LightMANO: Converging NFV and SDN at the Edges of the Network

Roberto Riggio[†], Shah Nawaz Khan[†], Tejas Subramanya[†], Imen Grida Ben Yahia[§], Diego Lopez[‡]
[†]FBK CREATE-NET, Trento, Italy; Email: {rriggio,s.khan,t.subramanya}@fbk.eu
[‡]Orange Labs, Paris, France; Email: imen.gridabenyahia@orange.com
[‡]Telefonica, Madrid, Spain; Email: diego.r.lopez@telefonica.com

***Abstract*—Network Function Virtualization (NFV) has raised tremendous attention in the academic and industrial communities alike. The former have been attracted by the service centric orchestration challenges. The latter have found that, by decoupling network functions from the underling hardware, significant savings can be obtained by means of infrastructure homogenization and service automation. As opposed to the initial NFV solutions, which heavily relied on a centralized cloud model, the emerging Multi–access Edge Computing trend is calling for a distribution of computational capacity at the customers' sites. This change of paradigm could result in the deployment of tens of thousands or even millions of Points–of–Presence. In this article we discuss the fundamental challenges of deploying NFV in scattered environments, then we introduce the *LightMANO* framework, a Multi–access Network Operating System converging SDN and NFV into a single lightweight platform for management and orchestration of network services over distributed NFV infrastructure. Finally, we report on a proof–of–concept implementation of *LightMANO* and on its evaluation.**

***Index Terms*—Network programmability, 5G, NFV, SDN, Multi–access Edge Computing, LTE, Wi–Fi**

I. Introduction

Deploying new services takes a network provider a significant amount of time. This is due to logistical and technical challenges like delivering new (typically) dedicated hardware boxes to the customer premises, configuring them, and performing troubleshooting. Network Function Virtualization (NFV) promises to simplify the network service creation workflow by turning traditional network functions, e.g. firewall, into software and by hosting them on general purpose servers. Combined with the recent advances in cloud computing, NFV can enable fast service provisioning, scaling, and migration.

NFV has received significant attention from both academia and industry with the former group focusing on the fundamental challenges of service provisioning such as workload placement, resiliency, and multi–layer resource optimization, and the latter focusing on scalable and deployable orchestration stacks. In the standardization arena the most notable efforts are spearheaded by the European Telecommunications Standards Institute (ETSI) within the NFV Industry Specification Group[1]. The group defined the ETSI Management and Orchestration (MANO) framework which is nowadays taken as reference NFV architecture by several open–source and commercial NFV platforms [1].

Nevertheless, despite the fact that the emerging 5G and Industrial IoT scenarios call for more flexibility at the edges of the network, i.e. the part of the infrastructure located closest to the end–users, there are still a number of open challenges that need to be addressed before NFV can be effectively deployed in such environments. This includes distributed management and orchestration of network services, lightweight virtualization of computing resources, security, support for highly heterogeneous access technologies, multi–tenancy, and advanced interplay between SDN and NFV.

The goal of this paper is twofold. First, we analyse the challenges and requirements of a MANO framework suitable to be deployed in massively distributed environments which can be composed of millions of small sites and where the limited computing resource would make a full MANO stack not a viable option. Second, we introduce *LightMANO* a Multi–access Network Operating System converging SDN and NFV into a single lightweight platform for management and orchestration of network services over distributed NFV infrastructure. *LightMANO* builds upon lightweight virtualization technologies such as Containers for computing nodes (i.e. Docker [2]) and programmable packet processing pipelines for networking nodes (i.e Click [3]) and can interface with state–of–the–art radio access and backhaul network controllers. Finally, we present a proof–of–concept implementation of *LightMANO* and we release the entire *LightMANO* stack under a permissive APACHE 2.0 licence for academic purposes[2].

The rest of the paper is structured as follows. In Sec. II we introduce the challenges and requirements for a distributed MANO framework. Section III describes the *LightMANO* architecture. The implementation details are provided in Sec. IV. Section V reports on the prototype evaluation in a mobile–edge caching scenario. Finally, Sec. VI concludes the paper.

II. Challenges and Requirements

In this section we will first compare the distributed and centralized NFV deployment models. Then we will analyse the open challenges for NFV in scattered and massively distributed environments. For a more general discussion on NFV challenges and requirements we refer the reader to [4].

A. Distributed Vs. Centralized

The ETSI MANO reference model defines a set of logical components and their interfaces. The framework consists of

Research leading to these results received funding from the European Unions H2020 Research and Innovation Action under Grant Agreement H2020-ICT-644843 (5G–ESSENCE Project).

[1]http://www.etsi.org/technologies-clusters/technologies/nfv

978-1-5386-3416-5/18/$31.00

[2]Online resources available at: http://lightmano.create-net.org/

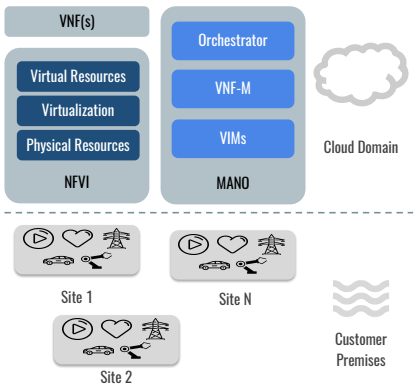
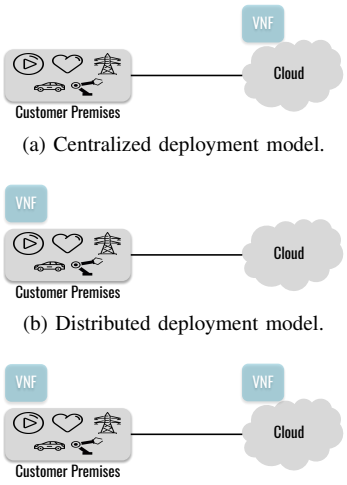


Fig. 1: The ETSI MANO reference architecture in a C–NFV scenario.



(a) Centralized deployment model.

(b) Distributed deployment model.

(c) Hybrid deployment model.

