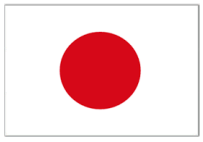

D4.3 – End-to-end Network slice

Aalto, Ericsson, Orange, EURECOM, MI, UT, KDDI, HITACHI

Document Number	D4.3
Status	Version 1.0
Work Package	WP 4
Deliverable Type	Report
Date of Delivery	January 7, 2019
Responsible	EURECOM/KDDI
Contributors	Aalto, Ericsson, Orange, EURECOM, MI, UT, KDDI, HITACHI
Dissemination level	PU

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 723172, and from the Swiss State Secretariat for Education, Research and Innovation.

Change history

Version	Date	Status	Author (Company)	Description
1.0	07/01/2019	Final	EI	First published version

AUTHORS

Full name	Affiliation
Adlen Ksentini	Eurecom Institute
Pantelis Frangoudis	Eurecom Institute
Sławomir Kukliński	Orange
Lechosław Tomaszewski	Orange
Cédric Crettaz	MI
Nicklas Beijar	Ericsson
Kashif Khan	Ericsson
Tarik Taleb	Aalto
Miloud Bagaa	Aalto
Ibrahim Afolabi	Aalto
Ilias Benkacem	Aalto
Itsuro Morita	KDDI
Yoshinori Kitatsuji	KDDI
Zaw Htike	KDDI
Akihiro Nakao	UT
Cédric Crettaz	MI
Daisuke Okabe	HITACHI
Hidenori Inouchi	HITACHI

Executive summary

Building end-to-end network slices, which span over multiple technological and administrative domains, is one of the ultimate objectives of the 5G!Pagoda project. In 5G!Pagoda, we assume that a slice is composed by a set of sub-slices, which stitched together build the end-to-end connection. A sub-slice is composed by a set of virtual or physical resources from the same technological domain, e.g. RAN, CN, transport. In this context, the role of the Slice Orchestrator (SO) is vital, as it is in charge of the resource instantiation and management for each sub-slice, as well as the stitching of the sub-slices to create an end-to-end slice. In this document, we detail the vision of 5G!Pagoda on the role of the SO via the implementation of three scenarios, corresponding to different use-cases: (i) End-to-end slice including RAN; (ii) IoT end-to-end slices; (iii) ICN/CDN end-to-end slice. The three scenarios are used to illustrate the different functions that the SO needs to implement and ensure aiming at building the end-to-end network slices; i.e. instantiation, stitching and management of resources for each sub-slice. Three different SO have been implemented and tailored to the specific needs of the use cases.

Table of contents

List of tables

List of figures

1. Introduction

1.1. Motivation and scope

In 5G!Pagoda, the Slice Orchestrator (SO) is in charge of the instantiation, management and deployment of a Network Slice over multiple administrative domains, and using resources from different technological domains. A Network Slice in 5G!Pagoda is a composition of sub-slices, which are in turn constituted of virtualized resources (CPU, storage) in case of a Core Network (CN) sub-slice, or/and physical resources in case of a RAN sub-slice. The SO has a vital role in the 5G!Pagoda architecture as it is in charge of building the end-to-end slices, using exposed resources from each technological domains, which can be provided by different administrative domains. Several orchestration and management procedures to ensure scalability have been discussed in D4.1 [1], while intra and inter-slice interaction has been discussed in D4.2 [2]. In this document, the focus is on the SO functions related to the instantiation and deployment of an end-to-end over multiple domains (i.e. technological and administrative). To this end, we used three scenarios of end-to-end network slice deployment in order to illustrate the functions of the SO; that is instantiation of sub-slices, the stitching process to create and end-to-end slices. We selected three scenarios to demonstrate 5G!Pagoda's achievement: (i) End-to-end mobile network slice that comprise RAN; (ii) IoT end-to-end slices; (iii) ICN end-to-end slice. Each scenario shows a set of SO functions that are tailored to the specific needs of the use-case. In case of the end-to-end network slice with RAN, we show the SO functions related to the creation of RAN sub-slice, EPC sub-slice via MANOaaS, and the transport slices; which stitched together build the on demand end-to-end mobile network slice. In the IoT end-to-end slice case, we show functions related to the orchestration of multi-domain IoT slices, featuring scalability functions. In the ICN/CDN case we show the functions of CDNaaS orchestrator (or SO), which instantiates two sub-slices over a multi-domains virtualized platform: a CDN sub-slice, an ICN sub-slice; stitches them to build the combined ICN/CDN network slice. For the three scenarios, we made efforts to align the Slice Orchestration architecture and framework with the ETSI NFV architecture [3], as well as the 3GPP orchestration and management model [4].

In this document, we included all the functional details, while implementation details are provided in the deliverable D5.2 [5].

1.2. Structure of the document

Following this introductory section, the remaining part of the document is structured as follows:

- Chapter 2 provides key terminologies used in the document. It includes the descriptions of abbreviations, and technical terms.
- Chapter 3 recalls the main concepts, which will be used in the following chapters
- Chapter 4 details the SO functions in case of the deployment of the mobile network end-to-end slice including RAN functions.
- Chapter 5 exposes the SO functions in case of the deployment of the IoT end-to-end slice.

- Chapter 6 shows the SO functions in case of the deployment of the ICN/CDN end-to-end slice.

2. Terminology

2.1. Abbreviations

Throughout this document, the following acronyms, listed in Table 1, are used.

Table 1 – List of acronyms

Abbreviations	Original terms
3GPP	3rd Generation Partnership Project
5G PPP	5G Infrastructure Public Private Partnership
AMF	Access and Management Function
API	Application Programming Interface
BW	BandWidth
CDN	Contents Delivery Network
CDNaaS	CDN as a Service
CN	Core Network
CP	Content Publisher
CPU	Central Processing Unit
CSMF	Communication Service and Management Function
DNS	Domain Name System
NSMF	Network Slice Management Function
NSSMF	Network Slice Subnet Management Function
E2E	End-to-End
eNB	E-UTRAN/evolved Node B
EPC	Evolved Packet Core
EPCaaS	EPC as a Service
EM	Element Manager (ETSI NFV)
eMBB	enhanced Mobile BroadBand
ETSI	European Telecommunications Standards Institute
FIB	Forward Information Base
GW	GateWay
IaaS	Infrastructure as a Service
ICN	Information Centric Networking
ICNaaS	Information Centric Networking as a Service
IoT	Internet of Things
JSON	JavaScript Object Notation
MANO	Management and Network Orchestration (ETSI NFV)
MANOaaS	Management and Network Orchestration as a Service

MEC	Mobile Edge Computing
MDSO	Multi-Domain Slice Orchestrator
MME	Mobility Management Entity
mMTC	massive Machine Type Communications
NDN	Named Data Network
NFV	Network Function Virtualization
NFVI	NFV Infrastructure (ETSI NFV)
NFVO	NFV Orchestrator (ETSI NFV)
PNF	Physical Network Function
OAI	Open Air Interface
PIT	Pending Interest Table
RAN	Radio Access Network
REST	REpresentational State Transfer
S-NSSAI	Single Network Slice Selection Assistance Information
SDN	Software Defined Network
SIID	Slice Instance IDentifier
SO	Slice Orchestrator
TOSCA	Topology and Orchestration Specification for Cloud Applications
UE	User Equipment
uRLLC	Ultra-Reliable and Low Latency Communications
VIM	Virtual Infrastructure Manager (ETSI NFV)
VM	Virtual Machine
VNF	Virtual Network Function (ETSI NFV)
VNFM	VNF Manager (ETSI NFV)
xMBB	Extreme Mobile BroadBand

3. Network slicing

To recall network slicing is the process of sharing network resources, such as computing, networking, memory and storage, which are available on a single set of physical infrastructures, among different virtual tenants with a certain degree of logical or physical separation as detailed in [6] and [7].

3.1. Slice components, types and templates

3.1.1. Slice components

In general, a network slice consists of a number of VNFs interconnected and optimally placed to fulfil a specific set of requirements and to meet specific network constraints in order to realize a particular use-case scenario. Since every network slice is designed to deliver a specific use-case scenario, it is then intuitive that the constituting set of VNFs needed to create an instance of a network slice will be somewhat different from another one (as highlighted in the next chapters). However, despite the possible differences in the design of network slices, 5G technology aims to incorporate simplicity and flexibility in network slicing by sharing and reusing as many network functions as possible in the instantiation of different network slices [8].

3.1.2. Slice types

In the context of end-to-end network slicing in 5G, a network slice will consist of components, in the form of VNFs, from the Radio Access Network (RAN), the Core Network (CN), and the transport network. Some are instantiated over a virtualized platform (hosted at the central or edge clouds), while some use physical components (e.g. eNB) that can be treated by MANO as PNFs. Even though the structure of one network slice could differ from another in terms of VNFs, the fundamental resources on which the VNFs are deployed are actually the same.

According to [9], 5G network slices are broadly classified into three major categories: Extreme/Enhanced Mobile BroadBand (xMBB/eMBB), Ultra Reliable Low Latency Communication (uRLLC) and Massive Machine Type Communications (mMTC). Each of these categories is characterized by its specific performance requirements, which are reflected through the VNF types, their connectivity, eventual redundancy and the amount of virtual resources (i.e. CPU, storage and memory) used as well as their allocation priorities.

3.1.3. Slice templates

The aforementioned categorization of network slices facilitates the definition, design and optimization of blueprints for different network slices belonging to the different categories. These variant network slice blueprints are known as ***network slice templates***. A network slice template is a functional blueprint from which a particular type of network slice could be instantiated. The blueprint defines all the necessary VNFs, their composition, VNF forwarding graph, virtualization technology type, location(s) of instantiation, level of elasticity and end-to-end network resources

needed by a network slice. In addition, the slice's lifespan (i.e. the operational duration of the slice) is determined through its blueprint by the values given for its periodic activation and termination times (e.g. in days, weeks, months or even in years), respectively.

3.2. End-to-end mobile network slice structure

At a high level, regardless of the underlying system and the functional requirements of a network slice and its VNF composition, an end-to-end network slice should always be composed of three sub-slices: RAN, CN, and Transport.

3.2.1. RAN sub-slice

Ensuring that a mobile network slice is end-to-end implies that the RAN network must be also sliceable. The sliceability of the RAN means that both its service and resources should be sliceable. While a service slice of the RAN will provide radio access service, the resource slice will be provided as virtualized Radio Resource Blocks needed to run the radio access service. A notable challenge here is how to provide the necessary level of performance isolation across slices which are sharing typically the same access network, where each slice has its own resource requirements. These requirements are basically defined by the available amount of the Physical Radio Resource Blocks (PRRBs) and the frequency of scheduling them (determining the slice's bandwidth and latency). This physical resource could either be statically or dynamically allocated at the access network for use by the RAN sub-slices.

The isolation between RAN sub-slices should be ideally carried out across the protocol layers responsible for the aforementioned scheduling and physical radio resource allocation; particularly, the PHY and MAC scheduling layers of the mobile access node [10]. The RAN sub-slicing could be enabled for instance by assigning each sub-slice with a unique ID [8]. This ID is used to identify the RAN sub-slices and is also used to enforce RAN-level differentiated traffic treatment per sub-slice.

3.2.2. CN sub-slice

This sub-slice includes the elements that correspond to the CN. These elements naturally lend themselves to a virtualized implementation, although the integration of Physical Network Functions (PNFs) in a slice instance should be also supported. NFV facilitates tailoring a slice to the needs of its respective service. Allowing the deployment of a virtualized CN over an NFV Infrastructure (NFVI) comes with the flexibility to customize the compute and other resources allocated to the slice. It also allows the slice to scale on demand for cost and performance optimization, and to select the appropriate functional configuration of the C N components (e.g. decide on the number of VNF instances for each CN function to maximize reliability). These tasks are supported by maturing relevant standards, such as the ETSI NFV Management and Orchestration (NFV-MANO) framework [3] and available implementations of MANO software stacks.

3.2.3. Transport sub-slice

After instantiating a virtual CN sub-slice from a slice template, the VNFs constituting the core sub-slice should be chained together, as well as connected with the appropriate RAN sub-slice and with the Internet or other external networks. The transport paths that are involved form the transport sub-slice, whose management is enabled by the use of technologies such as SDN (Software-Defined Networking) or Virtual LANs.

All sub-slices provide a communication service (service slice) at different technological domains. Also, they include the necessary dedicated resources (resource slice) needed to operate the technology-specific service. For instance, at the RAN level, the service is radio access and the resources are virtual resource blocks. The core sub-slice provides EPCaaS, which is supported by specific dedicated virtual storage and compute resources. The transport sub-slice provides a connectivity service to external networks, with dedicated virtual network links.

3.2.4. End-to-end slice orchestration and management

As described in D4.1 and D2.3, an end-to-end slice needs to be orchestrated and managed. The 3GPP has identified the four following phases which are related to the lifecycle management: Preparation, Instantiation, Configuration and Activation; Run-time; Decommissioning. In addition, the 3GPP has identified 3 management functions [4] related to network slicing management:

- Communication Service Management Function (CSMF): this function is responsible for translating the communication service related requirement to network slice related requirements.
- Network Slice Management Function (NSMF): this function is responsible for the management (including lifecycle) of NSIs. It derives network slice subnet related requirements from the network slice related requirements. NSMF communicates with the NSSMF and the CSMF.
- Network Slice Subnet Management Function (NSSMF). This function is responsible for the management (including lifecycle) of NSSIs. The NSSMF communicates with the NSMF.

In [11] a possible mapping with the ETSI NFV model is detailed and shown in the following figure:

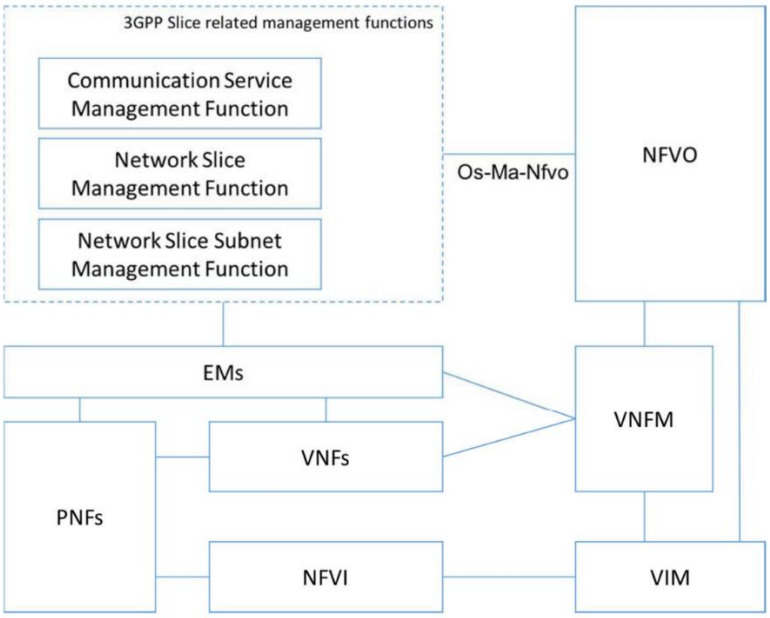

Figure 1. Relation between 3GPP management components and the NFV components [11].

The 3GPP approach requires no modification of the ETSI MANO framework, the CSMF, NSMF and NSSMF functions are deployed within OSS/BSS part of the MANO framework. The OSS/BSS internal functionalities are not defined by ETSI MANO.

