An ETSI-compliant Architecture for the Element Management System: The Key for Holistic NFV Management

Vinicius Fulber-Garcia
Department of Informatics
Federal University of Paraná
vinicius@inf.ufpr.br

José Flauzino
Department of Informatics
Federal University of Paraná
jwvflauzino@inf.ufpr.br

Carlos R. P. dos Santos
Department of Applied Computing
Federal University of Santa Maria
csantos@inf.ufsm.br

Elias P. Duarte Jr.
Department of Informatics
Federal University of Paraná
elias@inf.ufpr.br

***Abstract*—The Element Management System (EMS) is defined by the ETSI NFV reference architecture as responsible for the instrumentation of Virtualized Network Functions (VNF). The EMS acts as a gateway to each function, executing monitoring and control requests received from the VNF Manager (VNFM) and Operation and Business Support Systems (OSS/BSS). Surprisingly, the EMS has been either entirely ignored or only employed in very restricted settings. In this work, we propose an ETSI-compliant architecture that allows the development of EMS solutions that can be used to manage arbitrary VNF instances while abstracting particular protocols and technologies, thus enabling the interoperability among heterogeneous systems (VNF, VNFM, and OSS/BSS). The architecture was implemented as the Holistic, Lightweight, and Malleable EMS Solution (HoLMES). Evaluation results are presented showing the overhead of using HoLMES as a gateway for typical management operations. The slight increase in execution times is a fair price to pay for the benefits that the holistic solution represents in terms of the effectiveness of NFV management.**

I. Introduction

The Network Function Virtualization (NFV) paradigm allows the implementation in software of network functions and services that run in the network core. Commercial off-the-shelf servers executing virtualization technology replace middleboxes that are often implemented as physical appliances based on dedicated and proprietary hardware. Both the ETSI and the IETF have been specifying recommendations, models, and enablers to standardize the NFV paradigm. In particular, the ETSI has proposed a widely adopted NFV reference architecture [1], and the IETF has proposed several specifications in the context of virtualized network services [2], [3].

The ETSI NFV reference architecture revolves around three different domains: MANO (MANagement and Orchestration), NFVI (NFV Infrastructure), and VNF (Virtualized Network Function). The NFV-MANO domain is responsible for supporting the virtualized infrastructure, managing network functions, and orchestrating network services. The NFVI, in turn, includes the physical resources and their virtualization, making them available to the other domains. Finally, the VNF domain handles the execution and management of the lifecycle of network function instances.

A VNF is actually defined as consisting of two parts [4]: the Network Function (NF) and the VNF Platform on which it is executed. Examples of such VNF platforms include ClickOS [5], OpenNetVM [6], Click-on-OSv [7], and COVEN [8]. The NF processes packets applying whatever functionalities it was built to accomplish. Network Functions run on VNF Platforms, that provide a myriad of resources for their execution [9], [10].

Two blocks are defined for the management of VNF instances: the VNF Manager (VNFM) and the Element Management System (EMS). The VNFM has been receiving much attention: multiple APIs [11], [12], applications [13], [14], and implementations [15], [16] have been recently developed. Surprisingly, in the context of NFV, the EMS has been either completely ignored or employed in restricted settings.

The EMS is responsible for VNF instrumentation and executes typical management operations (*i.e.*, FCAPS: Fault, Configuration, Accounting, Performance, and Security) on VNF instances [17]. The EMS also acts as a gateway, providing the interface for VNF Managers and Operation and Business Support Systems (OSS/BSS) to interact with VNF instances. Despite its importance, the EMS has been under-explored in this context. Even when it is present, the EMS is typically restricted to specific applications. Those EMS implementations are heterogeneous, often highly different from each other. Examples of application-oriented EMS solutions can be found in the context of 5G resource allocation [18], NFV support for multimedia communications [19], and in the context of fast packet processing [20].

Notwithstanding its importance, there is no reference architecture for implementing an EMS in the context of NFV. This fact, in practice, makes it hard for existing EMS solutions to cooperate with other operational blocks from the ETSI NFV reference architecture due to the lack of standard features and interfaces. In this way, each different EMS solution usually provides specific functionalities, both in terms of operation syntax and management capabilities. Different solutions often interact in wildly different ways with other blocks of the reference architecture, typically only working with specific VNFM, OSS/BSS, and VNF Platforms. Thus, this situation leads to the violation of two fundamental NFV requirements [21]: (i) integration (referred to as "coexistence with existing networks and transition"), as current EMS solutions only work in specific systems developed by particular vendors; and (ii) portability, as it is often impossible to employ a given EMS in heterogeneous NFV environments.

In this work, we propose a comprehensive and extensible architecture for the EMS operational block. This architecture enables the development of different yet compatible categories of EMS solutions. Furthermore, internal modules and communication interfaces are fully compliant with the ETSI NFV reference architecture. Thus, it is relevant to highlight that we do not contemplate other proposed standards and protocols for the EMS, such as the IETF and 3GPP ones. The proposed architecture allows the creation of EMS solutions that are holistic in the sense that they are not restricted to specific applications, protocols, and technologies, thus enabling the interoperability of VNFs, VNFMs, and OSS/BSSes from multiple vendors [8], [22].

The architecture was implemented as the Holistic, Lightweight, and Malleable EMS Solution (HoLMES - https://github.com/ViniGarcia/HoLMES). HoLMES was evaluated in the execution of typical NFV management operations. Results show that HoLMES is capable of successfully managing heterogeneous VNF instances providing seamless integration with different VNFM and OSS/BSS platforms. The slight increase in execution times is a fair price to pay for the benefits that the holistic solution represents in terms of the effectiveness of NFV management.

The rest of this paper is organized as follows. Section II presents definitions on the Element Management System. Next, Section III gives an overview of the state-of-art NFV management systems related to this work. In Section IV, the proposed internal architecture for the Element Management System is presented. HoLMES, the open-source EMS implemented is described in Section V. Evaluation results follow in Section VI. Finally, Section VII concludes the paper.

II. The Element Management System

The Element Management System (also called Element Manager – EM) was first proposed in 1988 in the context of TMN (Telecommunications Management Networks) [23] of the International Telecommunication Union (ITU). In that context, TMN processes have *managed* and *managing* roles. While the managed role refers to providing information related to managed resources, the managing role receives notifications and issues requests for information on managed resources. Equipment (or virtual equipment) executing a managed or managing process in TMN is considered to be an EMS.

