Online VNF Lifecycle Management in an MEC-Enabled 5G IoT Architecture

Ioannis Sarrigiannis, Student Member, IEEE, Kostas Ramantas, Prodromos-Vasileios Mekikis, Angelos Antonopoulos, Senior Member, IEEE, and Christos Verikoukis, Senior Member, IEEE

Abstract—The upcoming fifth generation (5G) mobile communications urge software-defined networks (SDNs) and network function virtualization (NFV) to join forces with the multiaccess edge computing (MEC) cause. Thus, reduced latency and increased capacity at the edge of the network can be achieved, to satisfy the requirements of the Internet of Things (IoT) ecosystem. If not properly orchestrated, the flexibility of the virtual network functions (VNFs) incorporation, in terms of deployment and lifecycle management, may cause serious issues in the NFV scheme. As the service level agreements (SLAs) of the 5G applications compete in an environment with traffic variations and VNF placement options with diverse computing or networking resources, an online placement approach is needed. In this article, we discuss the VNF lifecycle management challenges that arise from such heterogeneous architecture, in terms of VNF onboarding and scheduling. In particular, we enhance the intelligence of the NFV orchestrator (NFVO) by providing: 1) a latency-based embedding mechanism, where the VNFs are initially allocated to the appropriate tier and 2) an online scheduling algorithm, where the VNFs are instantiated, scaled, migrated, and destroyed based on the actual traffic. Finally, we design and implement an MEC-enabled 5G platform to evaluate our proposed mechanisms in real-life scenarios. The experimental results demonstrate that our proposed scheme maximizes the number of served users in the system by taking advantage of the online allocation of edge and core resources, without violating the application SLAs.

Index Terms—Fifth generation (5G), Internet of Things (IoT), live migration, multiaccess edge computing (MEC), network function virtualization (NFV), scaling, software-defined network (SDN), testbed, virtual network function (VNF) orchestration, VNF placement.

I. INTRODUCTION

The exponential increase in requests for a variety of services creates the need for an omnipresent network, which should be faster, more responsive and reliable, and easily accessed under any conditions. According to recent reports, by 2024 the mobile subscriptions will reach 8.3 billion, a number that exceeds the current worldwide population by 0.2 billion [1]. Approximately, 45% of this cellular traffic is expected to be generated by an expanding ecosystem of smart connected devices, known as the Internet of Things (IoT) paradigm. The IoT growth is further accelerated by the penetration of IoT applications and services in our everyday life, as well as in a large segment of vertical industries, such as connected cars, smart homes, smart metering, and industry automation [2].

The wide range of IoT services calls for a disruptive, highly efficient, scalable and flexible communication network, able to cope with the increasing demands and number of connected devices, as well as the diverse and stringent application requirements. For instance, ultrahigh definition video streaming or augmented reality applications have increased bandwidth requirements, whereas autonomous driving, tactile Internet, and factory automation require low end-to-end (E2E) latency, in which some cases should remain below 1 ms. In this context, the emerging fifth generation (5G) of wireless communications, bringing together a set of enabling technologies, will support and advance the potential of the IoT technology.

Software-defined networking (SDN) is one of the key enabling technologies that paved the road toward the 5G revolution, by permitting the replacement of specific network equipment, used in a dedicated way, with software that can be executed in generic purpose hardware, enabling the separation of the data, and the control plane. Furthermore, the network function virtualization (NFV) technology [3] enables the virtualization of this networking software; hence, application and network functionalities are handled as virtual network functions (VNFs) and managed by an NFV orchestrator (NFVO), able to have control upon the various locations of a distributed system [4]. The flexibility offered by the SDN/NFV network design is taken one step further with the network slicing paradigm, which enables the creation of multiple logical networks over a common physical infrastructure, offering the necessary isolation to support multiple 5G services with different requirements [5].

Manuscript received August 1, 2019; revised September 10, 2019; accepted September 15, 2019. Date of publication October 1, 2019; date of current version May 12, 2020. This work was supported in part by the research projects SPOTLIGHT under Grant 722788, in part by AGAUR under Grant 2017-SGR-891 and Grant 2017-DI-068, in part by SPOTSG under TEC2017-87456-P, in part by SEMIoTICS under Grant 780315, and in part by 5G-SOLUTIONS under Grant 856691. (Corresponding author: Ioannis Sarrigiannis.)
The virtualization of functions and the flexibility in their placement are highly aligned with the concept of multiaccess edge computing (MEC), recently proposed by the European Telecommunication Standards Institute (ETSI) [6]. MEC technology is defined as the cloud computing capabilities offered at the edge of the network, at the end users’ proximity. MEC is responsible for delivering computing, storage, and networking resources to the end user, thus achieving significant reductions in service response times and increasing reliability and security, since services are located much closer to the users, instead of a remote cloud. IoT is widely considered one of the key use cases of the MEC technology [6], [7]. First, a wide range of IoT services can be deployed to the edge, including IoT data aggregation services, big data analytics, video streaming transcoders, etc., ensuring low-latency and ultra-reliable performance. Second, IoT devices, which often have limited computational and storage resources (e.g., sensors, smart meters, etc.), can significantly enhance their capabilities by offloading tasks and services to the edge [8]–[15].

Even though MEC is clearly one of the major players toward the 5G realization and the future IoT services, the technology is still at its infancy [16], and challenges [17], such as efficient deployment, resource allocation and optimization, and application lifecycle management, arise. Only recently, the ETSI MEC group released the initial phase two specifications that deal with the architecture, framework and general principles for service application programming interfaces (APIs), but no agreement on the standardization has been yet achieved [18]. Another nontrivial issue refers to the scheduling and placement of VNFs over the underlying infrastructure, including different MEC and remote cloud locations. In [19], cloud service deployment is modeled as a graph embedding problem, where service VNF forwarding graphs (VNFFGs), or VNF chains, are embedded on top of a network of hypervisors (or compute nodes). This is an NP-complete problem, and, hence, it is very time consuming to find an optimal solution even for small networks. However, reaction times in modern cloud-native infrastructures have gradually shortened to the point where services are individually scaled-out and scaled-in, responding to user demand in a matter of seconds. In order to keep up with the challenging cloud-native environments service orchestration and scale operations must be orchestrated in real time, with minimal computational complexity.

