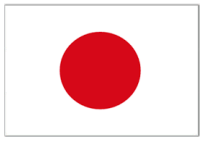

D3.2 – First slice components

UT, FOKUS, AALTO, Ericsson, Orange, WU, EURECOM, MI, NESIC

Document Number	D3.2
Status	Version 1.1
Work Package	WP3
Deliverable Type	Report
Date of Delivery	June 29, 2018
Responsible	Fraunhofer FOKUS
Contributors	Partner organisations
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.

Authors

Full Name	Affiliation
Abdelquoddouss Laghrissi	Aalto University
Tarik Taleb	Aalto University
Nicklas Beijar	Ericsson
Eleonora Cau	Fraunhofer FOKUS
Fabian Eichhorn	Fraunhofer FOKUS
Julian Siemiatkowski	Fraunhofer FOKUS
Marius Corici	Fraunhofer FOKUS
Hiroshi Takezawa	NESIC
Ping Du	University of Tokyo
Yoshiaki Kiriha	University of Tokyo
Toshitaka Tsuda	Waseda University

Change history

Version	Date	Status	Author (Company)	Description
1.0	26.06.2018	Final	Eleonora Cau (Fraunhofer)	Initial version
1.1	05.11.2018		Fabian Eichhorn (Fraunhofer)	Section 3.5 revisited

Executive summary

This deliverable provides an in-detail specification of the key technology features for the lightweight core network, the programmable data plane and also of the emerging network slice and in-service orchestration mechanisms. The deliverable represents an accompanying report for the work developed within the WP3 of the 5G!Pagoda project.

It follows and extends the work presented within "D3.1: Slice Components Design" [1] towards a detailed design of the different components. For clarity of the presentation as well as for the clear separation of the work-related concerns during the development, the same structure as for the D3.1 is maintained.

First, the deliverable describes the developments within the lightweight control plane area, concentrating on the different binding mechanisms, which can be deployed for the aggregation of the different functionality according to the specific requirements as well as on the different customized core network templates. Following, the same section describes the mechanisms for the slice selection enabling the devices to attach to the appropriate slice. With these mechanisms, the full flexibility of the control plane can be reached to address the different service requirements as well as the integration of the multiple slices within a single coherent system.

Following, the deliverable describes the programmable data plane specifications concentrating on the different functions, which have to be integrated within the data plane for the different slices. This includes the functionality related to the optimization of the RAN processing, the interaction with the control plane as well as the ICN and security related functionality. Albeit limited in scope, the functionality included represents the key technologies, which have to be integrated at the data path level to enable the same user processing elements to be customized for the need of the multiple slices, through this, proving that the complete slice separation at data path level is not necessary.

Furthermore, a new section is added to this deliverable related to the in-slice orchestration mechanisms. During the development of the project, it was observed that a set of mechanisms and algorithms are required to benefit fully of the flexibility provided by the new software networks. These mechanisms include the slice stitching algorithms as well as the mechanisms for state sharing as the basis for seamless scaling.

A final and more detailed set of information will be presented in the D3.3 deliverable, concluding the work of WP3. The practical implementation coming from this work package will be integrated within the testbed of WP5 together with the work from WP4.

Table of contents

List of figures

List of tables

1. Introduction

1.1. Objectives

The main objectives of 5G!Pagoda WP3 are to define the key functionality elements deployed as a part of the slices, which will enable a massive multi-slicing environment. The main premise of multi-slicing is that a highly customized network can be deployed for each service according to its requirements as a key differentiator from the current single uniform system, next to the privacy, isolation and transparency of infrastructure.

The work includes the functionality needed within the control and data plane as well as the slice composition functionality required for a full customization of the network slices according to the service needs. With this, each network slice can be customized during deployment according to the service requirements and can be deployed as a part of an end-to-end software network composed of multiple slices.

The functions and mechanisms are implemented, enhanced and deployed as part of the integrated tested developments and then validated within the end-to-end testbed within WP5. With this, the main benefits of such functionality are practically proven and, at the same time, they are brought together to prove that an end-to-end software network can be customized and deployed in the form of slices.

1.2. Motivation and scope

Within the initial steps of software networks, a large amount of attention was given towards the implementation of the appropriate hypervisors and management and orchestration frameworks and that is to enable the deployment of the software networks. With this in place, virtually any generic software network can be deployed and customized to the needs of the specific applications.

However, in order to be able to make the complete system function in an optimized manner, there is a need to consider also the adaptations to the software network functions themselves towards the use cases as well as towards efficient end-to-end deployment and service communication. This need stemmed from the initial results obtained by porting directly physical functions to virtual environment whereby the following effects were observed:

- Control plane functionality was not optimized towards the service requirements. For use cases which are similar, this solution is efficient enough. However, for highly differentiated services, the control plane shall be further customized. To answer to this requirement, 5G!Pagoda aims to provide a new lightweight core network architecture as well as the means to increase the network complexity according to the service needs. Furthermore, a mechanism for "in-slice next slice" selection is presented which enables during runtime, a component within a slice to select a next component to communicate to in the next slice.

- It was observed that data path functionality, whenever it was distributed among the different slices, introduced a large delay of processing due to a large number of intermediary nodes. To reduce this delay, 5G!Pagoda proposes a new deep data programmability concept wherein the dedicated functions for the different slices are deployed within the same switches, thus sharing a short data path while at the same time being customized to the service requirements.
- Furthermore, it was observed that deploying full virtual networks implied creating a very large redundancy in deployed network functions. This is due mainly to the same functionality deployed in multiple slices. For reducing this redundancy, the solution adopted by 5G!Pagoda is the "common slice" concept, as defined in 3GPP TR 23.799 [5]. For this, the critical technology element is the algorithms for slice composition (named slice stitching).

For these three directions, the scope of this deliverable is to build forward the different technologies following the directions and on top of the technologies from D3.1 [1].

1.3. Relationship with other WPs

This deliverable builds on top of the work presented in D3.1 [1]. Additionally, it has the following relationship with the other WPs:

- This document describes technical contributions for achieving the 5G!Pagoda system defined in WP2: D2.1 [2] and D2.3 [3];
- The details on Operations and Management are aligned with the work described in D4.2 [4], the equivalent second deliverable from WP4;
- The results of the work from WP3 including the one reported in this deliverable, the WP5: WP5 Integrated Testbed and Validation.

1.4. Structure of the document

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

- Section 2 provides key terminologies to be used in the document. It includes the descriptions of abbreviations and technical terms.
- Section 3 provides the overview of the lightweight control plane investigations from Task 3.1, continuing the work described in Section 4 of D3.1 [1];
- Section 4 provides an overview of the data plane programmability investigations form Task 3.2, continuing the work described in Section 5 of D3.1 [1];
- Section 5 describes the in-service orchestration mechanisms from Task 3.3, continuing the work described in Section 6 of D3.1 [1];
- Section 6 draws a set of conclusions and underlines future research work.

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
AAA	Authentication, Authorization and Accounting
AF	Application Function
AMF	Access Management Function
API	Application Programming Interface
APN	Access Point Name
CCNF	Common Control Network Functions
C-IoT	Cellular Internet of Things
CDN	Contents Delivery Network
DNS	Domain Name System
EBI	EPS Bearer Identity
FQDN	Fully Qualified Domain Name
gNB	next Generation Node B
HSS	Home Subscriber Service
ICN	Information Centric Networking
IRF	Interconnection and Routing Function
LRU	Least Recently Used
MF	Micro Functions
MME	Mobility Management Entity
NEF	Network Exposure Function
NRF	Network Repository Function
NSID	Network Slice Identifier
NSID-O	Network Slice Identifier for Orchestration
NSSAI	Network Slice Selection Assistance Information
NSSF	Network Slice Selection Function
OFS	OpenFlow Switch
PCF	Policy Control Function
PDN	Packet Data Network
PDU	Packet Data Unit
PGW-(C/U)	Packet data network Gateway (Control/User)

RAN	Radio Access Network
S-NSSAI	Single Network Slice Selection Assistance Information
SBA	Service Based Architecture
SD	Slice Differentiator
SDN	Software Defined Network
SGW-(C/U)	Serving Gateway (Control/User)
SIID	Slice Instance ID
SMF	Session Management Function
sNF	source Network Function
SSC	Session and Service Continuity
SST	Slice/Service Type
TEID	Tunnel Endpoint Identifier
tNF	target Network Function
UDM	Unified Data Management
UDR	Unified Data Repository
UE	User Equipment
UPF	User Plane Function

3. Lightweight Control Plane

3.1. Introduction

As described in D3.1 [1], a development roadmap was established for the development of the lightweight control plane, divided into five different steps.

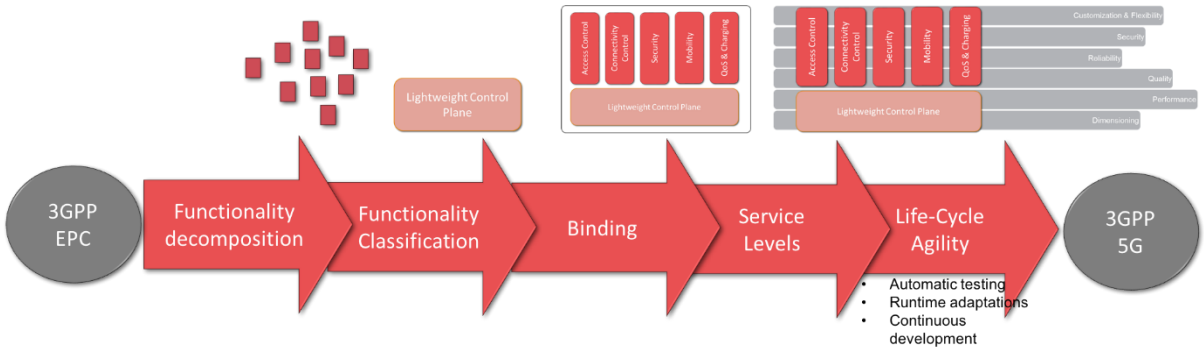

Figure 3-1 – Functional approach

As illustrated in Figure 3-1, the steps include:

- The functional decomposition of the core network – the core network was split into micro-network functions (henceforth called "micro-functions" or MFs). This work was presented in D3.1 [1].
- The functionality classification – the classification of the micro-functions in different classes, which enable their gradual development within the same functions or as different ones. This work was presented in D3.1 [1].
- Analysis of the binding mechanisms – to make the multiple network functions as an end-to-end system, there is a need for a set of communication mechanisms. This work was presented in D3.1 [1] and it is updated with the work described in Section 3.2.
- Service levels – to be able to deploy customized core networks and to update them in real-time, there is the need to define a set of customized core network models. A set of representative customized core networks is presented in Section 3.3.
- Life-cycle agility – modification of the customized core network during runtime may be needed as to adapt to other service requirements. This functionality will be implemented and described in the next deliverable of this work package.

3.2. Binding mechanisms

In D3.1 [1], a set of binding mechanisms between the different network functions was described, shortly summarized underneath. In this section, these mechanisms are updated with more detailed information as well as with new mechanisms, which have to be considered for the final implementation.

3.2.1. General considerations

The following classification of the mechanisms is considered:

Shared libraries – different micro-functions are connected to each other within a network function component by library calls. This mechanism is used to compose network functions from different implemented modules. The mechanism is to enable the integration of work of the different development teams as well as to customize the network functions deployed for the specific requirements. A major example for the success of this mechanism is the C user space libraries of the Linux Operating System where different implementation groups have developed their new functionality using the one provided by previous developments.

The major advantage of shared libraries is the natural integration of existing functionality as a part of the new component, practically without instance of implementing the same functionality twice (main success principle of software development).

However, shared libraries mechanism is prone to large software functionality overhaul when the shared library is updated. For this reason, the interface to the shared library should maintain the same structure, new functionality being rather added to the library instead of modifying existing functionality (main success principle of the 3GPP standardization). In reality, this is not always happening, resulting in a need for very often updates of network functions, which use the functionality. A shared libraries mechanism was developed as part of the Open5GCore as described in the following sub-sections.

Network Repository Function (NRF) – to find an appropriate network function to communicate with, a micro-function may use a network repository function.

As such requirement is usually needed only when a next network function has to be determined, the NRF mechanism should be used only when there is some dynamicity needed, compared with the shared libraries, which should be grouped for a lower level dynamicity. This dynamicity includes at least the selection of a next micro-function to serve the specific node as well as the scaling of the different components.

Additionally, it may enable, when a new function is added to the system, to broadcast the information about its appearance to other functions as well as to find its communication parties within the rest of the system.

For this reason, it is assumed that the NRF should be deployed only at the network function level and not at micro-function level where the shared libraries would be more appropriate.

Such functionality was implemented already in the form of a DNS server within the 4G core network, and this functionality will be ported to the 5G system. A short description of the DNS functionality as already implemented in the Open5GCore is presented in the next sections. Further considerations on how to implement the NRF for micro-services will be presented in a separate subsection.

Dedicated Protocols – communication using dedicated protocols is the main mechanism used within telecom networks with physical components. Such mechanism was implemented within the

4G network and is currently adapted by 3GPP for 5G networks. As 5G!Pagoda will mainly follow the standardization activities, there are no further considerations presented in this deliverable.

This solution is already implemented in the Open5GCore and is currently updated towards 5G protocols.

Interconnection and routing function – within D3.1 [1], a careful consideration was given to the implementation of an interconnection and routing function, which will enable the different network functions to communicate with each other. As presented in D3.1 [1], such a network function implementation introduces additional delay, as an intermediary component (possibly with a message queue) between the different components. In section 3.2.4, new considerations on how to implement such functionality are made and advanced, in an aligned manner and extending the 3GPP Service Based Architecture basic principles.

This solution will be implemented during the next phase within the Open5GCore.

State sharing – state sharing represents a mechanism, through which different network functions have the same information on a subscriber. Until 4G, the shared memory mechanisms where used for the communication within a network function between the different internal components enabling load balancing between multiple servers. With 5G, a new consideration has to be given: the state sharing should happen between components, which are not co-located within a high-speed bus.

Ultimately, any control layer core network procedure is based on the subscriber state and will alter this state. For example, a subscriber is passing from de-registered into registered state during the network registration procedures or the subscriber is bound to a target eNB instead of a source eNB in case of a handover. However, the procedures are not only this. They also influence the state of the other components through triggering state changes in other components using either dedicated protocols or interconnection and routing functionality. Because of this, state sharing cannot work independently and ultimately can be used only for the synchronization of the information between components of the same types, while other binding mechanisms should be used for other communications.

Furthermore, to be able to maintain the state consistency within the specific component as well as consistency of the communication with the other components, this component needs to be executed by a single network function. Thus, as a single component is communicating with the other components at a time, it is the only one, which could actively change the state. Because of this, the state sharing procedure should be triggered after the procedure is completed, or only in essential steps, by the active component. The state should be also communicated to all similar functions within the system, resulting in a broadcast of information towards the other network functions.

This solution was already implemented within the Open5GCore. The mechanism will be updated and evaluated as part of the next developments.

3.2.2. Grouping of binding mechanisms

The practical proof-of concept within the Fraunhofer Open5GCore showed that all these mechanisms are required for addressing different conditions of binding. Furthermore, it was observed that these mechanisms could be in some extent deployed within the same system, considering the specific state sharing mechanisms, as illustrated in Figure 3-2:

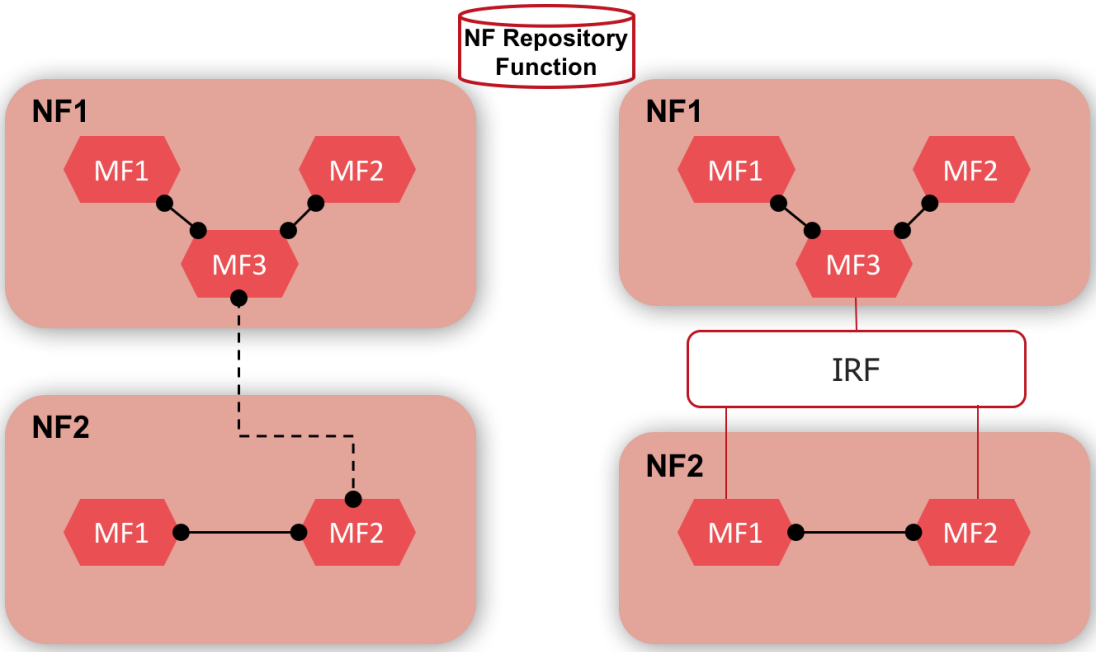

Figure 3-2 – Micro-functions binding mechanisms

3.2.3. Network Repository Function considerations

The Network Repository Function (NRF) has as a main goal to enable network functions to find other network functions to communicate with. In this section, this goal will be extended towards the comprehensive requirements coming from 5G networks following the architectural advancements from 3GPP TS 23.501 [6] and TS 23.502 [7] and using directly as implementation support the 3GPP TS 29.510 [8] implementation. These extensions will extend the packet core functionality to enable load balancing between network functions of the same type, a main requirement for the deployment of scalable software networks.

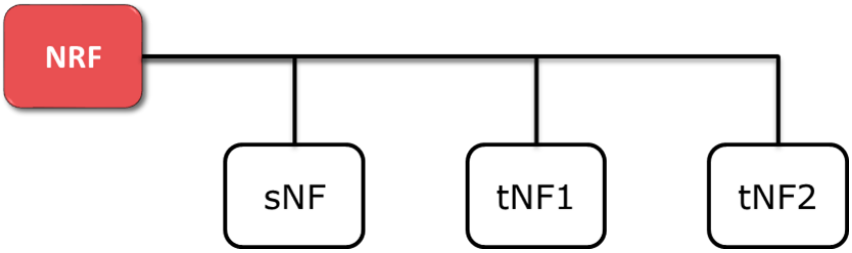

Figure 3-3 – NRF simplified architecture

The main requirements for the NRF functionality are:

- Enable a source network function (sNF) to discover a target network function (tNF1 or tNF2) to communicate with – this function was used to scale-in the network functions within the 4G system.
- Enable the sNF to receive regular notifications or event-based ones when the status of tNFs changes such as when a new tNF is deployed or retired – this functionality is currently under standardization for the 5G system.
- Enable the sNF to receive notifications on the load status of the tNFs for load balancing purposes – in the 4G system, load percentage was given as a parameter by the DNS function to enable load balancing between different components. However, in a software system, the network functions will have similar size, as it makes little sense to deploy network functions within virtual machines of different sizes. Hence, the percent based split will not be required anymore. Instead, a congestion notification mechanism is needed. In 4G, the network functions were always sending requests towards network functions, which were more centrally placed. So, the dimensioning of the system was easily realized based on the dimensioning of the first entities from the core network. However, with 5G, more of the network functions may be placed at the edge in a truly distributed system.
- Send re-selection events to the sNF – this will result into the deletion from the state of the subscribers of the information on the sNF, through this forcing that at the next procedure the NRF is again interrogated for the identity of which tNF to use.
- Send notifications on when a network function is planned to become inactive – with this functionality, the sNF can still send requests to the tNF for the specific subscribers bound to it, but not new ones. With this, a soft handover of the subscribers can be handled without requiring that the state if continuously and fully replicated between network components.
- Send events when network functions become inactive – these events should include information on the tNF that becomes inactive as well as what would be the procedure to execute for the subscribers, for which the tNF was used. The two possible options are: the usage of another tNF, replacing the tNF terminated or re-selection events as considered before.