In 2018, the 3rd Generation Partnership Project (3GPP) released a document specifying the service-based management architecture [24]. The EMS is an element responsible for managing and orchestrating functions and services in this architecture. In the 3GPP architecture, the EMS is called a Management Service (MnS) and has the same roles defined originally in the TMN specifications – managed and managing – here called MnS producers and consumers. In the 3GPP architecture, an MnS is provided and accessed by a logical entity called Management Function (MnF), which can operate within a network function or as an independent element.

3GPP adopts the service-based management architecture in the NFV architecture proposed by the ETSI. A VNF holds an MnS producer working as a Management Agent [8], and the EMS becomes an MnS consumer for that VNF, while also working as an MnS producer for other NFV architecture blocks, such as the VNFM, NFVO, and OSS/BSS. However, the 3GPP stops at this point, *i.e.*, it neither provides any hint on how these management services can be implemented nor on how to deal with the data models and management operations specified by the ETSI for NFV [17].

A. The Element Management System in NFV

The Element Management System (also called Element Manager – EM) is defined by the NFV reference architecture [1] as the block responsible for the instrumentation of VNF instances [25], [26]. The EMS also acts as a gateway, collecting and processing information from one or more VNF instances and interacting with other management blocks of the NFV architecture [17], in particular the VNFM but also the OSS/BSS. We next describe and classify the EMS according to its functionalities and characteristics, including operational features and requirements [1], [17], [27]–[29].

First, an EMS can be classified according to its relationship with the NFV-MANO domain. Two alternatives are possible [28]: either the EMS is `non-MANO` or `MANO-compliant`. A non-MANO EMS does not have any restrictions in terms of the NFV environment in which it is executed, *i.e.*, the EMS can be deployed regardless of the existence of any other standard NFV blocks and, if there is such a block, the EMS interacts with that block as if with a generic "user". An example of a non-MANO EMS is one designed for self-managed software running on bare metal; another example is the EMS designed for systems deployed outside the NFVI and that do not communicate with NFV-MANO. Yet other non-MANO EMS solutions are those under the control of an OSS/BSS.

A MANO-compliant EMS, in turn, is fully aware of the NFV environment and properly communicates/cooperates with the other management blocks of the reference architecture. The MANO-compliant EMS solutions can be of three different types depending on how they are deployed [28], [29]: `VNFC EMS`, `VNF EMS`, and `Standalone EMS`. A VNFC EMS runs within a network function instance and can be considered an internal component of a network function. On the other hand, a VNF EMS runs independently of the managed network function and is recognized by NFV-MANO as a VNF instance itself, but of a special class: it does not process network traffic. Finally, a Standalone EMS executes on a management plane of the VNF working domain that is decoupled from both the VNF instances (such as a VNFC EMS is) and is not under any direct control of any NFV-MANO block (such as the VNF EMS is). However, different from a non-MANO EMS, a Standalone EMS does employ the standard communication interfaces to execute management operations as it interacts with the NFV environment.

Another classification of EMS solutions is based on where it is deployed concerning the corresponding VNF instances it is responsible for managing. In this case, an EMS can be either an `Internal EMS` or an `External EMS` [29]. An Internal EMS runs together with the corresponding VNF instance, resulting in a self-managed VNF. It is important to notice that every MANO-compliant VNFC EMS is an Internal EMS, but the opposite is not true (A standalone EMS can share the same virtual instance and resources as a VNF, with the VNF remaining agnostic of it). An EMS classified as non-MANO or MANO-compliant Standalone can execute as an Internal VNF provided that they execute within the VNF Platform operation system. An External EMS, however, does not run within the corresponding VNF instance, and the communication between an external EMS and a VNF employs protocols that can change according to the VNF platform used to execute the network functions (those protocols are out of the scope of the ETSI specifications).

An EMS can be also classified as `Generic` or `Specific` [29], depending on whether it executes only the set of standard management operations defined in [17] or also particular operations of specific VNF Platforms. A Generic EMS interacts with the NFV environment using a comprehensive interface that allows the execution of the basic set of management operations. For a Specific EMS to execute operations besides those of the basic set, it employs some pre-agreed protocol (*e.g.*, defined by VNF platform vendors) to communicate with the platforms running the VNF instances.

Finally, an EMS can manage one or multiple VNF instances and can be classified accordingly [1], [27], [29]: a `Dedicated EMS` manages a single VNF instance and an `Umbrella EMS` manages multiple VNF instances. While a Dedicated EMS does not require mechanisms to identify on which VNF instance it is supposed to execute an operation, an Umbrella EMS does. An Umbrella EMS not only needs to keep track of the multiple VNF instances but also employ a routing strategy to make operations reach the corresponding functions. The implementation of an Umbrella EMS solution is thus more complex than that of a Dedicated EMS.

III. Related Work

Although the EMS is defined in the ETSI NFV reference architecture, current NFV implementations are often restricted to specific MANO projects, such those of OpenStack Tacker [15], CloudStack Vines [16], Open Source Mano (OSM) [30], and OpenBaton [31]. The situation is complicated by the fact that each of those implementations adopts a different communication protocol employed to establish the communication between VNFM and EMS. Often, each EMS is based on management operations that are also specific to the particular MANO project. Next, we describe existing EMS solutions, also classifying each solution according to the criteria defined in Section II.

Tacker [15] is an NFV-MANO project that includes an NFV Orchestrator (NFVO) and VNFM. Tacker employs OpenStack as VIM and NFVI. There is no native EMS available for Tacker. However, Cloud-Init can be considered a primitive type of EMS, running Shell Scripts to execute configuration routines on VNF instances. Nevertheless, Cloud-Init cannot be considered a complete EMS, and might be categorized as `non-MANO, internal, and dedicated` (kind of) EMS – generic and restrict classifications do not apply in this case.

OSM [30] is an NFV-MANO project from the ETSI that is based on containers. OSM also works with the OpenStack cloud as VIM and NFVI on which VNFs and services are instantiated. OSM also allows users to employ Kubernetes to virtualize and manage VNF instances. OSM does not include a proper EMS, but it is possible to configure a VNF internally using a system called Day-Like. Similar to Cloud-Init, Day-Like is a configuration agent, not exactly an EMS. Furthermore, the execution Day-Like scripts rely on tools that must have been previously installed on the VNF Platforms, such as Secure Shell (SSH) and proxy charms. Thus, the Day-Like system can be categorized as `non-MANO, internal, and dedicated`.

