Lightweight edge Slice Orchestration Framework

Sagar Arora, Adlen Ksentini, Christian Bonnet
Eurecom, Sophia Antipolis, France
firstname.lastname@eurecom.fr

***Abstract*—Edge computing is one of the critical components enabling low-latency demanding services in beyond 5G networks. Indeed, the deployed applications at the edge benefit from their close position to end-users to guarantee low latency access. Considering the case of a network slicing enabled network, we introduce Lightweight edge Slice Orchestration (LeSO) framework, a cloud-native oriented orchestrator that orchestrates and manages the deployment of micro-services as sub-slices at the edge. Whilst the existing orchestration frameworks are greedy of computing resource consumption and fail to integrate with the Multi-access Edge Computing (MEC) domain, LeSO by design is very lightweight and integrates a MEC platform-like component to guarantee traffic steering to automate edge slice deployment. Experiment results show that LeSO necessities a small amount of CPU and memory, even when a high number of edge slices are deployed.**

***Index Terms*—NFV, MEC, CNF, Orchestration, Cloud-native**

I. Introduction

Edge cloud or Multi-access Edge Computing (MEC) [1] [2] provides a home to low latency demanding applications. Close placement of MEC to its end users makes it suitable to deploy emerging 5G services, such as Mission Critical Services (MCS), Virtual Reality (VR), and Augmented Reality (AR) based applications, etc. Including MEC in Network Slicing concept [3] can be an enabler for such applications in a Network Slice (NS). Applications hosted at the MEC require traffic redirection or DNS-based redirection rules to steer traffic to MEC Applications instead of the internet. As the traffic redirection needs to be done dynamically at the instantiation of the MEC application, ETSI MEC defined the traffic redirection as a rule in the Application Descriptor (AppD) describing the MEC application. Besides, the traffic redirection is enforced by the MEC Platform element that acts as an interface between the MEC and 5G domains. The reader may refer to [4] for more details on the MEC architecture, including MEP and AppD.

Network Slicing (NS) has emerged from the need to support heterogeneous 5G services sharing the same physical infrastructure. A Network Slice (NS) [5] is considered as an isolated logical layer on top of a physical infrastructure aiming at handling a specific type of traffic. According to 3GPP, when a NS is deployed, it is known as a Network Slice Instance (NSI). It is composed of one or more Network Slice Subnet Instances (NSSI), which may be dedicated to a NSI or shared among other NSI's. NSSI's contains either Virtual Network Functions (VNFs) or Physical Network Functions (PNFs) or radio resources or transport resources. NSSIs are deployed on top of different technological domains (i.e., radio, edge, transport network, cloud) and are stitched together to build the end-to-end network slice corresponding to a NSI. Usually, a Slice Orchestrator (SO) manages the life cycle of a NSI, which is composed of four steps: preparation, instantiation, configuration, activation, and decommissioning. Several LCM steps are delegated to the sub-slice orchestrators that manage the technological domain where a NSSI is deployed. In this context, a MEC Application can be considered to be a part of an edge sub-slice (i.e., NSSI), and the MEC orchestrator (MEO) [4] can be regarded as the edge sub-slice orchestrator.

The well-known existing orchestration frameworks, such as Open Network Automation Platform (ONAP)[1] and Open Source Mano (OSM)[2], were designed to manage the life cycle of VNFs or network services, a connected graph of multiple VNFs and PNFs. Currently, none of them perform life cycle management of MEC applications. Though they can instantiate a MEC application at the edge, they do not communicate to MEP or provide MEP capabilities to interface with the 5G network and steer traffic to the freshly deployed MEC application instance. Moreover, these frameworks have a complex design and high resource consumption, making their positioning unsuitable for the edge cloud.

In this paper, we propose a novel framework for orchestrating and managing the life cycle of Edge Sub Slices (ESS) that are crucial for low latency demanding services. The architecture of the framework is designed following micro-services and cloud-native principles [6]. It allows orchestrating and managing multiple slices at the same time. The paper's contributions are:

- A cloud-native Lightweight edge Slice Orchestration (LeSO) framework that is specifically designed to orchestrate micro-services based cloud-native MEC Applications and deploy them as fully isolated edge-sub-slices
- An Edge Sub-Slice Template (ESST) describing an ESS to manage cloud-native container-based applications following micro-services design. The template contains a modified version of AppD.