4. End-to-end orchestration of mobile network slices including RAN slices

In this section, we focus on the end-to-end orchestration of mobile network slices, including RAN slices. This use case requires functions related to the instantiation of three sub-slices (RAN, CN and transport), which are stitched using 3GPP LTE interfaces. While the CN components are based on LTE, the presented and exposed features can be easily extended to the 5G CN; particularly as the latter is virtualization ready via the concept of service-based interfaces.

4.1. High-level design

In compliance to the 3GPP specifications of [4], our high-level network slicing framework is shown in Figure 2. The Communication Service Management Function (CSMF) component, which is outside the scope of this work, receives a request for a communication service by a vertical and translates it into specific slice requirements. Then, it interacts with the **End-to-End (E2E) Slicer** via the latter's northbound interface (NBI) to request the instantiation of an end-to-end network slice with specific characteristics. The features, design, and implementation of the E2E Slicer are the main focus of this section. For more details on CSMF functions, the interested reader may refer to [4].

The lifecycle of a network slice is handled by the E2E Slicer's Network Slice Management Function (NSMF). In particular, based on the requested slice type and specific dimensioning information and requirements (e.g. slice reliability targets, number of UEs and traffic characteristics to handle), it retrieves a suitable slice template from a catalogue and selects the appropriate Network Slice Subnet Management Functions (NSSMF) that correspond to each sub-slice, as indicated in the template. The NSMF then proceeds by composing a customized slice instance by delegating the instantiation and management of each sub-slice to the appropriate NSSMF. Note that our framework supports the MANO-as-a-Service (MANOaaS) concept by design. Effectively, each slice template includes fields that indicate what type of NSSMF should be used to orchestrate the underlying sub-slice instance. This is particularly important for handling CN sub-slices. In this case, the NSMF will identify which NFV Orchestrator (NFVO) and the respective MANO stack to launch or reuse one slice instantiation type. Then, it will access the NFVO's NBI to request the creation of the appropriate CN sub-slice instance, appropriately customizing the included VNF instances (e.g. VMs implementing the Mobility Management Entity (MME) functionality) and the respective resources according to the specific service characteristics.

For a RAN sub-slice, the respective NSSMF includes a RAN resource allocator component, whose role is to translate slice requirements into radio resource allocations and carry out high-level RAN resource management. For this purpose, it needs to maintain the RAN's state in terms of the connected UEs per eNB and the quality of their radio connection, per UE/slice bitrate requirements, and the slice instances to which an eNB participates. The above information is used by the NSSMF to derive an appropriate RAN resource partition per cell in order to be able to satisfy the requirements of the coexisting slices (e.g. latency). This resource partitioning can be adjusted dynamically following updates in the RAN state, including changes in the radio conditions or the

deployment of new slices, and is enforced by the NSSMF using the appropriate NBI of the RAN controller.

In this context, the RAN controller can be seen as a RAN-specific Virtual Infrastructure Manager (VIM), while an agent installed at an eNB has an operation comparable to that of a hypervisor: It exposes a virtualized view of RAN resources and offers the necessary primitives to execute resource management tasks across slice-dedicated physical resources on the same radio hardware.

Finally, regarding the transport-level NSSMF, its role is to interact with network elements such as SDN controllers to manage the provisioning and isolation of the links connecting (virtual or physical) network functions of the access and core networks, and towards external networks.

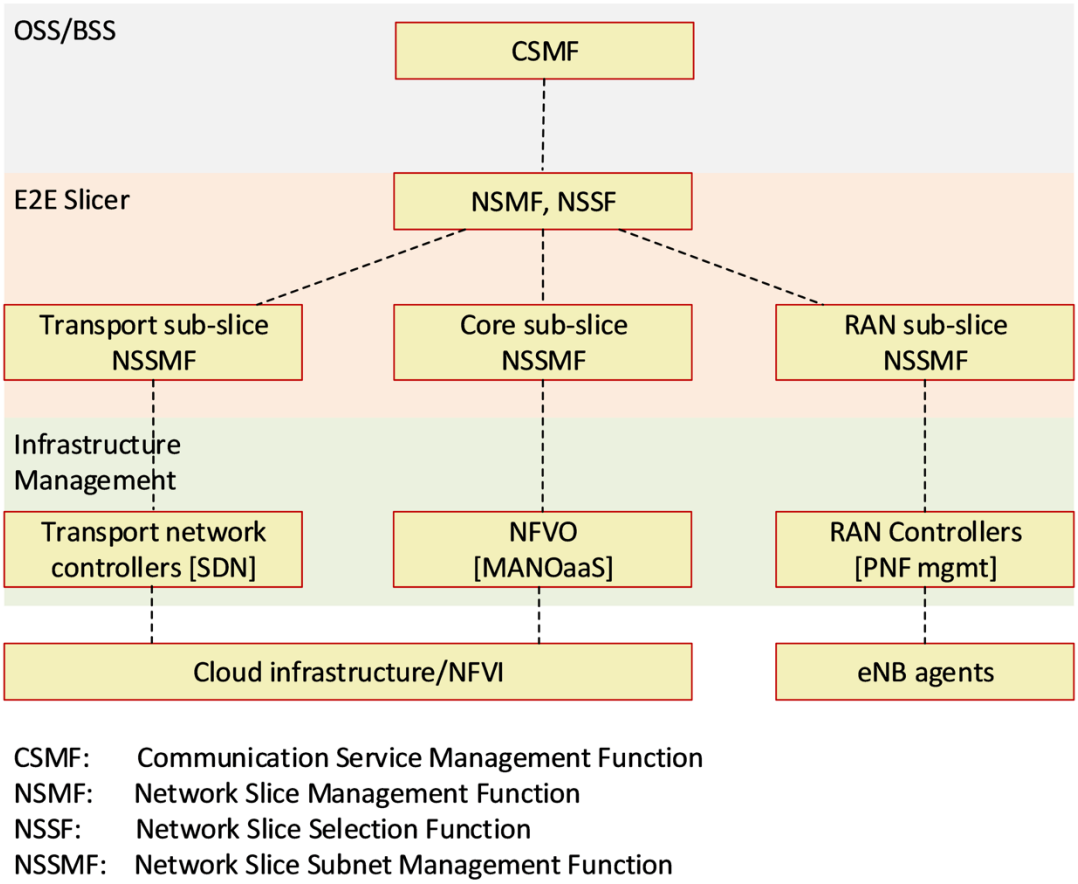

Figure 2. High-level view of our end-to-end network slicing architecture. Our design features an E2E Slicer which implements NSMF and NSSF components. It supports the integration of PNFs for RAN slicing and different MANO stacks per slice type and interacts with them via different NSSMF components per sub-slice.

A design with a similar target but distinct differences to our approach is JOX [12], a Juju-based slicing-oriented orchestration scheme. Although the design of JOX does not preclude an NSMF operation, it is more oriented towards providing NSSMF functionality and does not address slice selection issues. As such, it can be used in a complementary fashion to our architecture. In such a case, JOX could be used to implement different NSSMF plugins, e.g. abstracting the operation of the RAN controller by offering a RAN-specific NSSMF accessible via its NBI. Similar to this is the Open Source MANO (OSM), which is a general orchestration framework that offers lots of features

but with no native support for orchestrating and managing PNF from which the RAN sub-slice are orchestrated at the eNB. Another similar project is M-CORD [13] which has made significant contributions to E2E network slicing. At the architectural level, M-CORD includes slice definition components and allows stitching RAN and CN sub-slices, as in our case. Given these similarities at the architectural and functional level, it should be noted that our focus is more on ensuring the compliance of our design with the 3GPP specifications on network slicing, and on describing our own technical solutions for RAN slicing, slice-dedicated CN support, and network slice selection and orchestration, in a more light-weight design and implementation than M-CORD.

4.2. Use cases

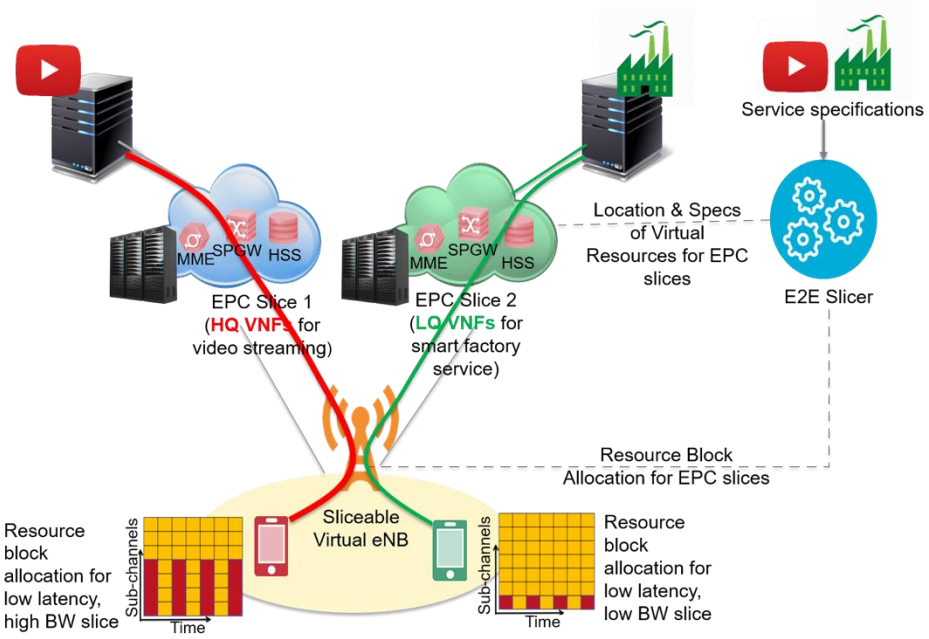

Figure 3. 5G vertical use cases over the envisioned network slicing architecture.

Different vertical industries have different slicing requirements, as depicted in Figure 3. In particular, we focus on the media and entertainment and the smart industry use cases and show how our E2E Slicer customizes and orchestrates two end-to-end network slices based on both system requirements as well as customer preferences. For example, in the case of the smart factory slice, shown in green, the service is deemed as a "low latency, low bandwidth" service. Effectively, information transmission of smart devices running in a smart factory would require very short latency as well as moderately small bandwidth at the access network. The access network resources are accordingly allocated in the form of one RRB (i.e. due to the "low bandwidth" nature of the smart factory service) to the slice and that is at certain predefined times corresponding to the "low latency" nature of the smart factory service. Correspondingly, the VNFs that would be instantiated at the CN would be more lightweight (i.e. also of low quality) since the traffic to be processed at the control and data planes are not resource demanding.

Conversely, in the case of the slice dedicated to a video streaming service shown in red, the service is deemed as a "low latency, high bandwidth" service. Consequently, the corresponding amount of network resources allocated at both the core and access parts of the network is higher.

Many more resource blocks are frequently allocated at the RAN compared to the smart factory use case. Similarly, the VNFs instantiated for the CN sub-slice are of high quality, able to process video packets and are running on more virtual resources.

4.3. Distinctive features

In this section, we present some of our distinctive design and implementation choices for the E2E Slicer. Our detailed architecture, following the high-level design principles of Section 4.1, is shown in Figure 2.

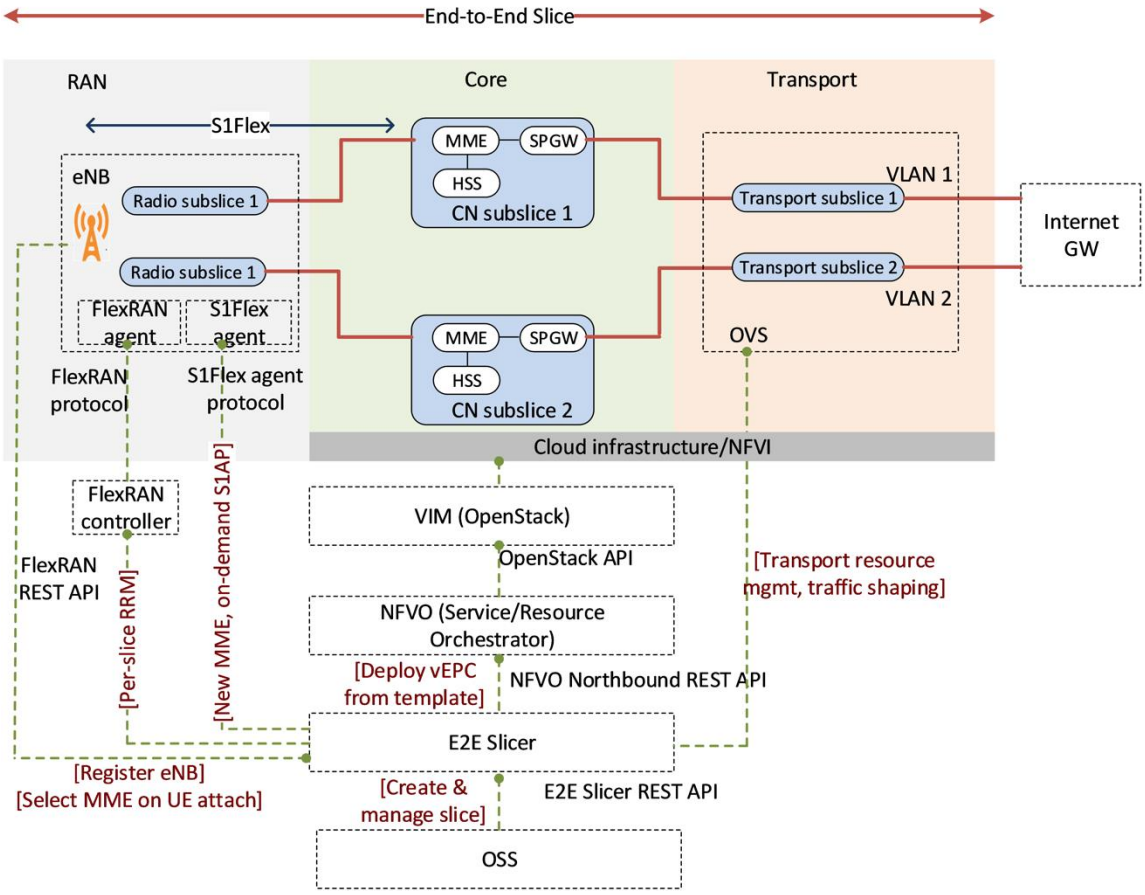

Figure 4. Detailed architecture of the proposed framework.

It shall be noted that our orchestration system supports instantiating and orchestrating network slices on multiple clouds/NFVOs. This functionality is provided by specific per-cloud NSSMFs. However, the scope of this work does not encompass the details of the individual functionalities and interfaces required to interact with each of the API endpoints provided by the different existing cloud administrative domain's NSSMF.

4.3.1. RAN slicing

To enable RAN slicing, we have adopted the FlexRAN approach [14]: Each eNB runs an agent which is managed by a centralized controller using a southbound protocol. The agent exposes RAN-level status information and, importantly, makes it possible to apply resource sharing policies at the

MAC layer. It is left to the RAN sub-slice NSSMF to derive the appropriate resource allocation ratios across all coexisting slices sharing an eNB, and the FlexRAN protocol allows the NSSMF to enforce them using the appropriate API call.

Various algorithms [15] for deciding how RAN resources should be shared across slices are possible; they combine awareness of service-specific slice requirements (e.g. a strict latency constraint for a uRLLC slice or a high-throughput demand for xMBB) and the current RAN conditions. The former are provided to the E2E Slicer by the slice owner at slice instantiation time via its northbound REST interface, and the latter are retrieved periodically or on-demand by eNBs using FlexRAN.