The Vines project [16] consists of an NFV-MANO system with a native VNFM and NFVO that employs the CloudStack cloud as VI and VIM. This project provides an EMS implemented as a network element of CloudStack. The Vines EMS can natively communicate and manage several network functions running over the Leaf Platform, also proposed as part of the project. Furthermore, the EMS can communicate with other VNF Platforms through a driver mechanism. The CloudStack Vines EMS can be classified as `MANO (VNF), external, restrict, and umbrella`.

The OpenBaton project [31] is an extensible NFV-MANO project that features a flexible NFVO and VNFM. OpenBaton employs OpenStack as VIM/NFVI, but it is possible to create drivers to allow other cloud platforms. OpenBaton provides an EMS that executes internally to the VNF Platforms. This EMS is customizable in the sense that it enables users to define internal management operations. Those operations are available through REST (Representational State Transfer) or AMQP (Advanced Message Queuing Protocol) interfaces. The OpenBaton EMS keeps running and responding to management requests during the complete lifecycle of a VNF instance. Thus, the OpenBaton EMS can be classified as `MANO (Standalone), internal, restrict, and dedicated`.

All the EMS solutions described above are designed each for a particular NFV-MANO project and cannot cooperate with a VNFM of other projects [32]. It is important to note that, besides design limitations, this is also a consequence of the fact that none adopts the standard protocol defined for the Ve-Vnfm-em reference point [17]. Furthermore, most of those EMS solutions are internal. This category of EMS can be adopted by VNF platforms running on multi-process operating systems, such as Ubuntu Cloud and Alpine Linux, but is not appropriate for minimalist single-process platforms that host one VNF instance, such as ClickOS [5] and Click-On-OSv [7]. Those minimalist operating systems typically do not support the installation of tools that those EMS solutions require.

IV. The Proposed EMS Architecture

In the context of NFV, current EMS solutions are widely different from each other, mostly due to the fact that there is no EMS standard architecture to follow. As the EMS acts as a gateway that integrates NFV management blocks (*i.e.*, VNF, VNFM, OSS/BSS) and plays an important role in the execution of critical operations – such as those to configure, monitor, migrate, and scale virtualized network functions – EMS solutions should meet the requirements of the NFV paradigm in terms of integration and portability.

In order to fill this gap, in this section, we propose an EMS architecture shown in Figure 1. The architecture consists of six modules: *(i)* Access Subsystem (AS); *(ii)* Internal Router (IR); *(iii)* Monitoring Subsystem (MS); *(iv)* VNF Subsystem (VS); *(v)* EMS Management Information Base (EMIB); and *(vi)* Configuration Module (CM). The modules were designed to be loosely coupled. The architecture specifies interfaces between internal modules and other management blocks and thus readily supports implementations and solutions from diverse vendors. Furthermore, the proposed architecture can work with both VNFs and PNFs (Physical Network Functions). In the figure, solid arrows represent the communication between internal modules, while dashed arrows represent data exchanged in the execution of tasks related to the configuration and maintenance of internal modules themselves. Interfaces to other NFV blocks are shown as double solid lines. Other external interfaces, outside the scope of EMS *per se*, are shown as double dashed lines.

The proposed EMS assumes that VNF Platforms execute a Management Agent (MA) [8], which provides an interface to monitor and control the execution of VNF instances. This interface consists of at least five operations: request, retrieve, start, stop, and monitor. The request operation is used to deploy a VNF Package (VNFP) [33]. Once a VNF instance is executing, retrieve operations can be used to obtain information, *e.g.* the VNF ID or network interfaces supported. Start and stop are VNF lifecycle operations. Finally, monitoring provides a standard way to obtain VNF state information and measure performance parameters. It is important to highlight that the MA is not mandatory, it can be replaced by any other way to allow the EMS to access a VNF instance, such as an SSH interface.

The EMS executes operations received from other NFV management blocks as well as from operators and sends back task outcomes to the respective requester. The EMS can also trigger management operations and send operation requests to other NFV management blocks. The internal modules are described next.

Access Subsystem (AS) – The AS is responsible for receiving, validating, and forwarding them to the Internal Router. Furthermore, after the operation is processed, the AS also returns the results to the request source. The AS consists of two agents: the Operation Agent (OA) and the Authentication Agent (AA). The OA implements the Access Interface (AIf) of an EMS. It provides all the operations accessible externally to the EMS, whether for the VNF instance management or for the management of the EMS itself. The OA must follow the specification of the Ve-Vnfm-em interface [17], thus being ETSI compliant. The AA, in turn, checks if received requests come from a valid source with the proper authorization. Although the AA is optional, it is strongly recommended in the ETSI documents related to the EMS [34], [35]. The AA executes a request authentication after it is validated by the OA and before the request is forwarded to the Internal Router.

Internal Router (IR) – The IR forwards requests across internal modules. If the request requires the execution of a management operation on a VNF instance, it is forwarded to the VNF Subsystem. Otherwise, if the request is for some internal management operation, it is forwarded to the Configuration Module. Overall, the IR must keep track of the path a request follows within the EMS, as well as partial results, warnings, and errors. The outcome of an operation executed is forwarded to the AS, which in turn sends the outcome to the request source.

Monitoring Subsystem (MS) – This module provides an execution environment for running scripts to monitor VNF instances. Although a monitoring script can employ any operation of the VNF Subsystem, most of its operations are related to checking the health of virtualized instances (*e.g.*, using heartbeat messages received from a virtual machine or container); checking the status of objects maintained by network functions (*e.g.*, filters and inspectors); and keeping track of general performance metrics (*e.g.*, latency and jitter). EMS users can set up monitoring scripts using any operation available, requested through the MS-VS interface. Those scripts are executed continuously by the MS. The MS can also send notifications, alerts, and reports after relevant events are identified. These notification messages are sent through the Monitoring Interfaces (MIf) of each agent using the MS-MIf interface, thus avoiding overloading the AS.

VNF Subsystem (VS) – The VS establishes the communication of the EMS with one or more VNF Platforms. This module recognizes the execution platform of each particular VNF to choose the procedures available to execute a management operation and handle results as well as errors. Whenever possible the VS does more than just apply an operation to a VNF instance, also being capable of interpreting the results and presenting proper responses to the requester. The EMS-VNF communication occurs through VNF Interfaces (VIf). The management of these interfaces can vary according to the implementation of the EMS, and there may be multiple VIfs. For example, multiple VIfs can be used so that a specific VIf is defined for each supported VNF Platform. Furthermore, an exclusive VIf can be set up to allow the communication of each VS monitoring script.