Fig. 2: NFV Deployment models.

two main stacks: the NFV infrastructure (NFVI) stack and the management and orchestration (MANO) stacks. The NFVI stack comprises both the physical and virtualized resources required to host the Virtual Network Functions (VNFs). Conversely, the MANO stack hosts the service and function orchestration and management logic as well as the virtual infrastructure managers in charge of controlling the physical and virtual resources available in the NFVI Point–of–presence (PoP). The reference ETSI MANO model is sketched in Fig. 1.

The scenario depicted above does not specify how the actual NFV platform must be implemented, nevertheless the flexible service provisioning requirements and the heavy reliance on virtualization led many implementers to reuse software stacks and best practises found in the data–center and cloud computing domains. This despite the fact that, being software instances, VNFs can essentially run on any kind of platform ranging from Raspberry PIs to multi–core servers.

Figure 2 depicts three NFV deployment models. In the centralized NFV (C–NFV) deployment model all the VNFs are deployed in large data–centers. In the distributed NFV (D–NFV) deployment model, VNFs runs on lightweight computing platform deployed at the customer premises. Finally, in the hybrid model VNFs are deployed both at the edge of the network and in the centralized cloud. The focus of this paper will be on the distributed deployment model.

The C–NFV architecture is particular suitable for cloud–based NFV deployments where computing resources are abundant. For example, an OpenStack controller node, a typical solution used as VIM, can easily require at least 2 CPU cores and 8 GB of RAM while compute nodes requirements are even higher. Popular Orchestrator and VNF Manager solutions, e.g. OpenBaton [5], have similar requirements.

The D–NFV model is currently receiving significant attention in that it is seen as one of the main technological enabler for the Multi–access Edge Computing (MEC) paradigm. MEC advocates for distribution of computational and storage resources very close to the actual end–users, possibly within the access network itself. This approach is particularly important in order to meet the bandwidth and latency requirements expected by the fifth generation of the mobile network architecture. The D–NFV deployment model can potentially deliver several advantages compared to the centralized model, this includes improved security and resiliency, computational locality, better performances (especially latency), and enhanced privacy. Nevertheless, while the first steps toward D–NFV are being made, several technical and scientific challenges must still be overcome before the expected gains can be obtained.

Among the various challenges associated with D–NFV significant attention is currently addressed toward lightweight virtualization *and* orchestration solutions that can be executed in resource–constrained and/or geographical challenging locations (e.g. oil plants, boats, airplanes). The problems to be addressed are not limited to one particular aspect of the NFV stack, on the contrary a full–stack purpose–built solution for D–NFV is needed. This includes lightweight compute and network virtualization solutions as well as lightweight MANO solutions. The latter may also collapse VIMs, VNF Managers, and Orchestrator into a single component.

B. Management and Orchestration

Network service on–boarding, monitoring, and scaling are some of the main NFV management and orchestration features. Among those, an important role is played by the placement of the VNFs over the NFVI. This problem is known as Virtual Network Embedding and has been already addressed for the single domain [6] and for the multi–domain cases [7], [8], [9], [10], [11]. Moreover, recent works can be found in literature investigating different architectural approaches for cross–domain orchestration [12], [13], [14], [15].

Nevertheless, if at the time of writing NFVI Infrastructures composed of a few hundreds of PoPs each of them with tens of CPU cores are common, in the near future we can imagine to have tens of thousands or even millions of such PoPs each of them with a single core (e.g., a residential CPE), multiple cores (e.g., an Enterprise CPE), or significant computation capabilities (e.g., a central office). This scenario calls for a new approach to NFV Management and Orchestration.

First, if in a C–NFV deployment all the MANO components are co–located and share the same LAN, in a D–NFV deployment also the WAN characteristics must be taken into

account. This includes intermittent or lack of connectivity (e.g. in maritime oil plants, boats, or air planes) as well as bandwidth and/or latency limitations. Moreover, in some situation it may be not practical or economically viable to upgrade the WAN while in other situations, like for industrial IoT deployments, relaying on an external WAN may not be possible due to security constraints.

Second, being designed with a cloud computing mentality, current ETSI MANO implementations assume that each VNF is hosted by a dedicated VM or possibly even multiple VMs. Such an approach would not scale in the D–NFV case where some functions may need to reside in computational constrained devices. Notice how, the limited computing power may not be necessary due to economic reasons, on the contrary it may be linked to cooling or space requirements.

Third, current ETSI MANO frameworks typically focus on centralized or carrier grade multi–domain deployments with strong SLA guarantees between domains. In a D–NFV setting the communications latency introduced by the WAN can delay the reaction to faults limiting the network reconfiguration and optimization possibilities (e.g. due to the lag incurred in collecting the measurements from the NFVI PoPs). D–NFV calls for moving part of the orchestration to the network edges.

C. Interplay between SDN and NFV

Cloud computing platforms such as OpenStack [16] have not been designed with NFV management and orchestration in mind and, as a result, their networking API is limited to rather simplistic VLAN–based models while the powerful virtualization and abstractions primitives available in state–of–the–art Software–Defined Networking (SDN) platforms are not yet exposed to the service orchestrator.

For example, VNFs deployed using VMs only allow cloning as migration method. This results in a significant waste of memory/networking resources in that unneeded state has to be migrated alongside the actual VNF [17]. Same considerations apply (to some extend) to solutions based on Docker. The result is that relatively simple requirements are imposed on the underlying networking fabric to support service migration and scaling. Conversely, approaches like OpenNF [18] and its derivatives [19], [20] focus on providing a platform for consistent VNF migration. Similar considerations can be made for Split/Merge [21], CoMB [22], and XoMB [23]. Nevertheless, such platforms cover only a fraction of the ETSI MANO stack.

D–NFV systems will require more complex interactions between SDN and NFV. In fact, while the focus of SDN has been on enabling programmatic access to wired [24], [25], [26], [27], [28], [29], [30], [31] and wireless networks [32], [33], most of these approaches do not provide mechanisms for managing and orchestrating VNFs. A convergence of SDN and NFV is expected in order to allow service providers to specify the logical sequence of VNFs a precise portion of the flowspace must traverse. Intent–based networking is expected to play a key role in this context by allowing network service providers to use a high–level declarative languages for service function chaining [34], [35].

