once the network service is ready, OSM acknowledges the successful instantiation and correct configuration, by showing a visual confirmation on the web client. The instantiation and configuration process has been verified by analysing the completion time performance.

A total of 100 VNF instances have been deployed in order to outline the average delay. As showed in Fig. 3, the total completion time (red) is 363.86 s and is divided into instantiation time (green) and configuration time (orange). The VNF configuration time largely depends on both the underlying infrastructure where OSM is running and the proxy charm structure itself, and includes two main phases: a) Juju bootstrap processes, which create a Linux Container (LXC) to then load the executable proxy charm on in (Juju app creation phase), b) the actual virtual core configuration phase, which executes the proxy charm and therefore applies the SOL002 configuration request to the VNF.

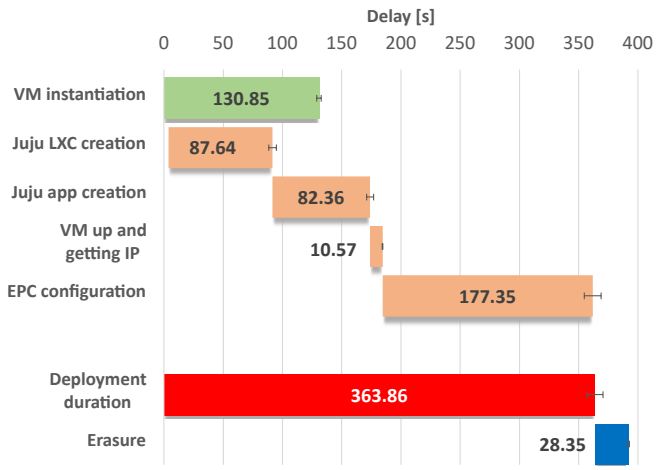


Fig. 3. VNF instantiation, configuration and deletion delay.

Actually, the two times are partially overlapping; in fact, the Juju bootstrap process is started a few seconds later the beginning of the VM instantiation, since these two steps are independent from each other and can start simultaneously. Finally, the average VNF deletion time is 28.35 s.

## VII. Conclusion

This paper presents the implementation and analysis of ETSI NFV testbed aimed at the automatic deployment and configuration of a virtual core network. The testbed relies on open-source software, interacting with the virtual core through the Ve-Vnfm standardized interface, enabling advanced configuration of the virtualized network functions. We demonstrated the feasibility of the process and verified the correct functionality of the virtual network configured via the SOL002 RESTful API. A time performance measurement showcased the efficiency of the software integration.

## References

[1] D.-H. Luong, H.-T. Thieu, A. Outtagarts and Y. Ghamri-Doudane, “Cloudification and Autoscaling Orchestration for Container-Based Mobile Networks toward 5G: Experimentation, Challenges and Perspectives”, in Proc. of *IEEE VTC Spring*, Porto, Portugal, Jun 2018.

[2] N. A. Johansson, Y.-P. E. Wang, E. Eriksson and M. Hessler, “Radio access for ultra-reliable and low-latency 5G communications”, in Proc. of *IEEE ICCW*, London, UK, Jun 2015.

[3] M. Giordani; M. Polese; M. Mezzavilla, S. Rangan and M. Zorzi, “Toward 6G Networks: Use Cases and Technologies”, *IEEE Communications Magazine*, vol. 58, no. 3, 2020.

[4] A. Bujari, M. Luglio, C. E. Palazzi, M. Quadrini, C. Roseti and F. Zampognaro, “A Virtual PEP for Web Optimization over a Satellite-Terrestrial Backhaul”, *IEEE Communications Magazine*, vol. 58, no. 10, 2020.

[5] S. Hamdoun, A. Rachedi and Y. Ghamri-Doudane, “Graph-Based Radio Resource Sharing Schemes for MTC in D2D-based 5G Networks”, *Mobile Netw. and App.*, vol. 25, no. 3, 2020.

[6] C. A. Kerrache *et al.*, “TACASHI: Trust-Aware Communication Architecture for Social Internet of Vehicles”, *IEEE Internet of Things*, vol. 6, no. 4, 2018.

[7] G. Marfia, M. Roccetti, A. Amoroso, M. Gerla, G. Pau and J.-H. Lim, “Cognitive Cars: Constructing a Cognitive Playground for VANET Research Testbeds”, *ACM CogART*, vol. 29, 2011.