Having a closer look in the state-of-the-art works regarding the VNF placement problem, its study can be performed through studying the placement and management of virtual machines (VMs) [20]. Furthermore, we can find two different approaches in the literature: 1) offline, where the placement decision is taken in order to satisfy end-users’ requests, under various constraints and 2) online, where, in addition to the initial placement decision, real-traffic data, i.e., load, are utilized to trigger possible VM reallocation events [21]. On the one hand, concerning the offline works, Laghrissi et al. [22] presented an advanced predictive placement algorithm, where the optimal placement location is defined by the least used location that is closest to the majority of the user equipment (UE). In [23], a mathematical optimization model for VNF placement and provisioning is proposed, guaranteeing the quality of service (QoS) by including latency into the VNF chaining constrains. They focus, however, only on the placement of the virtualized LTE core functions, omitting the management and orchestration of the cloud applications and services that are co-hosted in the same infrastructure. Eramo et al. [24] studied the dynamic deployment of network services (chains) on different VMs and formulate their reallocation of VNFs as a mixed integer program, focusing on server power consumption. The migration solution provided in this article, though, is applicable only in networks with repetitive, over a specific time interval, traffic scheme. One basic limitation of the aforementioned works is that the performance of the proposed algorithms is assessed only through simulations.

On the other hand, regarding the online solutions, Jia et al. [25] studied how to deploy and scale VNF chains on the fly, using VNF replication across geo-distributed datacenters for operational cost minimization. Nevertheless, they limit each VNF chain deployment and scaling within the same datacenter. Furthermore, a traffic forecasting method for placing or scaling the VNF instances to minimize the inter-rack traffic is presented in [26], in the premises of a cloud datacenter. Even though a real implementation is offered, along with operator traffic driven simulations, the placement method in this article does not take into consideration the different requirements each VNF might have. Finally, both works are limited within the premises of a datacenter, failing to exploit the potential benefits offered by the edge-cloud architectures.

On a different note, there is a very limited amount of experimental works in the literature that tackles with MEC implementations, where new challenges arise in order to deploy and orchestrate a programmable and flexible MEC-enabled 5G testbed. For instance, a 5G-aware proof of the concept of an evaluation testbed with MEC capabilities has been described in detail in [27], without conducting though any real experiments to provide results. Furthermore, [28] and [29] are based on containers, another virtualization technology, which share the host operating system and provide process level isolation only, whereas their orchestrator has limited capabilities without the ability to support migration features. Finally, these works are limited to the technical implementation of the testbeds and do not tackle the VNF placement problem. To the best of our knowledge, there is no related work that: 1) combines the interplay of the MEC with the cloud, in a virtualized manner; 2) proposes and implements an online VNF placement algorithm; 3) exploits VNF migration and scaling capabilities to meet the service demands in real time; and 4) provides experimental results over a real 5G testbed implementation.

In order to efficiently manage the NFV ecosystem, there is the need for online and agile techniques for scheduling and orchestration of VNFs, as well as real environments that this technology can be applied to, in order to provide transparent and diligent testing and assessment. In [30], we presented an MEC-enabled 5G architecture, distributing the computational and network resources to the edge and core tiers. In this article, we take a significant step further by implementing this architecture in a real-testbed environment, utilizing VM
technology. Furthermore, we propose two novel algorithms for the joint orchestration of the MEC and cloud resources, thus enhancing the NFVO capabilities. Specifically, we first present an algorithm for the VNFFG embedding of virtualized chained services, taking into account their latency requirements and service priorities (e.g., based on their criticality). Then, we propose another algorithm for the real-time allocation of the VNFs to the MEC and cloud resources, leveraging real-time service scale-out, and scale-in features to meet the user service requests. Additionally, the second algorithm supports live service migration to further enhance the initial service placement, in order to handle efficiently the latency critical applications. Finally, we proceed to the validation of our proposed algorithm in an MEC-enabled 5G testbed implementation, deployed using open-source software over generic purpose hardware. The obtained experimental results, based on real-world 5G scenarios and cloud applications, provide useful insights for the potential of MEC-enabled architectures for real-life applications.

The remainder of this article is organized as follows. Section II presents the overall NFV-enabled architecture, along with some key concepts and the considered system model. Section III provides the proposed orchestration algorithms for VNF onboarding and scheduling. Section IV discusses the 5G testbed implementation and the employed open-source tools for its realization. Section V delivers the obtained experimental results, thoroughly explaining the different experimental scenarios, whereas Section VI is devoted to the conclusions of this article.

II. NFV Architecture

We consider an MEC-enabled 5G IoT architecture depicted in Fig. 1. A heterogeneous radio access network (RAN) topology is considered for the connection of the IoT devices that may employ different wireless technologies. In particular, we consider a network that includes standalone 5G base stations (gNBs), IoT access points (APs), and a cloud RAN deployment, where base band units (BBUs) are connected with remote radio head (RRH) units. This architecture fully supports NFV by enabling the virtualization of compute and network resources at the MEC and cloud hypervisors, located at the edge and core tier, respectively. The virtualized infrastructure manager (VIM) is responsible for the management and control of the compute, storage, and network resources of the NFV infrastructure (NFVI), while the NFVO performs the compute and network resource orchestration.

A. Edge Computing

The considered architecture includes two tiers of computational resources: 1) the cloud at the core tier and 2) the MEC at the edge tier. These are in the form of hypervisors (or compute nodes), where application and network VNFs are hosted for the duration of their lifecycle. Hypervisors are interconnected with an SDN data plane, forming a leaf-spine topology, i.e., a mesh with a constant number of hops. Although different topologies have been considered in the literature, the leaf-spine is standardized in modern data centers as it simplifies VNF scheduling and guarantees a fixed latency for the data
A company, an MEC node is deployed to the edge that hosts two applications. It must be noted that the edge tier, i.e., the MEC hosts, contains limited computing resources. These are typically allocated to VNFs that should be placed closer to the UE-side to satisfy specific service requirements (typically low latency).