EMS Management Information Base (EMIB) – This module stores and provides access to information related to the management of the EMS instance itself. The EMIB can be implemented in various ways, from a simple collection of local text files to a complex database system. The choice of

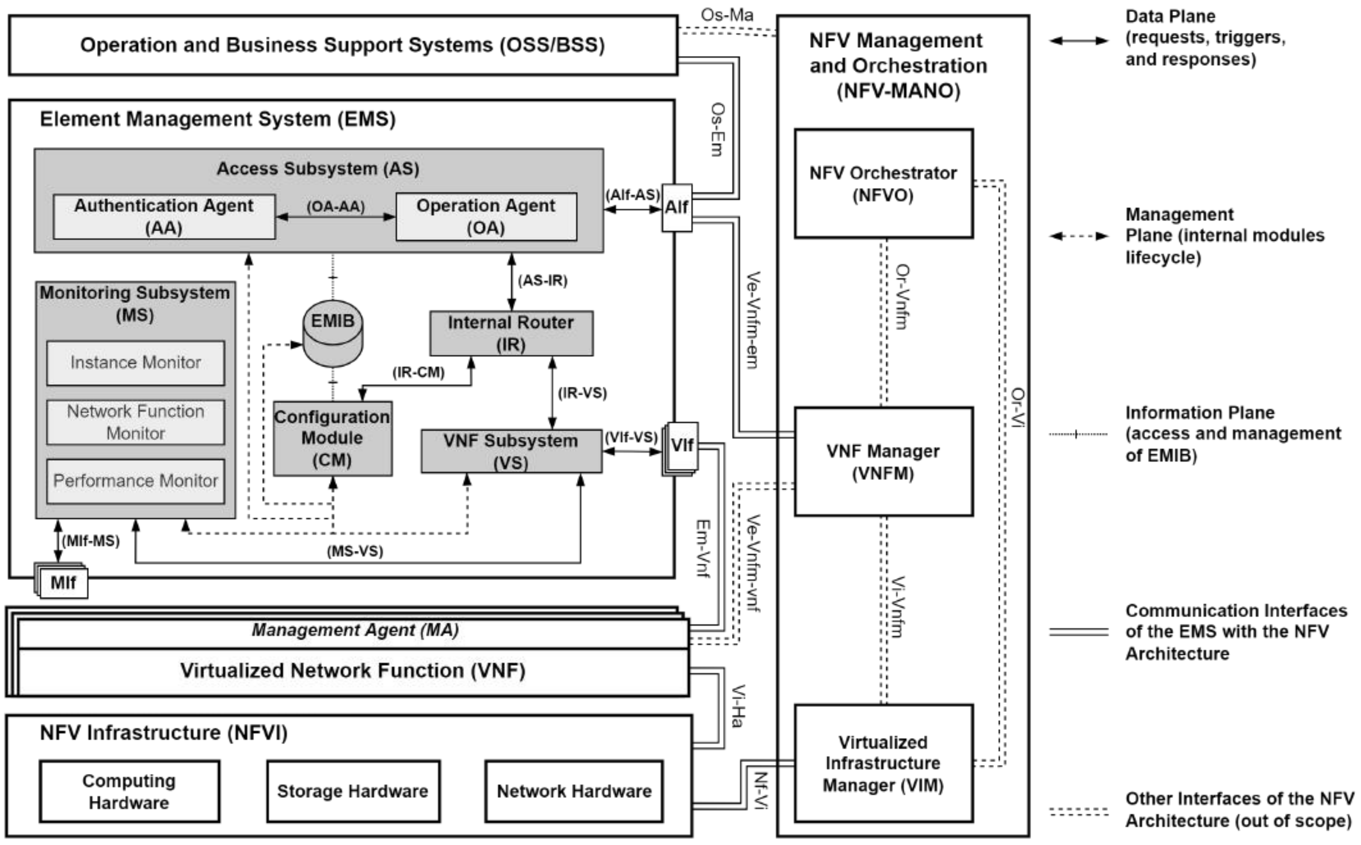
Operation and Business Support Systems (OSS/BSS)
Os-Ma
Os-Em
NFV Management and Orchestration (NFV-MANO)
NFV Orchestrator (NFVO)
Or-Vnfm
Or-Vi
VNF Manager (VNFM)
Vi-Vnfm
Virtualized Infrastructure Manager (VIM)
Element Management System (EMS)
Access Subsystem (AS)
Authentication Agent (AA)
(OA-AA)
Operation Agent (OA)
(AIf-AS)
AIf
Ve-Vnfm-em
(AS-IR)
Monitoring Subsystem (MS)
Instance Monitor
Network Function Monitor
Performance Monitor
EMIB
Internal Router (IR)
(IR-CM)
(IR-VS)
Configuration Module (CM)
VNF Subsystem (VS)
(VIf-VS)
VIf
(MIf-MS)
(MS-VS)
MIf
Em-Vnf
Ve-Vnfm-vnf
Management Agent (MA)
Virtualized Network Function (VNF)
Vi-Ha
NFV Infrastructure (NFVI)
Computing Hardware
Storage Hardware
Network Hardware
Nf-Vi
Data Plane (requests, triggers, and responses)
Management Plane (internal modules lifecycle)
Information Plane (access and management of EMIB)
Communication Interfaces of the EMS with the NFV Architecture
Other Interfaces of the NFV Architecture (out of scope)

Fig. 1. Element Management System Architecture

a particular implementation model depends on the complexity and amount of information that must be kept. Examples of information maintained by the EMIB include credentials of the EMS users; description, localization, and access models of managed VNF instances; the set of monitoring scripts and policies available; and data to support the communication with different NFV management blocks/modules (*e.g.*, VNF/MA, VNFM, and OSS/BSS).

Configuration Module (CM) – The primary objective of the CM is to configure the internal modules of EMS as it is started, according to information kept by the EMIB. Then the CM continues to run as a module responsible for controlling the EMS, which can update configurations, activate and deactivate monitoring scripts, insert or remove information about the users and managed VNF instances. The CM operations can be executed by users registered in the AS with the authorization level to execute a control task.

