5G Programmable Infrastructure Converging disaggregated network and compUte REsources

D5.4 Integrated prototype (across tasks and work packages)

This project has received funding from the European Union's Framework Programme Horizon 2020 for research, technological development and demonstration

5G PPP Research and Validation of critical technologies and systems

Project Start Date: June 1st, 2017 **Duration**: 34 months

Call: H2020-ICT-2016-2 **Date of delivery**: 17th January 2020

Topic: ICT-07-2017 **Version** 1.0

Project co-funded by the European Commission
Under the H2020 programme

Dissemination Level: Public

Grant Agreement Number:	762057
Project Name:	5G Programmable Infrastructure Converging disaggregated network and compUte REsources
Project Acronym:	5G-PICTURE
Document Number:	**D5.4**
Document Title:	Support for multi-version services
Version:	1.0
Delivery Date:	30th November 2019 (**17th January 2020**)
Responsible:	**I2CAT**
Editor(s):	Daniel Camps (**I2CAT**)
Authors:	Daniel Camps (**I2CAT**), Ferran Cañellas (**I2CAT**), Ricardo Gonzalez (**I2CAT**), Azahar Machwe (**ZN**), Jorge Paracuellos (**ZN**), Sevil Dräxler (**UPB**), Hadi Razzaghi Kouchaksaraei (**UPB**), Kostas Choumas (**UTH**), Dimitris Giatsios (**UTH**), Kostas Katsalis (**HWDU**), Thierno Diallo (**UNIVBRIS-HPN**), Osama Arouk (**EUR**)
Keywords:	5G OS, Network Function Virtualization, Software-Defined Networking, Heterogeneous Infrastructure.
Status:	Final
Dissemination Level	Public
Project URL:	http://www.5g-picture-project.eu/

Revision History

Rev. N	Description	Author	Date
0.1	Table of contents and document structure	I2CAT	13.08.2019
0.2	ToC update after WP telco	I2CAT	10.09.2019
0.3	Sections 2.1, 2.3, 3.3, 3.6 (partial)	I2CAT	13.10.2019
0.4	Sections 2.2, 3.5, 2.3 (partial)	I2CAT	22.10.2018
0.5	Added sections 2.5, 3.1, 4.2	I2CAT	4.11.2019
0.6	Added section 4.3	I2CAT	08.11.2019
0.7	Updates to section 3.1 and 3.2	I2CAT	20.11.2019
0.8	Integrate updates in section 3.2, 2.2.4 and 4.2	I2CAT	3.12.2019
0.9	Integrate updates in sections 4.1 and 4.3	I2CAT	19.12.2019
1.0	Final version and submission to the EC	I2CAT, IHP	17.01.2020

Table of Contents

List of Figures

List of Tables

Executive Summary

The deliverable at hand “D5.4 Integrated prototype (across tasks and work packages)” concludes the work carried out in WP5 providing a practical implementation of the 5G Operating System (5G OS) concept, which is used to illustrate two main ideas: i) The 5G OS is capable of orchestrating end-to-end (e2e) services composing wireless access, compute and transport resources provided by distributed domains, and ii) the 5G OS can be used to orchestrate virtual network functions developed in WP4.

The main accomplishment of this work has been to demonstrate that the 5G OS can provision a complete virtual Wi-Fi and LTE network service in less than two minutes, leveraging an integrated multi-domain testbed set up by four different partners in the project. This is a very significant step forward in automating the provisioning of the complex network services involved in the operation of mobile network architectures. This effort is in addition fully aligned with the 5G-PPP overarching Key Performance Indicator (KPI) of “*reducing service provisioning time from 90 hours to 90 minutes*”.

Complementary demonstrations are also included validating how key 5G-PICTURE technologies, such as TSON and Ethernet TSN, which could not be included in the multi-domain integrated testbed due to lack of resource, can be integrated within the 5G OS.

1 Introduction

The work in WP5 throughout the 5G-PICTURE project has been focused on the concept of the 5G OS, which is seen as a lightweight operative system that binds loosely coupled domains offering compute, transport and wireless access resources in order to provision end-to-end (e2e) services in a dynamic and autonomous manner.

The 5G OS vision and architecture was first introduced in deliverable D5.1 [1], with particular aspects developed in successive deliverables. For example, deliverable D5.2 [2] introduced a hierarchical SDN control plane used to connect Virtual Network Functions (VNFs) instantiated in different NFV infrastructures through a multi-domain network. In addition, deliverable D5.3 [3] introduced and evaluated the concept of multi-version network functions, which is one of the main innovations of the 5G OS.

The main goal of the deliverable at hand is to describe a practical e2e instantiation of the 5G OS concept. For this purpose a multi-domain testbed comprising different wireless, compute and transport network domains has been set up joining the labs of four 5G-PICTURE partners through an overlay network. This multi-domain testbed is described in section 2, which also features a detailed discussion of the interfaces developed to connect each domain to core component of the 5G OS, namely the Multi Domain Orchestrator (MDO).

The multi-domain testbed is used in section 3 to benchmark the time required to provision two e2e services, namely a multi-domain virtual Wi-Fi network service and a multi-domain virtual LTE network service. Both services consist of a virtual wireless access network dynamically provisioned by the MDO in Spain and Greece respectively, and a virtual core network dynamically provisioned by the MDO in Germany. In order to connect the wireless access to the core a dynamic e2e connectivity service through a transit provider in the UK is executed. The two end-to-end services are provisioned e2e, i.e. the wireless access functions, the core network functions and the multi-domain transport network connections, in less than two minutes, which fully validates the 5G OS vision of automating e2e service provisioning across multi-domain scenarios.

Section 4 contains additional 5G OS developments that have not been integrated in the 5G OS multi-domain testbed. These developments include a management plane to provision Time Sensitive Network (TSN) connections through a RAN orchestrator featured in WP5 called JOX [4]. The second development extends the work on multi-version VNFs introduced in deliverable D5.3 [3] with a new use case. The third development provides a second instantiation of the 5G OS MDO concept in a smaller setting where two compute domains orchestrated by Open Source MANO (OSM) [5] are connected through a Time-Shared Optical Network (TSON) network domain.

Besides providing a practical implementation of the 5G OS concept, a secondary objective of this deliverable is to demonstrate how the 5G OS is indeed capable of orchestrating network functions developed in WP4, hence validating the overall vision of 5G-PICTURE regarding the relation between the WP3 programmable platforms, the WP4 physical and virtual functions and the WP5 5G OS. The best example in this regard is the multi-domain testbed itself, where the hierarchical control plane to provision connectivity services across the various domains has been developed and evaluated in deliverable D4.3 [6]. In this deliverable the compute and wireless access domains supporting the virtual Wi-Fi and LTE services have been connected to said testbed. This collaboration between WP4 and WP5 has been key in order to accomplish the substantial integration efforts required to build the aforementioned multi-domain testbed. Other examples of WP4 functions orchestrated by the 5G OS are included in this deliverable. First, in the multi-domain integrated testbed a joint access and backhaul Wi-Fi function developed in WP4 is orchestrated by the 5G OS in the virtual Wi-Fi service. Another example are the network functions based on OpenAirInterface (OAI) [7] used in the virtual LTE service. A third example is the integration between JOX and the TSN management system, which is a management plane supporting the TSN data-plane work featured in the RAN integrated prototype described in deliverable D4.3 [6]. Finally, the integration of the MDO with the TSON control plane functions developed in WP4 constitute another example of WP4 functions managed by the 5G OS.

2 Integrated demonstrator of the 5G OS for end-to-end service provisioning

2.1 Overview and goals of the integrated demonstrator

The goal of the 5G OS integrated demonstrator is to validate how the 5G OS can be used to orchestrate the automated provisioning of end-to-end (e2e) network services over a distributed and multi-domain compute and network infrastructure.

Following the design of the 5G OS architecture, the key component enabling the orchestration of e2e services is the Multi-Domain Orchestrator (MDO) that interacts with all the network and compute domains involved in the service creation.

To illustrate the goals of the 5G OS integrated demonstrator Figure 2-1 depicts a multi-domain topology that has been jointly set-up by several project partners to support the instantiation of e2e services orchestrated through the MDO. In particular this demonstrator includes:

- The I2CAT domain: Featuring wireless nodes developed in WP4, which support the instantiation of virtual WiFi access and backhaul connectivity services through a custom management plane developed in WP5.
- The UTH domain: Providing compute and network nodes and featuring an orchestrator based on Open Source MANO (OSM), integrated with the MDO in WP5.
- The UPB domain: Providing compute and network capabilities, while supporting both container-based VM-based virtualization and featuring a custom service orchestrator known as Pishahang, extended and integrated with the MDO in WP5.
- The ZN domain: Offering on-demand transport network slices through its Dynamic Slicing Engine (DSE) technology extended and integrated with the MDO in WP5.

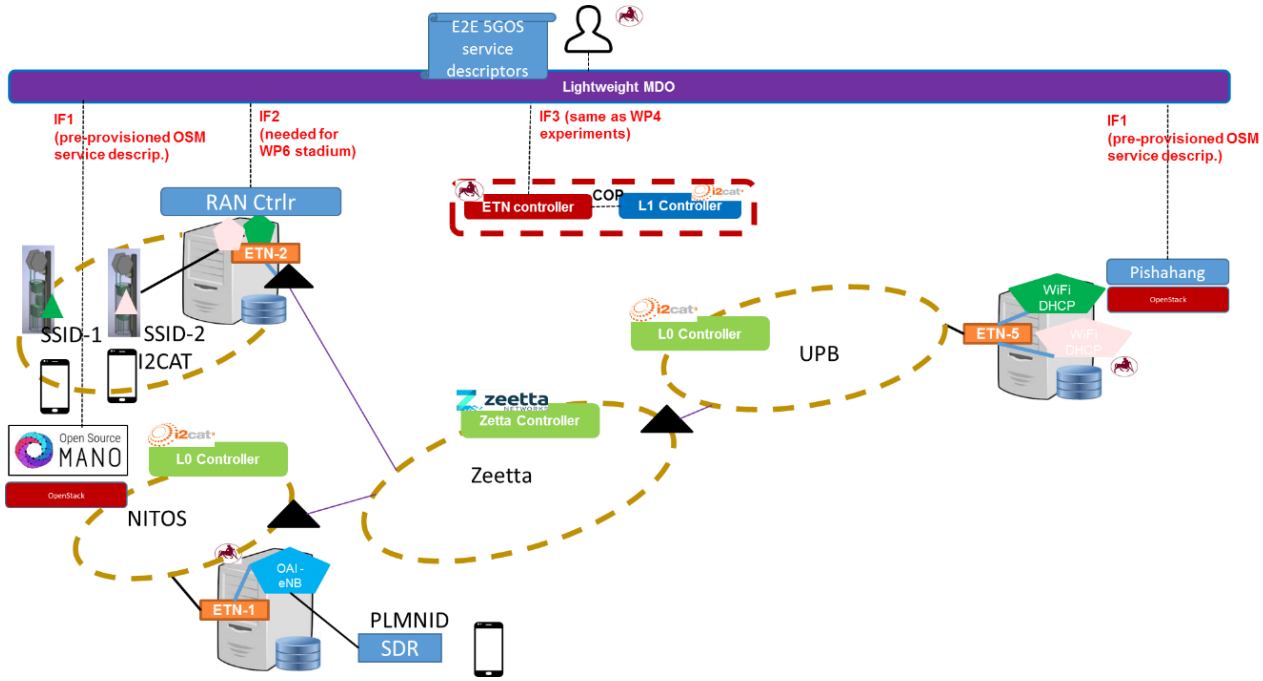

Figure 2-1: 5G OS integrated demonstrator.

The MDO orchestrates all the aforementioned domains through the integration of custom interfaces exposed by each domain, namely:

- The **IF2** interface is used to orchestrate the provisioning of virtual Wi-Fi services within the **I2CAT** domain.
- The **IF1** interface is used to deploy Virtual Network Functions (VNFs) through the OSM in the **UTH** domain.
- The **IF1'** interface is used to deploy container or VM-based VNFs through the Pishahang orchestrator in the **UPB** domain.
- The **IF3** interface is used to provision e2e connectivity across domains using the hierarchical SDN control plane reported in deliverable D5.2 [2], and evaluate using a similar multi-domain testbed in deliverable D4.3 [6].

2.2 Relation between the proposed demonstrator and the 5G OS architecture

This section relates the work in this Task 5.4 with the 5G OS architecture defined in deliverable D5.1 [1]. 5G OS consists of three main concepts:

1. Presence of Administrative Domains that own and provide resources such as connectivity, compute and network devices.
2. Multi-domain orchestrator (MDO) that can provision e2e services using resources provided by different Administrative Domains.
3. Interfaces between Orchestrators owned by different Administrative Domains (i.e. a Domain Orchestrator) and the MDO.

Figure 2-2 shows how these concepts can provide different types of e2e slices that map to one or more services that have clear business value. In the figure there are three Administrative Domains: Domain 1, 2 and 3. Each has a single Domain Orchestrator and Domain Controller which provides some kind of resource interface to the Multi-domain Orchestrator. The key responsibility of a Domain Orchestrator is to provide an accurate view of available resources to the MDO and on request reserve/configure these for use by a service. The MDO is responsible for taking the provided resources and integrating them within a network slice to provide isolation, abstraction and other benefits. It should also be able to maintain the slice in case one or more resources it is using are no longer available.

Three types of slices (or services) are shown in Figure 2-2 which has three tenants being supported. Tenant 1 has requested a slice with full control interfaces that allow deployment of a Tenant Orchestrator and Control (e.g. a MVNO use-case). Tenant 2 has requested a slice without any control interfaces. In this case all functions are externally managed or are unmanaged. Tenant 3 has requested a connectivity slice; this is the simplest possible slice.

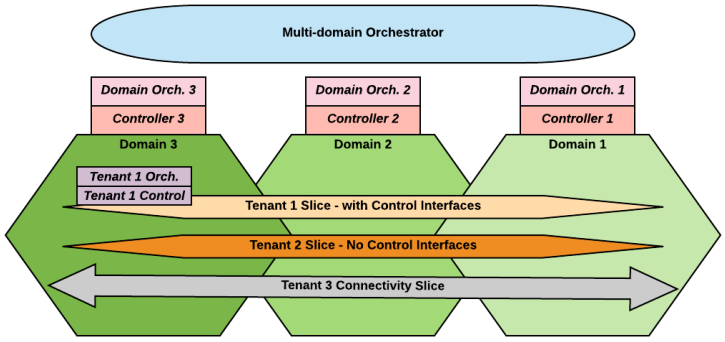

Figure 2-2: 5G OS Architecture used to provide multi-tenant slices across different administrative domains.

These domains are also recursive. In the sense that a Domain may take one or more resources from different providers repeating the same architecture at a smaller scale.

2.3 5G OS multi-domain integrated testbed

In this section we describe the detailed design of the integrated multi-domain testbed, starting with an overview of the complete testbed in Figure 2-3, which highlights the **UTH**, **I2CAT**, **UPB** and **ZN** domains.

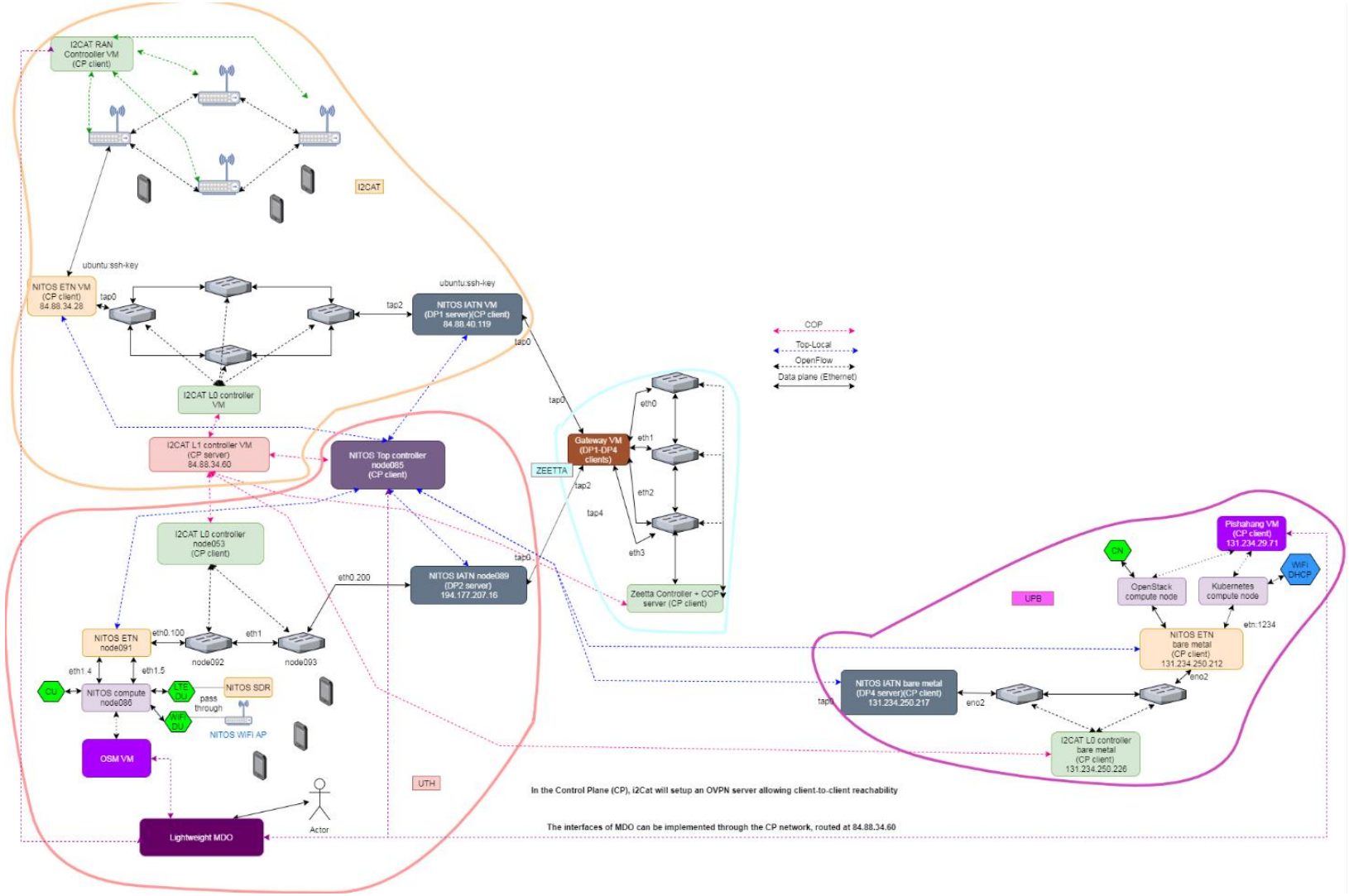

Figure 2-3. Overall integrated 5GOS multi-domain testbed

The different elements of the 5G OS, namely the MDO and the various controllers involved are depicted in Figure 2-3 as colored boxes and have been deployed inside virtual machines deployed in each domain. All the control functions communicate using a dedicated virtual private network deployed using OpenVPN [8]. In the following sub-sections we describe in detail each of the considered domains.

2.3.1 I2CAT domain

Figure 2-4 depicts the detailed topology of the I2CAT domain, where we can distinguish two separate segments:

- A radio access segment at the top of the figure composed of wireless nodes that simulate small cell devices with joint access and backhaul capabilities. A photo of the physical nodes in the I2CAT lab is included in the upper-right corner of Figure 2-4. The management plane to control these devices is deployed inside a VM labelled as I2CAT RAN Controller (shown in green), which exposes interface IF2 towards the MDO. End user devices can connect to the virtual radio access points deployed through the IF2 interface. The elements of this domain are highlighted in red in Figure 2-4.
- A wireless transport segment, located at the bottom of the image, which is composed of 4 additional nodes controlled by the I2CAT L0 controller (shown in green). The elements of this domain are highlighted in blue in Figure 2-4. The interested reader is referred to deliverable D4.3 [6] for a detailed description of the transport segment and the several components involved, e.g. ETN/IATN functions and the L1 and L0 controllers.