For implementing the additional requirements, an extended network function internal functionality will be implemented. This includes:

- A notification mechanism from the newly started network functions when they are becoming active.
- A close interaction with the monitoring system to receive notifications on the status of the specific network functions – the monitoring system is able to determine when a network function will become inactive.
- A notification mechanism from the active network functions to notify the NRF when they become congested.
- An internal interaction map, based on automatically generated FQDN identities, to determine the specific location of the network function within the end-to-end system.

- A DNS-like system between the multiple NRFs in the system, in which each NRF is updated locally and the information is available globally through a tree-based NRF structure with NRFs located within the edge nodes as end nodes and the central NRF as a central one.

The additional functionality will be implemented using the specific 5G service based architecture mechanisms as specified in 3GPP TS 29.500 [9] and TS 29.510 [8].

3.2.4. Interconnection and Routing Function considerations

In D3.1 [1], an initial set of considerations was given for the interconnection and routing function, also shortly re-iterated in the introduction of this section. In this sub-section, several additional considerations are made, in order to enable the means to implement in an efficient manner the IRF. The same principles are considered for the Service Based Architecture (SBA) in 3GPP TS 23.501 [6].

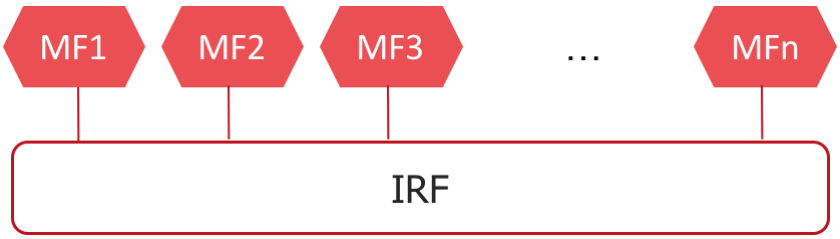

Figure 3-4 – Interconnection and Routing Function

A synchronization of the system is not needed. The IRF as a message queue provides a way to serialize all the requests in the system that do not need to be synchronized, as the control plane state at subscriber level is handled by the signalling protocols themselves and state machines in the specific network functions.

Subscribers are bound to network functions. When a specific network function is selected for a subscriber, then the requests will have to be forwarded to the specific network function as much as possible (handover between network functions should be possible, but not continuous, such as every step of a procedure to be executed by another network function). Thus, the specific requests have to be forwarded to a specific network function and not to a queue where to be directed randomly. A direct connection between the source NF and the target NF is considered better in this case as an intermediary routing function will only introduce an intermediary element and as the synchronization for the same subscriber can be executed at the source NF.

Events should be sent to groups of network functions. When a specific event happens, it has to be sent to multiple network functions, which are interested into the specific notification, and not to the others. This means that a multi-cast like system has to be implemented, which enables the network functions to subscribe to the events (and be part of the group) and not receive any other events. As the number of other events is expected to be very large, only the checking of the events would be very resource consuming, thus a broadcast-like system being inefficient. Instead, a direct subscribe-notify mechanism would be preferred.

Having these considerations, a new direction for the implementation of the interconnection routing function has to be considered, which uses only a logical bus and no specific common functionality.

The components are directly connected to the target components for the messages and the communication is done. The synchronization of the messages is executed at the source NF by not sending a next message before receiving the response for the previous one.

The following additional considerations, to be introduced within the next stage of the 3GPP service-based architecture, include the capacity for high availability and for load balancing. With the here considered mechanisms, the network functions are able to communicate directly and thus, circumvent the limitation of an intermediary function. However, such a function is usually used in the form of a redirect server or as a load balancer. With removing this function, additional considerations have to be made:

- The load balancing and the high availability should be able to function at procedure step level as not to introduce any state loss within the system.
- The load balancing and the high availability should not rely on existing functions and not add new functions to the system.

The only other network location where to put such a system is the source Network Function. To be able to execute load balancing, external information on the status of the target Network Function should be received, as presented from the NRF. With this information, the sNF can select the appropriate tNF at each step of the procedures.

A further implementation of the mechanisms within the Open5GCore will provide the additional validation results as well as the possibility to determine the best-practice engineering.

3.3. Customized Core Networks

To be able to address the different communication requirements for the different slices, the core network will have to be functionally customized to deploy and to customize a system specific for the purpose. In this section, a set of considerations on the customized core networks are made, enabling the slice providers to better adapt their offer towards the tenants.

The requirement of a customized core network stems from the following underlying considerations:

- Deploying a complete system is too complex for some of the tenant networks – a complete telecom system is assumed to have more than 200 active components, due to the requirements put on the carrier-grade communication for high availability, load balancing and redundancy as well as on the lawful interception, charging, border control and roaming peering with other operators, etc. Albeit in a simplified prototype, the reduction of one or two network functions may not look significant, for the full end-to-end system, this would prove large advantages. This includes the resources consumption of the system as well as the required management personnel for handling the software system (although slicing promised automatic management, still, due to a large number of parameters, the final decisions will have to come from a human administrator).
- Deploying the same system in all of slices would be easier to deploy and to manage. A main counter-argument to the customized networks is that the same system deployed multiple times

will have the same issues and the same limitations, so its management would be less complex. Such system could be customized to address the service requirements of the tenant through the means of introducing proper policy-based management information as well as through not using the functionality not required in the specific slice. Thus, a full system answering all requirements could be deployed, while a large amount of the functionality would remain unused. Additionally, the deployment and lifecycle management of unused functions also need efforts, and in case of unfavourable licensing model, the network operator will pay license fees for unused functionalities.

This argument does not hold, as the communication patterns of the subscribers being highly different (not like in our case only related to human subscribers and their behaviour), the behaviour of the system would be highly different. Therefore, the advantage of the same type of management is practically lost, as the system will require different interactions with its human administrator.

- A unique management framework could be deployed for the multiple customized slices. While the customization relates to the usage of the same software platform for the different use cases (i.e. the same network functions are customized for the use cases), they will use the same network management systems. Albeit different in the management requirements and events, having the same management system would enable a fast learning and adaptation of the system administrators to a new network, as the same interface will be used. With this, deploying customized networks is overall more efficient than deploying dedicated networks coming from different suppliers.
    - Note: a very strong tendency from the network operators was observed in the last years to consolidate the management of the software network functions, as well as PNF-based networks, around a very limited number of management and orchestration frameworks. Considering that this is already happening, operators can request from the vendors that the same management system will be used for the different components, practically offering an opportunity to unify the management plane in the benefit of optimized. However, such developments usually are secondary to the development of the new services to the subscribers, and until now have not succeeded mainly due to the main requirement for the development of the network to address the needs of subscribers in order to have a larger market share, while operations optimization being left for later dates.
- The differences in requirements between the different slices are not large – although different requirements in terms of size of the network and in terms of communication characteristics such as delay and reliability are pushed from the different potential slice owners, this may not be required in the end-to-end system. Before actually seeing the system working, the slice owners tend to exaggerate on their requirements, as they are considering use cases, which would be implemented only in specific situations or in later deployment phases. However, it was observed that once a generic system is deployed and the slice owner is in charge of its customization, the perspective changes towards investment into new functionality, which is usually very carefully pondered. Because of this, only the really required requirements are pushed for the implementation, while the others remain not implemented. Based on this technology development and deployment phenomenon, we could assume directly that the slice

requirements from today are highly exaggerated for most of the use cases. To prove this, the following generic slice models, albeit do not respond to all the requirements, provide a sufficient variation base to cover most of the slice requirements.

3.3.1. Mobile Broadband Slice

The mobile broadband slice represents the continuation of the mobile broadband service currently provided by the mobile network operators to their subscribers. It gives a large capacity communication service for a relatively large number of subscribers.

For this, a properly dimensioned network is deployed where the radio access network resources remain the main bottleneck, while the rest of the network is appropriately dimensioned to handle the specific load. When the radio components are scaled for another load level, this implies that the core network will be shortly needed to scale too. Considering this property, it is foreseen that the core network will be scaled "at once" (i.e. all the components at the same time), according to the scaling of the radio resources, automatically and independently of the momentary load of the core network.

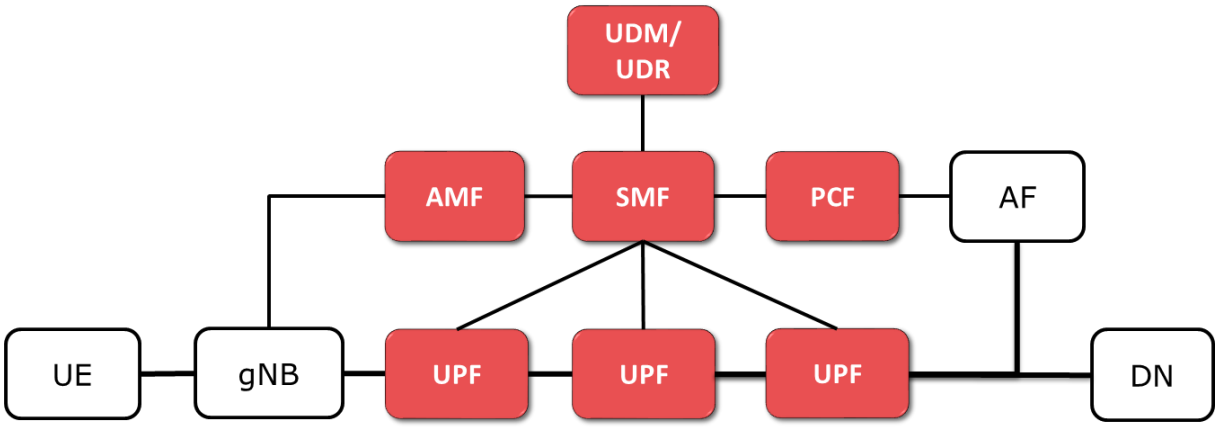

Figure 3-5 – Mobile Broadband Slice

As illustrated in Figure 3-5, the mobile broadband slice includes the main components of the 5G core network:

- Access Management Function (AMF) – for access and mobility management of the subscribers in the system, able to provide support for the access control and RAN handover support;
- Session Management Function (SMF) – to establish the end-to-end data path across multiple, chained UPFs. With this, the data path is established from the border of the network to some centralization points (may be many), according to the 3GPP standardization;
- Policy Control Function (PCF) – providing subscriber and network oriented policies and events for managing sessions and the access;
- Unified Data Management/Unified Data Repository (UDM/UDR) – maintaining the subscription profiles for the subscribers;
- User Plane Function (UPF) – forwarding entities within the data path. With 5G, the carried data packets do not have to be IP only, thus the system being also usable for other communication

types. The main requirement for non-IP communication across the data path is that there is a direct forwarding mechanism between the last UPF and the AFs using the specific communication technology, as to enable the end-to-end data path.

3.3.2. Small Messages Slice

The small messages slice is a specific type of slice, which enables the communication with the devices in a uni- or bidirectional way. Additionally, the small message slices could rely on different security levels, from immediate broadcasting to all devices connected up to bidirectional authentication and authorization and encrypted small messages.

For this system, a control plane only core network is deployed, which enables the communication of messages with the UE and their forwarding to the specific application.

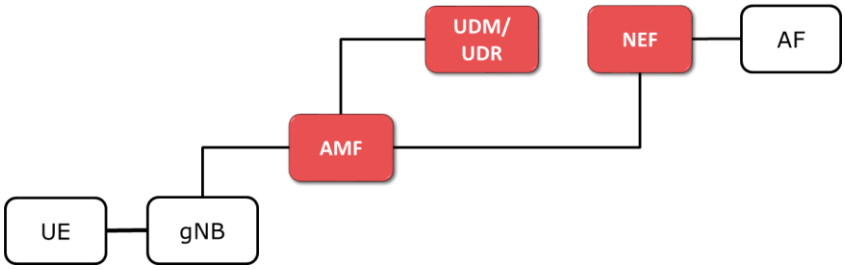

Figure 3-6 – Small Message Slice

As illustrated in Figure 3-6, the small message slice includes the main components of the 5G core network:

- AMF – handing the authentication and authorization, reachability of the devices as well as the forwarding of the messages between the UE and the NEF;
- UDM/UDR – subscription profile repository, providing authentication and authorization information as well as reachability information;
- NEF – the Network Exposure Function (NEF) to forward the non-IP data packets received over the control plane to the AFs acting as a gateway of the packet core.

3.3.3. Distributed/Low Delay Data Path Slice

For being able to address the ultra-reliable low delay communication, a new type of core network slice is required, in which multiple data paths can be established covering the following situations:

- Low-delay communication to a local offloading point-of-presence – a UPF has to be placed within the edge location to enable the local offloading of data traffic. The offloading should be selective for the devices within the specific location.
- Low-delay communication towards remote data network points-of-presence – for supporting low delay end-to-end communication, especially of non-IP type, an outbound UPF should be placed close to the point-of-presence.
- Handling locally mobility procedures – a set of chained UPFs should be placed in the network, enabling that the local mobility procedures are kept locally at the data path level.

- Selective forwarding to different data networks – a set of chained UPFs should be placed in the network, enabling the classification and the forwarding of the data traffic towards the multiple domain network points-of presence depending on the momentary conditions (even within the same bearer).
- Data path reselection – in case of congestion on the data path, the data path should be re-selected using other intermediary components to avoid congested data path elements.

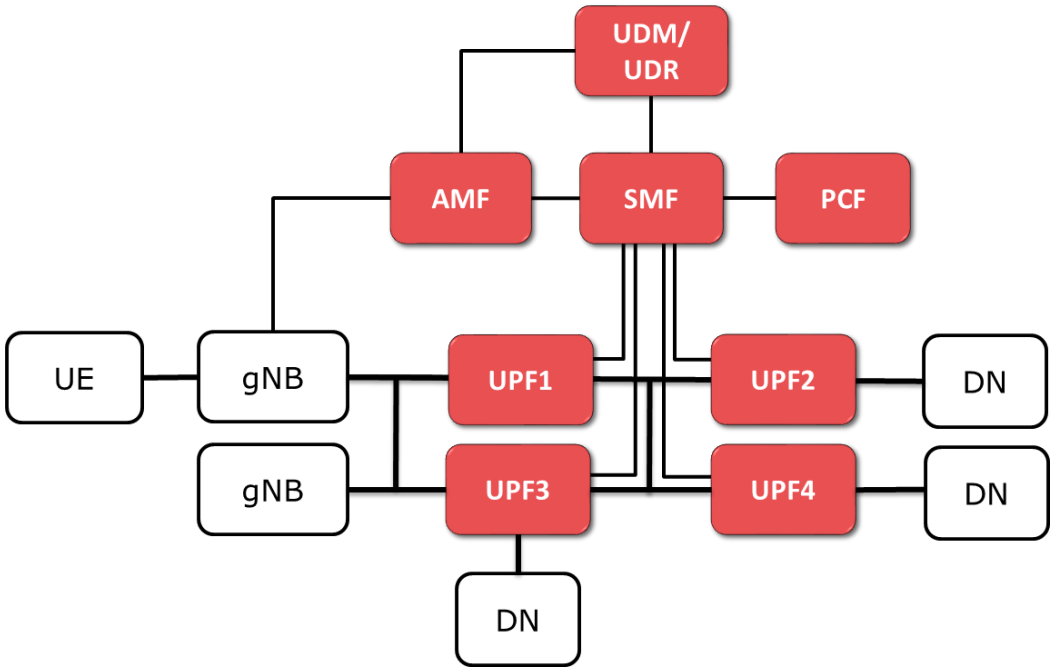

Figure 3-7 – Distributed Data Path Slice

For this slice, a set of UPFs is added to the system, covering the previously described functionality:

- UPF2 and UPF4 represent the UPFs close to the Domain Networks within the network, enabling the control of the outbound data traffic.
- UPF3 represents an offload UPF enabling the forwarding of the data traffic to the local data network.
- UPF1 is able to classify data traffic and to forward it to the UPF2 or UPF4 depending on the target data network.
- UPF3 is able to classify the data traffic and forward it either locally or to the UPF2 or UPF4 data networks.
- In case of a handover between the gNBs, the data path can be anchored in UPF1 or in UPF2 depending on the handover procedure (e.g. from a data path gNB1-UPF1-UPF2 to gNB2-UPF1-UPF2 *or* gNB2-UPF3-UPF2). The selection can be done based on the subscriber identity and on the specific local network topology.
- In case of a congestion at the UPF1 level, an internal data path change procedure can be executed (e.g. from gNB1-UPF1-UPF2 to gNB1-UPF3-UPF2).

This example slice model uses 4 UPFs to be able to cover these procedures in a prototype form. It is expected that in real deployments, the same principles of network function placement will be used, however resulting into other topologies.

3.3.4. Edge-Central Split Slice

A specific type of slice has to be considered for a network, which enables the deployment of edge functionality, as this edge functionality can be used for the optimization of the local communication or for the improvement of the signalling across unreliable backhauls.

In this sub-section, different split models between the edge and the central location are considered. A careful attention is given to the deployment scenario where the edge is deployed in Japan and the central node in Europe (or vice versa) as this represents from the 5G!Pagoda project the most likely deployment of an end-to-end slice offering services on both locations.

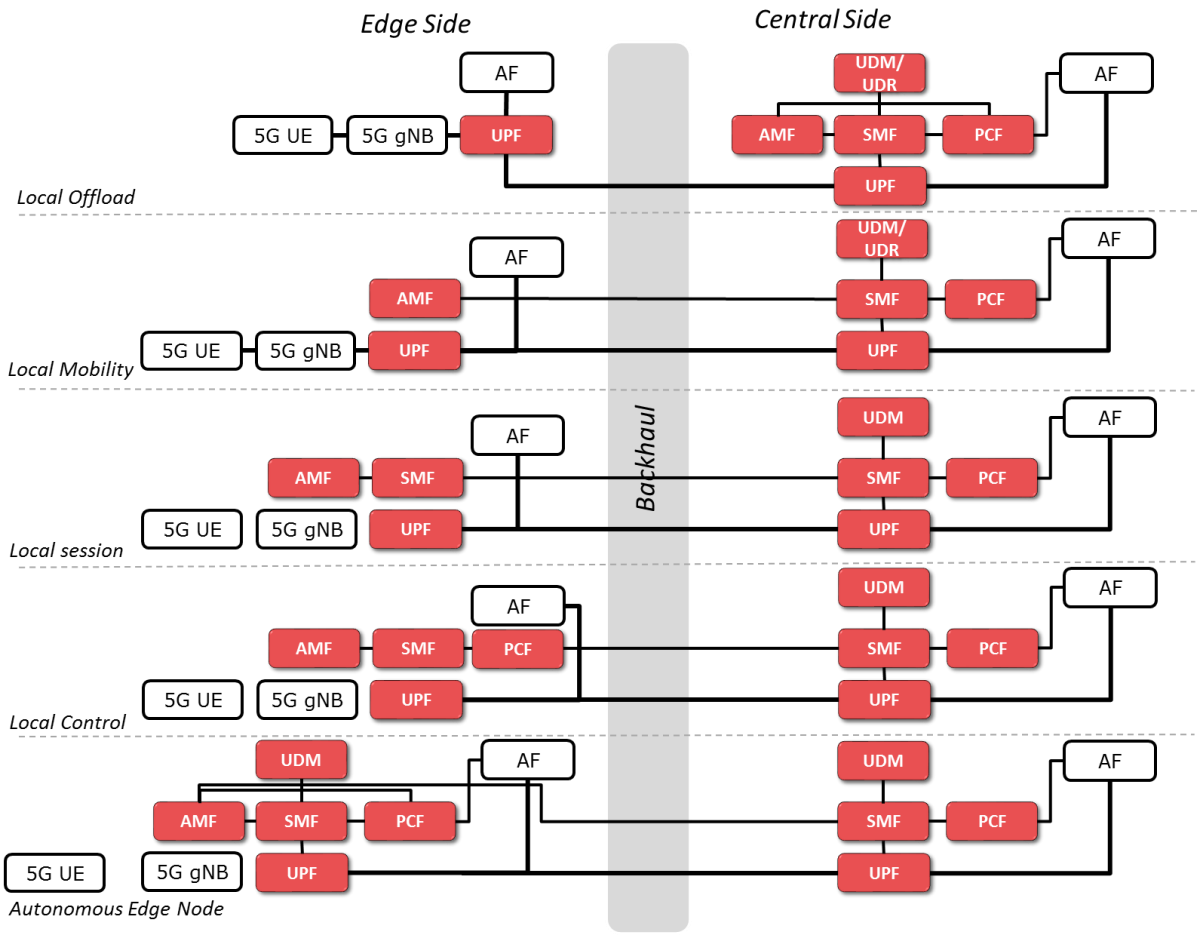

Figure 3-8 – Models of Edge-Central split

The following Edge-Central Split slice templates are considered:

- Local offload – centralized control plane and local offloading of data traffic. This slice template is highly efficient when the backhaul network has a low delay and relatively high network capacity, and the front-end 5G network has extremely large capacity. In this context, signalling will be executed as in the centralized networks, while the data traffic can be locally terminated;

- Local mobility – the access control and mobility management are executed at the remote side, while the session management is centralized. In this case, to be able to use the local UPF, there is a need that at least 4 messages are exchanged across the backhaul (AMF-SMF, SMF-UPF and responses), which may be problematic for a high delay backhaul. Instead, for a low delay backhaul, a local offload would provide a more suitable solution, as less AMFs are deployed.
- Local session – moving an SMF also at the edge node and then forwarding the requests to the centralized SMF, which handles the interaction with PCF. The local session split is currently not supported by the 5G standard as it implies the chaining of SMFs. However, this is possible if the edge node represents a visited domain. To make this slice model possible, an internal operator roaming model should be designed since currently SMFs are supposed to be chained only in the roaming scenario.
- Local control – in a local control slice, also the PCF would be placed within the edge node. In this case, the edge node will have all the functionality necessary to establish "end-to-end" data paths between the UE and the backhaul. However, an interaction with the home central SMF would be required as in the case of the local sessions. Compared to the local session slice model, the only main difference is in the delay, on which the SMF and AMF receive PCF policies, and events, which in most of the cases could take longer. Considering this, a local session slice would be better to be deployed for both low and high delay backhauls.
- Autonomous edge network – an autonomous edge network includes all the components of the 5G core. This would include a UDM/UDR with the subscription profiles of the users. Alternatively, only a UDM may be deployed, which acts as a proxy for the single centralized UDR and maintain local caches of subscription profiles seen in the specific area. Furthermore, the AMF is communicating with 2 SMFs directly: the one located in the edge network and the one in the central location, with this, eliminating the need for the SMF-SMF interface.