The general workflow of the proposed EMS can be described as follows. Initially, the AS receives a request to execute an operation from the AIf (AIf-AA) interface, and the OA proceeds with operation validation. If the operation is available and the request has the proper structure and content, it is sent by the OA to the AA (OA-AA). The AA authenticates the requester and confirms the authorization to execute the intended operation. Then, the AS forwards valid and authorized requests to the IR (AS-IR). The IR registers the request and forwards it to the VS (if the operation request is a VNF management routine - IR-VS) or to the CM (if the operation request is related to the management of the EMS itself - IR-CM). In the case of a VNF management request, the VS identifies the platform of the requested VNF and its respective virtual instance address (information provided by the AS during request validation). With this information, the request is properly modified and forwarded through the respective VIf (VS-VIf) interface. On the other hand, in the case of an EMS configuration request, the CM identifies the target module and intermediates the execution of the operation. Results follow the opposite path until they are sent back to the requester by the AS.

V. HoLMES: Holistic, Lightweight and Malleable EMS Solution

The proposed EMS architecture was implemented as an open source solution: the Holistic, Lightweight and Malleable EMS Solution (HoLMES)[1]. HoLMES includes every internal module as well as all the interfaces described in the EMS architecture, being fully compliant with the ETSI NFV reference architecture [1]. We designed HoLMES to communicate with different VNF Platforms and NFV-MANO projects. HoLMES also allows the management of multiple VNF instances si-

[1] Available at https://github.com/ViniGarcia/HoLMES

multaneously. In terms of the classification proposed in Section II, HoLMES is `MANO (standalone), external, restrict, and umbrella`. The EMS is provided as a package that can be installed to run in Linux/Windows environments and also as a pre-configured virtual machine image for Ubuntu Cloud [36].

HoLMES was implemented with the Python 3 programming language. The Access Subsystem employs the Flask library to create an HTTP Access Interface. Furthermore, the Access Subsystem natively provides all the operations specified for the Ve-Vnfm-em interface [37], which allows the VNFM to communicate with the EMS. Drivers are employed to enable the Operation Agent (Access Subsystem) to communicate with different NFV-MANO and OSS/BSS systems. These drivers include the Ve-Vnfm-em operations. HoLMES provides a template that can be used to implement arbitrary drivers (*VnfmDriverTemplate*). The Authentication Agent provides methods with distinct levels of security levels to authenticate the requests. These methods vary from *no authentication* to those that require a username and password. The Internal Router was also fully implemented and takes care of message exchanges between the internal modules. The Internal Router uses labels to identify operation requests, and the corresponding results, warnings, and errors.

The Monitoring Subsystem was implemented with the multiprocessing Python library. A monitoring script communicates with the corresponding VNF instances through the VNF Subsystem. Therefore, a script can make use of all operations available in the platform of each VNF instance. Authorized users can subscribe to receive notifications from the Monitoring Subsystem, triggered by the occurrence of predefined events. These notifications are sent through Monitoring Interfaces defined case by case. The model for the implementation of a monitoring script is available as a general template (MonitoringScriptTemplate).

The VNF Subsystem allows the EMS to communicate with the VNF Platforms. It employs drivers similar to those employed by the Operation Agent of the Access Subsystem. A driver allows the VNF Subsystem to recognize and provide the management operations for a particular VNF Platform. Thus, for each VNF Platform the corresponding driver specifies management operations and establishes a VNF Interface (VIf) with the proper communication technology, such as REST, Socket, or RMI. The template for creating the drivers is available as part of the HoLMES package (VnfDriverTemplate).

HoLMES employs an EMS Management Information Base (EMIB) based on the SQLite relational database. The EMIB keeps the information necessary for the execution and management of the EMS. Examples of such information include credentials and privileges of users, VNF Platform drivers, AS drivers, monitoring scripts, and data about managed VNF instances. The Configuration Module manages the EMS itself using EMIB information. Thus, all requests that are not directly related to operations executed on VNF instances are processed by the Configuration Agent. Management operations from the Configuration Agent are available (with a specific protocol) at the Access Subsystem/Operation Agent. These management operations are accessible to users with EMS administration privileges.

Finally, HoLMES employs three classes of messages for internal communication: (i) IRMessage, the standard message exchange model based on the IR. This class of message includes information about the source and destination of a request, and keeps track of the paths traversed internally in the EMS (it is used in the As-Ir, Ir-Cm, and Ir-Vs interfaces); (ii) CMMessage, this message class refers to CM messages (Ir-Cm). Those messages carry specific information required to execute management operations on the EMS itself; and (iii) VSMessages, message class to the VNF Subsystem (Ir-Vs), used to execute operations on VNF instances.

VI. Evaluation

In this section, we present the results of the evaluation of the proposed EMS architecture obtained from experiments executed with HoLMES. We measured the overhead to execute management operations with the addition of the EMS interfacing the communication between an OSS requesting the execution of different types of operations on multiple VNF Platforms. The OSS employed a single interface to request the EMS to execute operations on several VNF Platforms and VNFM. In order to measure the overhead, we executed the same experiments with the OSS itself implementing multiple interfaces, each dedicated to a specific VNF Platform or VNFM (without the EMS). We employed three VNF Platforms with interfaces based on two distinct technologies in the evaluation: Click-On-OSv (COO – HTTP interface) [7], Leaf (HTTP interface) [16], and COVEN (L3 Socket and HTTP interfaces) [8]. Moreover, we used the VNFM of the CloudStack Vines NFV-MANO [16]. We developed drivers for the VNF Platforms and VNFM[2].

The environment in which the experiments were executed consisted of two hosts connected to a GbE network. The first host, hereby called VNFM/VNF, is based on an Intel Core I5-3330 3.0GHz, 8GB RAM DDR3, Ubuntu 16.04, running the KVM hypervisor. This machine hosted the VNFM of the CloudStack/Vines environment and the VNF instances. The second machine, called OSS/EMS, is based on an Intel Core I3 4010-U 1.9GHz, 8GB RAM DDR3, and Ubuntu 16.04. This machine hosted the OSS and the HoLMES EMS. Each experiment was executed 150 times, and we removed 10% of the best and worst results to eliminate outliers. Thus, 120 median results were used to define the mean and standard deviation of the times to execute the management operations. The following subsections describe the experiments and results in detail, as well as a discussion of the evaluation.

A. Management of VNF Platforms

Three distinct management operations with different performance profiles were executed on the VNF Platforms: (i) `Get Status` for checking the status of the platform, a lightweight

[2] All scripts, and raw results obtained are available at https://github.com/ViniGarcia/HoLMES/tree/Experiments

operation; (ii) `Post NF` to send a network function package to the platform, a data transfer operation; and (iii) `Post Configure and Start`, an operation to configure and start a network function on the platform, an operation with intensive computing requirements. The time interval for the execution of each operation was measured from the instant the request is requested to the instance on which the corresponding response is received by the OSS.