The paper also provides the placement of the framework in global network slicing orchestration architecture. Finally, we provide performance evaluation results of the LeSO framework to prove its low computing resource consumption as it is as-

This work was partially supported by the European Union's Horizon 2020 Research and Innovation Program under the 5G!Drones project (Grant No. 857031).

[1]https://www.onap.org
[2]https://osm.etsi.org

sumed to be deployed at the edge. The performance evaluation results were obtained via experiments as LeSO is deployed in EURECOM 5G trial facility [7] deployed in Eurecom, Sophia Antipolis.
The paper is organized into five sections: background, related work and motivation, framework and implementation, performance evaluation, and conclusion.

II. Background: Cloud-Native

One of the challenges related to deploying a MEC application at the Edge sub-slice is the limited computation resources available at the Edge node. Indeed, the first deployments of VNF were considering the usage of Virtual Machine (VM) or virtualization technology, which requires a full operating system that increases computing resource consumption. Besides, significant software image size constraints the MEC application design to be monolithic. In this context, replacing VM with containers and virtualization with containerization will provide several advantages for MEC application deployment. First, containers allow designing micro-services-based (or cloud-native) MEC applications, which bring the benefit of being more flexible and having faster failure recovery compared to VM-based MEC applications. Second, containers require only application-specific packages. They do not need an entire operating system, which reduces their computational resource consumption and instantiation time, making them lighter than VMs. For these reasons, it is more efficient to consider containerization over virtualization technology to use for Edge Slice deployment bringing cloud-native network functions (CNF) to the edge.

Basically, the architecture of cloud native application, by its definition, follows a micro-service design. Ideally, each functional component of the application should run in a separate container, which should run only one single computing process. Today Kubernetes[3] is an industrial de facto standard for container orchestration. The smallest entity which can be scheduled by Kubernetes is a Pod, a group of containers. One of the containers of the Pod contains the function of the application and rest of the containers are sidecars or initialization containers i.e, can be used for configuration management etc. Hence, a cloud native application deployed on Kubernetes contains multiple Pods which communicate with each other.

Modern network functions, Container-based Network Functions, or CNFs follow the above-defined design strategy for Kubernetes. MEC applications are equivalent to CNFs, but they are less sophisticated and require different life cycle management and additional traffic redirection rules. To define these applications using ETSI MEC defined Application Descriptor AppD [8] is not possible. Originally AppD was designed to define VM-based MEC applications. It provides the possibility to define one container or software image.

There is a possibility to use several AppDs instead of one single AppD to define such a MEC application, but again it does not allow defining multiple container images inside one single AppD. To overcome this limitation, we proposed an Edge Sub Slice Template (ESST) that contains our proposed modified version of the original AppD. It allows describing a cloud native MEC application.

III. Related Work and Motivation

There are several network service orchestrators that function as slice orchestrators. But none of them were designed for running MEC applications or network functions at the edge. Mainly, they were designed to run VM-based VNFs, and later some have been adapted to CNFs.

- **Open Network Automation Platform (ONAP)**, Initially designed to manage VM-based VNFs, and with the recent release, it is following cloud-native principles. ONAP's complicated architecture results in high resource consumption. Making it unsuitable for deploying it at the edge, to be close to its managed edge resources.
- **Open Source MANO (OSM)** is a network service orchestrator. Designed to orchestrate VM-based network services, it supports container-based network functions. OSM can deploy network functions at the edge cloud but not MEC Applications, which require MEP support.

ONAP and OSM both have their own CNF or network service descriptor packages, and apart from that, they need VIM-specific packages like helm-charts[4]. This forces the CNF provider to be infrastructure and platform aware. Besides these two tools, there is SONATA Service Platform [9], which can orchestrate CNFs; but again, it does not orchestrate MEC applications. Finally, in [10] the authors have briefly mentioned about a MEC orchestrator that orchestrates Linux Container (LXC) based MEC applications using AppD. To the best of our knowledge, this work is the only one that mentions about MEC orchestrators.

The motivation behind our work is the absence of a lightweight edge slicing orchestration framework. Which provides the features of MEC, follows cloud-native principles, is able to orchestrate container-based micro-services applications, and abstracts the platform and infrastructure-related information from MEC application or CNF providers. The providers should only be aware of their own application.

IV. Lightweight edge Slice Orchestration Framework

This section introduces the LeSO framework highlights its placement in the global end-to-end network slicing architecture and its design and implementation. It also describes the skeleton of the proposed ESST.