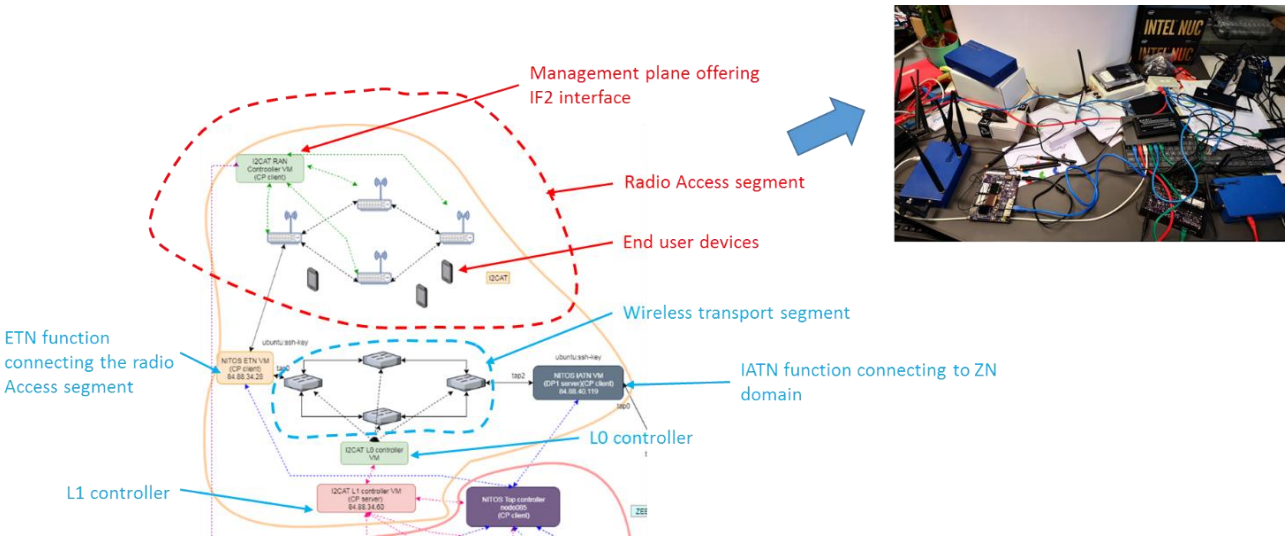

Figure 2-4: I2CAT domain.

The I2CAT domain will be used to demonstrate how a virtual e2e Wi-Fi service can be setup between the I2CAT and UPB domains allowing end-users (mobile devices) in the I2CAT domain to access service functions (VNFs) in the UPB domain over the multi-domain testbed.

2.3.2 UTH domain

Figure 2-5 depicts the detailed topology of the UTH domain. The UTH domain consists of 7 physical nodes, called Icarus, offered by the NITOS testbed of UTH. In addition, one server running OSM/OpenStack in NITOS is exploited for the purposes of the proposed demonstrator. The 7 nodes are the compute node hosting the VMs, one ETN, one IATN, two TNs, the L0 Controller of UTH domain and the Top Controller of all domains. The OSM server and the Icarus nodes are interconnected through an OpenFlow and a L2 switch, for the data and control planes respectively, while the OpenFlow switch is appropriately configured to support the data plane depicted in Figure 2-5.

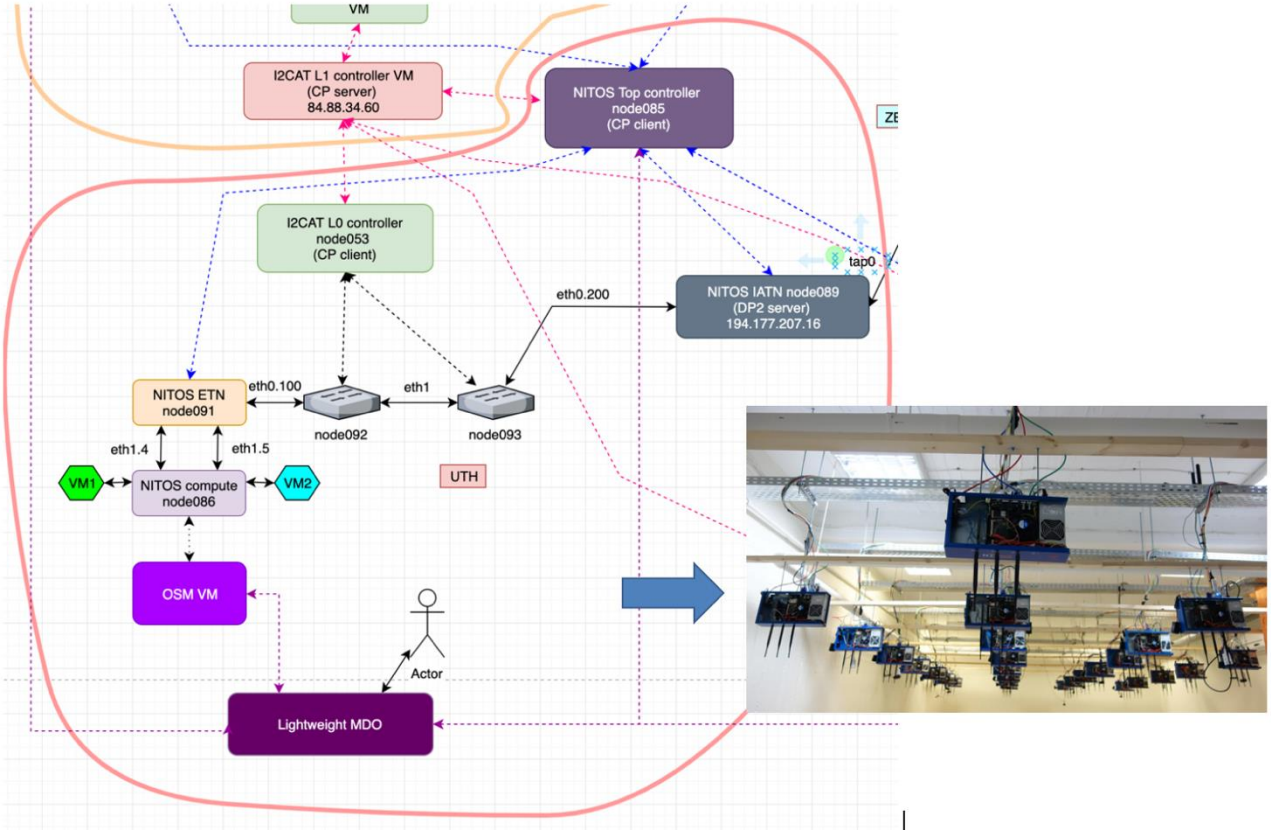

Figure 2-5: UTH domain.

2.3.3 UPB's domain

Figure 2-6 shows the detail topology of UPB domain which consists of eight nodes. In this domain, the Pishahang orchestrator runs on a VMware virtual machine, Kubernetes is hosted by a Dell workstation, also OpenStack runs on a Dell workstation, the two Open vSwitch (OVS) switches are hosted by two x86 PCs, and I2CAT's L0 controller and UTH's IATN and ETN also run on x86 PCs.

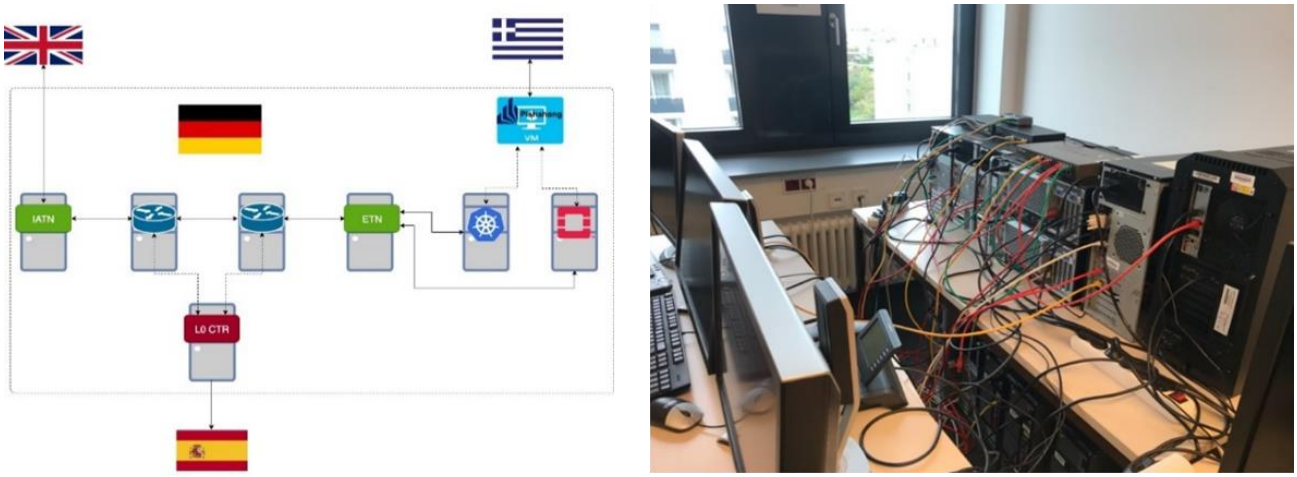

Figure 2-6: UPB domain logical topology (left) and real setup (right).

2.3.4 ZN domain

The Zeetta Networks (ZN) domain models the behaviour of a transit provider that supports multiple tenants. It represents a separate administrative domain and therefore has its own Domain Orchestrator and Controller

(as described in the 5G OS). It provides connectivity on request to interconnect the different tenants' sites together. The tenants in this case are three research partners:

- University of Paderborn (**UPB**) based in Paderborn, Germany.
- **I2CAT** based in Barcelona, Spain.
- University of Thessaly (**UTH**) based in Volos, Greece.

There are therefore three main types of requests possible: "UPB – i2CAT", "I2CAT – UTH", and "UTH – UPB", each with one of two possible class of service qualifiers: Gold (bandwidth: 100Mbps) and Silver (bandwidth: 10 Mbps). However, in our multi-domain testbed this class of service is respected only within the ZN domain. The main idea behind these is to show different types of slices can be provisioned depending on incoming requests. These requests are mapped to slices within the ZN domain and therefore correspond to Tenants requesting different slices. In Figure 2-2 these are the Tenant 2 slice with unmanaged functions (in our case Class of Service) and Tenant 3 slice with connectivity. Tenant 1 slice is also presented, where the functions are linked together using the connectivity slice provided by ZN domain. Figure 2-7 shows how the Zeetta Domain within the 5G OS architecture.

Testbed details

The testbed for Zeetta's domain consists of three EdgeCore 4610-54p switches and two servers (see Figure 2-8). The solid blue lines represent the data plane (numbers on the link indicating interface identifiers on the switch) where each link is 1Gbps. The dashed red lines represent the control plane.

The VPN server runs the OpenVPN clients to the three tenant sites (in Germany, Spain and Greece respectively) for the tests. It consists of three bridges connecting the OpenVPN interfaces with different physical interfaces on the server (if2, if3 and if4). Each of these then connect to a different switch. For ease of debugging, interface 2 on each switch is the interface pointing to the connected external site.

The Orchestration and Management server runs the orchestration (Dynamic Slicing Engine - DSE) and control software (NetOS). This server also hosts the ZN Control Orchestration Protocol (COP) Adapter (developed as part of 5G-PICTURE) designed to allow tenants to use their infrastructure to request services from the transit provider and to integrate ZN's DSE with the MDO. This has several benefits including:

- Low integration costs for tenants.
- Allowing resources borrowed from the transit provider to be included in tenant networks.

The control/management OpenVPN link to the L1 Controller being hosted by i2CAT is anchored to this server.

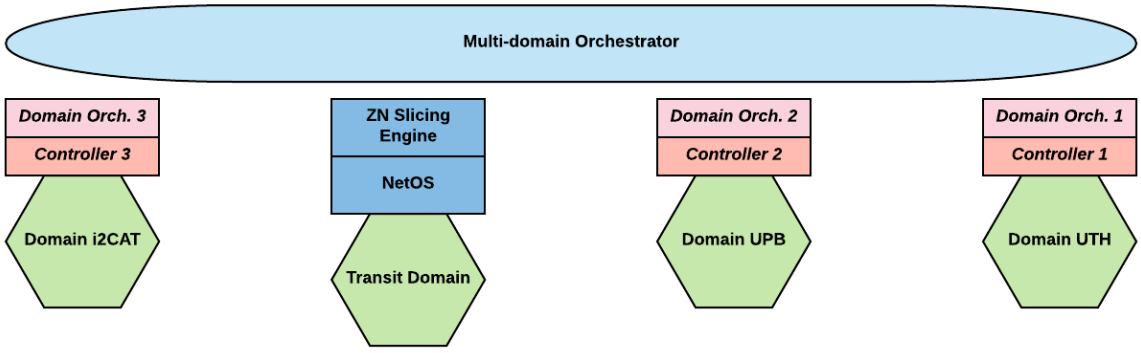

Figure 2-7: Zeetta Domain within the 5G OS architecture.

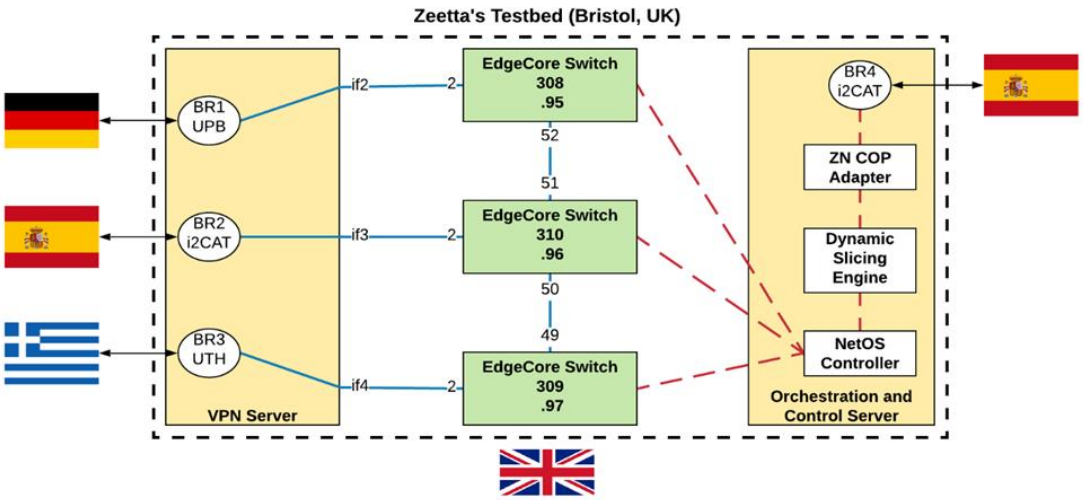

Figure 2-8: Zeetta Networks' domain in detail.

Slice Requests

The abstraction of a 'slice' may not be used universally even though it is a simple way of modelling services and resources. Tenants may choose to model services and resources they use using different frameworks (e.g. TMForum Information Model [9]).

In the case of the demonstrations for Task 5.4, the tenants use the concept of a hierarchy of controllers that provide topologies to higher level controllers and accept connectivity across their topology using the Controller Orchestration Protocol. To allow for maximum compatibility with minimum effort, the simplicity of the slice model was used and COP requirements were mapped on using a COP Adapter created by ZN.

As seen in Figure 2-2, the Tenants (**i2CAT**, **UPB** and **UTH**) have no idea that they are being provided connectivity via slices. They only see the resource and service artefacts that they care about: topology and connectivity. To support multiple tenants, it is important to not have high cost of integration. The slicing model provides this flexibility where if a tenant is a direct customer (see Figure 2-9) then they can continue using the slice model and simply request for slices. This has the advantage of allowing them to integrate directly with their slices as both models are the same. Finally, for any other tenants who choose to hide their models, for any reason, they can build their own adapters to map to the simple and adaptable slice model.

This flexibility also allows the Transit provider to provide services using any possible mix of resources without impacting the interface with the Tenant. This is an important feature to have considering the mix of resources used to provide a basic yet fundamental service like connectivity can be quite diverse.

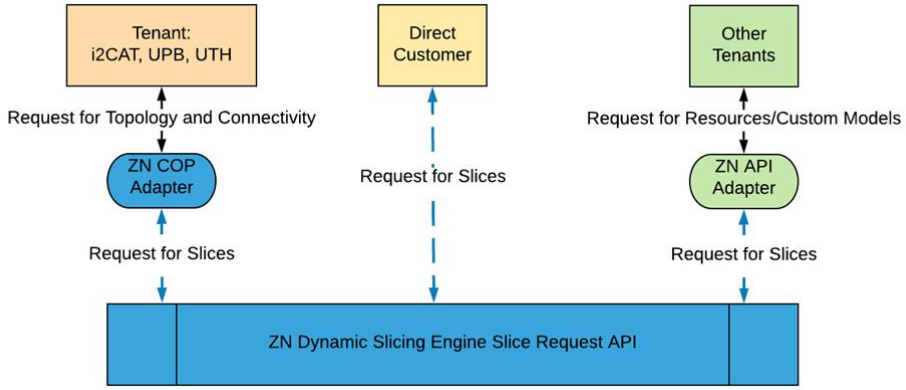

Figure 2-9: Slicing request flow, APIs and multi-tenancy support.

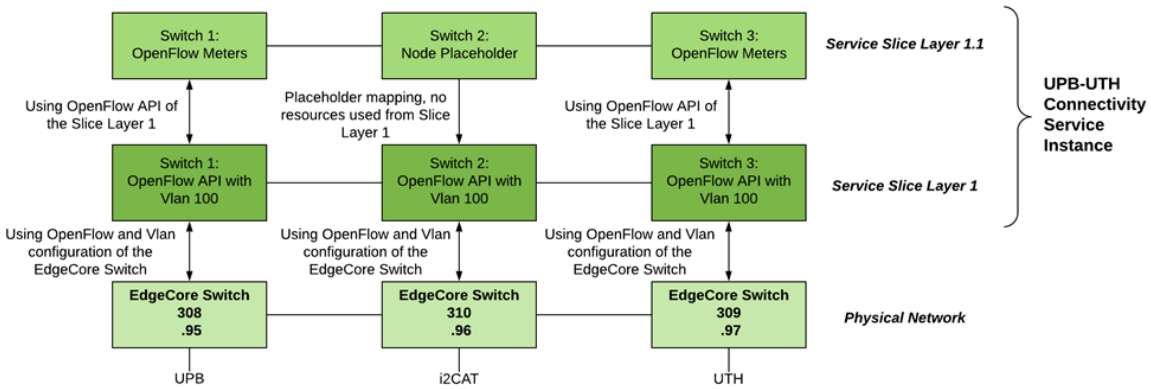

Figure 2-10: Connectivity service implementation mapped to slice layers.

The flow of requests for the demonstration described in this document is:

1. Tenants request topology that describes how the three sites are connected through the **ZN** domain.
2. **ZN** Adapter can provide any kind of abstract topology it chooses (thereby hiding the actual topology) as long as the edge interfaces are correct and the abstract topology can be mapped to the real one within the **ZN** domain (the tenant needs to know nothing about the real topology).
3. Once the topology is registered with the tenants, they can make requests for connectivity with a suitable class of service modifier.
4. For every connectivity request, **ZN** COP Adapter, internally maps it to two slices layered one on top of the other, this is shown in Figure 2-10 with an example:
    a. Lower slice defining the connectivity via Layer 2 (VLANs) and provides OpenFlow APIs.
    b. Upper slice defining the unmanaged class of service functions (meters and queues) using the OpenFlow APIs provided by the lower layer.