4.3.2. Slice-dedicated CN support

A critical feature in order to support end-to-end slicing is the ability of an eNB to maintain associations with multiple CNs simultaneously, a feature typically termed as **S1-flex**. This is important since a shared eNB instance may be hosting multiple RAN sub-slices, which in turn are stitched with separate core sub-slices potentially running over different NFVIs. In our case, we have implemented a special S1-flex agent in OpenAirInterface (OAI), our LTE software of reference that handles new eNB-MME associations (carried out using the S1AP protocol) when instructed so by the E2E Slicer. Notably, this feature not only enables the communication of an eNB with different slice-dedicated CNs, but also serves for load balancing purposes: a single slice may necessitate the instantiation of multiple replicas of its CN components (e.g. MME) to share the UE load; the E2E Slicer will have to use the same procedure to notify the involved eNBs about this.

4.3.3. Network Slice Selection Function

Beyond the E2E Slicer's task of managing the association between RAN and CN sub-slices, it also has a central role in network slice selection. In traditional 4G networks, upon UE attachment, the eNB selects the MME responsible for handling that UE based on the specified PLMN (Public Land Mobile Network) identity included in the attach request. In our case, this decision is delegated to the slicer itself. In particular, each time a UE connects to an eNB, the latter accesses a special REST API endpoint of the E2E Slicer to query for the appropriate core sub-slice's MME to direct the UE to. This offers flexibility, since it facilitates the development of sophisticated per-slice CN selection policies implemented as E2E Slicer plugins, S1AP traffic load balancing algorithms, etc. However, as the required flexibility comes with centralizing slice selection decisions, it can lead to increased attachment latencies due to the additional eNB-slicer communication, but also potential overloads at the slicer level. In section 4.4, we discuss this flexibility-performance trade-off.

4.3.4. Slice instantiation and management

The E2E Slicer provides a RESTful HTTP interface to the CSMF for slice instantiation and runtime management. This interface allows, among others, to select the type of slice from the ones available in the slice catalogue and continue with the composition of the slice as described in Section 4.1.

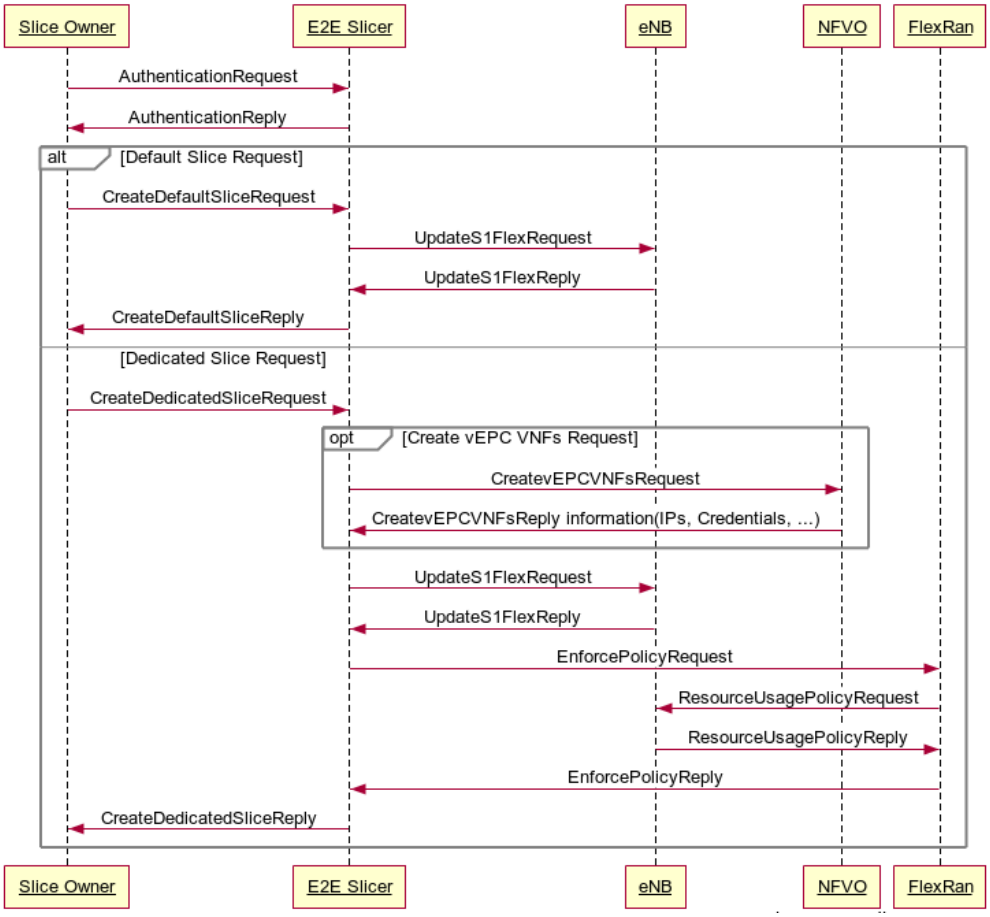

Figure 5. Sequence diagram for the slice instantiation procedure in our implementation.

Figure 5 depicts the sequence of actions and the interactions between the components of our architecture that take place from the moment slice instantiation, which is requested until the slice is fully operational end-to-end. Note that we make the distinction between **default** and **dedicated** slices. In our design, the deployment of a default slice is necessary as a fall-back for the NSSF of the E2E Slicer when a UE is not associated with any specific slice. In a typical scenario, the CN sub-slice's NSSMF corresponding to the default slice may be simpler, in the sense that the default slice may involve an already running legacy CN, thus not requiring its dynamic deployment over a NFVI and its orchestration using a MANO stack. The discussion in this section focuses on the case for a **dedicated** slice, which in turn involves the dynamic instantiation of a slice-dedicated CN.

Note that the E2E slicer is aware of which eNBs correspond to each service area, so that it can appropriately configure the deployed RAN sub-slices. Based on the slice type as specified in the slice template, the E2E Slicer's NSMF selects the NSSMF which is responsible for the instantiation of the CN over the respective cloud/NFVI platform. The NSSMF then takes over the communication with the respective NFVO in order to launch and configure a virtual CN instance (virtual Evolved Packet Core, vEPC network). Different CN deployment scenarios are possible here, depending on the selected slice template and the related service requirements. This information dictates the exact vEPC characteristics (e.g. software images, number of VNF instances, and allocated resources), and is passed on to the NFVO, which then instantiates all the needed VNFs for the creation of the vEPC by communicating with the underlying VIM. Upon completion of the request, the NFVO notifies the

E2E Slicer of the vEPC service information, such as the IP address of the MME VNF, which is used by the E2E Slicer to setup an S1AP association between the serving eNBs and the instantiated vEPC through the S1-Flex interface.

As soon as the S1AP association is established between the eNB and vEPC, the E2E Slicer employs the RAN sub-slice's NSSMF to determine an updated resource sharing policy to apply to the eNBs taking part in the slice, which will be taking into account the radio resource requirements of all coexisting slices per eNB. In our implementation, as shown in Figure 5, after computing the new RAN resource sharing, the slicer's NSSMF will communicate with the FlexRAN controller over its NBI, and the controller will in turn enforce it by communicating with each involved eNB's FlexRAN agent. After this procedure is completed, the slice instance is ready for utilization.

During the slice's lifecycle, which spans until a predefined time specified in the slice creation request or until its explicit termination by the slice owner over the E2E Slicer's NBI, the E2E Slicer collects runtime monitoring information to perform slice lifecycle management operations [16]. Retrieving monitoring data and performing such management tasks per sub-slice is delegated to the respective NSSMFs. Beyond enhancing RAN and CN performance via appropriate resource scaling and reconfiguration actions, such runtime information can also help the E2E Slicer operator to better understand the overall short- and long-term system behaviour and optimize its orchestration. Tools from artificial intelligence and data analytics could be used to this end, which we consider a critical topic for future research.

4.4. Evaluation

The results derived from the evaluation of our end-to-end slice orchestration platform are presented in this section. Our aim is to quantify the price to pay for the flexibility offered by our slicing design in terms of latency overhead and provide guidelines on how the system operator could use our performance results to customize the operation of the deployed slices. We have carried out our evaluation from two major perspectives: (i) a system-centric perspective, which pertains to the overheads associated with slice instantiation and (ii) a user-centric one, which aims to quantify the overhead imposed for end-users during the slice operation.

4.4.1. System-centric evaluation: slice instantiation overhead

There are quite a number of overheads that could be evaluated from instantiating a network slice, such as ensuring certain E2E latency, bandwidth, extremely low jitter or error rate probability. In this paper, our focus is on the system-imposed latency cost. From a slice management viewpoint, we are interested in measuring the delay from the moment slice instantiation is requested over the E2E Slicer's NBI until the slice is fully operational, i.e. the overall slice deployment time. This system-imposed latency is usually incurred once, and that is always during the process of deploying a network slice. As depicted in Figure 5, this overhead can be broken down to the following components: (i) the time taken for the E2E Slicer to instantiate the slice-dedicated CN VNFs on the IaaS platform; (ii) the time to communicate successfully with the eNB to notify it of the new vEPC instance and instruct it to create a new association with it, and (iii) the delay to configure the RAN sub-slice for the participating eNBs, which involves accessing the NBI of the RAN controller to

update the radio resources to be shared at the RAN. The overall latency overhead is dominated by (i) and is a function of the image size for the VNFs to be deployed, some IaaS configuration options, and, importantly, the Network Core sub-slice specific reliability and other requirements. Our testbed measurements revealed that while the E2E Slicer takes a few tens of milliseconds to communicate with both the eNB and the RAN controller (i.e. 36 ms and 27 ms, respectively), deploying CN VNF images in the underlying IaaS platform, which in our case is OpenStack Ocata, takes orders of magnitude more time. To demonstrate this, we have carried out the instantiation of VNF images of 10 different sizes, ranging from less than 1 GB to up to about 19 GB as presented in **Figure 6**. We note that drastic performance improvements can be achieved if VNF images are already cached at the compute nodes that will host the VNFs or if LXD containers are used with the Z File System (ZFS), since the time to transfer them from the image store is eliminated. It is thus to the advantage of both the E2E Slicer and the IaaS operator (if these two entities do not coincide) to proactively cache images on all the deployed compute nodes at least for VNFs that are expected to be frequently used. This will help speed up the rate at which new VNFs will be instantiated from such images.

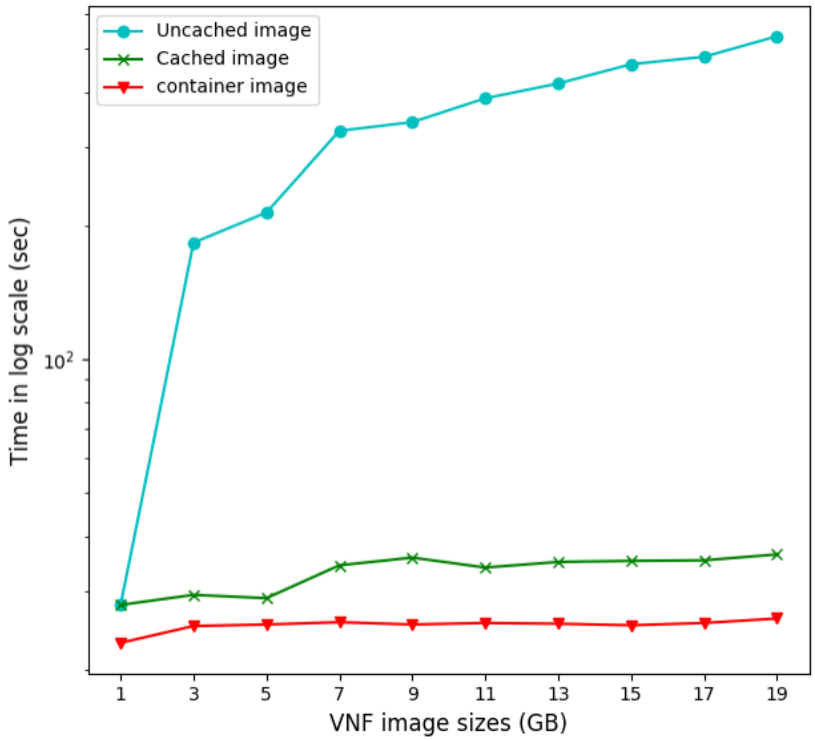

Figure 6. VNF component instantiation times as a function of the VM and Container images size. Caching VM images at compute nodes drastically reduces the time to launch an instance.

This discussion also implies that the time to instantiate a slice and the characteristics and requirements of the latter are correlated and a trade-off for the E2E Slicer operator to address is revealed. We elaborate this by an example. Consider a vertical which requests the instantiation of a highly-available network service. In this case, and assuming a CN sub-slice which includes 4G EPC elements, the E2E Slicer will decide to launch multiple replicas of the MME VNF and appropriately balance UE association requests among them. How many virtual instances of a specific function it will launch is a matter of balancing among slice instantiation delay, available cloud resources, and

the desired slice reliability levels (i.e. more VNF replicas means more time to launch them, potentially higher cost, but also increased slice robustness and availability). Measurement studies, such as the one we present here, could assist the Slicer operator with such decisions.

4.4.2. User-centric evaluation: slicer scalability and UE association delays

In our design, the E2E Slicer has a critical role in the Network Slice Selection Function (NSSF). The level of centralization that we impose in the decision on which CN instance a UE is attached to, can increase the UE connection latency in the following two ways, which we quantify in this section:

- An additional communication exchange needs to take place anyway between the eNB and the E2E Slicer using the latter's REST interface upon UE attachment. It is the E2E Slicer that executes the core sub-slice selection function and drives the UE towards the appropriate CN instance.
- Since compared to the default 4G CN selection decision, which is carried out in a distributed fashion by eNBs, in our design the E2E Slicer centrally assumes this role. It is therefore susceptible to increased latency when the request load at its end increases.

In this section, we aim to measure the effect of the above characteristics on the 4G, UE Attachment procedure execution time. First, we report on the minimum overhead that we impose in this procedure. Compared to the default 4G Attachment procedure, the additional message exchange between the eNB and the E2E Slicer causes approximated additional 36 ms latency under non-loaded E2E Slicer conditions.

This additional time is quite negligible and can be accommodated within the maximum allowable time (T3410 = 15 s), which is a timer normally started by the UE at the point of attachment to the network as specified in 3GPP TS 24.301 [17].

However, as the attach request workload increases, this latency is expected to also increase. We therefore carried out the following experiment. We launched the E2E Slicer as a virtual instance on an OpenStack cloud, to which we assigned one virtual CPU. Then, we generated parallel HTTP requests towards the E2E Slicer's UE attachment API endpoint and measured its response times. As **Figure 7** shows, after a specific request workload, the E2E Slicer's response times steeply increase. For example, when it serves a workload of more than 125 requests/s, the response time goes significantly beyond 1 s.