  The autonomous edge network could work with intermittent or even without a backhaul to the central location, representing the best option when the central location and the edge one have equivalent resources. This would be the situation for most of the EU-Japan deployments.

3.4. Slice Selection Function

According to D2.3, the 5G!Pagoda architecture should support the following requirements on slice selection.

- A UE should be mapped to a specific slice. UEs that have the necessary support should be able to connect to multiple slices, using multiple parallel PDU sessions belonging to different network slices.
- The network should provide a slice discovery and selection mechanism for informing a slice aware UE about the available slices and allow it to make a selection of a slice matching its requirements.
- Mechanisms should allow for attachment of slice unaware UEs and UEs with predefined slices (for example in SIM).

- The slice selection process can be driven by the UE or by the slice operator. A negotiation between the network and UE during the slice selection process should also be possible. Slice selection should be based upon policies configured by the operator.
- The mechanism should support the creation of a slice on-demand, if a required slice cannot be matched.
- The Network Slice Selection Function should be backward compliant with NextGen and 5GS mechanisms described in 3GPP TR 23.799 [5] and TS 23.501 [6].

Concerning slice selection, we build on the foundation specified in 3GPP TS 23.501 [6]. This involves primarily two network elements: the Network Slice Selection Function (NSSF) and the Access and Mobility Function (AMF). A slice, as visible to the UE, is identified by a Single Network Slice Selection Assistance Information (S-NSSAI). The S-NSSAI is composed of a Slice/Service Type (SST) describing the type of service to which the network slice is designed, and the optional Slice Differentiator (SD), which is used to different network slices of the same service type. UEs may contain a set of pre-configured S-NSSAI stored in a local storage. This is called the Configured NSSAI. In its registration to the network, the UE requests the access to a set of slices by specifying the Requested NSSAI. The Requested NSSAI can be a subset of the Configured NSSAI. The slice access request is sent via the AMF to the NSSF. The AMF also adds the set of Subscribed NSSAI to the request as obtained in the subscription information from the UDM. The NSSF replies via the AMF with the set of Allowed NSSAI. The Allowed NSSAIs are specific to a PLMN and a traffic area (TAI). The UE is responsible for storing the Allowed NSSAI. On PDU session establishment, the UE indicates the S-NSSAI that is the target for the session. The AMF sends this request to the NSSF, which replies with a slice instance for the session.

While the S-NSSAI is the slice identifier visible to the user, the instance as implemented in the network is identified by the Slice Instance ID (SIID). There may be several slice instances mapped to the S-NSSAI, allowing to deploy different instances in e.g. different areas. In principle, a slice instance could also be mapped to several S-NSSAIs, e.g. if a more generic slice replaces a set of legacy slices.

In the 3GPP architecture, both NSSF and AMF are part of the Common Control Network Functions (CCNF). This is extended in the 5G!Pagoda architecture, where the Slice Selection Function is part of both the Common Slice and the Dedicated Slice. The Slice Selection Function in a Dedicated Slice complements or replicates the Slice Selection Function in the Common Slice. In 5G!Pagoda, each network slice is identified by Network Slice Identifier (NSID) and by Network Slice Identifier for Orchestration (NSID-O).

According to 3GPP TR 23.799 [5], it should be possible for network functions to change the set of network slice instances accessible for particular UE during ongoing session. However, there is no description of the process for this mechanism available. The available mechanisms for slice selection can be classified as:

- Network-based mechanisms, in which the network is in charge of selecting the appropriate slice for the UEs while the devices are unaware, which would be the selected slice;

- UE-controlled slice selection mechanisms, in which the UE makes the decision to which slice to connect;
- UE-assisted slice selection mechanisms, in which the UE makes an indication to which slice to connect.

Within 5G!Pagoda, Ericsson has implemented a slice orchestrator, an AMF and an NSSF in order to prototype new concepts related to slice selection. Slice selection is coordinated by the NSSF. Currently, the NSSF is running as part of the Common Slice, but it can be replicated to Dedicated Slices. Slice selection is assisted by the AMF, which is the core control plane gateway for the UE. The AMF interfaces the UDM in order to retrieve the information about the Subscribed NSSAI as part of the subscription information of an UE. The implementation of the NSSF is based on the current version of 3GPP TS 29.531 Rel. 15.0.0 [10]. The prototype implements the Nnssf_NSSelection service. The Nnssf_NSSelection service implements two queries, which are independent from each other even if they use the same API endpoint:

- During the registration of an UE, for requesting the Allowed NSSAI based on the Subscribed NSSAI and Requested NSSAI.
- During the PDU session establishment, for requesting a slice instance for an indicated S-NSSAI.

Both requests are made by the AMF on behalf of the UE. The presence of the fields sliceInfoRequestForRegistration or sliceInfoRequestForPDUSession determines which of the requests are in question. Our current version of the implementation follows the Nnssf_NSSelection API defined in 3GPP TS 29.531 [10] closely, apart from omitted AMF selection. It further extends the API with a few fields as explained below.

3GPP TS 29.531 [10] also defines another service named Nnssf_NSSAIAvailability, which we have chosen to omit because of our approach to support dynamic slice creation rather than static slices. While 3GPP TS 29.531 [10] assumes that the AMF indicates their supported slices to the NSSF, which provides a repository of running slices, we take the approach that the NSSF interworks with the slice orchestrator to determine the available slices. This gives more control to the slice orchestrator, which enables dynamic slice deployment. Ericsson's prototype implementation extends slice selection with the following features:

- Slice discovery and description: Describing the slice in more details to the UE in order to guide slice selection of the UE.
- Dynamic slice deployment: Deploying slices on-demand.
- Slice extension to edge: Extending slices on-demand to the edge nodes only where the UEs are located.

As currently defined in the 3GPP standards, the UE has no further information about the slices than the S-NSSAI. Thus, it relies on preconfigured service types. This is adequate for a limited number of slices that do not frequently change. However, as proposed by 5G!Pagoda, the number of slices may be high and a UE can be connected to multiple slices, e.g. for each application. Applications need to be able to identify and connect to the correct slice. In the case of application specific slices, e.g. in the case the service provider provides its own application (e.g. an application for a video

streaming service), the S-NSSAI can be hardcoded in the application. For more generic slices, a way for selecting the slice based on the identified properties of slices is needed. We have extended the reply to the Nnssf_NSSAIAvailability service with additional information to each of the Allowed NSSAI. As the GBA is based on JSON, the natural extension is a JSON object added to the Allowed NSSAI entry. The slice description object maps keys to values, where keys include properties such as bandwidth, max duty cycle, cost, priority, supported protocols and protocol ports, etc.

In our prototype, we have implemented dynamic creation of slices when a slice is requested that is not instantiated. It comes in two versions:

- When the AMF requests a slice (S-NSSAI), on behalf of a UE, which is not deployed at all in the network, this slice is deployed on demand.
- When the AMF requests a slice (S-NSSAI), on behalf of a UE, which is deployed in the network, but not in the deployed in the current area (as identified by the TAI) of the UE, a new slice instance is deployed on demand in the correct area.

During the deployment, the UE is allocated to a temporary slice, which may be a slice of the requested type but provisioned in a different area (or in the central cloud), or a more generic type of slice.

For on-demand deployment, the NSSF interfaces with the slice orchestrator. In our current prototype implementation, both are implemented in the same software package. The NSSF requests the slice orchestrator to deploy a new slice instance in the area of the UE.

Because the deployment time of a slice might be fairly long (in the order of minutes), we have introduced an optimization in the process. Instead of instantiating the slice when the PDU session is created, i.e. when it is needed, we can deploy a slice speculatively. This is done in the registration phase. In our implementation, when the AMF registers a UE and requests the Allowed NSSAI, it also provides the Subscribed NSSAI and Requested NSSAI to the NSSF. The NSSF can determine, which slices the UE is able to connect to, and, in advance, can deploy these slices in the area of the UE. This speeds up the process so that the slice may be available, or at least quicker available, when the PDU session to the slice is set up. The disadvantage is that some slice might be deployed, to which the UE never attaches.

Once the dynamically deployed slice is available, the UEs that have been allocated to temporary slices are transferred to the intended slice. For this, the AMF must keep track of UEs that have been allocated to temporary slices.

In order to signal the state of the dynamic deployment to the AMF, we have extended the reply to the two queries. The added fields indicate whether a new slice is being deployed dynamically, that a temporary slice has been assigned. Moreover, information about the deployed area can be provided, which can be utilized later in predictive deployment based on estimated user mobility.

We are currently working on network assisted slice selection, in which the network can select or modify the slice of a UE. This will be reported in D3.3.

As currently defined in 3GPP, the NSSF does not receive the identity of the UE when performing the slice selection. It only relies on the Subscribed NSSAI and the Requested NSSAI. We extend the

request with the UE identity, which allows the NSSF to perform selection based on the policy for a certain UE. Moreover, the UE identity is necessary to guarantee UE may provide set of parameters in the form of NSSAI in order to provide consistent replies to queries, in case a given UE has been transferred to another slice. The prototyping of this feature is currently ongoing work.

3.5. Implementation in the Open5GCore

In this section, we present the implementation carried out within Open5GCore to optimize its configurability in order to customize and to deploy slice networks according to the requirements of the subscribers, as described in Section 3.3.

3.5.1. Open5GCore Framework

Open5GCore is a 3GPP aligned implementation of the 5G mobile core network. It is developed on top of FOKUS' C/C++ framework for multi-threaded and highly distributed services, supporting x86, x64 and ARM based Ubuntu Linux systems. To enable highly diverse deployments of software components, the framework provides the following functionalities:

- Pool based memory management
- POSIX compatible thread management
- Worker pools
- Logging
- An XML based configuration system
- Integration into Google testing framework
- Flexible modularization layer.

Furthermore, the framework provides utility functions for networking, encryption and common data structures. The modularization layer allows the parallel implementation, testing, and deployment of different services. It is based on separating the functionality at compile time into different binary files. The binary files can then be loaded during framework initialization, as needed, to provide the required services.

3.5.2. Network Function Development

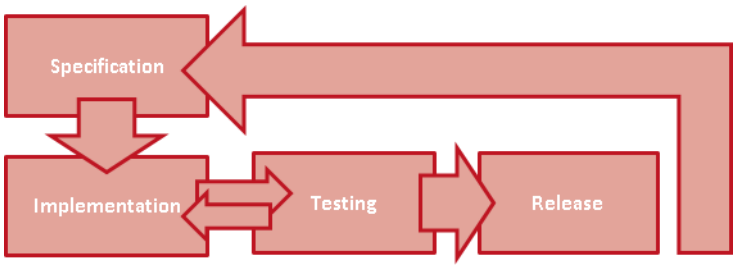

Figure 3-9 – Open5GCore Network Function Implementation: Overall Process

Figure 3-10 - Open5GCore Network Function Specification

Virtual NFs can be developed as services of this framework. The development of NFs follows standard procedures. As presented in Figure 3-9, the overall implementation process is comprised of the following:

1. Specification
2. Implementation
3. Testing
4. Release

The specification process is depicted in more detail in Figure 3-10. To implement a NF, first its functionality is identified. Following the principle of Divide & Conquer, the functionality is divided into groups of related functions. For example, if a NF requires a specific communication protocol, all the functions needed for the protocol would be grouped together. After the functional analysis, the modules and interfaces to be implemented are defined. Usually, a module would comprise a functional group, and the functions it provides determine the interface. Then, the test cases and environments are specified to cover possible usage scenarios and ensure the completeness of features.

Following the specification, the implementation and testing are conducted. These processes are tightly intertwined, as depicted in Figure 3-11. The NF is implemented as a set of reusable, standalone compilation units called modules. The implementation of the modules can often happen in parallel. Each module defines an API of the functionality it provides, for other modules to access. The modules can be loaded during the initialization process of the framework. A module can be used by more than one NF. Two modules implementing the same API can be used interchangeably. A set of unit tests may be implemented to ensure that the desired functionality is achieved. The prototypical network function is deployed in a test environment to undergo integration testing. If integration tests are completed successfully, the NF can be compiled and deployed for acceptance testing. As is the norm in agile software development, it is possible to deviate from the process, if the need arises. After the NF passes acceptance testing, it is ready for release.

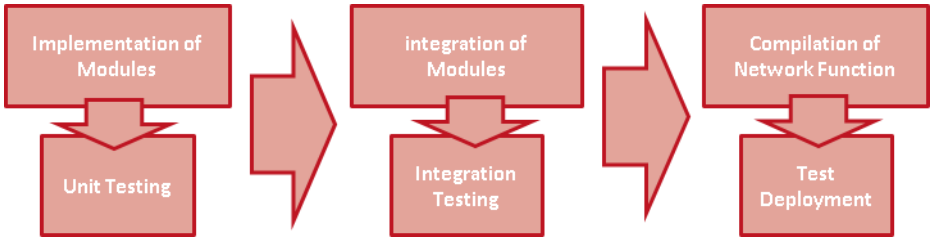

Figure 3-11 – Open5GCore Network Function Implementation and Testing

3.5.3. Small Messages

To support small messages slices on Open5GCore, the Non-IP Data Delivery (NIDD) feature was implemented.

NIDD was first introduced as part of the Cellular-IoT EPS Optimizations [14]. With 3GPP Release 14 and the standardization of the Narrow-Band IoT (NB-IoT) extensions [15], a step was taken towards supporting massive numbers of devices transmitting only small amounts of information. As shown in Figure 3-12, the NIDD architecture provides efficient support for the sporadic transmission of small amounts of data by minimizing the network signalling overhead. With NIDD, the data bearer assignment is no longer needed, since the user data payload is encapsulated into NAS signalling messages. As a consequence, user data is using the packet core messaging service and not a data path in the strict sense.

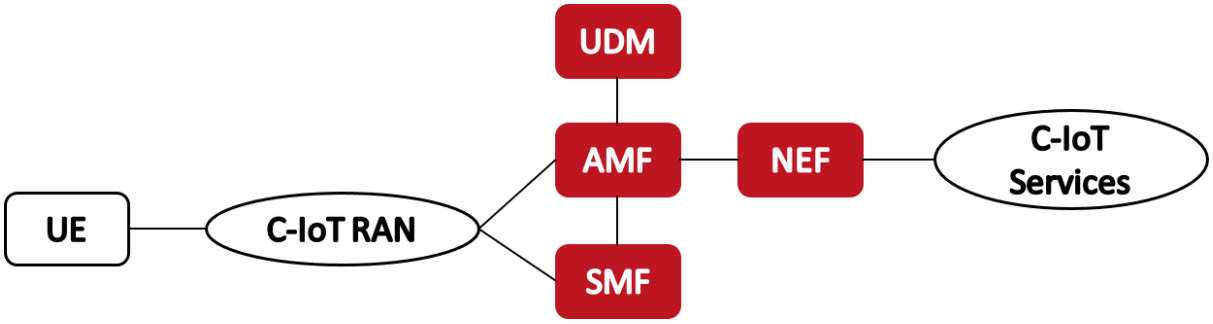

Figure 3-12 - Small Messages Slice Architecture

The Network Exposure Function (NEF) was developed to support the NIDD architecture. Through the NEF, the device data is sent over the control plane, establishing a connection to the C-IoT services. The NEF implementation covers the following functionalities:

- T6a/T6b Connection Establishment [14]
- NIDD configuration [14]
- Mobile Terminated NIDD procedure [14]
- Mobile Originated NIDD procedure [14]
- Network attachment (NB-IoT)

3.5.4. Distributed Data Path

In the core network operating multiple UPFs, a dynamic bearer path selection/modification mechanism was implemented. When a UE session is created (CreateSessionRequest), a set of UPFs is selected based on metrics such as the appropriate up-link and down-link paths. The packets from UEs are encapsulated by the gNodeB using GTP. To route the packets along the path, TEIDs are used to identify the next hop. Then, they are allocated by the core network, with the exception of the downlink TEID allocated by the gNodeB. After receiving the downlink TEID from the gNodeB (ModifyBearerRequest), the path is allocated. The path can be altered in the core network, as long as the first UPF after the gNodeB remains.

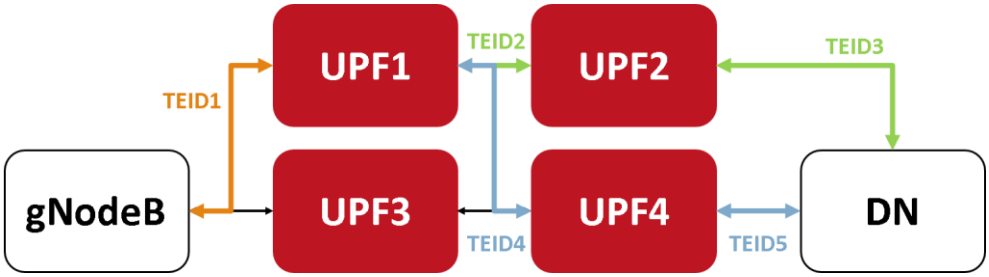

Figure 3-13 - Different bearer paths to the same DN

Figure 3-13 depicts a core network example with multiple paths available between a gNodeB and a DN. The TEIDs for two different paths are highlighted; one path passing through UPFs 1 and 2, and another path passing through UPFs 1 and 4. It is possible to create paths between multiple gNodeBs and DNs through any network of UPFs. This way, different path requirements can be fulfilled based on the given network topology. More possibilities can be inferred from Figure 3-7. Internally, the network topology is specified as an XML document describing a graph with weighted links. The link elements contain information on the IP address and type of connected nodes. The path can be modified dynamically through a command line interface.

3.5.5. Edge-Central Split

In the slices of the 5G!Pagoda system, core network functions can be distributed between edge and central nodes. The split of the core network between edge and central nodes can vary depending on the slices and slice types. The choice of NF distribution has to take into account both the use case requirements and system parameters. The availability of resources at the edge and the load on both edge and central nodes can affect this decision. Furthermore, to ensure the correct operation of all NFs, it is important to carefully pay attention to their respective synchronisation requirements.

With this in mind, let us consider the various slice configurations presented in Section 3.3.4.