A. Global Architecture

Network Slice Orchestrator or simply a Slice Orchestrator (SO) communicates with several sub-slice or domain-specific orchestrators, which manages the sub-slice of their own domain. In terms of basic functionality, SO and other sub-slice

[3]https://kubernetes.io/

[4]https://helm.sh/

orchestrators can be considered equivalent to 3GPP Network Slice Management Function (NSMF) and Network Sub Slice Management Function (NSSMF), respectively. In this paper, we will not focus on the placement of these orchestrators, but we consider that these orchestrators are placed in their domain or near to their domain close to their managed resources.

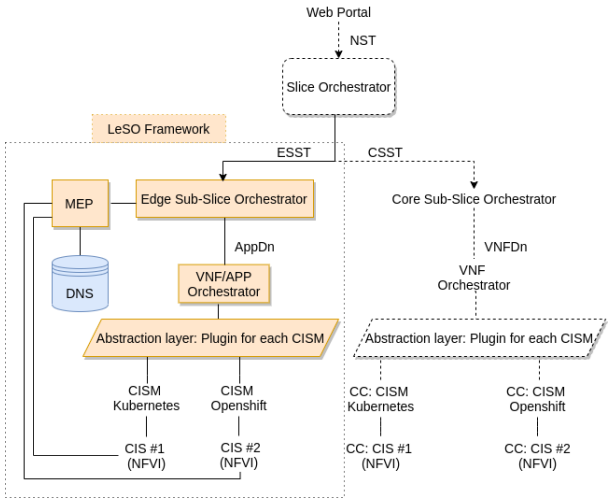

Fig. 1: Placement of the LeSO Framework in Global NS LCM

Each of these domains is responsible for managing different types of network functions (VNFs or PNF) or resources. For example, the edge domain for latency-sensitive network functions or vertical applications, the core domain for core network functions, the transport domain for physical or virtualized transport resources, and the RAN domain for physical or virtualized RAN resources.

Fig 1 proposes the placement of the LeSO framework in the global NS LCM architecture. In the figure, we have only shown edge and cloud sub-slice orchestrators, but for end-to-end life cycle management, there are also transport and radio sub-slice orchestrators.

B. Edge Sub-Slice

We consider an ESS to contain multiple MEC Applications or CNF or both, not connected or connected to each other. MEC Applications and CNFs consume the services of one and others via their own service discovery mechanism if needed. Fig 2 shows an example of an edge slice that contains, *MEC App1* which follows micro-service architecture divided in *MEC App1:1* and *MEC App1:2*. MEC App2 and CNF are independent.

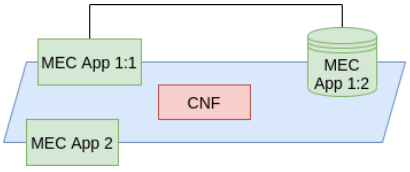

Fig. 2: Edge Sub Slice Skeleton

The slice of fig 2 can be described using our proposed Edge Sub Slice Template (ESST) that contains multiple modified AppDs. In the previous sections, we defined the limits of AppD and why it can not be used to describe a micro-services based container application. Fig3 defines a skeleton of ESST.

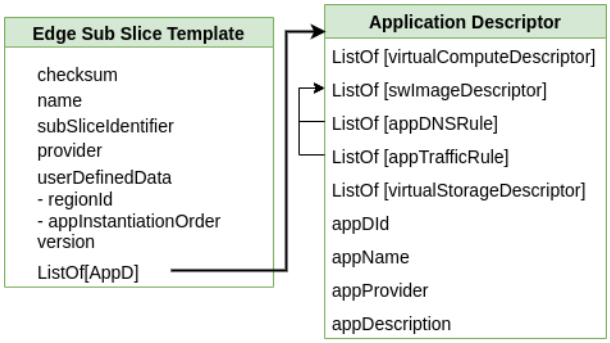

Fig. 3: Skeleton of Edge Sub-Slice Template

Multiple AppDs allow defining multiple MEC applications or CNFs connected or not connected to each other. Below are the major modifications proposed in the original AppD to adapt it for the cloud native container world. Definition of the AppD fields remains the same as described in the original AppD,

- Array of *swImageDescriptor* instead of single cardinality and use it to define multiple container images and application configuration.
- Included a parameter *configuration* in *swImageDescriptor*, to configure application container specific configuration.
- Included a parameter *port* in *swImageDescriptor*, to define networking ports exposed by the container. This parameter is used to provide the orientation of the exposed ports. It can be either towards: (i) Mobile Network(MN) if the application running inside the container needs to be exposed to MN. This is for edge use cases where user equipment will use the application deployed at the edge via traffic redirection rather than going to the internet. (ii) Container Network (CN) for internal communication between AppDs of the same ESST, (iii) the Internet (IN), for container application to expose its GUI towards the internet.
- Array of *virtualComputeDescriptor* instead of single cardinality to define resource consumption of each container of a pod.