B. Virtual Network Functions

Fully adopting the NFV paradigm, we consider that the 5G cloud applications are implemented in the form of VNFFGs that result in VNF chaining. Each virtual link has its own bandwidth and latency requirements, which are typically encoded in the VNF descriptor (VNFD) file, along with other VNF metadata. During VNF placement, network slicing can be employed to guarantee the networking requirements of the VNFs. Network slicing ensures service isolation and offers performance guarantees to the service tenants by reserving appropriate resources as denoted by the VNFD. Network slicing in 5G networks is supported by the programmable infrastructure, via appropriate northbound APIs. However, dedicated slices bear a significant cost for service providers, as resources are reserved even when they are not used by clients, hence, negating any potential statistical multiplexing gains. Therefore, dedicated slices are typically associated with services with high QoS requirements.

Furthermore, based on their delay constraints, VNFs can be classified as latency-critical VNFs (LCVNFs), which are sensitive to latency, and latency-tolerant VNFs (LTVNFs) that can tolerate a higher degree of delay. Accordingly, the 5G cloud applications can be classified into three categories: 1) real-time applications, consisting of high priority LCVNFs (HP LCVNFs); 2) near-real-time applications, consisting of low priority LCVNFs (LP LCVNFs); and 3) non-real-time applications that consist of LTVNFs. The VNF chaining feature, though, allows us to combine and connect the aforementioned VNFs. In general, due to its limited resources compared with the cloud, the MEC entity is usually reserved for LCVNFs, which are placed in proximity to the UEs, in order to minimize latency. On the other hand, LTVNFs, or even LP LCVNFs in specific situations, can be safely deployed to the cloud.

An example of VNF chaining is given in Fig. 2. A set of VNFs is chained both in the same and in separated hypervisors, in order to identify a person at the entrance of a company. Although the face recognition application [31] is broadly known through the cloudlets edge computing concept, it is a use case that is also compatible with MEC [32]. To achieve faster response time for the employees of the company, an MEC node is deployed to the edge that hosts two chained services: 1) a face recognition VNF and 2) a database (DB) VNF. If the person is identified in the employee DB, the whole process is finalized. Otherwise, VNF #1 sends its output to the face recognition VNF in the cloud, i.e., VNF #3, where the same procedure occurs with a general DB (VNF #4), including employees of the company from other locations, or customers.

C. VNF Lifecycle

Each individual VNF has a lifecycle, which is controlled and managed by the NFVO. The NFVO resides in the core tier and can be considered as the central controller of the system, in terms of filtering the incoming requests and reallocating the compute and network resources. It executes periodic checks in order to monitor the current availability of compute and network resources and ensures that the NFVI adapts to traffic variations. Overall, the VNF lifecycle consists of the following.

1) Day-0 configuration that includes VNF onboarding and resource allocation along with network service configuration.

2) Scale-out, where horizontal scale-out involves creating more instances of a given VNF; for load balancing purposes and is typically triggered when the allocated CPU, memory or network resource utilization is increased upon increased traffic.

3) Scale-in, which is the opposite process of scale-out and is triggered when a VNF is underutilized.

4) Live migration that involves moving a VNF to a different hypervisor for optimization purposes, without service interruption [21]. It includes running both instances (in the old and new hypervisor) in parallel, while service migration is performed, and only migrating RAM contents as a final step.

D. System Model

In our system model, Fig. 3, we focus on the VNF placement and resource allocation between the edge and the core tiers. At the core tier, there are M cloud hypervisors (Cloud(M)), with maximum capacity HCloudMax(M) and current utilization HCloud(M) per hypervisor. Respectively, at the edge tier there are N MEC hypervisors (MEC(N)), with maximum capacity HMECMax(N) and current utilization HMEC(N) per hypervisor. In this article, we focus on the interconnection of each MEC hypervisor, in a leaf-spine topology with the cloud hypervisors. There are incoming VNFs in the system, some of which could be chained. For each VNF(Type, Resources, Hypervisor) we define a type, i.e., HP LCVNF, LP LCVNF, or LTVNF, the required resources, that cannot exceed the hypervisor with the maximum capacity, and the hypervisor where it can be deployed. With respect to the service onboarding, we consider the following setup: 1) the real-time applications, hosted in HP LCVNFs, are deployed to the edge; 2) the near-real-time applications, hosted in LP LCVNFs can be either allocated to the edge or core tiers; and 3) the non-real-time applications, hosted in LTVNFs, are deployed to the core tier.
The scaling functionality of our system is being triggered based on the incoming requests per VNF, as multiple users can request data from the same VNF, resulting in increased VNF load. More specifically, we define: 1) the scale-out threshold, which defines the value of the CPU utilization, above of which a new VNF of the same type is instantiated; 2) the scale-in threshold, which defines the value of the CPU utilization, below of which the last created VNF is deleted; and 3) the cooldown period, which is the predefined time interval that should pass before a consecutive scaling event at the same VNF may occur. Finally, the live migration functionality can be triggered upon a scale-in or scale-out event and involves only the shifting of the LP LCVNFs.

### III. VNF Orchestration Algorithms

In this section, we discuss the role of the NFVO in the VNF lifecycle management, as well as the actual orchestration algorithms. In order to keep up with the challenging cloud-native environments, where subsecond reaction times are sometimes required, fast online algorithms are proposed. More specifically, the VNF scheduling problem is split in three phases, which are centrally controlled by the NFVO.

1. **Algorithm 1 VNFFG Embedding**
   1. The VNFFG embedding process starts from the services with the highest QoS. We traverse all VNFs in the VNFFG breadth-first, starting from the entry point where the UE connects.
   2. If the latency constraints of the VNF links exceed the round-trip time to the cloud, the VNF is assigned to the MEC. Otherwise, it is assigned to the cloud.
   3. If the MEC resources are exhausted, further deployment of HP LCVNFs is blocked, as well as their chained VNFs, unless they can tolerate the increased latency associated with the core tier (LP LCVNFs).