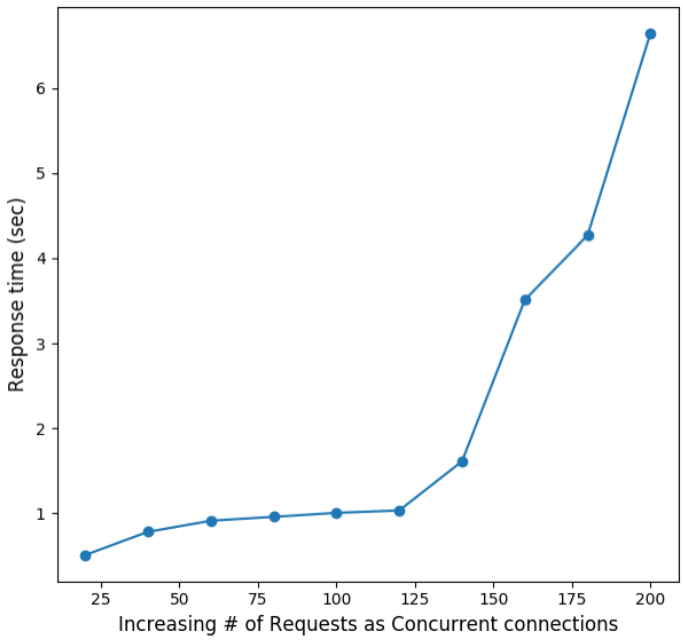

Figure 7. Response time as a function of concurrent connections for increasing request workloads.

Such results are of interest for the E2E Slicer operator. The slice consumer might pose specific average UE attachment time constraints encoded in the form of an SLA. The operator then has to appropriately manage the CPU and network resources allocated to the E2E Slicer to guarantee this performance objective. In our case, if for instance the slice consumer sets a 1.5 s upper bound to the UE attachment time, the operator needs to monitor the request workload on the NSSF component and, if an increasing load is detected, the operator shall scale in/out the allocated resources to maintain low response times.

5. IoT end-to-end network slices orchestration

In this section, we detail the orchestration functions needed to orchestrate IoT end-to-end network slices over multiple domains. While the orchestration platform has been designed for generic slices, it has been optimized for the IoT use case. In this use-case, we mainly focus on the scalability issue raised when deploying IoT slices, which may involve a high number of IoT devices.

5.1. Problem statement

The Internet of Things (IoT) is a large galaxy encompassing a lot of different devices, like sensors and actuators, which need specific functionalities provided by the network layer. Additionally, each use case directly linked to the IoT requires some unique features like real time data transmissions, reliability, high availability on the network bandwidth, etc. In the context of 5G!Pagoda project, the creation of different network slices dedicated to specific IoT requirements is a good solution for the different domains associated to the IoT. Of course, a Multi-Domain Slice Orchestrator (MDSO) is needed to manage all the slices from their creation to their destruction.

5.2. The 5G!Pagoda approach

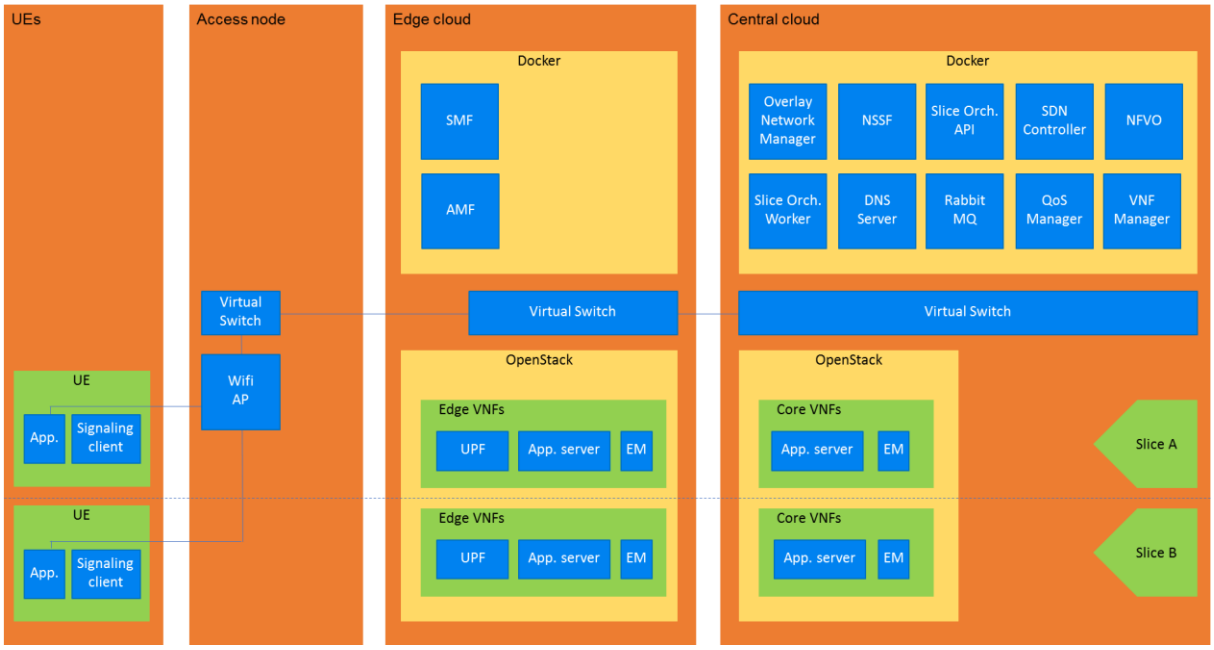

Figure 8. Multi-Domain (Administrative) Slice Orchestration Framework.

5.2.1. Multi-Domain Slice Orchestrator for IoT

The target of the MDSO is to orchestrate IoT slices across multiple domains. This includes both technological domains including cloud and transport, and regional domains, including edge and core domains. Moreover, its purpose can be extended by orchestration between administrative domains, using resources residing in e.g. another provider's network. To test the orchestration

concept, Ericsson has made a prototype implementation of the Multi-Domain Slice Orchestrator called MuDoSO. It has in particular been deployed to the IoT use case.

5.2.1.1 Concepts

An *orchestration domain* is, as defined by the MDSO, a part of the network based on similar technology and with similar configuration. A domain may be an edge or core domain or a third-part service provider. The domain configuration defines the elements involved in a slice operation and the sequence of calls to these. For example, a deployment operation may in a particular domain include a call to a NFVO, SDN controller, Overlay Network Manager, DNS server. Each of these elements has a set of endpoints, configuration settings, etc. Each domain may be configured in a different way.

A *slice* is identified to the UE using an S-NSSAI [18]. The orchestrator allows several *slice instances* to be set up based on a single slice definition. These instances are based on a common slice definition, but can have minor differences, such as deployment location, capacity or version. This simplifies version control and enables edge computing. It also increases reliability and security by subdividing a slice into several independent instances. Each slice instance has a unique Slice Instance ID (SIID). The SIID is invisible to the UE. Thus, a UE requests to connect to a slice as identified by its S-NSSAI and is assigned to a slice instance of the requested slice. The assignment is performed by the NSSF. Using extensions, described later, instances can be deployed on-demand when a UE requests access to them.

5.2.1.2 Micro-service structure

The design of the MDSO has followed a micro-service approach, particularly paying attention to scalability, which is an important point in an IoT deployment. The multi-domain IoT slice orchestration service is divided into several types of logical functions implemented as micro-services. Each micro-service can be running in a separate container (e.g. Docker).

- *API*: This type of micro-service is the interface toward other services. The API is a HTTP REST API with endpoints for managing slices, slice instances, slice descriptors and domains. The API allows the user e.g. to create a new slice definition and based on this definition to create instances of the slice for a particular purpose or domain linked to IoT.
- *Worker node*. The worker node takes care of the actual work of slice operations, e.g. slice instance deployment. This is accomplished by calling a series of plugin modules depending on each domain configuration. The plugin modules may communicate with external services using specific interface modules. The worker structure is described in the following subsection.
- *Authentication*: This micro-service authenticates clients for accessing the API. Successful authentication provides a JSON Web Token (JWT) including authorization information.
- *Database*: The database is a regular SQL database (PostgreSQL used in our prototype) accessed by the API and worker nodes for storing and retrieving persistent information, including the slice definitions and running slice instances status.

- *Message bus*: The API communicates with Worker nodes using a message bus mechanism (RabbitMQ used in our prototype). The message bus distributes the operation to the following available worker node and replies back the intermediate and final responses of the operations to the API node initiating the operation.

Each of these nodes can scale individually. For example, more worker nodes can be added for an increased work load of slice operations, for instance when the IoT traffic is increasing punctually. Several API nodes can be added for availability and increased capacity handling if the number of operation increases.

In addition to the micro-services, the orchestrator software consists of a frontend which is served by a frontend web. The frontend is implemented in JavaScript with Angular and runs in the web browser of the user. The frontend communicates with the API nodes using asynchronous REST calls.

The platform can be set up, managed and scaled by Kubernetes.

5.2.1.3 Plugin module structure

The worker node carries out the work of slice operations. Slice operations include

- Creation of a slice definition.
- Removal of a slice definition.
- Deployment of a slice instance based on a slice definition.
- Removal of a slice instance.

The worker node consists of a coordinating module and a set of plugin modules. Plugin modules can be dynamically added and the choice of plugin modules depends on the definition of the orchestration domain. Thus, each domain may require different orchestration operations managed by different plugin modules. The coordinating module calls the plugin modules in the defined sequence with parameters specified in the domain configuration as well as provided by the slice definition. Each plugin module performs their respective part of the slice operation and returns the result and their state back to the coordinator module. The plugin modules may call external services, such as calling a NFVO or an SDN controller. The results returned to the coordinating module can include state to be stored with the slice instance data. For example, deploying a slice instance creates state about the identifiers about various components (VMs, virtual networks, etc.) belonging to a slice. This state will later be needed when the slice is removed in order to remove all components that the slice consists of. The coordinating module stores the slice and slice instance state in the database and retrieves the previous state from the database when the following operation is performed on the same slice or slice instance.

The current prototype includes the following plugin modules, which are also described in detail in later sections:

- NFVO modules: This module deploys a slice within a domain using a NFVO. This module takes care of uploading TOSCA slice definitions, creating the service configuration and the VNF definitions as well as controlling the deployment of VNFs to VMs. Our prototype includes a NFVO module for OpenBaton.

- SDN controller modules: This module creates networks via an SDN controller. Our prototype has an SDN-based access network and therefore we provide an SDN controller module for our custom implemented access network controller.
- Overlay network module: This module creates networks with an Overlay Network controller. Overlay networks are used for interconnecting networks in different domains, where direct SDN- or VIM-based networks are not applicable. Our prototype includes an overlay network module for the Overlay Network Manager responsible for interconnecting VNFs across domains.
- DNS module: This module registers addresses exposed by VNFs in DNS allowing other VNFs as well as end users to locate the services provided by the VNF. Our prototype includes a DNS module that uses dynamic DNS registrations toward a BIND server.
- REST notification module: This module is a general purpose module allowing external services to subscribe to notifications when slices are deployed and removed. Extending the platform is therefore easy without implementing new plugin modules for simple notification needs. This allows core elements, such as NSSF and AMF, to receive up-to-date information about available slices. In our prototype, AMF currently uses this interface to subscribe to information about new or removed slices.

Since the operations consist of multiple phases, it is important to handle error conditions correctly. The worker node needs to take care of rolling back operations if one of the phases fails. The whole operation must be seen as a single atomic operation and there cannot be any state remaining from an incomplete operation.

5.2.1.4 Example

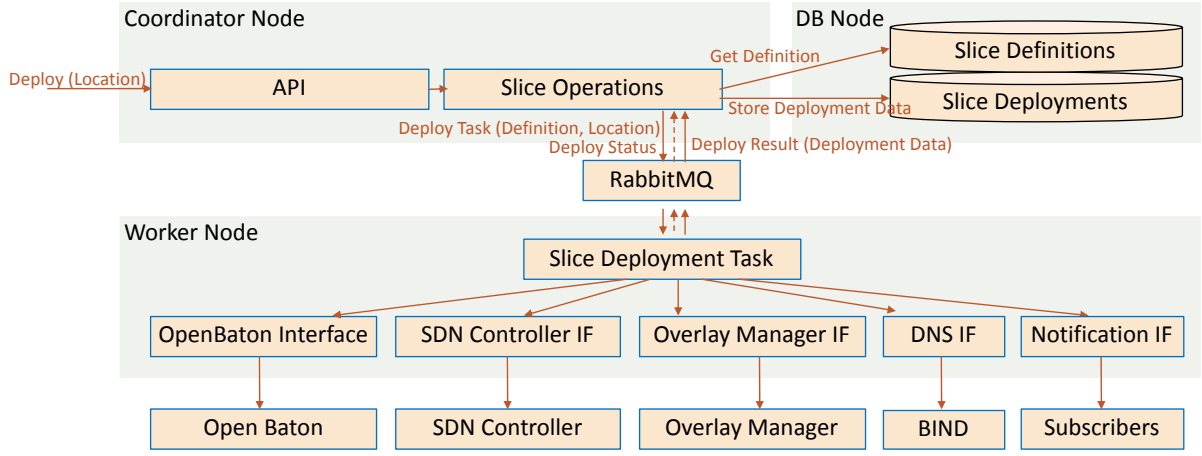

Figure 9. Slice Orchestrator internal structure.

Figure 9 presents an example of a slice instance deployment operation. The deployment request is received by the API node through the slice operations endpoint. An optional parameter indicates the location (TAI) target of deployment when the slice is deployed at or extended to an edge domain. The definition of the specified slice is retrieved from the database. The request is sent via the message bus and the first available worker node picks up the request. Based on the domain definition, the sequence of plugin modules as well as their parameters are identified. Each module

is called to perform a particular phase of the slice instance setup. On completion, the modules return state and status information, and the collected state from all modules is aggregated and stored in the database as slice deployment state. The worker node also sends back intermediate state to the API node on which phase is active, which can be used to present the user with feedback.

5.2.1.5 Placement algorithm

The orchestrator uses a concept of *placement focus* for controlling where VNFs of a slice instance will be placed. The placement focus is typically at an edge node for a normal slice. Within slices VNFs are marked with *placement priorities*, specifying how important it is for the VNF to be located near the placement focus, typically the network edge. Many types of slices will have VNFs with high placement priority that should, if possible, be placed at the edge node and VNFs with lower placement priority that do not need to be placed near the edge. A placement priority may also be negative, expressing that the VNFs must not be placed at the edge.

When a slice is deployed (by a management interface, an external business orchestrator as defined in D2.1 or on-demand by 5GC AMF), the placement focus is given as a parameter. The placement focus can be given as a TAI or similar parameter. The placement focus is given as an input to the placement algorithm, which has been specified in Deliverable D3.1 (section 6.4.4).

The MDSO primarily selects the placement through domain selection. The domain location covers both the areas within a single network as well as remote networks. Domains are identified with the locations they cover, including the central location. Domain selection aims to select a domain that can cover a slice instance end-to-end. If such a domain is not available, several domains are selected. Within the domain, the placement is done on a node level.

5.2.2. Overlay Network Manager

The orchestration framework provides slice isolation based on an overlay networking approach. The overlay networking approach provides tunnelling between multiple cloud domains. The Overlay Network Manager performs a centralized management of the overlay network. Overlay Network Manager provides connectivity through tunnels between access network switches and the UPF. The overall concept of the overlay network manager is novel in this orchestration framework. It is designed to allow VNFs to be placed on different cloud domains without worrying about the low-level networking configurations.