MEC applications can connect to each other in cloud native way by communicating with the other application using its CN exposed port. The service attached to the exposed port can be accessed using Pods Fully Qualified Domain Name (FQDN) and the port number. Other important fields of ESST are; (i) **appDInstantiationOrder** it is a list of appDId that describes in which order the AppDs will be treated. This parameter is important for connected AppDs, which have dependencies on other AppDs present in the same ESST. (ii) **regionId** is a list of edge sites where ESST will be created.

The LeSO framework only requires a ESST to handle the life cycle management of MEC applications. It does not

require any other package or descriptor which are used by other VNF or service orchestrators. ESST is designed to be described by the owners of the MEC applications, who should be unaware of the underlying platform or infrastructure. ESST is agnostic to the type of platform or infrastructure MEC Application will run.

C. LeSO Design

The edge clouds are designed to support latency-sensitive applications, which do not consume high computational resources. LeSO framework is specifically designed for edge clouds, where the computation resources are scarce.

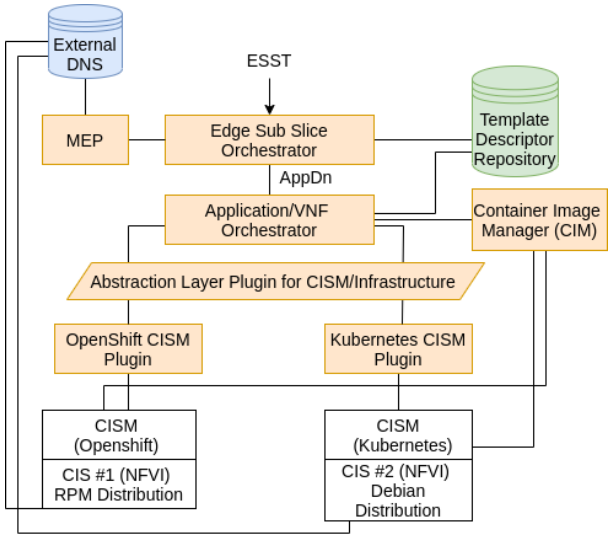

Fig. 4: Lightweight edge Slice Orchestration Framework

All the components expose a restful northbound Application Programming Interface (API) for communication. The design and deployment follow cloud-native principles. The current version of the framework can deploy MEC Applications or simple CNFs on top of container-based clouds, such as Openshift[5] and Kubernetes. Below is the component description,

- **Edge Sub-Slice Orchestrator (ESSO)**: Manages the life cycle of ESST and communicates with MEP and SO.
- **Container Infrastructure Service Management (CISM)** and **Container Infrastructure Service (CIS)**: Corresponds to container orchestration platform and CIS to the infrastructure, respectively. Their functioning is defined in the ETSI GR NFV-IFA 029 specification [11].
- **Application/VNF Orchestrator**: Manages the life cycle of MEC applications or CNFs. It is agnostic to the type of CISM, and it communicates to CISM via CISM plugin.
- **Template and Descriptor Repository (TDR)**: Database to store ESSTs and application descriptors.
- **MEC Platform**: Performs traffic redirection or DNS redirection by communicating with the 5G core and DNS, respectively.
- **External DNS**: All the COE/CISM have a DNS running in their cluster. The cluster DNS relies on this external DNS to resolve FQDN that is not present in the cluster DNS. This DNS is managed by the MEC platform to perform DNS redirection.
- **CISM Plugin**: Is an abstraction layer that takes the information provided by the application orchestrator and converts it into CISM-specific Yet Another Markup Language (YAML) templates. That are needed by CISM to create objects and workloads.
- **Container Image Manager (CIM)** manages container images. It can pull container images from all the public repositories make them from the source code present in a git repository. Lastly, it provides the possibility to download container images in tar format from a secure link.