3.5.5.1 Local Offload

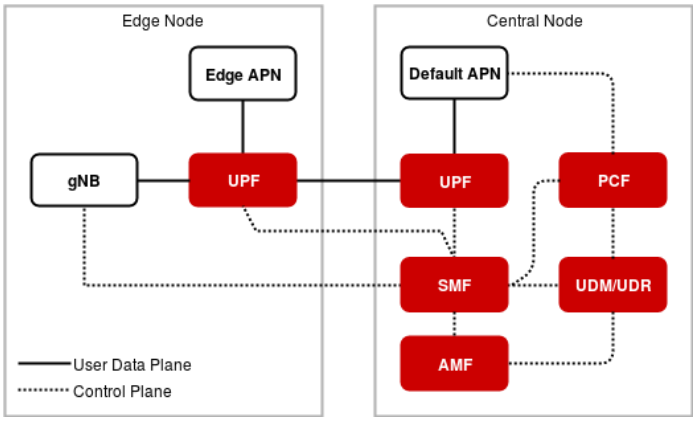

Figure 3-14 - Local Offload

In this slice, the data plane traffic for the slice-specific use cases can be offloaded to the edge. As seen in Figure 3-14, the local offload slice involves running a gNB, a UPF, and an application server

on the edge. Control channels for the gNB and UPF have to be routed between edge and central along with the user data plane when the UE requests the default APN.

3.5.5.2 Local Mobility

In addition to running data plane traffic at the edge, the local mobility slice, shown in Figure 3-15, also runs the AMF on the edge node. This requires the control plane traffic, between the AMF and SMF as well as between the AMF and UDM/UDR, to be routed from edge to central and back. The UDM information could be cached to further improve latency during attachment and other authorization operations.

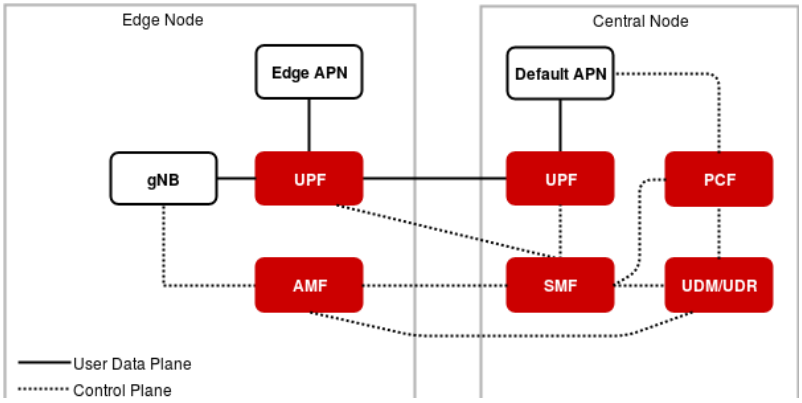

Figure 3-15 - Local Mobility

3.5.5.3 Local Session

The local session slice is presented in Figure 3-16. It contains an additional SMF along with the AMF at the edge. The connections between both SMFs and UPFs have to be routed between edge and central nodes. In the current implementation, both SMF instances maintain control over both UPF instances. However, this redundant control should be phased out as the synchronization is refined. In this case, caching UDM data could be even more significant.

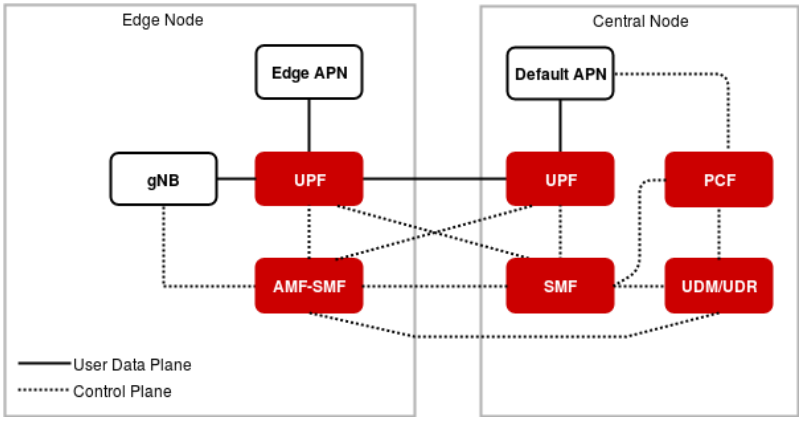

Figure 3-16 - Local Session

3.5.5.4 Local Control

Next, the local control slice shown in Figure 3-17, positions UPF, AMF, SMF and PCF instances at the edge. In this situation, the communication with the PCF is local to the edge node, but the PCF

itself has to contact the UDM/UDR located in the central node. As with the previous examples, the SMF and UPF communication crosses also between edge and central nodes.

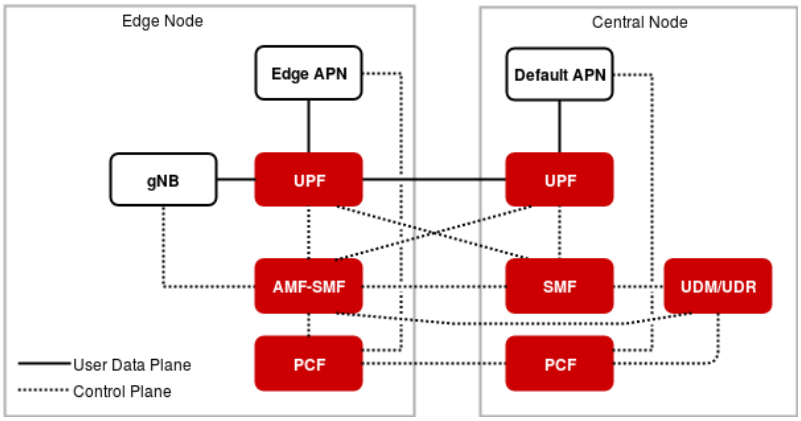

Figure 3-17 - Local Control

3.5.5.5 Autonomous Edge

Figure 3-18 presents the autonomous edge slice, which involves running instances of all core network functions at the edge. In this case, the state between the edge and central UDR instances has to be synchronized. However, this can be avoided by only deploying the UDM at the edge, as was suggested earlier.

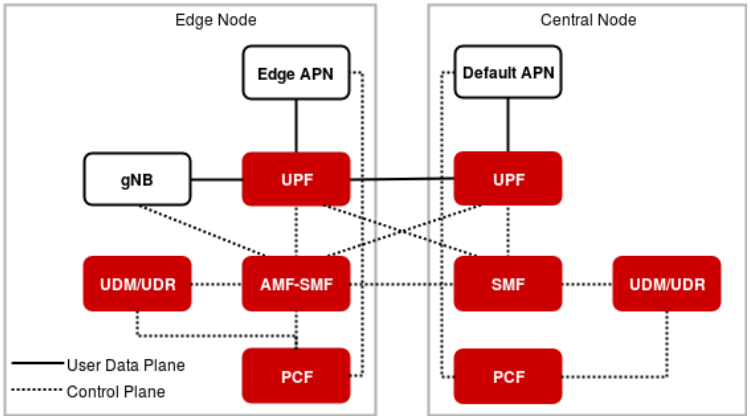

Figure 3-18 - Autonomous Edge

3.5.6. Open5GCore Network Function Configurations

The Open5GCore network functions provide numerous configuration parameters to enable and disable features and to alter their behaviour. For each NF, these parameters are defined in an XML file. A NF parses and interprets its configuration during initialisation.

The configurable features of a NF include core network aspects, employed communication protocols, database connections and implementation-specific features. The following tables list some of the more prominent features for the main network functions, such as MME, HSS/UDM and OpenFlow Switch (OFS).

Feature	Description
Mobility Management	Procedures and information for the management of UE mobility
Session Management	Procedures and information for the management of UE sessions
AAA	Features related to AAA, including the HSS/UDM
Database Connectivity	Connectivity to databases such as MySQL and Redis
Monitoring	Real time monitoring and supervision of network functions
NAS	Implementation of the NAS protocol
GTP	Implementation of the GPRS Tunnelling Protocol
Diameter	Implementation of the Diameter protocol and several Diameter applications
OpenFlow	Implementation of the OpenFlow SDN protocol

Table 2: MME configurational features

Feature	Description
AAA	Features related to AAA, including the HSS/UDM
Database Connectivity	Connectivity to databases such as MySQL and Redis
Monitoring	Real time monitoring and supervision of network functions
Diameter	Implementation of the Diameter protocol and several Diameter applications

Table 3: HSS configurational features

Feature	Description
OpenFlow	Implementation of the OpenFlow SDN protocol
Packet switching and forwarding	Implementation of mechanisms for fast and efficient handling of packets
Monitoring	Real time monitoring and supervision of network functions

Table 4: OFS configurational features

4. Programmable Data Plane specification

4.1. Introduction

In this section, the programmable data plane specification is presented. This includes the optimization of the data path within the software base station implementation as well as the upgrading of the core network to support the functionality of data path diversity as presented in Section 2.3.3.

4.2. Flare-related advancements

4.2.1. Real-time optimization in eNB

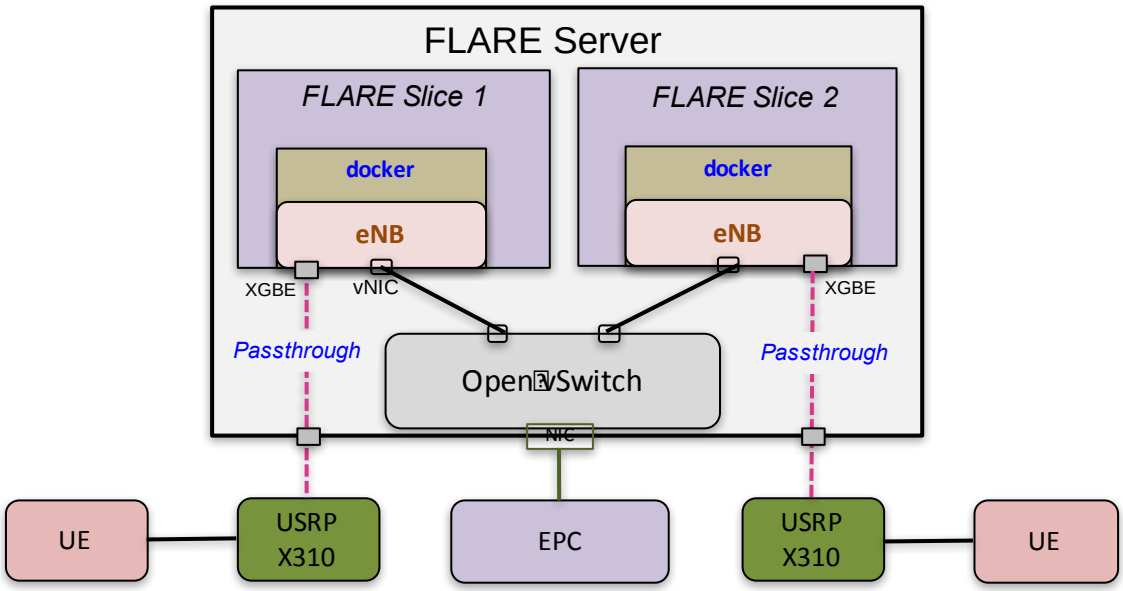

Figure 4-1 – Slicing eNB on FLARE server.

As shown in Figure 4-1, we run an eNB instance in a Docker instance inside a FLARE server Each eNB in a Docker instance is isolated and replaceable within a FLARE server.

In our experimental LTE network, UEs connect to an eNB instance via software radio platform USRP X310, which connects to an eNB via XGBE pass-through technology. The traffic flows of different slices are isolated using VLANs on the FLARE node running Open vSwitch. The packets from different eNBs are classified and tagged with different VLAN IDs and then diverted to an EPC instance.

The PHY layer of eNB is responsible for receiving packetized signal frames from USRP X310, performs symbol level processing and delivers sub-frames to MAC layer for further processing. Since each transmitted sub-frame has to be acknowledged back at the USRP X310 to allow for retransmission, there is an upper bound for the total delay for each sub-frame processing time. The eNB RAN functions must finish processing before the deadline.

To provide frame/sub-frame timings successfully, the execution environment of the software eNB has to support strict deadlines of the code execution. OAI community has the following real-time optimization issues for efficient signal processing over General Purpose Processors (GPPs), rather than the traditional implementation over Application-Specific Integrated Circuits (ASICs).

- First, State-of-art CPU with vector processing instructions (AVX2) is used to optimize the turbo decoding and FFT processing.
- Second, disabling CPU automatic under-clocking function so that the processing time will not be increased to save the power.
- Third, using low-latency (or real-time) kernel provided Ubuntu to reduce the pre-emption delay in Linux scheduling.

During our experiment, we found the critical real-time issue in the above OAI's ingredients unresolved due to the inefficient and unnecessary actions. For example, we observed some sub-frames cannot be processed within the deadline due to risk of the sub-frame processing thread pre-emption by another thread even in a low-latency kernel. Moreover, Intel CPUs are a dominant cost component in a FLARE node and are rarely negotiable.

To addresses the above issues, we did the following extra optimization.

- First, we use AMD CPUs with high frequency due to the cost of AMD CPU is only about 30% of that of the same-level Intel CPU.
- Second, we assign dedicated data-plane CPU cores to sub-frame decoding/encoding threads to reduce processing delay by reducing context-switching

Moreover, we isolate the Tx/Rx antennas using separate feeders to reduce signal interference of each other and improve signal channel quality.

4.2.2. Preliminary experimental results

To avoid interfering with other mobile networks (AU, Softbank, and DoCoMo) in Japan, we deployed a Flare LTE network in Band 3 (1.7 GHz). Figure 4-2 shows the throughput measured with SpeedTest Application running on a Fujitsu's Arrows M04 Android phone.

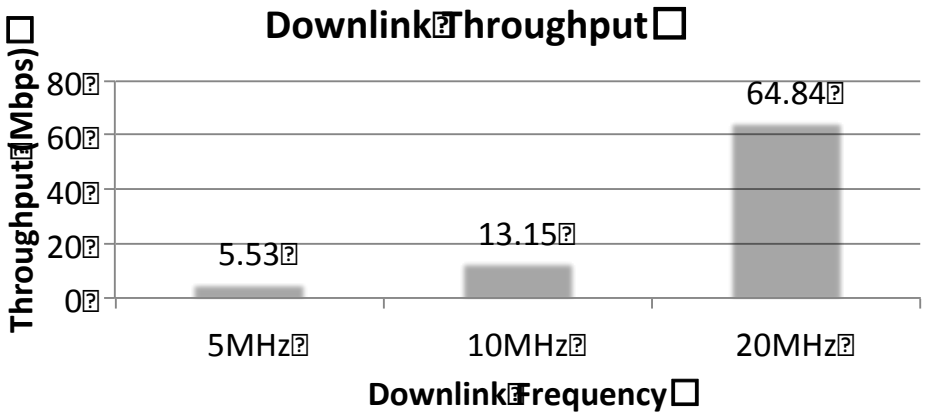

Figure 4-2 – Downlink throughput on 1.7 GHz

4.3. Control Plane interactions

In this section, the interactions between the data plane and the control plane are described in details. This includes the developments of the distributed data path slice as considered within the Section 2.3.3

4.3.1. Programmable Data Plane introduction

The solution presented in this section adds multiple UPFs to the 5G Packet Core implementation, keeping the rest of the control plane un-modified. Instead of allocating the bearer path through one S-GW and P-GW as it was in the case of LTE, the path should be allocated through the network of UPFs. Figure 4-3 shows the data plane part of the network with 4 UPFs introduced. The control plane part is presented in Figure 4-4.

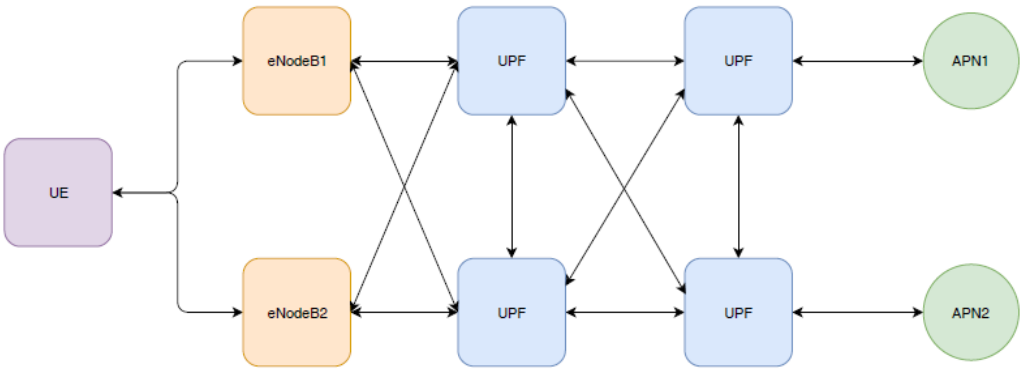

Figure 4-3 – Distributed data plane with multiple UPFs

The bearer path is allocated during Initial Attachment and PDN Connectivity Request. In LTE, first the link between P-GW and S-GW is established, then the link between S-GW and eNodeB is established. In 5G, S-GW and P-GW are replaced by one component, UPF. The allocation of the path should be done in a similar fashion. First, the links between consecutive UPFs should be chosen and then the last link between eNodeB and the first UPF from the path should be established. Looking at it from the LTE perspective, this first UPF is treated as S-GW and the standard control message flow is extended. After the path selection, eNodeB knows which UPF to route the traffic to (as in LTE), this UP routes it to another, and so on until the edge of the core network.

To conform to the Session and Service Continuity (SSC) requirements, there should be also a possibility to reallocate the path. This should be done by the controller. The path modification in LTE environment is problematic, because eNodeB has to be informed about the changes of the first UPF on the path. There is no control messaging that could be used for that case. For that reason, the first link should be maintained and the path reallocation should influence only the rest links. It means all UPFs but the first one can be changed.

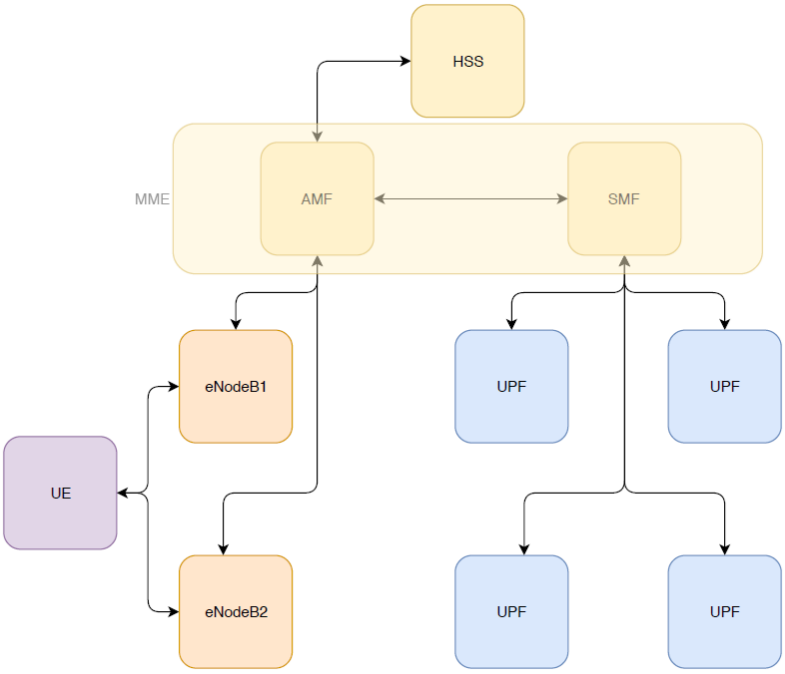

Figure 4-4 – Control Plane Interactions of the distributed data plane

4.3.2. Specification of the path selection through multiple UPFs

There are several UPFs added to the network and the Create Session Request message triggers the selection of the UPFs used to build the data path. The path consists of several nodes and TEIDs (uplink and downlink) allocated for every link between them. The user data traffic that goes from eNodeB to S-GW and then to P-GW (or from gNodeB to UPF and then to another UPF in 5G) is encapsulated in GTP packets. To route GTP packets, TEIDs are used. The core network allocates all TEIDs except the downlink TEID allocated by the eNodeB and exchanged with the network at the end of the procedure. This does not change in 5G standards (3GPP TS 23.502 section 4.3.2 [7]). After receiving eNodeB's missing downlink TEID in Modify Bearer Request message, the path is successfully allocated.

While modifying the bearer path from the core network's side, the link between eNodeB and the first UPF have to be maintained. The same TEID pair, for uplink and downlink, has to be kept the same after the modification. This is caused by the aforementioned problem with the need to inform eNodeB about this change, which is not supported in LTE. By changing the path on the core network's side, one has to take into account that eNodeB will still direct the traffic to the first UPF, which data were delivered to eNodeB during Initial Attachment. If all links, but the first one, are changed, eNodeB does not care about it, while sending the data packets to the same UPF, which then routes the traffic along the modified path. To modify also the first link, the corresponding message to eNodeB would have to be sent, in which the data of a new UPF would have to be delivered. There are no messages defined in LTE standard, by which eNodeB could be informed about the changes in the data path, that are triggered by the core network alone, thus the first link should stay unchanged during this procedure. However, all other links can be reallocated, modifying the path that leads through different UPFs to the given APN. The choice is made

according to the network topology, where there should be a few possibilities to reach given APN. This is presented in Figure 4-5, where there are two possible paths to the same APN.