D. Handling Heterogeneity

Current MANO frameworks are not designed to mix physical and virtual network functions, nor to interface with legacy systems. Nevertheless, by being also required to operate at the network edges and possibly within the customer premises, D–NFV systems will also have to cope with heterogeneous access network technologies ranging from Ethernet to the various wireless and mobile networks standards, such as Wi–Fi, LoRA, LTE, and the future 5G New Radio, and will be expected to support a rich set of management protocols, such as NETCONF, SNMP, and CLI.

In the computing domain this is the norm since decades. Operating systems and applications can be compiled for different target architectures. The Linux kernel, for example, can run on embedded platforms with a single core and few MBs of RAM as well as on large clusters made of thousands of cores and TBs of RAM. Similarly, a distributed MANO framework will have to cope with different network architectures and protocols. In this context being able to *compile* network management and orchestration directives for different *targets* is of capital importance. Semantic models are expected to play a key role in enabling automatic translation between high–level policies to imperative network configuration commands.

E. Software Engineering and DevOps

NFV is already allowing Telcos to deploy complex services across virtualized computing, storage, and networking resources. This, in time, is slowly changing the mentality of Telcos which are transitioning from *configuring* network services to *programming* network services. As a matter of fact, ETSI MANO frameworks already rely on declarative languages and on highly automated system operations especially to react to events that can happen at runtime. This essentially leads to the so–called DevOps which brings together Development and Operations creating a tight creation, testing, deployment and operation lifecycle.

Nevertheless, if the DevOps approach is percolating into the Telcos' organization, significant steps still need to be taken in order to take full advantage of a D–NFV environment. In the early days of virtualization, VMs have been used to provide isolation between different web applications and services. Subsequently, this proved not scalable and the micro–service model emerged with a specific application decomposed in its elementary building blocks, e.g. data–base, web front–end, load–balancer and so on.

The situation with NFV is similar. In fact, while so far we are using VNFs to replace particular network functions, we should instead embrace the micro–services model and decompose network functions into their elementary components and deploy/scale them individually. For example, a video transcoding network function does not need a full IP stack on the contrary it needs some well defined interfaces to the content server and a way to operate on the video stream. This, besides resulting in a more lightweight implementation, is also intrinsically more secure since lacking a full IP stack makes the VNF immune to a broad range of attacks.

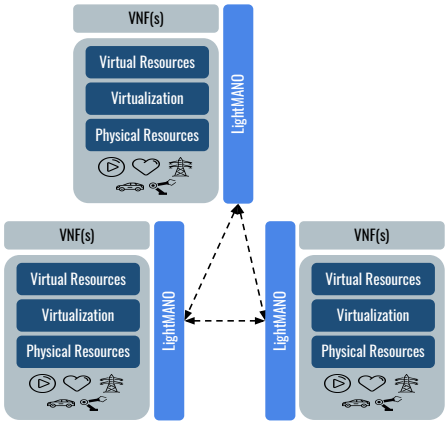


Fig. 3: The *LightMANO* reference architecture in a D–NFV scenario.

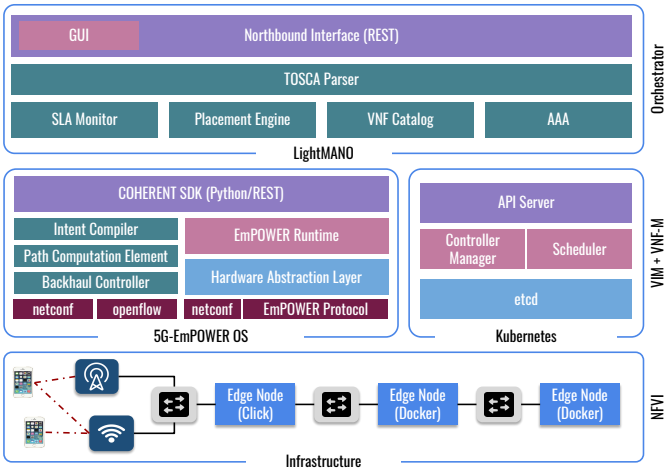


Fig. 4: The *LightMANO* System Architecture.

III. *LightMANO*

In this section we will describe the *LightMANO* system architecture, components, and interfaces.

A. Overview

The massive number of NFVI PoPs expected present in a D–NFV deployment each of them possibly running tens or even hundreds of VNFs will simply not fit the capabilities of most ETSI MANO implementations. A purpose–built solution is needed in order to address the challenges related to security, scalability, and performance that are raised by D–NFV. Figure 3 sketches the *LightMANO* network architecture. As it can be seen, each site includes the virtualizaed computing and networking resources as well as part of the MANO stack. The *LightMANO* component embeds a lightweight VIM (described in the next section) and part of the orchestration logic. Moreover, VM–based virtualization solution are replaced with lightweight alternatives such as Docker and Click. Communication between different *LightMANO*s can be envisioned to allow for multi–site service deployment.

Figure 4 depicts the *LightMANO* system architecture. As it can be seen it consists of three layers: the infrastructure layer, the management layer, and the orchestration layer. The infrastructure layer hosts the physical computing and networking resources. The management layer essentially covers the functionalities of the VIM and of the VNF–Manager in the ETSI MANO reference model. Finally, the Orchestration layer embeds the service management and orchestration logic.

The management layer consists of two main components: *5G–EmPOWER* and Kubernetes. *5G–EmPOWER* [32] is a Multi–access Edge Computing Operating System (MEC–OS) which consolidates SDN and NFV into a single platform supporting lightweight virtualization and heterogeneous radio access technologies[3]. Kubernetes [36] is used as platform for managing containerized applications. It supports a wide range of container tools, including Docker. Within *LightMANO* we use Kubernetes as framework for automating the deployment, scaling, and operations of containerized applications and service. Nevertheless, it is worth noticing that *LightMANO* does not depend on Kubernetes and can be easily ported to similar solutions such as Docker Swarm [2].

