Cloud Native Lightweight Slice Orchestration (CLiSO) Framework

Sagar Arora, Adlen Ksentini and Christian Bonnet

Eurecom, Communication Systems, Biot, 06410, France

ARTICLE INFO

Keywords:
Network Slice Orchestration
Cloud-native
Microservices
Containers

ABSTRACT

Cloud-native network functions are becoming promising for 5G and beyond networks. They provide the much needed agility and flexibility that was missing from virtual machines. Though, this paradigm shift of using cloud-native application design principles, containerization, microservices, high resilience, and on-demand scaling has created challenges for legacy orchestration systems. They were designed for handling virtual machine-based network functions. Indeed, network slice orchestration requires interaction with multiple technological domain orchestrators, access, transport, core network, and edge computing. The specifications and existing orchestrators are made on top of the legacy virtual machine based network function orchestration. Hence, this limitation constrains their approach to managing a cloud-native network function. To overcome their challenges, we propose a novel Cloud-native Lightweight Slice Orchestration (CLiSO) framework extending our previously proposed Lightweight edge Slice Orchestration (LeSO) framework. In addition, we present a technology-agnostic and deployment-oriented network slice template. To allow zero-touch management of network slices, our framework provides a concept of Domain Specific Handlers. The framework has been thoroughly evaluated via orchestrating OpenAirInterface container network functions on public and private cloud platforms.

1. Introduction

Network Function Virtualization (NFV) is the key enabler of Network slicing [1] in 5G and beyond networks. It enables hosting multiple communication services on top of the same physical infrastructure without compromising their Quality of Service (QoS). Handling a network slice that spreads across different technological domains, i.e., Radio Access Network (RAN) [2], [3], Transport Network, Core Network (CN), and Edge computing is a difficult task. Indeed, a network slice is divided into sub-slices, which are managed by domain-specific service orchestrators. These orchestrators translate domain-specific Service Level Objectives (SLOs) to resource-level objectives and forward them to resource orchestrators. The resource orchestrators manage the infrastructure for network functions. The job of a network slice orchestrator is to manage interactions with sub-slice/service orchestrators and deliver a network slice. Figure 1 shows multi-domain network slice orchestration.

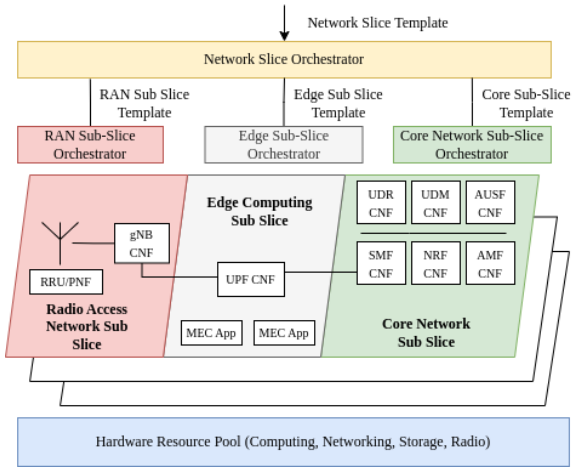

Figure 1: Multi-Domain Network Slice Orchestration

Sub-slice or service orchestrators are responsible for handling the network function's life cycle. The transition from virtual machine based Virtual Network Functions (VNF) to Container based Network Functions (CNF) has created new challenges for service orchestrators. One such challenge is following cloud-native application design principles, containerization, microservice, and on-demand scaling. Containerization is the process of packaging software in containers. It simplifies the software packaging, orchestration mechanism and reduces software deployment time with lower computational cost as compared to Virtual Machines (VM). All these benefits of using containers amplify when fused with cloud-native principles. Hence, the transition from VNF to CNF requires re-designing VNFs and service orchestrators. Whereas, the recent release of ETSI Network Function Virtualization (NFV) Management and Orchestration (MANO) [4] adds another layer of components to orchestrate containers alongside VMs. Rather than re-designing the service orchestration framework to fully support containers. This approach complicates the VNF descriptors and hides the simplicity of using containers. However, ETSI Multi-access Edge Computing (MEC) [5] specifications for orchestrating applications at the edge of the network, still do not clearly mention container-based MEC applications.

The well-known service orchestration frameworks presented in [6], ONAP and OSM followed the same path as ETSI-NFV MANO and added the support for CNF orchestration to their legacy orchestration mechanism. Leading to a complicated workflow and multiple redundant components

sagar.arora@eurecom.fr (S. Arora); adlen.ksentini@eurecom.fr (A. Ksentini); christian.bonnet@eurecom.fr (C. Bonnet)
ORCID(s): 0000-0003-0729-8260 (S. Arora); 0000-0001-6857-2679 (A. Ksentini)

in the service orchestration framework to support containers and VMs both. In addition, these orchestrators can not orchestrate MEC applications or provide MEC Platform (MEP) services. MEP is essential for hosting applications at the edge of the network. It allows re-directing user traffic to the edge application. These frameworks do not provide a clear mechanism for orchestrating isolated network slices and ONAP demands high computational and networking resources. Hence, this inability to orchestrate MEC applications, high resource consumption, complicated service orchestration workflow, and missing isolation between network slices inspired us to propose an end-to-end network slice orchestration framework based on cloud-native design principles.

Cloud-native Lightweight Slice Orchestration (CLiSO) framework is designed to handle Radio Access Network (RAN), Core Network, and MEC domains. The framework can be deployed in resource constraint environments due to its lean design and low resource consumption. The framework allows hosting CNFs on public, private, or hybrid clouds. Our proposed framework is an extension of our previous work, the Lightweight edge Slice Orchestration (LeSO) framework [7]. The previous framework only focused on the MEC domain, whereas the new framework adds additional support for RAN and Core Network domains.

On the other hand, Zero-touch Service Management (ZSM) [8] plays an important role in enabling self-managed services. ZSM allows each service orchestrator to perform a closed-loop automation and heal their services in case of errors. Existing orchestrators or ETSI ZSM specification provides a notion of ZSM via state-of-the-art machine learning and artificial intelligence algorithms. Whereas, such a solution would completely rely on the intelligence of the service orchestrator and could result in a single point of failure.

To address this challenge our second contribution in this paper is a proposal for Domain Specific Handlers (DSHs), which is a management network function designed to manage the life cycle of one or many network functions via communicating with the service orchestrator's Application Programming Interfaces (APIs). Service orchestrators deploy DSH as a CNF and are responsible for managing DSH's health. However, DSH communicates with service orchestrators using a dedicated interface to manage the health and resource consumption of its network functions. This will allow offloading sub-slice operational management from service orchestrators to DSH. DSH may contain vendor-specific logic to handle its network functions. This provides the freedom to customize network function life cycle management rather than relying on the service orchestrator's generic mechanism to handle all the network functions.

Finally, management of network slices enabling mission-critical services requires dynamic management of infrastructure resources. For example, a fleet of drones acting as edge servers and hosting a mission-critical service. Our proposed slicing framework provides an interface to dynamically manage infrastructure resources. It allows a container-based Virtual Infrastructure Manager (VIM) to offer its resources to the hardware resource pool managed by the resource orchestrator. Facilitating on-demand life cycle handling of mission-critical services. To summarize our paper's contributions:

* We proposed an end-to-end Cloud-native Lightweight Network Slice Orchestration (CLiSO) framework capable of orchestrating CNFs and MEC applications on public, private, and hybrid cloud and Physical Network Functions (PNF). CLiSO allows dynamic management of infrastructure.

* We presented a concept of Domain Specific Handlers (DSHs) to allow Zero-touch Service Management of sub-slices.

* We proposed a Network Slice Template (NST) based on a modified version of the ETSI Physical and Virtual Network Function Descriptor (PNFD and VNFD) and MEC Application Descriptor (AppD).

Table 1 summarizes the abbreviations we used in this paper.

2. Background

In this section, we will explain the state of the art and terminologies used in the rest of the paper. It forms the foundation for the other sections.