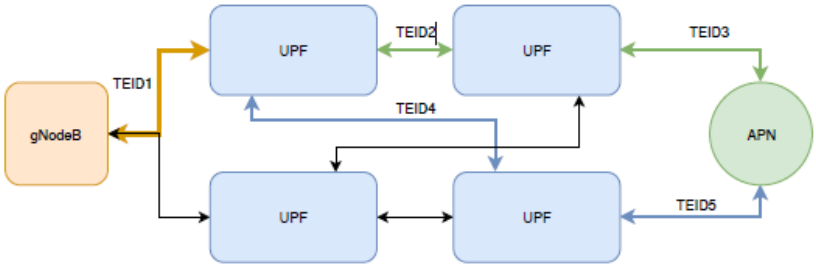

Figure 4-5 – Two data paths example (orange-green/orange-blue) for the same APN

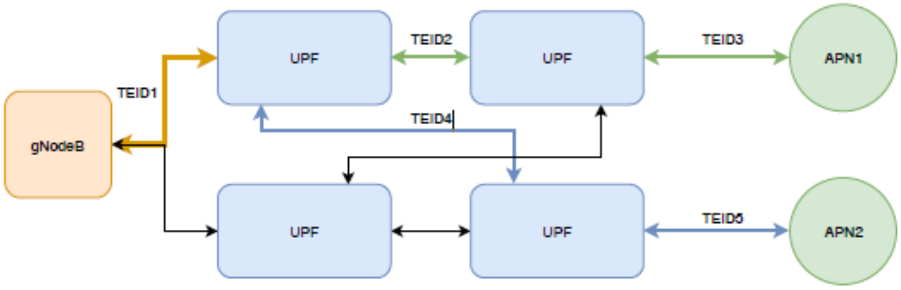

Figure 4-6 – Two data paths example (orange-green/orange-blue) for different APNs

In the situation from Figure 4-5, the first path leads through two UPFS at the top and then could be changed keeping the left-top UPF and allocating the new, bottom-right to modify the path. Every link has to have its own, unique set of identifiers (TEIDs), to route the traffic through the network. Another situation is depicted in Figure 4-6, where there are two paths established to two different APNs, keeping the same choice of UPFs as in the previous single APN situation. The given cases are just examples how the path could be allocated. Any other combination of UPFs could be selected.

4.3.3. Implementation considerations

4.3.3.1 Data Path Selection

Implementation of path selection moves Open5GCore closer to full 5G environment. The standard setup consists of the one DP-SWITCH. 5G, on the other hand, predicts more switches in the network, called UPFs. For the purpose of this solution, more DP-SWITCHES were introduced to Open5GCore and creation of EPC Bearer was modified to use the static topology as the help to create the data path that goes through multiple switches.

In LTE, S-GW is selected after Location Update Request and before Create Session Request and the creation of EPC Bearer is finalized after eNodeB's downlink TEID is delivered in Modify Bearer Request. In Open5GCore, the switch is selected in PGW-C, which is the controller for the DP-SWITCH. However, adding more switches to the network requires modifying the function that allocates the switches in few steps. First, every node of the data bearer should allocate its own unique uplink and downlink TEID and also have the information about the TEIDs of the neighbour.

The standard LTE TEID exchange procedure is kept, however only between eNodeB and the first DP-SWITCH on the path. The uplink TEID of the first switch in the chain has to be delivered to eNodeB and the allocated downlink path has to be updated by the TEID allocated by eNodeB. Figure 4-7 depicts the uplink and the downlink exemplary path with random TEIDs.

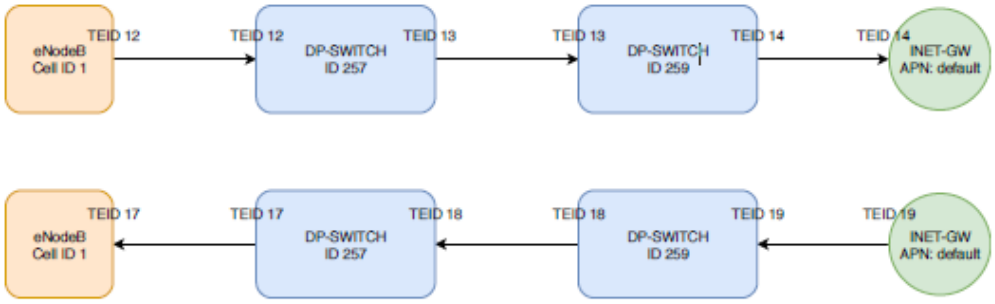

Figure 4-7 – Exemplary path, uplink part (top) and downlink (bottom)

The path selection is done after Create Session Request is received by PGW-C. The paths are created by calling corresponding functions and the result is two paths, one for the uplink and one for the downlink as shown in Figure 5.3. The path is created by taking two arguments, the first node and the last node, eNodeB and APN (or APN and eNodeB when talking about the downlink). As was already mentioned the nodes are identified by ID. PGW-C has the information about APN, however it does not have Cell ID, which is needed to identify eNodeB. This problem was solved by adding non-standard field in the GTP-C message (Create Session Request). MME knowing Cell ID passes it to SGW-C in the Create Session Request, which in turn is included in the same message sent to PGW-C (please note that even though MME, SGW-C and PGW-C are the same component, there exists standard LTE message exchange between them, they are logically separated).

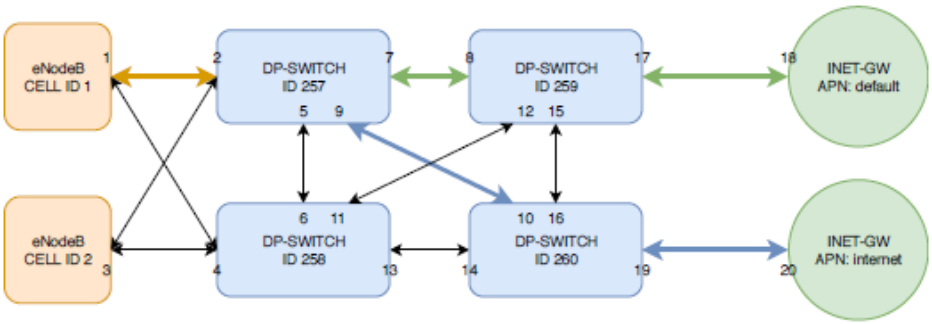

Figure 4-8 – Graphical representation of the topology with interface numbering

4.3.3.2 Data Path Modification

Both paths, for the uplink and the downlink, are placed in the bearer structure that is identified by UE's IMSI and EPS Bearer Identity (EBI). UE can have multiple APN connections, which differ by EBI. Two different UEs may have the bearers with the same ID, that is why the proper identification is done through these two identifiers. Path Modification is implemented in PGW-C component and can be triggered by the command line function. The function takes IMSI and EBI as the arguments to find the corresponding path. Apart from those two parameters, the IDs of the switches have to

be given, through which the new path should go. The path reallocation is done in a way Access Network is not informed about the change. Routing table in eNodeB is created after it gets the S-GW TEID (and also IP address) during the attachment and it is not modified during the path modification done only in P-GWC. Because of that, the first switch in the path has to stay the same. The example of the path change can be seen in Figure 4-9. The initial path is selected during the attachment to APN (internet, in this case), it goes through the switches with ID 257 and 260. After calling the command line function with 5 parameters "`pgw.modify.bearer 'IMSI' 'EBI' 257 258 259`", the path is changed so it goes through the switch with ID 257, 258 and 260. The completely new path is being allocated and the first list is copied from the old one so TEIDs and eNodeB parameters stay unchanged.

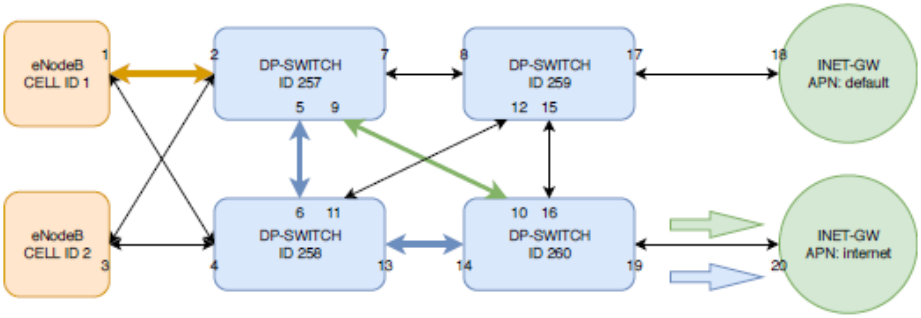

Figure 4-9 – Example of data path modification from orange-green to orange-blue

4.3.4. Evaluation of Open5GCore implementation

The evaluation of the dynamic path selection and modification was done according to the 5 test cases, presented in the next sections, which simulate the situations expected in the public mobile network. In the evaluation, two APNs are used, default and internet provide the access to the Internet regardless of the name. Every APN has different IP addresses pool, from which the IP for UE is chosen. After connecting to both APNs, the UE has a possibility to connect to the Internet using two different source IP addresses.

Currently, only functional tests were executed, to be followed in the next stage (and deliverable) by some basic performance tests.

Test Case 1: Multiple PDN Connections Simultaneously

The first case tests the possibility to connect with multiple APNs with the selection of different paths to each one, through the network of UPFs. Every new PDN Connection is established after MME receives new PDN Connectivity Request.

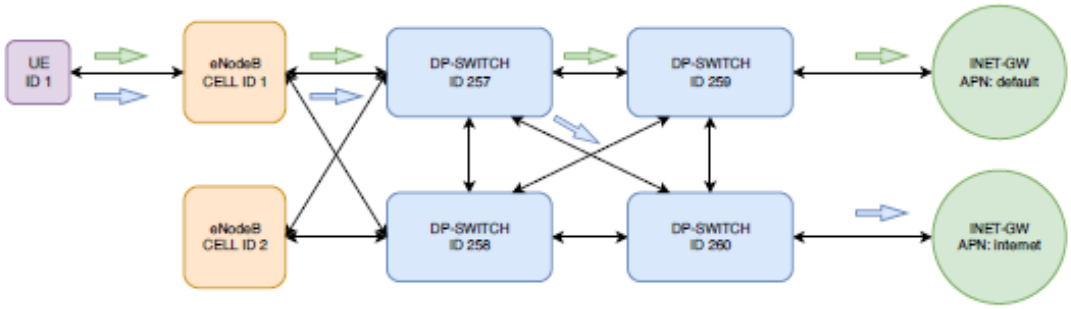

Figure 4-10 – Test Case 1

ASSUMPTION	It should be possible to have multiple PDN Connections to different APNs leading through different UPFs in the network. The paths should be independent of each other, even if they share same UPFs.
TEST PERFORMED	Two PDN Connections were initiated for one UE. Connectivity check with both was done, also simultaneously.
TEST PROCEDURE	UE was connected to the network through Initial Attachment procedure. This caused the opening of the PDN Connection to default APN. After successful attachment, PDN Connectivity Request was triggered to the second, internet APN. After that the connection with the Internet through two different paths created was checked.
RESULTS	After establishing two PDN Connections, UE had two IP addresses allocated. It was possible to reach the Internet using both IP addresses, even simultaneously. The paths selected through different set of UPFs did not interfere with each other.

Test Case 2: Many UEs Connected Simultaneously

In the second case, the ability to allocate the same path for several different UEs was tested. The traffic originating from and coming back to two users is routed through the same set of UPFs, therefore it is important that no data is lost and there is no interference between them.

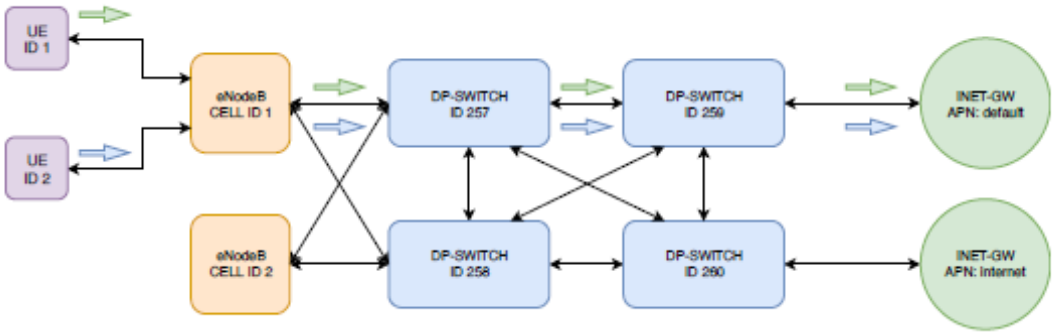

Figure 4-11 – Test Case 2

ASSUMPTION	It should be possible to have several UEs attached to the same PDN. Every UE has the same data path allocated and they should not interfere with each other.
TEST PERFORMED	Two UEs were attached to the network and to the same PDN. Connectivity check with the Internet was done simultaneously for all UEs.
TEST PROCEDURE	Each UE was connected to the network through Initial Attachment procedure. Every UE connected to default APN selecting the same path to it.
RESULTS	All UEs could connect with the Internet using the same path through UPFs' network at the same time.

Test Case 3: Dynamic change of UPF in PDN Connection

During this test, the ability to change the paths dynamically was checked. This is the main purpose of the bearer path modification function implemented for the purpose of this thesis. After the connection with PDNs is established, it should be possible to modify the path on the core network's side, without informing the access network about this fact. For this reason, the first link, as described in more detail in the previous sections, stays unchanged.

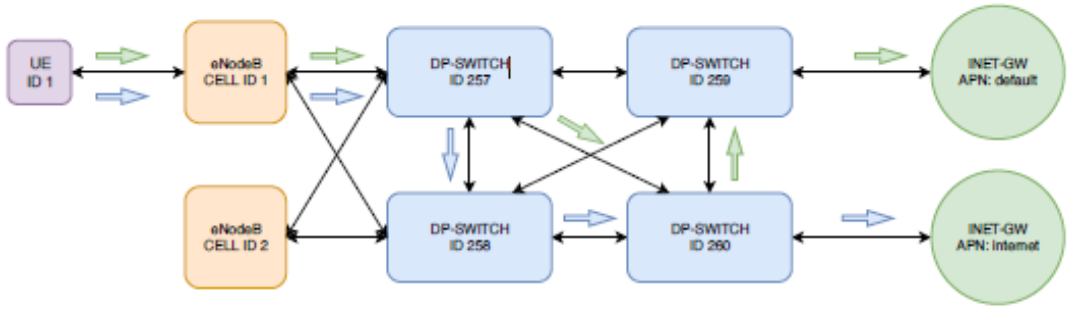

Figure 4-12 – Test Case 3

ASSUMPTION	It should be possible to reallocate existing paths to PDNs on the core network's side. UE initially has multiple PDN Connections as in Test Case 1. The network should be able to dynamically change paths, allocating different set of UPFs for every path.
TEST PERFORMED	Two PDN Connections were initiated for one UE. Then the bearer path modification function was triggered changing the path to each PDN.
TEST PROCEDURE	UE was connected to the network through Initial Attachment procedure. This caused the opening of the PDN Connection to default APN. After successful attachment, PDN Connectivity Request was triggered to the second internet APN. Then the path to every PDN was modified by calling

	the PGW-C command line function. The connection with the Internet was constantly monitored.
RESULTS	The path reallocation was successfully performed, and the connection with the Internet was maintained. No connection break was observed.

Test Case 4: Handover and multiple UPFs

The path selected has two end sides, eNodeB and APN. During Handover the first node, eNodeB, is changed and that has to be also reflected in the core network, as the controller has to update the allocated path. Contrary to the previous cases, in this one the first link on the data path changes, which is supported by LTE standard and it is Handover procedure.

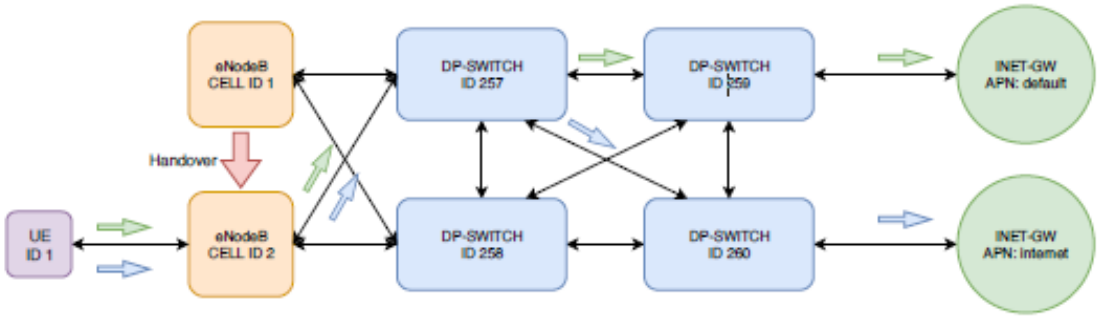

Figure 4-13 – Test Case 4

ASSUMPTION	It should be possible to keep the same selected paths through multiple switches also during the handover between two eNodeBs. This corresponds to the handover in LTE where S-GW and MME are not changed.
TEST PERFORMED	Two PDN Connections were initiated for one UE as in Test Case 1. After that S1 Handover was performed between two eNodeBs and the connectivity check was done.
TEST PROCEDURE	UE was connected to the network through Initial Attachment procedure. This caused the opening of the PDN Connection to default APN. After successful attachment, PDN Connectivity Request was triggered to the second, internet APN. S1 Handover was triggered from the UE.
RESULTS	After the successful handover, UE was able to connect to the Internet using the same paths through UPFs, sending the packets only through the target eNodeB.

Test Case 5: UE-to-UE Communication

In this case, the ability to send the data traffic between two users was tested. Both UEs can be connected to the core network through different eNodeBs. However, the communication between

them should exist only if they are using the same service, which means are connected to the same PDN.

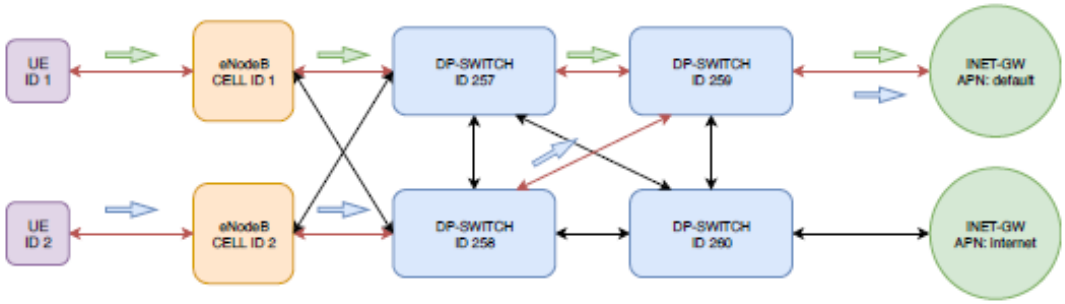

Figure 4-14 – Test Case 5

ASSUMPTION	Two UEs connected to the same PDN, but with different paths selected should be able to reach each other through the established paths.
TEST PERFORMED	Two UEs were attached to the network, to the same PDN. Both selected paths had the same UPF, acting as the Anchor Point, but other allocated UPFs were different. Connection between UEs was checked.
TEST PROCEDURE	Both UEs were connected to the network through Initial Attachment procedure. This caused the opening of the PDN Connection to default APN in both cases. Each UE was connected through different eNodeB causing the different path selection.
RESULTS	UE-to-UE communication was possible to achieve. The packets going from the one UE were forwarded to the INET-GW (APN default), where they were rerouted to the second UE through the path assigned to it.

4.4. ICN related Functions

This section describes the data plane related ICN functions. Among architectures classified as ICN, NDN is used for implementation. Therefore, the following describes functions dedicated to NDN.

4.4.1. ICN function concept

(Forwarding)

- Name-based forwarding function: a request is forwarded by attaching the name or ID of the content to the request. Particularly, a large content object is managed by a set of identifiable chunks with a recognizable name or ID of each chunk.
- Request aggregation function: ICN nodes are equipped with the function to aggregate requests to the same content objects to reduce network traffic and server load.