5.2.3. Access network SDN controller

SDN controllers allow applications to make decisions on network optimizations without having any a priori knowledge of the underlying physical networking architecture. In this orchestration framework SDN controller is deployed to manage the connectivity from UEs to UPFs by deploying forwarding rules to access network switches. The rules are deployed using OpenFlow. The SDN controller interacts with Overlay Network Manager and AMF to obtain the information on the deployed slice instances and about the UEs.

5.2.4. NFVO

The NFV Orchestrator (NFVO) is a key component in this orchestration framework as it performs the resource orchestration and network service orchestration as defined by ETSI NFV. NFVO can work with either VIM or NFV Infrastructure (NFVI) directly to orchestrate compute, storage, and network resources for networking services. In our orchestration framework, we have used Open Baton as the NFVO and VIMs to set up slices containing of multiple NFVs. In addition to the NFVO component, we also use the Generic VNFM and the EM of the Open Baton project. The NFVO controls the orchestration and maintains the lifecycle of NFVs as well as deploying slices on one or several VIMs. It also defines the networks internal to a VIM. In this orchestration the NFVO is deployed in a Docker container.

5.2.5. VIM

The Virtualized Infrastructure Manager (VIM) is responsible for managing the NFVI compute, storage and networking resources usually within an operator's domain as defined by ETSI NFV. In our orchestration framework we have used OpenStack as the VIM. The proposed concept uses no code modification to OpenStack but uses OpenStack as NFVI only with configuration. This gives the flexibility to the orchestration framework to be deployable to any third-party VIMs. The VIM keeps inventory of the allocation of virtual resources to physical resources, manages the internal network between VMs and also manages a repository of NFVI hardware as well as software resources. The core role of VIM is to coordinate physical resources necessary to deliver network services.

5.2.6. Interaction with 5G control plane

5.2.6.1 Slice operation clients

The slice orchestrator provides an API towards slice user for dynamically creating a slice.

5.2.6.2 AMF

AMF provides UE-based authentication and mobility management functionalities irrespective of the different access technologies associated with UE. In simple terms AMF provides the control plane interface toward the UE. This orchestration framework includes a rather basic AMF with the functions only related to network slicing. In the CN, the AMF interfaces with the NSSF for slice selection. It also interfaces with the SDN controller to establish the slice connectivity of UEs.

5.2.6.3 UPF

The User Plane Functions (UPF) are VNFs within a slice that interfaces the UEs. It is the first VNF receiving user plane traffic from the UE. Our orchestration framework contains a virtual switch (Open vSwitch) that terminates tunnels toward switches in the access network. The UPF receives the user plane traffic on a particular network interface, and supports different packet routing and forwarding towards slice services. In our prototype, the UPF implements network address translation between the UE addresses and the service address space. It contains Slice Agent software that enforces bandwidth allocations, manages the slice start-up, and registers to the Overlay Manager to obtain tunnelled connectivity.

5.2.6.4 UDM

The User Data Management (UDF) supports user identification handling, access control and authorization, authentication management (key generation and key agreement) and subscription management.

5.2.7. DNS-server

In this orchestration framework, the DNS server acts as a service enabler and provides names for services in the slices. The DNS server receives dynamic DNS update from slice orchestrator and allows UEs to query IP addresses based on service names. In this way UEs can contact the slice services easily even if IP address is reassigned for each slice instance deployment. In our testbed, BIND is used to provide DNS *named* service.

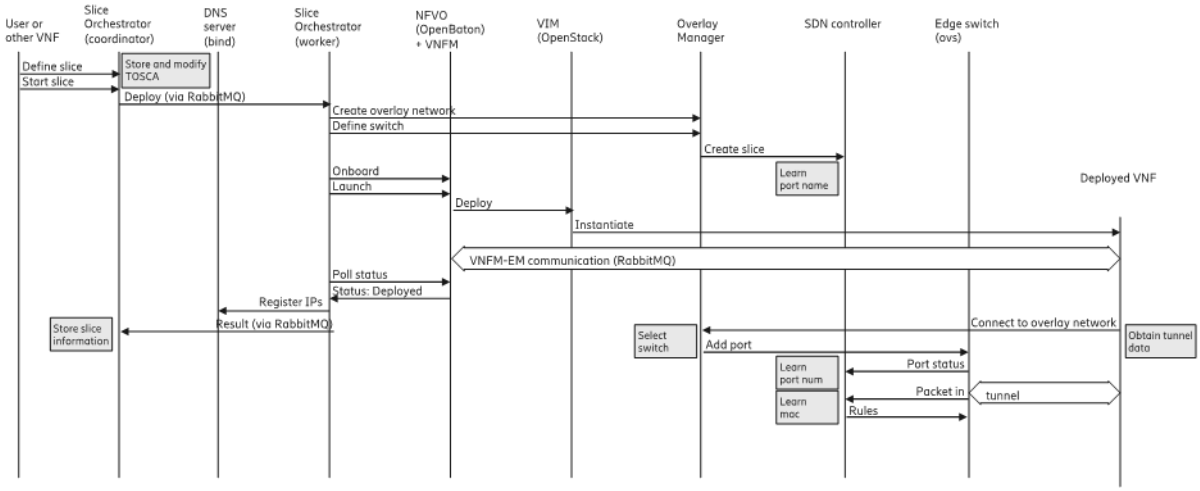

Figure 10. Slice deployment.

5.2.8. Sequence diagram

The sequence diagram that characterizes the IoT-orchestration use case is described in the following section.

Figure 10 shows the process of deploying a slice. The slice deployment can be initiated by a slice user or another VNF. At first the slice is defined using network service description in the TOSCA format and the slice definition is communicated to the slice orchestrator using the REST API of the Slice Orchestrator. In this example, we use the MuDoSO Orchestrator (as described in Section 4.3.1) developed by Ericsson. In addition to the slice definition the TOSCA CSAR (Cloud Service Archive) also contains the scripts for creating and configuring the VNFs in the slice.

Once the slice has been defined, MuDoSo defines an overlay network and also defines a switch in the overlay network using a REST API call to the Overlay Manager, developed by Ericsson. As mentioned earlier, the Overlay Manager allows VNFs to locate and connect with tunnels to named networks. The Overlay Manager is configured to make a call-back on the API to the SDN controller, which receives the information about an available slice network and can connect this overlay network to the SDN network.

Once the networks have been defined, MuDoSo uses the NFVO (we have used OpenBaton developed by Fraunhofer FOKUS) to create the VNFs. The NFVO creates the VMs using the VIM (we have used OpenStack) and deploys the VNF as well as the EM on the created VMs. The VNFM of NFVO requests the EM to run the scripts for the different life cycle phases. These scripts install essential VNFs on the VMs and configure it. The scripts also connect the VNFs to the overlay networks. While this process is ongoing MuDoSo polls the status from the NFVO until the service is completely deployed. In the next step, MuDoSo registers the public addresses of the services into DNS. Once the DNS registration is done, MuDoSo worker node collects all state information and sends it to the MuDoSo coordinator node, which stores it into a database. The state information is then used in a reverse order for removing the slice.

5.2.9. Network slicing concept

5.2.9.1 Dynamic slice instantiation and on-demand slice instantiation

Dynamic slice instantiation allows a slice to be set up automatically without manual configuration. It requires parameter values to be obtained and the slices to be configured automatically. Also the dependencies must be considered. We specify slice definitions on TOSCA templates, which define the nodes including VNFs and VMs as well as the connections and dependencies between them.

Related to slice instantiation is slice extension, where an already deployed slice is extended to a new location, such as a new edge location. In our approach, we deploy a new instance of the slice at the edge location.

5G!Pagoda envisions a scenario with thousands of slices, each designed for a very specific purpose. Many of these slices will be sparse, i.e. the number of users is relatively low and located in a few geographical areas. Such slices may be slices for city specific utilities, sensors or governmental purposes, or for private purposes such as enterprise networks or networks connecting a manufacturer's devices or services. In those cases, it does not make sense to deploy the edge VNFs in each area. Instead, VNFs should only be deployed in the areas with connected UEs. Moreover, in areas with a low number of UEs, the serving VNFs may be combined for multiple areas to provide the services to the UEs.

Consequently, we propose to deploy slices on demand when the first UEs connect to an area. When a slice is first deployed, it would only provide the core VNFs. Edge VNFs are added as UEs connect to that slice. Mobility also affects the placement and new edge VNFs must be deployed if a UE moves to an area not yet covered by the slice.

A baseline approach would be to deploy the slice when a session is established through a slice that does not yet have any instance deployed in that particular area. This is triggered by the NSSF receiving a query from the AMF for a slice instance (due to a PDU session). When the NSSF detects that a slice instance is not available in the TAI in the request, it asks the MuDoSo to create a new slice instance.

The challenge is the time constraint to create a slice instance. If VNFs are run on VMs, the time may be in order of minutes. VNFs may also need to be configured and connect to other services before being operational.

In order to reduce delay, we propose the following methods:

- *Pre-deployment based on registration*. The slice instance is deployed already when a UE registers to the network and obtains allowance to use the specific slices. Thus, all slices to which the UE can connect will be available even if the UE does not create PDU sessions on them.
- *Pre-deployment based on mobility*. The mobility pattern of a UE may be used to estimate the following area to which the UE is moving. The slice can be extended to the most likely neighbour areas of a UE based on the probability of the UE moving into them.
- *Pre-deployment based on history*. Slice instances are deployed in the most likely areas where UEs have been historically connected to that slice.

In all these cases, redundant slice instances are created that may never be utilized. This may be motivated if the cost of creating an instance at the edge is low. Unused slice instances can be removed after a time out.

While a slice is being deployed at the current edge, the UE can be directed to corresponding VNFs in a non-optimal location, e.g. in the core. Once the slice instance has been deployed the UE is transferred to the instance deployed in the optimal location. This approach requires network-initiated slice selection, as described in the following section.

5.2.9.2 Network-initiated slice selection

Slice selection, as specified today, is driven by the UE selecting the slice and requesting the permission to connect to that slice. The operator can provision the subscription with a set of configured NSSAIs from which the UE selects a subset as requested NSSAIs. The network can accept or deny individual S-NSSAIs and return the set of Allowed NSSAIs. Our slicing platform supports a new feature which is the possibility to transfer UEs between slices, where the transfer is initiated by the network. We specify this concept as *Network-Initiated Slice Selection*.

This can be utilized, for example, to

- Transfer UEs to an updated version of a slice
- Transfer UEs to a slice deployed closer to the UEs
- Transfer UEs to a slice version with different QoS characteristics
- Transfer UEs for security reasons, e.g. to a more isolated or restricted version of a slice

The transfer can be performed on two levels:

- Remapping a NSSAI of a UE to another NSSAI. This transfer is visible to the UE and requires changes to signalling.
- Transferring the UE to another instance of the same NSSAI.

In most cases, the latter is preferable. Special attention needs to be paid to session transfer. With persistent state stored outside the VNFs and shared between slice instances, sessions can be resumed in the new slice instance.

6. ICN/CDN end-to-end network slices orchestration

In this section, we describe the orchestration functions needed to deploy ICN/CDN end-to-end network slices. In this use-case, we highlight the deployment of sub-slices on multiple technological and administrative domains; and the stitching process to compose an end-to-end network slice.

6.1. Problem statement

Content delivery, especially video content, is a major contributor to mobile traffic, occupying around 70% in mobile communications. This video traffic is expected to increase much further in case of 5G. To cope with this ever-increasing mobile video traffic, we propose a new approach that aims at providing an efficient content delivery service by combining the dynamic CDN slicing concept with the ICN slicing concept. In this integrated ICN/CDN slicing, ICN takes care of request routing, in-network caching and delivery of popular content, while CDN serves and publishes the latest content only.

6.2. The 5G!Pagoda approach

In the ICN/CDN deployment, the slice owners authenticate to the multi-domain orchestrator and create CDN slices of VNF instances (e.g. cache, transcoder, and streamer) distributed over their envisioned service area, then link the CDN slice to an existing ICN slice of Named Data Network (NDN) nodes through its respective NDN gateway. End users can upload video contents to the platform, and operate the transcoding process of these videos remotely using virtual transcoders. Upon the first request for a particular content from a particular region, the requesting viewer will be served from the nearest cache in the CDN slice. Simultaneously, the content will be published into the ICN slice, serving the location of the viewer, through the NDN gateway. Subsequently, for the upcoming requests for the same content from the same region, the content will be served from the ICN network instead of being serviced from the CDN caches. Each ICN node (i.e. virtual router + content storage) will cache any chunk of the video content traversing the network while routing back the content to the requester. Hence, the next requests will be retrieved from the ICN slice without reaching the CDN slice. In-network caching strategies are defined locally at the NDN nodes based on several criteria, such as expiry time and hit-ratio.

6.2.1. ICN/CDN architecture

The aim of the proposed architecture, depicted in the **Figure 11** is to enable quick and flexible programmability to adapt to various user demands while ensuring better service delivery time. It also enables ICN/CDN slicing across multiple administrative cloud domains. In the following, we describe the main components shown in the **Figure 11**.

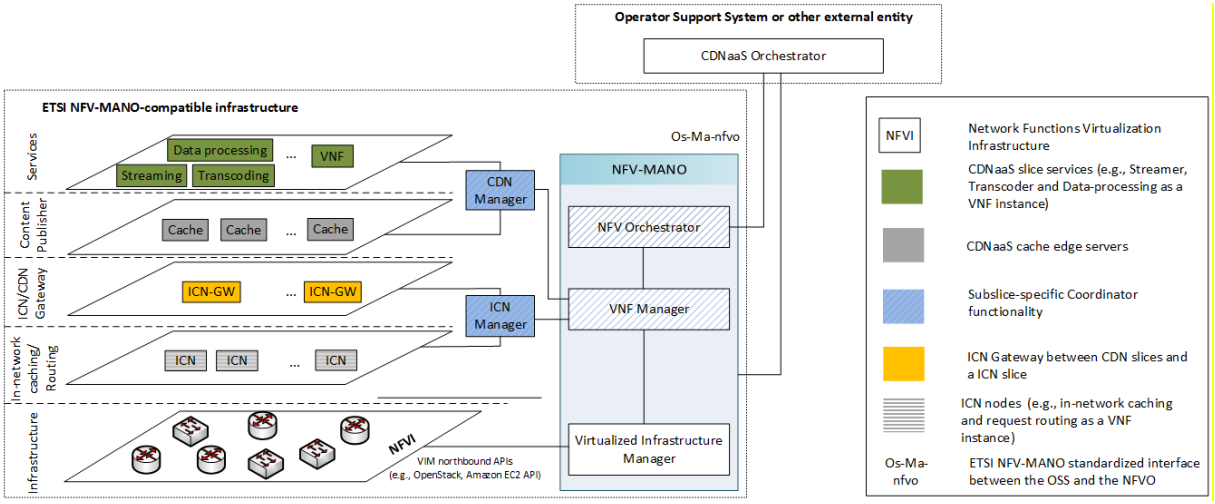

Figure 11. Multi-Domain ICN/CDN Slice Orchestration Framework.

6.2.2. Multi-Domain Orchestrator

Multi-Domain Orchestrator is the main component of the architecture. It is responsible for managing the whole system. This component has the global view of the overall topology and represents the multi-domain orchestration plane. The CDN/ICN Orchestrator allows content providers to deploy virtual CDN instances dynamically over a federated multi-domain edge cloud. Our system exposes a northbound API through which subscribers can request the creation of CDN slices and ICN slices for a specific duration over a specific area of service. A CDN slice consists of VNFs, deployed over the multi-domain cloud, including virtual transcoders, virtual streaming servers, and virtual caches. From another side, the ICN slice consists of VNFs, also deployed over the multi-domain cloud, including virtual routers and in-network caching components which forward the request hop-by-hop and route back the data to the end-users. These VNF instances are appropriately chained together and configured with optimally-assigned resources for specific service-level performance targets. The CDN/ICN Orchestrator constantly updates VNF managers regarding any action made to their respective VNFs under their directions.