   ![System model](image)

   **Fig. 3.** System model.

   - The VNFFG embedding algorithm is devised to handle this operation, while a live migration step might be performed in cases of insufficient edge resources.
   - Service scale-in is also a periodic process, which erases VNF instances when the user demand decreases, to free up resources when they are not needed. We propose a live service migration step to be performed after the scale-in operation to further optimize the VNF placement.

   VNF scheduling is based on a cost function, which takes into account the hypervisor resources consumed by the VNF, i.e., CPU, memory, disk size, as well as bandwidth costs to interconnect the VNFs in the VNFFG. In general, the minimum cost is achieved when all VNFs of the same VNFFG are placed at the same hypervisor. It gradually increases as VNFs are placed at different hypervisors occupying network links for communication, while MEC hypervisors are generally assigned a higher cost than cloud hypervisors.

   - **A. VNFFG Embedding**
     Although many different topologies have been considered in the literature for the core and edge tiers, in this article, we consider a standard leaf-spine topology. This topology simplifies the VNFFG routing over the physical infrastructure, as all hypervisors in the core and edge tier are interconnected in a mesh with a fixed number of hops (Fig. 3). The VNFFG embedding is performed during the service bootstrapping phase at the NFVO that assigns VNFs to the core or edge tier based on their delay constraints. The edge tier hosts have a higher operational expenditure (OPEX) than the core tier hypervisors, and hence, a higher deployment cost which is reflected on the cost function. Thus, typically only a limited number of edge VNFs is deployed to the edge. Algorithm 1 explains the basic steps of the VNFFG embedding process.

   - **B. Online VNF Scheduling**
     VNF scheduling is an online problem, as VNFs are typically scaled-out and scaled-in within very fast time-frames, in the order of seconds, based on current traffic. Although many works in the literature solve an offline version of the problem, where the total number of VNFs is known during service bootstrapping, and do not take into consideration the real traffic of the VNFs, this assumption is not valid in modern cloud infrastructures. In this article, we assume that only the VNF assignment to the core or edge tier has been completed during the service bootstrapping phase, hence, the
Algorithm 2 Online VNF Scale-Out/Scale-in and Dynamic Live-Migration Scheduling

Input: $H_{\text{MEC}} \{ \text{Max} \{N\} \}$, $H_{\text{Cloud}} \{ \text{Max} \{M\} \}$, $H_{\text{MEC}} \{N\}$, $H_{\text{Cloud}} \{M\}$, VNF[Type, Resources, Hypervisor]

Triggering Event $e$, where $e$ in {scale-in, scale-out} VNF$e$

Output: Hypervisor for VNF placement

1: if $e = \text{scale-out}$ of VNF$e$ then
2: if VNF$e_\text{[3]} = \text{MEC}[e]$ then
3: repeat
4: if VNF$e_\text{[2]} \leq \text{MEC}[e]$ then
5: allocate VNF$e$ on MEC$[e]$
6: update MEC$[e]$ resources
7: else if VNF$e_\text{[1]} = \text{LP LCVNF and VNF}_e[2] \leq \text{max}(H_{\text{Cloud}})$ then
8: allocate VNF$e$ on max(H_{\text{Cloud}}) & flag it
9: update max(H_{\text{Cloud}})
10: else if VNF$e_\text{[1]} = \text{LP LCVNF exists on MEC}[e]$ then
11: if max(VNF$e_\text{[2]} \leq \text{max}(H_{\text{Cloud}})$ then
12: live migrate VNF$e$ to max(H_{\text{Cloud}}) & flag it
13: update HMEC$[e]$
14: update max(H_{\text{Cloud}})
15: end if
16: else
17: reject scale-out request
18: exit algorithm
19: end if
20: until VNF$e$ is allocated
21: else if VNF$e_\text{[3]} = \text{Cloud}[e]$ then
22: if VNF$e_\text{[2]} \leq \text{HCloud}[e]$ then
23: allocate VNF$e$ on Cloud$[e]$
24: update HCloud$[e]$ resources
25: else if VNF$e_\text{[2]} \leq \text{max}(H_{\text{Cloud}})$ then
26: allocate VNF$e$ on max(H_{\text{Cloud}})
27: update max(H_{\text{Cloud}}) resources
28: else
29: reject scale-out request
30: exit algorithm
31: end if
32: end if
33: else if $e = \text{scale-in}$ of VNF$e$ then
34: release VNF$e_\text{[2]}$
35: update hypervisor$e$ resources
36: if VNF$e_\text{[3]} = \text{MEC}[e]$ then
37: while flagged LP LCVNF exist on Cloud AND HMEC$[e]$ do
38: live migrate flagged VNF$e$ to HMEC$[e]$
39: update HMEC$[e]$, HCloud
40: end while
41: end if
42: end if

Please note that our algorithm can be executed in combination with any NFVO that supports scaling capabilities and VIM with live migration support.

In more detail, regarding the scale-out operation, we try to place the new VNF at the same hypervisor with the original VNF that is being scaled, in order to eliminate interhypervisor delays. Table I depicts the actions that are being performed, depending on the triggering event. As Fig. 4 demonstrates, in case the original VNF resides in an MEC hypervisor and there are available resources, the new VNF is allocated to the same hypervisor as well. In case of insufficient MEC resources, in the event of: 1) LP LCVNF type, it can be directly allocated to the cloud hypervisor with the most free resources, and is being flagged; 2) HP LCVNF type, a live migration of existing LP LCVNFs takes place to the cloud hypervisor with the maximum available resources, starting with the VNF that occupies the most resources, in order to free up MEC resources for the incoming VNF, along with updating the bookkeeping of the migrated VNFs (flagging); or 3) no LP LCVNF type exists at the MEC hypervisor, the scale-out request gets rejected. Conversely, on the occasion of a scale-in triggering event, the resources of its
hypervisor are released. In the case of an MEC hypervisor, we migrate back possible flagged LP LCVNFs, according to our bookkeeping.