[3]Online resources available at: http://empower.create-net.org/

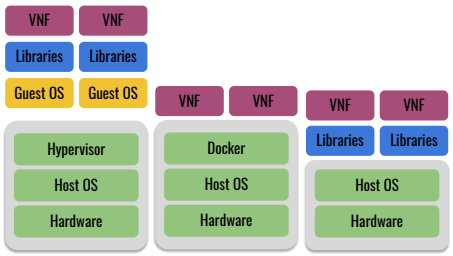
B. Lightweight Virtualization

LightMANO builds on Docker and Click as lightweight computing and networking virtualization solutions.

Containerization is lightweight OS virtualization solutions that allows to isolate (i.e. containerise) certain process and resources from the host operating system. As opposed to VM–based virtualization, containers *share* the same kernel and may also share some of the system libraries. Containers are fast to deploy (few seconds) and have a small footprint which in time allow easy consolidation of several VNFs on the same host. The main drawbacks are in terms of reduced isolation and security (compared to standard VM–based solutions). Docker uses standard resource isolation features available in the Linux Kernel, such as namespaces, in order to allow certain applications to run isolated from each other within the so–called *containers*. A daemon, executed by Docker–enabled hosts, is in charge of communicating with the host kernel to create, operate, and manage containers.

Recently a new trend has emerged for implementing data–plane processing functionalities. Known as kernel bypass filters, such solutions essentially implement a specialized API to directly read/write packets from/to the Ethernet interface. Examples includes DPDK [37], PF_RING [38], and Snabb-switch [39]. The main drawback of these libraries is that they are very low–level. However, when combined with programmable packet processing pipelines like The Click Modular Router, kernel bypass filters provide the ideal platform for implementing extremely lightweight VNFs (few MB) which are very fast to boot (few milliseconds).

The Click Modular Router allows implementing arbitrary programmable packet processing pipelines by combining different *Elements*. Developers can either use the *Elements* bundled with Click or they can develop their own in C++. An instance of Click has an extremely low memory footprint (approx 6MB for a pure packet forwarding VNF) and can to boot very quickly (less than 100ms).



(a) Virtual Machines. (b) Containers. (c) Click.

Fig. 5: A comparison between different computing virtualization solution in term of additional overhead.

C. Multi–access Edge Operating System

The 5G–EmPOWER MEC–OS consists of a hardware abstraction layer converging several radio access networks control and management protocols into a unified set of abstractions that are then exposed to the *LightMANO* orchestration layer. Such abstractions allow the *LightMANO* orchestration layer to implement joint NFV and SDN resource management operations. This includes, for example, joint mobility management and VNF placement/migration schemes as well as radio access and backhaul load balancing. 5G–EmPOWER currently support Wi–Fi and LTE radio access technologies while support for LoRA and other Low–Power Wide–Area Networks is currently been added. Interaction with SDN–based backhauls is enabled trough an Intent–based networking interface [34]. In the rest of this section we will provide a short summary of the main abstractions supported by 5G–EmPOWER. A more extensive description can be found in [32].

1) Light Virtual Access Point (LVAP): The *LVAP* abstraction [33] provides a high–level interface for wireless clients state management. The implementation of such an interface handles all the technology–dependent details such as association, authentication, handover, and resource management. A client attempting to join the network, will trigger the creation of a per–client virtual access point (the *LVAP*) which becomes a potential candidate AP for the client to perform an association. Similarly each AP will host as many *LVAP*s as the number of wireless clients that are currently under its control. Removing a *LVAP* from an AP and instantiating it on another AP effectively results in a handover.

2) Light Virtual Network Function (LVNF): The *LVNF* is a generalization of the *LVAP* abstraction. However, unlike the *LVAP*, the *LVNF* abstraction allows arbitrary packet processing blocks to be exposed to the developer trough an high–level declarative interface. *LVNF*s are instantiated starting from templates called *Image*s. Each network function corresponds to an *Image* and consists of a Click script together with some additional information such as the number of input/output ports, and the list of Click handlers[4] exposed by the *Image*. Handlers are used in order to manipulate the internal state of the *LVNF*. For example, in the case of a Firewall *LVNF*, specific handlers shall be defined in order to allow the network operator to add/remove firewall rules.

3) Network Graph: The Network Graph provides network programmers with a full view of the network state. The network graph is exposed as a directed graph $G = (V, E)$ where V is the set of clients and radio access network elements (i.e. the Wi–Fi APs) and E is the set of edges or links. A weight $\omega_e(e_{n,m})$ is assigned to each link $e_{n,m} \in E$: $\omega(e_{n,m}) \in \mathbb{R}$. Another weight $\omega_v(n)$ is assigned to each node $n \in N$: $\omega_v(n) \in \mathbb{R}$. The weights assigned to nodes and links can model different aspects of the links (e.g. the RSSI in Wi–Fi networks or the RSRP/RSRQ in LTE networks) or of the nodes (e.g. the channel utilization in Wi–Fi networks or the PRB utilization in LTE networks).

4) Transmission Policy: The *Transmission Policy* specifies the range of parameters that radio access network elements can use for their communication with the wireless clients. For example, in the case of a Wi–Fi network such parameters include: the set of valid MCSes, the RTS/CTS threshold, the ACK policy, and the multicast policy.

*5) Virtual Port: LVNF*s do not define which kind of traffic they should process. The *Virtual Port* abstraction allows developers to define the logical sequence of *LVNF*s a certain portion of the flow–space must traverse. Each *Virtual Port*, is associated with one, and only one, *LVAP* or *LVNF* and is mapped to a physical network interface (e.g. a port of an OpenFlow switch). Developers need not to know the physical port id, instead, they can perform *LVNF* chaining by connecting their *Virtual Port*s. Each mapping between two *Virtual Port*s can be defined over a precise portion of the flowspace specified as an OpenFlow match rule. The translation from *Virtual Port*s into forwarding instructions in performed by the Intent Compiler and by the Path Computation Engine. More information about this aspect can be found in [34].