6.2.3. Slice-specific VNFM

Slice-specific VNFM is considered as the brain of the target slice. The slice-specific VNFM communicates with the CDN manager and ICN manager reporting the virtual resources usage at the nodes level. In case of over-loaded ICN or CDN slice, the respective ICN manager or CDN manager will report the needs to the VNF manager. The latter will validate the request and send it to the orchestrator to add more resources in the suitable administrative cloud domains by connecting to their respective VIMs. Also, the CDN/ICN managers ensure the communication between the separate groups of VNF instances (e.g. CDN caches, transcoders, streamers, and NDN nodes).

6.2.4. Domain-specific VIM

As described by the ETSI NFV-MANO standards, the VIM is responsible for carrying out resource management and allocating virtual resources to VNFs. Virtual resources include computation resources, storage and networking resources. The NFV orchestrator communicates with VIM(s) through a Southern API to respond to the needs of slices for virtual resources.

6.2.5. CDN-VNFs

A CDN slice consists of three main functions: (i) Cache nodes that store the content published by the end-users. It is considered as the Content Publisher (CP) from the point of view of the ICN slice; (ii) Transcoder nodes are considered as a virtual network function that transcodes remotely videos from virtual cache servers to make the contents available in different qualities and resolutions for the users; and (iii) Streamer nodes are considered as a virtual network function for load balancing end-user requests and streaming the selected video with an appropriate resolution.

6.2.6. Dynamic NDN Gateway

Dynamic NDN Gateway is assigned dynamically at each ICN slice based on distance, either to the content publishers or the end user distribution. The ICN node selected as gateway should have a considerable amount of resources due to the heavy work it performs while converting and publishing the content from the CDN slice to the ICN slice. The NDN gateway has the protocol translation function between ICN and IP when needed and reads out the content cached in CDN server then reformats the content and transmits it chunk-by-chunk to the next NDN node or end-user.

6.2.7. NDN node

NDN node consists mainly of two functionalities: routing and in-network caching. It is a network node with full function of ICN protocol so that the forwarding, caching functions and data exchange with content repository are handled in a manner that satisfies the network resources and the underlying network configurations and policies. The NDN node utilizes PIT (Pending Interest Table) and FIB (Forward Information Base) as the fundamental data structures for forwarding process and Content Store (CS) in order to cache the content in the network. ICN nodes are equipped with the function to aggregate requests to the same content objects to reduce network traffic and server load. NDN node components consist of three main parts: (i) PIT: records retrieval path and aggregates requests of the same content. It indicates the return path of the requested contents and requests aggregation function when the same content request arrives at the node; (ii) FIB: forwards requests based on content name: Name-based forwarding; and (iii) CS: acts as in-network cache storage of the ICN node to reduce the duplicated traffic of the same content objects, and also shorten the response time.

6.2.8. ICN Management function

ICN Management function is responsible for network management functions. It handles the key network management functions, such as configuration, network performance, QoS, congestion control, and fault tolerance management.

In summary, this architecture proposed for the integration of CDN with ICN can demonstrate a solution for one of the 5G network problems on handling the increasing contents traffic. It provides an efficient content delivery service covering a wide geographical area that extends across the continents by the strength of pre-planned resource allocation with various services that CDN slice will provide. It also provides less volume of the regional traffic and shorter response time of the ICN slice by the use of in-network caching capacity closer to end-users.

6.3. Pre-assumptions

- CDN slice will be created under the control of EU testbed and ICN slice will be created under the Japanese testbed.
- In CDN slice, four kinds of major VNFs will be used, namely the coordinator, the cache server, the transcoder, and the streamer. The streamer will not be used for this ICN/CDN combined contents delivery service. The transcoder will be used in future, but it is not used for this system integration.
- One of the cache servers will be located in Japan, and connected to ICN slice created in Japanese testbed.
- In the following sequence, it is considered that the interactions with Japanese side orchestrator will be done by the coordinator. It, however, is possible that some part of the interaction will be done by EU side orchestrator.
- It is assumed that the ICN slice will be created as one of the slices on FLARE nodes.
- ICN VNF is implemented on FLARE nodes as Docker Image, and can be activated by the control of Japanese side orchestrator.
- ICN VNF has the capability to work as gateway. This function is activated by Japanese side coordinator.

6.4. Service sequences

6.4.1. FLARE resource registration

This sequence is the example of the use-case using FLARE (that is the programmable network node) as a resource to realize ICN node. In this example sequence, Orchestrator uses the API of the management software called FLARE Manager to control FLARE. And this sequence is preparatory for ICN networking to register FLARE as a resource to the resource pool.

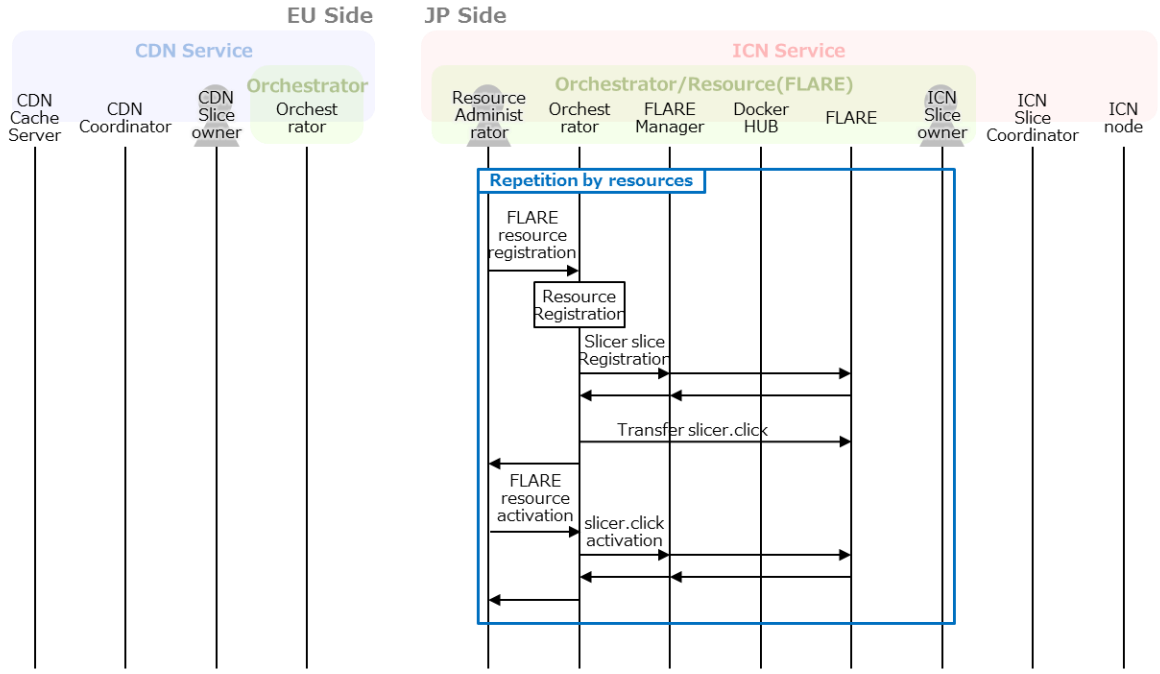

Figure 12. Sequence diagram for ICN slice creation with CDN slice.

6.4.2. ICN Slice creation

(1) ICN Slice owner requests the slice creation to Japanese side orchestrator via ICN Slice Coordinator:
- Deploy and activation on ICN node Docker Image
- Orchestrator assigns the resources and create the ICN slice

(2) ICN Slice Coordinator initializes the ICN node:
- Initiation of routing table (FIB)

Inputs from the owner are the number of nodes, node IDs, link configuration among nodes, link capacity, routing table information of each node.

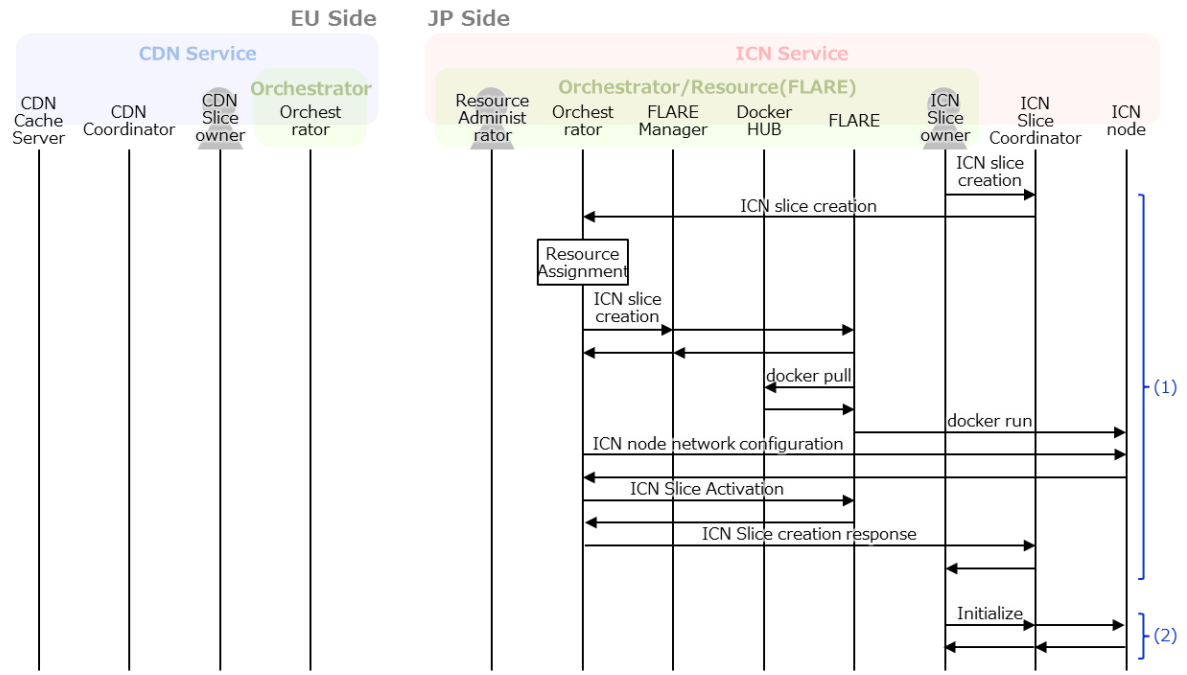

Figure 13. Sequence diagram for ICN slice creation with CDN slice (details of the steps 1 and 2).

(3) CDN slice is created by the control of EU side orchestrator via CDN Coordinator.

(4) CDN Coordinator notifies the CDN creation to Japanese side ICN Coordinator and request the slice stitching:
- CDN cache server location and its IP address, and coordinator IP address are informed

(5) Japanese side CDN Coordinator selects the ICN node which works as a gateway and indicates the corresponding node:
- Node selection algorithm will take into the account the situation
- Japanese side informs of the corresponding NDN cache server IP address to the gateway node

(6) When possible, Japanese side orchestrator will increase the storage capacity assignment to the gateway node by the instruction of the ICN Coordinator.

(7) Japanese side ICN Coordinator creates IP links between the gateway and CDN cache server, and CDN coordinator

(8) Japanese side ICN Coordinator informs the gateway node IP address and node ID to CDN coordinator

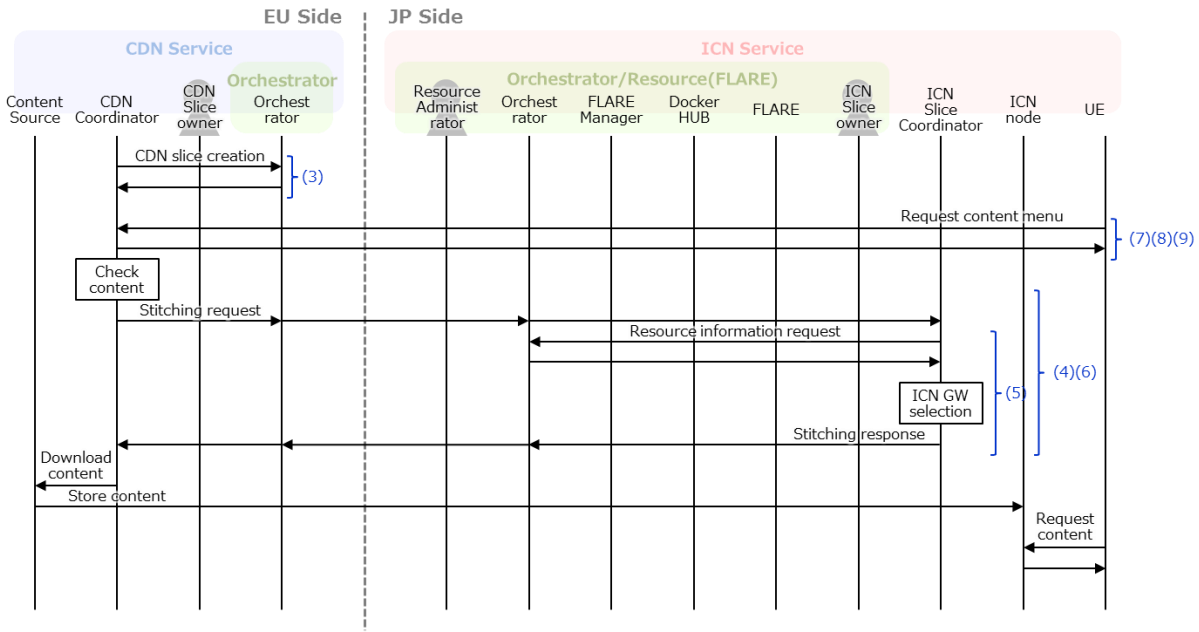

Figure 14. Sequence of the ICN slice stitching and content delivery service.

Content delivery service

(9) User requests the contents menu table to the coordinator of the newly created dynamic CDN slice

(10) CDN coordinator returns the contents menu table to the requested user

(11) User selects the contents. Selection is informed to the CDN coordinator

(12) Coordinator returns the exact name of the selected content to the user which is used in ICN slice:

- Exact Name is configured with the selected content name and the gateway ID as a routable prefix
- Contents file size is also informed
- Sequence 9 to 12 enables the CDN coordinator know the content popularity, which will be used for the cache server location decision and the efficient contents caching strategy

(13) CDN coordinator checks whether the request to the content is the first one or not:

- If it is not the first request, CDN coordinator leaves the sequence
- If it is the first request, sequence 14 to 16 are carried out

(14) CDN coordinator checks whether the requested content is already cached in the cache server

(15) If the requested content is not cached yet, CDN coordinator initiates the download process

(16) CDN coordinator communicates with the gateway, and initiate the content download procedure to the gateway carried by the gateway

(17) User requests the content to the ICN slice using the exact name returned from the CDN coordinator

(18) The sequence afterward follows the basic one of the ICN

6.5. Slice stitching

In this section, we describe the proposed framework that aims to integrate a CDN slice with ICN slice(s). The overall network system envisioned for stitching CDN and ICN slices is shown in the

Figure 16. In the envisioned system, a CDN slice consists of multiple virtual caches nodes, globally deployed across multiple cloud domains and supported by multiple virtual services as transcoding servers to convert original video content into several standards and resolutions. The VNFs in the CDN slices are hosted inside the containers, which are controlled by the SDN controller as described in the Figure 15.

