6G BLUR

6G BLUR: The Blurring RAN

JOINT

Grant No. TSI-063000-2021-57

E4: Mechanisms and results report

(JOINT)

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA
Centre Tecnològic de Telecomunicacions de Catalunya

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

Abstract

This report will describe the final mechanisms/algorithms developed and the results obtained within the 6G BLUR JOINT project inside the different key concepts that are later included in the defined proof of concepts of the different use cases.

Document properties

Document number	E4
Document title	Mechanisms and results report
Document responsible	Jorge Baranda, Albert Bel (CTTC)
Document editor	Jorge Baranda, Albert Bel (CTTC)
Authors	Jorge Baranda, Albert Bel, Sergio Barrachina, Miquel Payaró, Engin Zeydan, Josep Mangues-Bafalluy, Marc Carrascosa, Selva Via, Manuel Requena, Katerina Koutlia, Ana Larrañaga, Gabriel Carvalho, Biljana Bojovic, Sandra Lagén (CTTC), Manuel Lorenzo, Saravanan Kalimuthu, Marc Molla, Jose Luis Jimenez, Inmaculada Rafael, Miguel Angel Lopez Serrano, Alvaro Vlad (Ericsson), Carlos Javier Soleto Ramos, Rubén Cerezo, Diego San Cristobal Epalza, Fernando Beltran Gonzalez, Alejandro Ramiro Muñoz (Ericsson), Mattia Lecci, Haoxin Sun, Frankie Garcia, F. Javier Rivas Tocado, Carles Navarro Manchón (Keysight), David Gregoratti, Ismael Gómez (SRS), Luis Miguel Contreras Murillo, Javier Velázquez Martínez, Óscar Gil, María Teresa Aparicio, Elena Serna Santiago (Telefónica), Ramón Agüero Calvo, Luis Francisco Díez Fernandez, Fátima Khan, Neco Villegas (Unican)
Target dissemination level	Public
Status of the document	Final
Version	1.0
Delivery date	31 December 2025
Actual delivery date	31 December 2025

Document history

Revision	Date	Issued by	Description
0.1	17/09/2024	Albert Bel, Jorge Baranda (CTTC)	Initial ToC
0.2	12/11/2024	Jorge Baranda (CTTC)	Joint K1.1, K1.2
0.3	09/12/2024	Saravannan, Jose Luis Jimenez (Ericsson), David Gregoratti (SRS), Jorge Baranda (CTTC)	Ericsson : Joint K1.3, Joint K1.4, Joint K3.2, UC1PoC1, UC2PoC2, UC3PoC1 ; SRS : Smart K1.1, Smart K2.1, UC1PoC2, UC2PoC1; CTTC: Document edition
0.4	10/06/2025	Katerina Koutlia, Ana Larrañaga, Jorge Baranda (CTTC), Javier Velázquez Martínez, Luis M. Contreras, Oscar Gil Lucia (Telefónica), Luis Díez, Fátima Khan, Neco Villegas (UNICAN), Carles Navarro, Mattia Lecci (Keysight)	CTTC: Smart K1.2, Smart K2.3, Smart K2.5, UC1PoC2, UC2PoC1, document edition; Telefonica: Smart K2.2, Smart K2.4, Smart K2.6, Joint K2.1, UC1PoC2, UC2PoC1; UNICAN: Joint K2.2, Joint K2.3, UC2PoC1, Keysight: Joint K3.1, UC2PoC1
0.5	10/11/2025	Albert Bel, Nikolaos Bartzoudis, Jorge Baranda (CTTC)	Smart K1.4, Smart K3.1, Joint K3.1, UC3PoC2, UC3PoC3, document Editions
0.6	25/11/205	Albert Bel, Jorge Baranda (CTTC)	Document division: Smart-Joint
0.7	01/12/2025	Albert Bel, Jorge Baranda (CTTC)	Final Review
1.0	12/12/2025	Jorge Baranda, Albert Bel (CTTC)	Final Edition of JOINT E4 document

Disclaimer

This document has been produced in the context of the 6G BLUR Project. The research leading to these results has received funding from the Ministerio de Asuntos Económicos y Transformación Digital (MINECO), under grant TSI-063000-2021-56/-57.

For the avoidance of all doubts, the MINECO has no liability in respect of this document, which is merely representing the authors' view.

Contents

List of Figures

List of Tables

List of Acronyms

5QI - 5G QoS Identifier

A3C – Asynchronous Advantage Actor-Critic

AD – Anomaly Detection

AF – Application Function

AGV – Automated Guided Vehicle

AI – Artificial Intelligence

API – Application Programming Interface

APN – Access point name

APU – Application Processing Unit

AS – Application Server

ASP – Authorized Service Provider

AXI - Advanced eXtensible Interface

B5G – Beyond 5G

BFP – Block Floating Point

BH – Backhaul

BWP – Bandwidth part

CaaS – Container as a Service

CAPEX – Capital Expenditures

CC – Components Carriers

CG – Cloud Gaming

CLI – Command Line Interface

CP – Control-plane

CR - Control and Scheduling Signaling rate

CSP – Communications Service Provider

CV – Computer Vision

DCI - Downlink Control Indicators

DCI – Downlink Control Indicators

DDP – Drift plus penalty

DL – Downlink

DMA - Direct Memory Access

DNN – Data Network Name

DPDK – Data Plane Development Kit

DRB – Data Radio Bearer

DRL – Deep Reinforcement Learning

DSCP – Differentiated Services Code Point

DT – Digital Twin

EDA – Exploratory Data Analysis

EDA – Exploratory Data Analysis

FFT – Fast Fourier Transformation

FFS – Flexible Functional Split

FH – Fronthaul

FLR – Frame Loss Ratio

gNB – next Generation Node B (5G node B)

GNN – Graph Neural Network

HL – Hierarchical Level

HOL – head-of-line

iFFT – Inverse Fast Fourier Transformation

ILP – Integer Linear Programming

IRU – Indoor Radio Unit

KPI – Key Performance Indicator

KPM - Key Performance Measurement

LC – Logical Channel

LLM – Large Language Model

LSTM – Long Short-Term Memory

MAC – Medium Access Control

MAE – Mean Absolute Error

MANO – Management and orchestration

MCS – Modulation and Coding Scheme

MDP – Markov Decision Process

MH – Midhaul

MILP – Mixed Integer Linear Programming

MIMO – Multiple-Input, Multiple-Output

MKP – Multiple Knapsack Problem

ML – Machine Learning

MLP – Multi-layer Perceptron

MNO – Mobile Network Operator

MPC – model-predictive proactive scheduler

MPSoC – Multiprocessor System-on-Chip

MSE – Mean Square Error

NDT – Network Digital Twin

Near-RT – near Real Time

NEF – Network Exposure Function

NF – Network Function

NIC – Network Interface Card

NPN – Non-Public Network

NRP – Network Resource Partition

NS – Network Service

NSSAI - Network Slice Selection Assistance Information

OAM – Operations, Administration and Maintenance

OFH – Open Fronthaul

OPEX – Operational Expenditures

OS – Operative System

OSM – Open Source Mano

OSS – Operational Support System

OTA – Over the air

OTIC – Open Testing and Integration Centre

OWD – One Way Delay

PDB – Packet Delay Budget

PDU – Packet Data Unit

PF – Proportional Fair

PL – Programmable Logic

PLMN – Public Land Mobile Network

PN – Public Network

PNI-NPN – Public Network Integrated- Non-Public Network

PoC – Proof of Concept

PoP – Point of Presence

PPO – Proximal Policy Optimization

PR – Peak Rate

PRACH – Physical Random Access Channel

PRB – Physical Resource Block

QoS – Quality of Service

R2C – Revenue to Cost Ratio

RAN – Radio Access Network

RAG – Retrieval-Augmented Generation

RDS – Radio Dot System

REG – Resource Element Groups

RFSoC – Radio Frequency System-on-Chip

RIC – Radio Intelligent Controller

RL- Reinforcement Learning

RLC – Radio Link Control

RNN – Recurrent Neural Networks

RR – Round Robin

RRM – Radio Resource Management

RSVP - Resource ReSerVation Protocol

RTT – Round-Trip Time

RU – Radio Unit

RW – Reception Window

SBA – Service Based Architecture

SCS – Subcarrier spacing

SDF – Service Data Flow

SDP – Source Demarcation Point

SDR – Software Defined Radio

SDU – Service Data Units

SIMD – Single Instruction, Multiple Data

SLA – Service Level Agreement

SMF – Session Management Function

SRB – Signalling Radio Bearer

SRS – Software Radio Systems

SST – Slice service type

SVR – Support Vector Regressor

TAE – Time Alignment Errors

TBS - Transport Block Size

TDD – Time Division Duplex

ToR – Top of Rack

TTI - Transmission Time Interval

TTI – Transmission Time Interval

TW – Transmission Window

UE – User Equipment

UL – Uplink

UP – User-plane

UPF – User Plane Function

URLLC – Ultra-reliable Low-Latency Communication

VLiM – Virtual Link Mapping

VM – Virtual Machine

VNE – Virtual Network Embedding

VNF – Virtual Network Function

VNoM – Virtual Node Mapping

VNR – Virtual Network Request

WRR – Weighted Round Robin

XR – eXtended Reality

ZDM – Zero-Defect Manufacturing

ZMQ – Zero Message Queuing

ZSM – Zero-touch Service Management

Executive Summary and Key Contributions

In this deliverable *(E4: Mechanisms and result reports)*, we present the algorithms/procedures and the results obtained within the different key concepts and the derived proof of concepts defined in the 6G-BLUR JOINT project.

This project specifically targets: (i) to design the 6G disaggregated mobile network, from RAN to core, according to service-based approaches in a way that it can be treated as any other virtual service that requires some specific hardware support, and (ii) to design and to validate of joint RAN and transport orchestration mechanisms that adapt to the needs of the disaggregated 6G network.

Based on the work developed in previous deliverables, namely E1 and E3, this report focuses on the beyond state of the art work developed within nine Key concepts, which explored the dynamic deployment of disaggregated cloud-native mobile networks using open-source software and orchestration tools, the automated deployment of non-public networks (NPN) using commercial equipment, the creation of network digital twins based on the deployed NPN to apply closed-loop optimization strategies, development of monitoring platform based on O-RAN architecture (i.e., using KPM monitoring for the E2 interface based on xApps running in the near-RT RIC), study of transport orchestration components to automate the end-to-end management of mobile networks, among other aspects.

All these key concepts, together with other key concepts developed within the other subproject of the 6G-BLUR project (6G-SMART) have been integrated four Proof of concepts (PoCs), showing the complementarity between subprojects. The key contributions of such PoCs are summarised as follows:

- A commercial 5G NPN can initialize/auto-configure, auto-integrate with the public network, auto-integrate with the edge network functions, and become fully operational after 10 minutes, thus helping companies to enrich their communication infrastructure performing a Zero-touch deployment.
- A complete end-to-end deployment of a disaggregated over-the-air cloud-native mobile network comprising Radio access network (RAN), core and transport segments and including O- RAN architectural principles can be achieved by integrating open-source solutions such as srsRAN and Open5GS, leveraging Kubernetes-based deployments using Helm charts embedded into OSM descriptors and the Network Slice Controller interacting with TeraFlow SDN controller to dynamically provisioning of differentiated transport network slices with specific requirements in terms of latency and bandwidth.
- O-Distributed Unit (DU) pooling at centralized locations is indeed feasible with the current O-RAN O-FH specifications, provided that the relevant parameters of the O-DU are

appropriately adjusted. It is crucial to adjust the O-DU TX and RX windows in order to reflect the O-RU timing and the packet delay and jitter introduced by the O-FH. The E2E latency achieved by a system with a long O-FH is affected by multiple factors: number of registered UEs, computational capabilities of the servers running the RAN functions, SNR at the air interface, etc. Adjusting the time available for processing at the O-DU can be used to optimize the E2E latency, but conservative settings should be prioritized to avoid connectivity problems when the system is under high load conditions.

- Based on the available NPN network, a network digital twin was created. This full setup has been used to integrate prediction, recommendation and analytics application programming interfaces (APIs) to illustrate how enterprises can utilize the Design Recommendations API to deploy new connectivity use cases or reconfigure those running tailored to specific target KPI requirements, such as the AGV deployment in industrial settings.

This summary is a high-level overview, but all the details can be found and are extensively developed within the different sections of this document.

1 Introduction

The aim of this document is to present the algorithms/procedures and the results beyond state-of-the-art (SoA) obtained for each KC and later present the final integrated setups for the different PoC at each UC to obtain further results/procedures (e.g., steps of the zero-touch NDT or steps to perform the e2e orchestration of the o-ran based network with emulated transport topology).

The blurring RAN (6GBLUR) project, funded under the UNICO MINECO program, is structured into two complementary subprojects: 6GBLUR-SMART, focused on intelligent decision-making and efficient end-to-end resource management, and 6GBLUR-JOINT, centred on joint RAN and transport-network control and orchestration.

Next-generation RAN deployments must address increasing demands in terms of spectral efficiency, energy consumption, resource pooling, scalability, and cross-layer coordination. Standardization bodies such as 3GPP and O-RAN are driving this evolution by promoting virtualized and disaggregated architectures where conventional base-station functionalities—traditionally co-located at tower sites—are split and distributed across logical nodes connected through fronthaul, midhaul, and backhaul domains. These flexible architectures enable fine-grained functional splits and support the aggregation of baseband processing for multiple cells in centralized environments, adapting dynamically to network conditions and operator policies. This leads to the progressive *blurring* of traditional RAN boundaries and enables the emergence of new deployment paradigms increasingly adopted by vendors and MNOs to reduce CAPEX/OPEX and improve energy-aware performance.

Despite the benefits, flexible virtualized RANs face five core challenges: (i) fronthaul constraints that intensify under 6G features (massive MIMO, higher bandwidths, modulation orders, carrier aggregation), (ii) the need for a hierarchical control architecture spanning real-time and non-real-time decision loops, (iii) the selection and coordination of multiple dynamic functional splits across distributed cells, (iv) KPI-dependent variable latency requirements, and (v) trust and security in open, multi-vendor environments.

The objective of 6GBLUR is to provide an end-to-end architecture capable of efficient resource orchestration in adaptive and virtualized scenarios. This involves managing radio spectrum, transport capacity, compute and storage, energy consumption, and processing functions. To do so, AI/ML-based mechanisms are leveraged for real-time and non-real-time decision loops across O-RAN and NPN environments. The resulting architecture has been validated through a set of Use Cases (UCs) and PoCs addressing zero-touch deployment, fronthaul-aware optimization, QoS-driven control, flexible functional split adaptation, digital-twin-enabled NPN management, and coordinated RAN-transport orchestration.

Within this framework, 6GBLUR-JOINT specifically targets: (i) the design of a disaggregated 6G architecture extending from RAN to core using service-based principles, and (ii) the development and validation of joint orchestration mechanisms capable of adapting to dynamic transport and RAN conditions.

To materialize and evaluate these technical objectives, several UCs and PoCs have been implemented to validate the final Key Concepts (KCs) of the project. The definition and assignment of KC ownership—established in the November 2023 technical meeting—guided the progress of WP2 and WP3 and forms the basis of the demonstrators and proof of concepts included in this deliverable.

This document presents the final architecture and reports the validation results obtained through the PoCs, demonstrating the feasibility, performance gains, and technological advances associated with each KC. In addition, it consolidates the final contributions of the consortium and showcases the experimental maturity achieved by the project.

The document is organized as follows: Section 1 introduces the scope and motivation of the project; Section 2 presents the final key concepts (KCs) developed; Sections 3 presents the results obtained in 6GBLUR-JOINT's PoCs based on the integration of the different KCs.

1.1 JOINT Key Concepts Overview

The nine key concepts in 6G BLUR-JOINT are summarized below. We present all the key concepts that are under study in both work packages, 2 and 3 to be tested in the framework of WP4. All these key concepts are the ones that are aligned with the principal objective of 6G BLUR-JOINT project, that is to construct a disaggregated mobile network integrating all the virtualized RAN and core components.

TABLE 1. JOINT KEY CONCEPTS AND MAPPING TO POCS.

Category	KC	UC1PoC1	UC1PoC2	UC2PoC1	UC2PoC2	*UC3PoC3*[1]
K1. End-to-end orchestration	**K1.1** Data plane reconfiguration (UPF) and Zero-Touch		**X**			

[1] Although UC3PoC3 has been developed at 6GBLUR smart project, the PoC has partially based on the K3.1 developed in 6GBLUR JOINT project. For more information, please refer to 6GBLUR SMART E4 deliverable.

for multi-domain disaggregated mobile networks	**K1.2** Distributed deployment of e2e slices, including O-RAN		X			
	K1.3 Non-public-network	X			X	
	K1.4 Network Digital Twin	X			X	
K2. Backhaul and end-to-end transport management	**K2.1** Transport network optimization		X	X		
	K2.2 Entities' placement decisions			X		
	K2.3 QoS policies and scheduling			X		
K3. Monitoring and AIMLP	**K3.1** Monitoring platform					X
	K3.2 AI/ML Platform (AIMLP)	X			X	

Also, in Table 1, it is shown the relation between Proof of Concepts and Key Concepts. As it can be seen major key concepts are focused on the orchestration and monitoring of RAN and transport components. Following the main objective of the 6G BLUR-JOINT project, the different PoCs that have been developed, focus the efforts on optimizing the performance of a 6G network.

2 JOINT Key Concepts Procedures/Algorithms and results

2.1 JOINT-K1.1: Data plane reconfiguration and Zero-Touch

2.1.1 System design

B5G/6G networks are being built upon architectural concepts like Service Based Architecture (SBA), function disaggregation and enabling technologies like virtualization/cloudification to achieve the required flexibility, dynamicity, and automation capabilities to manage and orchestrate them and to transition towards autonomous networks with close-loop capabilities. For that, in the work developed within this KC, we have dig into the benefits and possibilities of the SBA approach, considering the disaggregation of mobile core Network Functions (NFs), to automate the distributed deployment of a cloud-native 5G mobile infrastructure using open-source software. Unlike other efforts in the literature, this work, presented in [1], goes beyond them allowing the disaggregation of the mobile core deployment into control plane and user plane network functions, so this deployment can be distributed and done on-demand, starting/stopping the different entities (i.e., multiple UPFs) when they are needed. In other words, mobile core control-plane functions can be deployed in a centralized point of presence (PoP), allowing user-plane functions to be distributed across various locations, as depicted in Figure 1. The control and user-plane functions are treated as individual and independent ETSI NFV Network Services (NSs), leveraging Open Source MANO (OSM), an ETSI NFV Management and Orchestration (MANO) stack, thus providing the automated and flexible capabilities to the deployment process. It is worth mentioning that thanks to the proposed architecture using open interfaces and standard ports, this deployment is compatible and has been tested with multiple-vendor radio access networks (RANs) like Amarisoft HW and srsRAN software, as presented in next key concept about end-to-end distributed deployments.

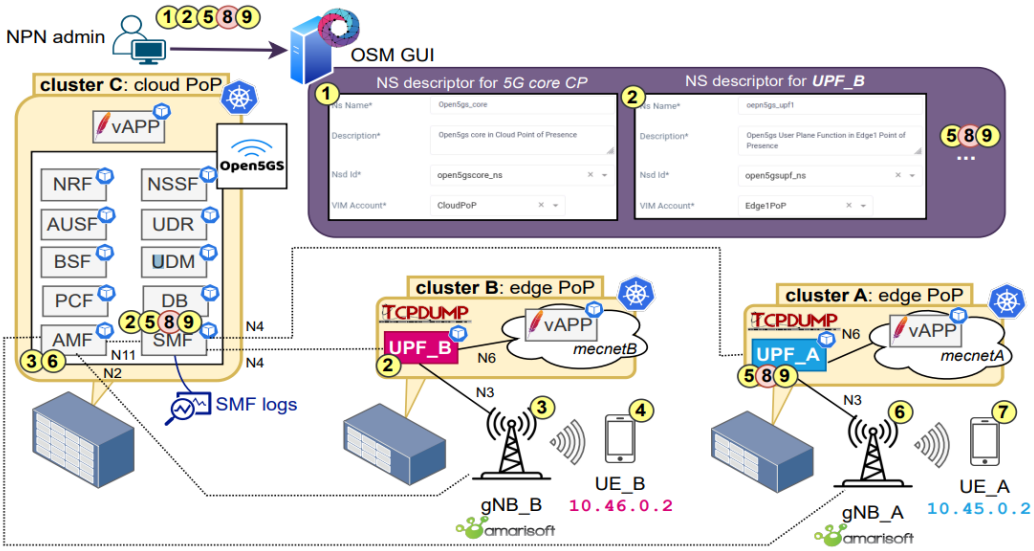


FIGURE 1. DISTRIBUTED 5G MOBILE CORE DEPLOYMENT DEVELOPED IN [1]

The following sub-sections describe what are the different developed artefacts to achieve this kind of deployments [2], as well as a numeric evaluation about the time required to instantiate and terminate the 5G mobile core control-plane network functions and user-plane network functions.