Both *LVAP* and *LVNF* are implemented using the Click modular router. An agent executed by *LVAP/LVNF*–enabled hosts is in charge of spawning and managing *LVAPs/LVNFs* under the control of *5G–EmPOWER*. Communication between *LVAP/LVNF*–enabled hosts and *5G–EmPOWER* happens over a persistent TCP connection using a custom–design protocol.

D. Distributed Network Service Orchestration

Edge services can be volatile requiring fast provisioning and decommissioning. Deploying part of the orchestration logic at the network edges can help in this direction by providing faster reaction to changing network conditions while at the same time ensuring higher resiliency and security. The *LightMANO* orchestration layer is in charge of exposing the resources available at a given site. Its design is inspired by the ETSI MANO architecture and as such it support basic VNF and network service life–cycle management operations (creation, chaining, and deletion) as well as basic monitoring capabilities over the instantiated VNFs.

Site resources are advertised using TOSCA–based descriptors over the *LightMANO* REST interface. Such descriptors provide the identifiers, the features, and the monitoring capabilities of the VNFs available at each site. Users can compose network services either by using the embedded web–based management dashboard or trough any other client capable of

[4]Handlers are access points through which users can interact with elements in a running Click router or with the router as a whole.

consuming a REST interface. The VNF Placement Engine and the SLA manager are in charge of, respectively, on–boarding new network services and ensuring that their requested KPIs are enforced throughout the entire network service lifetime. In is worth noticing that, in *LightMANO* a network service definition encompasses the following aspects:

- *Radio access network slicing.* For example in the case of a Wi–Fi radio access network, users can request the creation of a new virtual hotspot with its own SSID (the network name) and authentication parameters. User can also specify performance (resources allocated to the slice, slice priority) and functional (slice lifetime, maximum number of users) parameters.
- *Service composition.* The user can select the network functions that should be deployed and the way they should be interconnected (i.e. the VNF forwarding graph). Operational parameters as well as interfaces to external monitoring and/or management entities can also be specified during this step. Notice how the details of deploying an VNF as an LVNF or as Docker Container are hidden away by the VNF placement engine that dynamically selects the network function hosting node according to the node capabilities and available resources.
- *Life–cycle management.* Each network service can embed one or more Python scripts which allow users to customize the way network services are deployed and managed. Such scripts can implement arbitrary operations on the entire network service. For example they can implement a traffic aware mobility management application for a Wi–Fi network which can migrate and scale a security or a content distribution network service according to wireless users mobility events.

Finally, access to a site resource is allowed trough a AAA (Authentication, Authorization, and Accounting) system. The granularity at which the site information are exposed trough by the *LightMANO* northbound interface can be defined on a per–user basis. This can allow to realize a multi–layer, multi–domain, network service orchestration architecture where resources are pooled across cloud, transport, and edge segments.

IV. Implementation Details

Figure 6 depicts a sample *LightMANO* deployment. Two types of nodes can be identified: RAN Nodes and Edge Nodes. Both types of node run a Docker daemon and are part of a Kubernetes cluster.

RAN nodes implement wireless access functions. In particular LTE RAN nodes consists of Intel NUC boxes (dual–core Intel i7 Kaby–Lake CPU) equipped with 16GB of RAM and 256GB of storage. These boxes run Ubuntu 17.10 and are used for deploying the LTE baseband units containers. In the current implementation we support srsLTE [40] as open LTE stack implementation. Conversely, Wi–Fi RAN nodes are based on PCEngines APU2 boxes (quad–core AMD Jaguar CPU) equipped with 4GB of RAM and 64GB of storage. These boxes run LEDE 17.01 and are used to deploy the Wi–Fi data–path. Notice how RAN nodes require specialized hardware in order to support the radio access containers, namely a supported SDR platform for LTE nodes and a supported Wi–Fi interface for Wi–Fi nodes. Edges nodes on the other hand are standard machines with no particular hardware requirements beside those of the VNFs they must execute. In our deployment we use a combination of Intel NUC, Soekris 6501, and Raspberry PIs.

As it can be seen from Fig. 6 the various *LightMANO* components such as *5G–EmPOWER*, the Ryu Controller, and the *LightMANO* Orchestrator itself are all deployed as containers. Although in the picture these containers are co–located, there is no particular location constraint and they can be effectively placed on different nodes. Standard VNFs, such as the Squid proxy or the LTE core network components (HSS, MME, and S/P–GW) depicted in the figure, are also run in this type of containers. In the figure we can also identify another type of container, namely the *LVAP/LVNF*–Host container. This type of container runs either the *5G–EmPOWER LVAP* Agent or the *5G–EmPOWER LVNF* Agent.

Figure 7 shows the architecture of an *LVAP/LVNF*–host. Notice how each *LVNF* consists of a dedicated Click instance (userlevel, kernel, or DPDK–accelerated). Each *LVNF* is attached to one or more ports of a software switch (e.g. OpenVSwitch). Conversely, all *LVAP*s share the same virtual interface. This derives from the fact that *LVAP*s (which we remind the reader are as many as the number of wireless clients associated to a given Access Point) are expected to be migrated between nodes more often than regular *LVNF*s. This design choice allow us to avoid adding and removing a port for the local software switch every time a handover occurs.

V. Use Case: Mobile Edge Caching

A. Overview

Mobile data traffic has been growing exponentially over the last few years [41]. Mobile network operators are trying to keep up with this surge in demand by deploying denser radio access networks and by employing the most recent advances in mobile communications (e.g. multi user MIMO, CoMP, etc.). However, in order to address the requirements of future low–latency, high–bandwidth applications, such as augmented reality and virtual reality, a fundamental rethink of the mobile network is necessary. By deploying computational resources very close to the end–users, possibly even within the mobile radio access network, MEC promises to relieve the mobile backhaul of at least part of the current data traffic while also decreasing the latency experienced by mobile applications.