Testbed Results

The interaction between L1 controller, COP Adapter (L0) and Slicing Engine starts with the Registration process for the ZN domain that exposes a COP interface:

1) L1 Controller requests for the domain topology from all L0 controllers to register the different domains.
2) COP Adapter responds with a topology of the ZN domain (see Figure 2-11 left and middle).
3) Registration process is completed.

Once registration is successful, services can be created across the domains where connectivity within a domain is delegated to the corresponding L0 domain controller. In this case all COP requests to the ZN COP Adapter are translated to internal Slice Creation Request calls:

1) L1 Controller requests ZN COP Adapter to create a path between two IATNs at the edge of the ZN domain with a specific class of service.
2) ZN COP Adapter translates the request into a Slice Creation Request call and forwards it to the ZN Dynamic Slicing Engine.
3) The ZN Dynamic Slicing Engine creates required connectivity and meters and returns path details to the L1 Controller (see Figure 2-11 - right).

Figure 2-11: Response Snippet for COP 'get-topology' call showing an edge (left). Response Snippet for COP 'get-topology' call showing a node (middle). Response Snippet for COP 'set-path' call showing a slice 'path' created by the ZN COP Adapter.

The Slice generated in response to a L1 path creation call is shown in Figure 2-12 and Figure 2-13.

Figure 2-12: Generated slice definition snippet showing slice meta-data, a link and a node.

Figure 2-13: Generated slice definition snippet, showing details of a node and its tps.

As a part of the Slice hierarchy (see Figure 2-10) created for connectivity we install a set of OpenFlow flows on the nodes to provide connectivity (via VLANs) and metering (via OpenFlow meters). An example of the installed flows can be seen in Figure 2-14. Similarly, an example of the installed meters, used in the flows, can be seen in Figure 2-15.

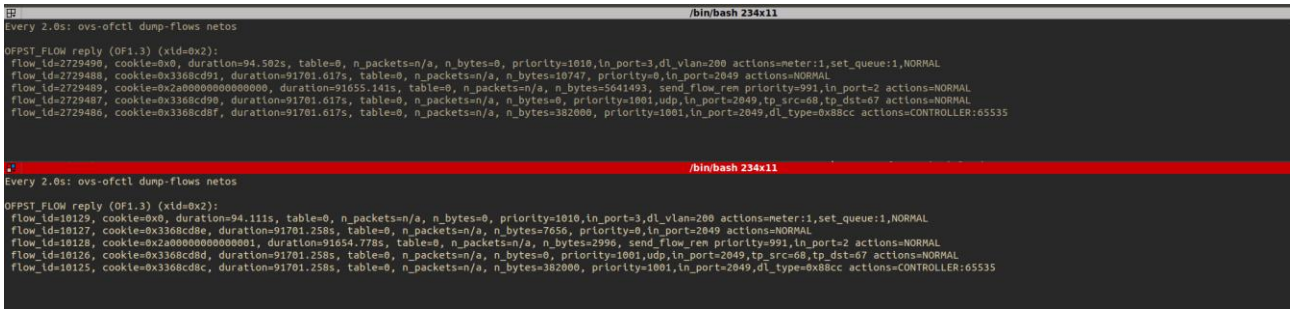

Figure 2-14: Installed flows as part of slice creation process, note VLAN match and meters.

Figure 2-15: Meter definition created for the slice, used to create different class of services, in above for Gold Class (100 Mbps).

Once the slices have been setup in the ZN domain and other domains have been setup successfully, end to end tests (between sites) can be carried out to measure KPIs such as service creation times, bandwidth and jitter.

2.4 Design of hierarchical control plane

5G OS uses a hierarchical multi-domain SDN control-plane to be able to autonomously orchestrate connectivity services connecting VNFs instantiated on distributed compute domains. The 5G OS hierarchical control plane is based on the control plane originally proposed in the 5G-XHaul project [10], which was further enhanced in deliverable D5.2 [2], and has been extensively evaluated using the heterogeneous transport technologies considered here in deliverable D4.3 [6]. Even though this hierarchical control plane has been reported in previous deliverables we briefly summarize it here for the sake of completeness.

The 5G-PICTURE architecture is composed of three main functions in the dataplane, namely the Transport Nodes (TNs), depicted as circles in Figure 2-16; the Edge Transport Nodes (ETNs), depicted as squares in Figure 2-16; and the Inter-Area Transport Nodes (IATNs), depicted as triangles in Figure 2-16. In a nutshell TNs are simple forwarding nodes that only know how to forward packets across connections defined in their domain, ETNs bind host (virtual) network functions to the appropriate per-domain connections in order to provide end-to-end connectivity, and IATNs stitch connections between domains.

The control plane of the 5G-PICTURE transport network is composed of a hierarchy of logical controllers, as illustrated in Figure 2-16. The Top controller is responsible for orchestrating the required connectivity across different areas or domains. The Level-0 controller is responsible for the provisioning and maintenance of transport tunnels in the TNs to connect the ETNs and IATNs of a given area. A set of Level-0 controllers are logically organized under a Level-1 controller. The latter is technology-agnostic, and is in charge of maintaining connectivity between the corresponding areas. Finally, ETNs and IATNs, which lie at the edges of transport areas, are directly controlled by Local Agents which are the glue between their datapaths and the Top controller with which they interact.

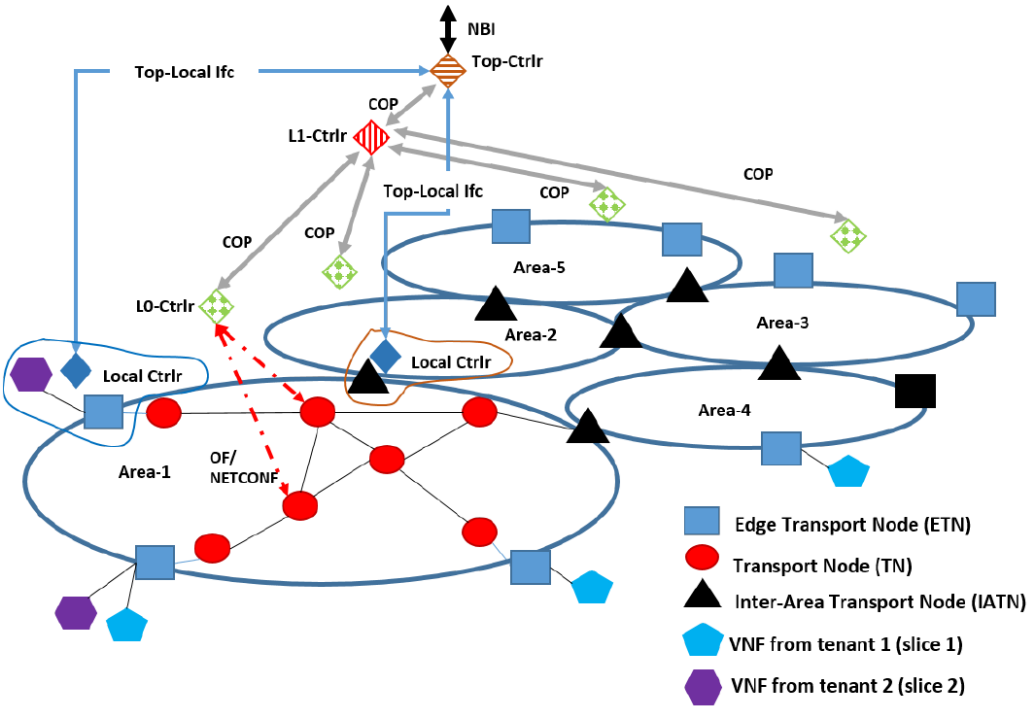

Figure 2-16: Proposed hierarchical Control Plane.

A detailed description of the signaling flows between the L0, L1 and Top controllers is included in deliverable D5.2 [2].

2.5 Design of the Multi Domain Orchestrator (MDO)

For the purpose of validating the 5G OS architecture a lightweight MDO is developed as a tailor-made software producing a sequence of HTTP requests to the REST APIs offered by the various domains in our multi-domain testbed, namely IF1, IF1', IF2 and IF3. As we will explain later in detail, the IF1 and IF1' interfaces belong to the OSM and Pishahang orchestrators respectively, and they are called first when new VNFs have to be deployed in the respective domains. The other example of end-points are the devices connected to the RAN, which have to be declared to the I2CAT RAN Controller through the IF2 interface. Finally, the MDO uses IF3 to communicate with the Top Controller, described in the previous section, and build the tunnels between the ETNs, where the end-points (either VNFs or wireless devices) are attached.

We describe now in detail the different interfaces involved in our proposed 5G OS prototype.

2.5.1 Design of interface IF1

In this section we describe the design of the IF1 interface, that allows the MDO to provision Network Services (NSs), including various VNFs. The interface is identical to the OSM Northbound interface (REST API), since we exploit OSM for the NS deployment. OSM supports with multiple VIMs, operated either by OpenStack, OpenVIM, etc. Our developed proxies in Task 5.2 enable OSM to spread the VNFs of a NS over multiple Points of Presence (PoPs) managed by different VIMs. A demonstration of this enhanced functionality of OSM has been done in NetSoft 2019 [11]

2.5.2 Design of interface IF1'

Pishahang facilitates the communication between different MANO frameworks using an adaptor component, by wrapping the REST APIs of MANOs in Python code. For this, UPB has developed the Python MANO Wrappers (PMW) as a uniform and standards-oriented Python wrapper library for various implementations of NFV MANO REST APIs. PMW follows the conventions from the ETSI GS NFV-SOL 005 (SOL005) RESTful

protocols specification [12]. This makes it easy to follow and the developers can use similar processes when communicating with a variety of MANO implementations. PMW is easy to install and use and is well documented. Code usage examples are available along with the detailed documentation[1]. Using PMW, features like scaling hierarchies of MANO frameworks, on-boarding of NSD and VNFD, instantiation and termination of network services can be performed easily in a MANO-independent way.

Our standards-based approach is based on ETSI SOL005, which contains a blueprint for all of the methods used in standard solutions. These methods are divided into different sections:

- auth: Authorization API.
- nsd: NSD Management API.
- nsfm: NS Fault Management API.
- nslcm: Lifecycle Management API.
- nspm: NS Performance Management API.
- vnfpkgm: VNF Package Management API.

Figure 2-17 shows the high-level architecture of PMW. We have implemented support for common orchestration solutions like Open Source MANO (OSM), SONATA, and Pishahang based on the common interface provided by PMW. PMW can be installed using the following command:

```
pip install python-mano-wrappers
```

A simple script to get started with PMW is shown in Figure 2-18. Here, the wrappers are imported, and a client object is created according to the used MANO framework (e.g., OSM or SONATA). Such a client object can be used to make REST calls relevant to the MANO framework. An example usage to retrieve all the network service descriptors of OSM is show in Figure 2-19. In this figure, the OSMClient module is used to first fetch an auth token. Then, using the auth token, the required information, in the case the NSDs can be fetched.

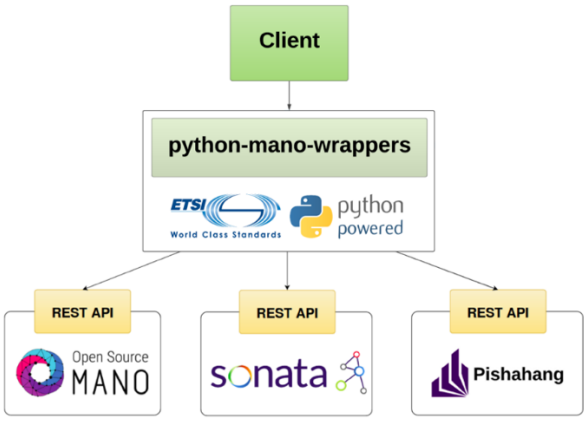

Figure 2-17: PMW high-level architecture:

[1] https://github.com/CN-UPB/python-mano-wrappers

```python
import wrappers

username = "admin"
password = "admin"
mano = "osm"
# mano = "sonata"
host = "osmmanodemo.com"

if mano == "osm":
  _client = wrappers.OSMClient.Auth(host)
elif mano == "sonata":
  _client = wrappers.SONATAClient.Auth(host)

response = _client.auth(
              username=username, password=password)

print(response)
```

Figure 2-18: Simple wrapper code to fetch a token.

```python
import wrappers

osm_nsd = wrappers.OSMClient.Nsd(HOST_URL)
osm_auth = wrappers.OSMClient.Auth(HOST_URL)

_token = json.loads(osm_auth.auth(
                    username=USERNAME,
                    password=PASSWORD))

_token = json.loads(_token["data"])

response = json.loads(osm_nsd.get_ns_descriptors(
                      token=_token["id"]))
response = json.loads(response["data"])
```

Figure 2-19: Wrapper code to fetch all NSDs in OSM.

2.5.3 Design of interface IF2

In this section we describe the design of the IF2 interface that allows the MDO to provision virtual Wi-Fi services over the joint access-backhaul technology component developed by I2CAT in WP4, and reported in D4.2 [13]. This interface is based on the NETCONF API developed in WP3 and reported in D3.2 [14]. In addition, a preliminary benchmark of this interface was reported in D6.2 [15], as it was used to support a demonstrator prepared for EuCNC 2019. In this section we formally report the design of all the 5G OS components required to manage the lifecycle of virtual Wi-Fi access and backhaul services.

As reported in D4.2 [13], the basic idea of this technology component is to control an infrastructure composed of distributed Small Cells, where only a subset of these Small Cells have a wired connection to the network and the rest are connected using wireless backhaul capabilities. Said infrastructure is considered to be a neutral in the sense that multiple Mobile Network Operators (MNOs) can instantiate a virtual connectivity services to connect their customers over the same infrastructure. Figure 2-20 depicts an example of the joint access-backhaul Small Cell infrastructure.

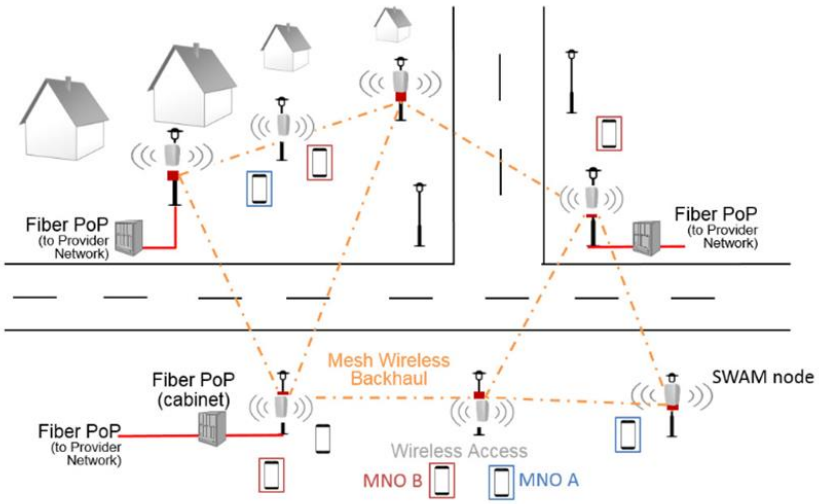

Figure 2-20: Joint access-backhaul infrastructure.

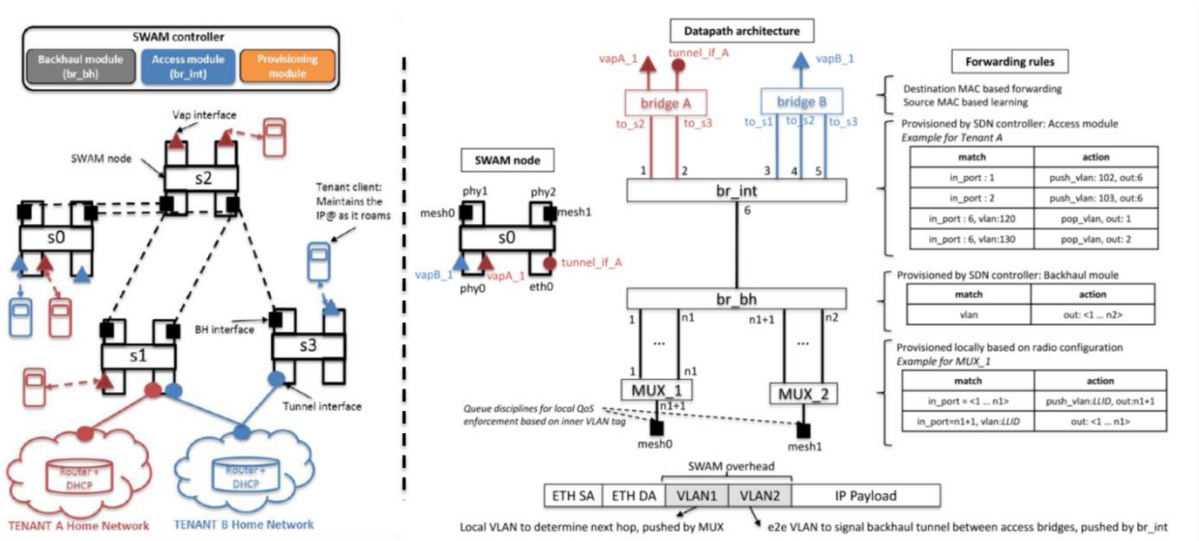

Figure 2-21: SWAM services and data-plane architecture.

Over this infrastructure we define a connectivity service as a set of virtual Wi-Fi access points (vAPs) instantiated over a subset of the physical Small Cells, which broadcast the signal (SSID) of a given MNO. We refer to this service as a SWAM service, as introduce in deliverable D4.2 [13]. The customers of the MNO can connect to the vAPs of said SWAM service, while the infrastructure forwards their traffic along the wireless backhaul until a point where a wired network connection is available, from where a tunnel is deployed that carries the customer traffic until the MNO's home network. This concept of a SWAM service is illustrated in Figure 2-21.

In deliverable D4.2 [13] we introduced the data and control plane required to provide joint access backhaul SWAM services over our Small Cell infrastructure. The data plane is composed of a set of interconnected software bridges instantiated on each node (I2CAT box) depicted on the right hand side of Figure 2-21. The control plane is composed of an SDN control that programs the forwarding data base of said software bridges. The interested reader is referred to D4.2 [13] for a detailed description of this architecture.

As part of the 5GPICTURE 5G OS we have developed the management plane able to manage the lifecycle of SWAM services, i.e. configure the infrastructure, provision and configure services, and delete them when they

are not needed anymore. This management and control is composed of a set of distributed software components that are depicted in Figure 2-22, which hereafter is referred to as the SWAM management plane. In Figure 2-22 we can see how an I2CAT joint access and backhaul box is controlled using three types of south-bound protocols, namely:

- NETCONF is used to configure physical interfaces (phy0 – phy3 in Figure 2-22) and to manage the lifecycle of the virtual APs instantiated over the physical interfaces. In order to do so an open source Netconf server (netopeer 2.0 [12]) has been extended to expose the capabilities of the I2CAT box through a YANG model reported in deliverable D3.2 [10], and to interact with the Linux wireless management tools and with hostpad [13]. We also developed a custom Netconf client that exposes these capabilities to the core components of the management plane.
- OVSDB [18] is used to provision software bridges inside the I2CAT boxes. These are the software bridges required to instantiate the SWAM data-path depicted in Figure 2-21. As described in D4.2 [13], for each new SWAM service a new access bridge is provisioned and connected to the integration bridge. We use the OVSDB services available in OpenDayLight Fluorine, but build a think OVSDB client on top of it to simplify the operations of the management plane.
- OpenFlow 1.3 is used to configure forwarding rules inside the integration bridge of the SWAM data-path, and to program the forwarding paths in the wireless backhaul. We implement a COP server module inside OpenDayLight Fluorine to expose a simple interface to program paths in the wireless backhaul.