2.1.2 Implementation

The process to be able to decouple control-plane (CP) and user-plane (UP) of the 5G mobile core starts with the development of the Dockerfile packaging the Open5GS open-source software into a Docker container. Later, two different configuration blueprints (i.e., Helm charts) were developed for the disaggregated deployment of the 5G mobile core (Open5GS v2.6.4) network functions. One blueprint contains the Kubernetes' manifests to deploy the control-plane functions (AMF, AUSF, BSF, database, user interface, NRF, NSSF, PFC, UDM, UDR, SMF). In this Helm chart, each control-plane function runs in an independent container executing the same image but different process. This blueprint acts as the centralised point of management of the whole mobile network, and it can be deployed in a cloud PoP. The other blueprint consists of the manifests belonging to the User-Plane Function (UPF). This blueprint provides flexibility to this mobile core setup, as different instances can be activated/deactivated at the considered edge PoPs when required.

The flexibility and easy templating capabilities of Helm charts allow the parametrization of the different 5G mobile core network functions. For the inter-pod communications of the control-plane blueprint, most of NFs use Kubernetes ClusterIP service type, so the associated pods are only accessible within the Kubernetes cluster (i.e., providing security to prevent non-required external

access). However, this first blueprint required the provision of external interface access to connect with the RAN segment and UPF entity (i.e., AMF N2 interface and SMF N4 interface). For such external access to the AMF and SMF cluster pods, the LoadBalancer service type is used. This type of service is also used in the user-plane blueprint to enable the handshake with the corresponding control-plane functions. This kind of Kubernetes service favours the distribution of mobile network entities across independent endpoints and allows the use of the standard ports (i.e., port 38412 for SCTP connection on N2 interface and port 2152 for GTP tunnelling connection on N3 interface), helping with the inter-operability with other equipment (e.g., Amarisoft RAN devices).

Finally, the corresponding OSM descriptors (VNF and NS packages) and OSM parameter configuration files exploiting exposed Helm chart parametrization capabilities were developed to increase the degree of automation in the deployment process. This is for instance, allowing the generic creation of multiple instances of the UPF in different edge PoPs or the deployment of different instances of the control-plane functions by using a single NS definition with its associated Helm chart, the selection of the deployment PoP and the corresponding the suitable OSM parameter configuration file.

2.1.3 Results

This section presents the time required to instantiate and terminate the CP and UP artefacts across an experimental setup deployed in EXTREME® xG Testbed located in CTTC (Castelldefels, Spain). This MANO framework is distributed among three commercial-of-the-shelf servers. One server runs an instance of ETSI Open Source MANO (OSM) Release 14 and the two remaining servers act as PoPs, where a single-node Kubernetes cluster in each PoP acts as a Container Infrastructure Manager. In order to give statistical meaning to the results, each experiment is repeated 50 times, and the results are presented using boxplot representation. The boxplot includes the mean value, the median value, quartiles, and potential outliers. The boxplot spans the interquartile range, representing the median of the measured time for each operation in a solid black line, with the bottom and top edges indicating 20% and 80% percentiles, respectively. The solid blue line represents the average value.

During this evaluation, we consider different deployment options, taking advantage of the capabilities of the developed artefacts. On one hand, we consider the deployment using OSM application programming interface (API) and on the other hand directly using the Helm command line interface (CLI) tool to directly start the Helm Chart definition. It is worth remembering the flexibility provided by OSM, which allows the deployment of the NS in any of the registered PoP remotely by just contacting OSM instance, while for the deployment using the Helm Chart definition, it is needed to directly access the Kubernetes cluster where you want to deploy such Helm Chart.

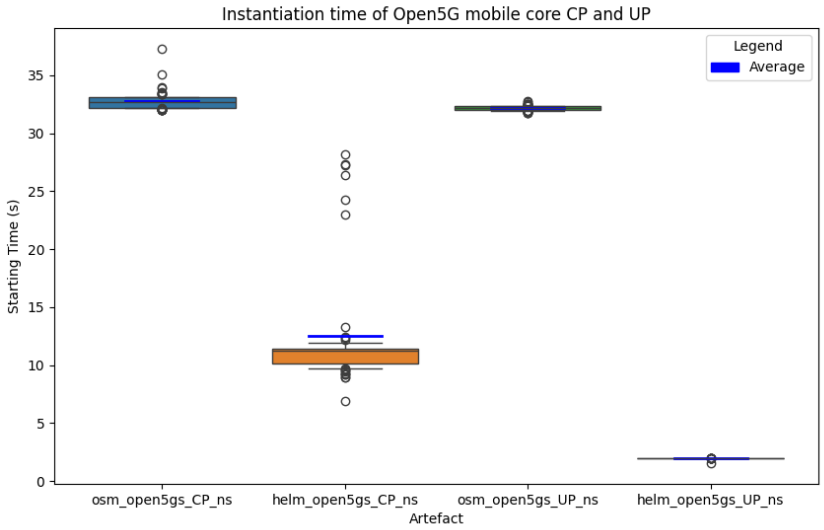


FIGURE 2. COMPARISON OF INSTANTIATION TIME OF OPENGS CP AND UP ARTEFACTS

Figure 2 presents a comparison among the time required to instantiate the Open5Gs CP and UP artefacts using the two mentioned methods. It can be observed that the mean time required for OSM to declare as instantiated both the Open5Gs CP and UP NS is on average around 32 seconds, while for the Helm chart is around 12 seconds for the CP and 2 seconds for the UP artefacts, respectively. Comparing instantiation methods, the required time to instantiate the CP using OSM is quite stable through repetitions, as seen from the width of the boxplot. The same trend is observed for the case of UP considering OSM and Helm chart. Although the underlying Helm chart used by the OSM package is the same that it is used during the direct Helm CLI execution, the difference in experienced time between using OSM or directly deploying the Helm Chart comes due to how OSM handles the interactions with Kubernetes. OSM plugin for Kubernetes goes beyond checking container deployment status and performs additional actions (e.g., readiness for Day-1 & Day-2 options). Actually, the time we measured with the deployment of Helm Chart considers the time it takes for Kubernetes to declare all the pods associated to the Helm Chart in *Ready* state. Furthermore, the difference between CP and UP instantiation times come from the fact that when starting the Helm Chart associated to the control-plane, up to 11 different Kubernetes pods, services resources are started parallelly, one for each CP function. While in the case of the UP using Helm tool, only a single Kubernetes pod, service resources are started. In addition to this, some of the pods in the CP Helm chart require to do several re-start operations as there are dependencies between functions, such as for instance the need to contact and register into the Network Repository Function (NRF). This would also explain the higher experienced variability (width of the boxplot) for the instantiation of the Open5GS CP using directly the Helm CLI, where the maximum experienced instantiation is closer to

those experienced by OSM. A final remark comes when comparing the Kubernetes-based results with the instantiation of a similar NS using virtual machines (VMs). As reported in [3], the time required to deploy a 4G mobile core NS using a single VM containing the multiple network functions (CP and UP) jumps up to 150 seconds. In this setup, the combined time (CP and UP) is in the order of 64 seconds using OSM, more than 40% of reduction, without mentioning that with the produced artefacts a more flexible, distributed, sequential deployment is achieved.

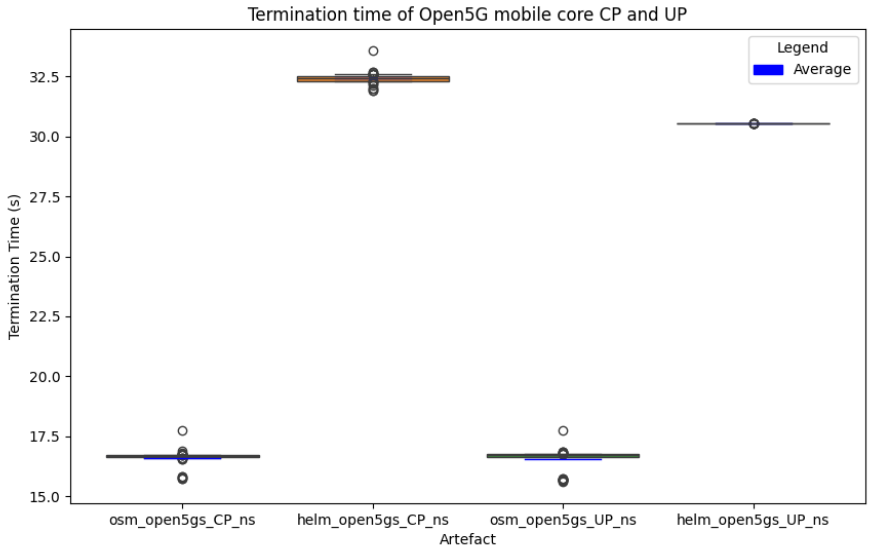


FIGURE 3. COMPARISON OF TERMINATION TIME OF OPENGS CP AND UP ARTEFACTS

Regarding the termination operation, presented in Figure 3, measured times present more stability of results across repetitions, according to the width of the boxplot. However, results here are the opposite. OSM declared the NS service as terminated, disappearing from its database records, in around 16 seconds, while however, the time measured with Helm chart is around 30 seconds. This is because OSM software declares the service as terminated, before all the pods of the given NS are terminated, which is what is measured when directly using the Helm tool. This is very important to highlight because in case of doing a re-deployment with OSM immediately after terminating, there could be a collision of Kubernetes resources because they are still in the process of being de-allocated and this re-deployment may fail. Regarding closer termination times reported for CP and UP, they are due to the fact that the deallocation of the multiple pods, services and resources associated to the multiple CP network functions are done in parallel. Of course, CP may take additional extra time, when observing Helm chart-based measurements, because 11 pods are terminating in comparison to the single pod for the case of the UP.

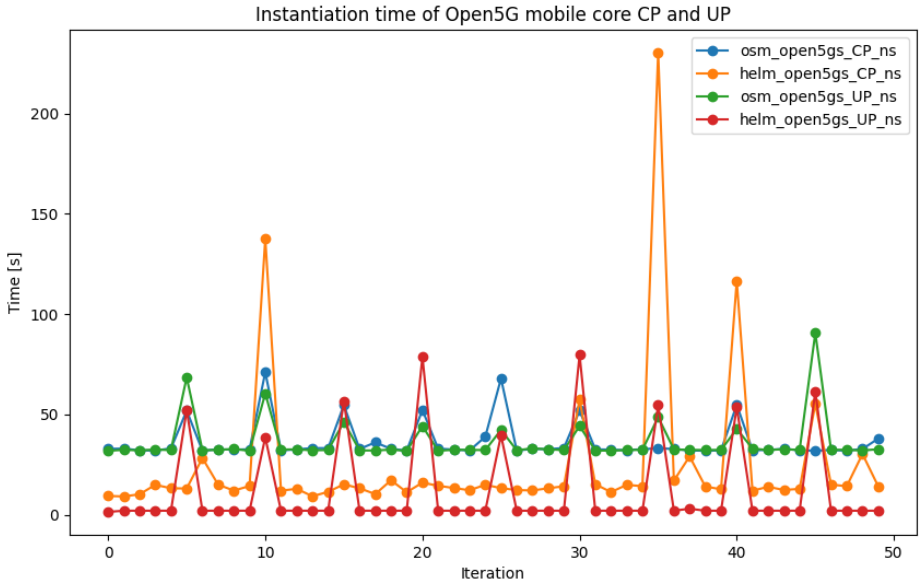


FIGURE 4. EVOLUTION OF INSTANTIATION TIME ACROSS ITERATIONS FOR THE OPENGS CP AND UP ARTEFACTS

Figure 4 shows the time required to instantiate the different NSs (i.e., Open5Gs CP or Open5Gs UP) through OSM or the Helm chart directly, when the container image is not available (i.e., cached) in the Kubernetes cluster. For that, every 5 iterations, the container image is removed from the Kubernetes cluster where the deployment is being done so a new download of the generated Open5Gs container from DockerHub source is required. It is worth mentioning that by default, the produced Helm charts artefacts have an "Always" pull policy for container. However, there is a "cache checking" operation before starting the container so, the Always pull policy is executed if a change is detected from the Dockerhub source. This is why it is required to completely remove the container image from the Kubernetes cluster to force a new pull operation. As expected, the instantiation times grow up in those iteration requiring the downloading of the container image, which is this case is 566MB. In particular, these times can jump up to around 90 seconds to instantiate the Open5gs UP with OSM, as its worst-case. In such repetitions when a container download is needed, there is not so much difference between the time required to instantiate a given artefact using either OSM or the Helm Chart. We can also observe, that in some of the repetitions requiring to download the container, OSM achieves lower values than with Helm, which can be explained due to the variability in the connection between the Kubernetes cluster and the Dockerhub source. In the worst-case scenario, Helm required 230 seconds to start an instance of the Open5gs CP artefact. Nevertheless, this still means that the mobile network can be deployed in the order of minutes.

2.1.4 Conclusions

The current SBA architecture of mobile core network allows the disaggregated and on-demand deployment of the elements of an open-source cloud-native 5G mobile core, which can be done in the order of few minutes (i.e., less than ten minutes), even if a completely new container image is required to be downloaded. Besides the easy re-deployment and the portability of the mobile core deployment to other similar setups, the proposed approach enables exploiting the benefits of edge locations according to the needs of the mobile network. This is possible thanks to the design and implementation of the corresponding artefacts and the exploitation of the capabilities of open-source tools like OSM and Kubernetes. Overall, this work acts as a starting point for the (multi-cluster) disaggregation of 5G/6G core network functions, paving the way for the development of more sophisticated scenarios incorporating multiple slices and enabling data-driven approaches for proactive orchestration.

2.2 JOINT-K1.2: Distributed deployment of e2e slices, including O-RAN

2.2.1 System design

To progress towards an end-to-end view for the mobile network, the RAN needs to embrace similar conceptual approaches like the mobile core network based on function disaggregation and open interfaces while also integrating cloud-native principles. Such a trend is being developed in the RAN segment, where the Open RAN (O-RAN) alliance [4] is proposing an architecture based on disaggregation and softwarisation to naturally include close-loop operations enhancing the performance of the network while promoting interoperability among multi-vendor providers. In this context, it is also essential to enable the appropriate MANO procedures to automate such deployments, ensuring an efficient use of computing, memory, and network resources distributed across the different available PoPs. The work developed in this KC focuses on the RAN part, including O-RAN entities, employing the work developed in SMART K1.1 concept about CU and DU implementation and complementing the work in Section 2.1 to be able to deploy end-to-end slices. Working also with open-source software, this work goes beyond current efforts in literature, which just consider single-host deployment just starting to embrace containerized deployments employing Docker and requiring the execution of the software processes directly from the involved hosts. Herein, following the same approach as in Section 2.1, the available open-source software has been analysed to generate the appropriate artefacts and propose software enhancements (e.g., announcing proper IP addresses on the different interfaces assuming a deployment using containers)

enabling an automated, distributed, disaggregated, flexible, and sequential deployment of different entities at the RAN segment (considering also different working modes), making possible advanced scenarios considered in O-RAN specifications [5], like the one in the Figure 5 below.

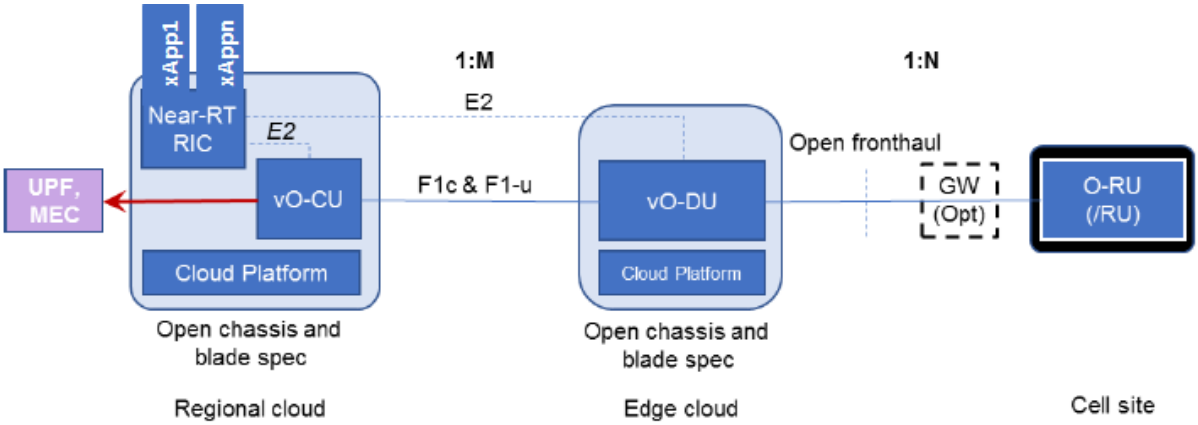


FIGURE 5. DISTRIBUTED AND DISSAGREGATED DEPLOYMENT OF O-RAN ENTITIES. BASED ON [5]

Given the deployment of the CP and UP entities of the mobile core, the provider can later choose different locations for the cloud-native instance of the monolithic gNodeB (gNB), the CU, the DU entities, the Near Real-Time (near-RT) Radio Intelligent Controller (RIC), and corresponding xApps to be distributed among available edge PoPs, as we have showed in [6], [7]. As in previous Section 2.1, these entities are treated as individual NSs, which can be deployed through an ETSI NFV MANO platform to facilitate the operation of the setup. The combination of artefacts developed for this KC and KC1.1 enhances the automation, flexibility and portability capabilities of the deployment process by keeping track of the many configuration and executable files to configure the whole end-to-end mobile network. As an example, thanks to the parametrization applied in the artefacts, end-to-end mobile network deployment footprints with different slice configuration at the mobile core and the RAN components can be easily deployed supporting different slice configuration. Moreover, this approach can facilitate the continuous adaptation of the network to evolving requirements, supporting the on-demand creation and deletion of different entities, such as UPF, gNBs or xApps to enable data-driven close-loops orchestration procedures.

The following sub-sections describe what are the different developed artefacts to achieve this kind of deployments [3], as well as a numeric evaluation about the time required to instantiate and terminate the different O-RAN entities.

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA
cttc Centre Tecnològic de Telecomunicacions de Catalunya

2.2.2 Implementation

Following the methodology described in Section 2.1, the process to disaggregate the different entities at the RAN segment, considering the mentioned O-RAN entities, starts with the development of the Dockerfile packaging on one hand the srsRAN project [8] implementing gNB (i.e., joint CU/DU), CU and DU entities (as developed in SMART KC 1.1), and on the other hand the Flexric project [9] implementing the near-RT RIC and xApps. It is worth noting that for a correct execution of the srsRAN or Flexric processes and to allow dynamic (re)deployments, the generated container has a script to dynamically replace in the configuration file the IP address where the process is listening to with the IP address assigned dynamically to the container prior to start the corresponding process.

Later, different configuration blueprints (i.e., Helm charts) using the defined containers were developed for the disaggregated deployment of the different O-RAN network functions. More specifically, there are three different blueprints to be able to launch the gNB and the DU, respectively. These different blueprints are prepared to exploit the possibilities the srsRAN software provides to use different Radio Unit (RU) elements: an emulated RU using Zero Message Queuing (ZMQ) networking library (no over-the-air transmission), a software defined radio (SDR) element (e.g., USRP B210) for Split 8 and an O-RU element (e.g., Benetel 550) for Split 7.2 using Open Fronthaul (OFH) interface. Then, there is a blueprint for the CU, to interconnect the DU with the mobile core. Two blueprints are devoted to the FlexRIC framework (*br-flexric branch)*. More specifically, one blueprint defines the deployment of the near-RT RIC component, while the other blueprint allows the deployment of a monitoring xApp enabling the selection of the O-RAN Key Performance Measurement (KPM) parameters provided by the gNB or the DU srsRAN implementation to be monitored. In total, 9 different blueprints have been created to implement the different entities at the RAN segment and exploit the possibilities of the available SW for testing deployments including over-the-air transmission. Finally, for the case of using the O-RU (i.e., Split 7.2), an additional Helm chart has been defined to configure the tight timing synchronisation required between the DU and the RU element. This synchronisation is provided by an O-RAN switch & PTP grandmaster (i.e., Falcon-RX/812/G Switch) inserted between the RU HW and the Kubernetes cluster where the DU network service is running.