ESSO can handle multiple edge sites, where each site has a specific **regionId**. ESST use the **regionId** field of ESST to define its placement. ETSI MEC in NFV framework or ETSI NFV-MANO framework proposes to have a dedicated VNF manager (VNFM) for each VNF or MEC application. This results in an extra entity that consumes more computational resources. Instead, we proposed a App/VNF Orchestrator which manages the life cycle of each MEC Application or CNF.

The abstraction layer provides the flexibility to include different CISMs. Northbound of all the CISM plugins is uniform, and their southbound adapts to the CISM API.

D. Isolation between slices

LeSO framework is slice aware by creating fully isolated edge slices. All the applications and CNFs of a slice run in an isolated environment, known as a namespace (corresponding to Kubernetes namespace). It provides basic isolation in terms of CISM workloads and objects segregation. The computational resource isolation is provided by fixing the amount of vCPU, RAM, and Storage that an application or CNF can use. Network isolation is provided using Openshift and Kubernetes networking policies and their Container Network Interface (CNI). Fig5 depicts isolation between two edge sub-slices.

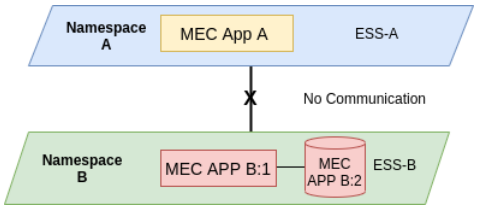

Fig. 5: Isolation between slices

E. Working

Northbound API exposed by ESSO can be used by SO or any other entity to perform LCM on ESST. A ESST goes through three different life cycle phases, creation phase, modification phase or deletion phase. These phases have four different stages,

[5]https://www.redhat.com/en/technologies/cloud-computing/openshift

- **On-boarding**: Gathering and reserving resources. In this stage, the application orchestrator gathers all the container images via CIM, reserves the computational and network resources required by each container image of each AppD. Defines required CISM objects needed for instantiation and stores them in the repository.
- **Instantiation**: Creates the CISM objects defined earlier and instantiates the application described by each AppD in order described by **appDInstantiationOrder**. Once the ESST is instantiated and if required, it requests the MEP for traffic redirection or DNS redirection.
- **Termination**: Gracefully deleting each application of the AppD in order as described by **appDInstantiationOrder**.
- **Off-boarding**: Removing the stored container images and un-reserve the computational and network resources for each application.

Fig 6 shows the phases and stages through which ESST goes during its LCM.

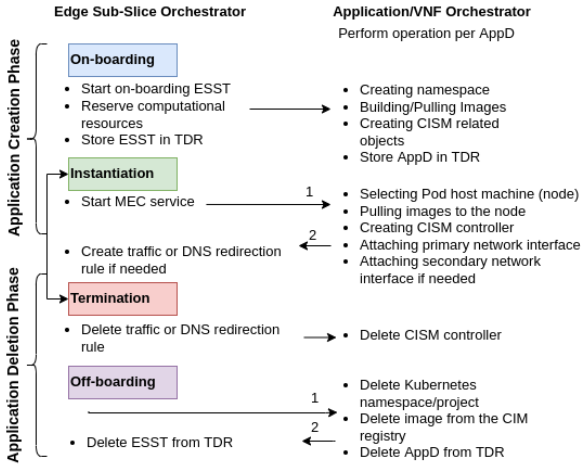

Fig. 6: LCM of ESST

ESSO via the Application Orchestrator provides the possibility to modify the computational resources required by an application or CNF. But it results in terminating the application and re-instantiating the new application. This behavior is because it is not possible to change the computational resources allocated to a container at run time. We consider that the application is able to preserve its state in a persistent volume or store it in a database before a graceful termination. The modification phase includes the termination and instantiation stages.

V. Performance Evaluation

The 5G trial facility deployed in Eurecom has a production-grade Openshift cluster with 7 worker nodes and 3 master nodes to run container-based VNFs. In total there are 320 CPU Cores and 624 GiB of RAM present in the cluster.

To analyze the resource consumption of the LeSO framework, we replicated a real trial scenario by creating four different edge slices using four ESST. Each ESST contains a different number of container images and AppDs. The container images of ESST ES1, ES2 and ES3 were already present in the cluster image repository. Hence their on-boarding time was shorter. One of the images of ESST ES4 was present in the public image repository, and the remaining were present locally in the cluster. Table I describes the time taken to create and delete each ESST. Table II describes the time spent on each stage of the LCM of the slice, the runtime of each edge slice was 30 mins.