Finally, consuming all the REST APIs exposed by the previous modules we have the component of the SWAM management plane that we name RACOON: "*RAN Controller based on OpenFlow, OvsDB and Netconf*". RACOON maintains two main internal abstractions, namely:

- The "*Chunk abstraction*": Where a chunk is a set of physical resources where a tenant is allowed to deploy a service. Physical resources can be physical interfaces in an I2CAT box, or links in the wireless backhaul.
- The "*Service abstraction*": Where a service is a set of virtual APs radiating the same BSSID, which are instantiated over a subset of the physical interfaces included in that service chunk. Automatic wireless backhaul paths connecting the service vAPs to a wired node are also automatically deployed.

The aforementioned Chunk and Service abstractions offered by the SWAM management plane to the rest of components of the 5G OS, in particular to the MDO. These abstractions are better understood through an example, as the one depicted in Figure 2-23.

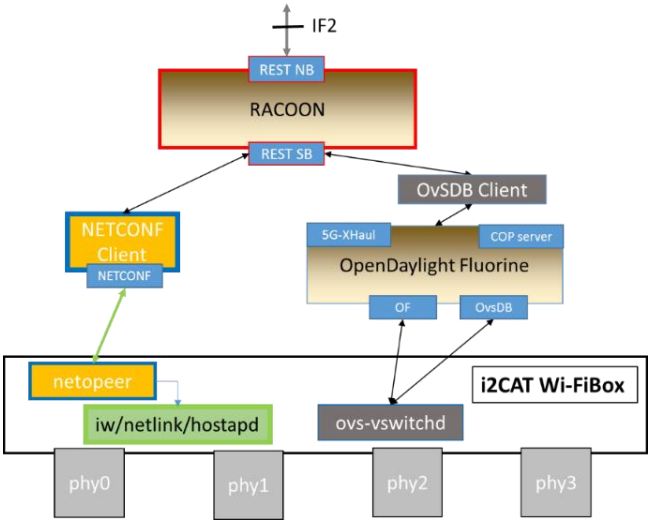

Figure 2-22: 5G OS components to manage virtual Wi-Fi services.

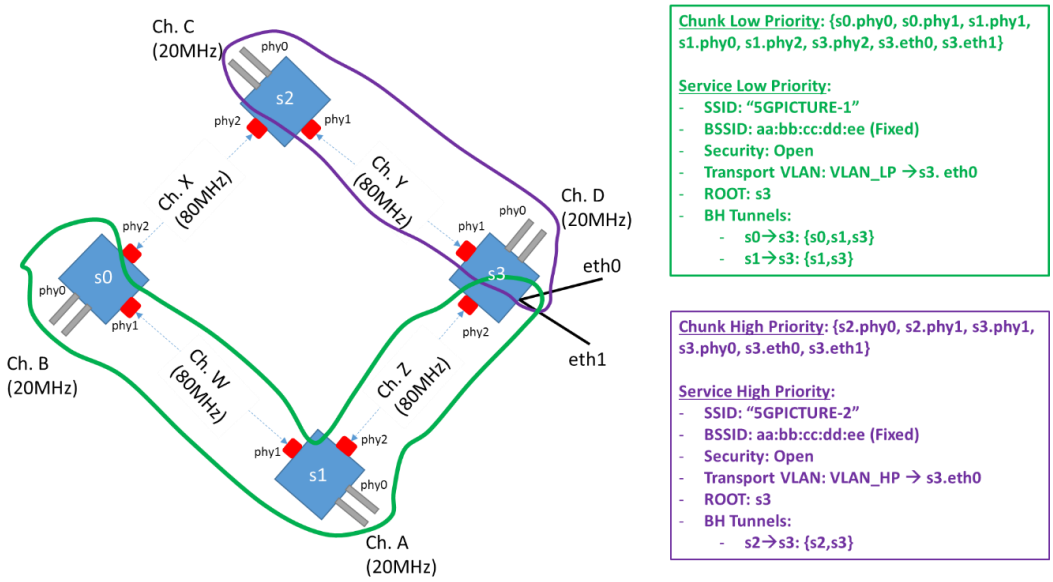

Figure 2-23: Chunk and Service abstractions offered by IF2.

We can see in Figure 2-23 an example infrastructure composed of four I2CAT box nodes (*s0*, *s1*, *s2*, *s3*), where only one of them (*s3*) is connected to the wired network. All I2CAT box nodes in the example have three Wi-Fi physical interfaces, two of which are setup as backhaul links (depicted as red squares), and one is used for access (depicted with two dipole antennas). In this infrastructure two separate chunks are setup. The high priority chunk, depicted in purple, uses as resources the access interfaces in nodes *s2* and *s3*, and the backhaul link "*s2 – s3*". The low priority chunk, depicted in green, is composed of the physical access interfaces in nodes *s0* and *s1*, and the wireless backhaul links "*s0 – s1*" and "*s1 – s3*". As illustrated in Figure 2-23, the previous chunks support two different virtual Wi-Fi services. The high priority chunk instantiates two virtual APs in nodes *s2* and *s3* radiating BSSID: "*5GPICTURE-2*" and pushes a wireless backhaul tunnel between nodes *s2* and *s3*. Similarly the low priority chunk supports a wireless service radiating BSSID: "*5GPICTURE-1*" in nodes *s0* and *s1*, and pushes two wireless backhaul tunnels to route traffic from *s0* to *s3* and from *s1* to *s3*.

The aforementioned chunk and service abstractions are provided by the SWAM management plane to the MDO through the IF2 interface, which based on REST. Figure 2-24 depicts the various available end-points in this interface that include a view of the Topology, which reports on the I2CAT boxes present in one deployment, end-points to create a chunk in the infrastructure, and end-points to deploy a SWAM service.

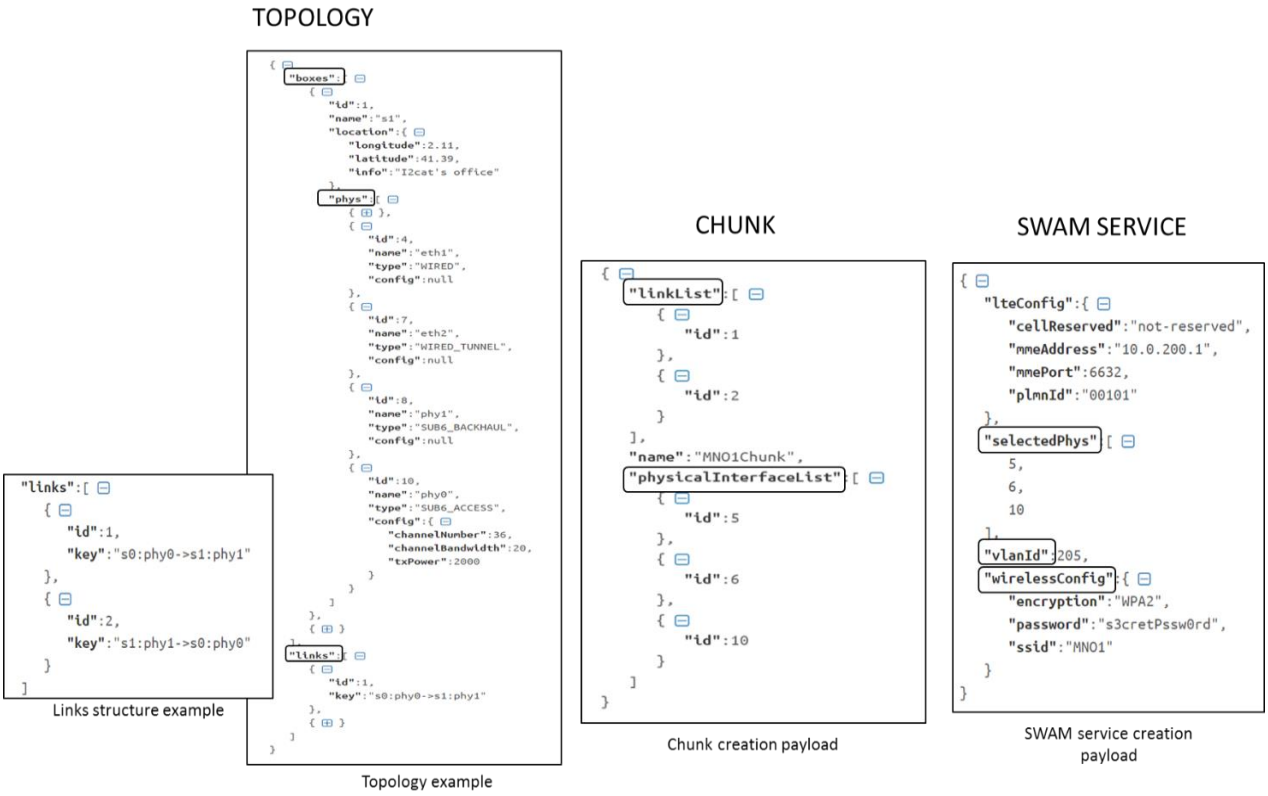

Figure 2-24: REST endpoints offered by the SWAM management plane to the MDO in the IF2 interface.

Finally, we highlight the fact that the IF2 interface is abstract enough to be used by different clients. In WP5 this client is the MDO that includes IF2 as part of its end-to-end service provisioning workflow, but additional clients can be built. For example Figure 2-25 depicts a GUI client that has been built to provision SWAM services over the joint access and backhaul infrastructure that will be used in WP6. The dashboard shows three boxes representing 3 SWAM nodes connected with two wireless backhaul links, and exposing several physical interfaces (red triangles) over which virtual access points can be deployed.

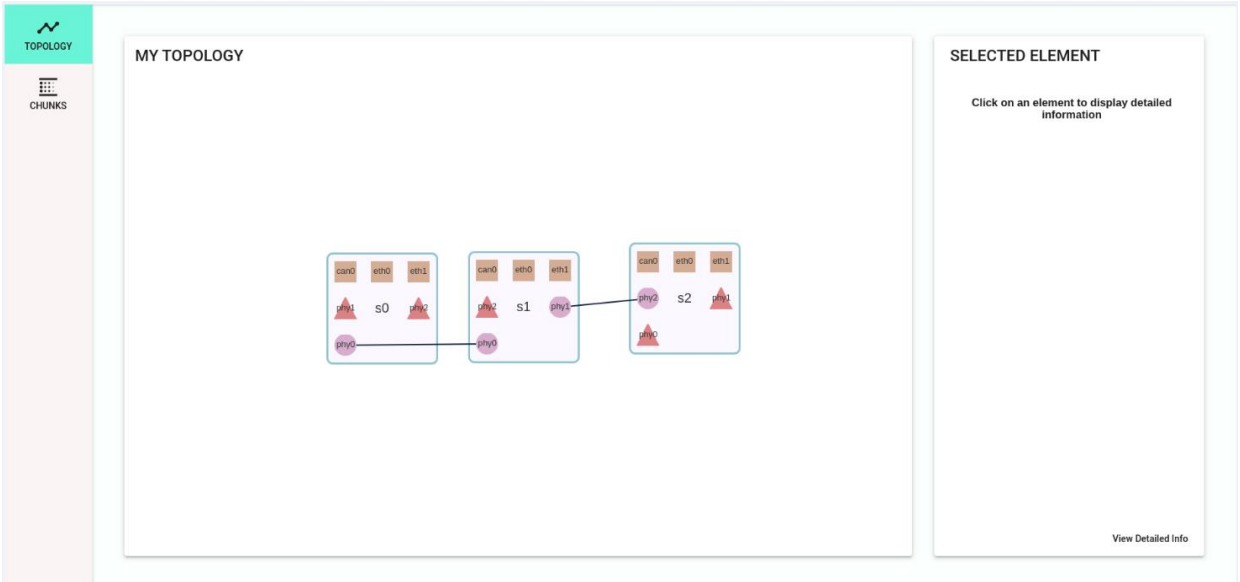

Figure 2-25: Dashboard for the SWAM management system.

2.5.4 Design of interface IF3

In this section we describe the design of the IF3 interface, which allows the MDO to communicate with the Top Controller in the hierarchical control plane described in section 2.4. The interface is a REST API, which is used by MDO to setup the tunnels between the ETNs. The HTTP requests sent by MDO and received by the Top Controller through this interface are translated to a sequence of HTTP requests sent to the underlying L1 and L0 Controllers, as it is described in detail in deliverable D4.3 [6].

3 Benchmarking the 5G OS prototype

In this section we benchmark the service provisioning time delivered by our 5G OS prototype. For this purpose we first define two end-to-end services, and then evaluate the provisioning times of each of the service component domains, as well as the end-to-end service provisioning times.

The main KPI of our evaluation relates to the overall 5G-PPP KPI of "*Reducing the average service creation time cycle from 90 hours to 90 minutes*" as reported in [19].

3.1 Considered 5G OS end-to-end services

We describe in this section the two end-to-end services that will be used to benchmark the service provisioning times using the 5G OS, namely:

- A virtual cellular service between the UTH and UPB domains, whereby an OpenAirInterface (OAI) [16] powered eNB will be instantiated at the UTH domain, and the corresponding OAI core network at the UPB domain.
- A virtual Wi-Fi service instantiated between the I2CAT and UPB domains, which will allow mobile devices over at the I2CAT lab to access a virtual service (Web server) instantiated over at the UPB domain.

Notice that the two considered services require orchestrating VNFs across different compute domains, which use custom MANO orchestrator solutions, and deploying a multi-domain connectivity service across heterogeneous transport technologies. Hence, we claim that the successful deployment of this service demonstrates the viability of the 5G OS architectural vision put forward by 5G-PICTURE.

3.1.1 Service 1: Multi-domain virtual LTE network

The virtual LTE service will be instantiated over the UTH and UPB domains and involves the MDO, the IF1, the IF1' and the IF3 interfaces.

Figure 3-2 details the signaling flow required to provision the service, which we detail next:

- The MDO starts by invoking the IF1 interface to deploy a virtual LTE service. The MDO indicates to OSM over IF1 that it wants to deploy the OAI VNFs for the LTE access support. The data-plane packets belonging to this service tagged with VLAN 20. OSM completes the VNF deployment and returns the MAC address of the VNF (MAC 1).
- The second step if for the MDO to instantiate the LTE core network service at the UPB domain through the IF1' interface. Notice that the MDO also provides a service ID equal to VLAN 20 over the IF1' interface. This means that the UPB domain needs to re-direct all traffic tagged with VLAN 20 to this service instance. The UPB domain returns to the MDO the MAC address of the deployed OAI VNF (MAC 2), which is managed by OpenStack.
- The third step is for the MDO to invoke a multi-domain connectivity service binding the VLAN 20 tagged traffic between the UTH and UPB domains. For this purpose, the MDO interacts with the ETN controller through the IF3 interface requests and end-to-end path setup between the ETN functions of the UTH domain and the ETN function of the UPB domain. The respective ETN functions have registered at the ETN controller at the network provisioning stage. Notice that the MDO needs to provision the ETN controller with the MAC addressed of the involved network function endpoints retrieved over the IF1 and IF1' interfaces. This is required so that the ETN controller can deploy ETN-MAC bindings in the ETN data-paths.
- The ETN controller then interacts with the L1 controller using COP to request the end-to-end connectivity, obtaining in result the network specific VLANs at each domain, i.e. UTH, ZN and UPB. Notice that these VLANs are different from the aforementioned service identified (VLAN20), and are

used by the underlying L0 controllers to steer traffic in each domain. Armed with this information, and the MAC addresses received from the MDO, the ETN controller can program the service bindings in the ETN and IATN functions, e.g. encapsulating traffic received with VLAN20 at the UTH domain with a PBB header and pushing the network VLAN that will forward this traffic to the appropriate IATN connecting to the ZN domain; Similarly, the ETN controller programs IATNs to implement VLAN translation between domains.

- Once the end-to-end connection is available the ETN controllers notified the MDO, and traffic can start flowing between the involved end-points.

In addition, in order to prove that the virtual LTE service is indeed deployed correctly and packets can flow from a device attached to the virtual LTE access network deployed in the UTH domain and reach the OpenStack operated LTE CN deployed in the UPB domain, we depict in Figure 3-1, a set of Wireshark captures in the various domains, showing that:

- Point 1: A standard Ethernet packet leaves the device connected to the virtual LTE DU.
- Point 2: When leaving the UTH domain, packets are tagged with VLAN 10, which is used as the end-to-end identifier for this virtual layer 2 service. Moreover, the ETN function at UTH looks up the destination MAC address, and binds the packet to the tunnel that has been provisioned through the control plane. This is done by adding an extra VLAN header with VLAN tag 97 used to indicate the path that the packet needs to follow.
- Point 3: After reaching the IATN function joining the UTH and the ZN domains, the packet's outer VLAN is swapped to VLAN 101, which signals the slice set up in the ZN domain to forward the packets until the UPB's domain.
- Point 4: The IATN function joining the ZN and UPB domains swaps again the outer VLAN ID to 311, which signals the path that packets need to follow within the UPB domain to reach the UPB ETN.
- Point 1 (final): The UPB ETN decapsulates the VLAN headers and delivers the packet to the VNF.

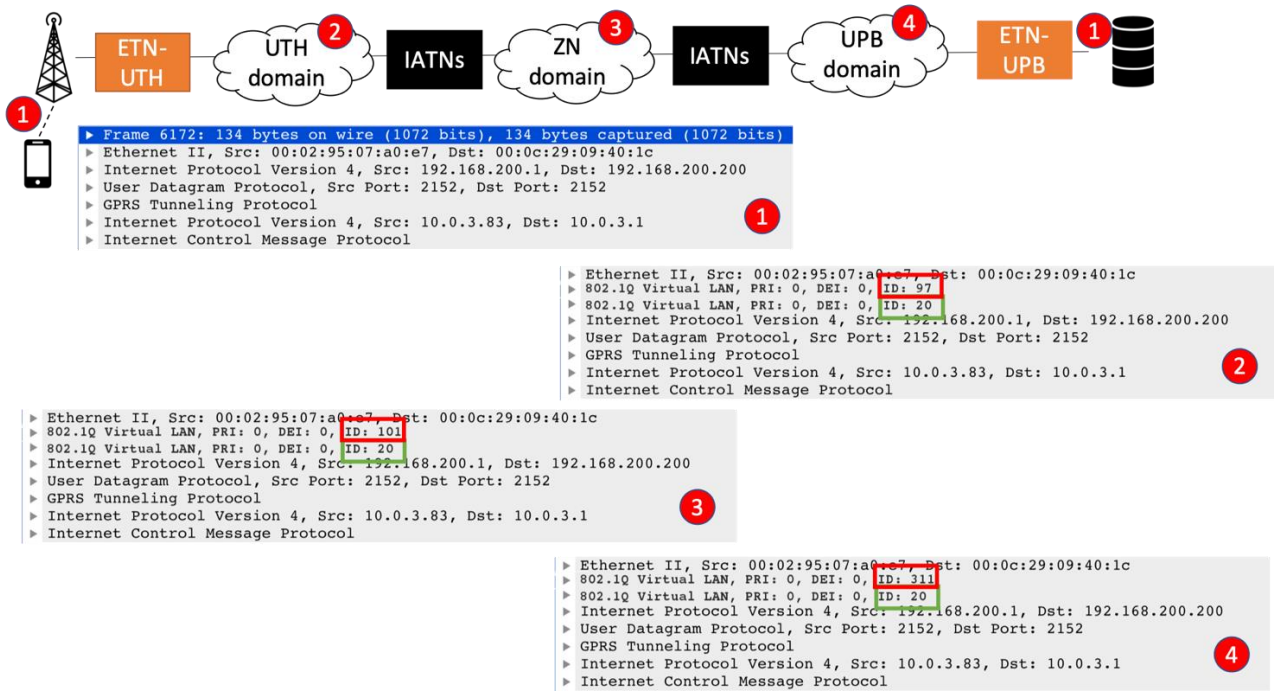

Figure 3-1: Data plane captures of the provisioned virtual LTE service.