As in Section 2.1, the developed artefacts make use of the flexible templating capabilities of Helm charts to enable the distributed deployment across available Kubernetes clusters and the compatibility with other SW/HW components. These blueprints provide external access for the pods by exposing standard ports defined in 3GPP specifications and making use of Kubernetes *LoadBalancer* type of service. The gNB blueprints expose the created pod to establish a handshake with the AMF, and the UPF specified by SMF upon a UE association request, through the N2 and N3 interfaces using port 38412 for SCTP and port 2152 for GTP connections, respectively. In addition to

this, the gNB also exposes the port 36421 to establish a SCTP connection with the near-RT RIC making use of the O-RAN defined E2 Interface protocol. In the case of the blueprint version using the O-RU device for Split 7.2, the "*hostNetwork*" feature of a Helm chart deployment template needs to be set to true to receive the samples coming from the O-RU. The CU blueprint also exposes the pod and the same ports as the gNB on the *northbound* to interact with the mobile core. On the *southbound,* it exposes 38472 to establish the F1-c connection towards the DU by means of SCTP protocol. The port 36421 is also exposed to interact with the near-RT RIC by means of a SCTP connection. It is worth mentioning that N3 and F1-U interfaces use a GTP connection with the same port (i.e., 2152). To enable distributed deployments using this cloud-native approach while avoiding port collision, a patch is needed in the srsRAN SW to modify the F1-u communication between CU and DU. This patch establishes the F1-u communication between CU and DU enabling additional ports 2153 and 2154. The DU blueprints expose the created pod to establish a handshake with the CU through the F1-c interface using port 38472 for SCTP connection. It also exposes the port 36421 to report metrics to the near-RT RIC by means of another SCTP connection, and the ports 2153 and 2154 to establish the F1-U connection with the CU. As in the case of using the O-RU with Split 7.2, the DU blueprint needs to include the "*hostNetwork*" feature in the Helm chart deployment template. The parametrised configuration included in the DU Helm charts also considers the possibility of attaching multiple DU instances to the same CU instance. The near-RT RIC blueprint exposes the created pod so gNB/CU and DU can send the KPM reports to it and the monitoring xApp blueprint can retrieve the measurements through the E2 interface using two SCTP connections running on ports 36421 and 36422, respectively. As the last step, all these blueprints are packaged in OSM descriptors (VNF and NS packages). The available parameterization in these blueprints allows the creation of appropriate configuration files to feed OSM instantiation commands, thus enabling the on-demand and generic creation of instances of such entities in the different available Kubernetes clusters.

2.2.3 Results

This section presents the time required to instantiate and terminate the different entities in the RAN segment, namely a gNB, CU, DU, near-RT RIC and xApp across the experimental setup deployed in EXTREME® xG Testbed located in CTTC (Castelldefels, Spain), as described in Section 2.1.3. Again, each experiment is repeated 50 times, and the results are presented using boxplot representation. The boxplot includes the mean value, the median value, quartiles, and potential outliers. The boxplot spans the interquartile range, representing the median of the measured time for each operation in a solid black line, with the bottom and top edges indicating 20% and 80% percentiles, respectively. The solid blue line represents the average value.

During this evaluation, we also compare the instantiation/termination time using OSM API against deployments using directly the Helm CLI. Next, some additional insights on how these experiments have been executed. First, the experiments considering the deployment of the gNB and the DU were done considering the version of the blueprint using the ZMQ library to avoid stressing the RU HWs (i.e., USRP N210 and Benetel 550). Isolated experiments showed similar performance in terms of experienced instantiation time. Second, to run the proposed experiments, it is needed the deployment of a complete mobile core (CP and UP). For that, Section 2.1 deployments were re-used. Third, the configuration files of the whole system (i.e., mobile core CP, UP, gNB/CU/DU) are configured to handle two slices, identifying them with a different slice service type (SST) identifier pointing to different defined access point names (APNs).

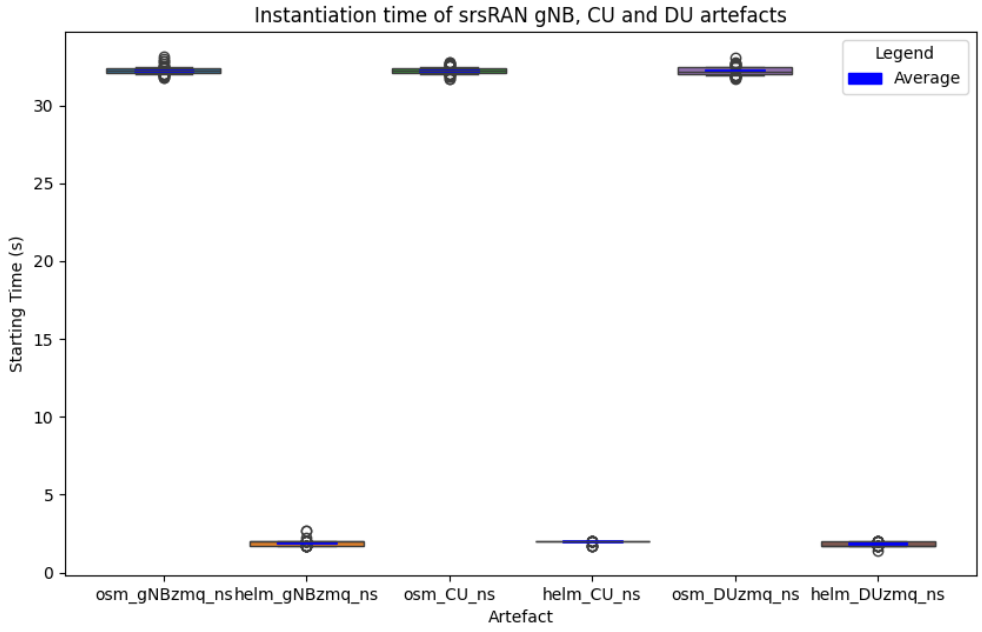


FIGURE 6. COMPARISON OF INSTANTIATION TIME OF SRSRAN-BASED GNB, CU, DU ARTEFACTS

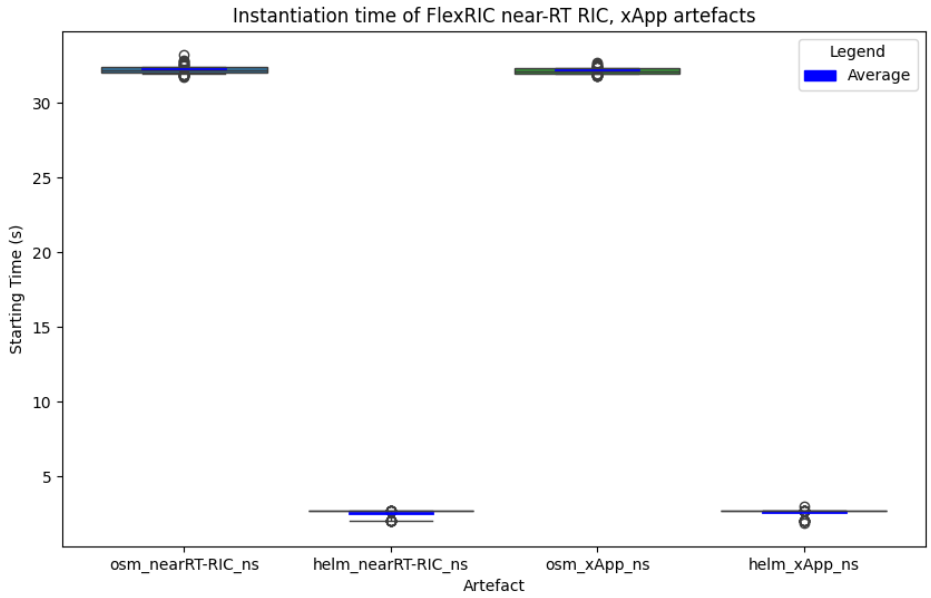


FIGURE 7. COMPARISON OF INSTANTIATION TIME OF FLEXRIC-BASED NEAR-RT RIC AND XAPP ARTEFACTS

Figure 6 and Figure 7 present a comparison of the time required to instantiate srsRAN-based gNB, CU and DU arterfacts and FlexRIC-based near-RT RIC and xApp artefacts using the mentioned deployment methods. We can observe similar trends to those observed in Figure 2, where mean instantiation times using OSM is around 32 seconds, while the time required by a Helm install operation to declare the associated pods in *Ready* state is around 2 seconds. Bear in mind that in these artefacts, there is only a single pod, service Kubernetes resource defined, like in the Open5Gs UP artefacts. Due to this *simpler* Helm chart definition, the measured instantiation times are more regular across repetitions, according to the width of the boxplot. This is similar to the trend observed for the Open5Gs UP artefact in Figure 2. The size of the used container image for srsRAN-based artefacts[2], which is around almost three times bigger than the one used in Open5GS artefacts (1.45GB vs 566MB), is not impacting in the experienced instantiation time since in this experiment, the container image is cached at the Kubernetes cluster. It is worth mentioning that in the container generated for the srsRAN SW, this SW is compiled to be compatible to work with all possible types of available RU: ZMQ library, SDR-based devices and O-RU for OFH, so lower size of container could be possible if using different images for the different options. Nevertheless, as previously mentioned, this difference in container image is not relevant when this image is cached in the Kubernetes cluster.

[2] For the sake of completeness, the size of the container image used with FlexRIC SW is 521MB

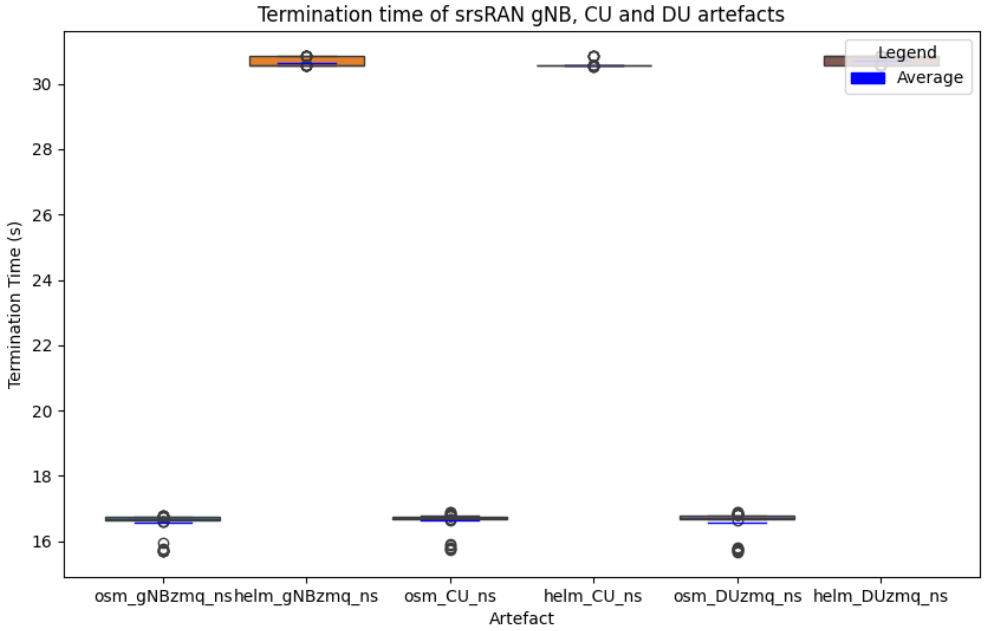


FIGURE 8. COMPARISON OF TERMINATION TIME OF SRSRAN-BASED GNB, CU, DU ARTEFACTS

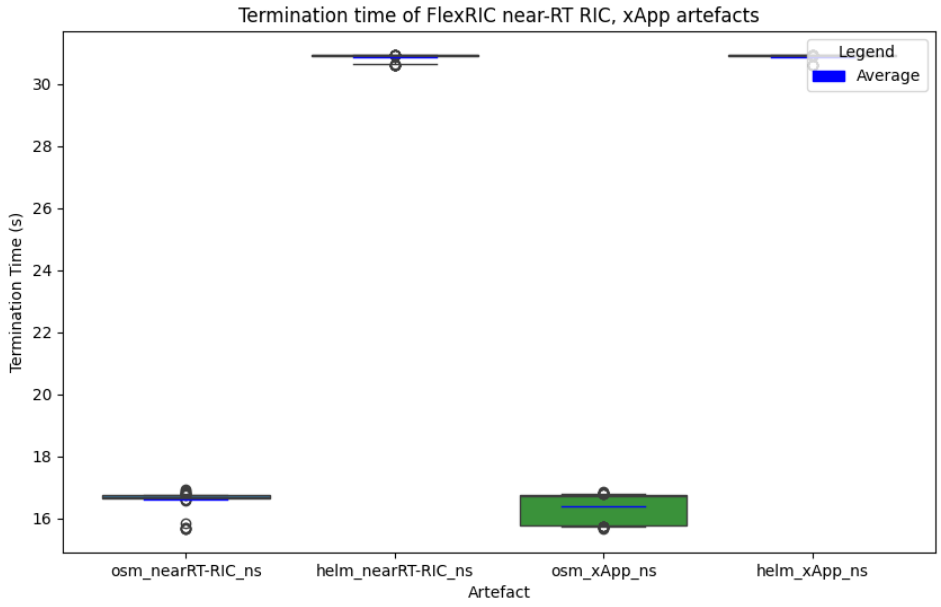


FIGURE 9. COMPARISON OF TERMINATION TIME OF FLEXRIC-BASED NEAR-RT RIC AND XAPP ARTEFACTS

Figure 8 and Figure 9 present the measured times for the termination operation and both methods (i.e., OSM API and Helm chart). As observed in Figure 3, the measurements reported by OSM are lower to those measured with Helm CLI tool (i.e., around 17 seconds versus 31 seconds). Nevertheless, the values provided by OSM represent the time required to OSM to declare the NS as finished. However, the pods associated to this NS are still terminating, preventing the use of such resources. Bear in mind that to enable distributed deployments, these pods are exposed to the network assigning an IP address corresponding to the *LoadBalancer* pool, so a new deployment using the same configuration will not be successful. In Figure 9, a thicker boxplot to terminate xApp NS with OSM is measured. Nevertheless, the interquartile time between 80% and 20% is around two seconds.

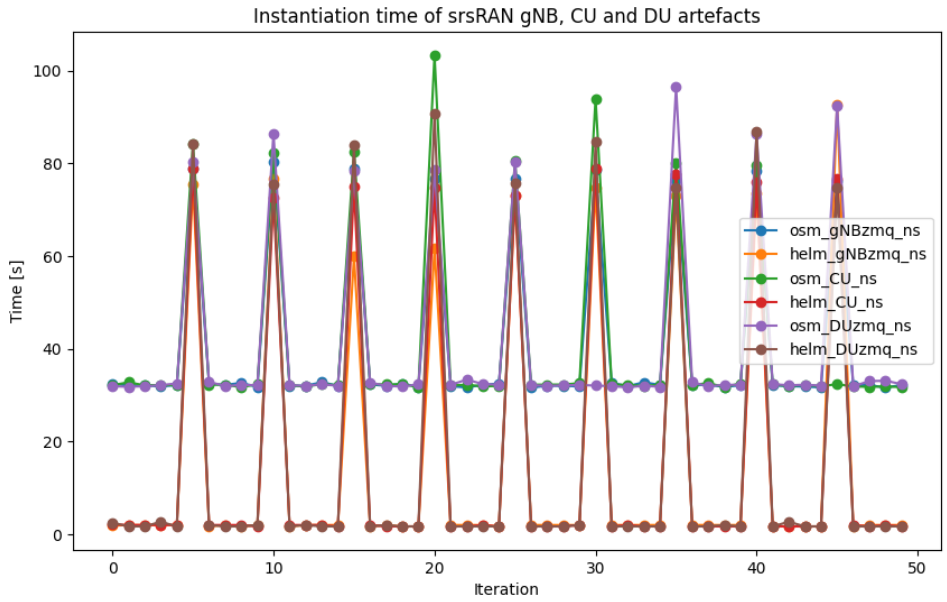


FIGURE 10. EVOLUTION OF INSTANTIATION TIME ACROSS ITERATIONS FOR THE SRS-BASED ARTEFACTS

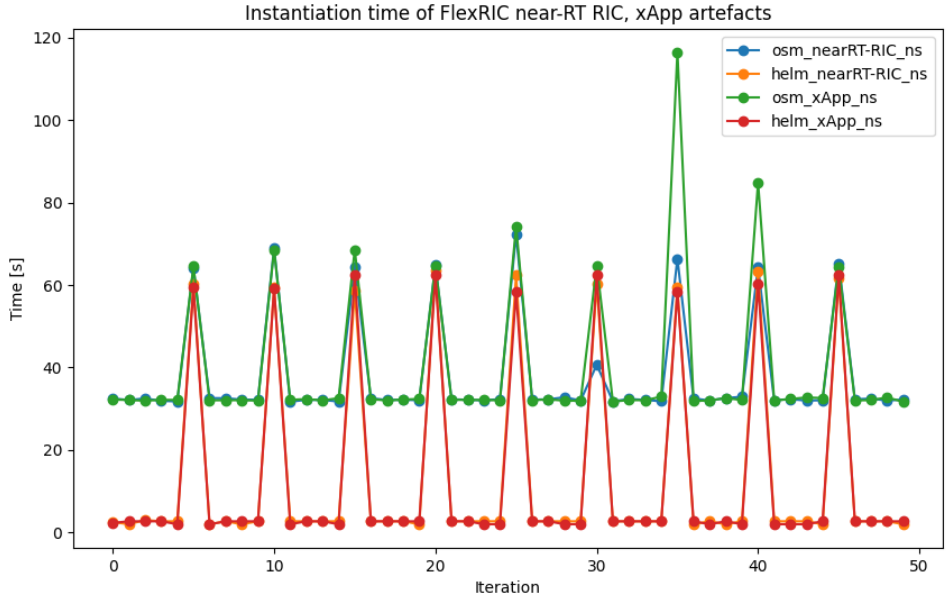


FIGURE 11. EVOLUTION OF INSTANTIATION TIME ACROSS ITERATIONS FOR THE FLEXRIC-BASED ARTEFACTS

To complete this analysis, Figure 10 and Figure 11 the time required to instantiate the different NS using srsRAN-based and FlexRIC-based artefacts for the gNB, CU, DU, near-Real Time RIC and xApp when the container image is not cached in the Kubernetes cluster (i.e., every 5 repetitions out of the temporal serie of 50 repetitions, the container image is erased from the cached repository at the Kubernetes cluster). As expected, based on the observations of Figure 4, the instantiation time grows when a new download of the container image is required. In such iterations, the time required by OSM and Helm are closer, meaning that the operation time is impacted by the container download process. Nevertheless, although the size of the srsRAN container image is considerable bigger than the one of the FlexRIC (1.45GB vs 521MB), there is not a considerable difference in the maximum experienced time, so additional operations inside the server where the Kubernetes cluster is hosted are also contributing to a higher experienced instantiation time besides the downloading process. Indeed, in another repetition of the experiment evaluating the instantiation time of the nearRT-RIC executed during night, we can see that the low network activity in the testbed allows for a bigger portion of the available bandwidth to download the container image, making almost imperceptible this operation according to the measured times with OSM, as depicted in Figure 12.