2.1. 3GPP Approach Towards Managing Network Slices

Network slicing depends on resource virtualization and the orchestration mechanism depends on whether the network functions are, realized as physical machines, virtual machines, containers, or a combination of both. 3GPP SA5 group in Technical Specification 28.53X (X in 0,1,2,3) defines the network slice management and automation mechanism. The 3GPP approach has two key concepts Network Slice Instance (NSI) and Network Slice Subnet Instance (NSSI). A NSSI is analogous to a network sub-slice. The 3GPP defines the following management functions related to NSI management, listed below in the order corresponding to their hierarchy:

* Communication Service Management Function (CSMF): Responsible for translating the communication service-related requirement(s) to network slice-related requirement(s). It forwards the request to Network Slice Management Function (NSMF) to manage the life cycle of a NSI.

* Network Slice Management Function (NSMF): Manages the life cycle and monitors the KPIs of NSIs. It derives network slice subnet-related requirements from the network slice-related requirements. NSMF communicates with several NSSMFs, one for each domain to manage the life cycle of the Network Slice Subnet Instance (NSSI).

Table 1
Summary of Abbreviations

Notation	Definition
AMF	Access and Mobility Management Function
AUSF	Authentication Server Function
AppD	Application Descriptor
CISM	Container Infrastructure Service Management
CNF	Cloud-native Network Function
CSSO	Core Sub-Slice Orchestrator
CU-CP	Central Unit-Control Plane
CU-UP	Central Unit-User Plane
DU	Distributed Unit
DSH	Domain Specific Handlers
ESSO	Edge Sub Slice Orchestrator
GST	Generic Network Slice Template
KPI	Key Performance Index
MEC	Multi-access Edge Computing
MEP	MEC Platform
MEO	MEC Orchestrator
NFV	Network Function Virtualization
NRF	Network Registry Function
NSD	Network Service Descriptor
NST	Network Slice Template
NSO	Network Slice Orchestrator
NSSO	Network Sub Slice Orchestrator
PNFD	Physical Network Function Descriptor
RNIS	Radio Network Information Service
RSSO	RAN Sub-Slice Orchestrator
RU	Radio Unit
SLO	Service Level Objectives
UDM	Unified Data Management Function
UDR	Unified Data Repository
UPF	User Plane Function
VIM	Virtual Infrastructure Manager
VNF	Virtual Network Function
VNFD	Virtual Network Function Descriptor
ZSM	Zero-touch Service Management

* Network Slice Subnet Management Function (NSSMF): Manages the life cycle and monitors the KPIs of NSSIs. There can be a dedicated NSSMF for each functional domain, access network, transport network, edge network, and core network.

A NSI has four phases preparation, commissioning, operation, and decommissioning. In the preparation phase, NSI does not exist, the phase includes calculating instance requirements, on-boarding needed entities, reserving resources, and preparing the environment. NSI's life cycle stage includes the commissioning, operation, and decommissioning phase. In these phases, NSI goes through different stages creation, activation, modification, de-activation, and termination. A NSI is described using a Network Slice Template (NST). GSMA defines a Generic network Slice Template (GST)[9] to describe slice behavior, required quality of service, network functions required by the slice, etc. It does not focus on the deployment aspect of a NSI, the hardware requirement of network functions, their configuration, and links between them, etc. In [10] authors have proposed to use ETSI NFV Orchestrator and ETSI MEO as NSSMF for core and edge domains respectively.

2.2. Containerization Support in ETSI NFV-MANO

ETSI NFV network service can comprise one or more physical or virtual network functions or a combination of both. The life cycle of a network service is managed by ETSI NFV Orchestrator (NFVO). ETSI group report NFV-EVE 012 V3.1.1, explains the relationship between 3GPP proposed NSI and a network service. A network service is a resource-centric view of a network slice if a NSI contains at least one virtual network function. The group report highlights how 3GPP slice management functions can be integrated with ETSI NFVO using *os-Ma-nfvo* interface. Figure 2 shows that interface. The legacy NFVO architecture was not capable to manage modern CNFs. ETSI GS NFV 006 v4.4.1 proposed new functional blocks shown in Figure 2 for managing the life cycle of CNFs. It should be noted ETSI refers CNFs as container-based VNFs. In this paper, we have used both the terminologies, CNF, or container-based VNF.

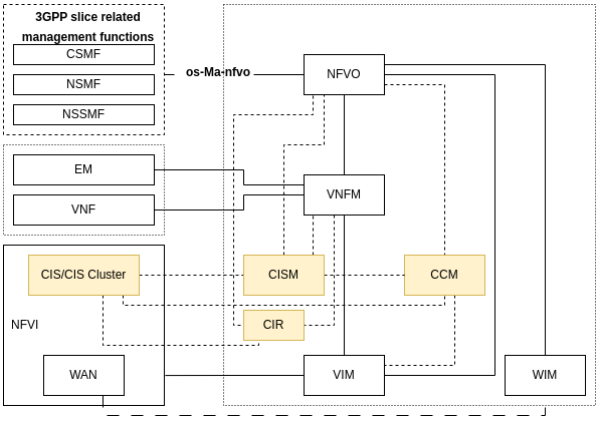

Figure 2: NFV-MANO architectural framework with support for containers

For simplicity, we only highlighted the newly added blocks for container management and used dotted lines for connection. We did not mention the names of all the interfaces to avoid confusion. Container Infrastructure Service (CIS) provides run-time infrastructural dependencies, computational, storage, and networking resources for one or more containerization technologies. It can be considered the cloud-native equivalent of a virtual-machine hypervisor. Hypervisors provide infrastructure to host virtual machines. Container Infrastructure Service Management (CISM) manages containers executed by CIS. It is responsible for container deployment, monitoring, and life cycle management. CIS Cluster Management (CCM) is responsible for managing the life cycle of CISM.

Network services are described using a Network Service Descriptor (NSD). A NSD contains multiple Virtual Network Function Descriptors (VNFDs). The new specifications contain special fields such as *osContainerDesc* to

include container description. The new fields were designed to deploy CNFs on Kubernetes-backed infrastructure. Each VNFD inside a NSD can have multiple containers described via the *osContainerDesc* field, one for each container. The number of containers depends on the VNFD provider. However, the VNFD template still lacks some fields needed to configure the security, networking, and configuration of network functions in a cloud-native environment. It should be noted that the current specification has a tight coupling with Kubernetes, a CISM. Apart from NSD, ETSI NFVO requires other packages that are needed for orchestrating network functions. These packages rely on helm-charts[1] which are tightly coupled with Kubernetes.

2.3. Edge Sub Slice Orchestrator

In [7], we proposed a Lightweight edge Slice Orchestration (LeSO) framework to handle an edge domain sub-slice. The framework proposed an Edge Sub Slice Orchestrator (ESSO) inspired by ETSI MEC Orchestrator (MEO). MEO is responsible to orchestrate MEC applications defined using Application Descriptor (AppD). It communicates with the MEC Platform (MEP) to provide services like service registry, service discovery, Radio Network Information Service (RNIS), DNS redirection, and traffic redirection to MEC applications. The LeSO framework is a crucial component of our proposed end-to-end network slicing framework. It handles the edge domain.

2.4. Kubernetes a CISM

Kubernetes is an industrial de-facto standard for container orchestration, inspired by Google's Borg platform [11]. ETSI NFV-MANO standard mentions Kubernetes as one of the possible container orchestration platforms, similarly, other service orchestrators use Kubernetes. Today Kubernetes has an important role in realizing a network slice built on top of CNFs. Kubernetes is capable of managing multiple pods spread across a cluster of nodes. A Kubernetes Pod is a group of co-located containers. It is the smallest entity that can be scheduled by the Kubernetes scheduler. The containers inside a Pod share the same network namespace, separate computational resources, and can have common storage. Pods are designed to run multiple co-located processes with a degree of isolation. Whereas, containers are designed to host a single process. The containers inside a pod communicate with each other using the local network (loopback interface), shared memory IPC (inter-process calls), and shared volumes if the volumes are common. Figure 3 shows the design of a Kubernetes Pod. It should be noted Kubernetes is an example of CISM but there can be alternatives; other alternatives are out of the scope of this paper.