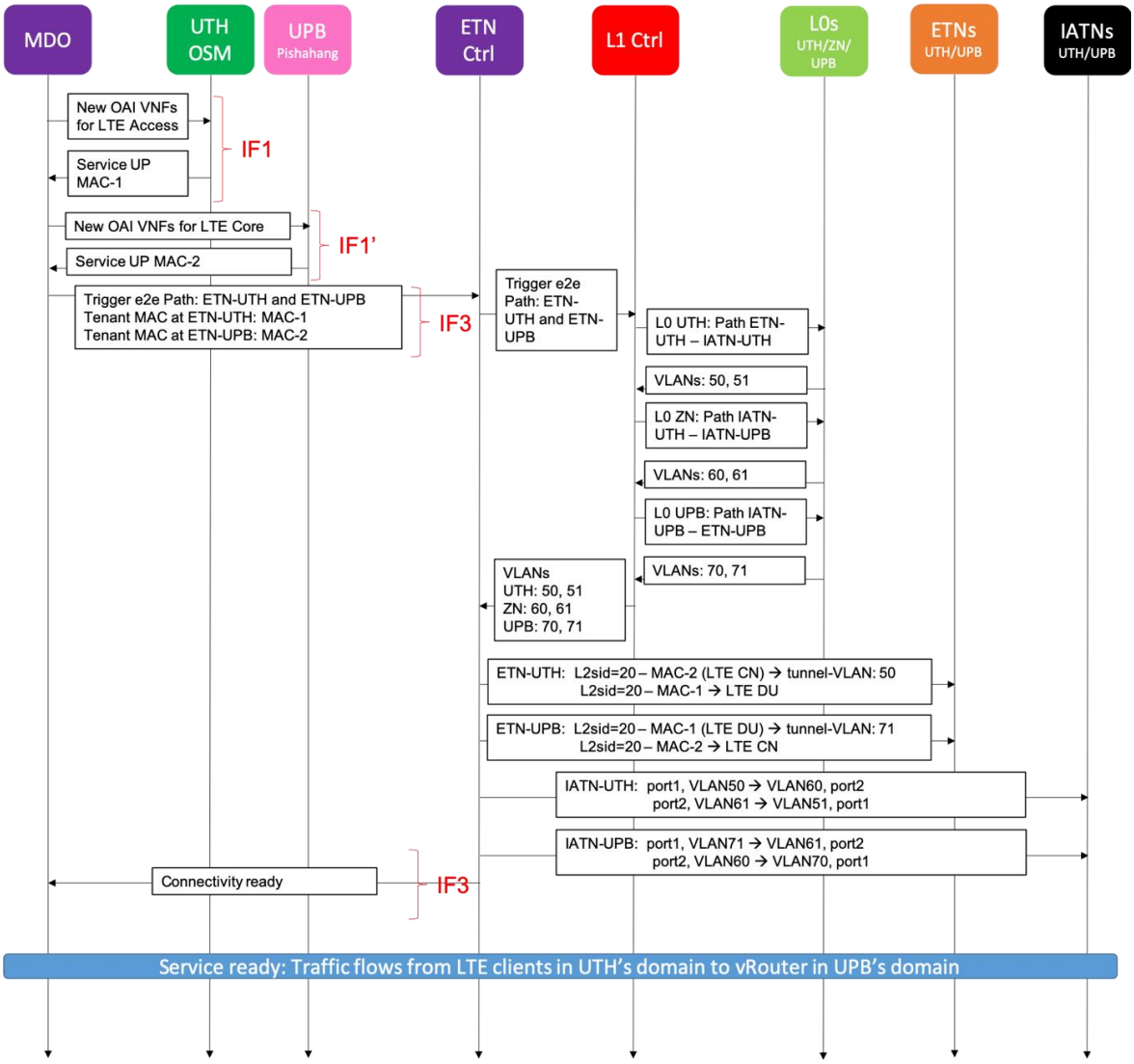

Figure 3-2: Signalling flow from MDO to UTH, Transport Network and UPB domains.

3.1.2 Service 2: Multi-domain virtual Wi-Fi network

The virtual Wi-Fi service will be instantiated over the I2CAT and UPB domains and involves the MDO, the IF2, and the IF1' and the IF3 interfaces.

Figure 3-4 details the signalling flow required to provision the service, which we detail next:

- The MDO starts by invoking the IF2 interface to deploy a virtual Wi-Fi service. The MDO indicates to the SWAM management plane over IF2 that it wants to receive the data-plane packets belonging to this service tagged with VLAN 10. The SWAM management plane completes the service instantiation and returns a white list with the MAC addresses of the clients that will be allowed to connect to the service (MAC 1 and MAC2). Notice that this is a limitation of our current implementation. A production ready solution should allow to connect end user devices to a service dynamically, for which we would require a dynamic mechanism to provision MAC-ETN bindings, as we discuss later.
- The second step if for the MDO to instantiate the core network service at the UPB domain through the IF1' interface. In this case the core network service consists of a DHCP server, a virtual router and a Web server deployed inside a container. Notice that the MDO also provides a service ID equal to VLAN 10 over the IF1' interface. This means that the UPB domain needs to re-direct all traffic tagged

with VLAN 10 to this service instance. The UPB domain returns to the MDO the MAC address of the deployed container.

- The third step is for the MDO to invoke a multi-domain connectivity service binding the VLAN 10 tagged traffic between the I2CAT and UPB domains. For this purpose the MDO interacts with the ETN controller through the IF3 interface requests and end-to-end path setup between the ETN functions of the I2CAT domain and the ETN function of the UPB domain. The respective ETN functions have registered at the ETN controller at the network provisioning stage. Notice that the MDO needs to provision the ETN controller with the MAC addressed of the involved network function endpoints retrieved over the IF2 and IF1' interfaces. This is required so that the ETN controller can deploy ETN-MAC bindings in the ETN data-paths.
- The ETN controller then interacts with the L1 controller using COP to request the end-to-end connectivity, obtaining in result the network specific VLANs at each domain, i.e. I2CAT, ZN and UPB. Notice that these VLANs are different from the aforementioned service identified (VLAN10), and are used by the underlying L0 controllers to steer traffic in each domain. Armed with this information, and the MAC addresses received from the MDO, the ETN controller can program the service bindings in the ETN and IATN functions, e.g. encapsulating traffic received with VLAN10 at the I2CAT domain with a PBB header and pushing the network VLAN that will forward this traffic to the appropriate IATN connecting to the ZN domain; similarly the ETN controller programs IATNs to implement VLAN translation between domains.
- Once the end-to-end connection is available the ETN controllers notified the MDO, and traffic can start flowing between the involved end-points.

In addition, in order to prove that the virtual Wi-Fi service is indeed deployed correctly and packets can flow from a device attached to the virtual Wi-Fi AP deployed in the I2CAT wireless access domain and reach the Kubernetes container deployed in the UPB domain, we depict in Figure 3-3 a set of wireshark captures in the various domains, showing that:

- Point 1: A standard Ethernet packet leaves the device connected to the virtual Wi-Fi AP.
- Point 2: When leaving the I2CAT wireless access domain packets are tagged with VLAN 10, which is used as the end-to-end identifier for this virtual layer 2 service
- Point 3: When receiving the packets marked with VLAN 10, the ETN function at i2CAT looks up the destination MAC address, and binds the packet to the tunnel that has been provisioned through the control plane depicted in Figure 3-4. This is done by adding a PBB header with VLAN tag 113 used to indicate the path that the packet needs to follow. Notice that the PBB header has a source MAC address the one from the I2CAT ETN function, and uses a broadcast MAC as destination. This is done like this because forwarding is done only based on the path identifier (VLAN in the PBB header), and not the destination MAC address.
- Point 4: After reaching the IATN function joining the I2CAT and the ZN domains, the packet's outer VLAN is swapped to VLAN 102, which signals the slice set up in the ZN domain to forward the packets until the UPB's domain.
- Point 5: The IATN function joining the ZN and UPB domains swaps again the outer VLAN ID to 533, which signals the path that packets need to follow within the UPB domain to reach the UPB ETN.
- Point 6: The UPB ETN decapsulates the PBB header and delivers the packet to the Kubernetes node using VLAN 10, which is the service identifier used by Kubernetes to bind the packet to the correct container provisioned for this service.
- Point 7: Finally the packet reaches the Kubernetes container. It is worth noticing that the destination IP and MAC address of this packet are changed. This is a standard Kubernetes behavior, whereby external devices speak only to the Kubernetes node, which then maps the request to the internal container.

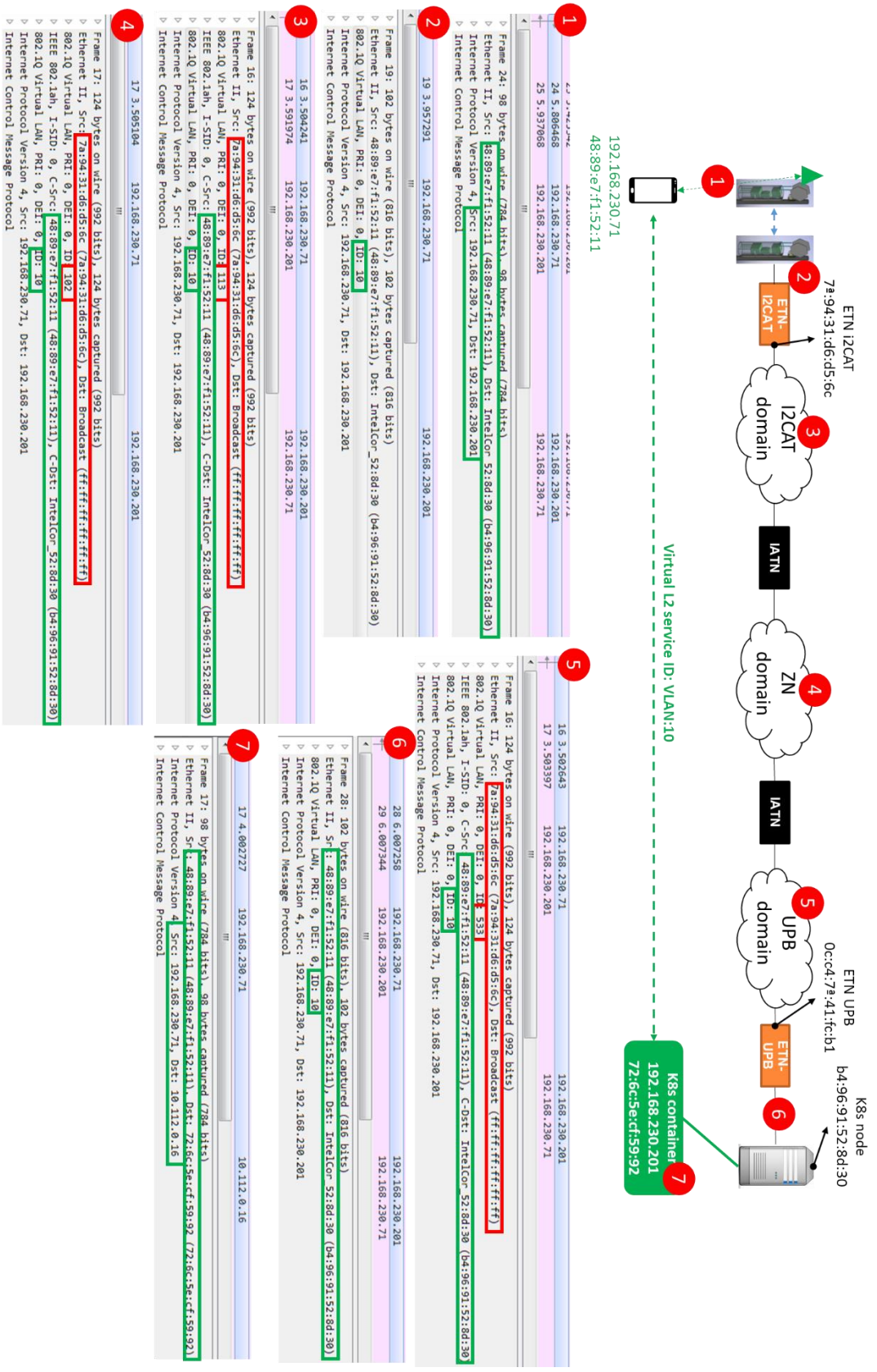

Figure 3-3: Data plane captures of the provisioned virtual Wi-Fi service.

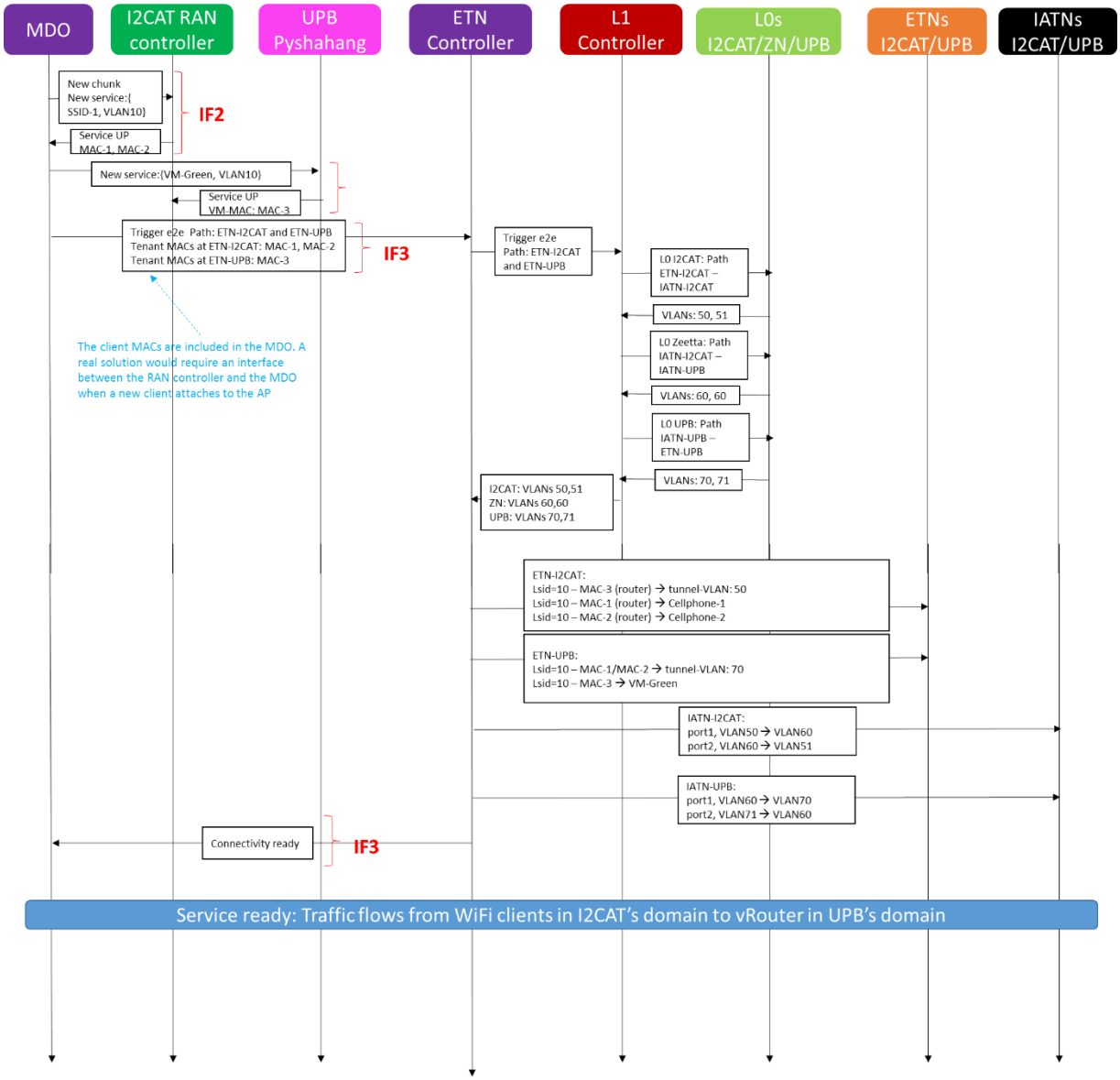

Figure 3-4. Signaling flow in the virtual Wi-Fi service.

3.2 Benchmarking the 5G OS service provisioning times

We benchmark in this section the service provisioning times of our 5GOS prototype. We start first benchmarking the performance of connectivity services across the Transit Provider, and then study the service provisioning times for the IF1, IF1', IF2 and IF3 interfaces in each of their respective domains. We conclude depicting the overall provisioning times of the two end-to-end services described in the previous section.

3.2.1 Benchmarking the Transit Provider and Dynamic Slicing Engine

In this section we benchmark the performance of ZN's DSE using the testbed described in section 2.

Using 'iperf' generated UDP traffic, connectivity between sites was tested using two different classes of service: Gold and Silver. Gold uses metering (within ZN Transit domain only) of 100 Mbps, whereas Silver uses metering of 10 Mbps.

It is difficult to generalise these results outside the ZN domain because:

1) Rest of the domains provide no traffic metering or QoS.

2) This network had 'best effort' VPN-based, inter-site links to connect the different test-beds.

Data was collected to validate the presence of transit connectivity via slices that meter traffic to ensure fair use by multiple tenants without exposing any details.

This data is compared in Figure 3-5 (for Jitter), Figure 3-6 (for Bandwidth) and Figure 3-7 (for Transfer).

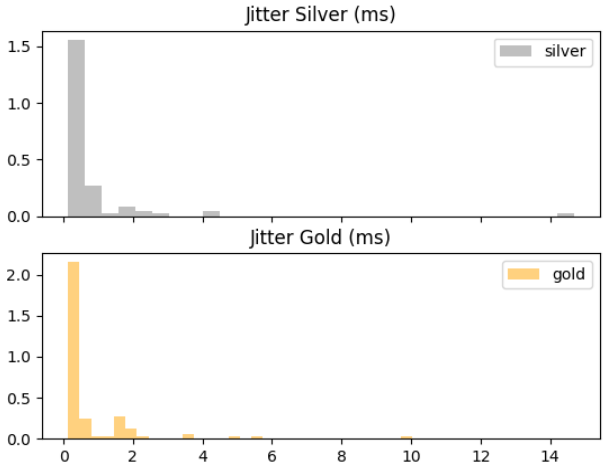

Figure 3-5: Jitter Comparison between Gold and Silver Service Classes (Histogram).

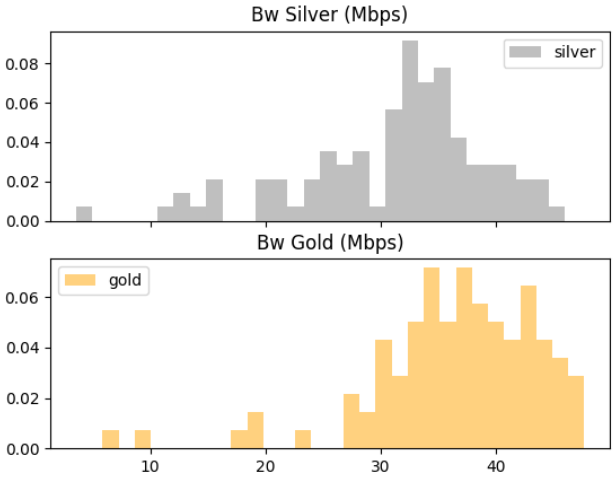

Figure 3-6: Bandwidth (Bw) Comparison between Gold and Silver Service Classes (Histogram).

Figure 3-5 shows a normalised histogram for jitter in Gold and Silver service class. We can clearly see that the Gold class of service provides more reliable performance in terms of Jitter (measures in milliseconds). In the Silver case jitter is more pronounced and unreliable (see outliers ~17 ms). One possible explanation for the decreased reliability of jitter in case of silver is that the ZN domain acts as a bottleneck (see next section).

Figure 3-6 shows a normalised histogram of the end-to-end bandwidth in case of Gold and Silver service classes. Here while it is impossible to see strict adherence to the bandwidth (given non-controlled inter-site links and limited traffic metering), we can still see a significant difference between the two even though the bandwidth for silver is not restricted to 10 Mbps anywhere outside the ZN testbed. This is because while different sites are supporting higher bandwidths (~50 Mbps) than the metered ZN transit network, only in the Gold case is the transit bandwidth much higher than what can be achieved over non-controlled, VPN--based inter-site links. In case of Silver the ZN transit acts as a bottleneck keeping the overall Bandwidth down.