ESST	Creation Time (s)	Deletion Time (s)	AppD	SwImage
ES1	20.42	10.32	1	6
ES2	15.26	10.32	1	1
ES3	30.52	15.29	2	6,1
ES4	50.43	20.4	3	1,1,1

TABLE I: Time spent (in seconds) in LCM of 4 slices

ESST	On-boarding Time (s)	Instantiation Time (s)	Termination Time (s)	Off-boarding Time (s)
ES1	15.26	5.16	5.17	5.15
ES2	5.16	10.1	5.16	5.16
ES3	20.17	10.35	10.1	5.19
ES4	25.31	25.12	15.24	5.16

TABLE II: Time spent (in seconds) in each stage of ESST LCM

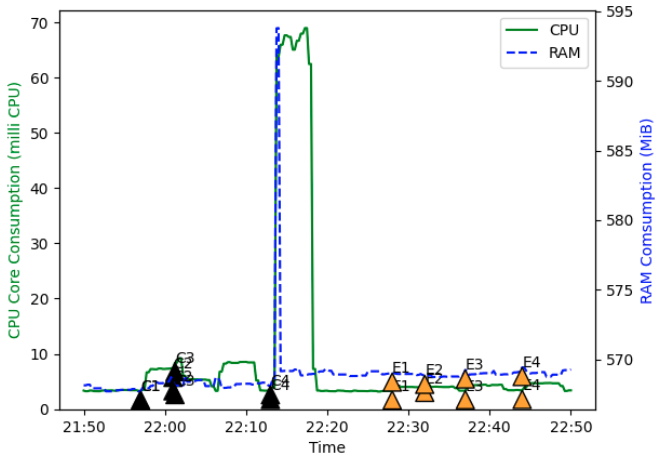

Fig. 7: Computational Resource Consumption for 4 Edge Slices

Fig 7 shows the computational resources consumed by the LeSO framework in a time span of 1 hour when the trial was conducted. CPU core consumption is in milli (m) CPU, 1000m CPU is one CPU core. Black triangles indicate the time at which the slice creation request was received, and orange triangle indicates the ending of the slice. Time span between CX and EX indicates the life span of a slice. All $X \epsilon [1, 4]$. The peak in CPU and RAM consumption is due to the use of CIM for pulling an image for ES4 for other slices the image was locally present. CIM is using Podman[6] for image management; hence its resource consumption behavior is not controlled by

[6]https://podman.io/

this framework. From the tables and the figure below analysis can be drawn,

- The composition of ESST affects the time spent in each LCM stage.
- Slice creation is a computationally expensive task than slice deletion
- Resource consumption before C1 and after 22:20 is the same. Hence, the framework does not consume resources until a new slice creation request or modification request is received

To analyze the multi-tenancy of the proposed framework, we created and deleted multiple slices at the same time. This allowed us to understand how the framework handles parallel requests. We started with creating 10 slices, and after 30 seconds of runtime, we deleted them. This pattern was continued for handling 10, 20 up to 50 edge sub-slices. All the slices used the same ESST A, and the container image was already present in the cluster image repository. We performed the same experiment 100 times for each data point, using Monte Carlo simulation. These data points were collected in a period of 4 days. Fig 8 only shows the creation time of these