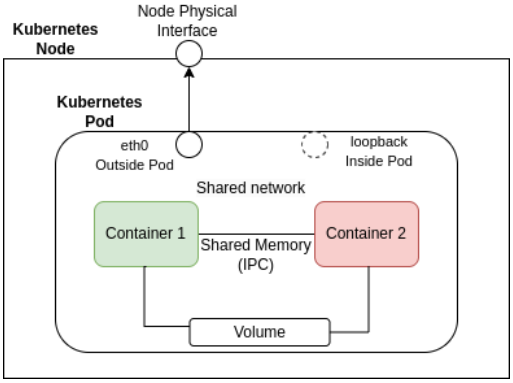

Figure 3: Kubernetes Pod

3. Related Work and Motivation

The existing slice or service orchestrators and their network slice templates or service descriptors were designed to orchestrate VNFs. Later they adapted their architecture and service descriptors to orchestrate CNFs. The service descriptors of the orchestrators are either based on standard ETSI NSD or a tailored NSD. The orchestrators are capable of placing the applications at the edge, but they do not provide the capabilities of MEP to edge applications. Below are some of the known slice/service orchestrators,

* Open Source MANO (OSM): A network service orchestrator proposed by ETSI as a reference design based on ETSI NFV standard. 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. For supporting MEC applications the ETSI MEC framework proposes an interface with NFVO. Starting OSM version 5 network slices can be orchestrated using the OSM network slice template. The network slice template contains several ETSI NSDs. The template does not clearly allow resource isolation. Apart from NSD, the orchestrator requires helm-charts or juju-charms[2] to describe the CNF configuration, required software images, and computation resources.

* Open Network Automation Platform (ONAP): Designed to manage VM-based VNFs, and with the recent release, it is following cloud-native principles to orchestrate CNFs. ONAP's complicated architecture with a large number of components results in high resource consumption. Frankfurt release of ONAP introduced the functionality of network slice orchestration using Topology and Orchestration Specification for Cloud Applications (TOSCA) template. The network slice template is convoluted and it is difficult to define resource isolation. The complex realization of ONAP demands the involvement of a big team in taking care of service orchestration. The high resource consumption makes it unsuitable to deploy in a resource constraint environment.

* 5G Transformer Service Orchestrator (5GT-SO) [12]: 5G transformer framework proposed a vertical slicer as the

[1] https://helm.sh/docs/topics/charts

[2] https://charmhub.io

gateway for different verticals to instantiate their network services. The 5GT-SO is built on top of OSM and Cloudify but only supports VNFs.

* 5G Growth [13]: A network slice orchestration framework for verticals is an extension of the 5G Transformer framework. The framework and the proposed VNFD were designed for orchestrating VM based VNFs. Though it is capable of orchestrating container-based VNF via using a Kubernetes Plugin. Apart from this, the framework can orchestrate PNF via a Radio plugin and MEC applications via a MEC plugin. There is no detailed information on the capabilities of the radio plugin and MEC plugin.

* 5G SONATA [14]: A service orchestrator capable of orchestrating VNFs and CNFs. However, it does not orchestrate MEC applications. Making it unsuitable to orchestrate an end-to-end network slice.

* OptiMized Edge Slice OrchestratioN (MESON) [15]: This framework is designed for optimized and secure cross slice communication within edge clouds. The framework is inspired by ETSI-MANO architecture and uses MEC AppD to enable cross slice communication. The framework is mainly designed for edge slices using VM-based MEC applications. The authors does not mention other aspects of MEC AppD, traffic redirection, and DNS-based redirection. They mostly focus on service discovery and service registry aspects of the MEP.

* JOX [16]: Is an event-driven orchestrator for 5g network slicing. The framework relies on Juju-charms based orchestration. It was not designed to orchestrate microservices based CNFs or MEC applications. The framework mentions integration with a MEC agent but there are no further details on the capabilities or functioning of the MEC agents.

* Open Baton [17]: One of the initial network service orchestrators highly inspired by ETSI MANO architecture. The framework only supports VM-based VNFs and does not offer MEP capabilities.

ONAP and OSM both have their own tailored NSDs and NSTs and apart from that they need VIM-specific packages like helm-charts and juju-charms. This forces the network function provider to be infrastructure and platform aware. OSM has a tight coupling with Juju-charms and Ubuntu-based Linux. Besides these orchestrators, Google proposed Nephio[3]. It is integrated with Kubernetes and it requires the users to be aware of Kubernetes. The project is at an early stage and there is nothing mentioned about network slicing. The authors of [18] proposed SliMANO, an expandable framework for the management and orchestration of End-to-end Network Slices. Their proposed framework interacts with existing MANO frameworks, RAN controllers, and SDN controllers. Further, the paper does not mention interactions with a MEC orchestrator. SliMANO is restricted by the capabilities of the frameworks used for orchestrating different domains i.e., if the domain orchestrator supports container-based VNFs then it will be able to orchestrate them else it can not.

The authors of [19] proposed a MEC application slicing concept via a proof of concept framework. They proposed a MEC Application Slice Subnet Descriptor to describe container-based MEC applications of a MEC slice. Their proposed descriptor and framework highly rely on helm-charts to describe application deployment/configuration-related information. Such an approach requires application providers to be infrastructure aware.

The core and edge domains both will have container-based network functions or applications. In fact, according to Open RAN Alliance (O-RAN)[4] architecture it is possible that some of the access network functions for example CU-CP and CU-UP can be deployed as CNFs. ONAP addresses the challenges of orchestrating RAN network functions. However, its high resource consumption restricts its usage to a limited group of users. Our proposed work is motivated by the absence of a:

* lean ease to use lightweight end-to-end slicing orchestration framework. A framework that can simultaneously orchestrate ran, edge, and core domain slices.

* framework that follows cloud-native principles to orchestrate and manage CNFs on public, private, and hybrid clouds.

* framework that abstracts the platform and infrastructure-related information from NST providers.

4. Cloud-native Lightweight Slice Orchestration (CLiSO) Framework

This section will introduce the architecture of the CLiSO framework. The roles and functioning of its different components. The concept of Domain Slice Handler(s) and how they can communicate with sub-slice orchestrators to manage their sub-slice(s). The section also presents the skeleton of the proposed cloud native NST and its different fields and their parameters.

4.1. CLiSO Framework Architecture

The proposed framework has a hierarchical architecture starting from the top, Network Slice Orchestrator (NSO), Network Sub-Slice Orchestrator (NSSO) (ran, edge, and core domain), CISM, and Container Image Registry (CIR). To compare our framework with 3GPP proposed network slice management framework, NSO is analogous to NSMF and NSSO is analogous to NSSMF. Apart from these components, there is a template registry and a global Domain Name Server (DNS); these components are not shown in Figure 4. The transport domain is out of the scope of the proposed slice orchestration framework. The different layers are highlighted to distinguish the role of the components.

[3] https://nephio.org

[4] https://www.o-ran.org

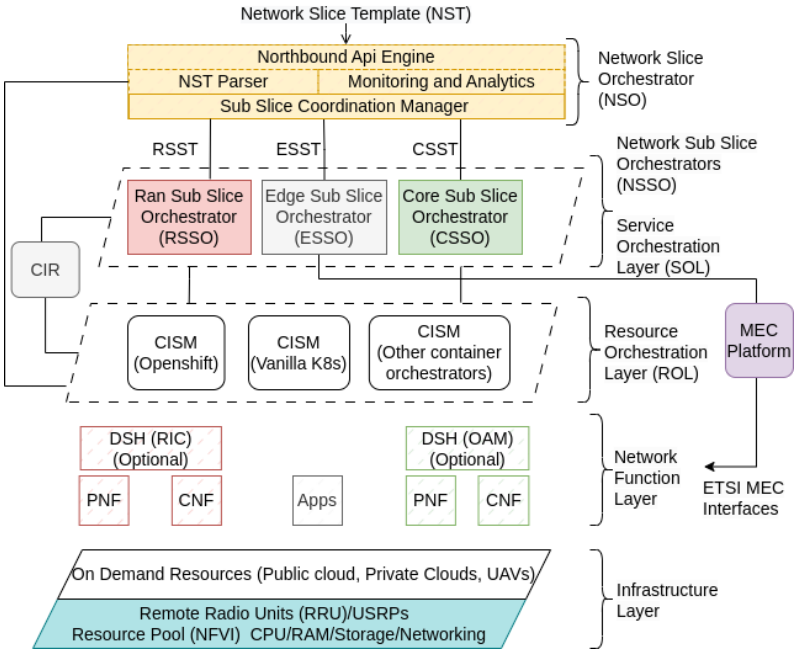

Figure 4: Proposed Cloud-native Lightweight Slice Orchestration Framework

The Service Orchestration Layer is responsible for translating SLOs to Resource Level Objectives and coordinating with resource controllers. Whereas Resource Orchestration Layer manages the resources via communicating with the underlying resource pool. The purpose of each component of the framework is described below,

* Network Slice Orchestrator (NSO): It is responsible for creating Network Sub Slice Templates (NSST) for different domains and coordinating the life cycle management of sub-slices. It receives the monitoring data from different sub-slice orchestrators to extract slice-level monitoring information.