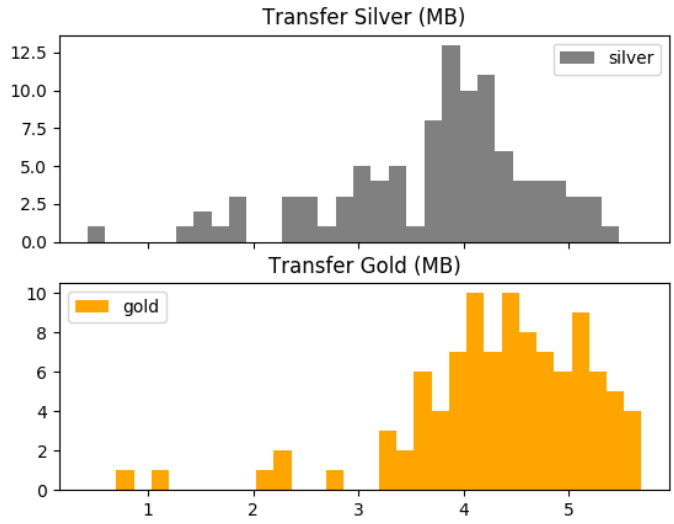

Figure 3-7: Transfer Rate Comparison between Gold and Silver Service Class (Histogram).

Table 3-1: Analysis of Gold and Silver Service Class and Impact on End-to-End Performance.

Gold	**Std**	**Mean**	**Count**		*t*	*df*
Bandwidth	7.41	36.5	100		4.837550807	198
Transfer	0.88	4.35	100		4.837574045	198
Silver	**Std**	**Mean**	**Count**			
Bandwidth	8.07	31.2	100			
Transfer	0.96	3.72	100			

Higher transfer rates (see normalised histogram for Transfer in Figure 3-7) are also apparent in case of Gold Service. This is again because Gold Service overprovisions transit compared to what the inter-site links can provide. Where-as the Silver Service under provisions transit.

Analysis

In this section we prove that Gold and Silver slices lead to different performance in the end-to-end test case (using iperf UDP traffic). Table 3-1 shows the 't-test' between Bandwidth and Transfer for Gold and Silver class of services. The t-score at degrees of freedom = 198, allows us to reject the null hypothesis and conclude that there is significant difference in Bandwidth and Transfer between Gold and Silver transit slices.

Since each service is mapped to a Slice instance within the ZN domain, we can measure the slice setup time and see how it changes between subsequent requests. Ideally the slice creation time should be a constant, low value.

When we turn to Slice Creation Time for First and Second service calls (similar topology requested) we find that (see **Error! Reference source not found.**):

1) The first slice (for the first service) is created slightly quicker than the second slice
2) Average slice creation time for the first service to be requested is ~ 18.2 seconds
3) Average slice creation time for the second service to be requested is ~ 19.8 seconds

Table 3-2: Slice Creation Times for first and second service calls.

	First Service Setup Time (sec)	Second Service Setup Time (sec)
	18	17
	17	19
	18	20
	17	23
	21	20
Average	18.2	19.8

This is a sub-optimal result, the cause of which has been identified as the safe application of device configuration. Currently, for each new service created, the full configuration that takes into account existing services, is generated and applied. This is also the most basic interaction model supported by legacy as well as modern devices.

While this provides an 'always consistent' device-level configuration, the network convergence times will increase with each new service that is instantiated.

One possible solution is to provide 'delta' calculations for configuration and applying the delta rather than full configuration. This leads to higher computational complexity for delta calculations but potentially smaller configuration change on the device leading to faster network convergence times.

3.2.2 Provisioning times over IF1

In this section, we benchmark the performance of the IF1 interface to OSM, presenting the average provisioning times for all OAI components required for the LTE service.

We deploy four different VNFs for the LTE network service as follows:

- A Core Network VNF, running the OAI Core Network software.
- A CU VNF, running the PDCP and above layers.
- A DU VNF providing the LTE access part of the network.
- A Wi-Fi DU VNF, in charge of transmitting/receiving data to/from the Wi-Fi UEs of the network from/to the CU side.

For the DU sides of the network, we use PCI pass-through for the LTE DU VNF, in order to attach the USB3 controller that hosts the SDR device needed for transmitting data over the network. We employ a similar approach for the Wi-Fi DU, but handles the modes of the wireless card. Hence, we pass-through the PCI device that holds the Wi-Fi card of the network. The Wi-Fi DU part of the network encapsulates data received from the UEs in its own format (details on the framework are shown in [23]) and transmits them to the CU. The reverse process is done for downlink traffic. We use the extensions in the orchestrator in order to support a Wi-Fi based path in the backhaul network (between the routing VNF and the Core Network). The topology for the deployed VNFs is shown in Figure 3-8.

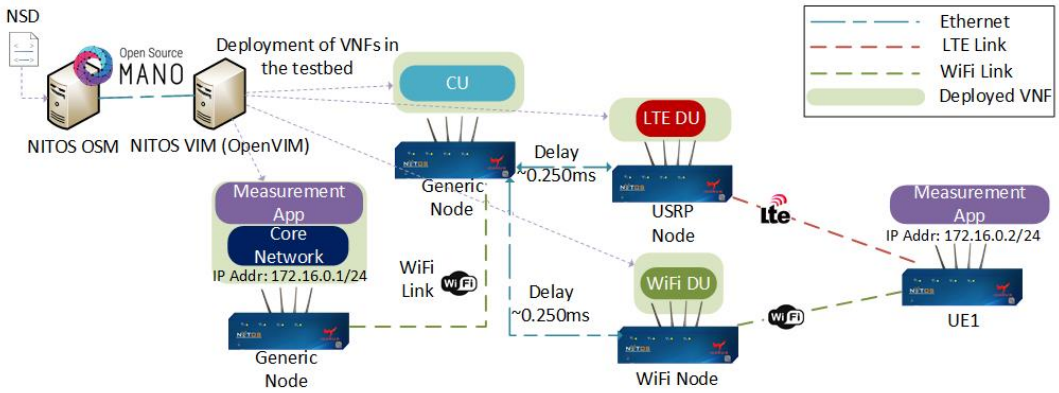

Figure 3-8: The network topology in the UTH domain, used for the deployment of the LTE service.

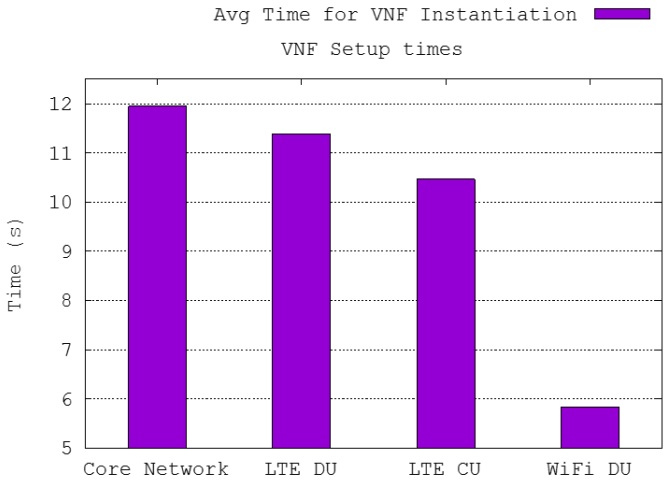

Figure 3-9: VM provisioning times over the IF1 interface in the UTH domain.

Figure 3-9 illustrates the respective times for the instantiation of each VNF involved in the service descriptor. The entire time for deploying the network service is approximately 96 seconds, averaged from 20 experiment runs. As we see, the two VNFs that are using our extensions for Wi-Fi connectivity (Core Network and DU VNFs) take approximately the same time as for the VNFs. The Wi-Fi DU and CU VNFs take less time, as the first requires only a pass-through connection for the wireless interface and the second is more lightweight.

3.2.3 Provisioning times over IF1'

Figure 3-10 depicts the measured provisioning time for the containerized DHCP server VNF that was deployed using the Pishahang MANO framework. In this particular experiment, the DHCP server has been deployed five times on a Kubernetes domain. Figure 3-10 (middle) shows the time that has been taken by Kubernetes to deploy the service. In Figure 3-10 (right), you can see the time required by Pishahang MANO framework to prepare the infrastructure, deploy the service, and store the service metadata. Finally, Figure 3-10 (left) depicts the overall time for provisioning the container-based DHCP server VNF.

We conclude that a typical VNF such as a DHCP can be deployed in only a few seconds, which is a result very well aligned with the target of provisioning end-to-end services in few minutes.

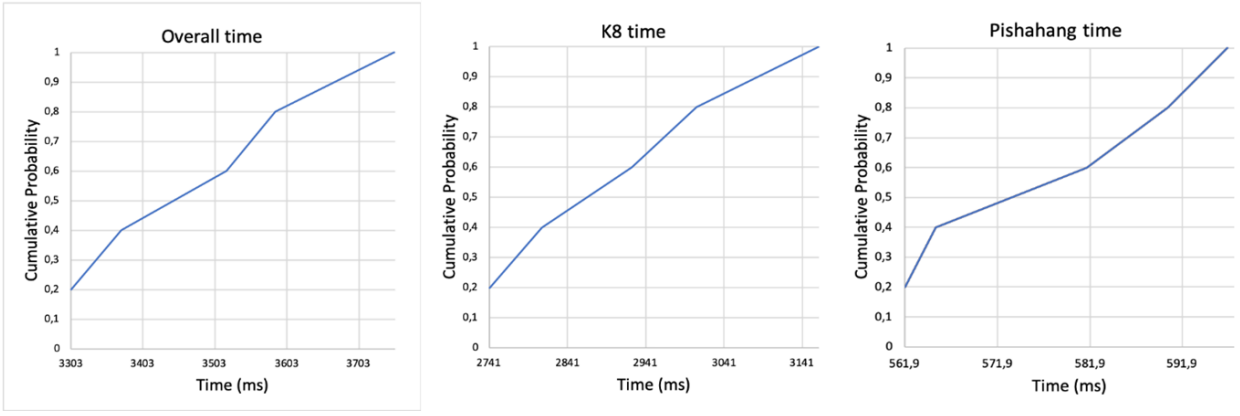

Figure 3-10: Benchmark of the container provisioning times over the IF1' interfaces in the UPB domain.

3.2.4 Provisioning times over IF2

In this section we evaluate the times over the IF2 interface required to instantiate virtual Wi-Fi services. For this purpose we set up a linear topology with four nodes as depicted in Figure 3-11. Over these topology we setup a chunk that contains all the backhaul links and wireless access interfaces, and sequentially deploy four

different services Service 1 to Service 4, where Service *i* contains *i* wireless access interfaces, each one in a different node, and *(i-1)* backhaul links between nodes. These services are depicted in Figure 3-11.

Figure 3-12 depicts the measured service provisioning times for Services 1-4 where before provisioning a new service the previous one is deleted. We can see that the SWAM implementation delivers service provisioning times in all cases around 10 seconds, with a very slight increase as the number of virtual access interfaces and backhaul links increases. The service provisioning time is mostly dominated by fixed timeouts in the SWAM devices required to ensure that the instantiation of virtual access points is executed correctly.

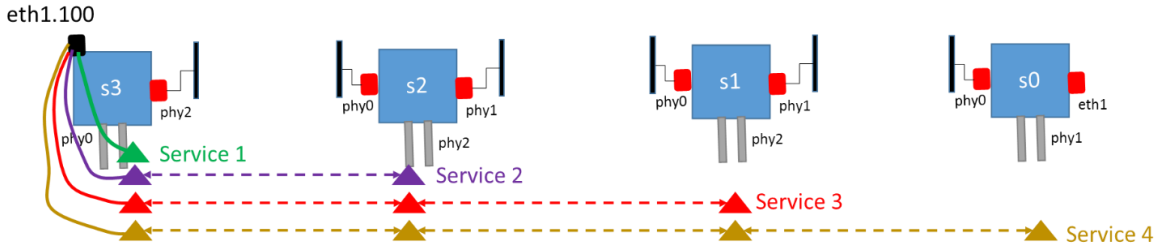

Figure 3-11: Test Setup to evaluate IF2.

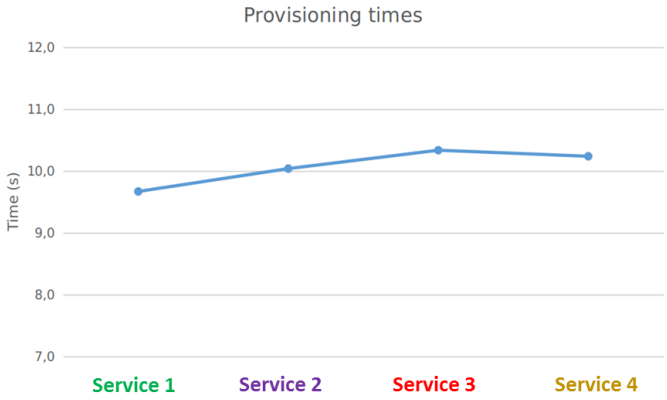

Figure 3-12: Measured service provisioning times.

3.2.5 Provisioning times over IF3

A detailed report on the provisioning times over IF3 is given in deliverable D4.3 [6], where we can see end-to-end connectivity provisioning times from 3 to 15 seconds. The main factor contributing to these provisioning times is the Transit Provider implemented in the ZN domain.

The variance (3 to 15 seconds) in the connectivity provisioning time introduced by the Transit Provider can have multiple reasons, some that can be quantified and others that can't be. A clear factor to be indicated, that is beyond our control, is the OpenVPN based control plane link which provides variable performance. Apart from this there may be few reasons for this variance. Firstly, the DSE is decoupled from the operation of the Tenants. Therefore, any requests for connectivity by the Tenants, must be processed at the Adapter before it is passed to the DSE, where it must be reprocessed. The Adapter must wait for the result, which must be passed back to the Tenant(s) once available. This process currently is synchronous with fixed polling intervals set to 10 seconds between the Adapter and the DSE for the slice creation. In a practical implementation Tenants and the Provider must use some form of asynchronous event based interface to avoid using fixed length timers.

3.2.6 Provisioning times for the end-to-end services

Figure 3-13 depicts the complete end-to-end service provisioning times for the virtual LTE and the virtual Wi-Fi services described in section 3.1, measured in the multi-domain testbed introduced in section 2.

Experiments are repeated 10 times in order to obtain statistically significant results, and we plot the resulting Cumulative Distributed Functions (CDFs) in Figure 3-13. We can see how we can deploy the virtual Wi-Fi network consisting of a virtual access at I2CAT and a DHCP server at UPB in around 30 seconds. We observe more variability in the virtual LTE service, with provisioning times spanning between 90 and 170 seconds. The virtual LTE service is composed of two virtual machines (VMs), one for the eNB and one for the Core Network, instantiated using OSM and Pishahang in the UTH and UPB domains respectively. Naturally, the instantiation time for VMs is higher than the time required to bring up the container-based DHCP function in the virtual Wi-Fi service.

Looking at these results we conclude that the 5G OS has accomplished its design goal of being able to setup and end-to-end service comprising compute functions and connectivity services across a multi-domain network in less than two minutes.

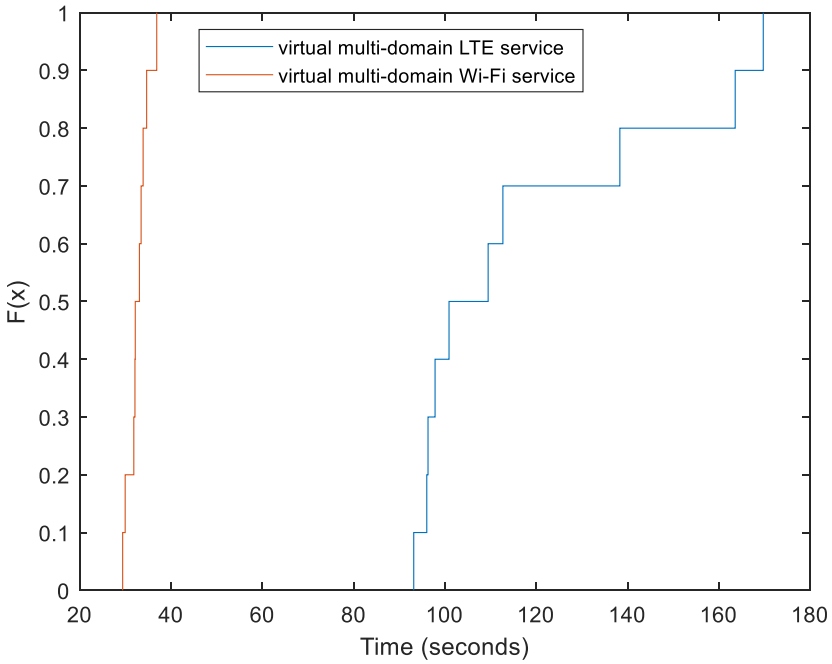

Figure 3-13: Provisioning times for the e2e virtual LTE service and virtual Wi-Fi service over the multi-domain testbed.

4 Complementary 5G OS demonstrations

This section presents complementary demonstrations that have not been integrated in the multi-domain testbed described in the previous section. These additional integrations are used to highlight complementary aspects of the 5G OS. In particular we present three different works. First, in section 4.1 we present the integration of the JoX orchestrator with a management plane to set up TSN equipment. Second, in section 4.2 we describe an extended demonstrator that can manage multi-version VNFs using Pishahang. Third, in section 4.3 we describe the integration of a TSON domain with an alternative MDO using COP.

4.1 Orchestrated Time-Sensitive Networking-enabled Mobile networks

In this research activity we are exploiting JOX open source event-based orchestrator[2] which natively supports network slicing, as an integrated orchestrator for both the mobile network and the Time Sensitive Enabled (TSN)-enabled network. Using JOX, each network slice can be independently optimized with specific configurations on its resources, network functions and service chains.

JOX is mainly composed of two entities: JoX NBI and JoX core. JoX Northbound Interface (NB) exposes a REST API that can be run independently of JoX core through a message broker mechanism, while JoX core is composed of NFVO and common services, as illustrated in Figure 4-1.

JOX was presented in D5.2 [2] and its ability to rapidly deploy both the virtualized infrastructure needed while also the mobile network services necessary for an integrated mobile network. In this activity we also incorporate the Transport network segment and more specifically an IEEE-TSN underlay network.

In principle the IEEE 802.1 TSN TaskGroup focuses on time synchronization issues and physical and link layer techniques to achieve a guaranteed delivery of data with bounded low latency, low delay variation and low loss. Time aware shaping (IEEE 802.1Qbv) and frame preemption (IEEE 802.1Qbu) are the two key mechanisms used to limit Ethernet frame propagation latency, while similar mechanisms regarding the IP layer are investigated by the Internet Engineering Task Force (IETF) Detnet Working Group. An evaluation of the ability of TSN to support fronthaul and backhaul services is included in deliverable D4.3 [6].