Overall, the runtime complexity of the proposed algorithm is $O(n^2)$, as it is determined by the most important term, i.e., the max(), which is nested in one loop.

1) Scan the VNF array to find LP LCVNFs at the MEC hypervisor $\rightarrow O(n)$.

2) Calculate the maximum value of the array HCloud $\rightarrow O(n)$, nested in one loop $\rightarrow O(n^2)$.

In terms of runtime memory, we need:

1) four 1-D arrays to store the maximum and current capacities of the MEC and cloud hypervisors. Specifically, two arrays of size $N$ for the HMECMax and HMEC values, and two arrays of size $M$ for the HCloudMax and HCloud parameters will be allocated to the runtime;

2) one 2-D dynamic array with size $i \times 3$, to store the Type, Resources, and Hypervisor data for each VNF.

Regarding the execution of Algorithm 2, assuming the presence of $i$ VNFs in the system, the maximum number of iterations can be calculated for the worst-case scenarios. Specifically, for the first loop of the algorithm (steps 3–20), the maximum number of iterations is $(i-1)$. This occurs when an HP LCVNF needs to be scaled-out and all the remaining VNFs at the MEC are LP LCVNFs and must be all migrated to the cloud to release sufficient resources for the high priority function. Conversely, the maximum number of iterations for the second loop of Algorithm 2 (steps 37–40) is also $(i-1)$, under the occasion that the aforementioned LP LCVNFs return to their original hypervisor after the scale-in process of the HP LCVNF.

IV. TESTBED IMPLEMENTATION

In order to demonstrate the potential of the described architecture, we introduce a real implementation of an MEC-enabled 5G testbed, depicted in Fig. 5. The hardware of the testbed, as seen in Table II, consists of five physical servers, where the functionalities of the core tier (e.g., the cloud and the NFVO) and the edge tier (e.g., the MEC) are deployed to, as well as another physical server that enables the management, in terms of infrastructure virtualization. In terms of compute resources, the physical server at the edge site has significantly lower computational power compared to the servers at the core. In terms of networking, the physical servers are connected to two routers through 1 Gbp/s Ethernet interfaces.

With respect to the software installation, Openstack [33], on its Queens release, is the open-source Infrastructure-as-a-Service platform that is employed as the VIM, in order to deploy and control the VMs that will host the VNFs. The Openstack controller node, deployed to one physical server as shown in Fig. 5, hosts the compute and network management components for the virtualization and management of the infrastructure, while the compute nodes (or hypervisors), deployed to three physical servers, provide a pool of physical resources, where the VMs are being executed. Openstack is based on services, and in order to provide the needed isolation and management, they are deployed to LXD containers. For instance, the Nova service, part of the Openstack compute services that reside in all compute nodes, is responsible for spawning, scheduling, and decommissioning the VMs on demand, while the neutron service, which resides in all four nodes, is responsible for enabling the networking connectivity.
Additionally, the Openstack telemetry service (based on the Ceilometer service) is deployed to collect monitoring data, including system and network resource utilization, based on which further actions are taken. All nodes need two network interfaces, namely, the management, i.e., control plane, for the communication among the Openstack services and the NFVO, and the provider network, i.e., data plane, for the communication among the VMs, while each application has its own virtual tenant network.

Moreover, it is worth noting that Openstack supports two important features, namely, the horizontal scaling, i.e., the expansion of the physical resources, simply by adding new physical servers, where the compute node services are deployed to, and the live migration of the VMs. The migration is classified as live due to the fact that after a VM migration is complete, the VM status resumes exactly from the same state it was before the migration, without service interruption. The duration of the live migration might take from few seconds to several minutes, depending on various factors, including, but not limited to: 1) the virtualization platform; 2) the underlying hardware; 3) the type of the hypervisor; 4) the type of storage; 5) the footprint of the VMs in terms of vCPUs, RAM, and storage; 6) the current network load; and 7) the current VM load. Without any doubt, there should be an upper limit for the duration of the live migration, in order for the system to be agile and adaptive to the real-time traffic changes, but this is related to the actual system and the limitations that are being imposed by the hardware, the architecture decisions, and the virtualization platform.

The NFVO, which is responsible for the computing and network resource orchestration and management, is deployed as an independent entity to the fifth server, at the core tier, in alliance with the ETSI NFV information models, and is based on the Open source mano (OSM) [34], on its sixth release. Although there is a variety of NFVOSs in [16], its low hardware requirements, combined with the capabilities it offers, made OSM the most suitable NFVO for our system. OSM supports descriptor files written in yaml, namely, the VNFD and the network service descriptor (NSD). The former defines the needed VNF resources in terms of compute resources and logical network connection points, the image that will be launched at the VM, as well as the auto-scale thresholds (e.g., scale-in, scale-out, and cooldown period, minimum or maximum number of VNFs) based on the metrics that are being collected from the Telemetry service of the VIM. The latter is responsible for the connection point links, using virtual links, among the interconnected VNFs, mapping them to the physical networks provided by the VIM.

### Table II: Hardware Characteristics

| Hardware Characteristic | Controller Node | Cloud Compute Node x2 | MEC Compute Node | OSM |
|-------------------------|----------------|------------------------|-----------------|-----|
| CPU                     | Intel i5-8500  | i5-8500                | i5-7400         | i5-8400 |
| Cores                   | 6              | 6                      | 4               | 6    |
| RAM                     | 32GB           | 16GB                   | 8GB             | 8GB  |
| Storage                 | SAMSUNG 960 EVO 250GB | SAMSUNG 960 EVO 250GB | WD M2 2280 120GB | WD M2 2280 120GB |
| Network Interfaces      | 2x1Gbps Ethernet | 2x1Gbps Ethernet       | 2x1Gbps Ethernet | 2x1Gbps Ethernet |

Fig. 6. Scale-out process to accommodate increased incoming traffic.