* Network Sub-Slice Orchestrators (NSSO) or Service Orchestrators: They are responsible for handling sub-slices of their respective domains and collecting monitoring data to share with NSO. They expose APIs for DSH to consume slice-specific KPIs,

  - Ran Sub-Slice Orchestrator (RSSO): Handles the life cycle of ran sub-slices composed of PNFs or CNFs. It communicates with CISM to manage the CNFs and PNFs. To manage each PNF, RSSO creates ephemeral containers with short lifespans when needed. CISMs managed by RSSO are capable of handling radio access network functions, RU, DU, CU or CU-CP, CU-UP.
  - Edge Sub-Slice Orchestrator (ESSO): Handles the life cycle of edge sub-slices composed of MEC Applications. It coordinates with the MEC Platform to provide the necessary services like traffic redirection, DNS-based redirection, or RNIS to the MEC Apps. ESSO only handles container-based MEC Apps. MEC Platform is also part of the proposed framework. The detailed functioning of ESSO is described in [7].
  - Core Sub-Slice Orchestrator (CSSO): CSSO handles the life cycle of core network sub-slices composed of PNFs or CNFs. It communicates with CISM to manage PNFs and CNFs.

* CISM: As described in the previous sections, it is responsible for orchestrating containers. It creates the necessary communication links between network functions to deliver the required slice behavior. NSSOs communicate with CISM via a CISM agent hosted on the CISM platform. The framework is capable of orchestrating containers on different distributions of Kubernetes, Openshift, Vanilla Kubernetes (also known as K8s), and K3s[5]. A new distribution can be supported by creating a plugin. The Vanilla Kubernetes plugin can be used for orchestrating network functions on the public cloud Kubernetes distribution. It has most of the required functionalities.

* Container Image Registry (CIR): Manages and stores Open Container Initiative (OCI) format container images. It is capable of pulling images from public or private repositories and building images from source code.

* Template Registry: Database to store all the Network Slice Template (NSTs) and Network Sub-Slice Templates (NSSTs)

* Global Domain Name Server: Allows the network functions hosted on different CISM infrastructures to resolve

[5] https://k3s.io

the fully qualified domain name of other network functions belonging to the same slice.

Our proposed framework follows cloud-native design principles, microservices architecture, and supporting on-demand scaling of the components. Each component exposes a REST API. It is possible to downsize the framework if required and deploy only selected components. For example, to manage only core sub-slices, it is required to deploy only CSSO. RSSO and ESSO are optional.

The framework proposes dynamic resource management based on the CISM registration mechanism. Unlike ETSI NFV, CLiSO does not follow the concept of fixed infrastructure. CISM is added to the individual NSSO CISM repository via NSO. A CISM agent should be running on the CISM platform. It is registered using the CISM registration template shown in figure 5. The fields in underline are required fields and others are optional. *listOfSubnet* allows CISM to describe its subnets and this information is used to allocate IP-address to network functions. *dpdk* Boolean value is to specify if the interface supports data plane acceleration technology, Data Plane Development Kit (DPDK). *annotations* field allows mentioning resources apart from CPU and RAM that CISM exposes, for example, hugepages.

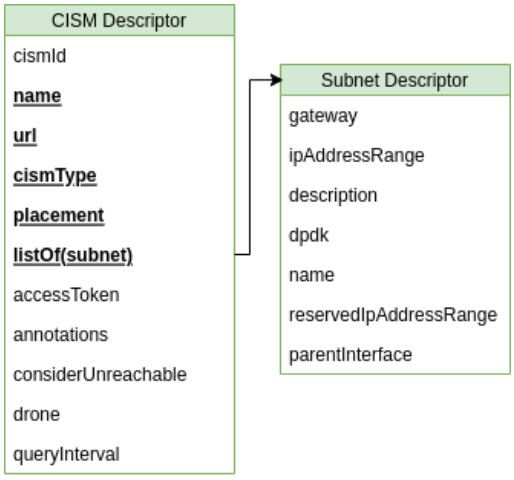

Figure 5: Proposed CISM Descriptor

Registered CISM shares regular heartbeats with their respective handlers in NSSOs. This allows NSSOs to be infrastructure aware and make decisions when there is a problem at the infrastructure level. The field *drone* allows registering UAV-based CISM. This field instructs the CISM agent not to remove the network slice when a heartbeat is not being shared with NSSO. CISM can manage PNFs by using ephemeral containers, short-lived containers which can communicate with PNFs and perform any required action based on the instructions provided in NSST.

MEC platforms are registered dynamically only to ESSO via NSO. ESSO directly communicates with MEC platforms and gets the list of services hosted at the MEP. Figure 6 shows the MEP descriptor. The fields in underline are required fields and others are optional. ESSO has the capability to query the MEP regularly for heartbeats and the list of hosted services.

Figure 6: Proposed MEP Descriptor

4.2. Domain Specific Handler (DSH)

Domain Specific Handlers are management network functions responsible for handling the life cycle of one or many network functions belonging to the same slice. They are analogous to an Element Management System (EMS) but designed with an ability to communicate with the CLiSO framework to negotiate the SLOs of the slice. A slice may have multiple DSHs depending on the NST provider. Their administrative domain is restrictive to their slice. Few key points about DSH:

* They get life cycle notifications of all the network functions from NSSO. They have a direct communication link with all the network functions of their administrative domain. They fetch network function level KPIs and push them to the CLiSO monitoring engine.

* They can subscribe to infrastructure-level KPIs, computation, and networking resources consumed by the network functions. Based on this information, they can request the NSSO to increase or decrease the resources consumed by the CNF or MEC application.

* They can request the NSSO to upgrade the container software image of a CNF instance.

DSH can provide ZSM if they collect KPIs from network functions and the CLiSO framework to handle the life cycle of their managed network slice. DSH may use Artificial Intelligence and Machine Learning algorithms to perform ZSM. CLiSO Framework does not have an inbuilt DSH as they are external to the proposed framework. There are no standards around designing an EMS similarly nor for DSH. But to communicate with the NSSO they should have a dedicated interface. DSH provides the freedom to network function providers or vendors to manage their network functions rather than using NSO's generic management algorithm.

Figure 4 shows some examples of network functions that can be used as DSH if appropriate APIs are implemented to communicate with CLiSO. 1) RAN Intelligent Controller (RIC) that manages RAN network functions via analyzing

their and infrastructure level KPIs from CLiSO. 2) Operations And Maintenance (OAM) that manages the life cycle of core network functions via subscribing to notifications from Access and Mobility Management Function (AMF), Session Management Function (SMF) and Network Data Analytics Function (NWDAF) and consuming infrastructure level KPIs from CLiSO framework. However, DSHs are not mandatory components of a sub-slice. If network functions have a self-management mechanism, they can communicate with NSSOs to manage their own resource consumption.