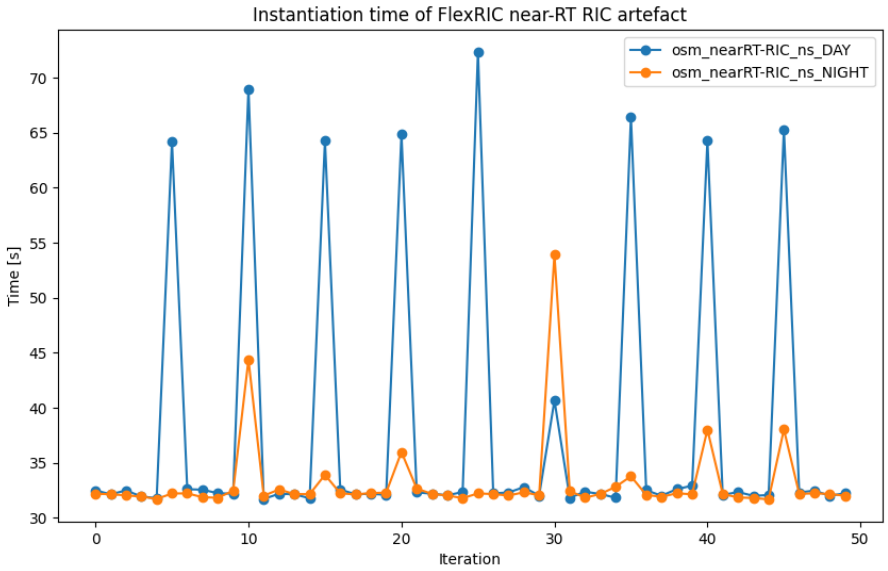


FIGURE 12. EVOLUTION OF INSTANTIATION TIME ACROSS ITERATIONS FOR NEAR-RT-RIC ARTEFACT EXECUTING THE EXPERIMENT IN DIFFERENT TIMES OF THE DAY

2.2.4 Conclusions

This KC focused on the RAN/O-RAN components, to following the example of KC1.1 in Section 2.1, enabling the end-to-end deployment of a distributed and on-demand cloud-native 5G mobile network featuring slices, encompassing both Core and RAN functionalities. The key point is the advancement in the deployment of RAN components following a cloud-native approach, which is not available in the literature. Adding the obtained results in both key concepts, the complete deployment of a mobile network could be done from scratch in less than 5 minutes when considering the case of instantiating with OSM an instance of the mobile core (CP and UP), the CU, the DU, the nearRT-RIC and a xApp. The deployment strategy is based on designing and implementing effective descriptors that leverage orchestration tools like OSM and Kubernetes, thereby enabling flexible and automated deployment capabilities. This approach empowers B5G/6G network management to dynamically adapt and scale according to operational requirements, facilitating autonomous orchestration decisions driven by closed-loop feedback mechanisms. The outcomes of the work developed in SMART K1.1, K2.1 and JOINT K1.1, K1.2 and K2.1 key concepts, dealing with the dynamic configuration of a transport network interconnecting Kubernetes clusters containing different mobile network entities (i.e., CU and DU), are the pillars for the realisation of UC1PoC2 presented in Section 3.1.2.

2.3 JOINT-K1.3: Non-Public network

A groundbreaking concept emerging within the core network landscape is that of Non-Public Private Networks (NPNs). NPNs revolutionize network provisioning by offering dedicated, private network environments specifically tailored to the requirements of enterprises, industries, and government organizations. By providing enhanced security, reliability, and control, NPNs empower businesses to deploy customized network solutions aligned precisely with their needs. This revolution paves the way for optimized operations, tailored services, and increased operational efficiency.

2.3.1 System design

Starting from Release 16, 3GPP 5G Phase 2 specifications provide requirements, capabilities, and solution sets for the support of NPNs. NPNs can be classified into the following categories or models:

Stand-alone NPN (SNPN): It is an NPN that does not rely on network functions provided by a Public Land Mobile Network (PLMN). This is the well-known Standalone or dedicate private network. This is the model that has been mostly used so far for Large Enterprises.

Public Network Integrated NPN (PNI-NPN): It is an NPN deployed with the support of one (or more) PLMN(s). This second model refers to a hybrid model, with a small private network on customer integrated with the public network of the service provider.

Within 6G BLUR project, we believe that a hybrid model fits better in medium size enterprise and even in many cases in large enterprises, as mid-size companies, will need differentiated services at scale, in a short time and at an affordable cost, meaning that Communication Service Providers (CSP) needs to develop the concept of "mass-customization."

For the development and implementation of the **Public Network Integrated NPN (PNI-NPN)**, we believe that we can make use of a number of complementary technologies that, when integrated together, will allow us to offer this mass customization at a reasonable cost.

Therefore, the system design revolves around the development of following capabilities (some of them are explained in detail as other Key Concepts in this document):

Zero Touch: We believe that the private network delivery model should make use of zero touch capabilities, i.e. being able to ship the Radio and User Plane part pre-configured with factory defaults. Zero touch capabilities will allow plug and play installation and configuration.

Network Digital Twin: In addition, the communication service provider will make available to the Enterprise customer a digital twin of its private network. This digital twin with AI and ML capabilities

will offer very powerful configuration, prediction and simulation services for the private 5G network segment. See Section *2.4 JOINT-K1.4: Network Digital Twin* for further details.

Network Slicing: The service provider should also have massive networking capabilities to offer differentiated services that adapt to the different use cases of that enterprise customer.

Exposure of Network APIs: Finally, the digital twin should expose a series of network APIs that support the enterprises during the whole innovation cycle, i.e. not only exposing analytical data of their live system but also offering functionalities during the design of the use case during the pre-production of the use case and giving the possibility to optimize the system once it is live.

Edge computing: Moving computing power closer to the end-user in order to enable applications and services requiring unique connectivity characteristics such as ultra-low latency. Placing relevant applications near the base station not only offers advantages to consumer and enterprise end users, it also reduces the volume of traffic offloaded to the core network and minimizes operational costs (OPEX) and helps to address security and data governance issues. In our system design, the NPN hosts the Radio components together with the UPF (User Plane Function), while the Control Plane functions remain in CSP central location.

Following Figure 13 represents the integration of different but complementary technologies implementing the PNI-NPN architecture.

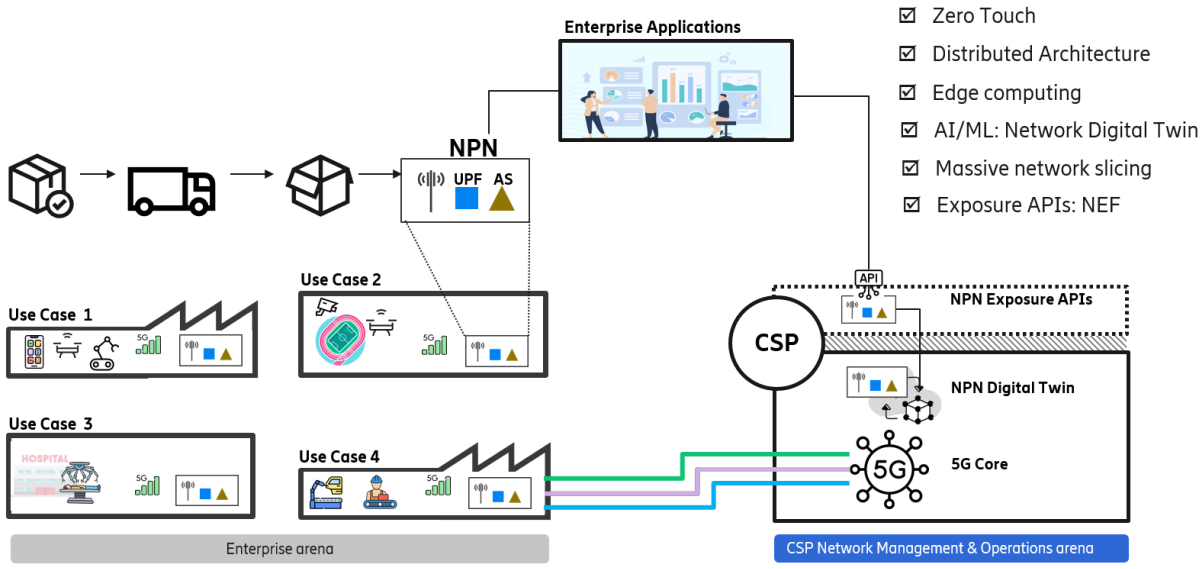


FIGURE 13. PNI-NPN MODEL – INTEGRATING TECHNOLOGIES

2.3.2 Implementation

When it comes to the practical implementation of the PNI-NPN, we can start with a geographical description of the setup deployed.

FIGURE 14. GEOGRAPHICAL PNI-NPN SYSTEM COMPONENTS

The main components are:

- PNI part were deployed in 5TONIC lab at Leganes Madrid. Consisting of 5GC and Network Digital Twin Platform.
- NPN part were delivered and deployed in CTTC Lab at Castelldefels, Barcelona. Consisting of a portable NPN system flight-rack cabinet hosting the following components, as depicted in Figure 15:

  1. 5G Radio DOT System (RDS): It provides 5G NR connectivity and is specially designed for indoor coverage.
  2. 5G Indoor Radio Unit (IRU): The main purpose of the IRU is the transmission of signals providing an interface to the RDS through the Radio DOT Interface and supplying power to the RDS through this interface.
  3. 5G Baseband: It provides switching, traffic management, timing, baseband processing, and radio interfacing.

4. IP Router: It is a high-capacity access router, designed to provide high-density 10G interfaces. It supports VPN services over IP/MPLS networks, service provider SDN, service exposure using NETCONF/YANG, extensive quality of service, and precise synchronization features.
5. GPS: a GPS receiver consists of a GNSS Active antenna and a GPS03. The GPS signal is needed to have a Time & Phase Synchronization required for 5G with TDD strategy.
6. Dedicated Commercial of the Shelf (COTS) Dell PowerEdge server for UPF: It is the physical server that hosts Kubernetes K3S distribution as the Container as a Service (CaaS) layer and UPF CNF will be deployed over this layer.
7. Dedicated COTS Dell PowerEdge server for hosting application function: It is the physical application server (AS) that hosts application function (AF) software (i.e., robot-controller logic).
8. Power Supply Unit: It is used to provide power supply to baseband, IRU, and IP router.
9. Flight Rack: 19" rack flight case to host all physical NPN components explained above.

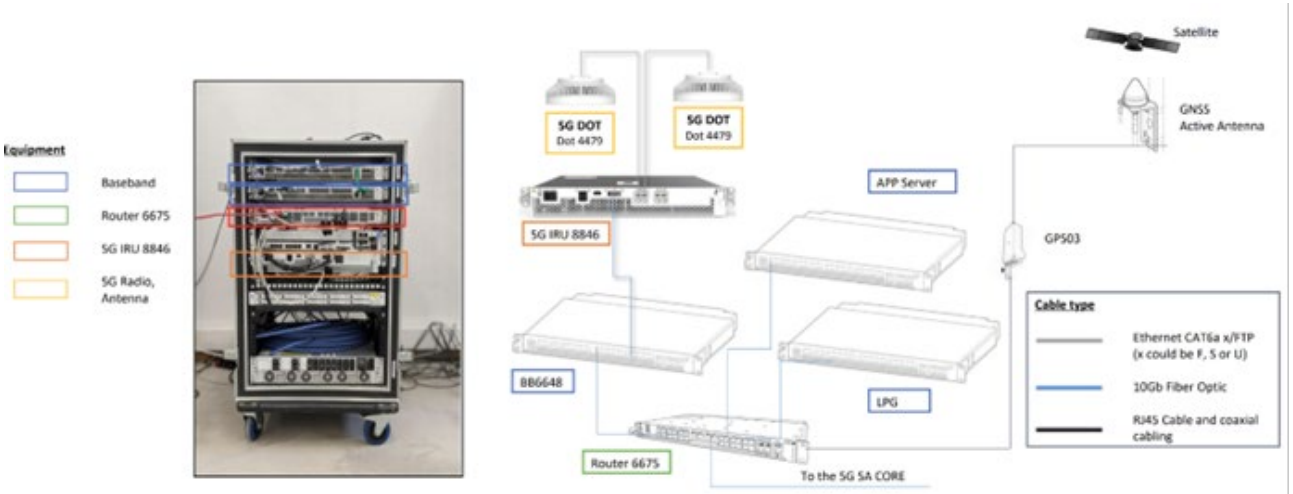


FIGURE 15. PHYSICAL NPN SYSTEM COMPONENTS

2.3.3 Results

The results are given by the fact that the capabilities required by the NPN and mentioned in section 3.4.2 of this document have been developed and implemented in a practical way. In this sense, with regards to:

Zero Touch:

NPN system delivered to CTTC counts with Zero-touch capabilities. Flight-rack was received as a Plug & Play box. A previous pre-integration work (factory configuration) together with self-configured

and self-integrated capabilities integrates the NPN network with the PN having a fully operational Private 5G Network in few minutes.

The work consisted of the identification of the graceful shutdown of the NPN components, together with the health check mechanism that should be incorporated during the bootup sequence.

When it comes to the self-configuration and self-integration part, Network Digital Twin (see next paragraph) plays a relevant role, as it need to work in conjunction with the physical twin (NPN system) to apply final NPN configuration and keeping track of the administrative state of the system (see Section 3.1.1 about UC1PoC1: *Zero-Touch distributed 3GPP network for delivering Non-public Networks* for details on ZT capabilities delivered and NPN boot up time).

Network Digital Twin:

NPN system delivered to CTTC counts with its own digital twin instance in the PNI site. Digital Twin allows to perform system Configuration, system Optimization and Experimentation with self-designed traffic models. Digital twin can even generate KPI Prediction scenarios and Data driven recommended configurations based on real data analysis.

Network Digital Twin platform reduces integration costs but also provides a closed loop for assurance and optimization of the NPN system according to the different use cases being supported. Digital Twin concept is explained in detail in Section *2.4* JOINT-K1.4: Network Digital Twin of this document.

Network Slicing:

Ericsson has developed and implemented a Network-slicing blueprint consisting of 5 slices with the following characteristics.

Slice uRLLC:
Intended for use cases where data timeliness is the relevant quality parameter. The concept, for this type of slice, focuses on a combination of low latency (single-digit milliseconds) and high reliability (prioritized traffic), with services being provided in-house by the Enterprise customer (with a local UPF + local application server (AS)).

Slice eMBB Local:
In general, intended for use cases related to human-centric and enhanced access to multimedia content, services, and data with selected balance of speed and capacity. Here the defining quality parameter is the data rate. In the NPN scenario, this slice also keeps a local user plane.

Slice eMBB Central (Internet):
Same performance profile, however, the concept focuses on having both, Control plane and User Plane rely on NFs located in the CSP central DC.

Slice Quarantine:
After anomaly detection (i.e., high energy consumption users), the network might decide to reallocate subscriber for further analysis.

Slice Monitoring:
For CSP use only, i.e., internal performance testing.

Following picture represent the slicing blueprint being used by CTTC Pelican NPN.

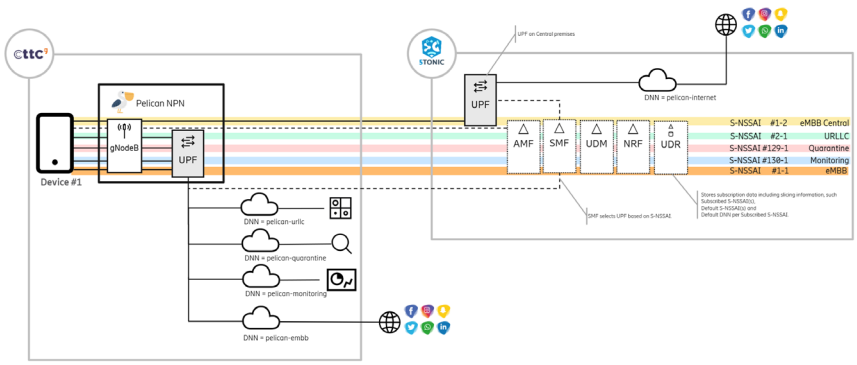


FIGURE 16. EXAMPLE OF CTTC NPN-PELICAN SLICING DESIGN VIEW

The initial design of the network slicing blueprint is based on a 1:1 relationship between DNN←→Slice. Any given subscriber is served by one slice at a given time, and the allowed slices (S-NSSAIs) are based on the slicing profile that is applied to that subscriber. Figure 17 below describes the "slices portfolio" and the characteristics of each defined slice that is applied when a subscriber is configured with the *pelican_slicing profile*.

Available Network Slices

Slice ID	S-NSSAI SST	S-NSSAI SD	DNN	MBR Uplink	MBR Downlink	5QI	Profile	ARP Priority Level	ARP Preempt Capability	ARP Preempt Vulnerability
pelican-embb	1	000001	pelican-embb	256 Mbps	1 Gbps	9	pelican_slicing_profile	10	NOT_PREEMPT	PREEMPTABLE
pelican-Monitoring	130	000001	pelican-Monitoring	256 Mbps	1 Gbps	9	pelican_slicing_profile	10	NOT_PREEMPT	PREEMPTABLE
pelican-Quarantine	129	000001	pelican-Quarantine	128 Kbps	128 Kbps	9	pelican_slicing_profile	15	NOT_PREEMPT	PREEMPTABLE
pelican-internet	1	000002	pelican-internet	256 Mbps	1 Gbps	9	pelican_slicing_profile	10	NOT_PREEMPT	PREEMPTABLE
pelican-URLLC	2	000001	pelican-URLLC	128 Mbps	512 Mbps	9	pelican_slicing_profile	5	MAY_PREEMPT	NOT_PREEMPTABLE

FIGURE 17. EXAMPLE NPN SLICING PROFILE CHARACTERISTICS.

While an initial slice selection for a User Equipment (UE) in a 5G network can be made during the registration procedure, the definitive slice for a particular service or data session is selected during the PDU (Packet Data Unit) session establishment procedure.

AMF, **SMF** and **Subscription data** (stored in **UDR NF**) play critical roles in network slice selection. See in figure below the Network slicing configuration points.

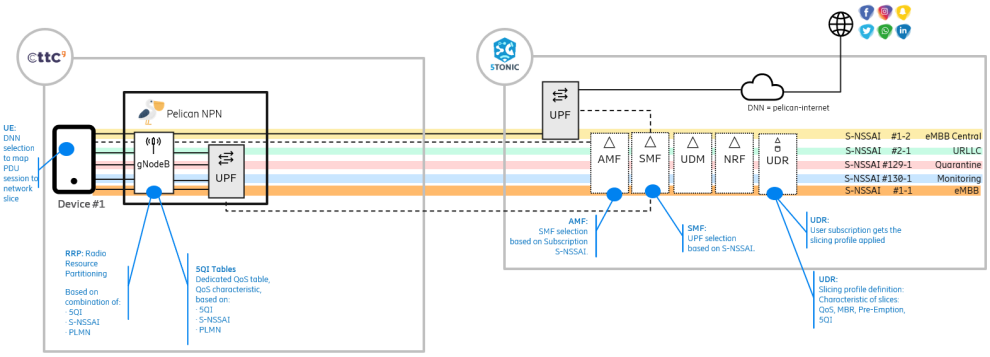


FIGURE 18. NETWORK SLICING CONFIGURATION POINTS.

The slices portfolio (see previous Figure 17) contains S-NSSAI [SST + SD] lists with its correspondent DNN as well as the QoS characteristics for each of the slices will be defined by CSP for a given Subscriber slicing profile.

The subscribers/IMSIs ranges for each Enterprise-customer will get applied the corresponding slicing profile at provisioning time.

- AMF selects an initial slice during the registration procedure based on UE subscription data and operator-defined rules.

- SMF selects the appropriate slice during PDU session establishment, considering the UE's requested slice, subscription data, and policy rules.

The AMF and SMF ensure that the network slice selection process still adheres to the operator's policies and provides the required services to the UE, maintaining the integrity and efficiency of the network slicing mechanism.

Exposure of Network APIs:

Finally, the NPN digital twin offers a variety of powerful exposure capabilities, making it easier for enterprise customers deploying small 5G private networks to design, deploy, and optimize 5G use cases. These capabilities go beyond the standard offerings, such as the 3GPP Network Exposure Function (NEF) specification.

While 3GPP NEF (3GPP TS 29.522) mainly focuses on providing Analytics APIs that deliver real-time data on network performance or specific subscribers to the API consumer (AF), this is just one part

of the picture. The NPN digital twin's system architecture provides a full portfolio of Network APIs that support customers through all stages of 5G use case development.

- During **Use case design**, tools like the Recommender API and KPI Prediction API help with planning.
- In the **Pre-Production stage**, the Configuration API and Traffic Experimentation API assist with testing and setup.
- Once in **production**, these APIs can work together to enable **data-driven optimization**, as demonstrated at the Ericsson Imagine Days corresponding to the PoCs described in Sections 3.2.2 and UC3 PoC3 described in 6GBLUR SMART deliverable E4.

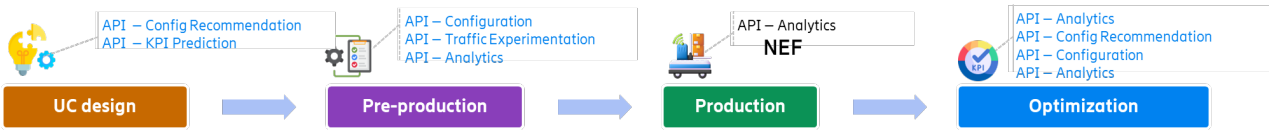


FIGURE 19. EXPOSURE APIS PORTFOLIO.

We can think of the changing needs of the Enterprise, where its connectivity needs to evolve over time, this means that the configuration of the private network must be adapted to new use cases or to optimizations of existing ones. From the CSP's point of view, when using these capabilities, it is very easy to offer customized connectivity services, in a short time and on a large scale. That is, to have massive customization capabilities for all types of Enterprise customers.

The APIs portfolio and some examples of the capabilities provided are detailed in Section 2.4 of this document.