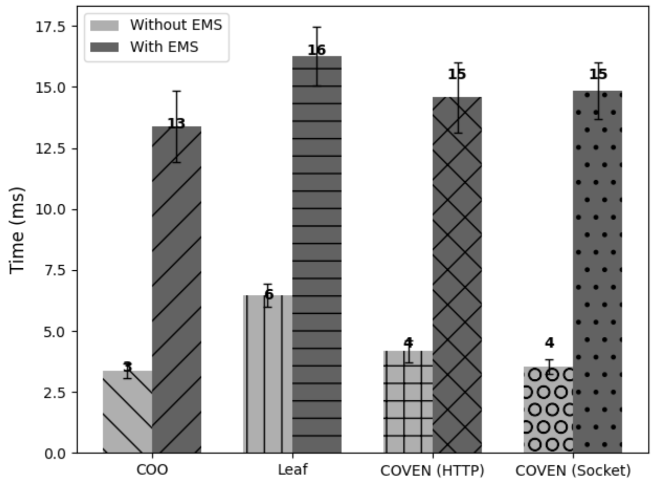

Fig. 2. Mean Execution Times (`Get Status`)

The first experiment executed the operation that returns the status of the VNF Platforms as a string: `Get Status`. This operation is lightweight once it does not involve file transfer, and requires a negligible time to be executed by the VNF Platforms. Figure 2 presents the mean execution times of `Get Status` for all experimented VNF Platforms. The overhead due to the employment of the EMS varied from 166% to 333%. The Leaf and Click-On-OSv platforms presented, respectively, the smallest and highest relative overheads in this experiment. The COVEN Platform for both interfaces (HTTP and Socket), however, produced the highest execution time overhead: 11ms. As it is a lightweight operation, most of the execution time of `Get Status` is spent to send the request and receive the response. Thus, the communication with the EMS, both as the request is sent from the OSS to the VNF Platform and the response travels back, is the cause of the overhead measured in this experiment. The differences in the execution times of the VNF Platforms are due to differences in how they execute the operation.

In the second experiment, network function packages are sent to the VNF Platforms. `Post NF` can be classified as a data transfer operation. The mean execution times of this operation are shown in Figure 3. In this experiment, the overhead of using the EMS between the OSS and the VNF Platforms varied from 158% (Leaf) to 875% (COVEN Socket). In the middle are the overheads of Click-On-OSv at 300% and COVEN HTTP at 388%. The higher overhead compared to the first experiment (*i.e.*, *Get Status*) occurs due to the fact that here files are first transferred to the EMS, which forwards them to the VNF Platforms. It is also important to notice that the network packages differ according to the VNF Platform. For Click-On-OSv, we sent a simple packet forwarder with 352 octets; for Leaf, the network function is one previously available on the platform (Apache2 server), and the NF package sent had 2797 octets; for COVEN, the network function consisted of a 4-component forwarder, and the package sent had 9287 octets. The differences of the package sizes do have an influence on the execution times and the overheads measured. The highest proportional overhead was measured for the COVEN Socket and is a consequence of using a layer 7 protocol (between the OSS and the EMS) that is absent when the EMS is not employed. Thus, a larger number of messages and extra headers are processed during the data transfer, which generates higher delays. However, we notice that even considering the worst proportional overhead scenario, the extra time did not exceed 35ms.

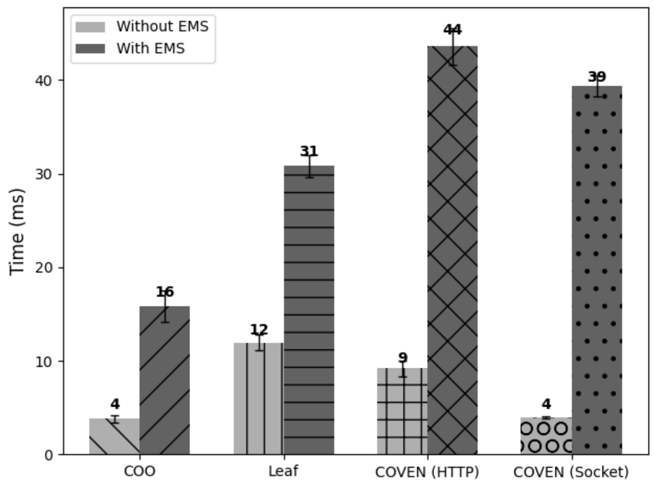

Fig. 3. Mean Execution Times (`Post NF`)

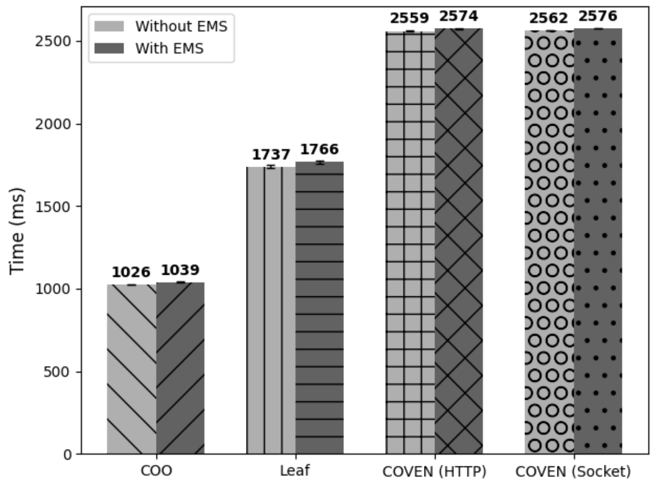

Fig. 4. Mean Execution Times (`Post Configure and Start`)

In the third experiment, the network functions previously transferred to the VNF Platforms are initialized and configured. These operations are classified as computation-intensive. Figure 4 presents the mean execution times. The overhead varied from 0.55% (COVEN Socket) to 1.67% (Leaf). Results for Click HTTP and Click-On-OSv were of 0.58% and 1.27%, respectively. This experiment presented the smallest overheads. The reason is that most of the execution is within the VNF Platforms, with minimal transmission delays. The highest overhead of this experiment was 29ms for the Leaf Platform.