4.3. Network Slice Template

Our proposed network slice template in figure 7 is deployment oriented and contains network function deployment specific information. We consider that slice or sub-slice level SLOs can be translated to network function's configuration. The proposed NST contains dedicated sections to define Core, Edge, and RAN sub-slices. NSO accepts a NST package in tarball format. This allows providing dedicated Yet Another Markup Language (YAML) files for each sub-slice template and any additional files like scripts or configuration files required by the PNF, CNF, or MEC applications. Apart from the network slice template our proposed framework does not require any additional deployment-related packages like helm-charts or juju-charms as required by OSM and ONAP. To stay aligned with ETSI proposed virtual network function descriptor instead of using CNF Descriptor (CNFD) we use VNF Descriptor (VNFD) in our proposed NST. Below is the description for some of the fields of NST:

* *templateId*: A unique identifier for an onboarded slice. If the field is present for a slice then instead of creating a new slice the orchestrator will update the onboarded slice.
* *metadata*: This field contains only slice name as a mandatory parameter and other parameters are optional, annotations and order in which sub-slices should be instantiated.
* *commonData*: This field contains data that is common to all the sub-slices and it will be provided to all the sub-slices of a slice. It can contain any user-defined information and it will be forwarded to the network functions of a sub-slice. For example, this field can contain SLO information and DSH can use this information to tune the configuration of other network functions to achieve the required SLO.
* *s-nssai*: single network slice selection assistance information is a unique network slice identifier defined by 3GPP. It contains Slice Service Type (SST), a mandatory parameter to identify slice characteristics, and an optional parameter Service Differentiator (SD).
* *listOfRegions*: This field allows defining the regions where the network slice will be created. The regions are defined in the CISM descriptor. A region can be the name of a city, state, or custom value defined when a new CISM is added to the repository.
* When providing a new NST to the orchestrator, one of the sub-slices must be defined. If the sub-slice unique identifier, *coreSliceId*, *ranSliceId* or *edgeSliceId* is provided in the respective section then the orchestrator will automatically fetch the sub-slice template from the template registry. It is possible to define the subSlice template in a separate YAML file and provide its relative location to the package.

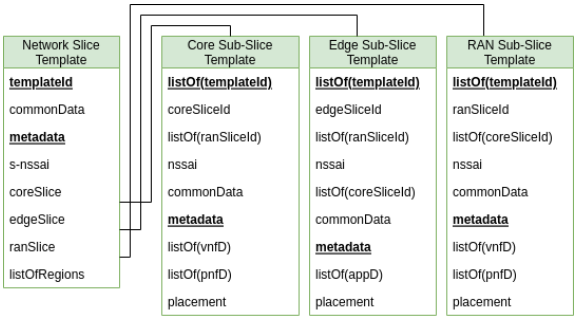

Figure 7: Proposed Network Slice Template

All the sub-slice templates have a similar design,

* They all contain a list of NST *templateId*, as a sub-slice can be shared among multiple slices.
* The unique sub-slice identifiers *coreSliceId*, *ranSliceId* or *edgeSliceId* are only present when the sub-slice is successfully on-boarded. If the field is present in the sub-slice template the sub-slice orchestrator will update the on-boarded sub-slice instead of on-boarding a new template.
* The *placement* field allows specifying the placement of the sub-slice network functions in a region. This can be the unique identifier of a CISM or the position tag, for example, FAR EDGE, ASSOCIATED EDGE, or CENTRAL CLOUD. These tags are customized and can be defined in the CISM descriptor. This placement will be common for all the CNFs of a sub-slice. Unless the VNF or MEC App descriptors also have the *placement* field and specifics the *cismId*. The placement field can be manually added in the NST, if not then NSO will automatically instruct the domain orchestrators about the placement.
* coreSlice contains a list of *ranSliceIds* using the coreSlice. Similarly, ranSlice contains a list of *coreSliceIds* it is using. edgeSlice contains a list of *coreSliceId* and *ranSliceId* through which the edgeSlice is reachable. This mapping is used to provide security isolation among sub-slices. For example, only allowed *ranSliceIds* can use the coreSlice.
* Edge sub-slice template is restricted to defining only MEC applications.
* *nssai* is a list of *s-nssai*. It allows enabling sharing of sub-slices with sub-slices of different network slices. This parameter is also used to provide isolation among different slices.

The vnfD, pnfD, and appD used by coreSlice, ranSlice, and edgeSlice templates are modified versions of ETSI NFV VNF Descriptor (VNFD), PNF Descriptor (PNFD) and ETSI MEC Application Descriptor (AppD). However, the recent version of VNFD allows describing containers using *osContainerDesc* field but there is no simple possibility to define the container's exposed ports, subnets it should be connected to, initial startup configuration, and placement-related information of VNF. To overcome such shortcomings we have proposed a modified VNFD and PNFD in figure 8. It should be noted that we have only highlighted the fields added or modified by us.

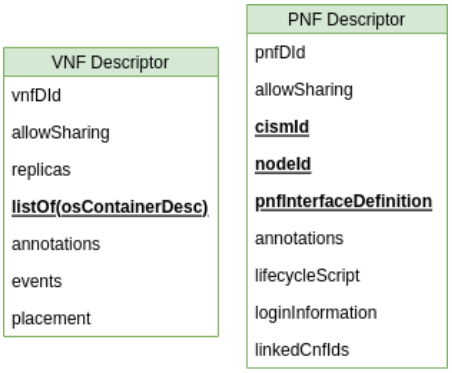

Figure 8: Proposed VNF and PNF Descriptor

* *vnfDId* and *pnfDId*: Are unique identifiers allocated by CISM when the network functions are already on-boarded. If already present, CISM will update the on-boarded descriptor rather than on-boarding a new descriptor.

* *osContainerDesc*: Describes the container images required to deploy the CNF. It contains several sub-fields to allow describing computational resources required and their limit, container image location, storage-related information, and monitoring parameter to be tracked for this osContainer. We extended the model to add information related to ports exposed by the container, the possibility to pass configuration to containers, startup command in case the container wants some initial command, liveliness to understand network function is always running, and readiness probe to know when the network function is ready to use, etc.

* *replicas*: Number of replicas of the CNF. The CNF should support load balancing between different replicas.

* *events*: Each CNF can subscribe to events published by CSSO or RSSO. This field is mostly relevant for DSH. The events give information about all network functions of the slice.

* *placement*: If mentioned it will supersede the placement defined in the sub-slice template.

* *cismId*: Mandatory field for PNF to define its location, and in case PNF is only accessible via one of the nodes of the CISM then it is mandatory to define *nodeId*.

* *pnfInterfaceDefination*: It allows describing the interfaces which are connected to the PNF and their IP-addresses.

* *linkedCnfIds*: This field defines the list of CNFs that will use the PNF. This will allow deciding the placement of CNFs to enable communication with their PNF. For example, if 5G New Radio DU will be placed on the node through which it can connect to Radio Unit (RU).

* *lifecycleScript*: A relative location or URL of a script to execute in the PNF at the time of on-boarding. It is executed via a short-lived container instantiated in the same network or host from where PNF is reachable.

* *annotations*: This field is to consume the resources exposed by CISM.

We have slightly updated the application descriptor, appD as described in [7]. We have replaced *swImageDescriptor* field with *osContainerDesc* as it is more relevant according to the ETSI NFV standards. ETSI MEC appD standards are still not adapted for container-based VNFs or CNFs.

4.4. Isolation between slices

Isolation between network slices is important for security concerns while providing the required Quality of Service (QoS) to the network slice. We have divided isolation into two categories, first resource isolation, and second communication isolation.

1. Resource Isolation: Isolation at the level of infrastructure resources used by a CNF. In the *osContainerDesc* field, it is mandatory to define the required CPU and RAM needed by the container and their limits. The CISM will always guarantee that these resources are allocated to the container.

2. Communication Isolation: For each container of CNF in *osContainerDesc* there is a sub-field *ports* to mention the exposed ports of the container and the network in which they are exposed. Outside this network, the ports will not be reachable. *nssai* field controls the communication between sub-slices of different slices. A sub-slice will only be able to communicate with slices defined in *nssai* else all the outgoing traffic to other slices will be rejected. Each instantiated CNF has an infrastructure-level container. This container guards the ingress and egress traffic of the CNF. An example of a tool that can manage resource isolation is Linux iptable rules.

4.5. Monitoring and Logging Capability of the Framework

To keep the framework lightweight, we only expose a limited set of KPIs using the inbuilt capability of the CISM. The framework provides CPU and RAM consumed by each network function of a slice. The PNF or their linked management CNFs can push their resource consumption to the monitoring engine. The framework provides the standard

output logs of each CNF of the slice. The PNF or their linked VNF have to push their logs to the logging engine. Our proposed framework can be plugged into the monitoring framework proposed by the authors in [20] to get elaborated KPIs related to each network function.