In the section we will first describe a practical MEC application implementing an edge content caching service. The network service consists of a single LTE small cell, a GTP encap/decap LVNF, and a content caching VNF. The service is on–boarded by *LightMANO* which coordinates the creation of a new virtual small cell instance and the deployment of the two VNFs. Notice how, the GTP encap/decap VNF is implemented as an LVNF while the content caching VNF is implemented using a containerized version of the Squid [42] server.

Before moving into the details of this MEC application we need to briefly recap how user data traffic is handled in the LTE Mobile Network. Once a User Equipment (UE) attaches to the network it can send/receive data to/from the Packet Data Networks (PDN) using the GPRS Tunneling Protocol (GTP).

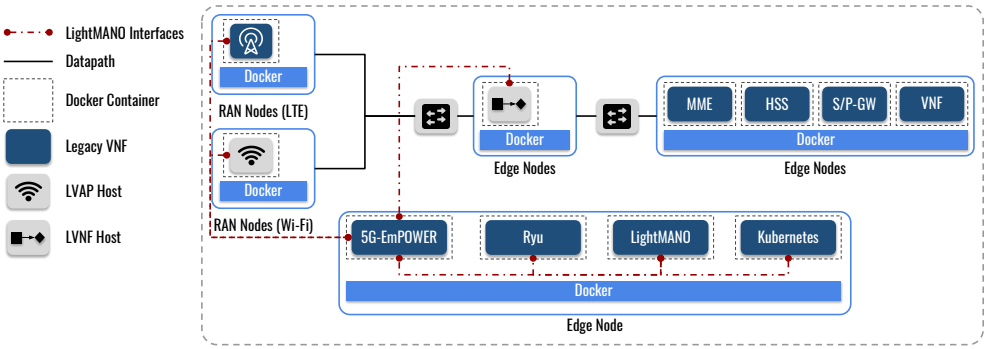


Fig. 6: LightMANO Proof–of–concept.

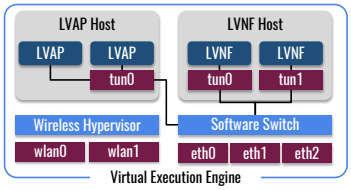


Fig. 7: The 5G–EmPOWER Virtual Execution Engine.

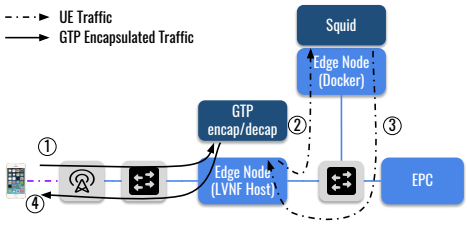


Fig. 8: The mobile edge caching network service.

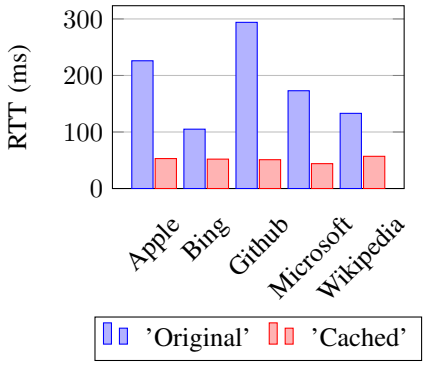


Fig. 9: Round Trip Time.

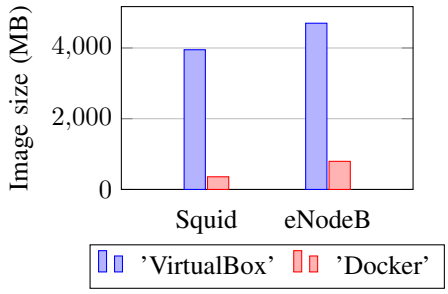


Fig. 10: Image size.

Uplink UE traffic received by an eNB over its air interface is encapsulated into a GTP packet and then delivered to the Serving Gateway (SGW) over a UDP/IP socket. This GTP tunnel is terminated at the SGW where a new GTP tunnel to the PDN Gateway (PGW) is created. Finally, the PGW removes the GTP header and forwards the UE traffic to its intended destination (e.g. the Internet).

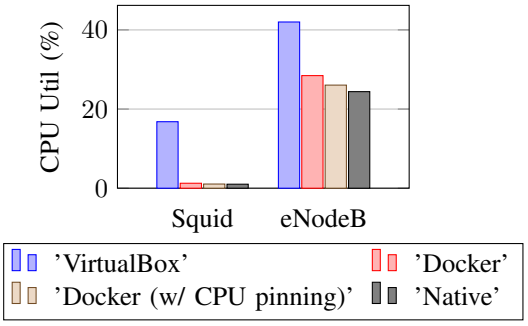
In order to operate, the Squid server needs to have access to the UE IP traffic. However, if placed within the RAN, Squid could access only the GTP–encapsulated traffic. The GTP encap/decap *LVNF* takes care of performing a statefull termination and recreation of the GTP session between the eNodeB and the SGW. Figure 8 sketched this process. The GTP–encapsulated UE traffic is redirected from the eNodeB to the network node running the GTP encap/decap VNF (Step 1). Here the GTP tunnel is terminated and the UE IP traffic, now accessible, is redirected to the node running the Squid server (Step 2). In case of a cache hit, the requested content is send back to the GTP encap/decap VNF (Step 3). Here the GTP tunnel is recreated and the cached response is finally delivered back to the UE (Step 4).

B. Evaluation Methodology

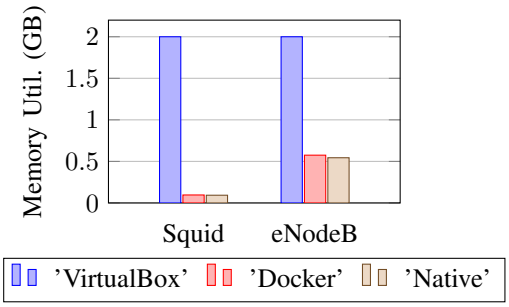
The experimental setup consists of an LTE small cell, a commercial EPC, a commercial Android smartphone with a programmable SIM card, and two standard x86 servers. The LTE small cell has been implemented using an Ettus B210 Software Defined Radio and srsLTE. Neither the LTE protocol stack nor the UE have been modified in anyway in order to perform the experiments.