- Content return path indication: When the requested content is found and arrived to the node, it is forwarded to the path, from which the request arrived. Request aggregation and the return path indication function are realized by PIT (Pending Interest Table).
- Subscription: ICN provides a mechanism for a consumer to register a name to identify one or more content object, in which the consumer is interested.
- Publication: ICN provides a mechanism for a publisher to upload content objects to ICN for their content delivery and distribution.

(Routing)

- Content object registration function: A content object is registered with a unique name or ID in ICN so that the consumers can access to it.
- Content object availability function: availability information of content objects is disseminated to help choosing a right direction of request forwarding.
- Network selection function: appropriate network interfaces are selected to forward requests in order to reach a specified content object.

(In network Caching: Content Store)

- ICN nodes are equipped with content cache to reduce duplicated traffic of the same content objects.
- The node caches the contents, which went through the node, and when the request to the content cached in the node arrives, the content is provided by the content store of the node.
- There are a variety of algorithms to determine the caching contents and the replacement. In this implementation, the basic algorithm of the "cache everything and replace the LRU (Least Recently Used)" is used.

(Mobility support: Optional)

Basic ICN provides with user mobility support function using retransmission of Interest packet. In case of the 3N (Named Node Networking) architecture, which Waseda University developed as an extension of NDN, the following functions are provided.

- Consumer mobility: provide a mechanism for a consumer to maintain undisrupted content flows while the consumer moves.
- Content object mobility: provide a mechanism for content object to relocate without disrupting content availability when the content objects change their location.
- Network mobility: provides an efficient mechanism for a network to maintain seamless network connection for mobility.

(Security: Optional)

- Access control: ICN is equipped with a mechanism to examine and confirm the authenticity of consumers and that content object is accessible only by the authorized consumers.

- Network security function from malicious attack: ICN has a mechanism to protect its functions from malicious network attacks.
- Content object availability: ICN provides a mechanism to ensure that the content objects published in network are available for authorized consumers.
- Content authentication and integrity: ICN is equipped with a mechanism to examine and confirm the authenticity and integrity of content objects.

(Gateway functions)

- HTTP to ICN translation: This is an important factor for stitching ICN slice with CDN slice.
- Gateway access the CDN cache when needed and download the content.
- Gateway separates the content into a number of small segments (chunk) and assign the unique content name to each chunk.
- The content name is made from the original file name followed by the sequence number. The gateway ID is attached to the content name as a routable prefix part.

4.4.2. Mechanism and objective

To realize the total network system for the 5G!Pagoda project, we aim to build ICN slice using NDN Image (NDNx protocol) in Docker for Japan domain with FLARE implementation developed by Tokyo University. The potential ICN implementation in FLARE follows the network topology (via the collaboration with University of Tokyo) shown in Figure 4-15.

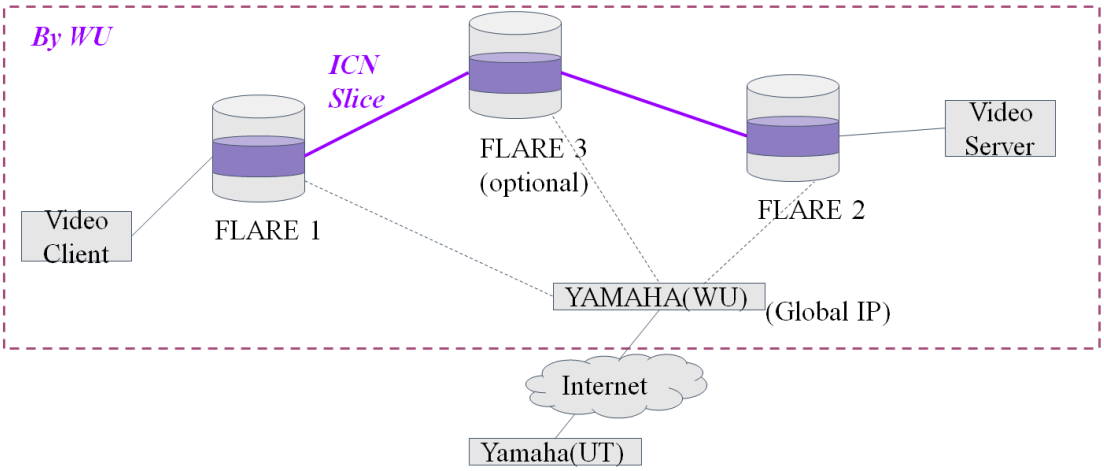

Figure 4-15 – ICN on FLARE network topology

4.5. Security slice functions

The security slice is realized by placing a security VNF based on the Data Plane programmable function on the slice, as showed in Figure 4-16. By judging whether or not to make communication, flow control realizes communication security by preventing communication not supposed by the administrator.

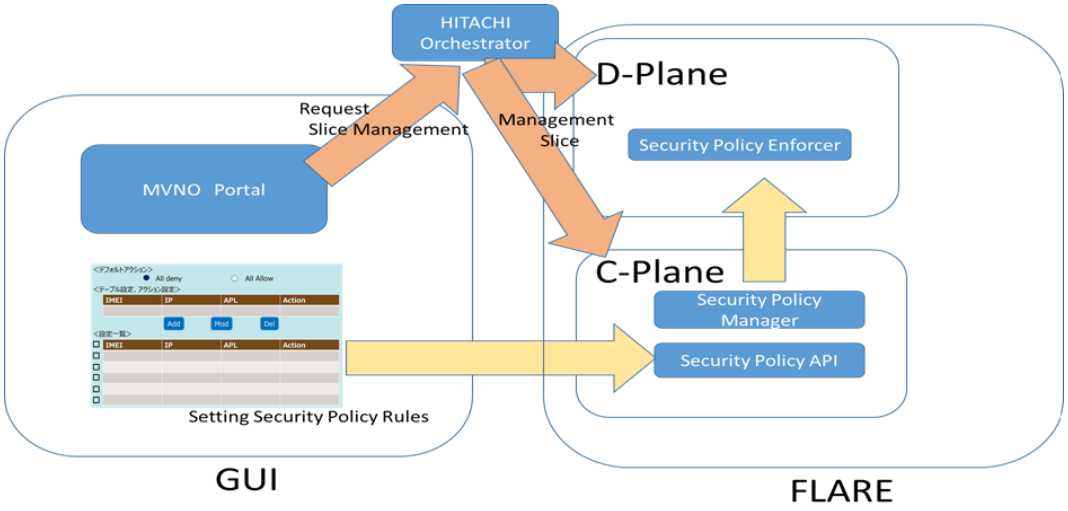

Figure 4-16 – Security slice functions

Because processing requires high Data Plane processing capability, it is appropriate to realize with a FLARE platform with a network core processor dedicated to Data Plane processing. The functions of the security slice are described as follows (see Figure 4-17 and Figure 4-1).

- A slice is deployed on FLARE platform. It contains software based S-GW, MME, HSS, and eNodeB. (Refer to D5.1, Figure 28 [11]).
- Some UEs with software, it works with the security VNF.

The security VNF is deployed on a slice. It is based of Deep Data Plane Programmability. It provides authentication function for terminals and applications.

- Hitachi's orchestrator is deployed.
- MVNO portal is deployed as an interface to control the APIs of Hitachi's orchestrators.
- The resources available from MVNO can be visualized.
- To be able to combine selection of visualized resources and resources owned by them and to form slices for each service.
- End-to-end communication quality (bandwidth, speed, QoS, etc.) can be set when forming a slice.
- These controls shall be realized by control through MVNO portal, orchestrator.

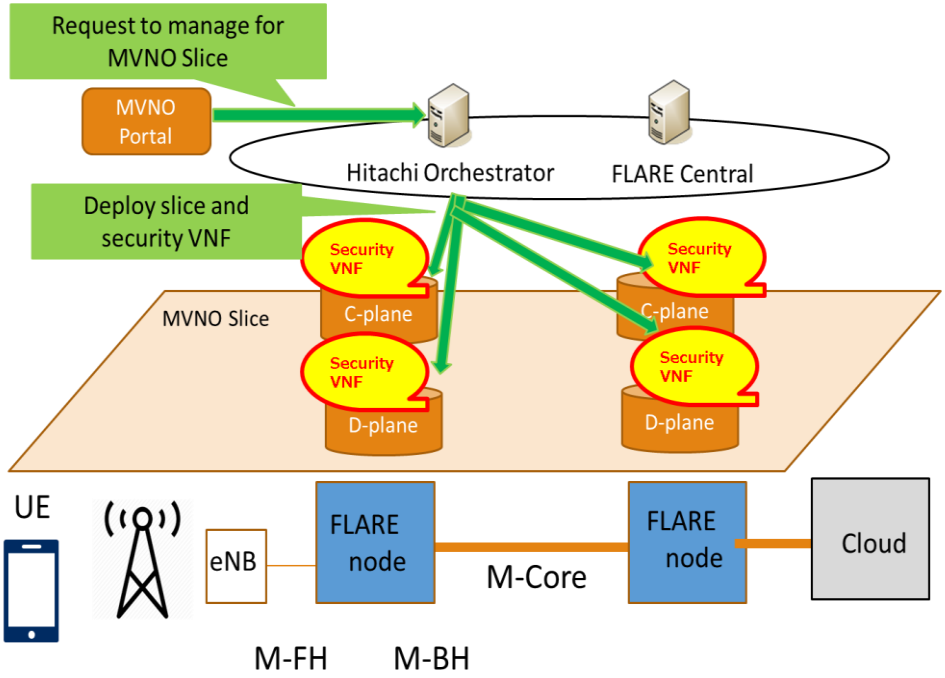

Figure 4-17 – Security slice architecture

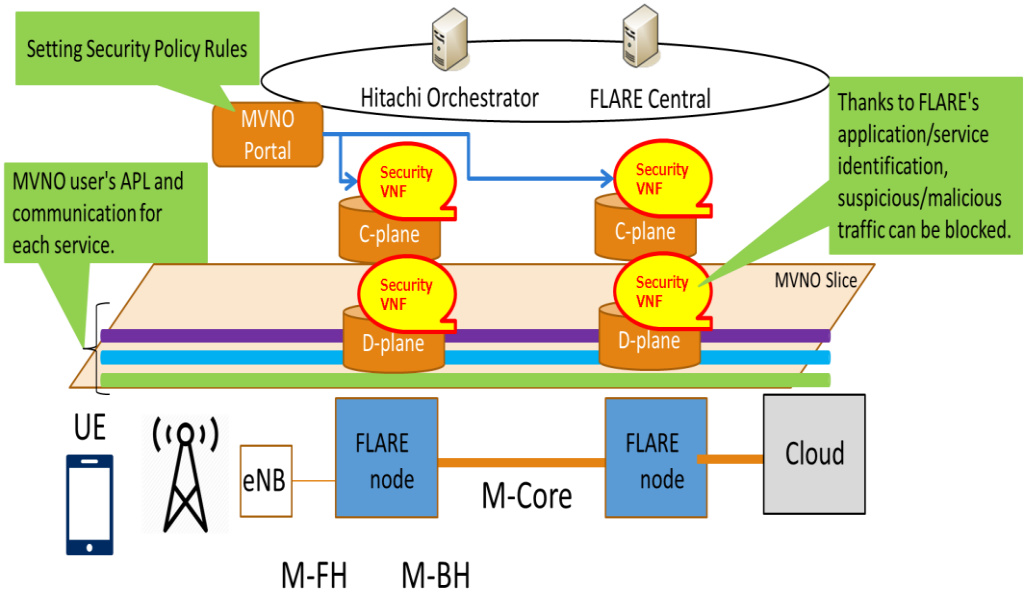

Figure 4-1 – End-to-end communication quality in the security slice

5. Slice composition algorithms and mechanisms

5.1.1. Network Slice Planner

In this section, we describe the new features and functions added to the Network Slice Planning (NSP) framework as a part of the work on slice composition algorithms and mechanisms.

5.1.1.1 Existing NSP system

As defined in D3.1 [1], NSP is a tool that simulates data consumption and mobility of users, based on real users' behaviours in specific regions, by allowing in the same time the personalization of several settings. This personalization is enabled by choosing wisely a set of configuration inputs, related to service consumption, mobility patterns and virtual machines flavours of network slices. NSP framework consists of three main modules: User Mobility Module (UMM), Mobile Edge Cloud Module (MECM) and Service Usage Module (SUM) (see Figure 5-1). These three modules are the pillars for VNF placement strategies, helping to define optimal network slices.

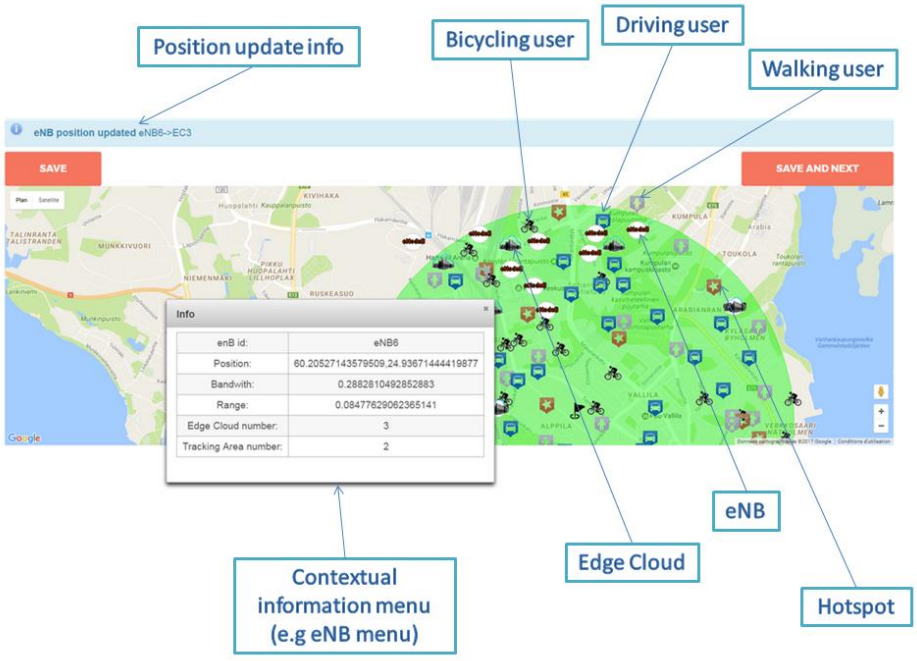

Figure 5-1 – The NSP framework

We summarize the previous version of NSP in the following points:

- Different mobility patterns are defined for UEs; they can be non-mobile, walking, driving or using bikes. The mobility speed for each UE is defined according to the mean of transportation.
- During the mobility of the users, they can access the following services: social networks (i.e. log in to their social network account, browse their photos, messages, their profile or that of their friends, their friends photos, or log out), video streaming (i.e. 360p, 480p, or 720p videos), or instant messaging (i.e. MIME messages).

- ECs locations are defined by latitude and longitude coordinates (lat, lon). The locations can be modified manually in a dedicated page. Each EC has a set of eNBs deployed in its vicinity. Each eNB has a range, a bandwidth and belongs to a unique Tracking Area (TA).
- Depending on the instances that UE jobs require, different VM flavours can be selected, each with different options. VM flavours depend on the type of services and applications launched. The flavours are divided into three main families: the standard flavours used typically for web services and software development, HPC flavours for scientific applications and I/O flavours for Hadoop/Spark, non-critical databases and clustered databases. VMs in NSP can be classified based on the number of cores activated when a flavour is used (vCPUs), the VM Disk capacity and Random-access memory (RAM) capacity.
- The simulation is carried out in real time, and it stops when users reach their favourite hotspots.
- Several parameters related to the transportation mode of users, ECs, eNBs, TAs, VNF's VMs flavours, and positions of each component can be personalized in dedicated pages.

5.1.1.2 Targeted system

To align with the vision of 5G!Pagoda, the new version of NSP is envisioned to support the following new features:

- It should be expanded to include a more extensive set of services such as IoT services, self-driving cars, and Unmanned Aerial Vehicles (UAV).
- Create an appropriate application level traffic (Up-Link (UL)/Down-Link (DL) Traffic) for the services modelled by NSP (i.e. video streaming, social networks, etc.).
- It must offer to its users the possibility to access a more extended set of graphs and logs that provide (i) complete information about each component of NSP (i.e. cell IDs, UE International Mobile Subscriber Identity (IMSI), UE Radio Network Temporary Identifier (RNTI), etc.); (ii) Key Performance Indicators (KPI) generated during each event; (iii) extended set of measurements (e.g. signal strength for LTE cell networks, DL throughput, etc.); and (iv) performance of network functions for several network technologies (i.e. 4G, 5G NR, EPC, 5G CN).
- Cope with the limitations of the previous version of NSP, adding the possibility to manipulate the speed of simulation, enhancing the complexity of the application, etc.

5.1.1.3 Current updates

In its new version, NSP extends its list of supported services to include, in addition to the already existing MBB usage of video streaming, social networks, and instant messaging, (i) UAV related services (home drones, packages delivery, etc.), and (ii) Multiple IoT services (air pollution detection, parking lots sensing, weather sensing, etc.).

It also enables to simulate the performance related to the utilization of eNBs and Edge Clouds in the given simulated area against the behaviour of a group of mobile, sensors, and UAVs. The updated NSP goes beyond providing basic logs about handoff operations, tracking area updates, and service requests issued over a specific geographical area during a specific time window. Now, it leverages on integrating LENA NS3 to have detailed information and statistics of KPIs (e.g., temporal variation of PHY Layer KPIs, temporal PDCP Layer KPI, etc.) and events (e.g., detailed logs

related to the signalling messages generated during a Handoff operation such as UE's position, time, cell id, UE's IMSI, average SINR, etc.). We detail these updates in the following.

5.1.1.3.1 UAV services

For a realistic simulation of UAVs behaviour, NSP integrated ArduCopter[1] and Software in the Loop simulator (SITL)[2]. It is possible to capture all the messages sent by drones, every MaVlink packet and log them into the database. The types of UAV services considered are Drone Transportation and Drone Delivery. Additionally, the ability for the drones to be linked with multiple ENodeBs have been taken into account. Figure 5-2 illustrates the latter.

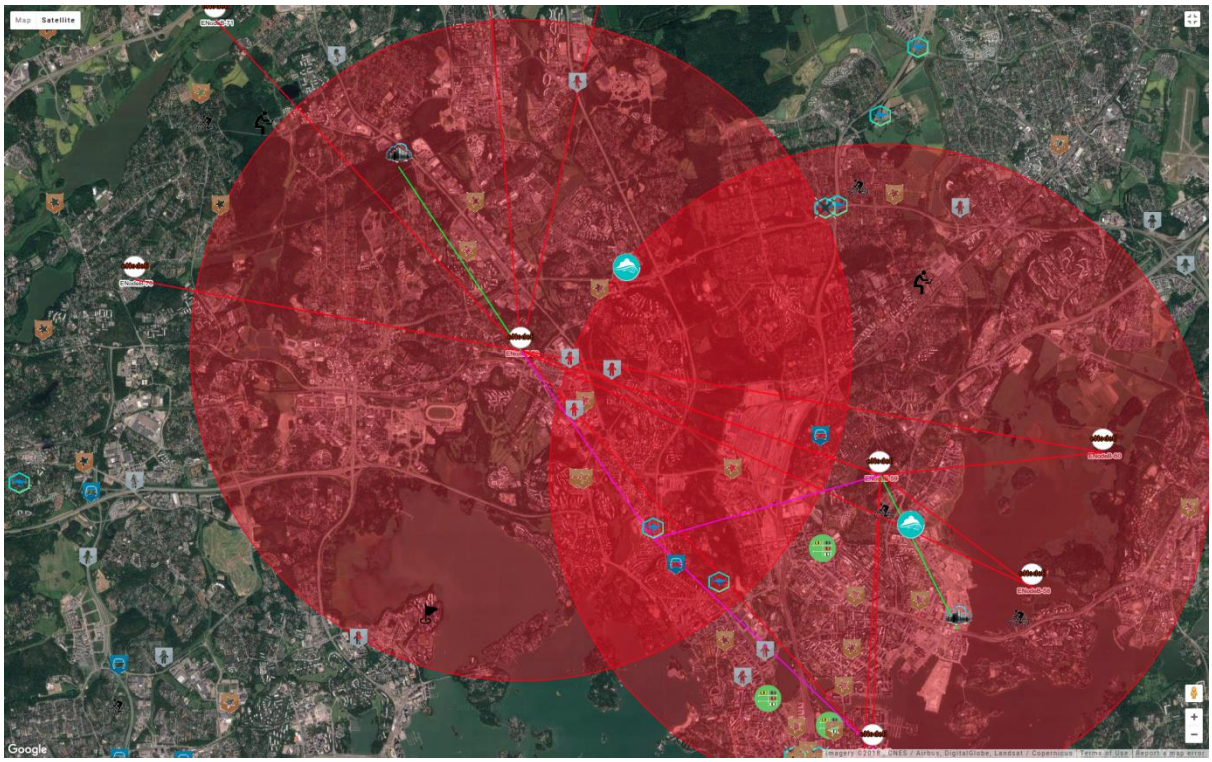

Figure 5-2 – Drones generated in a given area and linked to multiple eNBs

5.1.1.3.2 IoT services:

Throughout the area of simulation, several IoT services such as weather services, parking services, and air quality/pollution detection services were implemented as connected devices. Each device has a data size (i.e. the amount of data sent in a time interval). Data size is calculated based on their type, information received and the update frequency time.