4.6. Functioning of the Framework

A NST template defines a network slice instance. Each network slice instance has four crucial phases,

1. Preparation phase: It includes calculating the requirement of the network slice instance, on-boarding the required container image, reserving the required resources at the CISM level, and creating CISM deployment-specific definitions.

2. Commissioning phase: After the successful onboarding of the slice network functions, the network functions are instantiated and in the case of MEC applications if required, ESSO communicates with MEP to provide requested services. At this phase, the slice is ready and operational. The KPI collection can be started.

3. Operational phase: The slice instance can serve its consumers and KPI can be monitored. At this phase, it is possible to use the output of the monitoring engine to update the slice resource consumption.

4. De-commissioning phase: The slice instance will be terminated, all the reserved resources will be released and the container images will be off-boarded.

NSO exposes a northbound REST API to allow CSMF to request the creation of a network slice instance described using NST. NSO using the NST parser disintegrate the NST in different NSSTs based on the domains defined in NST. If the *listOfRegions* is defined in the NST and there is no *placement* field in NSST, NSO will calculate the placement of the sub-slice and add the *placement* field in the NSST. NSO contains the position and load-related information of each CISM. Currently, based on the targeted domain NSO adds the position tag of the CISM as described in section 4.3.

Further, the sub-slice coordination manager forwards these NSSTs to their respective NSSO. The NSSOs/domain orchestrators can simultaneously communicate with multiple CISM agents spread across different regions or in the same region to create the sub-slice. NSSOs read the *placement* field and if it is a position tag and there is more than one CISM. The NSSOs decide the final CISM based on certain factors such as:

- The total requested CPU, memory, and storage by CNFs or MEC Applications of the NSST.

- The total limit of CPU, memory, and storage that CNFs or MEC applications can utilize

- Requested ports to expose for the CNFs or MEC applications. NSSO verifies that CISM can expose the requested ports.

- The instantaneous load of the CISM (number of instantiated pods, available CPU, memory, and storage).

This placement approach may have limitations in case of large numbers of CISM associated with a position tag and located in the same region. Then different NSSOs can not select the appropriate CISM that will satisfy the slice SLOs. In this scenario, NSO can take advantage of its global position to provide the CISM for each sub-slice. This solution is not integrated into the current framework it needs further investigation.

If the VNFD or AppD defines the placement of the CNF or MEC application then it will supersede the placement value defined in NSST. Domain orchestrators add an infrastructure level container based on the *nssai* field of the NSST to enable communication isolation and translate the PNFD, VNFD, and appD to CISM level definition.

A CNF or MEC application defined using a VNFD or AppD respectively, corresponds to a Kubernetes Pod. All container images defined via *osContainerDesc* map to one Pod. In situations when CNF or MEC Application design does not require close placement of all their components or they require *replicas* for a particular component defined via *osContainerDesc* then such CNF or MEC applications can be defined using multiple VNFD or AppDs. Each VNFD/AppD may contain single or multiple *osContainerDesc*. This mapping between VNFD/AppD, Pod, *osContainerDesc*, and container allows allocating computation resources to every component of the CNF, defining CNFs that may require close placement of components and CNFs that require both close placement of some components and loosely coupled placement for other components. The CISM agent is responsible for placing replicas of a CNF or its components on different CISM nodes for the sake of availability. This mapping is inspired by the ETSI group report NFV-IFA 029 V3.3.1 [21] and ETSI VNFD [22]. In a previous work [23], we discussed the different possibilities of designing an ESST and the impact of the design on the latency and availability of the slice.

CISM level objects define the resource requirements of the pod, virtual interface definition, software configuration, etc. Domain orchestrator communicates with CISM via CISM agent to instantiate Pods. CISM communicates with PNFs via short-lived containers. The containers use the ICMP ping mechanism or Linux *NETCAT* command to check the connectivity with the PNF. They execute *lifeCycleScript* if provided in PNFD. CISM agent responds back to domain orchestrators with network information of the CNFs, PNF, and MEC application. If MEC applications request MEP services, DNS redirection, or traffic redirection as described in appD. ESSO communicates with the desired or default MEP and provides the requested services.

After the successful creation of all the network functions of a sub-slice, the slice instances are ready to serve their

customers. The termination of a slice instance follows a similar mechanism and results in terminating all the network functions, disabling services used by the MEC application, and off-boarding all the container images.

5. Comparision of LeSO and CLiSO Frameworks

The CLiSO framework is an extension of the LeSO framework described in section 2.3 and in [7]. The key differences between these two frameworks are:

1. LeSO framework is designed to orchestrate only edge or MEC sub-slices. It does not allow orchestrating CNFs that require multiple networking interfaces such as 5G UPF or PNFs.

2. The LeSO proposed ESST was not aware of slices of different domains. Whereas, CLiSO modifies the LeSOs ESST to be multi-domain slice-aware and allows the possibility to share an edge slice with multiple core or ran slices.

3. CLiSO allows sharing the sub-slices with other slices. Which was not the case with LeSO.

4. CLiSO provides the functionality to dynamically register and de-register CISM and MEPs. The registered CISM can have any number of physical or virtual interfaces. This ability was missing from LeSO.

5. The CISM plugin of CLiSO provides basic monitoring information which was not present in the LeSO CISM plugin.

6. Performance Evaluation

To evaluate the CLiSO framework, we used public and private clouds as CISM. Public cloud as Amazon Elastic Kubernetes Service (EKS), Google Kubernetes Engine (GKE), Azure Kubernetes Service (AKS), and OVH provides Kubernetes as Platform as a Service (PaaS). The Kubernetes life cycle and underlying infrastructure are managed by the cloud provider. To create a private cloud, we used Minikube[6] and Red Hat OpenShift Local[7], two command line tools to install Vanilla Kubernetes and Openshift, respectively. To evaluate our proposed framework, we performed three different experiments with different motives:

* Compatibility Testing: The motive is to evaluate the compatibility of our framework with different Kubernetes distributions.

* Configuration Testing: There are two motives for this test, First to evaluate the capability of the CLiSO framework to manage multi-domain slicing. Second, if the framework can be disintegrated and deployed on different platforms.

* Scalability Testing: The motive is to evaluate multi-tenancy and resource consumption when multiple slice creation requests arrive.

In all the experiments, we have used OpenAirInterface (OAI) [24] 5G Core and RAN Network Functions. Every experiment has a different slice configuration, different number of CNFs, or different deployment complexity to understand how many CNFs in a slice our framework can support.

6.1. Compatibility Testing

To evaluate the compatibility of the CLiSO framework with various public clouds, we managed the life cycle of a 5G Core Network Slice on different production level Kubernetes platforms. In the experiment, the core network slice went through all four phases on each public cloud. The core network slice contained AMF, SMF, NRF, UPF, AUSF, UDM, UDR, and two replicas of MYSQL. Each of the network functions and MYSQL instances were deployed as CNFs, slice contained 9 CNFs or Kubernetes Pods or 9 containers.

All the CISMs were single node clusters. Their hardware details are mentioned in Table 2. To manage the life cycle of the core network slice, we only used the CLiSO components, which are necessary, image registry, database, CSSO, and NSO. The CLiSO framework was deployed on an OVH cloud instance with 4vCPU, 15 GB RAM, and Kubernetes version 1.25.4-1.

We compared the results of the CLiSO framework with ETSI OSM. We deployed ETSI OSM in a Ubuntu 20.04 Virtual Machine with 4vCPU and 16GB RAM. We followed the instructions from the website [25]. Inside the virtual machine, the OSM deployer script deployed OSM on a *Kubeadm* based Vanilla Kubernetes cluster, a tool to create a Kubernetes cluster. We used OVH (II) and Kubeadm as CISM, their hardware details are mentioned in Table 2. OSM supports an older version of Kubernetes. In OSM terminology CISM is referred to as a dummy Virtual Infrastructure Manager (VIM). The 5G core network slice descriptor was using helm-charts [26].

Table 3 highlights the time taken (in seconds) to register different CISMs, create a core network slice and delete a core network slice using CLiSO and OSM. The values are averaged over 10 iterations. From the table we can draw some conclusions:

1. We observed that CLiSO takes less time to deploy the core network slice than OSM, which can be argued by the architectural difference between OSM and CLiSO. CLiSO's CISM agent communicates with Kubernetes using its REST APIs, whereas OSM communicates using the command line interface of Kubernetes and Helm-charts, *kubectl* and *helm*, respectively. This adds another layer of communication with the Kubernetes cluster.