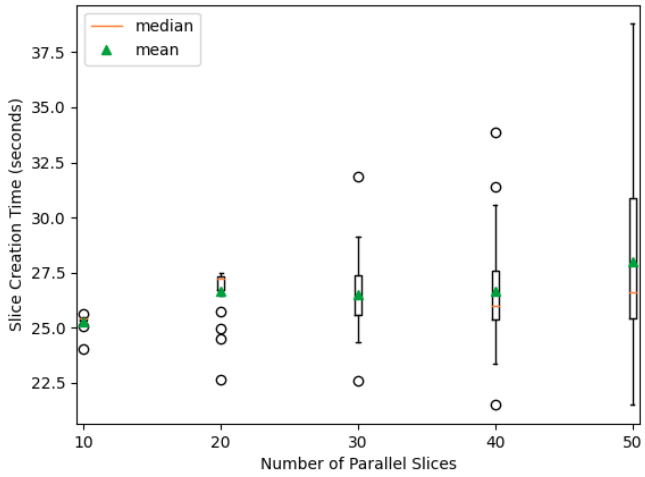

Fig. 8: Parallel Slice Creation Time

slices. The figure proves that the framework handles parallel requests and it immediately starts processing the request once received. Slice creation is the most time-consuming process where the CISM has to pull or build software images, schedule the CNF, attach a network interface and assign an IP address. The mean and median time needed to create a slice varies between 25 and 28 seconds on our Openshift cluster. The absolute value of creation time depends on the cluster design, its hardware configuration, and ESST design. To create 50 slices, the framework consumed 226m CPU Core and 1.19 GiB memory.

To summarize, our performance analysis LeSO framework requires 1 vCPU and 2 GiB of RAM for deployment and management of Edge Sub-Slices. Table III compares the resource requirement of LeSO, OSM and ONAP, referred from [12], [13]. It should be noted that other frameworks require much higher computational resources, and they do not provide the capability to instantiate MEC applications. Hence, the low resource consumption of LeSO framework makes it suitable to be deployed at the edge cloud and handle LCM of MEC applications.

Orchestrator	vCPU	Memory (GiB)
OSM-11	2	6
ONAP (Honolulu)	112	224
LeSO	1	2

TABLE III: Resource Requirement Comparison

VI. Conclusion

In this paper, we introduced LeSO a cloud-native orchestrator that handles LCM of edge slices. LeSO encloses a MEP element that allows dynamic deployment of edge slices. LeSO framework has a nominal resource consumption and requires 1 CPU and 2 GiB of RAM for installation. The multi-tenancy feature allows performing LCM on multiple slices at the same time. The ESST abstracts the platform and infrastructure-related information from the MEC Application providers and allows describing a cloud-native MEC Application which was not possible using the original AppD. In the future releases of LeSO, we will work on allowing the creation of partially isolated and shared slices. The source code of LeSO will be released soon as part of the OpenAirInterface (OAI) alliance.

References

[1] Multi-access Edge Computing (MEC); “Framework and Reference Architecture” ETSI GS MEC 003 V2.2.1 Dec 2020

[2] S. Dutta et al. “On-the-fly QoE-aware transcoding in the mobile edge”, IEEE Conference and Exhibition on Global Telecommunications (GLOBECOM) 2016, Washington DC, USA.

[3] S. Kekki et al., “MEC in 5G Networks,” ETSI, White Paper 28, June 2018.

[4] S. Arora et al, “Exposing radio network information in a MEC-in-NFV environment: the RNISaaS concept”, in Prof. of the 5th IEEE Conference on Network Softwarization, NetSoft 2019, Paris, France, June 24-28, 2019.

[5] I. Afolabi et al “Network Slicing and Softwarization: A Survey on Principles, Enabling Technologies, and Solutions,” in IEEE Communications Surveys & Tutorials, vol. 20, no. 3, pp. 2429-2453, thirdquarter 2018.

[6] Cloudnative.[Online].Available:https://github.com/cncf/toc/blob/master/DEFINITION.md

[7] 5geve.[Online]. Available:https://www.5g-eve.eu/

[8] Multi-access Edge Computing (MEC); MEC Management; Part 2: Application lifecycle, rules and requirements management,ETSI GS MEC 010-2 V2.1.1, Nov 2019

[9] S. Dräxler et al., “SONATA: Service programming and orchestration for virtualized software networks,” 2017 ICC Workshops, 2017, pp. 973-978, doi: 10.1109/ICCW.2017.7962785.

[10] A. Ksentini et al, “Toward Slicing-Enabled Multi-Access Edge Computing in 5G,” in IEEE Network, vol. 34, no. 2, pp. 99-105, March/April 2020, doi: 10.1109/MNET.001.1900261.

[11] Network Functions Virtualisation (NFV) Release 3; Architecture; “Report on the Enhancements of the NFV architecture towards Cloud-native and PaaS”, ETSI GR NFV-IFA 029 V3.3.1, Nov. 2019.

[12] Girma M. Yilma et al., “Benchmarking open source NFV MANO systems: OSM and ONAP”, Computer Communications, Volume 161, 2020, Pages 86-98, ISSN 0140-3664

[13] OSM.[Online].Available:https://osm.etsi.org/docs/user-guide/03-installing-osm.html#pre-requirements