NSP users can define the number of deployed UAVs and that of each type of IoT devices, as illustrated in Figure 5-3.

[1] ArduCopter is an open-source multicopter UAV platform created by the DIY Drones community based on Arduino.

[2] SITL is a simulator, which allows to run Plane, Copter or Rover without any hardware. It is a build of the autopilot code using a C++ compiler, providing a native executable that allows to test the behaviour of UAVs without hardware.

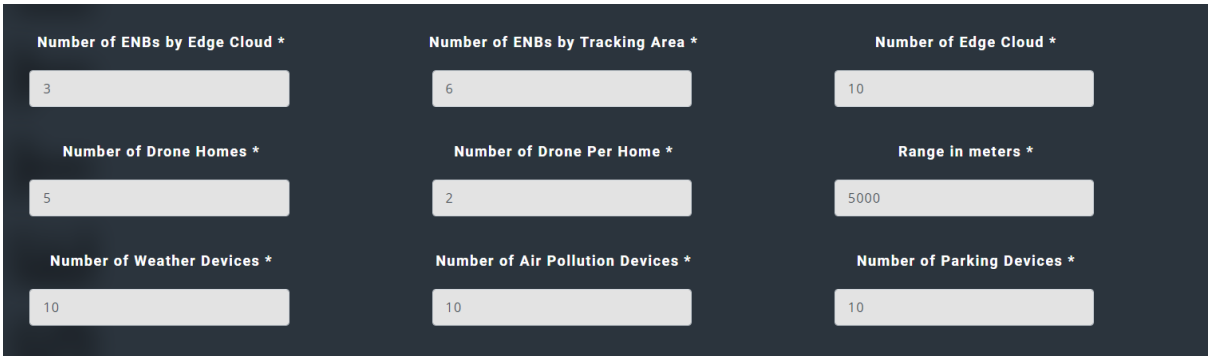

Figure 5-3 – Settings related to UAVs and IoT services

5.1.1.4 Network performance and events

To be able to achieve the objectives defined for the targeted system (see sub-section 4.1.1.2) NSP integrated LENA NS3. The following points were considered for this integration process:

- The simulation zone (in WGS84 coordinate system) is projected onto a Cartesian system (x, y coordinates), which permits to translate the extracted positions of IoT devices, UAVs, UEs, eNBs and MECs from NSP into a movement file to be used by NS3
- Services used in NSP are extracted then translated into applications running between users and a remote host as defined in NS3.
- To improve the complexity of running NS3, we devised the simulation into several independent sets, each to be treated separately on the NS3 side.

For each simulated data on NS3 side, various metrics were stored in NSP database. The main metrics are as follows:

- Temporal variation of PHY Layer KPIs such as RSRP and SINR reported by UEs
- Temporal PDCP Layer KPIs Such as; average PDU size and delay
- Detailed logs related to the signalling messages generated during a Handoff operation, including UE position, time, cell id, UE IMSI, average SINR, etc. as depicted in the Figure 5-4.

sim_time	cell_id	ue_imsi	ue_rnti	rsrp	avg_sinr	component_carrier_id
0.200214000	1	21	0	0.0000000000001464	308.713000000	0
0.200214000	1	38	0	0.0000000000001047	220.830000000	0
0.200214000	1	43	0	0.0000000000002144	452.082000000	0
0.200214000	1	45	0	0.0000000000001200	253.076000000	0
0.200214000	2	12	0	0.0000000000001189	250.598000000	0
0.200214000	2	16	0	0.0000000000001435	302.516000000	0
0.200214000	2	22	0	0.0000000000000153	32.162000000	0
0.200214000	2	24	0	0.0000000000000176	37.043800000	0
0.200214000	2	26	0	0.0000000000000150	31.671700000	0
0.200214000	2	28	0	0.0000000000001208	254.662000000	0
0.200214000	2	29	0	0.0000000000000612	128.964000000	0
0.200214000	2	33	0	0.0000000000002481	523.014000000	0
0.200214000	2	41	0	0.0000000000000098	20.643600000	0
0.200214000	3	6	0	0.0000000000000751	158.380000000	0
0.200214000	3	8	0	0.0000000000000984	207.461000000	0
0.200214000	3	18	0	0.0000000000000499	105.176000000	0

Figure 5-4 – Data about signalling messages generated during a handoff operation

- We can also see attach events, handoffs operations, detailed logs based on NSP inputs logs integration with NS3 as illustrated in Figure 5-5.

meas_start_time	meas_end_time	cell_id	ue_imsi	ue_rnti	lcid	nbr_tx_pdus	total_tx_bytes	nbr_rx_pdus	total_rx_bytes	avg_pdu_delay	std_dev_pdu_delay	min_pdu_delay	max_pdu_delay
1.000000000	1.250000000	7	1	10	3	45	3252	21	1062	0.004666670	0.003055050	0.003000000	0.010000000
1.000000000	1.250000000	6	2	1	3	76	3314	47	1527	0.005085110	0.005245410	0.003000000	0.024000000
1.000000000	1.250000000	8	3	2	3	9	3180	0	0	0.000000000	0.000000000	0.000000000	0.000000000
1.000000000	1.250000000	6	4	21	3	60	4338	50	3224	0.006220000	0.005140040	0.003000000	0.024000000
1.000000000	1.250000000	6	5	3	3	25	4266	19	3032	0.009631580	0.005936800	0.003000000	0.017000000
1.000000000	1.250000000	3	6	6	3	9	3180	0	0	0.000000000	0.000000000	0.000000000	0.000000000
1.000000000	1.250000000	6	7	2	3	72	5414	47	4809	0.005680850	0.004961180	0.003000000	0.024000000
1.000000000	1.250000000	3	8	7	3	10	4236	1	1056	0.003000000	0.000000000	0.003000000	0.003000000
1.000000000	1.250000000	6	9	11	3	29	3220	7	310	0.005000000	0.005291500	0.003000000	0.017000000
1.000000000	1.250000000	7	10	2	3	24	3210	2	6	0.010000000	0.000000000	0.010000000	0.010000000
1.000000000	1.250000000	7	11	4	3	32	4280	11	1547	0.003000000	0.000000000	0.003000000	0.003000000
1.000000000	1.250000000	2	12	12	3	20	5310	3	2297	0.010000000	0.012124400	0.003000000	0.024000000
1.000000000	1.250000000	6	13	27	3	25	3212	16	1591	0.008250000	0.006516650	0.003000000	0.017000000
1.000000000	1.250000000	8	14	3	3	6	3174	0	0	0.000000000	0.000000000	0.000000000	0.000000000
1.000000000	1.250000000	8	15	1	3	11	5292	1	1056	0.003000000	0.000000000	0.003000000	0.003000000
1.000000000	1.250000000	2	16	10	3	18	3198	0	0	0.000000000	0.000000000	0.000000000	0.000000000
1.000000000	1.250000000	6	17	24	3	61	3284	32	1180	0.005406250	0.005511630	0.003000000	0.024000000
1.000000000	1.250000000	3	18	2	3	9	3180	0	0	0.000000000	0.000000000	0.000000000	0.000000000
1.000000000	1.250000000	7	19	3	3	40	6404	21	3053	0.004000000	0.002509980	0.003000000	0.010000000

Figure 5-5 – Detailed information on events occurring from integrating NSP logs with NS3

- Data about KPIs such as temporal variation of PHY Layer KPIs (i.e. RSRP and SINR reported by UEs), temporal PDCP Layer KPIs (i.e. the average PDU size and delay) are also stored in NSP database, as illustrated in Figure 5-6.

event_type	sim_time	cell_id	ue_imsi	ue_rnti	ue_x	ue_y	column_x
CNX_ESTABLISHED	0.260214000	1	21	2	52503.500	21251.000	65535
CNX_ESTABLISHED	0.260214000	1	38	3	61316.600	20343.800	65535
CNX_ESTABLISHED	0.260214000	1	43	1	58617.300	23300.200	65535
CNX_ESTABLISHED	0.260214000	2	24	2	91628.400	13817.700	65535
CNX_ESTABLISHED	0.260214000	2	28	1	73769.300	930.038	65535
CNX_ESTABLISHED	0.260214000	2	41	4	103069.000	11477.100	65535
CNX_ESTABLISHED	0.260214000	3	18	2	29073.800	5421.660	65535
CNX_ESTABLISHED	0.260214000	3	34	1	41557.800	1.094	65535
CNX_ESTABLISHED	0.260214000	3	42	4	45386.300	10278.100	65535
CNX_ESTABLISHED	0.260214000	5	30	1	56727.000	9392.030	65535
CNX_ESTABLISHED	0.260214000	5	32	3	50809.000	18931.000	65535
CNX_ESTABLISHED	0.260214000	5	44	2	53079.900	18322.200	65535
CNX_ESTABLISHED	0.260214000	6	2	1	13655.800	9565.080	65535

Field	Type	Collation	Null	Key	Default	Extra	Privileges	Comment
sim_time	double(15,9)	NULL	YES		NULL		select,insert,update,references	Simulation time in seconds at which the allocation is indicated by the scheduler
cell_id	int(11)	NULL	YES		NULL		select,insert,update,references	Cell ID
ue_imsi	int(11)	NULL	YES		NULL		select,insert,update,references	unique UE ID or IMSI
ue_rnti	int(11)	NULL	YES		NULL		select,insert,update,references	UE RNTI
rsrp	double(20,16)	NULL	YES		NULL		select,insert,update,references	The RSRP
avg_sinr	double(15,9)	NULL	YES		NULL		select,insert,update,references	Linear average over all RBs of the downlink SINR in linear units
component_carrier_id	int(11)	NULL	YES		NULL		select,insert,update,references	N/A

Figure 5-6 – KPIs data

5.1.1.5 Additional features

In addition to the abovementioned, several improvements have been carried out, to get the best of NSP performance, on one side, and to make the simulation more realistic, on the other side. We can resume them in the following:

- Inter Edge Clouds links.
- Links between Edge Clouds and eNBs. eNBs are not bound to a single EC but could be shared between multiple ECs.
- Inter eNBs links to simulate the X2 links and better handle the handoff operations (i.e. with the help of NS3).

Figure 5-2 illustrates such links, the red lines represent links between eNBs (X2 interface), the green lines represent links between an eNB and an EC, and finally, the green lines represent the links between different ECs.

- The time of simulation was considerably reduced by using an offline polyline generation for google maps.
- The drones now have the ability to be linked not to one but multiple eNBs.
- A Scenario Player was added for a better visualization of the simulation as illustrated in Figure 5-7. NSP users can load their scenarios, play, pause, or speed up/down the simulation. They can also filter the visualization of components (i.e. UEs, IoT devices, etc.) and they can store their logs as JSON files.

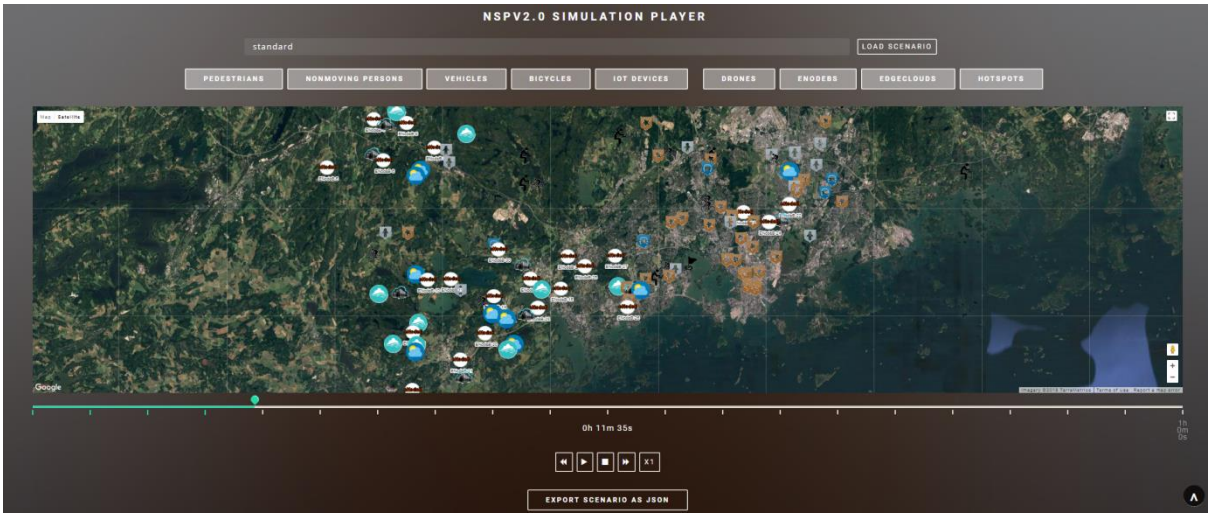

Figure 5-7 – Simulation player

5.1.2. VNF placement algorithms

In addition to the new version of NSP, as part of network slice composition algorithms, we have contributed with a new approach for the placement of network functions virtual resources based on conformal mapping (cf. [12], [13]).

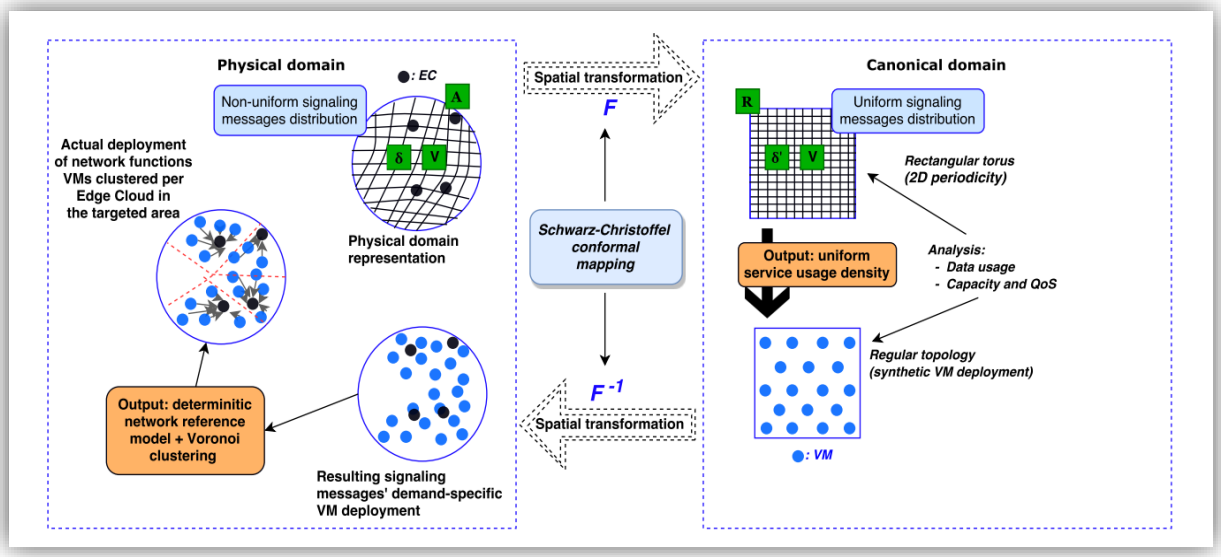

Figure 5-8 – Our proposed framework

Our proposed framework is depicted in Figure 5-8. The spatial transformation function F helps to map every point of the physical domain to a point in the canonical domain. The service distribution in the canonical domain is uniform and such a regular distribution makes it easier to decide where to instantiate network functions virtual resources. The inverse spatial transformation F^{-1} helps to map the generated VMs positions into the physical domain, to be assigned finally to the respective ECs in place using Voronoi clustering, which results in an optimal QoS-aware virtual resource deployment.

$$u = Re(\frac{1-i}{\sqrt{2}} cn(K_e \frac{1+i}{2}(x+iy) - K_e, \frac{1}{\sqrt{2}}))$$

$$v = Im(\frac{1-i}{\sqrt{2}} cn(K_e \frac{1+i}{2}(x+iy) - K_e, \frac{1}{\sqrt{2}}))$$

$$x = Re(\frac{1-i}{-K_e} E(cos^{-1}(\frac{1+i}{\sqrt{2}}(u+iv)), \frac{1}{\sqrt{2}})) + 1$$

$$y = Im(\frac{1-i}{-K_e} E(cos^{-1}(\frac{1+i}{\sqrt{2}}(u+iv)), \frac{1}{\sqrt{2}})) - 1$$

Figure 5-9 – Mapping functions

The choice of the analytic components u and v in Figure 5-9 reflects our spatial service demand distribution, which is graphically fitted to a rectangular grid within a circle. x and y are obtained by applying the inverse function, reflected by the incomplete elliptic integral of the first kind Ef.

The main algorithm for our framework, as depicted in Figure 5-10, is defined as follows: the first step consists of collecting the positions where the data are generated in the current snapshot of time. In the second step, using the mapping functions and our density functions, we obtain the positions and occurrences of signalling messages in the canonical domain. In the third step, due to the uniform nature of the canonical domain, we divide the canonical domain in rectangle tiles. Then, we calculate the needed number of VMs that could fulfil the requests generated in each tile of the canonical domain. In the fourth step, we apply inverse mapping functions to obtain the positions of generated VMs in the physical domain. The last step consists of clustering the generated VMs around the pre-positioned ECs using Voronoi diagram and based on the Harvesin distance of latitude longitudes coordinates.

Algorithm 1 CDSP-F core function algorithm

Require:

Γ: Signaling messages.
X: A set of x positions in the physical domain of signaling messages.
Y: A set of y positions in the physical domain of signaling messages.
X_e: A set of x positions in the physical domain of edge clouds.
Y_e: A set of y positions in the physical domain of edge clouds.

Ensure:

U: The set of signaling messages' u positions in the canonical domain obtained by F
V: The set of signaling messages' v positions in the canonical domain obtained by F
X_{VM}: The set of VM' x positions in the physical domain obtained by F^{-1}
Y_{VM}: The set of VM' y positions in the physical domain obtained by F^{-1}

1: **for all** $\Gamma_i \in \Gamma$ **do**
2: $(U_i, V_i) = F(X_i, Y_i)$; {Calculate position(u,v) in the canonical domain}
3: $GetNeededResources(\Gamma_i)$;
4: **if** $NeededResources_{\Gamma_i} > VMCapacity(U_i, V_i)$ **then**
5: $CreateNewVM(NeededResources_{\Gamma_i}, U_i, V_i)$;
6: **else**
7: $UpdateCurrentVM(NeededResources_{\Gamma_i})$;
8: **end if**
9: **end for**
10: $(X_i, Y_i) = F^{-1}(U_i, V_i)$ {Calculate position(x,y) in the physical domain}
Cluster VMs of the Physical Domain using Voronoi diagram
11: $\mathcal{AL} = VO(X_e, Y_e, X, Y)$;
12: **return** $\mathcal{AL}s$;

Figure 5-10 – Main algorithm of our proposed solution

We evaluated the performance of the proposed solution CDSP-F, and compare it to existing baseline approaches, the Best Fit Algorithm and the Random Fit. The variant solutions have been evaluated, mainly, in terms of end to end delay and cost defined in terms of number of activated VMs. Our framework gave satisfying results in comparison to the assessed algorithms.

5.1.3. Conclusions

To conclude, we can sum up our main contributions in the following points:

- A new update of NSP taking into consideration new services, including UAVs and IoT services, and offering a more complete set of performance indicators, graphs, and metrics.
- we proposed an analysis of service usage and network functions' deployment in the canonical domain, which ensures a uniform distribution by means of Schwartz-Christoffel conformal mappings. The transformation functions of such a mapping ushered efficient ways to quantify and define the needed virtual resources in a MEC environment, particularly to fulfil the service requirements of UEs.
- The initial modelling gave satisfying results and showed the interesting features of the usage of conformal mapping, by reducing the end-to-end delay and the number of activated VMs, in comparison to baseline algorithms.