The results presented above show that employing the EMS does have a cost impact in terms of the time to execute management operations, as expected. However, the EMS seamlessly allowed the integration of the different platforms, abstracting the heterogeneity of management entities and their respective technologies. This enables an OSS with a single interface to work with multiple different VNF Platforms. We note that even in the worst cases, the EMS overhead did not surpass 35ms, which can be considered a low price to pay to have holistic management.

B. Communication with VNFM

In this subsection, we measured the overhead of using the EMS in the communication between OSS and VNFM. The experiments were executed with a simple OSS and the CloudStack/Vines VNFM. We measured the overhead on the execution times of three management operations. These operations are from the Ve-Vnfm-em interface, and there are corresponding operations in the CloudStack/Vines VNFM, which does *not* implement Ve-Vnfm-em. We first measured the execution times of the Vines operations themselves, as they are requested directly by the OSS. Then we compared it with an alternative in which the EMS implements the Ve-Vnfm-em interface and receives requests from the OSS for the execution of standard operations. However, before the EMS forwards the operations to the VNFM, it has to map each requested operation to the corresponding one available at the VNFM. The overhead includes this mapping, for which HoLMES employs its Vines driver.

The CloudStack Vines management operations employed were: `List VNF Instances`, `Start VM Instance`, and `Stop VM Instance`. Those operations correspond to the following standard Ve-Vnfm-em operations, respectively: `Get vnf_instances`, `Post vnf_instances/{vnfInstanceId}/operate` (with a start flag in the arguments), and `Post vnf_instances/{vnfInstanceId}/operate` (with a stop flag in the arguments). In the experiments, we called these operations, respectively, `Get Instance` (GI), `Post Start` (PST), and `Post Stop` (PSP). The mean execution times and the standard deviation of each management operation are shown in Figure 5.

The GI operation returns information about all the VNF instances managed by the VNFM. In order to execute this operation, the VNFM interacts with the VIM, retrieving, in addition to the identifiers of the managed VNF instances, a set of configuration information for the respective virtual machines. GI can be considered a lightweight operation. However, the VNFM does need to communicate multiple times with the VIM. The overhead of using the EMS was 32%, which corresponds to 6ms.

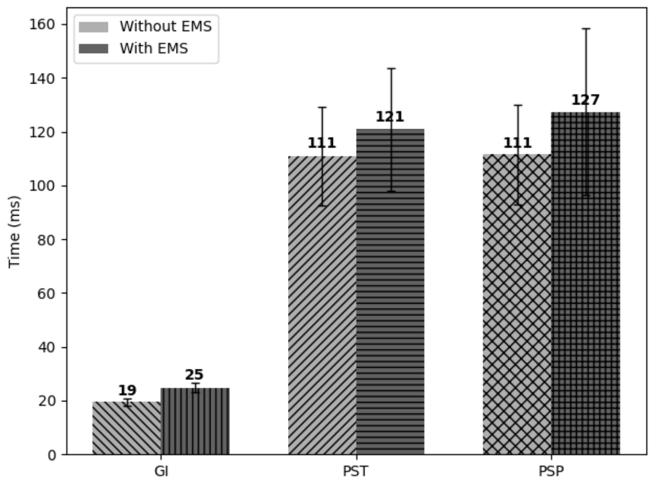

Fig. 5. Mean Execution Times (`VNFM Operations`)

The PST and PSP operations, in turn, request the VNFM to start up and stop a VM, respectively. This causes the VNFM to communicate with the VIM. These operations do not wait for the termination of the VIM processes, but only for confirmation that they have been received and accepted. Thus, despite being intensive in terms of computation, the operations present a low execution time compared to other compute-intensive operations (such as configuring and starting a network function in a VNF platform, for example). Therefore, employing the EMS to interface the operation request with the VNFM adds an overhead of 9% and 14% on the execution of PST and PSP, respectively. These overheads correspond to 10 (PST) and 16 (PSP) milliseconds.

The overhead of using the EMS in the communication of an OSS with the CloudStack/Vines VNFM was 32% in the worst case, which corresponds to 16ms.

Note that the time overhead in absolute terms typically varies considering the characteristics of the operation, the amount of processing done by the EMS, and the extra transmission time for relaying the request through the EMS. Thus, as in the two experiments, the requests are similar, the EMS does little processing, and the execution environment is the same, the absolute overheads are quite similar for operations with similar characteristics. Examples are the maximum time overhead for the operations that can be considered lightweight: 11ms for `Get Status` (VNF) and 6ms for `Get Instance` (VNFM); and also for operations that are computationally intensive: 29ms for `Post Configure and Start` (VNF) and 10-16ms for `Post Start` and `Post Stop` (VNFM). However, the differences in terms of percentage are greater, as these involve the total time required to perform the requested operation on a VNF instance or the VNFM. Thus, the overhead of using the EMS to execute Get Status corresponds a 333% increase which is about 10 times greater than the corresponding percentage for Get Instances (32%). Similarly, the overhead of Post Configure and Start is nearly 10 times smaller than the corresponding percentage for the Post Start and Post Stop operations.

VII. Conclusion

Although the EMS is defined by ETSI NFV reference architecture as the element responsible for VNF instrumentation, only a few limited EMS solutions are currently available. The main contribution of this work is to propose an architecture for the EMS that consists of loosely coupled internal modules yet exposes interfaces that are fully ETSI-compliant. The proposed architecture leverages the EMS to enable true holistic NFV management, as it provides the glue for the interaction of heterogeneous blocks (VNF, VNFM, and OSS/BSS) from different platforms/vendors. Thus, the proposed architecture supports the integration and portability requirements from the ETSI. The proposed EMS architecture was implemented as the open-source HoLMES system, which was employed to evaluate the overhead of using an EMS to execute different types of management operations on multiple VNF Platforms. Experiments confirm that the proposed EMS can be used effectively while abstracting particular protocols and technologies. Future work includes extending HoLMES to allow the native integration of more VNFM and VNF platforms and the automatic generation of VNF monitoring scripts based on policies. Furthermore, we aim to explore the IETF and 3GPP models, standards, and protocols regarding EMS in the context of the proposed architecture.

References

[1] E. N. ISG, "Network functions virtualization (nfv): Architectural framework," ETSI, Tech. Rep., 2014.

[2] J. M. Halpern and C. Pignataro, "Service function chaining (sfc) architecture," IETF, Tech. Rep. RFC 7665, 2015.

[3] V. Fulber-Garcia, A. Huff, C. R. P. dos Santos, and P. Duarte Jr, E, "Network service topology: Formalization, taxonomy and the custom specification model," *Computer Networks*, vol. 178, p. 107337, 2020.