2.3.4 Conclusions

Some final conclusions about PNI-NPN model developed:

1. We have deployed a real implementation of 5G-advanced and 6G standards, that's it a private network integrated with a public network in order to serve specific industrial or enterprise needs. This approach, i.e., the deployment of a hybrid system where we only deploy at the edge (customer site) what is critical for the correct operation of the use case, i.e., the radio and the user plane, already implies a cost reduction in the HW equipment as well as in the equipment footprint compared to having a complete 5G network onsite.

2. We think that the combination of complementary technologies, such as Exposure APIs, Edge computing and network slicing to implement NPN model together with Network Digital Twin is a potential game changer and disruptor for the Private 5G market.
    - It has the potential to boost the (otherwise stale) private 5G market, for both large enterprises and extending to middle-sized enterprises.
    - It reshapes the competition landscape putting the emphasis on digital-world key advantages: service predictability and knowledge-based growth.
3. The mere fact of being able to reduce from weeks to minutes the start-up time of the NPN system, as well as the time needed to find the most appropriate configuration for the NPN system based on the KPIs required by the enterprise customer, marks a significant advance in the cost savings and delivery times of this type of system.
4. On the other hand, having access to the real traffic patterns of the UEs through the analytics APIs allows the enterprise to fine-tune the network configuration and obtain an optimization of resources based on real data. The Machine Learning (ML) models implemented by the digital twin offer very accurate network performance prediction capabilities, allowing you to know in advance if the introduction of new use cases will affect performance, and thus anticipate the need for possible reconfigurations in your NPN system.
5. Vendors which can offer these types of capabilities can gain a huge competitive advantage over those competing only on price tag for delivering standard 5G private system.
6. Leveraging on this model and its capabilities, the CSPs can provide customized, efficient, and easy-to-use connectivity services to small and medium sized companies, at the same time, these companies can boost the implementation of 5G use cases at an affordable cost and in a reasonable timeframe.

2.4 JOINT-K1.4: Network Digital Twin

2.4.1 System design

An NPN Network Digital Twin is a digital representation of a physical instance of the Non-public part of a PNI-NPN set-up (refer Section 2.3). Several layers and systems make up the NDT platform pro-pose for supporting this concept, as depicted in Figure 20.

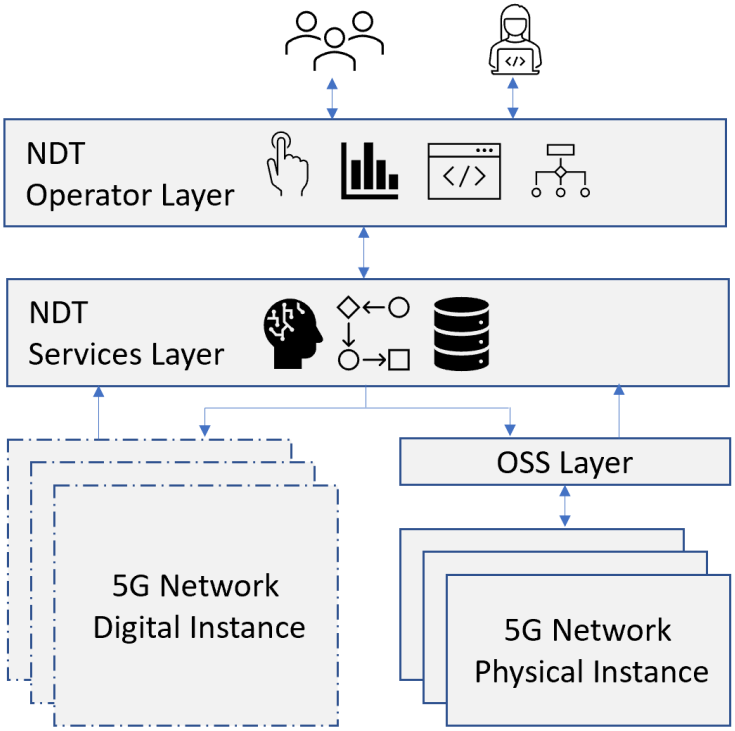


FIGURE 20. NETWORK DIGITAL TWIN PLATFORM

NDT Platform Components

- NDT Operator Layer: Is the layer offering the interfaces of the NDT platform to end users for either accessing information or composing customized services, over the platform.
- NDT Services Layer: Is the cornerstone of the NDT platform, including the model for NPN performance, and the intelligence for leveraging such model to be delivered through the core Use Cases of the NDT platform. Note that the NDT Services Layer provides (Northbound) the NDT Operator layer with APIs is in communication, leverages (Southbound) the operational support system (OSS) Layer for interacting with the twined 5G Network Digital Instances, and spawns and manages DT instances for each twined 5G physical instance.
- 5G Network Digital Instance(s): Are the different 5G Network models created by the NDT platform. These instances are generated with different parametrizations and using both actual performance data collected by the platform and Artificial Intelligence (AI) techniques.
- OSS Layer: The OSS is leveraged for the mission of collecting all the relevant data in terms of configuration and actual KPI, from the 5G Network Physical Instances, as well as for applying different configurations on the 5G Network Physical Instance.

- 5G Network Physical Instance: Consists of the actual part of the NPN that will be replicated by the NDT platform and is the objective part of the network to be monitored and controlled by the end user. Note that there could be more than one instance under the scope of the same NDT platform. For this project, the 5G Network Physical Instance is an NPN Portable System that consists in Radio Access Network (RAN) equipment and User Plane Function (UPF), integrated in a Flight Case system called "carrito".

2.4.2 Implementation

This section provides a description of the approach and relevant details of the implementation of the NDT platform, at several levels and from different complementary perspectives. First, the position of the NDT platform in the 3GPP architecture is explained. Then, the model for the Digital Twin is introduced. Next, components of the overall Digital Twin platform are outlined, followed by a review of NDT Use Cases. Finally, the NDT APIs and exposed capabilities are enumerated.

NDT platform in the context of 3GPP PNI-NPN

NPN, also referred to as a private network, is a network that it is intended for non-public (i.e. private) use. As mentioned in Section 2.3, 3GPP defines two major categories of NPNs: Standalone Non-Public Network (SNPN) and Public Network Integrated NPN (PNI-NPN). Our architecture is based on the Public Network Integrated NPN (PNI-NPN) category. The NDT platform described here is precisely and entirely focused on twining the gNB of, specifically, PNI-NPNs. NDT, in that specific applied context, can be a valuable tool for understanding and managing key aspects of performance and configuration of PNI-NPNs, such as:

- Simulating network scenarios: NDT can be used to test different PNI-NPN configurations and predict their impact on performance before real-world deployment.
- Monitoring and optimizing network performance: NDT can continuously monitor the PNI-NPN, providing insights for optimizing resource allocation, identifying potential issues, and ensuring service quality.

NDT is not a standardized element within 3GPP PNI-NPN specifications, and there is no standardization work ongoing or expected in that regards. So, in order to leverage the DT technology trend for bringing the services and benefits exposed to PNI-NPN scenarios, a strategy for integration of NDT in PNI-NPN scenarios is needed.

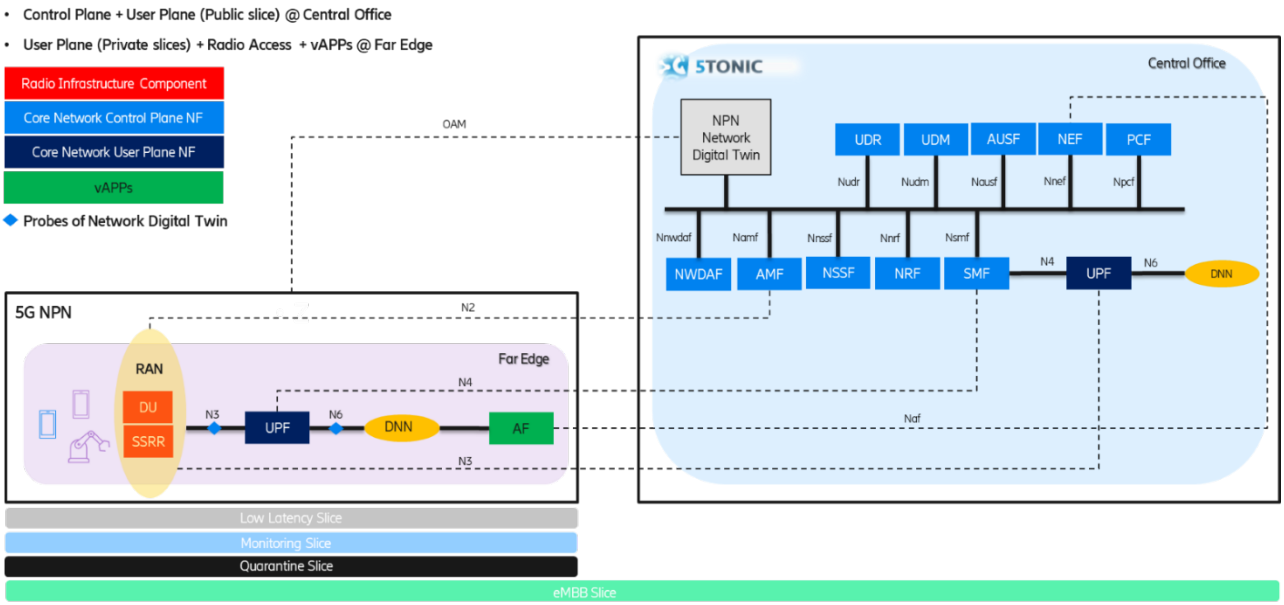


FIGURE 21. NETWORK DIGITAL TWIN PLATFORM IN THE FRAMEWORK OF 3GPP PNI-NPN SCENARIO

The proposed strategy, as represented in Figure 21, involves that NDT platform is i) connected to the NPN instances (on the left-hand side of the picture) to be twined, via Operation, Administration and Maintenance (OAM) interfaces, and, ii) integrated in the PN domain (on the right-hand side of the picture) as part of the 5GC Service Based Architecture. That duality enables, accordingly, in the first place, a privileged access to the NPN systems for remote monitoring and actuation, and, in the second place, smooth integration of a cost-effective central NDT platform within the PN instance, and the rest of centrally deployed NFs, as needed.

Network Digital Twin model for abstracting the NPN gNB system and its environment

The model created in the digital twin for abstracting the gNB system of an NPN and its environment encompasses the following aspects: Spectrum Assignment, Spectrum usage technique, Radio Unit type, gNB configurations, air conditions, KPIs and Traffic Models.

Spectrum assignment: 5G technology has standardized a large spectrum for the communications to operate divided into Low-Band (spectrum lower than <1GHz) Mid-Band (spectrum between 1GHz and 6GHz) and High-Band (spectrum higher than 6GHz, normally around GHz). The only band considered for the NDT platform implemented in the 6G BLUR project is Mid-Band in n78-B43 (Telefónica Spectrum: 3550MHz).

Spectrum usage technique: Is the strategy used in mobile communication to transmit the data in both forms of duplex. While FDD stands for Frequency-Division Duplex, TDD stands for Time-Division Duplex. 5G technology largely uses TDD strategy and support FDD in some cases. For NDT platform implemented in the 6G BLUR projects the model supported is TDD.

Radio Unit types: Based in the different radio types the NPN portable system could support 3 different radio models and are configurable in the Digital Twin:

- Small Cell DOT: Smallest 4x4 cell Ericsson that provides indoor coverage for small areas.
- Small Cell RRU4408: Indoor and outdoor 4x4 cell used for small-medium coverage areas.
- Macro Cell AIR6488: First 5G antenna 64x64 that could cover use cases, designed for large areas.

gNB configurations: Based in the configuration of cell and sectors of the NPN portable system supported, the model of the digital twin takes into consideration the following:

- Bandwidth: Width of the band in which the radio communication is performed. The values configurable in the model are: 20, 40, 80 and 100 MHz.
- TDD Pattern: In a Time division duplex communications, the pattern defines the number of downlink and uplink slots configured in the communications transmissions, the values configurable in the model are: 4:1 (DDDSU(10:2:2)) and 7:3 (DDDSUDDSUU(10:2:2)), in spite of the fact that the only configuration compatible with the current regulation is 4:1 (DDDSU(10:2:2)).
- Transmission power: Is the power configurable in the NPN portable system so the antenna could propagate the communication signal. The configurable values depend on the specific radio type selected. For the DOTs they would be 500 and 1000 mW, for the Remote Radio Unit (RRU) 4408 is goes from 1000mW to 10000mW in steps of 1000mW and for the AIR6488 we reduced the possibilities to 2400mW and 10000mW.
- MIMO: It is the method for multiplying the capacity of the radio link using multiple transmissions and receiving antennas to exploit multipath propagation. MIMO stands for multiple-input and multiple-output. It is a configuration that depends of the radio used and to get all capacity also in the end user device used. In the model when small cell radios are used, we can go up to 4x4 value (configuring value of 4), while using AIR6468 we can set 64x64 (configuring value of 0, max). Take into account the use of the maximum antennas also depends on the air conditions.
- Carrier aggregation: Is a technique used to increase the data rate per user, whereby multiple frequency blocks are assigned to the same user using different carriers in the same spectrum. For the NDT platform implemented in the 6G BLUR project single carrier is the one allowed.

Air conditions – radio link adaptation

The communication in a mobile network depends strongly on the air conditions. An external source of our system, transmitting in the same radio spectrum, could cause an interference that degrades the signal and affect KPIs as user data rate and latency.
The radio link algorithm is adapting, each millisecond, to the situation of the air interface, trying to stretch the channel at maximum to reach the highest radio signal quality. One of the aspects the radio negotiates with the end user device is the NR modulation and coding scheme also called: modulation and coding scheme (MCS). MCS defines the number of useful bits can be transmitted per resource element in a radio communication.
MCS depends on radio signal quality in wireless link:

- Good air conditions: good signal quality results in that higher MCS could be set and more useful bits can be transmitted with in a symbol.
- Bad air conditions: bad signal quality resulting in that lower MCS could be set and less useful data can be transmitted within a symbol.

Key performance indicators for model

The KPI refer to a set of quantifiable measurements used to quantify the performance of the NDT platform.

- User data rate (Network speed): Measures the number of data bits successfully transferred in one direction between specified reference points per unit time for the user service. Unit: Megabits per second (Mbps).
- Downlink Throughput: Measures the data download rate from the network towards the user.
- Uplink Throughput: Measures the data upload rate from the user towards network.
- Latency: Measures the time delay of a data packet to travel between the User Equipment (UE) and an application residing in the application server (AS) and it is measured in milliseconds (ms).
- Round-trip Time (RTT): Measures the duration between the transmission of a small data packet from the application layer connected to the UE and the successful reception at the application layer connected to the UPF, plus the equivalent time needed to carry the response back.
- One-Way Delay (OWD): Measures the duration between the transmission of a small data packet from the UE and the successful reception at the UPF, without considering the response back.
- Jitter: Jitter measures the variation in the delay of data packets as they are transmitted over a network. Jitter is measured in milliseconds (ms) and represents the absolute value of the difference between the One-Way Delay of two consecutive received packets belonging to the same stream.
- Packet loss rate: Measures the percentage of data packets that are lost during transmission over the network, including packets dropped, packets lost in transmission and packets received in wrong format.

- Reliability: The amount of sent packets successfully delivered to the user within the time constraint required by the targeted service, divided by the total number of sent packets. NOTE: the reliability rate is evaluated only when the network is available.
- Packet order rate: Measures the percentage and degree to which data packets are received in the correct order on a network.
- Energy consumed: Measures the energy consumed by the Physical Twin and is a key aspect for the energy saving objective of the NDT platform. The Unit is Watts per hour (Wh).

Traffic Models

A traffic model is a mathematical model of real-world network traffic, that from a service provider point of view is the problem of representing our understanding of dynamic demands by stochastic processes.
One of the main differences between a Public Network and Non-public Network are the devices that are connected to the network. A public network needs to handle a huge number of devices that has in common the similar use of the network, in other hand the NPN (and industrial factory) handles a considerable number of devices, but with very different use of the network.
As our NDT platform concept is focused on NPN, it is necessary to take in consideration the different uses of the network by different devices, so to correctly model the digital twin, we must consider the configuration of the physical twin and the traffic models the different device will have when using the network. This is necessary to model correctly the 5G Network Digital instance and to simulate different scenarios.
As described in [10], a traffic model could be classified as deterministic and non-deterministic, periodic and aperiodic, having the following subcases:

- Deterministic Periodic Traffic: where applications with this traffic type send messages at defined time interval. For example: Sensor-to-controller messages sent periodically between synchronized, time-sensitive applications.

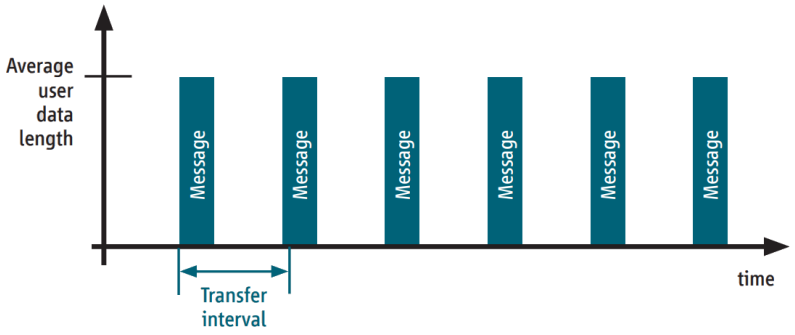


FIGURE 22. DETERMINISTIC PERIODIC TRAFFIC

- Deterministic Aperiodic Traffic: where applications with this traffic type send messages at not defined time interval and could vary depending on the particular use case. For example: Messages by process events are typically deterministic aperiodic. This traffic type consists of messages that are sent regularly but aperiodically, where there is no defined transfer interval between messages.

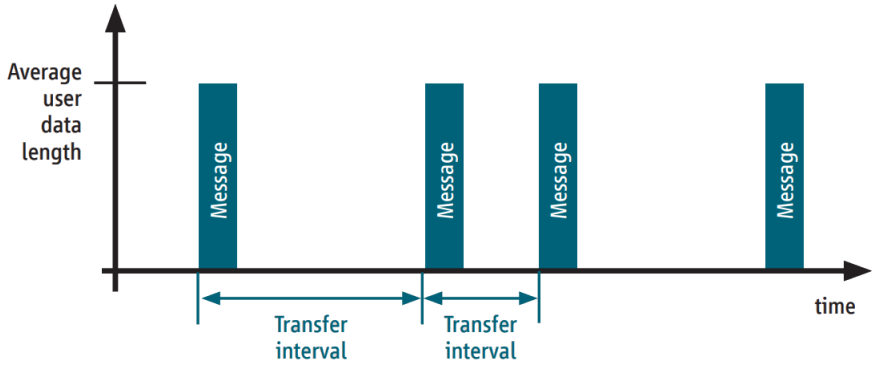


FIGURE 23. DETERMINISTIC APERIODIC TRAFFIC

- Non-deterministic (Burst Traffic) Periodic & Aperiodic Traffic: Messages of this traffic type are not expected to be received at the target endpoint within a specified time frame or within a specified time period between two consecutive messages.
  Burst traffic is characterized by a sequence of successive messages that are sent in a "burst", for example to transmit images. These messages can be periodic or aperiodic and are not expected to be received at the target endpoint within a specified time frame (transmission time).

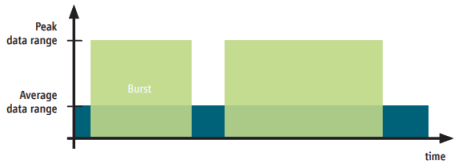


FIGURE 24. NON-DETERMINISTIC TRAFFIC

These different classifications would give the Traffic Type of the traffic model that will be use in the definition of the input.

A certain number of same devices could transmit the same traffic type in the same periodic time, and that should be consider in the description of the traffic model as the number of threads of the same traffic.
Transmission protocol is the communication standard that enables application programs to exchange messages over the network and is one of the most important things that must be taken in consideration when trying to characterize a traffic model. Depending on the type of requirements the communication needs will focus in reduce the transmission time, guaranteed the integrity of the data being communicated, etc.
Message size defines the dimension of the message the communication will use and determines the traffic model as will affect how the network will handle it.
Time interval is the amount of time between two given times and determines the periods the traffic is sent.
Traffic origin, destiny, and direction, determines how the communication is established between de devices, applications and is important in the traffic model description.
With the description of these traffic model characteristics, Table 2 covers all aspects that has to be taken in consideration when describing a traffic model. Currently, deterministic UDP, TCP, and ICMP are supported in the NPN system with the help of NDT. However, these supported capabilities can be extended based on use case requirements.