C. Results

The first experiment shows the difference in Round Trip Time (i.e., the time taken to send a web page request from the UE to the Web Server and back) when the web request is served from the Squid Edge node, instead of directly from the origin Web server. The measurements are taken by making a web request (the CURL command–line tool is used) to five



(a) CPU Utilization.



(b) Memory utilization.

Fig. 11: Resource Utilization.

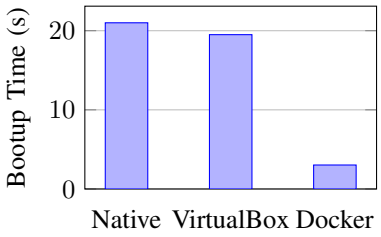


Fig. 12: System Bootup Time.

popular web pages from the UE. A significant performance improvement can be seen when the UE is served by the edge node rather than by the remote server (Fig. 9). Notice how, the goal of this test is not to demonstrate a performance improvement, solutions like Squid are indeed used since decades, on the contrary our objective here is two demonstrate that using a standard unmodified content caching software inside the mobile RAN is indeed possible with *LightMANO*.

In the second experiment we determine the different types of overhead for running applications in the containerized and the virtualized platform, compared to the native, non–virtualized platform. All of our tests were performed on x86 servers with 1.9 GHz Intel core i5 (4th generation) dual core processor with 8GB of RAM. We used Ubuntu 17.10 64-bit with Docker 17.06, and VirtualBox 5.1. All Docker containers used an Ubuntu 17.10 base image and all VMs used the Ubuntu 17.10 desktop image. We individually measure CPU, memory, boot-up time, and storage overhead by continuously requesting 1000 web pages from the UE associated to the eNodeB to derive 3 resource usage profiles: i) native, ii) Containerized (Docker), and iii) Virtualized (VirtualBox). We also note that the UE is allocated with 25 LTE resource blocks and scheduled in 2.6GHz frequency band.

The first comparison that can be drawn is the difference in size for similar images. As it can be seen in Fig. 10, the image containing Squid on top of Ubuntu weighs 3.95GB in VirtualBox, and only 358MB in Docker. Similar consideration can be made for the srsLTE application. The significant reduction in size is mainly due to the removal of kernel, peripheral drivers, etc., that are not required in a containerized environment, since the Docker image can access it directly through the host kernel.

CPU and memory utilization are often important performance metrics to be monitored at the edge and the RAN nodes, mainly due to its strict resource constraints. Raw single core utilization and memory consumption are monitored for the duration of the experiment at one second time interval by using the *top* Linux tool and from Docker Command Line Interface with *stats* command. The Squid application in the edge node utilizes 16.8% CPU when run in VirtualBox, whereas, Docker requires only 1.25% of the host CPU. In the case of Docker, pinning each virtual CPU to a physical CPU further reduces the overall CPU utilization to 1.05%. A similar test was also performed in the RAN node with srsLTE small cell and the overall analysis can be visualized from Figure 11a. In our experiment, both Docker and VirtualBox were limited to use a maximum of 2GB RAM for their operation. However, as seen in Figure 11b, in the case of VirtualBox the allocated memory is completely utilized for running Squid or srsLTE, whereas Docker only consumes 93.66MB of allocated memory for running Squid application and 554MB for running srsLTE.

The amount of time required to boot the system is another real-life metric that we have measured as shown in Figure 12. In case of VirtualBox, it took approximately 21 seconds for the operating system to start, whereas Docker takes only 3.2 seconds. This reduction is important for the *LightMANO* architecture, in which the nodes are initiated only when necessary.

VI. Conclusions

This paper aims at lowering the barrier for developing and deploying the next generation MEC application and services. Toward this end we propose *LightMANO* a lightweight MEC operating system converging SDN and NFV into a single platform for the management and orchestration of network services over scattered platforms. *LightMANO* builds upon lightweight computing and networking virtualization solutions like Docker and Click and moves part of the orchestration logic at the edges of the network.

A proof–of–concept implementation of *LightMANO* is also introduced and validated in a practical use case, namely content caching in mobile networks. The results of the evaluation show that it is indeed possible to use *LightMANO* to deploy standard caching applications with limited resource utilization *inside* the mobile RAN.

Our current work aims at enhancing the runtime system in order to improve system performances as well as the level of automation in deploying and managing network services. Particular attention will be devoted to SLA–enforcing algorithms and on joint RAN and edge resource allocation solutions.

REFERENCES

[1] European Telecommunications Standards Institute (ETSI), "Etsi gs nfv 002 network functions virtualisation (nfv); architectural framework," December 2014.

[2] "Docker." [Online]. Available: https://www.docker.com/

[3] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click Modular Router," *ACM Transactions on Computer Systems*, vol. 18, no. 3, pp. 263–297, 2000.

[4] R. Mijumbi, J. Serrat, J. l. Gorricho, S. Latre, M. Charalambides, and D. Lopez, "Management and orchestration challenges in network functions virtualization," *IEEE Communications Magazine*, vol. 54, no. 1, pp. 98–105, January 2016.

[5] "OpenBaton." [Online]. Available: https://openbaton.github.io/

[6] A. Fischer, J. F. Botero, M. Till Beck, H. De Meer, and X. Hesselbach, "Virtual network embedding: A survey," *Communications Surveys Tutorials, IEEE*, vol. 15, no. 4, pp. 1888–1906, 2013.

[7] M. Chowdhury, F. Samuel, and R. Boutaba, "Polyvine: policy-based virtual network embedding across multiple domains," in *Proc. of ACM VISA*, New Delhi, India, 2010.

[8] T. Mano, T. Inoue, D. Ikarashi, K. Hamada, K. Mizutani, and O. Akashi, "Efficient virtual network optimization across multiple domains without revealing private information," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 477–488, Sept 2016.