2. CISM agent is compatible with different public cloud platforms and can deploy a core network slice on various public cloud platforms.

[6] https://minikube.sigs.k8s.io
[7] https://developers.redhat.com/products/openshift-local

Table 2
CISM Details for Compatibility Testing

Platform	Location	Kubernetes Version	Resources (vCPU, RAM)
Azure (AKS)	Paris	1.24.9	4, 16GB
Google (GKE)	Paris	1.24.8-gke.2000	4, 16GB
OVH(I)	Strasbourg	1.25.4-1	4, 15GB
Amazon (EKS)	Ireland	1.24	4, 7.5GB
OVH(II)	Strasbourg	1.23.14	4, 15GB
Kubeadm	Local	1.23.17	4, 16GB

Table 3
Core Network Slice Life cycle On Various Cloud Platforms via CLiSO and OSM

Orchestrator	Platform	CISM Registration(s)	Creation (s)	Deletion (s)
CLiSO	Azure	0.502	30.048	1.436
	Google	0.396	43.926	2.317
	OVH(I)	0.513	44.1	2.394
	Amazon	0.718	65.46	1.538
OSM	Kubeadm	1.846	97.68	29.97
	OVH(II)	2.930	138.98	34.78

3. CLiSO framework can be deployed partially, if only the core network slice has to be handled then only CSSO is required.

Apart from this, the CLiSO framework can deploy replicas of a CNF. For example, MYSQL in this scenario. This is done by mentioning the *replicas* in VNFD. In OSM the replicas were only possible via manipulating the helm-charts manually.

6.2. Configuration Testing

The motive of this test is to understand the capability of the CLiSO framework to manage a multi-domain slice hosted on different CISM instances. Figure 9 shows the slice and required placement of the CNFs. This configuration of 5G network slice is meant for applications requesting low latency, as the user plane is hosted at the edge. The CLiSO framework was disintegrated, image registry and its database was deployed in public cloud OVH with 2vCPU and 4GB RAM. The rest of the components, CSSO, RSSO, NSO, and their database were deployed in a local Kubernetes instance with 4vCPU and 4GB RAM. The image registry has to be located in the public cloud where all the CISM can communicate. Hence, the image registry requires a public IP-address in this scenario. Hardware information related to CISMs is below:

1. Public Cloud (OVH): 4vCPU and 16GB RAM, location Strasbourg
2. Local Openshift: 4vCPU and 16GB RAM, Openshift Local version 4.12
3. Local Minikube Kubernetes: Baremetal Kubernetes with 4CPU (No Hyper-threading) and 16GB RAM, connected with USRP B210.

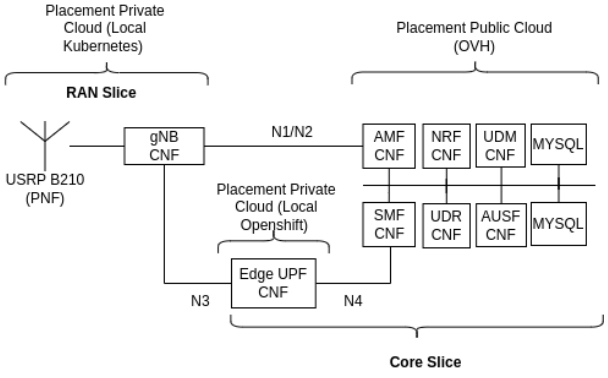

Figure 9: Ran and Core Slice Deployment

The public cloud slice had AMF, SMF, NRF, AUSF, UDM, UDR, and two replicas of MYSQL. At the edge, there was one UPF managed by Openshift CISM, and on the other edge instance, gNB CNF managed by Minikube CISM. In total, there were 10 CNFs or 10 Kubernetes Pod and USRP as PNF managed by gNB CNF. Table 4 shows the time taken (in seconds) to on-board, instantiate, terminate, and off-board the network slice. The values are averaged over 10 iterations. Though the above values are subjected to the hardware capabilities, the table concludes that the framework can manage a multi-domain slice hosted on multiple CISM instances. Like other orchestrators, OSM and ONAP connectivity between public and private clouds is a prerequisite and the CLiSO framework assumes that CISM instances can communicate with each other. ETSI OSM at the time of writing this paper did not support RedHat Openshift.

Table 4
Ran and Core Network Slice Life cycle Time (seconds)

Creation		Deletion	
43.042		3.545	
Onboard	Instantiate	Terminate	Offboard
3.259	39.783	2.431	1.114

6.3. Scalability Testing

To inspect the multi-tenancy capability of the complete CLiSO framework, we selected a slice configuration spread across all three technological domains and hosted on multiple CISM platforms. In this experiment the RAN slice used 3GPP proposed three split RAN architecture, DU, CU-CP, and CU-UP deployed on three different CISM respectively. For the experiment, we used OAI-DU in RF-simulated mode rather than connected to a physical radio unit. The control plane of the core network slice was deployed on one CISM and User Plane Function on another CISM. The MEC Application connected to UPF was a content caching server. In total 12 CNFs or 12 Kubernetes Pod were included in the end-to-end network slice.

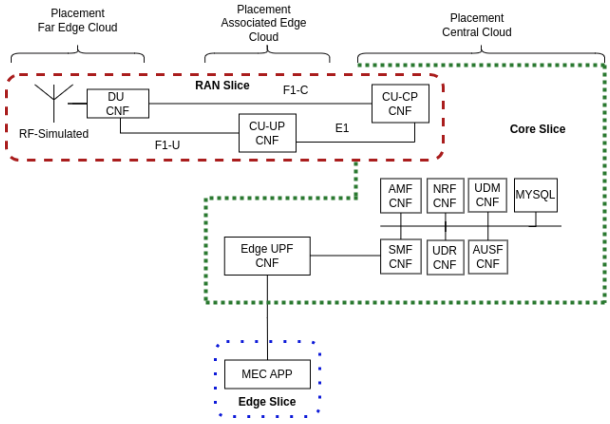

Figure 10: Slice in Three Technological Domains

We created four instances of OVH Kubernetes, two instances with 8 vCPU and 30GB RAM to imitate central and associated edge cloud, and other two instances with 4 vCPU and 16GB RAM for far edge cloud and hosting CLiSO components. For this experiment, we deployed all the components of the CLiSO framework. Due to computational resource constraints, we had to deploy all the CNFs and MEC application in sleep mode rather than functional mode. The slice network functions went through the four life cycle stages on-boarding, instantiation, termination, and off-boarding.

Figure 11 shows the slice creation time and CPU core consumption by the CLiSO framework while handling multiple slices, 2, 4 up to 14 at the same time. Due to Kubernetes default constrained of 110 Pods per host machine, we were not able to go beyond 14 slices. In OVH-managed Kubernetes this value is not configurable. Though the slice

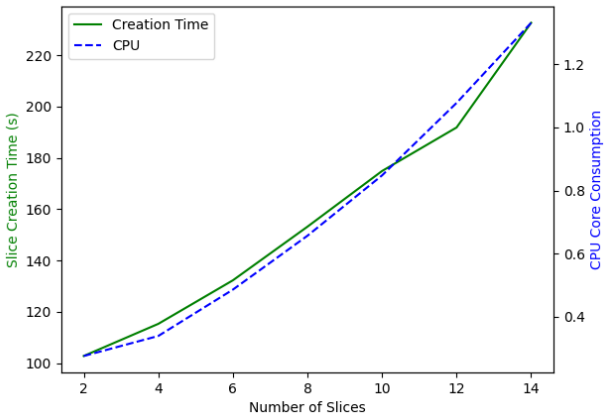

Figure 11: Slice Creation time vs CPU Consumed

Table 5
Resource Requirement OSM and ONAP

Orchestrator	vCPU	Memory (GB)
ONAP (Jakarta)	112	224
OSM-13	2	8

creation time depends on the hardware and the CNF, the graph reflects that the framework is capable of managing the life cycle of multiple slices at the same time. To create two slices the complete framework consumed 0.275 CPU and 340MB RAM. Whereas, for 14 slices the framework consumed 1.33 CPU and 365MB RAM. In addition, the framework consumes most of the resources at the time of slice creation. Once the slice is created the slice orchestrator and sub-slice orchestrators just share heartbeats among them and with CNFs. After the slice is created user can interact with the slice CNFs via CNFs management API or DSHs API/graphical user interface.