TABLE 2. TRAFFIC MODEL CHARACTERISTICS

Traffic Name	Traffic Type	Nº of threads	Transmission Protocol	Message size	Time interval	Traffic direction	Traffic origin and destination
[Name of the model]	[Deterministic, Non-deterministic, periodic, aperiodic, Bursty]	Number of strings of the same traffic model	[UDP, TCP, ICMP, RTSP, SSH, Telnet, HTTP]	[Bytes, KBytes, MBytes]	[ms]	Downlink or Uplink	UE/CPE->Server, Server->UE/CPE, UE/CPE->UE/CPE.

Based on Table 3 and analyzing the traffic that different stakeholders are using in 5TONIC for an industrial use case, the following table present an example of a generic traffic model that we aim to generate with the proposed NDT platform.

TABLE 3. TRAFFIC MODELS USED IN AN INDUSTRIAL USE CASE FOR THE PROPOSED NDT PLATFORM

Traffic Name	Traffic Type	Nº of threads	Protocol stack	Packet size	Time interval	Direction	Origin and destination

1xHD Video streaming	Bursty	1	UDP	1500	1sec	uplink	CPE->Server
8xHD Video streaming	Bursty	8	UDP	1500	1sec	uplink	CPE->Server
Remote Steering control	Aperiodic	1	TCP	Very small KB	N/A	Uplink	CPE->Server
Industrial manufactorig control	Periodic	1	UDP (Ethernet)	1440Bytes	10ms	Uplink	CPE->Server

NDT Machine Learning-Based Exposure APIs

Having described and explained the components of the NDT platform, this section explains how the NDT platform will address the needs of the user through the key use cases outlined in Section 3.4 of the 6G BLUR E3 JOINT deliverable Final E2E Architecture Design document. This is done through the NDT platform's exposure APIs. These APIs (demonstrated with real world use cases in UC2/PoC2 and UC3/PoC1) are a critical component that enables CSPs and enterprise customers to design, deploy, and optimize 5G connectivity use cases with unprecedented flexibility and precision. These APIs extend well beyond traditional capabilities, surpassing the standard Network Exposure Function (NEF) to provide a richer, more comprehensive set of tools for Non-Public Network (NPN) management and innovation. The comprehensive suite of exposure APIs offered by the NDT platform empowers CSPs and enterprises to fully leverage the capabilities of advanced 5G NPN. Through enhanced configurability, experimentation, network slicing, analytics, and optimization, these APIs facilitate a more agile and responsive network environment. Below is a detailed description of the key API implementations:

- **Configuration APIs:**
    - The Configuration APIs enable the application of customized NPN configurations based on specific connectivity use case requirements. This flexibility ensures that network configurations can be optimized for a wide variety of scenarios.

POST https:// {{EXPOSURE_API_IP}} /5tonic-exposure/v1/npn/Pelican/configuration Send

Params Authorization Headers (9) Body Scripts Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
{
    "name": "GNB_5TONIC_PELICAN_5G",
    "carrier_aggregation": false,
    "uplink_scheduler": "DS",
    "physical_sectors": [
        {
            "name": "S1",
            "cells": [
                {
                    "name": "PELICAN_5G_DOT-1",
                    "TDD_pattern": "DDDSUDDSUU(10:2:2)",
                    "modulation": false,
                    "spectrum_assignment": [
                        77,
                        78
                    ],
                    "sectors": [
                        {
                            "name": "1",
                            "bandwidth": 100,
                            "power": 1004,
                            "MIMO": 4
                        }
                    ],
                    "spectrum_usage_technique": "TDD",
                    "antennas": [
                        "RD4479B78L"
                    ],
                    "operational_state": "ENABLED"
                }
            ]
        }
    ]
```

Body Cookies Headers (5) Test Results 201 Created 3.87 s 226 B Save Response

Pretty Raw Preview Visualize JSON

```
"No new config provided"
```

FIGURE 25. CONFIGURATION APPLY EXAMPLE

- A complementary retrieval API allows enterprise users to access and review existing NPN configurations, facilitating a better understanding and management of network settings.

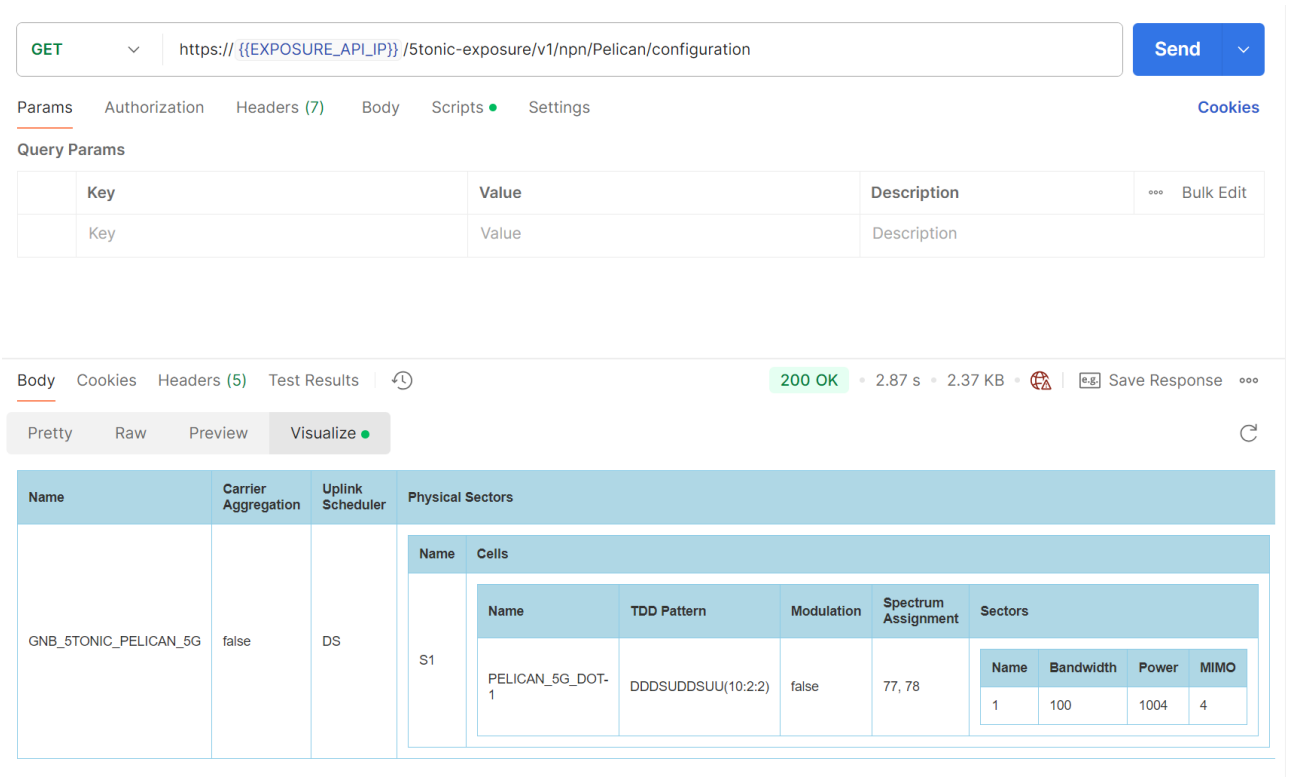


FIGURE 26. CONFIGURATION RETRIEVAL EXAMPLE

- **Experimentation APIs:**
  - These APIs support the scheduling and execution of various connectivity use case scenarios at predefined intervals. This capability is crucial for testing and validating network performance under different conditions.
  - The built-in traffic generator associated with these APIs can simulate diverse types of traffic, providing a robust environment for experimentation and refinement of network strategies.

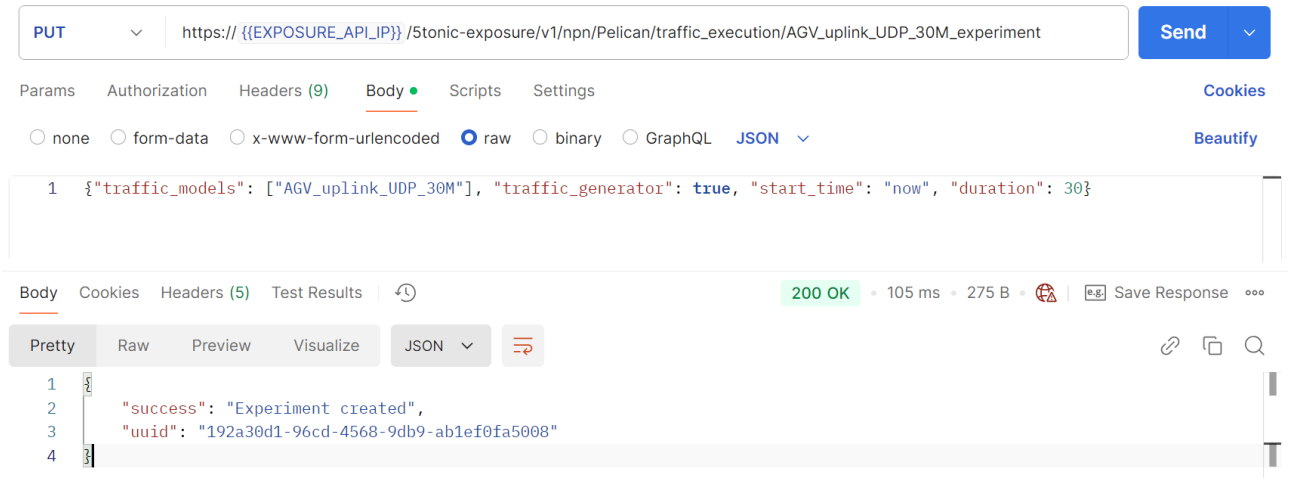


FIGURE 27. EXPERIMENTATION EXAMPLE

- **Network Slicing APIs:**
  - Network Slicing APIs provide comprehensive access to the network slicing capabilities of the NPN. Users can retrieve all available slices, access detailed slice definitions associated with an NPN, and update specific slice information as needed.
  - This level of control over network slicing allows for efficient allocation of network resources, ensuring optimal performance for different services and applications.
  - Example of **Network Slicing API in use**
    - ***Input***: NPN name
    - ***Output***: NPN network slicing details

GET http://10.3.3.41/5tonic-exposure/v1/npn/Pelican/slices Send

Params Authorization Headers (6) Body Scripts Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

Body Cookies Headers (4) Test Results 200 OK 236 ms 1.91 KB

Pretty Raw Preview Visualize

Slice ID	SNSSAI SST	SNSSAI SD	DNN	MBR Uplink	MBR Downlink	5QI	Profile
pelican-embb	1	000001	pelican-embb	256 Mbps	1 Gbps	9	pelican_slicing_profile
pelican-Monitoring	130	000001	pelican-Monitoring	256 Mbps	1 Gbps	9	pelican_slicing_profile
pelican-Quarantine	129	000001	pelican-Quarantine	128 Kbps	128 Kbps	9	pelican_slicing_profile
pelican-internet	1	000002	pelican-internet	256 Mbps	1 Gbps	9	pelican_slicing_profile
pelican-URLLC	2	000001	pelican-URLLC	128 Mbps	512 Mbps	9	pelican_slicing_profile
pelican-embb	1	000001	pelican-embb	128 Kbps	128 Kbps	9	pelican_quarantined_profile
pelican-Monitoring	130	000001	pelican-Monitoring	128 Kbps	128 Kbps	9	pelican_quarantined_profile
pelican-Quarantine	129	000001	pelican-Quarantine	128 Kbps	128 Kbps	9	pelican_quarantined_profile
pelican-internet	1	000002	pelican-internet	128 Kbps	128 Kbps	9	pelican_quarantined_profile
pelican-URLLC	2	000001	pelican-URLLC	128 Kbps	128 Kbps	9	pelican_quarantined_profile

FIGURE 28. SLICING API EXAMPLE

- **Analytical APIs:**
  - Analytical APIs deliver valuable insights into traffic patterns and Key Performance Indicators (KPIs), enabling CSPs and enterprises to monitor and analyze network performance effectively.
  - These insights are crucial for understanding user behavior, optimizing service delivery, and identifying areas for improvement in network performance.
  - Usage examples of this API can be seen in Sections 3.2.2 and UC3 PoC3 section available at 6GBLUR SMART E4 deliverable.
- **Optimization APIs (Prediction and Recommendations):**
  - The Optimization APIs leverage predictive analytics to assess new use cases against current demand and performance expectations. This foresight allows for proactive network management and resource allocation.
  - By providing data-driven recommendations, these APIs enable users to make informed decisions that enhance network efficiency and user experience.
  - Example of **Recommendation** API in use
    - ***Input***: UC Traffic Model + Network KPIs (i.e., Throughput, Latency) + Optimization type (Bandwidth, Throughput, Energy).

- ***Output***: NPN radio configurations fulfilling the Network KPIs for the given traffic model.

POST https:// {{EXPOSURE_API_IP}} /5tonic-exposure/v1/npn/Pelican/recommend Send

Params Authorization Headers (9) Body Scripts Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
{
  "optimizationInputData": [
    {
      "TrafficModel": {
        "protocol_stack": "UDP",
        "bandwidth": "50M",
        "traffic_direction": "uplink"
      },
      "MinThroughput": "50",
      "MaxOWDMean": "20"
    },
    {
      "TrafficModel": {
        "protocol_stack": "UDP",
        "bandwidth": "5M",
        "traffic_direction": "downlink"
      },
      "MinThroughput": "5",
      "MaxOWDMean": "20"
    }
  ],
  "optimizationType": "THROUGHPUT_OPTIMIZATION"
}
```

Body Cookies Headers (5) Test Results 200 OK 19.60 s 10.33 KB Save Response

Pretty Raw Preview Visualize

Recommend API Inputs: KPI's requirements

Recommendation Criteria	Uplink KPI Tput (Mbps)	Uplink KPI OWD (ms)	Uplink KPI Power (W)	Downlink KPI Tput (Mbps)	Downlink KPI OWD (ms)	Downlink KPI Power (W)
THROUGHPUT_OPTIMIZATION	50	20	N/A	5	20	N/A

Recommend API Outputs: Configurations sorted by THROUGHPUT_OPTIMIZATION

Priority	Bandwidth (MHz)	TDD Pattern	Uplink Max ach. Tput (Mbps)	Uplink Pred. OWD (ms)	Uplink Pred. Power Con. (W)	Downlink Max ach. Tput (Mbps)	Downlink Pred. OWD (ms)	Downlink Pred. Power Con. (W)
1	100	DDDSUDDSUU(10:2:2)	97.081	12.615	180.488	833.713	5.536	262.758
2	80	DDDSUDDSUU(10:2:2)	81.972	12.499	191.728	633.796	6.012	277.404
3	100	DDDSU(10:2:2)	73.013	13.288	174.782	833.945	5.779	286.529

FIGURE 29. RECOMMENDATION API EXAMPLE.

- Example of **KPI Prediction API** in use
  - **Input**: NPN radio configuration + Traffic Model
  - **Output**: Achievable KPIs (i.e., Throughput, Latency, Power Consumption)

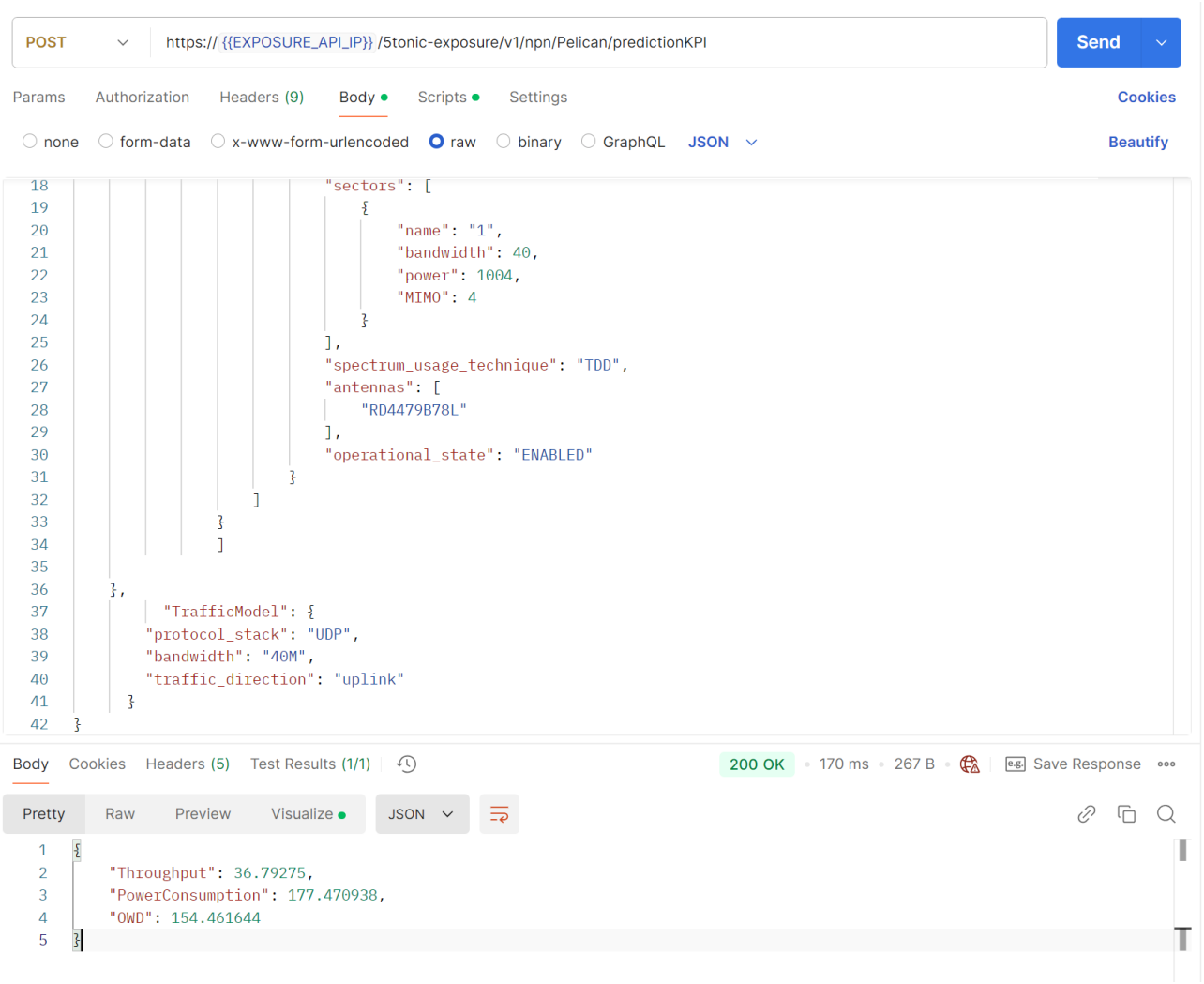


FIGURE 30. PREDICTION API EXAMPLE.

2.4.3 Results

The development and implementation of the Network Digital Twin (NDT) platform have demonstrated significant advancements in the management and optimization of Public Network Integrated Non-Public Networks (PNI-NPNs). The platform acts as a central orchestrator for zero-touch configuration, health checks, and experimentation, offering a suite of key functionalities integral to the efficient operation of NPNs:

- **Discovery and Registration**: The platform effectively discovers and registers NPNs with the Public Network, streamlining the initial setup process and ensuring all network components are accounted for within the digital twin environment.
- **Configuration Generation and Application**: Utilizing low-level design templates, customer use cases, and KPI requirements, the NDT platform generates initial configurations for NPN components, which are then systematically applied. This automation reduces human intervention, minimizes errors, and accelerates deployment times.
- **Health Checks and Validation**: The platform performs essential health checks and service validation tests to ensure network components function optimally, providing a reliable foundation for NPN operations.
- **Comprehensive Digital Twin Functionality**: By integrating performance monitoring, analysis, recommendations, and actuation, the NDT platform acts as a comprehensive one-stop shop for managing NPNs. This positions it as an innovative solution, ahead of potential standardization efforts expected in the mid-term.
- **Scenario-Specific Modelling**: The platform specifically addresses PNI-NPN scenarios by modelling NPN technology and performance against typical usage patterns and environmental conditions. This targeted approach allows for precise optimization and learning, which generic analytics tools cannot achieve.
- **Performance Optimization Capabilities**: Special network configuration capabilities are incorporated for performance optimization, addressing the unique demands of NPNs compared to Public Networks (PNs).
- **Tri-Model Approach**: The platform leverages three complementary performance models—theoretical, empirical, and machine learning (ML)—to provide a robust and adaptive understanding of network performance. These models combine theoretical expectations, lab-measured performance, and continuously improved insights from synthetic experimentation and real-world use case execution.
- **Integration with 5G Core**: By integrating with the 5G Core architecture, the NDT platform gains privileged access to network functions and resources, enhancing its ability to interact effectively with other network elements and application functions.
- **Versatile Deployment Strategies**: The platform supports both autonomic closed-loop strategies and cooperative approaches with open API exposure, enabling flexible integration with upper-layer functions and smart ML-based optimization.