Neither Openstack nor OSM are aware of the type of service that is being executed at the VNF. Furthermore, OSM is not aware of the hypervisor where the VM that hosts the VNF is being placed and leaves the VM placement to Openstack. This lack of placement control deprives OSM of controlling the migration feature. In order to gain control of this feature, we implemented this functionality with a bash script. Openstack supports four different placement methods via the compute schedulers; filter scheduling, based on filters and weights, chance scheduling that randomly selects the compute filters, utilization aware scheduling, based on actual resource utilization and availability zones scheduling where the compute nodes are divided into zones. None of the above options, though, take into account the actual service running at the VM or how to allocate LCVNFs or LTVNFs among the hypervisors. To that end, we devised the aforementioned bash script, which is based on our two proposed algorithms, and performs the onboarding, scale-out/in and live migration actions of the VNFs to the appropriate hypervisors.

### V. Experimental Results

In order to demonstrate the potential of the described architecture, we conducted a set of experiments, leveraging the MEC-enabled 5G testbed, as described in Section IV. In the following, we first provide an experimental setup, and then, we evaluate the performance of the proposed algorithms.

#### A. Experimental Setup

In our testbed setup, we assume one MEC and one cloud hypervisor. For our experiments, we define the max latency as: 1) 100 ms for the HP LCVNF and 2) 200 ms for the LP LCVNF. For the LTVNF, the latency is irrelevant as the
transmission is asynchronous. Note that the latency is measured as the E2E delay between the UE and the hypervisor, also corresponding to the response time of the application. The scale-out threshold is set at 90% CPU utilization, the scale-in at 30% and the cooldown period at 180 s. Since we assume the exponential service time of the LCVNF service, as soon as the CPU utilization exceeds the predefined threshold, the response time violates the service level agreement (SLA), so the scale-out process will take place prior to this violation. Please note that the aforementioned values are fully customizable, depending on the actual requirements of the diverse 5G applications.

Table III depicts the experimental setup in detail. Finally, the following three experiments run multiple times, separately, in a duration of 24 h each. Since most of the parameters were deterministic, the results were stable, with a variation of 2 ms.

### B. Autoscaling Experiment

In the first experiment, Fig. 6, we demonstrate the scale-out process. We start with one HP LCVNF and, as the traffic increases, the CPU utilization of the VNF increases accordingly. When it reaches the CPU utilization threshold at 90%, it is scaled-out and a second HP LCVNF is being instantiated. In order to equally distribute the traffic between the two VNFs, we deploy a load balancer VM with a round robin balancing policy. Hence, each VNF has approximately 45% CPU utilization when the new VNF is instantiated. While the traffic is further increased, another scale-out event is triggered and a third HP LCVNF is instantiated, with the load balancer distributing the incoming requests to three VNFs. This results in a 60% CPU utilization by the time the third VNF is instantiated. The measured scale-out duration for a VM with 1 vCPU, 1 GB of RAM, and 5 GB of storage, from the moment of the initial command to the VIM until the instantiation process was complete, was 15 s.

As we increase the traffic over time, we observe that the angle that is formed between the axes and the graph on Fig. 6 is reduced, according to the number of VNFs in the system that serve the requests. This is expected, as the traffic is equally distributed between two or three VNFs, while the traffic rate is increasing in a steady pace. With the autoscaling feature, we can accommodate more requests, compared with the legacy monolithic deployments that do not support such feature.

### C. Embedding Algorithm and Placement Experiment

In the second experiment, illustrated in Fig. 7, we demonstrate the various placement locations, validating that the algorithm for the onboarding process, i.e., Algorithm 1, provides the optimal placement result for maximizing the served requests. More specifically, we assume two chained VNFs, one HP LCVNF and one LTVNF, and we investigate three possible VNF placement methods: 1) all VNFs deployed to the **TABLE III**

| Parameter               | Value       | Parameter               | Value       |
|-------------------------|-------------|-------------------------|-------------|
| $H_{MEC_{Max}}$ (1)     | 3 vCPUs     | $H_{Cloud_{Max}}$ (1)   | 6 vCPUs     |
| HP LCVNF max latency    | 100 ms      | LP LCVNF max latency    | 200 ms      |
| LTVNF max latency       | Irrelevant  | Resources per VNF       | 1 vCPU      |
| Hypervisor HP LCVNF     | MEC         | Hypervisor LP LCVNF     | MEC/Cloud   |
| Hypervisor LTVNF        | Cloud       |Cooldown Period          | 180s        |
| Scale-out 90% CPU utilization | Scale-in 30% CPU utilization |
Fig. 8. Response time over traffic for the different deployment scenarios.

cloud (Fig. 7(a)); 2) the HP LCVNFs deployed to the MEC and the LTVNFs to the cloud (Fig. 7(b)); and 3) all VNFs deployed to the MEC (Fig. 7(c)). We reject the first solution as the SLA is being violated, because the HP LCVNF cannot tolerate the increased latency imposed by the MEC-cloud link. According to the embedding algorithm, the initial placement is performed based on latency constraints, i.e., the HP LCVNFs are allocated to the edge tier, while the LTVNFs are allocated to the core tier. After the initial placement, the HP LCVNF is hosted in the MEC (VNF1 {HP, 1, MEC}), while the LTVNF is hosted in the cloud (VNF2 {LTVNF, 1, Cloud}). This is the optimal placement solution as, in case of increased traffic, the HP LCVNF can scale-out twice, until the MEC resources are depleted (HMEC = 0), and serve more requests [Fig. 7(e)]. Finally, in the third deployment method, where everything is deployed to the MEC, the HP LCVNF can scale-out only once [Fig. 7(f)] and the MEC resources are depleted (HMEC = 0), since there is one LTVNF deployed to the MEC (VNF2 {LTVNF, 1, MEC}) that occupies 1 vCPU.