[8] A. M. Vegni, T. D. C. Little, “A Message Propagation Model for Hybrid Vehicular Communication Protocols”, in Proc. of *CSNDSP 2010*, Newcastle upon Tyne, UK, 2010.

[9] V. Loscri, P. Manzoni, M. Nitti, G. Ruggeri, A. M. Vegni, “A Social Internet of Vehicles Sharing SIoT Relationships”, in Proc. of *PERSIST-IoT*, Catania, Italy, 2019.

[10] L. D. Xu, E. L. Xu, L. Li, “Industry 4.0: State of the Art and Future Trends”, *International Journal of Production Research*, vol. 56, no. 8, 2018.

[11] ETSI. Group Report NFV001. “Network Functions Virtualisation (NFV); Use Cases”, Oct 2013.

[12] A. Salam *et al.*, “Implementation of Virtualised Network Functions (VNFs) for Broadband Satellite Networks”, in Proc. of *EuCNC 2019*, Valencia, Spain, 2019.

[13] Z. Xu, W. Gong, Q. Xia; W. Liang, O. F. Rana, G. Wu, “NFV-Enabled IoT Service Provisioning in Mobile Edge Clouds”, *IEEE Transactions on Mobile Computing*, vol. 20, no. 5, 2021.

[14] S. Mirri, C. Prandi, P. Salomoni, L. Monti, “Social Location Awareness: A Prototype of Altruistic IoT”, in Proc. of 8th IFIP NTMS, Larnaca, Cyprus, 2016.

[15] A. Bujari, M. Furini, F. Mandreoli, R. Martoglia, M. Montangero, D. Ronzani, “Standards, Security and Business Models: Key Challenges for the IoT Scenario”, *Mobile Networks and Applications*, vol. 23, no. 1, 2018

[16] A. Bujari, O. Gaggi, C. E. Palazzi, D. Ronzani, “Would Current Ad-Hoc Routing Protocols be Adequate for the Internet of Vehicles? A Comparative Study”, *IEEE Internet of Things Journal*, vol. 5, no. 5, 2018.

[17] A. Bujari, O. Gaggi, M. Luglio, C. E. Palazzi, G. Quadrio, C. Roseti, F. Zampognaro, “Addressing the Bandwidth Demand of Immersive Applications Through NFV in a 5G Network”, *Mobile Networks and Applications*, vol. 25, no. 3, 2020.

[18] M. A. Lema *et al.*, “Business Case and Technology Analysis for 5G Low Latency Applications”, *IEEE Access*, vol. 5, 2017.

[19] C. E. Palazzi, A. Bujari, “A Delay/Disruption Tolerant Solution for Mobile-to-Mobile File Sharing”, in Proc. of *IFIP Wireless Days*, Venice, Italy, 2010.

[20] ETSI, Open Source MANO (OSM) Project, accessed: 2021-02-27, [Online] Available: https://osm.etsi.org.

[21] ETSI. Group Specification NFV-MAN 001. “Network Functions Virtualisation (NFV); Management and Orchestration”, 2014.

[22] OpenStack, ”OpenStack project portal”, accessed: 2021-02-27, [Online] Availabe: https://www.openstack.org.

[23] Athonet, Private LTE and 5G Networks for Enterprise, accessed: 2021-02-27, [Online] Available: https://www.athonet.com/private-lte/.

[24] ETSI. Group Specification NFV002. “Network Functions Virtualisation (NFV); Architectural Framework”, Dec 2014.

[25] ETSI, OSM End User Advisory Group. “OSM White Paper: Experience with NFV Architecture, Interfaces, and Information Models”, May 2018.

[26] B. Chatras. “On the Standardization of NFV Management and Orchestration APIs”, in *IEEE Communications Standards Magazine*, vol. 2, no. 4, 2018.

[27] A. Esmaeily, K. Kralevska and D. Gligoroski, “A Cloud-based SDN/NFV Testbed for End-to-End Network Slicing in 4G/5G,” in Proc. of *IEEE NetSoft*, Virtual Conference, 2020.

[28] L. Bonati, M. Polese, S D’Oro, S. Basagni, and T. Melodia, “Open, programmable, and virtualized 5G networks: State-of-the-art and the road ahead,” *Comp. Netw.*, vol. 182, 2020.

[29] G. M. Yilma, F. Z. Yousaf, V. Sciancalepore, and X. Costa-Perez, “On the challenges and KPIs for benchmarking open-source NFV MANO systems: OSM vs ONAP”, *Computer Communications*, vol. 161, 2020.