- **ML-Based Exposure APIs**: The platform exposes APIs that can be used by Communication Service Providers or enterprise customers to easily design, deploy, and optimize 5G connectivity use cases. These APIs offer exposure capabilities that extend well beyond the standard, surpassing a Network Exposure Function (NEF).
    - **Configuration APIs**: The Configuration API helps apply various customized NPN configurations based on connectivity use case requirements. The retrieval API can help retrieve and view the NPN configurations. The Configuration API response takes around 30 seconds, while the Configuration retrieval API response time is around 3 seconds.
    - **Experimentation APIs**: Experimentation APIs help schedule various connectivity use case scenarios at specified intervals, and the built-in traffic generator can help generate and receive different types of supported traffic. The Experiment creation API response takes around 95 milliseconds, while the Experiment retrieval API response time is around 155 milliseconds.
    - **Network Slicing APIs**: These APIs retrieve all available slices, provide slice definitions associated with an NPN, and update specific slice information on the NPN. Slicing retrieval API response time is around 250 milliseconds.
    - **Analytical APIs**: Analytical APIs provide insights into traffic patterns and KPI requirements. The response time of the Analytical API varies depending on the duration of the data you are retrieving for analysis.
    - **Optimization APIs (Prediction and Recommendations)**: These APIs provide predictions to assess new use cases against demand and performance expectations, enabling data-driven decisions to optimize resource usage. The Recommendation API response takes around 20 seconds, while the Prediction API response time is around 170 milliseconds.

2.4.4 Conclusions

The implementation of the Network Digital Twin (NDT) platform marks a transformative advancement in the management and optimization of 5G Non-Public Networks (NPNs). Through its robust suite of features, the platform effectively orchestrates zero-touch configuration, health checks, and experimentation, enhancing the operational efficiency of PNI-NPNs.
The platform's ability to discover and register NPNs within the Public Network seamlessly integrates network components into the digital twin environment, streamlining the initial setup process. By

automating configuration generation and application through low-level design templates and specific customer use cases, the NDT platform minimizes human intervention, reduces errors, and accelerates deployment timelines, thus providing a reliable operational foundation.
One of the most significant achievements of the NDT platform is its comprehensive digital twin functionality, which includes performance monitoring, analysis, recommendations, and actuation. This positions the platform as an innovative, all-in-one solution that is ahead of anticipated standardization efforts. The scenario-specific modelling allows for precise optimization and learning tailored to PNI-NPN scenarios, offering insights and performance enhancements that generic analytics tools cannot achieve.

Moreover, the platform's integration with the 5G Core architecture grants it privileged access to network functions, enhancing its interoperability and effectiveness. The versatile deployment strategies, including autonomic closed-loop strategies and cooperative approaches with open API exposure, ensure that the platform can adapt to evolving network requirements and technological advancements.

The inclusion of specialized APIs—ranging from ML-based exposure to configuration, experimentation, network slicing, and optimization—further extends the platform's capabilities. These APIs empower CSPs and enterprise customers to design, deploy, and optimize 5G connectivity use cases with greater precision and insight.

In summary, the Network Digital Twin platform provides a pioneering service that stands at the forefront of telecom innovation. By delivering comprehensive, scenario-specific solutions, it empowers organizations to fully harness the potential of their 5G Non-Public Networks, setting a new standard for network management and optimization. This innovative approach not only enhances current operational efficiency but also establishes a foundation for future advancements and standardizations in digital twin technologies.

2.5 JOINT-K2.1: Transport network optimization

2.5.1 System design

As outlined in Deliverable E3, this key concept focuses on optimizing the transport network within an ORAN-based mobile network deployment. The goal is to enable network slicing in the transport by developing a specialized controller.

Currently, the industry lacks a standardized approach for defining a common Transport Slice Controller component capable of facilitating the provision of connectivity services in the form of slices. This component would manage slice requests and the associated procedures.

The proposed system, called Network Slice Controller (NSC), is illustrated in Figure 31.

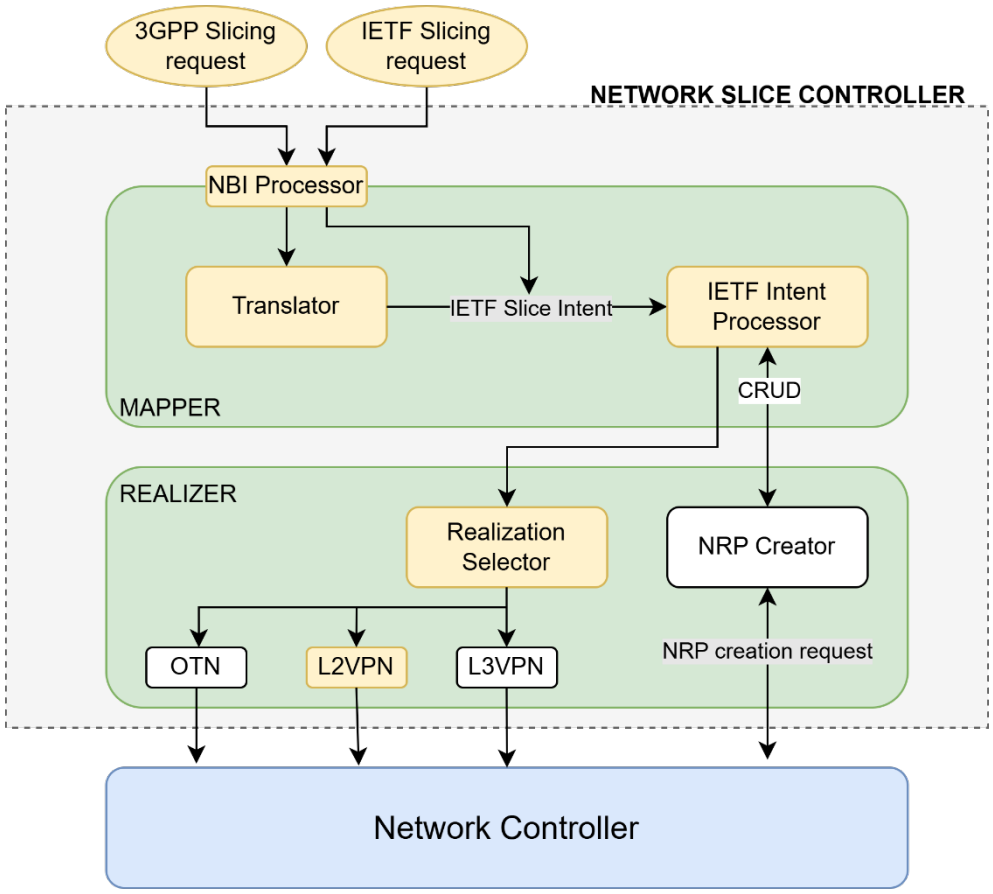


FIGURE 31. NETWORK SLICE CONTROLLER ARCHITECTURE

The components of this system, introduced in Deliverable E3, are further developed here:

- **Mapper**: it processes client network slice requests and correlates them with existing slices. The workflow starts when a slice request reaches the Northbound Interface (NBI), where the NBI Processor component inspects the request to determine whether to forward it directly to the IETF Intent Processor or send it to the Translator component. Requests coming from 3GPP are converted to IETF slice requests. Once the request is translated, the feasibility of the slice realization is assessed. Realizing a slice requires an existing Network Resource Partition (NRP) that meets the slice's requirements, which may not be available at the time of the request. If no suitable NRPs exist for instantiating the slice, the IETF Intent Processor instructs the Realizer to create a new NRP via interacting with the network controllers responsible for the

transport network handled by the NSC. This process iterates until the IETF Intent Processor confirms that the slice can be realized.

- NBI Processor: located at the NSC's NBI, it processes two types of slicing requests:
  - 3GPP slicing request: based on the 5G Network Resource Model (ETSI TS 28 541 V17.11.1) [32] (an example can be found in the Annex 6.1), this request consists of several modules where the network slice is defined in the RAN, transport, and core of the ORAN-based mobile network. Regarding the transport network, this model links the RAN and Core Network entities to the transport network through the *EpTransport* and *EpRP* objects. The *EpRP* object defines the logical association between two network entities (e.g., CU and DU), while the *EpTransport* object shows how these entities connect to the transport network. The NBI Processor inspects the model, looking for these two objects, which triggers a request to the Translator component.
  - IETF slicing request: it is based on the IETF Network Slice Service YANG data model [33], which defines a technology-agnostic slicing intent request for the transport network. The NBI Processor looks for the key module in this model, *ietf-network-slice-service*, for further processing.

  Regardless of the format, if multiple slices are included in the same request, each slice is processed independently.
- Translator: this component manages the 3GPP slicing requests coming from the NBI Processor and converts them into an IETF slice intent request. Figure 32 shows how these parameters are mapped and translated.
- IETF Intent Processor: it receives the IETF slice request and checks whether it can be implemented. To do so, it needs data about the NRPs in the network. This information is retrieved from an external module (beyond the scope of this document), which provides feedback on the slice's feasibility. As previously mentioned, if no NRPs are available, the Intent Processor requests additional resources from the network controller until the slice is realized. For simplicity in the tests conducted within the project, it is assumed that only one NRP, corresponding to the entire network, is available and accessible.

- **Realizer**: the Realizer module implements each slice by interacting with specific network controllers. It receives requests from the Mapper and determines the technologies to be used to instantiate the slice based on the selected NRP. For instance, in the project tests, Layer 2 VPN technology is employed to implement network slices. The Realizer generates a request for

the network controller to establish a Layer 2 VPN between two Source Demarcation Points (SDPs), adhering to the slice request's requirements.

- Realization Selector: this module decides which technology will be used to realize the slice. In the project tests, Layer 2 VPN is the chosen technology.
- L2VPN: This component serves as the bridge between the NSC and the transport network controller. In the 6G BLUR project, TeraflowSDN [48] is the controller used for this purpose. The L2VPN module retrieves the slice request parameters and generates a descriptor to provision the L2VPN service in the Teraflow domain. An example of this descriptor can be found in the Annex 6.2.

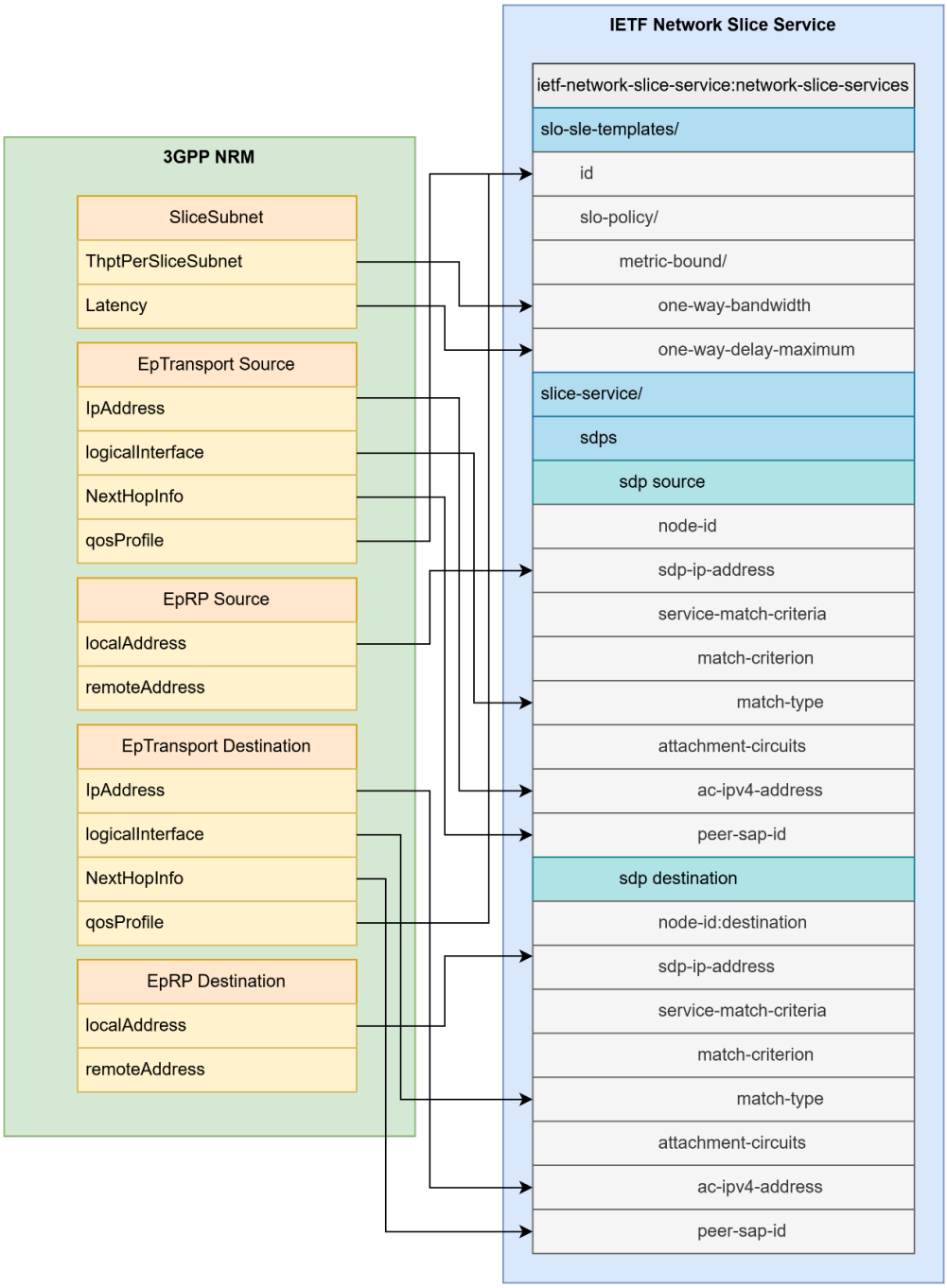


FIGURE 32. MAPPING BETWEEN 3GPP NETWORK ENTITIES AND IETF NETWORK SDPS

2.5.2 Implementation

The NSC works as a python application with an API listening for POST requests at "/intent" to submit slice requests and DELETE requests to remove all slices, as depicted in Figure 33.

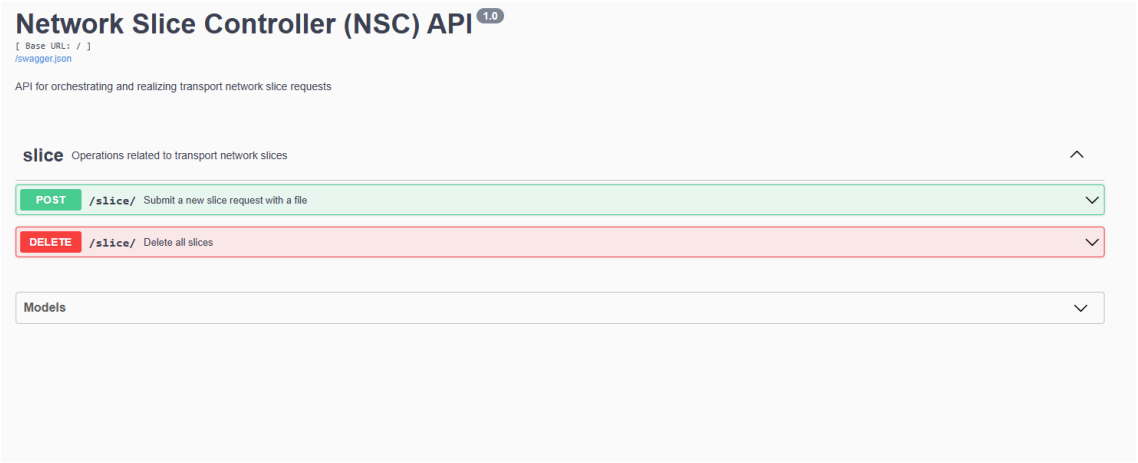


FIGURE 33. NETWORK SLICE CONTROLLER API

Testing the NSC requires a transport network. Eve-ng [22] software is used as the transport network emulation platform focused on network devices. This software is fed with Cisco xrv9000 [23] images, as well as Linux images for the purpose of these tests. The Cisco images are used to emulate transport network routers, following the topology depicted in Figure 34.

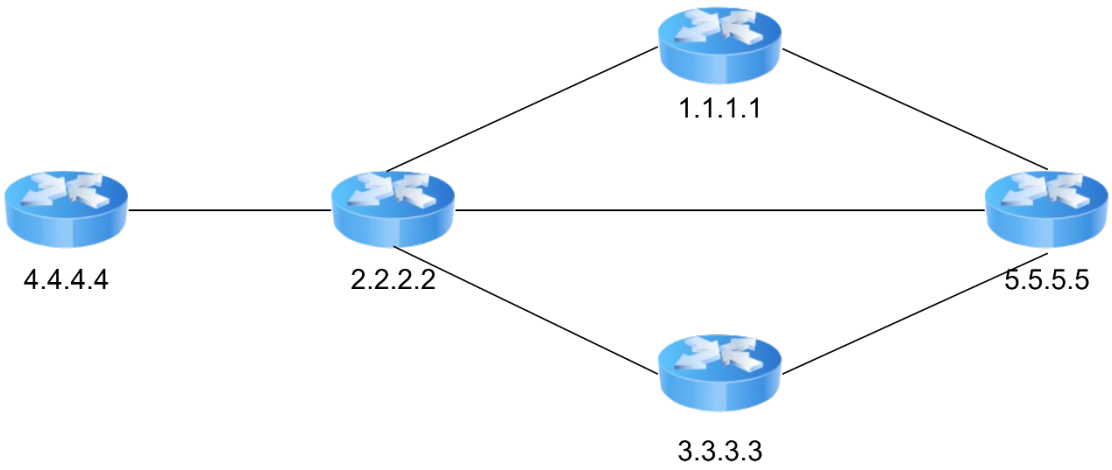


FIGURE 34. EMULATED TRANSPORT NETWORK TOPOLOGY

As the objective is establishing L2VPNs in the routers through different tunnels to provide different slices, these routers require previous configurations:

- An Interior Gateway Protocol (IGP) activated. In this case, OSPF [24], for routing traffic through the routers.
- MPLS Label Distribution Protocol (LDP) [25], required for enabling an MPLS-TE [26] topology.
- Resource ReServation Protocol (RSVP) [27] activated in routers port interfaces.
- MPLS-TE topology, required for creating traffic engineering tunnels in the topology.
- Traffic engineering tunnels, through the three possible paths for traffic between endpoints.

This topology is retrieved from Teraflow SDN controller [28] by using its module *BGPLS Speaker*, which uses the BGP protocol to extract the exposed topology information from the routers. It uses an added router acting as route reflector, with the BGP protocol activated and Teraflow SDN controller as a neighbour.

Once this is preconfigured, the topology is ready. When handling a request, the NSC configures a layer 2 VPN in the following way:

1. Configures sub interfaces in the endpoints for Layer 2 transport, associated with the VLAN specified in the request.
2. Creates a VPN between endpoints, setting up a pseudowire associated to one of the tunnels previously established in the topology.

Finally, it is worth to mention that for being integrated in an ORAN architecture, the transport needs to handle data and control packets from the RAN components. This requires mapping specific headers such as SCTP and GTP-U to assign them to VLAN values to be handled in the transport. Consequently, OpenvSwitch [29] switches are used due to their compatibility with SDN controllers based on Openflow [30] protocol, such as Ryu [31], which implement rules to tag and untag packets with VLAN identifiers at the ingress and egress of the network respectively.

In particular, data plane traffic is assigned to two fixed VLAN values (B and C), each one corresponding to a different slice, depending on the source and destination IP values of traffic, since each slice goes through a different UPF. This allows user data plane traffic differentiation, forwarding traffic over two different paths depending on the destination of traffic. Furthermore, control plane traffic gets another different VLAN value (A). Therefore, every control plane SCTP packet is assigned to VLAN A. At the egress, VLAN tags are removed. This way, traffic transverse the transport network with specific-path policies in a transparent way for the O-RAN entities and the used software.

The whole transport network implementation/emulation architecture is depicted in Figure 35.