In Fig. 8, the response time of the VNFs versus the traffic is depicted, depending on the hypervisor that the VNFs are placed. From this figure, we can observe that if all the VNFs are deployed to the cloud, no further investigation is performed as this deployment method violates the SLA (over 100 ms) for the HP LCVNF. For the MEC-cloud placement method, i.e., VNFs are placed between the MEC and the cloud, the system will be able to support up to three HP LCVNFs at the MEC in order to serve up to 270 requests/second without violation of the SLA. Finally, while the third deployment method has improved response time due to the elimination of the link for the communication of the HP LCVNF with the LTVNF (they are hosted in the same hypervisor), the total requests/second that can serve are limited up to 180, due to the fact that the MEC resources quota has been reached.

D. Online VNF Scheduling Experiment

In the third experiment, depicted in Fig. 9, we demonstrate how the live migration feature can be employed to support more requests when LCVNFs with different priorities are competing for the same MEC resources, without interrupting the availability of the near real-time application. In this scenario, we take advantage of the live migration feature, described in Algorithm 2. Initially, the embedding algorithm allocates both VNFs to the edge tier (VNF1 {HP, 1, MEC}, VNF2 {LP, 1, MEC}). While the requests for the VNF1 are increasing, the CPU utilization increases as well, resulting in the scale-out of VNF1 (VNF3 {HP, 1, MEC}). When a second scale-out (VNF4 {HP, 1, MEC}) takes place, the MEC resources have been depleted (HMEC = 0), triggering the scheduling algorithm to: 1) live migrate the LP LCVNF to the cloud (VNF2 {LTVNF, 1, Cloud}), as depicted in Fig. 9(b) and 2) place the scaled-out HP LCVNF (VNF4) at the MEC [Fig. 9(c)]. When the traffic at the HP LCVNF is decreased, a scale-in (termination of VNF4) occurs and the LP LCVNF (VNF2) is migrated back to its original hypervisor [Fig. 9(d)].

In Fig. 10, we evaluate the response time versus the time in minutes. The requests for the HP LCVNF are increased over time while the requests for the LP LCVNF are stable. As the HP LCVNF needs to scale-out at the minute 85, the script commands the VIM to live migrate the LP LCVNF from the MEC to the cloud, thus freeing up resources for the scale-out of the HP LCVNF. The live migration process, at the minute 85, lasts 28 s, for a VM with 1 vCPU, 512-MB RAM, and 3-GB local storage, while no service interruption was observed. Note that during the live migration process, we notice a slightly increased response time for the LP LCVNF that is not violating the SLA neither during nor after the migration has been completed. Finally, when the scale-in action occurs at the minute 145, the LP LCVNF is migrated back to its original hypervisor, in accordance with Algorithm 2.
VI. CONCLUSION

In this article, we presented an MEC-enabled 5G IoT architecture, able to exploit the interplay between the core and edge tiers in an NFV environment. We discussed the key enabling technologies and filled the gap between the NFVO and the VIM entities by proposing embedding and scheduling algorithms for the initial placement and online reallocation of the VNFs, respectively, leading in enhanced VNF lifecycle management. We applied our algorithms to a fully deployed MEC-enabled 5G testbed implementation, where applications with different priorities and latency constraints have been executed. The conducted experiments showed that through the proposed schemes, a better utilization of MEC and cloud resources can be obtained on the fly, enabling the system to serve a higher number of latency critical applications without SLA violation.

As future analysis, we aim to extend our work by including the RAN part, both in our architecture and in the real-deployment environment. Furthermore, we will investigate more VNF allocation and placement policies and we will measure the overhead introduced by the live migration process.

REFERENCES

[1] (Jun. 2019). Ericsson Mobility Report. [Online]. Available: https://www.ericsson.com/en/mobility-report/reports/june-2019

[2] “Cisco visual networking index: Forecast and trends, 2017–2022,” San Jose, CA, USA, Cisco, White Paper, Feb. 2019. [Online]. Available: https://www.cisco.com/c/ en/us/solutions/collateral/service-provider/visual-networking-index-vni/ white-paper-c11–741490.pdf

[3] ETSI NFV. (Dec. 2014). Network Functions Virtualisation (NFV); Management and Orchestration. [Online]. Virtualisation, https://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/02.01.01_60/gs_MEC003v020101p.pdf

[4] A. J. Gonzalez, G. Nencioni, A. Kamisiﬁs, B. E. Helvik, and P. E. Heegaard, “Dependability of the NFV Orchestrator: State of the art and research challenges,” IEEE Commun. Surveys Tuts., vol. 20, no. 4, pp. 3307–3329, 4th Quart., 2018.

[5] S. Sharma, R. Miller, and A. Francini, “A cloud-native approach to 5G network slicing,” IEEE Commun. Mag., vol. 55, no. 8, pp. 120–127, Aug. 2017.

[6] “MEC in 5G networks,” Sophia Antipolis, France, ETSI NFV, White Paper 28, Jun. 2018. [Online]. Available: https://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp28_mec_in_5G_FINAL.pdf

[7] P. Porambage, J. Okwuobe, M. Liyange, M. Ylianttila, and T. Taleb, “Survey on multi-access edge computing for Internet of Things realization,” IEEE Commun. Surveys Tuts., vol. 20, no. 4, pp. 2961–2991, 4th Quart., 2018.

[8] P. Mach and Z. Becvar, “Mobile edge computing: A survey on architecture and computation offloading,” IEEE Commun. Surveys Tuts., vol. 19, no. 3, pp. 1628–1656, 3rd Quart., 2017.

[9] C. X. Mavromoustakis, J. M. Batalla, G. Mastorakis, E. Markakis, and E. Pallis, “Socially oriented edge computing for energy awareness in IoT architectures,” IEEE Commun. Mag., vol. 56, no. 7, pp. 139–145, Jul. 2018.

[10] Y. Nikoloudakis et al., “Edge caching architecture for media delivery over P2P networks,” in Proc. IEEE 23rd Int. Workshop Comput.-Aided Model. Design Commun. Links Netw. (CAMAD), Barcelona, Spain, Sep. 2018, pp. 1–5.