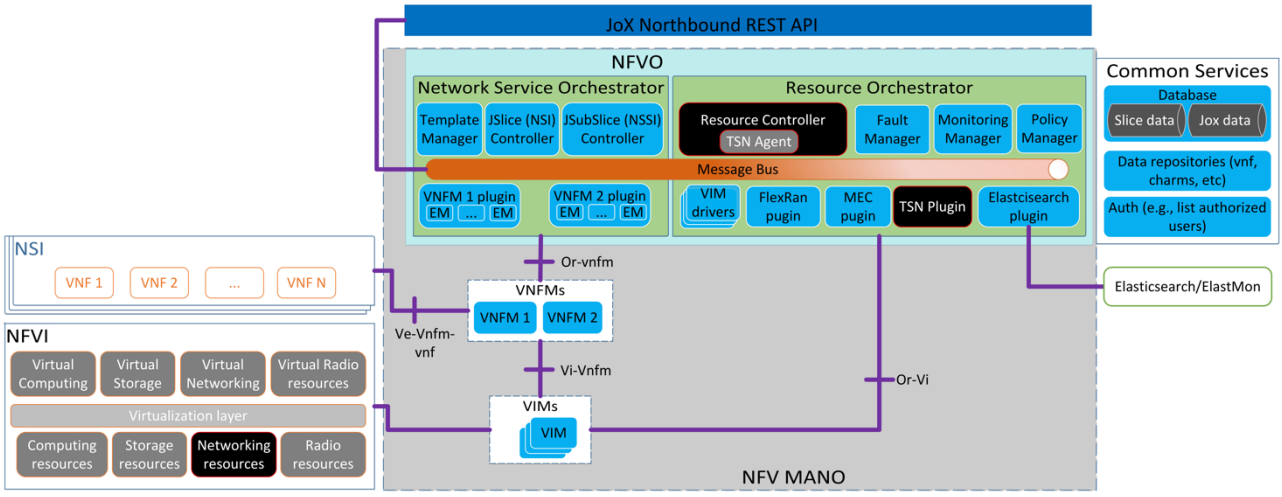

Figure 4-1: JOX High level architecture diagram.

[2] http://mosaic-5g.io/jox/

In the context of this activity a new JOX plugin was implemented in order to interact with a fully functional TSN prototype that was provided by HWDU. Through JOX the new plugin exposes functionalities for the following three categories: i) Time Aware Shaper 802.1Qbv, ii) VLAN/PVLAN configuration and iii) 802.1AS PTP.

As JOX natively offers Network Slice Management Function (NSMF) functionalities as defined in 3GPP TR28.801 [17] in order to support the entire lifecycle of Network Slice Instances (NSIs) with the new TSN plugin, though JOX the Transport segment can be also controlled in an end-to-end orchestrated way. Furthermore with the ability of TSN to provision network pipes with guaranteed performance by means of delay and jitter, QoS can be now provisioned end–to-end.

4.1.1 TSN JOX Plugin Description

In Figure 4-2 the primitives of the TSN plugin architecture for JOX are depicted. The HWDU TSN prototype solution exposes a NETCONF interface that is covering three aspects that are required in order to have a fully functional TSN solution: a) 802.1AS configuration used to enable end-to-end for clock synchronization; b) VLAN information (VID and priority tags); and c) Time Aware Shaping information.

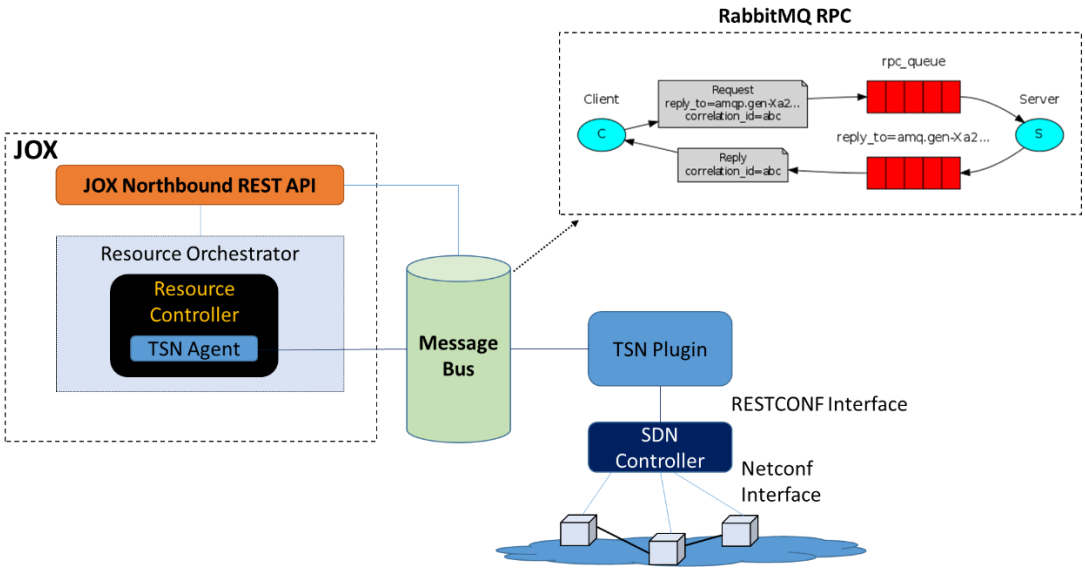

Figure 4-2. TSN agent/plugin implementation.

The ability to provide delay guarantees is supported by techniques like Scheduled Traffic (IEEE 802.1Qbv) and Frame Preemption (IEEE 802.3br, IEEE 802.1Qbu). These standards explain how frames belonging to a particular traffic class or having a particular priority are handled by TSN-enabled bridges. IEEE 802.1Qbv introduces a transmission gate operation for each queue as depicted in Figure 4-3. The transmission gates open/close according to a known time schedule. Scheduled Traffic ensures that the transmissions are controlled by a Gate Control List (GCL) which consists of multiple schedule entries.

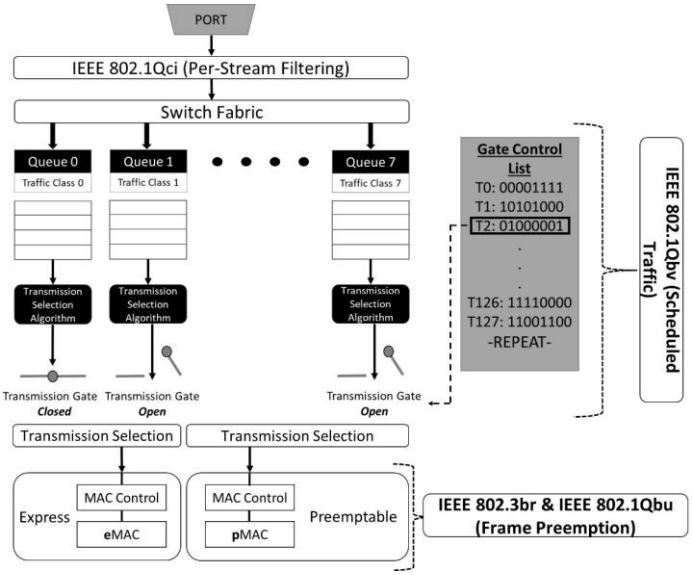

Figure 4-3: IEEE TSN Scheduled Traffic & Frame Pre-emption.

```
container qbv-status {
  list tsntas-status {
    key "interface";
    leaf interface {
      type uint8;
    }
    leaf gate-queue-state {
      type uint32;
    }
    leaf gate-state-set {
      type uint32;
    }
    leaf tick-granularity {
      type uint32;
    }
    leaf max-calendar-entries {
      type uint32;
    }
    leaf overrun-packet-express {
      type uint32;
    }
    leaf overrun-packet-preemption {
      type uint32;
    }
  }
}
container as {
  container as-display-global {
    container tsnptp-display-global {
      leaf disp {
        type string
```

```
container as {
  container as-display-global {
    container tsnptp-display-global {
      leaf disp {
        type string {
          length "0..32768";
        }
      }
    }
  }
  container as-display-interface {
    list tsnptp-display-interface {
      key "interface";
      leaf interface {
        type uint8;
      }
      leaf disp {
        type string {
          length "0..32768";
        }
      }
    }
  }
  container as-interface {
    list tsnptp-interface {
      key "interface";
      leaf interface {
        type uint8;
      }
      leaf pkt {
        type enumeration {
          enum disable {
```

Figure 4-4: TSN YANG files (TAS left, PTP right).

The TSN NETCONF interface exposed can be used to configure the TSN devices by any SDN solution that supports a Netconf client. In our testbed we are using the open source Opendaylight solution that offers the Netconf client but also exposes a RESTCONF interfaces as the Northbound. The RESTCONF interface is automatically generated based on the TSN YANG models that HWDU TSN switches provide. A sample from the YANG files is depicted in Figure 4-4.

For all the three messaging categories (VLAN ,PTP, TAS) the TSN plugin communicates with Opendaylight controller through its RESTCONF interface. The communication between the TSN plugin with JOX is made through a message broker service, implemented using the open source RabbitMQ framework.

A TSN Agent resides on the JOX core side and is actually a consumer or producer of messaging targeting the TSN plugin. With this modular architecture the TSN plugin can run in an independent process even in a different machine from the one JOX is deployed. Furthermore the same message brokering service is also

exploited by the JOX REST interface, however on every function all the necessary wiring inside JOX is updating the right data structures responsible to keep the operational state of every Network Slice Instance.

The TSN plugin exposes the methods depicted in Table 4-1.

Table 4-1: TSN plugin methods exposed to the message bus.

Category	Message
VLAN	vlan_create, vlan_add , vlan_remove, vlan_delete, Pvlan_set (used to set default vlan per port)
802.1AS	display interface, enable, LogSyncInterval, LogAnnounceInterval, LogMinDelayReqInterval, SyncReceiptTimeout, delayAsymmetry, minNeighborPropDelayThreshold, maxNeighborPropDelayThreshold, display clock-parent, display clock, clock domain, clock priority1, clock priority2, clock clockClass, clock profile, clock gmCapable, clock slaveOnly, display program, program timestamping, program delayMechanism, program transport, program timeaware-bridge, program phydelay_compensate, program phydelay_compensate_out, program servo_locked_threshold, program logging-level, program save-config, display time-properties
TAS	tsntas-cycle-time, tsntas-schedule-entry, tsntas-enable-schedule, tsntas-ptp-mode, tsntas-gate-ctrl-period, tsntas-profile, tsntas-status


```
{
        "object":       "tsnptp-interface",
        "oper": "query-config",
        "target":       "running",
        "messageId":    "1",
        "oper-result":  "ok",
        "tsnptp-interface":     {
                "interface":    1,
                "port-sync-state":      "master",
                "port-sync-path-delay": 0,
                "peer-802-dot1as-capable":      "false",
                "work-as-time-awared-bridge":   "false",
                "delay-asymmetry":      0,
                "log-sync-interval":    -3,
                "log-announce-interval":        0,
                "log-min-delay-req-interval":   0,
                "sync-receipt-timeout": 3,
                "transport-specific":   1,
                "max-neighbor-propdelay-threshold":     1000000,
                "min-neighbor-propdelay-threshold":     -1000000
        }
}
```


Figure 4-5. Query config example

As an example in Figure 4-5 we present the result of a query-config function for tsnptp interface:

4.1.2 NBI Extension examples

The functionalities exposed in the message brokering service are also exposed through the NBI Rest interface form JOX. As an example we present two endpoints in the REST API used for the following purposes:

1. Get the list of all supported switches, along with their configurations
2. Create, add, remove, and destroy vlan and their associated ports

Figure 4-6: REST API example: supported switches.

Figure 4-6 illustrates an example to get the configurations of all the switches, where they are sorted by their types, which is only TSN in this example. Since the VLAN and the ports can be used by more than one slice, JoX controls the actions (i.e., create, add, remove, destroy) that can be performed for the objective of preventing the violation of the other slices. Table 4-2 details this policy. For example, this policy prevents a slice from destroying a VLAN that is already used by other slices. Moreover, it also prevents a slice from removing a port used by other slices.

Table 4-2: VLAN policy example.

<table>
<tr><td></td><td colspan="4">Valn vid already exists?</td></tr>
<tr><td rowspan="2">Create vlan vid</td><td colspan="3">Yes</td><td>Non</td></tr>
<tr><td colspan="3">Error, vid already exist</td><td>i) Create vid, ii) register slice to vid</td></tr>
<tr><td></td><td colspan="4">Valn vid already exists?</td></tr>
<tr><td></td><td colspan="3">Yes</td><td>No</td></tr>
<tr><td>Add port prt to valn vid</td><td colspan="3">Port prt already added</td><td rowspan="3">Error vlan vid not exist</td></tr>
<tr><td>Yes</td><td colspan="2">No</td></tr>
<tr><td>Register the slice to the port</td><td colspan="2">i)Add the port to the vlan, ii) Register the slice to the port</td></tr>
<tr><td></td><td colspan="4">vlan already exists?</td></tr>
<tr><td></td><td colspan="3">Yes</td><td>No</td></tr>
<tr><td>remove port prt to valn vid</td><td colspan="3">Port prt exist</td><td rowspan="4">Error vlan vid not exist</td></tr>
<tr><td></td><td colspan="2">Yes</td><td>No</td></tr>
<tr><td></td><td colspan="2">Port prt used by one slice?</td><td rowspan="2">Error: no port prt found for the vlan vid</td></tr>
<tr><td></td><td>Yes</td><td>No</td></tr>
<tr><td></td><td>remove port prt</td><td>Unregister slice from port prt</td><td></td><td></td></tr>
<tr><td></td><td colspan="4">Valn vid already exists?</td></tr>
<tr><td></td><td colspan="3">Yes</td><td>No</td></tr>
</table>

Destroy valn vid	Vlan used by one slice?		Error vlan *vid* not exist
	Yes	No	
	Destroy vlan vid	Unregister slice from vlan vid	

Figure 4-7. (left) Slices Access rights, and (right) config file to create vlan 500 (top right) and output of JoX NBI (bottom right)

Moreover, we implemented also an access rights mechanism, where we defined which are the permitted actions for every slice owner, as illustrated in Figure 4-7. In this figure, the owner of slice "*mosaic5g_slice_1*" has the rights to add/remove ports, while he does not have the rights neither to create nor to destroy a VLAN.

In an implemented use-case, we present an example, which a slice consists of three subslices. The first subslice, in turn, consists of two services; i) mysql and ii) oai-hss that are connected to each other inside the subslicie (i.e., intra-subslice connection). The second subslice is composed of i) oai-mme and ii) oai-spgw that are connected to each other through intra-subslice connection. And, third subslice is composed of oai-enb service, in addition to flexran as RAN controller. The subslices are connected to each other, through inter-subslice connection, which connects oai-hss from the first subslice with oai-mme from the second subslice, whereas oai-mme from second subslice to oai-ran from third subslice. The network fabric to interconnect the different components is IEEE TSN based.

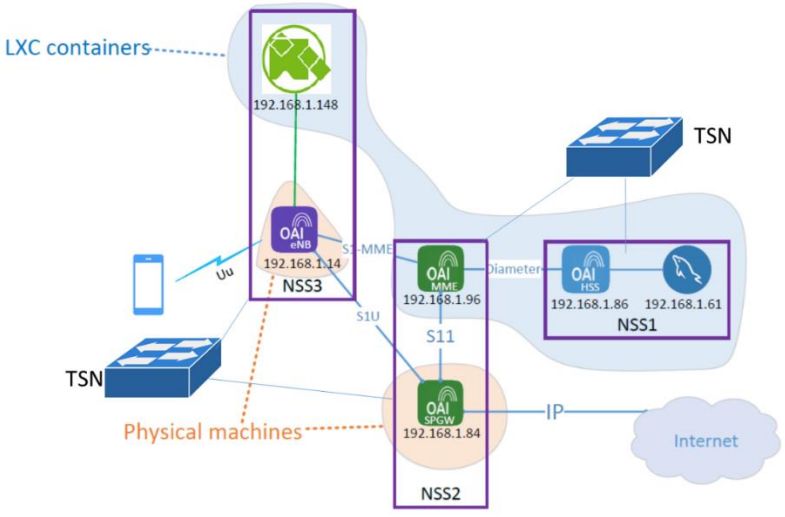

Figure 4-8: Physical network topology.

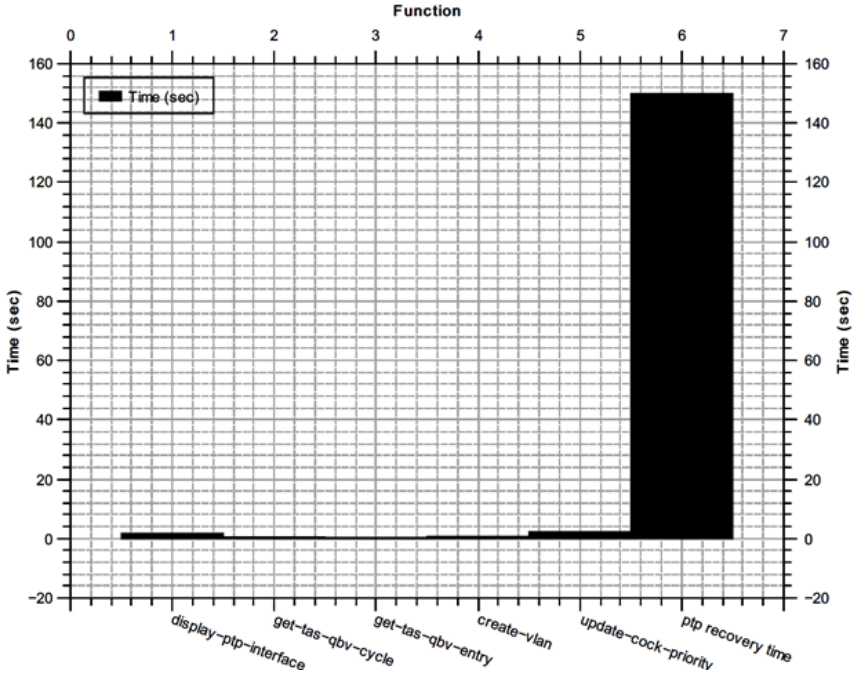

Figure 4-9: TSN network configuration measurements.

Results about TSN configuration times are depicted in Figure 4-9. In principle the overhead introduced by the message broker is in the level of few microseconds (hosted in the same machine with JOX installation). Calls from the JOX agent to the TSN plugin through the message broker and then to OpenDayLight and through Netconf to the TSN switch is in the level of few seconds for TSN and VLAN related operations. However we highlight that PTP convergence times especially in case of PTP errors, are in the level of minutes. PTP clock recovery in case of failures is an open issue.

4.2 Extended support for multi-version VNFs

While Pishahang was already supporting the management different version services namely GPU-, Container-, and VM-based Virtual Network Functions (VNFs), on-the-fly switching of different service versions was not supported. To this end, we have extended Pishahang MANO framework with a new plugin called Multi-version Service Manager (MSM) which allows Pishahang to dynamically switch between different versions of a service as the services requirements change. This plugin has been implemented based on the algorithm implemented in [18]. We have also extended the descriptor schema of the services in Pishahahng to describe multi-version services. An example of descriptor can be seen in Figure 4-10 (right).

A demonstration of the new added features to Pishahang has been planned. For this demonstration, we have implemented two different versions of a Transcoder VNF: (1) Virtual Machine (VM)-based transcoder and (2) Container (CN)-based transcoder. The VM-based transcoder is intended to run only on CPU while the CN-based transcoder run on both CPU and GPU. Transcoder has been chosen as a service example as it can benefit from both CPU and GPU, depending on the user requirements (data rate, video quality) [19]. The topology of the testbed used for demonstration is shown in Figure 4-10 (left). The test-bed topology consists of the following nodes:

- Pishahang running on a VMware Virtual Machine.
- OpenStack running on a Dell workstation.
- Kubernetes running on a Dell workstation equipped with a GPU.
- Zodiac FX OpenFlow Switch provide the connectivity.
- Transcoder client running on a Virtual Machine.
- Ryu controller running on x86 PC

In this testbed, Pishahang is responsible for high-level service management tasks. Kubernetes is used to manage container-based VNFs and OpenStack takes care of VM-based VNFs. Ryu controller is used to manage the Zodiac OpenFlow switch. The switch is (re) configured during the service lifecycle to redirect the traffic to different service versions on the fly. The client is the video viewer which requests multiple video resolutions during the service run. Figure 4-11 shows screenshots from Pishahang, Kubernetes, and OpenStack dashboards along with the transcoded videos that are shown to the client during the demo.