[9] I. Vaishnavi, R. Guerzoni, and R. Trivisonno, "Recursive, hierarchical embedding of virtual infrastructure in multi-domain substrates," in *Proc. of IEEE NetSoft*, London, UK, 2015.

[10] I. Houidi, W. Louati, W. B. Ameur, and D. Zeghlache, "Virtual network provisioning across multiple substrate networks," *Computer Networks*, vol. 55, no. 4, pp. 1011 – 1023, 2011.

[11] Q. Zhang, X. Wang, I. Kim, P. Palacharla, and T. Ikeuchi, "Vertex-centric computation of service function chains in multi-domain networks," in *Proc. of IEEE NetSoft*, Seoul, South Korea, 2016.

[12] P. Iovanna, F. Ubaldi, F. Giurlanda, S. Noto, A. Priola, L. M. Contreras, V. Lopez, and J. P. F. P. Gimenez, "Effective elasticity for data centers interconnection in multi-domain wan: Information modelling and routing," in *Proc. of ECOC*, Valencia, Spain, 2015.

[13] R. Guerzoni, D. Perez-Caparros, P. Monti, G. Giuliani, J. Melian, R. Figueiredo, A. Ramos, C. J. Bernardos, G. Biczk, B. Sonkoly, F. Tusa, A. Galis, I. Vaishnavi, F. Ubaldi, A. Sgambelluri, C. Santana, and R. Szabo, "Multi-Domain Orchestration and Management of Software Defined Infrastructures: a Bottom-Up Approach," in *Proc. of EuCNC*, Athens, Greece, 2016.

[14] C. Bernardos, L. Contreras, and I. Vaishnavi, "Multi-domain network virtualization," Working Draft, Internet-Draft draft-bernardos-nfvrg-multidomain-01, October 2016.

[15] European Telecommunications Standards Institute (ETSI), *Network Functions Virtualisation (NFV); Management and Orchestration; Report on Architectural Options*, Std. ETSI GS NFV-IFA 009, July 2016.

[16] "OpenStack." [Online]. Available: http://www.openstack.org/

[17] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 164–177, Oct. 2003.

[18] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella, "Opennf: Enabling innovation in network function control," in *Proc. of ACM SIGCOMM*, Chicago, Illinois, USA, 2014.

[19] B. Kothandaraman, M. Du, and P. Sköldström, "Centrally controlled distributed vnf state management," in *Proc. of ACM HotMiddlebox*, London, United Kingdom, 2015.

[20] A. Gember-Jacobson and A. Akella, "Improving the safety, scalability, and efficiency of network function state transfers," in *Proc. of ACM HotMiddlebox*, London, United Kingdom, 2015.

[21] S. Rajagopalan, D. Williams, H. Jamjoom, and A. Warfield, "Split/merge: System support for elastic execution in virtual middleboxes," in *Proc. of USENIX NSDI*, Lombard, IL, USA, 2013.

[22] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi, "Design and implementation of a consolidated middlebox architecture," in *Proc. of USENIX NSDI*, San Jose, CA, USA, 2012.

[23] J. W. Anderson, R. Braud, R. Kapoor, G. Porter, and A. Vahdat, "xomb: Extensible open middleboxes with commodity servers," in *Proc. of ACM/IEEE ABCS*, Austin, Texas, USA, 2012.

[24] T. L. Hinrichs, N. S. Gude, M. Casado, J. C. Mitchell, and S. Shenker, "Practical declarative network management," in *Proc. of ACM WREN*, Barcelona, Spain, 2009.

[25] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker, "Composing software-defined networks," in *Proc. of USENIX NSDI*, Lombard, IL, USA, 2013.

[26] A. Voellmy and P. Hudak, "Nettle: Taking the sting out of programming network routers," in *Proc. of ACM PADL*, Austin, TX, USA, 2011.

[27] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: A network programming language," *SIGPLAN Not.*, vol. 46, no. 9, pp. 279–291, Sep. 2011.

[28] A. Voellmy, H. Kim, and N. Feamster, "Procera: A language for high-level reactive network control," in *Proc. of ACM HotSDN*, Helsinki, Finland, 2012.

[29] A. K. Nayak, A. Reimers, N. Feamster, and R. Clark, "Resonance: Dynamic access control for enterprise networks," in *Proc. of ACM WREN*, Barcelona, Spain, 2009.

[30] C. Monsanto, N. Foster, R. Harrison, and D. Walker, "A compiler and run-time system for network programming languages," in *Proc. of ACM POPL*, Philadelphia, PE, USA, 2012.

[31] M. Casado, N. Foster, and A. Guha, "Abstractions for Software-defined Networks," *Commun. ACM*, vol. 57, no. 10, pp. 86–95, Sep. 2014.

[32] R. Riggio, M. K. Marina, J. Schulz-Zander, S. Kuklinski, and T. Rasheed, "Programming abstractions for software-defined wireless networks," *IEEE Transactions on Network and Service Management*, vol. 12, no. 2, pp. 146–162, 2015.

[33] L. Suresh, J. Schulz-Zander, R. Merz, A. Feldmann, and T. Vazao, "Towards Programmable Enterprise WLANS with Odin," in *Proc. of ACM Workshop on Hot Topics in Networks*, New York, 2012.

[34] R. Riggio, I. G. B. Yahia, S. Latr, and T. Rasheed, "Scylla: A language for virtual network functions orchestration in enterprise wlans," in *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, 2016.

[35] "NeMo: An Applications Interface to Intent Based Networks." [Online]. Available: http://nemo-project.net

[36] "Kubernetes." [Online]. Available: https://kubernetes.io/

[37] "DPDK." [Online]. Available: http://dpdk.org/

[38] "PF_RING." [Online]. Available: http://www.ntop.org/products/packet-capture/pf_ring/

[39] "Snabbswitch." [Online]. Available: https://github.com/snabbco/snabb

[40] "srsLTE." [Online]. Available: http://www.softwareradiosystems.com/tag/srslte/

[41] "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2015-2020," Cisco, Tech. Rep., 2016.

[42] "The Squid Caching Proxy." [Online]. Available: http://www.squid-cache.org/