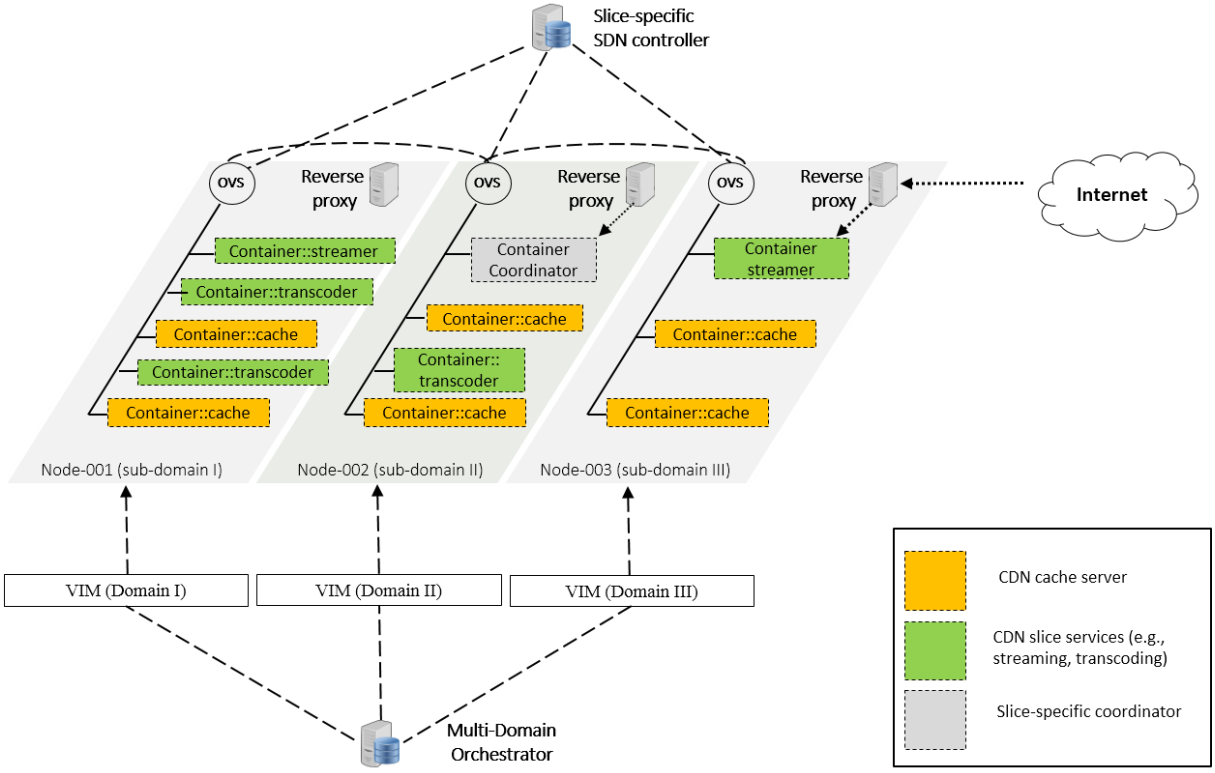

Figure 15. CDNaaS internal architecture.

A CDN slice is connected to its respective ICN slice via an ICN gateway as shown in the Figure 16. While a CDN slice can deliver content to viewers at the Globe scale, an ICN slice is used for the delivery of content over a specific region. The placement of CDN caches can be decided based on diverse metrics. They are considered as content publishers to ICN slices. Indeed, for an efficient integration between a CDN slice and its respective ICN slices, a gateway function is deployed in order to communicate with the CDN slice and convert the content into the ICN/NDN compatible packet format. The gateway function is provided in the ICN slice and is considered as the controller of ICN nodes of the ICN slice. From the viewpoint of traffic reduction, it is better to place the gateway functions closer to the cache servers. However, the number and the locations of caches may change according to the needs of users. Thus, there is a need to manage the number and locations of ICN gateways as per the changes happening at the CDN slice.

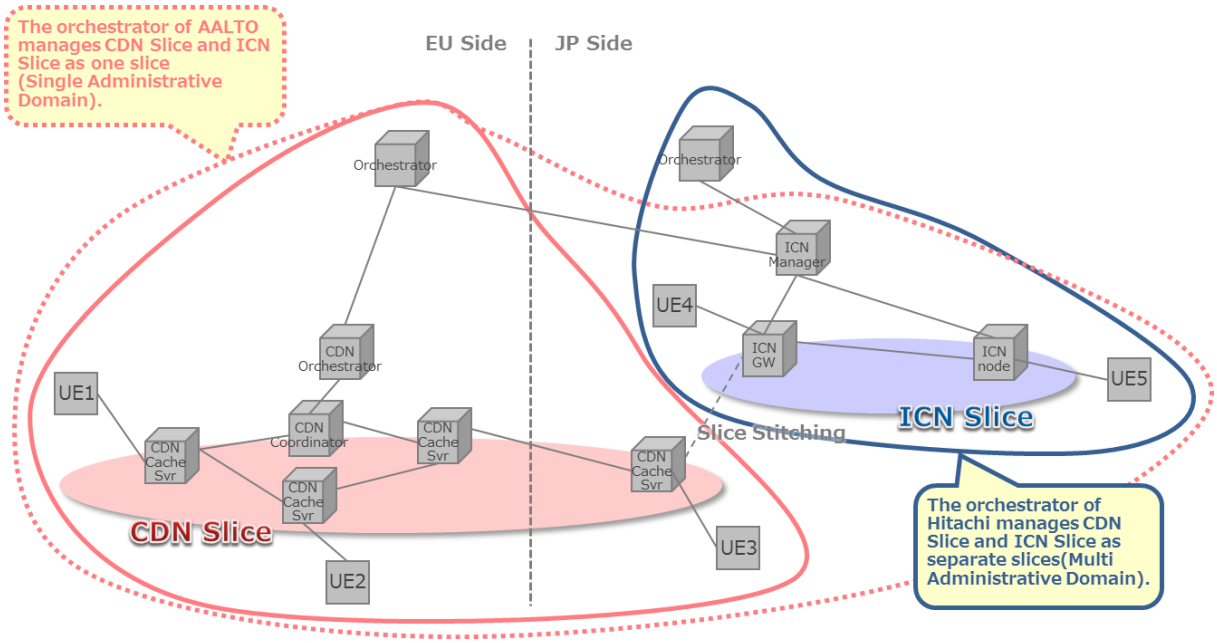

Figure 16. Outline of the ICN slice and CDN slice stitching.

UE4 and UE5 can connect to CDN Slice. So the CDN Coordinator can receive the trigger of the slice stitching from UE4 and UE5.

The ICN slice can be considered as a contents-delivery mechanism for the CDN slice in this configuration, even if the both are administrated independently. The ICN slice, however, is not just for the contents delivery of the CDN slice administrator. Other applications could utilise the ICN capabilities for their own purposes. The most appropriate configuration for such contexts should be the allocation of anti-corruption layer function on both sides, the CDN cache server and the ICN GW, those are interconnected for stitching in the data-plane as in the mid-case of **Figure 17**.

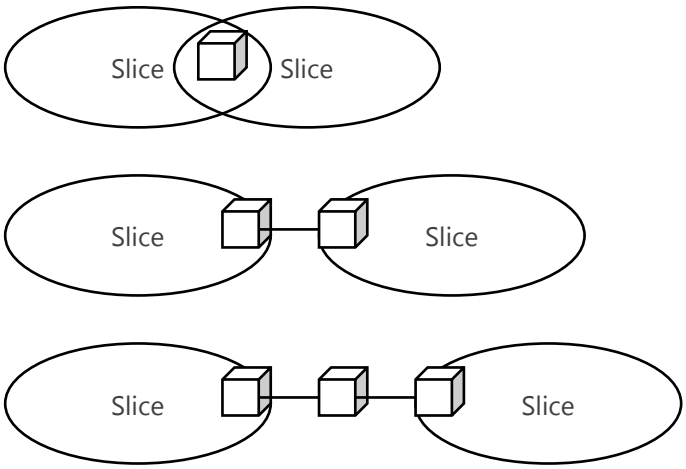

Figure 17. Slice stitching patterns.

The information for the traversal should be exchanged on the stitching request, where the syntax of the information is agreed in advance. The stitching coordination, which is considered as a sort of control-plane capabilities, should be essentially performed via communication between the

orchestrators of both sides (EU and Japan), nonetheless the CDN orchestrator directly talks to the ICN manager at the experimental configuration as a shortcut. The Hitachi Orchestrator might be considered as a null-function device. Logically the information exchange between the administrative domains is coordinated by the orchestrators in the experimental set-up.

The following information is exchanged between the CDN orchestrator (CO) and the ICN manager (IM) in the slice stitching sequence.

- Location of the cache node (CO -> IM)
- IP Address of the cache node (CO -> IM)
- IP Address of the CO (CO -> IM)
- Information of the selected ICN Gateway (IM -> CO)

There could be the possibility that an independent GW function (or business) performs the both of control- and data-plane capabilities depicted at the bottom of **Figure 17**.

6.6. Evaluation

In this section, we carry out a testbed experiment to evaluate the performance of the proposed framework that integrates CDNaaS with ICN. First, we focus on the mechanisms to publish content from CDN to ICN slice. Then, we present experimental results measuring the performance of content delivery through an integrated ICN/CDN slice.

To evaluate the performance of the whole integration, we set up our testbed using four IaaS cloud platforms: Two different OpenStack clouds in Finland, 1 OpenStack cloud in Japan and 1 Amazon EC2 in US West (Oregon). During the experiment, we consider a CDN slice that consists of CDN cache and streamer deployed on Openstack1 in Finland. We have also used an ICN slice that includes three NDN nodes distributed over three continents and hosted on Amazon EC2 in the West US, OpenStack in Japan and our second OpenStack in Finland. Finally, an NDN/CDN gateway is deployed in Finland, as illustrated in the Figure 18. These VNF instances were created from an Ubuntu server cloud image using a customized flavour. To introduce high load in the VM hosting the video service (i.e. busy CPU and exhausted RAM), parallel concurrent connections towards the server were launched with a total of 1000 requests being sent to the slice.

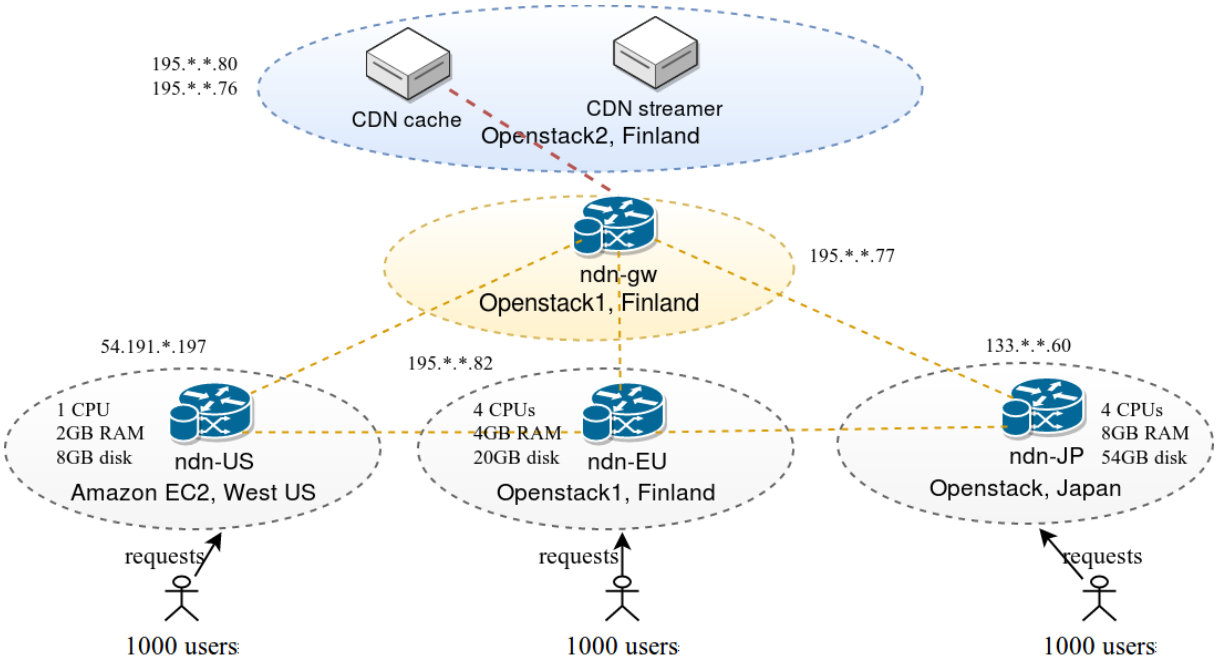

Figure 18. Sequence of the ICN slice creation and slice stitching.

In the beginning, the CDN cache contained a set of videos, linked to a NGINX[1]-based streaming server for content delivery. The NDN repository is empty, which means that the ICN network does not cache any content when the experiment starts.

Use case scenario 1: Contents reside only in the CDN slice. When the first request arrives, it will be forwarded hop-by-hop from a router to another till it reaches the CDN cache. Then, the NDN Gateway runs the protocol translation function between ICN and IP. Also, the NDN Gateway reads out the content cached in CDN server, reformats the content and transmits it chunk-by-chunk either to the user or to the next NDN node. The Figure 19 shows the time to publish content from CDN to ICN by varying the video content size.

Use case scenario 2: We consider a total of 3000 requests distributed equally over the United State, Europe, and Japan. Users are requesting the same 2 MB video content. As explained previously, only the first request will reach the CDN cache and then the ICN slice(s) will take care of the content delivery after that. The ICN slice works efficiently for content delivery thanks to the name based access and the autonomic content caching by the network nodes. The latter enables the content delivery from the nearby network node. Thus, it will reduce the duplicated content transfer on the transmission links, and also will provide shorter response times. We ran the experiments using the setup in Figure 18 taking variant measurements including mean delivery time by a node, throughput, network load at the core and edge network, and virtual resource usage.

[1] NGNIX : https://www.nginx.com/

6.6.1. Delivery time

The aim of our ICN/CDN integration is mainly to reduce the delivery time by caching the content in virtual routers near to the users. We run the experiment where *ndn-JP* is considered as the NDN node that serves the 1000 requests coming from Japan as shown in the Figure 19, and *ndn-EU* is considered as the NDN node that serves the 1000 requests coming from Europe, and *ndn-US* is considered as the NDN node that serves the 1000 requests coming from the United States. The first observation that we can draw from the figure is that time has a positive impact on the content delivery time. This can be explained as follows. In the beginning, the contents are served from CDN, then, along with time, users retrieve the content from the nearest virtual router (NDN node) in their respective regions. Consequently, the delivery time is reduced.