[11] C. X. Mavromoustakis, G. Mastorakis, and J. M. Batalla, “A mobile edge computing model enabling efﬁcient computation offload-aware energy conservation,” IEEE Access, vol. 7, pp. 102295–102303, 2019.

[12] S. Liao, Z. Zhou, S. Mumtaz, and J. Rodriguez, “Robust task offloading for IoT Fog computing under information asymmetry and information uncertainty,” in Proc. IEEE Int. Conf. Commun. (ICC), Shanghai, China, May 2019, pp. 1–6.

[13] M. Mukherjee, S. Kumar, M. Shojaifar, Q. Zhang, and C. X. Mavromoustakis, “Joint task offloading and resource allocation for delay-sensitive fog networks,” in Proc. IEEE Int. Conf. Commun. (ICC), Shanghai, China, May 2019, pp. 1–7.

[14] X. He, R. Jin, and H. Dai, “Deep PDS-learning for privacy-aware offloading in MEC-enabled IoT,” IEEE Internet Things J., vol. 6, no. 3, pp. 4547–4555, Jun. 2019.

[15] Z. Ning, P. Dong, X. Kong, and F. Xia, “A competitive partial computation offloading scheme for mobile edge computing enabled Internet of Things,” IEEE Internet Things J., vol. 6, no. 3, pp. 4804–4814, Jun. 2019.

[16] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, “On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration,” IEEE Commun. Surveys Tuts., vol. 19, no. 3, pp. 1657–1681, 3rd Quart., 2017.

[17] S. Shahzadi, M. Iqbal, T. Dagkiuklas, and Z. U. Qayyum, “Multi-access edge computing: Open issues, challenges and future perspectives,” J. Cloud Comput., vol. 6, no. 1, 30 Dec. 2017.

[18] Multi-Access Edge Computing (MEC): Framework and Reference Architecture, V2.1.1, document GS MEC 003, ETSI Group Specification, Sophia Antipolis, France, Jan. 2019. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/02.01.00_60/gs_MEC003v020101p.pdf

[19] L. Wang, Z. Lu, X. Wen, R. Knopp, and R. Gupta, “Joint optimization of service function chaining and resource allocation in network function virtualization,” IEEE Access, vol. 4, pp. 8004–8009, 2016.

[20] S. Herker, X. An, W. Kiess, S. Beker, and A. Kirstaedter, “Data-center architecture impacts on virtualized network functions service chain embedding with high availability requirements,” in Proc. IEEE Globecom Workshops (GC Wkshps), San Diego, CA, USA, Dec. 2015, pp. 1–7.

[21] A. Laghriessi and T. Taleb, “A survey on the placement of virtual resources and virtual network functions,” IEEE Comm. Surveys Tuts., vol. 21, no. 2, pp. 1409–1434, 2nd Quart., 2019.

[22] A. Laghriessi, T. Taleb, M. Bagaa, and H. Flinck, “Towards edge slicing: VNF placement algorithms for a dynamic & realistic edge cloud environment,” in Proc. IEEE Glob. Conf. Comput. Conf. (GLOBECOM), Dec. 2017, pp. 1–6.

[23] D. B. Oljira, K.-J. Grinnemo, J. Taheri, and A. Brunstrom, “A model for QoS-aware VNF placement and provisioning,” in Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN), Berlin, Germany, Nov. 2017, pp. 1–7.

[24] V. Eramo, E. Miucci, M. Ammar, and F. G. Lavacca, “An approach for service function chain routing and virtual function network instance migration in network function virtualization architectures,” IEEE/ACM Trans. Netw., vol. 25, no. 4, pp. 2008–2025, Aug. 2017.

[25] J. Jia, C. Wu, Z. Li, F. Le, and A. Liu, “Online scaling of NFV service chains across geo-distributed datacenters,” IEEE/ACM Trans. Netw., vol. 26, no. 2, pp. 699–710, Apr. 2018.

[26] H. Tang, D. Zhou, and D. Chen, “Dynamic network function instance scaling based on traffic forecasting and VNF placement in operator data centers,” IEEE Trans. Parallel Distrib. Syst., vol. 30, no. 3, pp. 530–543, Mar. 2019.

[27] L. T. Bolivar, C. Telisio, D. M. Area, and G. Tsolias, “On the deployment of an open-source, 5G-aware evaluation testbed,” in Proc. 6th IEEE Int. Conf. Mobile Cloud Comput. Services Eng. (MobileCloud), Bamberg, Germany, Mar. 2018, pp. 51–58.
A. C. Baktir, A. Ozgovde, and C. Ersoy, “How can edge computing benefit from software-defined networking: A survey, use cases, and future directions,” IEEE Access, vol. 7, pp. 72622–72633, 2019.

M. Z. Khan, S. Harous, S. U. Hassan, M. U. G. Khan, R. Iqbal, and I. Sarrigiannis, E. Kartsakli, K. Ramantas, A. Antonopoulos, and C. Verikoukis, “Application and network VNF migration in a MEC-enabled 5G architecture,” in Proc. IEEE 23rd Int. Workshop Comput.-Aided Model. Design Commun. Links Netw. (CAMAD), Barcelona, Spain, Sep. 2018, pp. 1–6.

M. Z. Khan, S. Harous, S. U. Hassan, M. U. G. Khan, R. Iqbal, and S. Mumtaz, “Deep unified model for face recognition based on convolution neural network and edge computing,” IEEE Access, vol. 7, pp. 72622–72633, 2019.

A. C. Baktir, A. Ozgovde, and C. Ersoy, “How can edge computing benefit from software-defined networking: A survey, use cases, and future directions,” IEEE Access, vol. 7, no. 4, pp. 2359–2391, 4th Quart., 2017.

The Openstack Foundation. Accessed: Jul. 30, 2019. [Online]. Available: www.openstack.org

Open Source MANO. Accessed: Jul. 30, 2019. [Online]. Available: osm.etsi.org