[4] V. Fulber-Garcia, L. d. C. Marcuzzo *et al.*, "An nsh-enabled architecture for virtualized network function platforms," in *International Conference on Advanced Information Networking and Applications*, 2019, pp. 376–387.

[5] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu *et al.*, "Clickos and the art of network function virtualization," in *Conference on Networked Systems Design and Implementation*, 2014, pp. 459–473.

[6] W. Zhang, G. Liu, W. Zhang *et al.*, "Opennetvm: A platform for high performance network service chains," in *Hot topics in Middleboxes and Network Function Virtualization*. ACM, 2016, pp. 26–31.

[7] L. Marcuzzo, V. Fulber-Garcia *et al.*, "Click-on-osv: A platform for running click-based middleboxes," in *International Symposium on Integrated Network Management*, 2017, pp. 885–886.

[8] V. Fulber-Garcia, L. d. C. Marcuzzo *et al.*, "On the design of a flexible architecture for virtualized network function platforms," in *Global Communications Conference*, 2019, pp. 1–6.

[9] L. Bondan, M. F. Franco, L. Marcuzzo, G. Venancio, R. L. Santos, R. J. Pfitscher, E. J. Scheid, B. Stiller, F. De Turck, E. P. Duarte *et al.*, "Fende: marketplace-based distribution, execution, and life cycle management of vnfs," *Communications Magazine*, vol. 57, no. 1, pp. 13–19, 2019.

[10] T. N. Tavares, L. d. C. Marcuzzo, V. Fulber-Garcia, G. V. de Souza, M. F. Franco, L. Bondan, F. De Turck, L. Z. Granville, E. P. Duarte Jr., C. R. P. dos Santos *et al.*, "Niep: Nfv infrastructure emulation platform," in *International Conference on Advanced Information Networking and Applications*. IEEE, 2018, pp. 173–180.

[11] B. Chatras, "On the standardization of nfv management and orchestration apis," *Communications Standards Magazine*, vol. 2, no. 4, pp. 66–71, 2018.

[12] G. Venâncio, V. Fulber-Garcia *et al.*, "Beyond vnfm: Filling the gaps of the etsi vnf manager to fully support vnf life cycle operations," *International Journal of Network Management*, vol. 31, no. 5, p. e2068, 2021.

[13] S. Peng, J. O. Fajardo *et al.*, "Qoe-oriented mobile edge service management leveraging sdn and nfv," *Mobile Information Systems*, 2017.

[14] B. Jaeger, "Security orchestrator: Introducing a security orchestrator in the context of the etsi nfv reference architecture," in *Conference on Trust, Security and Privacy in Computing and Communications*, 2015, pp. 1255–1260.

[15] OpenStack Foundation, "Tacker: Openstack nfv orchestration," 2023. [Online]. Available: https://wiki.openstack.org/wiki/Tacker

[16] J. W. V. Flauzino, "Vines: Holistic management of virtual network functions," 2021, https://www.inf.ufpr.br/jwvflauzino/vines/.

[17] E. N. IFA, "Network functions virtualisation (nfv): Management and orchestration; ve-vnfm reference point; interface and information model specification," ETSI, Tech. Rep., 2020.

[18] H. Hawilo, A. Shami *et al.*, "Nfv: State of the art, challenges, and implementation in next generation mobile networks (vepc)," *Network*, vol. 28, no. 6, pp. 18–26, 2014.

[19] G. Carella, M. Corici *et al.*, "Cloudified ip multimedia subsystem (ims) for network function virtualization (nfv)-based architectures," in *Symposium on Computers and Communications*, 2014, pp. 1–6.

[20] G. P. Sharma, W. Tavernier *et al.*, "Dynamic hardware-acceleration of vnfs in nfv environments," in *International Conference on Software Defined Systems*, 2019, pp. 254–259.

[21] E. N. ISG, "Network functions virtualisation (nfv): Virtualisation requirements," 2013.

[22] A. Huff, G. Venancio, L. d. C. Marcuzzo, V. Fulber-Garcia, C. R. P. dos Santos, and E. P. Duarte, "A holistic approach to define service chains using click-on-osv on different nfv platforms," in *Global Communications Conference (GLOBECOM)*. IEEE, 2018, pp. 1–6.

[23] I. T. S. Sector, "Principles for a telecommunications management network," ITU, Tech. Rep., 2000.

[24] G. T. S. G. Services and S. Aspects, "Management and orchestration; architecture framework," 3GPP, Tech. Rep., 2018.

[25] E. N. MAN, "Network functions virtualisation (nfv): Management and orchestration," ETSI, Tech. Rep., 2014.

[26] J. G. Herrera and J. F. Botero, "Resource allocation in nfv: A comprehensive survey," *Transactions on Network and Service Management*, vol. 13, no. 3, pp. 518–532, 2016.

[27] E. N. REL, "Network functions virtualisation (nfv): Resiliency requirements," 2015.

[28] E. N. IFA, "Network functions virtualisation (nfv): Report on architectural options," ETSI, Tech. Rep., 2016.

[29] ——, "Network functions virtualisation (nfv): Report on management of nfv-mano and automated deployment of em and other oss functions," ETSI, Tech. Rep., 2018.

[30] E. T. S. Institute, "Osm: Open source mano," 2023. [Online]. Available: https://osm.etsi.org

[31] OpenBaton Project, "Open baton: An extensible and customizable NFV MANO-compliant framework," 2023. [Online]. Available: https://openbaton.github.io/

[32] A. Huff, G. Venancio, V. Fulber-Garcia, and E. P. Duarte, "Building multi-domain service function chains based on multiple nfv orchestrators," in *Conference on Network Function Virtualization and Software Defined Networks*. IEEE, 2020, pp. 19–24.

[33] E. N. IFA, "Network functions virtualisation (nfv): Vnf descriptor and packaging specification," ETSI, Tech. Rep., 2021.

[34] E. N. SEC, "Network functions virtualisation (nfv); report on certificate management," 2019.

[35] ——, "Network functions virtualisation (nfv); access token specification for api access," 2020.

[36] Canonical, "Ubuntu cloud operating system," 2021, https://cloud-images.ubuntu.com.

[37] E. N. SOL, "Network functions virtualisation (nfv): Protocols and data models; restful protocols specification for the ve-vnfm reference point," ETSI, Tech. Rep., 2020.