6. Summary and outlook

In this deliverable, a next stage of the implementation of the components designed to be implemented as part of the software networks within the different slices were presented. In this stage, the roadmaps defined for the different components as well as for the algorithms as presented in D3.1 [1] are followed forward. To conclude this work, the next deliverable will include a final version of the various components.

In Section 2 of this deliverable, the advancements towards a highly customizable core network are presented. As considered from the initial developments of this work package, the current described features concentrated on the grouping of the different functionality within the software components, the way it can be configured and how the different network functions are communicating with each other. Compared to the more theoretical study from D3.1 [1], the work here presented has a clear direct application and implementation within the Open5GCore. The software companion enables the immediate testing of the different deployment designs as well as acting as a validation for the proposed architecture. Furthermore, Section 2 presented a comprehensive analysis of the most interesting slice templates to be deployed in a mobile network system, as well as on the split between the edge and the central entities. With this, a comprehensive analysis of what could be achievable through the configuration of the core networks was presented. As the slice templates are following only a set of deployment models, it is assumed that the rest of the functionality (such as which authentication algorithm to use) will remain at the latitude of the software components supplier on the availability side and of the slice tenant on the deployment side. More considerations in this direction would provide a too large customization, through this going too much in the direction of specific system deployments, specific to product deployments.

Furthermore, Section 2 presented the mechanisms for network slice selection, which have to be added within the software network functions in order to enable the appropriate selection of the slice to serve the subscribers. At the current moment, the most appropriate mechanism is the one considered in 3GPP Dedicated Core Networks (DÉCOR) work item where the RAN is able to proxy requests to the slices, while the mechanism provided in 3GPP 5G allows only that the data plane will be split towards the specifics of the different tenants. An implementation of both mechanisms will follow in the next software releases of the Open5GCore, to enable the validation of the specific communication.

Section 3 continued the work on the deep programmable data plane concentrating into two specific directions of interest within the 5G networks: the support of a separated high-efficient data plane for different slice components, and the support for the interaction of multiple data plane components with the control plane. Through the measurements, it was observed that a software processing of the data messages within the RAN can achieve similar results with a physical implementation, as long as specific optimizations are made available. In conclusion, dedicated hardware meant for the processing of data plane within a network system (such as the Flare nodes) are good enough to support the processing of the data plane also for radio components. This conclusion opens the possibility to split for the different slices only the radio data plane processing and not necessary the scheduling of the radio resources, through this being able to optimize the

data path processing directly from the RAN level, as well as to offload the RAN processing outside of dedicated hardware.

Furthermore, to showcase the capabilities of the deep data plane programmability, the Flare nodes were used for the implementation of ICN and security slices. Especially the support of the ICN within common hardware in parallel to other functionality such as RAN and core network one proves that currently the data plane system does not require anymore the requirement to carry only IP communication.

On the other side, the implementation of security VNFs as part of the data plane proves a very large advantage to the current networking solutions where dedicated security functions such as deep packet inspection or firewalls have to be added to the data path to support different data functions. With the provided solution, in which security applications can be placed on top of common hardware as part of the forwarding plane of the slices, it is not required anymore to have dedicated security appliances, instead replacing them with specific software functionality and benefiting from all the flexibility, re-configurability and deployment advantages of software components.

In Section 4, we have presented the latest developments of the network slice planner, especially concentrating on the specific customization of the provided implementation for the UAV and IoT services. With this, the network performance assessment of the provided solution is directly applied to two of the most demanding use cases.

Furthermore, the network slice planner is extended with a set of virtual network function placement algorithms, which will enable the further refinement of the placement of the network functions within a system with a large number of edge nodes. With this, the end-to-end communication is highly optimized depending on the available edge resources as well as on the momentary communication needs. This functionality will become essential with the evolution towards dedicated slices, which do not require comprehensive coverage within an area and could function efficiently having only some coverage.

References

[1] 5G!Pagoda, "D3.1: Slice components design (M13)", Aug. 2017; source: https://5g-pagoda.aalto.fi/assets/demo/attachement/delivrables/5G!Pagoda%20-%20D3.1%20-%20Slice%20Components%20Design.pdf.

[2] 5G!Pagoda, "D2.1: Use case scenarios, and technical system requirements definition", Jan. 2017; source: https://5g-pagoda.aalto.fi/assets/demo/attachement/delivrables/D2.1_Use_case_scenarios_and_technical_system_equirements_definition.pdf.

[3] 5G!Pagoda, "D2.3: Initial report on the overall system architecture definition", Jun. 2017; source: https://5g-pagoda.aalto.fi/assets/demo/attachement/delivrables/5G!Pagoda%20-%20D2.3%20-%20architecture.pdf.

[4] 5G!Pagoda, "D4.2: Network Slice orchestration", Jun. 2018.

[5] 3GPP, "TR 23.799, Study on Architecture for next Generation Systems", 3GPP Technical Report TR 23.799, version 14.0.0, Dec. 2016.

[6] 3GPP, "TS 23.501, System Architecture for the 5G System", 3GPP Technical Standard TS 23.501, version 15.1.0, Mar. 2018.

[7] 3GPP, "TS 23.502, Procedures for the 5G System", 3GPP Technical Standard TS 23.502, version 15.1.0, Mar. 2018.

[8] 3GPP, "TS 29.510, 5G System; NF Repository Services; Stage 3", 3GPP Technical Standard TS 29.510, version 15.0.0, Jun. 2018.

[9] 3GPP, "TS 29.500, 5G System; Technical Realization of Service Based Architecture; Stage 3", 3GPP Technical Standard TS 29.500, version 15.0.0, Jun. 2018.

[10] 3GPP, "TS 29.531, 5G System; Network Slice Selection Services; Stage 3", 3GPP Technical Standard TS 29.531, version 15.0.0, Jun. 2018.

[11] 5G!Pagoda, "D5.1: Initial report on the integration of the testbeds, experimentation and evaluation", Mar. 2018; source: https://5g-pagoda.aalto.fi/assets/demo/attachement/delivrables/Pagoda_D5.1_Initial%20Report%20on%20the%20integration%20of%20the%20Testbeds,%20Experimentation%20and%20Evaluation_1.0.pdf.

[12] A. Laghrissi, T. Taleb, and M. Bagaa, "Conformal Mapping for Optimal Network Slice Planning based on Canonical Domains", in IEEE JSAC. (to appear).

[13] A. Laghrissi, T. Taleb, and M. Bagaa, "Canonical domains for Optimal Network Slice Planning", in Proc. IEEE WCNC 2018, Barcelona, Spain, Apr. 2018.

[14] 3GPP, "TS 23.682 Architecture enhancements to facilitate communications with packet data networks and applications" , 3GPP Technical Standard TS 23.682, version 14.8.0, Jun. 2018.

[15] 3GPP, "TS 23.401 General Packet Radio Service (GPRS) enhancements for Evolved Universal Terrestrial Radio Access Network (E-UTRAN) access", 3GPP Technical Standard TS 23.401, version 14.8.0, Jun. 2018.

Appendix A.

Open5GCore MME Example configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Copyright (C) 2016 FhG Fokus
  This file is part of the Phoenix project.
  Environment variables can be used in the configuration file. Use "percent"VAR_NAME and it
will be replaced with the
  value of that environment variable. Will error if the variable is not defined.
 -->
<Phoenix>
    <Platform>
        <Debug level="7"/>

        <MemoryMgmt chunk_size="2048">
            <ChunkManager type="msg" size_value="10" size_metric="MB" />
            <ChunkManager type="trans" size_value="2" size_metric="MB" />
            <ChunkManager type="sess" size_value="50" size_metric="MB" />
            <ChunkManager type="perm" size_value="2" size_metric="MB" />
        </MemoryMgmt>

        <ProcessingUnitsMgmt spawn_type="0" />

        <WorkerPool count="8" queue_size="32" />

    </Platform>

    <Module binaryFile="modules/command/command.so">
        <![CDATA[
            <PhoenixCommand>
                  <Greeting text="MME-SGWC-PGWC>"/>
                  <Acceptor type="udp" port="10000" bind="127.0.0.1"/>
                  <Acceptor type="tcp" port="10000" bind="127.0.0.1"/>
            </PhoenixCommand>
        ]]>
    </Module>

    <Module binaryFile="modules/pdn_ops/pdn_ops.so" >
        <![CDATA[
         <PhoenixPdnOps ip_range_tag="pgw1">
            <Database
               hostname="192.168.253.71"
               username="pdn_ops"
               password="pdn_ops"
               database="pdn_ops_db" />
         </PhoenixPdnOps>
      ]]>
    </Module>

    <Module binaryFile="modules/s1ap/s1ap.so">
        <![CDATA[
            <S1AP>
               <Acceptor type="sctp" port="36412" bind="192.168.4.80"/>
         </S1AP>
         ]]>
    </Mdule>

    <Module binaryFile="modules/nas/nas.so"/>

    <Module binaryFile="modules/diameter/diameter.so">
        <![CDATA[
            <phDiameter
               FQDN="mme1.mgmt.o5gc.mnc001.mcc001.3gppnetwork.org"
               Realm="o5gc.mnc001.mcc001.3gppnetwork.org"
               VendorId="10415"
               ProductName="PhoenixDiameterPeer"
               AcceptUnknownPeers="0"
               DropUnknownOnDisconnect="1"
               Tc="30"
               Workers="4"
```

```
            QueueLength="32"
            ConnectTimeout="5"
            TransactionTimeout="5"
            SessionsHashSize="128"
            DefaultAuthSessionTimeout="60"
            MaxAuthSessionTimeout="300">

            <Peer
               FQDN="hss.mgmt.o5gc.mnc001.mcc001.3gppnetwork.org"
               realm="o5gc.mnc001.mcc001.3gppnetwork.org" port="3868" />
            <Peer
               FQDN="scef.mgmt.o5gc.mnc001.mcc001.3gppnetwork.org"
               realm="o5gc.mnc001.mcc001.3gppnetwork.org" port="3868" />
            <Acceptor type="tcp" port="3868" bind="192.168.254.80" />
            <Application type="auth" id="16777251" vendor="10415"/> <!-- 3GPP S6a/S6d -->
            <Application type="auth" id="16777346" vendor="10415"/> <!-- 3GPP T6a -->
            <SupportedVendor vendor="10415" />
            <SupportedVendor vendor="4491" />
            <SupportedVendor vendor="13019" />
            <Realm name="o5gc.mnc001.mcc001.3gppnetwork.org">
               <Route FQDN="hss.mgmt.o5gc.mnc001.mcc001.3gppnetwork.org" app="16777251" />
               <Route FQDN="scef.mgmt.o5gc.mnc001.mcc001.3gppnetwork.org" app="16777346" />
            </Realm>
            <Realm name="pcrf.o5gc.mnc001.mcc001.3gppnetwork.org">
               <Route FQDN="pcrf.o5gc.mnc001.mcc001.3gppnetwork.org" app="0" />
            </Realm>

            <DefaultRoute
               FQDN="hss.mgmt.o5gc.mnc001.mcc001.3gppnetwork.org" app="16777251" />
            <DefaultRoute
               FQDN="scef.mgmt.o5gc.mnc001.mcc001.3gppnetwork.org" app="16777346" />

         </phDiameter>
      ]]>
</Module>

<Module binaryFile="modules/s6ad/s6ad.so" />

<Module binaryFile="modules/t6ab/t6ab.so" />

<Module binaryFile="modules/gtp/gtp.so">
    <![CDATA[
     <GTP>
        <Acceptor type="udp" port="2123" bind="192.168.4.80" protocol="gtpc"/>
        <Acceptor type="udp" port="2124" bind="192.168.4.80" protocol = "s11_gtp"/>
        <Acceptor type="udp" port="2125" bind="192.168.254.80" protocol="pgw_s5s8_gtp" />
         <Acceptor type="udp" port="2126" bind="192.168.254.80" protocol="sgw_s5s8_gtp" />
         <Acceptor type="udp" port="2152" bind="192.168.4.80" protocol = "routing_gtpu"/>
     </GTP>
  ]]>
</Module>

<Module binaryFile="modules/gtp_s5s8/gtp_s5s8.so">
    <![CDATA[
  ]]>
</Module>

<Module binaryFile="modules/gtp_s11/gtp_s11.so" >
    <![CDATA[
     <gtp_s11>
        <sgw_fteid_c
           ipv4="192.168.4.80"
           ipv6=""
           type="11"
           teid="10020" />
        <sgw_fteid_u
           ipv4="192.168.4.210"
           ipv6=""
           type="1"
           teid="10020" />
     </gtp_s11>
  ]]>
</Module>

<Module binaryFile="modules/gw_bindings/gw_bindings.so">
    <![CDATA[
  <GW_BINDINGS
```

```
      hash_size="16">
      <Database
         hostname="192.168.253.71"
         database="gw_bindings_db"
         username="gw_bindings"
         password="gw_bindings" />
  </GW_BINDINGS>
  ]]>
</Module>

<Module binaryFile="modules/mysql/mysql.so"/>

<Module binaryFile="modules/addressing/addressing.so" >
    <![CDATA[
  <Addressing time_to_live="0">
     <Block domain="sgwc.netd.o5gc.mnc001.mcc001.3gppnetwork.org">
        <Address ip="192.168.4.80" weight="3"/>
     </Block>
     <Block domain="default.apn.o5gc.mnc001.mcc001.3gppnetwork.org">
        <Address ip="192.168.254.80" weight="3"/>
     </Block>
     <Block domain="hss.mgmt.o5gc.mnc001.mcc001.3gppnetwork.org">
            <Address ip="192.168.254.70" weight="3"/>
        </Block>
        <Block domain="scef.mgmt.o5gc.mnc001.mcc001.3gppnetwork.org">
            <Address ip="192.168.254.82" weight="3"/>
        </Block>
     <Block domain="google.com">
        <Address ip="192.168.1.9" weight="5"/>
     </Block>
  </Addressing>
  ]]>
</Module>
<Module binaryFile="modules/mme/mme.so">
    <![CDATA[
        <MME
           s11c_mme="192.168.4.80"
           s11u_mme="192.168.4.80"
           s11_dst_port="2124"
            s6a_mme="192.168.254.80"
            s1ap_mme="192.168.4.80"
            hash_size="200"
            monitoring="no"
            ipv6_workaround="yes"
            relative_capacity="1" >
            <GUMMEI
              mcc="1"
              mnc="1"
              mmegi="123"
              mmec="45"
              mme_name="MME_1.Open5GCore" />

        <Security
           eea="0"
           eia="1" />

        <Timers
            tau="240" />

        <TAU>
           <TAI_of_last_tau mcc="1" mnc="1" tac="1" />
           <TA_list>
              <TAI mcc="1" mnc="1" tac="2" />
              <TAI mcc="1" mnc="1" tac="3" />
           </TA_list>
        </TAU>
        <TAU>
           <TAI_of_last_tau mcc="1" mnc="1" tac="17" />
           <TA_list>
              <TAI mcc="1" mnc="1" tac="3" />
              <TAI mcc="1" mnc="1" tac="1" />
              <TAI mcc="1" mnc="1" tac="4" />
           </TA_list>
        </TAU>

            <Database
                hostname="192.168.253.71"
```

```
                    database="mme_db"
                    username="mme"
                    password="mme"
                    session_list_storage="no"
                    imsi_tmsi_mapping_storage="no"/>
                <fqdnParameters
                    networkID="o5gc"
                    operatorID="3gppnetwork.org"
                    scefName="scef.mgmt"
                    sgwcName="sgwc.netd"/>
            </MME>
        ]]>
    </Module>

    <Module binaryFile="modules/pgw/pgw.so">
      <PGW
         ipv4="192.168.254.80"
         ipv6=""
         s5s8_dst_port="2126"
         hash_size="32"
         seamless="1">
         <UserPlane ipv4="192.168.4.210" />
      </PGW>
      ]]>
    </Module>

    <Module binaryFile="modules/gw/gw.so" >
        <![CDATA[

      <Gw
         default_apn="default"
         ipcan="5"
         rat="1000"
         ipv4="192.168.254.80"
         ipv6=""
         lifetime="300"
         hash_size="32"
         pgw_list="0"
         >
         <!--
            Default PGW to contact. Currently multiple PGWs are supported. No default PGW is
needed, as long as
            there is at least one entry in the pgw_list table in the database. If that
            table is empty, the Gw will try to use the default PGW supplied with this file.
         -->
         <DefaultPGW
            ipv4="192.168.254.80"
            ipv6=""
            port="2125"/>

      </Gw>
      ]]>
    </Module>

    <Module binaryFile="modules/ofc_5g_rules/ofc_5g_rules.so">
        <!--
            Extension items:
                type    String item indicating the extension type. Possible values:
                        "sgw_out"
                        "sgw_pgw"
                        "pgw_out"
                        "pgw_gtp"
                mod_name   If GTP-U is to be used, the mod_name is expected to be "routing_gtpu".
                           If other encapsulation methods are to be used (GRE or IPIP),
                           "routing_encap" is expected.  The exact type of encapsulation
                           is specifid in this case in the protocol field.
                protocol   Protocol  to  use.  Used  only  if  the  mod_name  parameter  is
"routing_encap".
         -->
         <![CDATA[
      <PhoenixOfc5GRules>
         <Extension
            type="pgw_out"
            mod_name="routing_encap"
            protocol="gre" />
         <Extension
            type="pgw_gtp"
```

```
            mod_name="routing_gtpu" />
         <!-- Config item configuration -->
         <Config
            layer_mode="l2"
            sgi_mode="none"
            new_ver_enabled="1"
            default_east_ip="192.168.1.210"
            default_west_ip="192.168.4.210"
            default_gw_ip="192.168.1.43"/>
      </PhoenixOfc5GRules>
         ]]>
    </Module>

    <Module binaryFile="modules/ofp/ofp.so">
        <![CDATA[
         <phOFP>
            <Config
               channel_htable_size="8"
               conn_handshake_timeout="5"
               transaction_htable_size="8"
               transaction_timeout="2"
               task_queue_size="8"
               >
               <OFP_Version version="5"/>
               <Logical mode="controller" />
               <Controller>
                  <Acceptor type="tcp" bind="192.168.254.10" port="6633" protocol="openflow"/>
               </Controller>
            </Config>
         </phOFP>
      ]]>
    </Module>

    <Module binaryFile="modules/ofc/ofc.so">
        <![CDATA[
         <OpenFlowController>
         <Features>
            <Publish_Subscribe subscriber_hash_table_size="32" />
            <Working_Mode mode="none" />
            <Topology enabled="false" />
         </Features>
      </OpenFlowController>
      ]]>
    </Module>


    <Module binaryFile="modules/routing_gtpu/routing_gtpu.so" />

    <Module binaryFile="modules/routing/routing.so" >
        <![CDATA[
      <Routing hash_size="64">
         <!--
            Extension id is a string with one of the values: "sgw_out", "sgw_pgw", "pgw_out",
"pgw_gtp", "enb_out", "enb_gtp", "angw_out", "angw_gtp"
            where
            _gtp means it is towards a gtp encapsulated interface
            _out means it is out of the core network: pgw_out is towards the internet and
enb/angw_out is towards the UE
         -->
         <Extension id="enb_gtp" src_table="teid" mod_name="routing_gtpu">
            <Acceptor   type="udp"   protocol="gre"   intf_name="net_d"   bind="192.168.4.80"
port="2152"/>
         </Extension>
            </Routing>
      ]]>
    </Module>

    <Module binaryFile="modules/monitoring/monitoring.so" >
            <![CDATA[
                <Monitoring>
                    <Zabbix    server="192.168.254.61"    port="10051"    msg_queue_id="1234"
hostname="ctrl1" interval="0"/>
                    <Flowmon node_id="PDN-Gw" interval='5' enabled="0">
                        <!-- hostname="flowmon.o5gc.mnc001.mcc001.3gppnetwork.org" -->
               <Database
                  hostname="192.168.254.60"
                  database="flowmon_db"
```

```
                username="flowmon"
                password="flowmon" />

            <Capture interface="lo"
                filter="ip or ip6"
                node_id_suffix="local" />

            <!--
            <Capture interface="enp0s31f6"
                filter="port 80 or port 443"
                node_id_suffix="eth" />
            <Capture interface="tun0"
                filter="ip"
                node_id_suffix="vpn" />
            <Capture
                interface="eth0"
                filter="ip or ip6 and not port mysql and not port ssh"
                node_id_suffix="" />
            -->
          </Flowmon>
        </Monitoring>
    ]]>
  </Module>

</Phoenix>
```