Table 5 shows the resource requirement of OSM and ONAP taken from their official website. CLiSO resource consumption is based on the scaling test we performed but we do not have similar results for other orchestrators. It should be noted CLiSO framework supports the orchestration of MEC applications and interactions with MEP. Whereas other orchestrators do not have this ability. All the components of the CLiSO framework can be deployed on any public managed Kubernetes distribution or private Kubernetes instance.

7. Future Work

We presented different capabilities and features of the CLiSO framework. In the future, we would like to add additional functionalities. Most important is integrated Zero-touch Service Management, for now, ZSM is only possible via DSH. Ideally, we would like to implement a generic

ZSM in the CLiSO framework. ONAP's Control Loop Automation Management Platform (CLAMP)[27] provides a notion of ZSM. Apart from ZSM, we would like to include energy consumption metrics for each network slice and their respective network functions. In 5G and beyond networks energy consumption metrics for different network functions have started becoming a major focus.

8. Conclusion

In this paper, we introduced a novel end-to-end orchestration framework CLiSO. The framework is lightweight when compared with other orchestrators. It takes less time to deploy slices when compared with OSM. It allows orchestrating container-based VNFs or CNFs on multiple Kubernetes distribution, Amazon Elastic Kubernetes Service, Azure Kubernetes Service, Google Kubernetes Engine, OVH, RedHat Openshift, Vanilla Kubernetes and K3S a lightweight Kubernetes distribution supporting AArch64 (ARM64) architecture. It is possible to deploy the components of the CLiSO framework on all these Kubernetes distributions. Hence, making the framework cloud-native, and easy to port to multiple cloud platforms. The framework is designed with a plugin approach and network sub-slice orchestrators abstract the type of CISM. This allows supporting any other container orchestration framework. CLiSO framework is highly customizable and can be deployed partially, as shown in the evaluation section. The deployment-centric and CISM agnostic network slice template allows describing sub-slices and their required network functions or MEC applications by abstracting infrastructure-related information. The proposed concept of Domain Slice Handlers (DSH) distributes the responsibility of managing a sub-slice among themselves and network sub-slice orchestrators. DSH allows zero-touch service management of the sub-slice network functions. DSH are network function provider-centric, allowing the providers to personalize the life cycle management of their network functions. The source code of the CLiSO framework will be released under OpenAirInterface Software Alliance (OSA).

Acknowledgement

This work is partially supported by the European Union's Horizon Program under the 6GBricks project (Grant No. 101096954), the European Commission, and the Imagine-B5G project (Grant No. 101096452).

References

[1] I. Afolabi et al., "Network Slicing and Softwarization: A Survey on Principles, Enabling Technologies, and Solutions," IEEE Communications Surveys & Tutorials, thirdquarter 2018, vol. 20, no. 3, pp. 2429-2453, doi: 10.1109/COMST.2018.2815638.

[2] A. Ksentini et al., "Toward Enforcing Network Slicing on RAN: Flexibility and Resources Abstraction," in IEEE Communications Magazine, vol. 55, no. 6, pp. 102-108, June 2017, doi: 10.1109/MCOM.2017.1601119.

[3] Karim Boutiba et al., "NRflex: Enforcing network slicing in 5G New Radio", Computer Communications, Volume 181, 2022, Pages 284-292, ISSN 0140-3664, https://doi.org/10.1016/j.comcom.2021.09.034.

[4] ETSI GS NFV 006 V4.4.1 , "Network Functions Virtualisation (NFV) Release 4, Management and Orchestration, Architectural Framework Specification", Dec 2022

[5] ETSI GS MEC 003 V3.1.1, "Multi-access Edge Computing (MEC), Framework and Reference Architecture", March 2022

[6] Girma M. Yilma et al., "Benchmarking open source NFV MANO systems: OSM and ONAP", Computer Communications, 2020, Volume 161, pp. 86-98, https://doi.org/10.1016/j.comcom.2020.07.013

[7] S. Arora et al., "Lightweight edge Slice Orchestration Framework", ICC 2022 - IEEE International Conference on Communications, 2022, pp. 865-870, doi: 10.1109/ICC45855.2022.9838854

[8] C. Benzaid et al., "AI-Driven Zero Touch Network and Service Management in 5G and Beyond: Challenges and Research Directions," in IEEE Network, March/April 2020, vol. 34, no. 2, pp. 186-194, doi: 10.1109/MNET.001.1900252

[9] GSM Association, "Generic Network Slice Template Version 7.017", June 2022.

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

[11] B. Burns et al., "Borg, Omega, and Kubernetes: Lessons learned from three container-management systems over a decade", Queue 14, 1, 2016, pp. 70–93, https://doi.org/10.1145/2898442.2898444

[12] J. Mangues-Bafalluy et al., "5G-TRANSFORMER Service Orchestrator: design, implementation, and evaluation," 2019 European Conference on Networks and Communications (EuCNC), Valencia, Spain, 2019, pp. 31-36,doi: 10.1109/EuCNC.2019.8802038

[13] X. Li et al., "5Growth: An End-to-End Service Platform for Automated Deployment and Management of Vertical Services over 5G Networks," in IEEE Communications Magazine, vol. 59, no. 3, pp. 84-90, March 2021, doi: 10.1109/MCOM.001.2000730.

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

[15] G. Papathanail et al., "MESON: Optimized Cross-Slice Communication for Edge Computing," in IEEE Communications Magazine, vol. 58, no. 10, pp. 23-28, October 2020, doi: 10.1109/MCOM.001.2000207.

[16] K. Katsalis et al., "JOX: An event-driven orchestrator for 5G network slicing," NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium, Taipei, Taiwan, 2018, pp. 1-9, doi: 10.1109/NOMS.2018.8406236.

[17] Open Baton, https://openbaton.github.io/, Accessed July 2023

[18] F. Meneses et al., "SliMANO: An Expandable Framework for the Management and Orchestration of End-to-end Network Slices," 2019 IEEE 8th International Conference on Cloud Networking (CloudNet), Coimbra, Portugal, 2019, pp. 1-6, doi: 10.1109/CloudNet47604.2019.9064072.

[19] S. Bolettieri et al., "Towards end-to-end application slicing in Multi-access Edge Computing systems: Architecture discussion and proof-of-concept", Future Generation Computer Systems, Volume 136, 2022, Pages 110 127, ISSN 0167-739X, https://doi.org/10.1016/j.future.2022.05.027.

[20] M. Mekki et al., "A Scalable Monitoring Framework for Network Slicing in 5G and Beyond Mobile Networks," IEEE Transactions on Network and Service Management, March 2022, vol. 19, no. 1, pp. 413-423, doi: 10.1109/TNSM.2021.3119433

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

[22] ETSI GS NFV-IFA 011 V4.3.1, "Network Functions Virtualisation (NFV) Release 4;Management and Orchestration; VNF Descriptor and Packaging Specification", Jul 2022

[23] S. Arora et al., "Availability and Latency Aware Deployment of Cloud Native Edge Slices," GLOBECOM 2022 - 2022 IEEE Global

Communications Conference, Rio de Janeiro, Brazil, 2022, pp. 3441-3446, doi: 10.1109/GLOBECOM48099.2022.10001341.

[24] F. Kaltenberger et al., “The OpenAirInterface 5G New Radio Implementation: Current Status and Roadmap," WSA 2019, Vienna, Austria, 2019, pp. 1-5

[25] ETSI-OSM, https://osm.etsi.org/docs/user-guide/latest/20-tutorial.html, Accessed April 2023

[26] OpenAirInterface Software Alliance, https://gitlab.eurecom.fr/oai/cn5g/oai-cn5g-fed/-/tree/v1.5.0/charts, Accessed April 2023

[27] ONAP, https://docs.onap.org/projects/onap-policy-parent/en/istanbul/clamp/clamp/clamp-architecture.html, Accessed April 2023