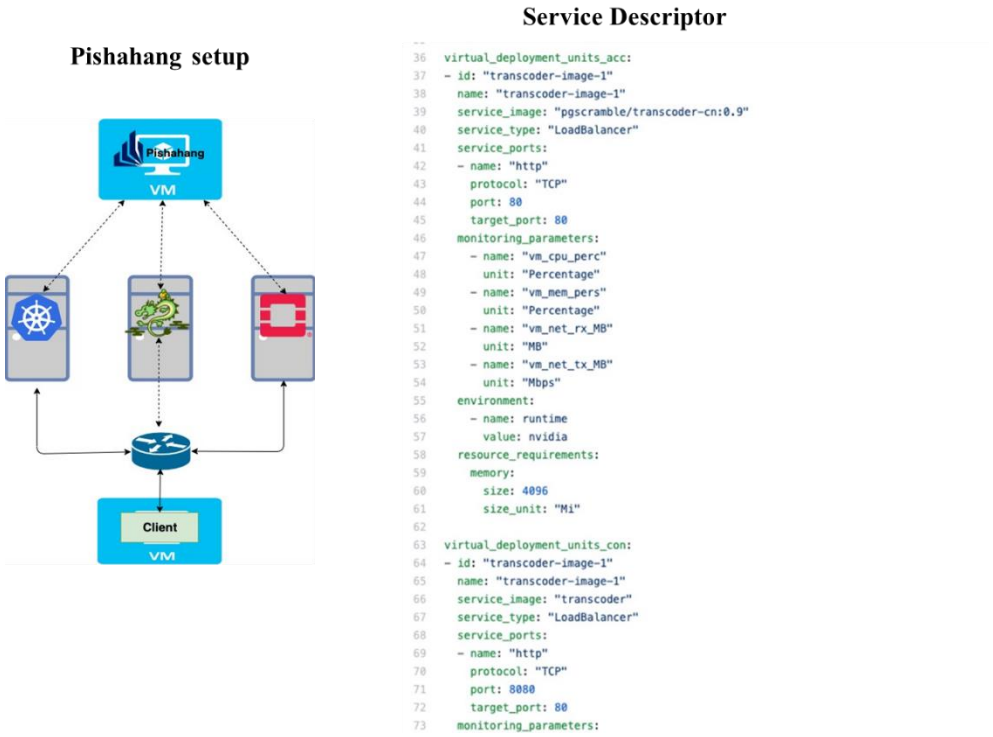

Figure 4-10: Pishahang setup and example descriptor.

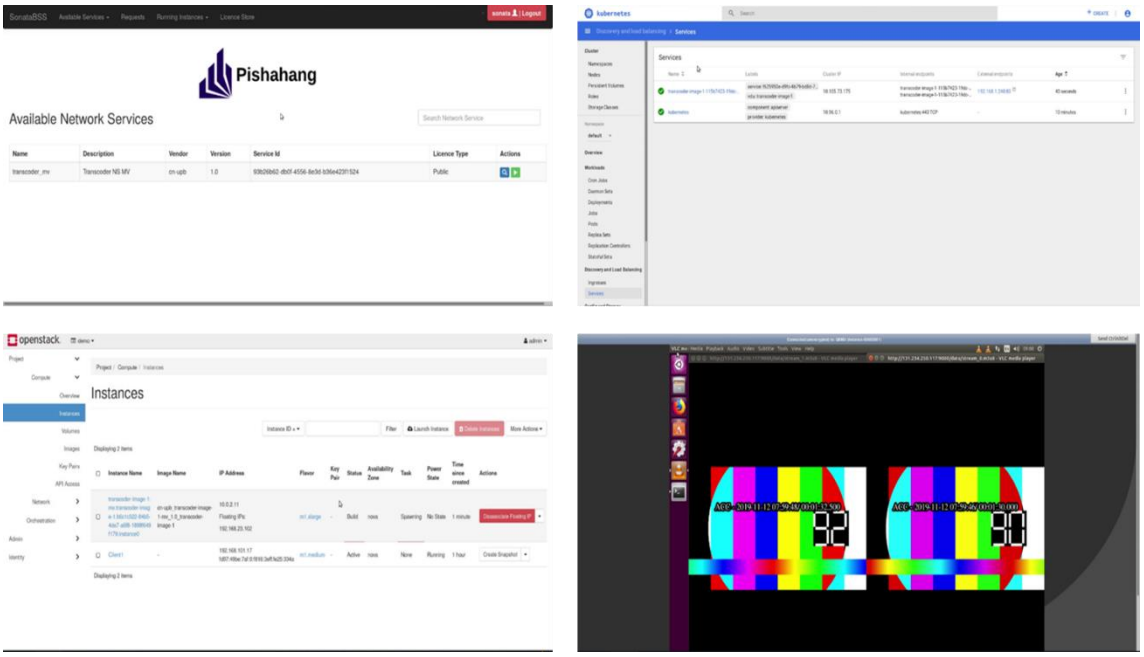

Figure 4-11: Screenshots from multi-version VNF demo.

4.3 Integration of a TSON domain with the 5G OS MDO

The purpose of this demo is to demonstrate the integration of the 5G UK Exchange as MDO within 5G OS. 5GUK Exchange (5GUKEx) is a platform developed by HPN of university of Bristol to orchestrate and interconnect different domains. We are going to demonstrate how to interconnect two different datacentres where each of them is administered by OSM. These datacentres are connected through the TSON domain which is managed by an ONOS SDN controller. In this demo, we are going to deploy 2 virtual network services, where the end to end deployment is managed by 5G UK Exchange. It is going to provision the different OSM managing each datacentre and provision the ONOS SDN controller managing the TSON domain.

The follows parts will show how the integration has been performed by describing the different implemented interfaces (IFs). Finally, the results of this integration will be presented.

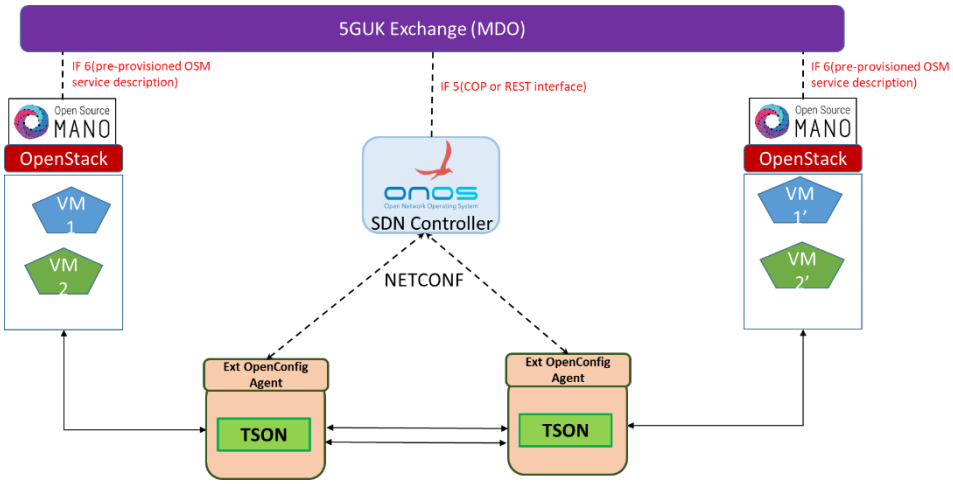

Figure 4-12: Integration of 5G UK Exchange within 5G OS.

4.3.2 5G UK Exchange

5GUK Exchange (5GUKEx) is a novel hierarchical architecture to enable end-to-end orchestration among multiple network administrative domains with minimum overhead in complexity and performance. 5GUK Exchange allows operators to plug-in their ETSI NFV standards compliant NFV Management and Network Orchestration (MANO) systems.

5GUKEx involves multiple components on top of MANO systems at each test network site. Island Proxy uses the NBIs of the MANO system to communicate with the Network Service Broker (NSB). Each MANO shares its Network Service (NS) catalogues with the Broker. The experimenters (on 5GUKEx) can compose the Network Services between multiple test networks to deploy it using the Network Service Manager. On deployment of NS on each test network, the endpoints are shared with the NSB. The Inter-Domain Connectivity Manager (IDCM), uses the endpoints to establish the network connectivity between the participating test networks.

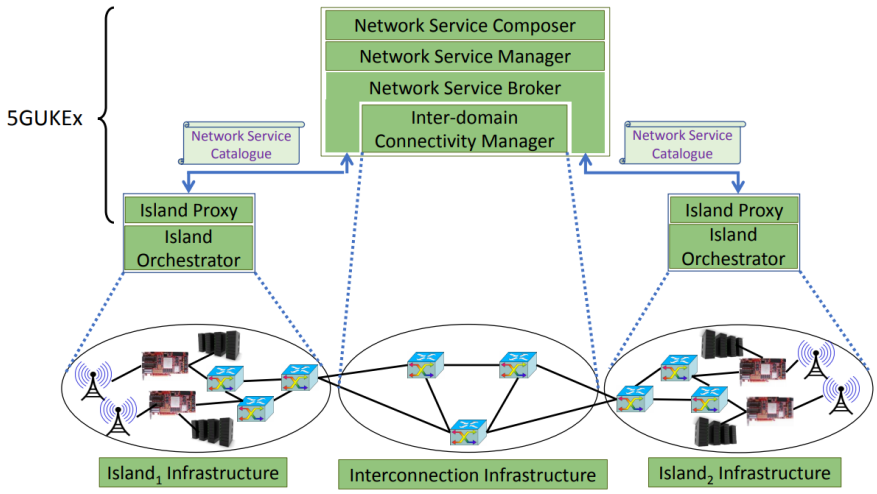

Figure 4-13: 5GUK Exchange Architecture.

4.3.3 Design and implementation of additional interfaces in the 5G OS (IFx)

4.3.1.1 The IF5 interface

IF5 interface defines the communication between the 5GUKEx as MDO and the SDN controller considered as the control point to manage the DO. This interface implements a REST API where the 5GUKEx sends a POST request to the ONOS SDN controller containing the end points representing the TSON edge node connecting the Islands, the client ports where the traffic is sent, the VLAN tag to established the end to end bidirectional communication and the TSON parameters. Figure 4-14 shows the POST request and the detail of the exchanged JSON message are depicts by the Figure 4-15.

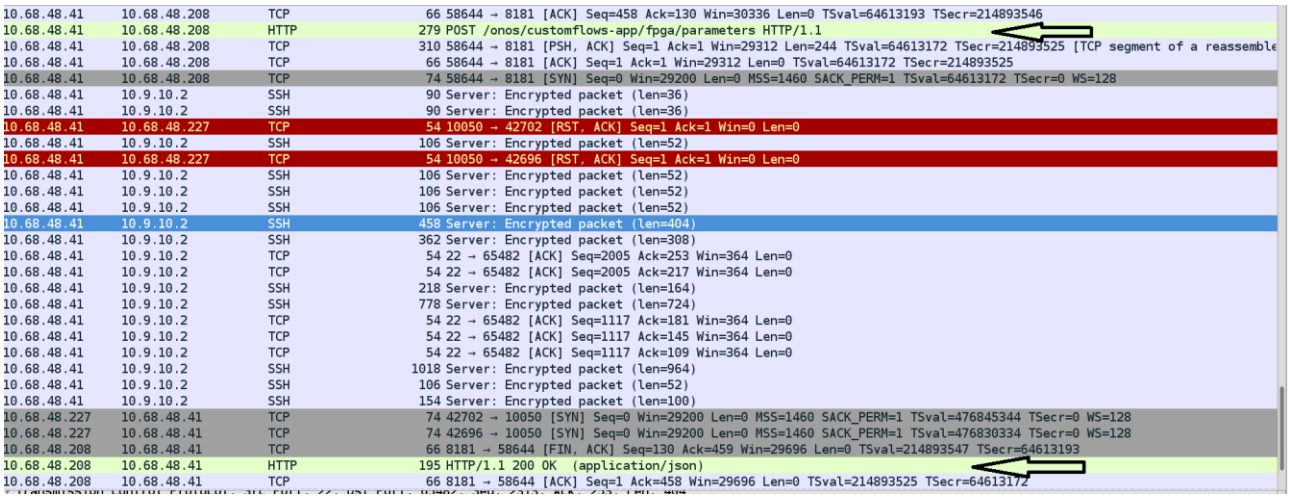

Figure 4-14: Post message exchanged between ICDM and ONOS controller.

```
{"a-end-node":"netconf:10.68.48.237:2022",
"a-end-port":10000,
"z-end-node":"netconf:10.68.48.238:2022",
"z-end-port":10000,
"vlan":20,
"transmission-mode":"TSON",
"frame-size":1250,
"time-slot-size":900
}
```

```
{"a-end-node":"netconf:10.68.48.237:2022",
"a-end-port":10000,
“a-line-port”:20000,
"z-end-node":"netconf:10.68.48.238:2022",
"z-end-port":10000,
“z-line-port”:20000,
"vlan":20,
"transmission-mode":"TSON",
"frame-size":1250,
"time-slot-size":900
}
```


Figure 4-15: Post request example (left). Post response example (right).

After fetching the two connected TSON nodes, ONOS is able to send a response to the post request which contains, the same information of the request and the line port information. The following figure shows up the response message.

4.3.1.2 The IF6 interface

IF6 defines the APIs between the 5GUKEx NSB and local MANO using the Island Proxy. Island proxy communicates with the NBI of underlying orchestrator using REST APIs for getting the list of Network Services at each site during registration. It also communicates with the SDN controller on the local site to deploy the network and to share the created endpoints with the 5GUKEx NSB. This API supports creation and deletion of end-to-end Network Services on multiple network administrative domains.

4.3.4 Evaluation of provisioning times over IF5 and IF6

In this part, we present the provisioning time to establish the end to end connection between the VM1 and the VM 1’. The table below describes the provisioning time to provide the deployment of the network service.

Table 4-3: Provisioning time for Network Service deployment.

	Local SDN setup time (s)	VNF setup time (s)	ONOS Setup time (s)
Island 1 (VM1)	1.06	22.73	0.2
Island 2 (VM1’)	1.04	22.07	

The Local SDN setup time represents the time when the 5GUKEx sends the request to each island proxy which have to be connected, to when the Island SDN controller installs the flows rules to the local switch in order to connect the Island the TSON network domain. This time has been evaluated to about 1.06 seconds. Concerning the setup time of the different VNFs, this time has been estimated approximatively to 22.73 seconds. In order to interconnect the both islands, a request is sent from the 5GUKEx to the ONOS controller which manages the TSON network domain. The time to provision and install the flow rules to the TSON nodes is estimated to 0.2 seconds.

5 Conclusions and Future Work

This deliverable concludes the work carried out in WP5 providing a practical implementation of the 5G OS concept, which is used to illustrate two main ideas:

i. The 5G OS is capable of orchestrating end-to-end services composing wireless access, compute and transport resources provided by distributed domains.

ii. The 5G OS can be used to orchestrate virtual network functions developed in WP4.

The main accomplishment of this work has been to demonstrate that the 5G OS can orchestrate a complete virtual Wi-Fi and LTE network service, including two wireless access domains in Spain and Greece, a compute domain in Germany and a transit network domain in the UK, in less than two minutes. This is a very significant step forward in automating the provisioning of the complex network services involved in the operation of mobile network architectures. This effort is in addition fully aligned with the 5G-PPP overarching KPI of "*reducing service provisioning time from 90 hours to 90 minutes*".

While constituting a significant step towards full automation of 5G network services, there are several areas within the 5G OS that require further work, including:

- Experimental validation for scalability: It should be studied experimentally that the 5G OS can scale to much larger network services, involving multiple compute domains and larger magnitudes of access devices. Such large scale validation has not been possible in this work due to the limited available resources.
- Hardware based domain interconnections: In the current multi-domain testbed inter-domain connections are implemented with a software based data-path that would limit throughput in a real setting. Some of the programmable platforms developed in WP3, such as the Open Packet Processor, could be used to implement an equivalent hardware based datapath.
- Including the TSON and TSN domains into an end-to-end multi-domain demonstration, which has not been possible in this deliverable due to the limited resources.
- Further developing the MDO concept beyond pure service provisioning, including monitoring and reliability features possibly driven by AI/ML techniques.

6 References

[1] 5G-PICTURE deliverable D5.1: “Relationships between Orchestrators, Controllers, slicing systems”, November 30th 2018.

[2] 5G-PICTURE deliverable D5.2: “Auto-adaptive hierarchies”, 31st May 2019.

[3] 5G-PICTURE deliverable D5.3. “Support for multi-version services”, 31st May 2019.

[4] Katsalis, K., Nikaein, N., & Huang, A. (2018, April). „JOX: An event-driven orchestrator for 5G network slicing”. In NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium (pp. 1-9). IEEE.

[5] Open Source MANO. Available at: https://osm.etsi.org/

[6] 5G-PICTURE deliverable D4.3: “Integration of developed functions with 5G-PICTURE orchestrator”, November 30th 2019.

[7] OpenAirInterface, avaiable at: https://www.openairinterface.org/

[8] OpenVPN. Available at: https://openvpn.net/

[9] TMForum Information Model available at https://www.tmforum.org/information-framework-sid

[10] H2020 5G-XHaul project. Available at: https://www.5g-xhaul-project.eu/

[11] Netsoft 2019 UTH

[12] ETSI GS NFV-SOL 005 (2018-02). “Network Functions Virtualisation (NFV) Release 2; Protocols and Data Models; RESTful protocols specification for the Os-Ma-nfvo Reference Point”. Available at: https://www.etsi.org/deliver/etsi_gs/NFV-SOL/001_099/005/02.04.01_60/gs_NFV-SOL005v020401p.pdf

[13] 5G-PICTURE deliverable D4.2: “Complete design and initial evaluation of developed functions”. November 30th 2018.

[14] 5G-PICTURE deliverable D3.2: “Intermediate report on Data Plane Programmability and infrastructure components”. November 30th 2018

[15] 5G-PICTURE deliverable D6.2: “Vertical Demo and Testbed setup and initial validation results”. November 30th 2018

[16] Netopeer 2.0, Available at: https://github.com/CESNET/Netopeer2

[17] Hostpad. Available at: https://w1.fi/hostapd/

[18] B. Pfaff, B. Davie, RFC 7047, “The Open vSwitch Database Management Protocol”, December 2013

[19] 5G-PPP KPIs. available at: https://5g-ppp.eu/

[20] 3GPP TR28.801, ““Telecommunication management; Study on management and orchestration of network slicing for next generation network”. Available at: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3091

[21] Dräxler S, Karl H. SPRING: Scaling, Placement, and Routing of Heterogeneous Services with Flexible Structures. In: 5th IEEE International Conference on Network Softwarization (NetSoft) 2019

[22] Razzaghi Kouchaksaraei H, Karl H. Quantitative Analysis of Dynamically Provisioned Heterogeneous Network Services. In: 15th International Conference on Network and Service Management (CNSM). Halifax, Canada

[23] N. Makris, C. Zarafetas, P. Basaras, T. Korakis, N. Nikaein, and L. Tassiulas, “Cloud-based Convergence of Heterogeneous RANs in 5G Disaggregated Architectures,” in IEEE International Conference on Communications (ICC), 2018.

7 Acronyms

Acronym	Description
5G OS	5G Operating System
COTS	Commodity off-the-shelf
DPI	Deep Packet Inspector
DSE	Dynamic Slicing Engine
e2e	end-to-end
GOP	Group of Picture
KPI	Key Performance Indicator
MANO	Management and Orchestration
MIP	Mixed-Integer Program
NFV	Network Function Virtualization
NFVO	Network Function Virtualization Orchestration
RAN	Radio Access Network
SDN	Software-Defined Networking
SLA	Service-Level Agreement
TSN	Time Sensitive Network
TSON	Time-Shared Optical Network
VIM	Virtualized Infrastructure Manager
VNF	Virtual Network Function
VNFM	VNF Manager
VNMP	Virtual Network Mapping Problem
vTC	Virtualized Transcoder