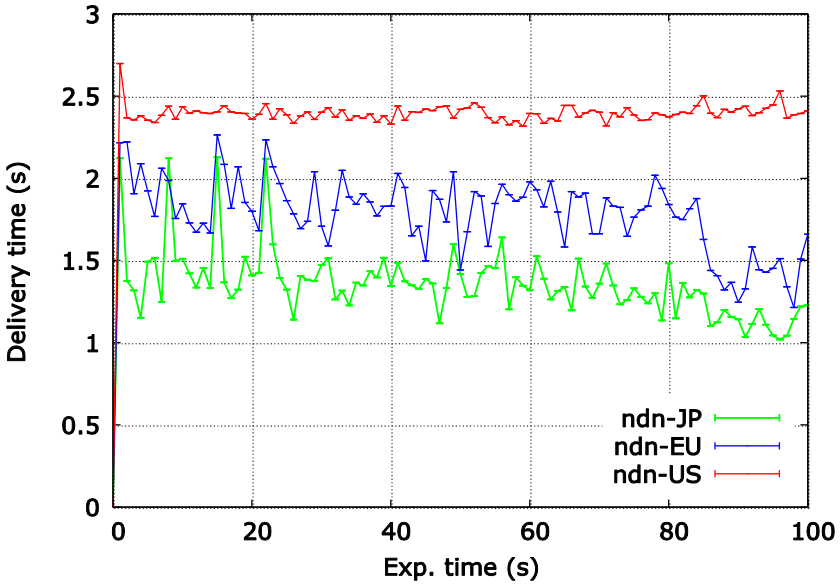

Figure 19. The average delivery time, from time of content request until the content is fully retrieved from ICN slice.

6.6.2. Throughput

During the experiments, we track the input/output of all nodes using Python. The Figure 20 summarizes the total received and transmitted packets during the experiments. Firstly, we notice that the CDN throughput is almost *nil*, since only the first requests reach the server. However, NDN gateway is considered as the most involved node for the transmission of packets in comparison to NDN nodes. Based on the observation that the NDN gateway is the controller node, most of the traffic passes through it. In the beginning, the content is converted from the CDN cache to the NDN gateway, and then all NDN nodes retrieve the content from the gateway until the content is fully cached in the concerned NDN nodes.

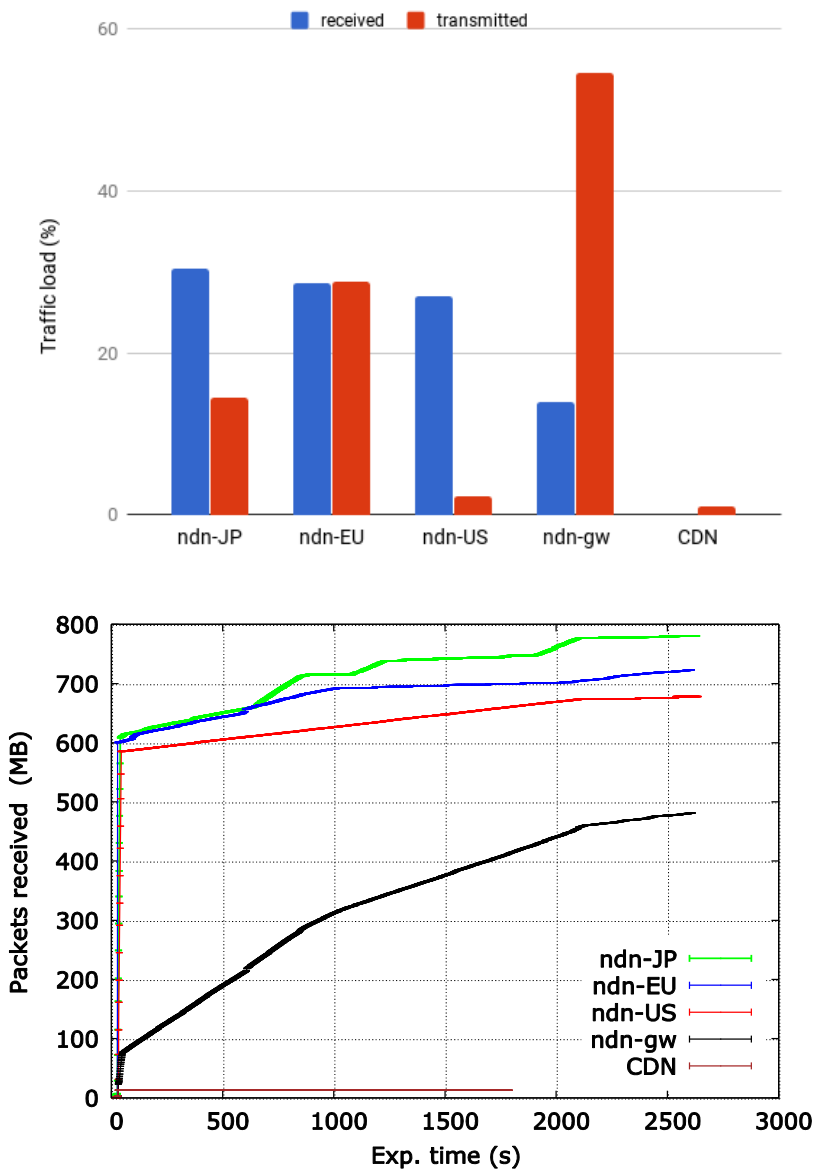

Figure 20. Traffic load on different ICN nodes. During our experiments, we keep track of the transmitted data packets and received packets through the node interface.

The figure also indicates that the nodes (JP, EU, US) received nearly the same traffic. However, the total of transmitted packets differs from a node to another which is explained by the inter-nodes communication and data exchange between NDN nodes. This demonstrate the key feature of ICN, whereby the network node with full functions of ICN protocol, including the forwarding, caching and data exchange with content repository, are handled in a manner that satisfies network resources, configurations, and policies.

Figure 20 supports the previous findings. At the first stage of the experiment, when an end-user requests the video content from Japan, the request will be redirected to the *ndn-JP* node, which is the nearest edge cloud to that user. In case that the content is not retrieved, then the node will request the next NDN node (e.g. *ndn-US* node). If the content is found, then it will be routed back to the requester following the same path. In case that the requested content is not found in the NDN nodes, the request will be redirected to the NDN gateway. Moreover, the gateway node will receive the lower network traffic load since its primary function is controlling and publishing the

content to the ICN network only when needed. In the reality, enhancing ICN with CDN is more efficient if we have a bigger slice of NDN nodes. Thus, either the *ndn-gw* or CP will be barely reached.

6.6.3. Virtual resource usage

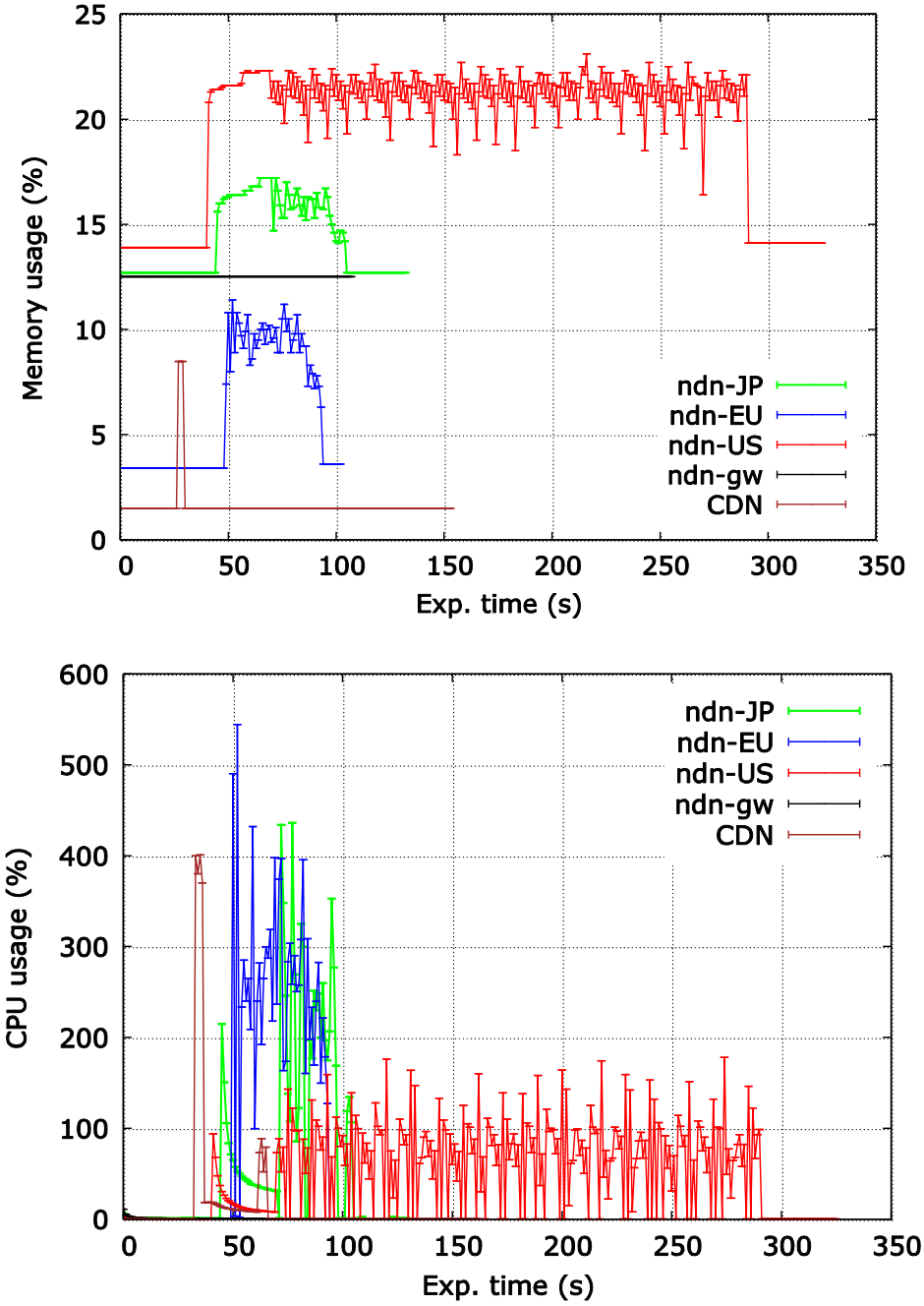

Figure 21. The virtual resource usage in different nodes in terms of CPU and memory.

In our developed ICN/CDNaaS platform, we use default in-caching strategies at the NDN node. We keep track of some measurements regarding the network, virtual resources usage (e.g. CPU and memory) to detect the involved nodes during a single request. In Figure 21, the CDN uses the least virtual resources. The resources are used only in the beginning of the experiment. The duration of

active usage of virtual resources differs from a node to another. Moreover, the figure indicates that the lengths (duration of time) of CPU and memory usage in *ndn-US* are due to its limited resources concerning the number of cores and RAM compared to other nodes.

7. Concluding remarks

In this document, we introduced the slice orchestrators devised in the 5G!Pagoda project. All slice orchestrator functions, i.e. the instantiation, management and deployment of a Network Slice over multiple administrative domains, and using resources from different technological domains, have been detailed and exposed via three different use-cases: end-to-end mobile network slice that includes RAN, IoT end-to-end slice and ICN/CDN end-to-end slice.

The logic behind it is to adapt to the use-case's needs in terms of functions. For instance: the end-to-end mobile network slice with RAN demonstrated slice orchestrator functions related to RAN sub-slice creation, enforcement and deployment over multiple technological domains; the IoT end-to-end scenario displayed scalable functions and deployment over multiple domains; the ICN/CDN showed functions related to the deployment over multiple domains and the stitching process.

The main features of the multi-domain orchestration presented in Section 4 lies on using of local orchestrators and using the slice stitching mechanism in order to obtain multi-domain slices. We demonstrated the usage of this approach in case of E2E orchestration of mobile networks. Such approach enables parallel orchestration of slice of each domain, requires however slice descriptors that should be included in slice requests.

The orchestrator described in Section 5 was used for orchestration of IoT slices. The orchestrator is doing multi-domain orchestration with different administrative domains, using resources residing in e.g. another provider's network; in case that the details of the internal architecture of the orchestrator have been provided. Due to the use the microservice approach, modular structure of the orchestrator and the message bus, the approach is easily extensible.

The CDN/ICN Orchestrator, described in Section 6 allows content providers to deploy virtual CDN instances dynamically over a federated multi-domain edge cloud. The solution exposes a northbound API through which subscribers can request the creation of CDN slices and ICN slices for a specific duration over a specific area of service. A CDN slice consists of VNFs, deployed over the multi-domain cloud, including virtual transcoders, virtual streaming servers, and virtual caches. From another side, the ICN slice consists of VNFs, also deployed over the multi-domain cloud, including virtual routers and in-network caching components, which forward the request hop-by-hop and route back the data to the end-users. For multi-domain slicing the slice stitching mechanism has been used.

The works detailed in this document rely on D2.3 [7] and complement the orchestration functions and procedures described in D4.1 [1], D4.2 [2]. Details on the implementation of the SO functions described in this document have been included in the D5.2 [5]. The described orchestration approaches will contribute to the final 5G!Pagoda architecture that will be specified in deliverable D2.5.

References

[1] 5G!Pagoda deliverable, "D4.1. Scalability-driven management system", 2017.

[2] 5G!Pagoda deliverable, "D4.2. Network Slice orchestration", 2017.

[3] Network Function Virtualization, "Management and Orchestration", ETSI Group Specification NFV-MAN 001, V1.1.1, Dec. 2014

[4] 3GPP TR 28.801, "Study and management and orchestration of network slicing for next generation network", ver. 15.1.0, Jan. 2018.

[5] 5G!Pagoda deliverable, "D5.2. First Report on Evaluation Methodology and its evaluation with use cases", 2017.

[6] 5G!Pagoda deliverable, "D2.1. Use case scenarios and technical system requirements definition", 2017.

[7] 5G!Pagoda deliverable, "D2.3. Initial report on the overall system architecture definition", 2017.

[8] 3GPP TR 23.799, "Study on architecture for next generation system", ver. 14.0.0 Dec. 2016.

[9] 5GPPP: the next generation of communication Networks and Services "5G Vision", technical report, Feb. 2015.

[10] A. Ksentini, N. Nikaein, "Towards enforcing Network Slicing on RAN: Flexibility and Resources abstraction", in IEEE Communications Magazine, Special Issue on Agile Radio Resource Management Techniques For 5G New Radio, June 2017.

[11] Network Functions Virtualisation (NFV) Release 3; "Evolution and Ecosystem; Report on Network Slicing Support with ETSI NFV Architecture Framework", ETSI Group Report NFV-EVE 012, V3.1.1, Dec. 2017.

[12] K. Katsalis, N. Nikaein and A. Huang, "JOX: an event-driven orchestrator for 5G network slicing", in Proc. of IEEE/IFIP NOMS 2018.

[13] Mobile CORD, http://wiki.opencord.org/

[14] X. Foukas, et al., "FlexRAN: A flexible and programmable platform for software defined radio access networks", in Proc. of ACM CoNext 2016.

[15] S. Vassilaras et al., "The algorithmic aspects of network slicing", in IEEE Communications Magazine, vol. 55, no. 8, Aug. 2017.

[16] F. Z. Yousaf and T. Taleb, "Fine-grained resource-aware virtual network function management for 5G carrier cloud", in IEEE Network, vol. 30, no. 2, Mar. 2016.

[17] 3GPP TS 24.301, "Non-Access-Stratum (NAS) protocol for Evolved Packet System (EPS); Stage 3", ver. 15.4.0, Sep. 2018.

[18] 3GPP TS 23.501, "System Architecture for the 5G System; Stage 2", ver. 15.4.0, Dec. 2018.