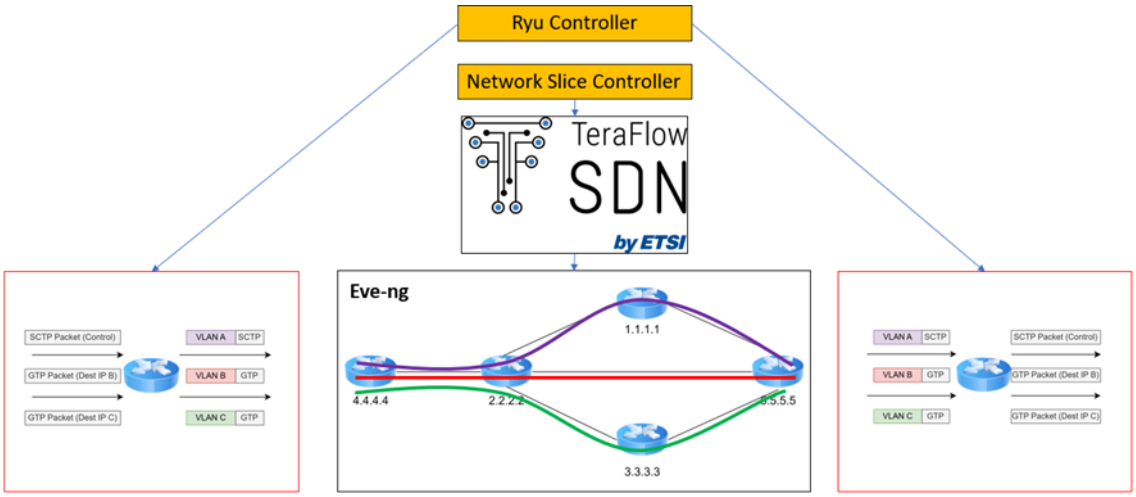


FIGURE 35. TRANSPORT IMPLEMENTATION ARCHITECTURE

2.5.3 Results

Several tests are performed in order to check the correct functionality of all components:

1. NSC requests processing

   The NSC is tested with slice requests comprising three different flows: a control plane flow, and two data plane flows, one of them with more demanding requirements. These requests, expressed in 3GPP TS 28541 NRM terms are shown in the Annex 6.1.

   When one of these requests is sent, the NSC API answers with the following body response:


```
{
 "status": "success",
 "code": 200,
 "slices": [
  {
   "id": "slice-service-366e05b7-34e6-4597-9e28-e580abaeda71",
   "source": "10.60.60.105",
   "destination": "10.60.11.3",
   "vlan": "100",
   "bandwidth(Mbps)": 2,
   "latency(ms)": 20
  }
```

],
"setup_time": 16,82

The answers shows that the request has been correctly processed, and the slice has been configured between the specified IPs, assigned a VLAN identifier and satisfying the requested requirements. From this result we can infer via the variable *setup_time* that the time it took to configure the slices in the transport has been around 17 seconds.

Figure 36 and Figure 37 show the mean setup time of a slice request for each of the paths, as well as for the three paths aggregated. Results show that it is more efficient to deploy all slices together rather than doing it separately. This behaviour is due to the time it takes for enabling a channel for configuring the slice in the topology. This behaviour is due to the fact that each request involves opening a communication channel between the controller and the devices, which adds additional time. As a result, individual requests take longer compared to the aggregate time of the three requests, since the time required to open and close the communication channel is only performed once.

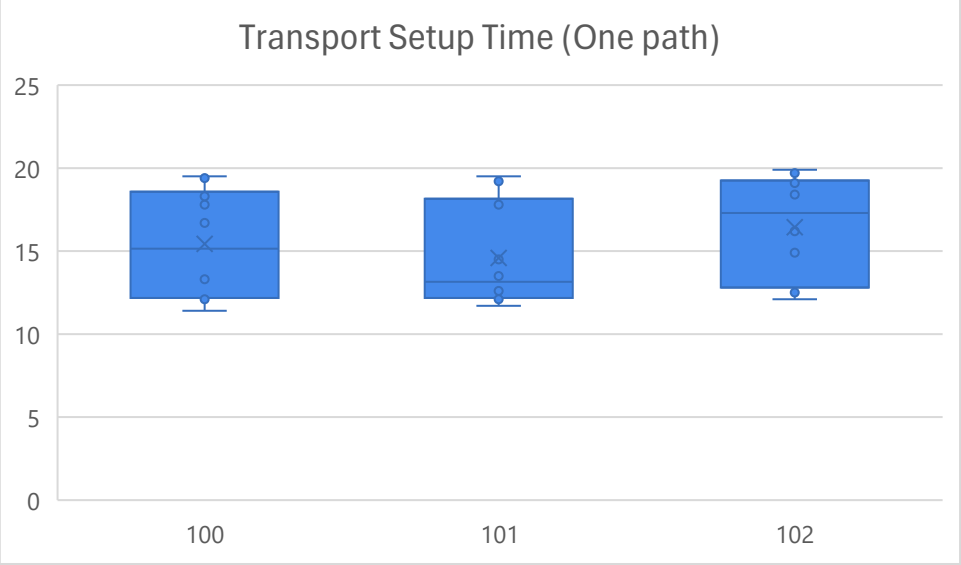


FIGURE 36. SETUP TIME FOR ONE PATH

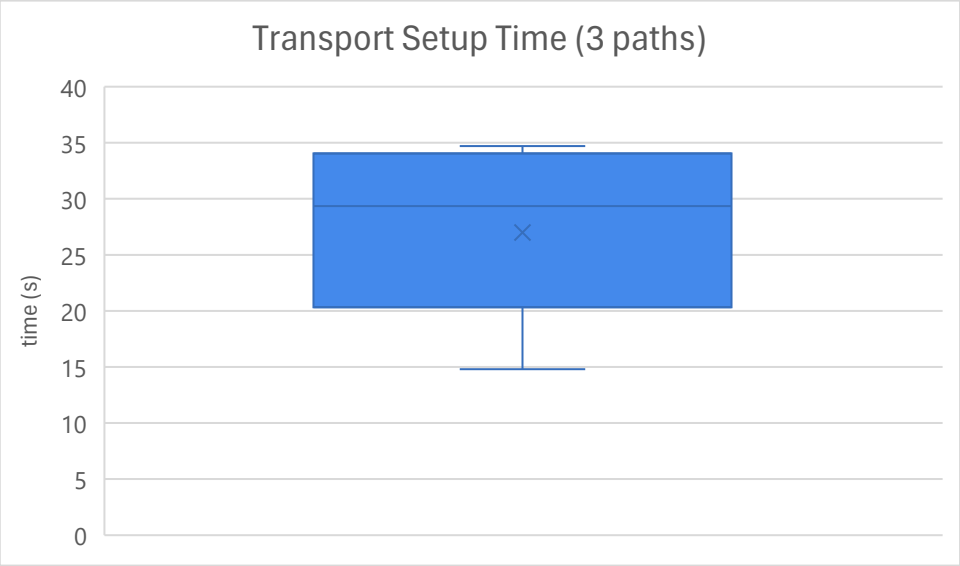


FIGURE 37. SETUP TIME FOR 3 PATHS

2. Teraflow configuration

Teraflow Controller web interface, depicted in Figure 38, shows that three L2VPN services have been deployed.

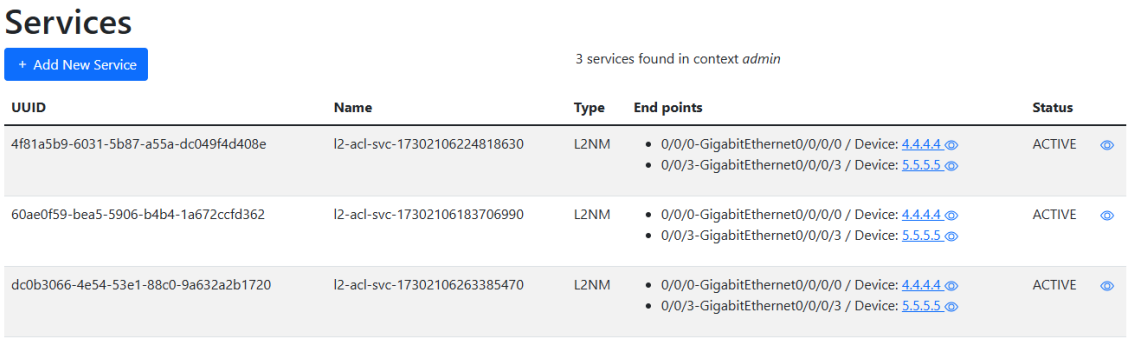


FIGURE 38. TERAFLOW GUI SHOWING CURRENT SERVICES

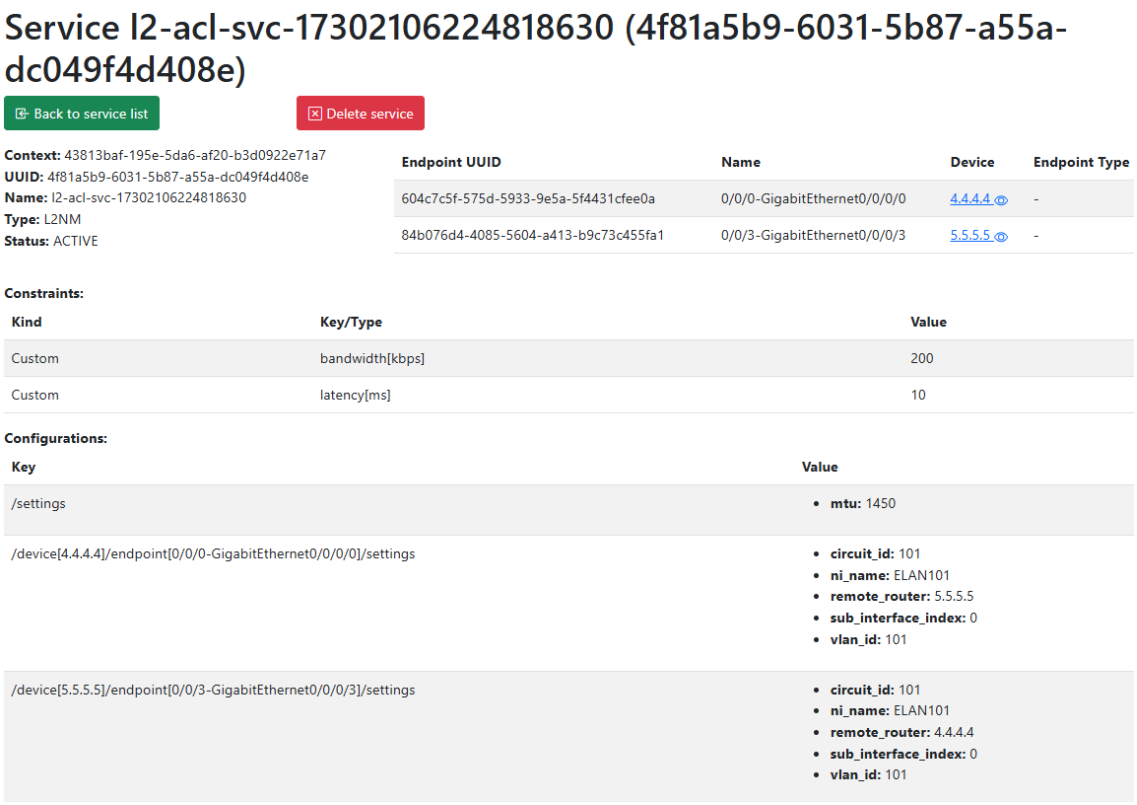


FIGURE 39. TERAFLOW SERVICE IN DETAIL

In Figure 38, the three services are deployed between endpoints. Figure 39 shows one of the services more in detail, where VLAN and requirements are specified.

3. Cisco Router tests: ping, latency, jumps

   In this section, transport routers configurations are checked by accessing the CLI of routers at the endpoints.

```
RP/0/RP0/CPU0:ios#sh l2vpn xconnect
Thu Sep 26 06:45:45.183 UTC
Legend: ST = State, UP = Up, DN = Down, AD = Admin Down, UR = Unresolved,
        SB = Standby, SR = Standby Ready, (PP) = Partially Programmed,
        LU = Local Up, RU = Remote Up, CO = Connected, (SI) = Seamless Inactive

XConnect                   Segment 1                       Segment 2
Group      Name       ST   Description            ST       Description            ST
------------------------   ------------------------------  ------------------------------
l2vpn_vpws_group_example
           ELAN100    UP   Gi0/0/0/1.100          UP       5.5.5.5         100    UP
------------------------------------------------------------------------------------------
l2vpn_vpws_group_example
           ELAN101    UP   Gi0/0/0/1.101          UP       5.5.5.5         101    UP
------------------------------------------------------------------------------------------
l2vpn_vpws_group_example
           ELAN102    UP   Gi0/0/0/1.102          UP       5.5.5.5         102    UP
------------------------------------------------------------------------------------------
```

FIGURE 40. ROUTER 4.4.4.4 WITH DEPLOYED VPNS

```
RP/0/RP0/CPU0:ios#sh l2vpn xconnect
Thu Sep 26 10:08:07.147 UTC
Legend: ST = State, UP = Up, DN = Down, AD = Admin Down, UR = Unresolved,
        SB = Standby, SR = Standby Ready, (PP) = Partially Programmed,
        LU = Local Up, RU = Remote Up, CO = Connected, (SI) = Seamless Inactive

XConnect                   Segment 1                       Segment 2
Group      Name       ST   Description            ST       Description            ST
------------------------   ------------------------------  ------------------------------
l2vpn_vpws_group_example
           ELAN100    UP   Gi0/0/0/2.100          UP       4.4.4.4         100    UP
------------------------------------------------------------------------------------------
l2vpn_vpws_group_example
           ELAN101    UP   Gi0/0/0/0.101          UP       4.4.4.4         101    UP
------------------------------------------------------------------------------------------
l2vpn_vpws_group_example
           ELAN102    UP   Gi0/0/0/1.102          UP       4.4.4.4         102    UP
------------------------------------------------------------------------------------------
```

FIGURE 41. ROUTER 5.5.5.5 WITH DEPLOYED VPNS

Figure 40 and Figure 41 show that the three L2VPNs are up.

At this point, three KPIs are measured, connectivity, latency and hop number, through every path.

3.1. Connectivity and latency

3.1.1. Tunnel path with VLAN 100

```
RP/0/RP0/CPU0:ios#ping mpls pseudowire 5.5.5.5 100 repeat 100
Tue Apr  8 11:14:32.986 UTC

Sending 100, 100-byte MPLS Echos to 5.5.5.5 VC: 100,
     timeout is 2 seconds, send interval is 0 msec:

Codes: '!' - success, 'Q' - request not sent, '.' - timeout,
  'L' - labeled output interface, 'B' - unlabeled output interface,
  'D' - DS Map mismatch, 'F' - no FEC mapping, 'f' - FEC mismatch,
  'M' - malformed request, 'm' - unsupported tlvs, 'N' - no rx label,
  'P' - no rx intf label prot, 'p' - premature termination of LSP,
  'R' - transit router, 'I' - unknown upstream index,
  'X' - unknown return code, 'x' - return code 0

Type escape sequence to abort.

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Success rate is 100 percent (100/100), round-trip min/avg/max = 27/28/35 ms
```

FIGURE 42. CONNECTIVITY THROUGH TUNNEL 1

3.1.2. Tunnel path with VLAN 101

```
RP/0/RP0/CPU0:ios#ping mpls pseudowire 5.5.5.5 101 repeat 100
Tue Apr  8 11:15:31.256 UTC

Sending 100, 100-byte MPLS Echos to 5.5.5.5 VC: 101,
     timeout is 2 seconds, send interval is 0 msec:

Codes: '!' - success, 'Q' - request not sent, '.' - timeout,
  'L' - labeled output interface, 'B' - unlabeled output interface,
  'D' - DS Map mismatch, 'F' - no FEC mapping, 'f' - FEC mismatch,
  'M' - malformed request, 'm' - unsupported tlvs, 'N' - no rx label,
  'P' - no rx intf label prot, 'p' - premature termination of LSP,
  'R' - transit router, 'I' - unknown upstream index,
  'X' - unknown return code, 'x' - return code 0

Type escape sequence to abort.

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Success rate is 100 percent (100/100), round-trip min/avg/max = 7/7/10 ms
```

FIGURE 43. CONNECTIVITY THROUGH TUNNEL 2

3.1.3. Tunnel path with VLAN 102

```
RP/0/RP0/CPU0:ios#ping mpls pseudowire 5.5.5.5 102 repeat 100
Tue Apr  8 11:16:02.418 UTC

Sending 100, 100-byte MPLS Echos to 5.5.5.5 VC: 102,
      timeout is 2 seconds, send interval is 0 msec:

Codes: '!' - success, 'Q' - request not sent, '.' - timeout,
  'L' - labeled output interface, 'B' - unlabeled output interface,
  'D' - DS Map mismatch, 'F' - no FEC mapping, 'f' - FEC mismatch,
  'M' - malformed request, 'm' - unsupported tlvs, 'N' - no rx label,
  'P' - no rx intf label prot, 'p' - premature termination of LSP,
  'R' - transit router, 'I' - unknown upstream index,
  'X' - unknown return code, 'x' - return code 0

Type escape sequence to abort.

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Success rate is 100 percent (100/100), round-trip min/avg/max = 15/16/23 ms
```

FIGURE 44. CONNECTIVITY THROUGH TUNNEL 3

As the logs in the figures show, all paths exhibit connectivity. The latency observed on each path is summarized in Figure 45. The results clearly indicate that the path using VLAN 101 has the lowest latency, followed by the path with VLAN 102, and finally the one with VLAN 100. This allows for the establishment of three types of transport slices, each suited to different latency requirements.

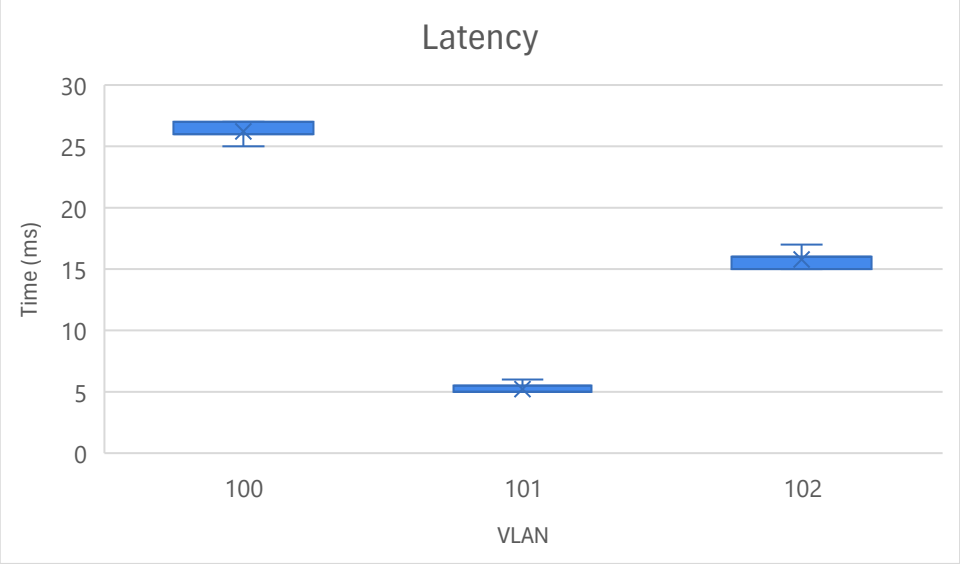


FIGURE 45. LATENCY OVERVIEW

3.2. Hop number

3.2.1. Tunnel path with VLAN 100

```
RP/0/RP0/CPU0:ios#traceroute mpls traffic-eng tunnel-te 1
Tue Apr  8 11:19:50.398 UTC

Tracing MPLS TE Label Switched Path on tunnel-te1, timeout is 2 seconds

Codes: '!' - success, 'Q' - request not sent, '.' - timeout,
  'L' - labeled output interface, 'B' - unlabeled output interface,
  'D' - DS Map mismatch, 'F' - no FEC mapping, 'f' - FEC mismatch,
  'M' - malformed request, 'm' - unsupported tlvs, 'N' - no rx label,
  'P' - no rx intf label prot, 'p' - premature termination of LSP,
  'R' - transit router, 'I' - unknown upstream index,
  'X' - unknown return code, 'x' - return code 0

Type escape sequence to abort.

  0 192.168.24.4 MRU 1500 [Labels: 19010 Exp: 0]
L 1 192.168.24.2 MRU 1500 [Labels: 18001 Exp: 0] 17 ms
L 2 192.168.12.1 MRU 1500 [Labels: implicit-null Exp: 0] 6 ms
! 3 192.168.15.5 33 ms
```

FIGURE 46. HOP NUMBER IN TUNNEL 1

3.2.2. Tunnel path with VLAN 101

```
RP/0/RP0/CPU0:ios#traceroute mpls traffic-eng tunnel-te 2
Tue Apr  8 11:20:44.766 UTC

Tracing MPLS TE Label Switched Path on tunnel-te2, timeout is 2 seconds

Codes: '!' - success, 'Q' - request not sent, '.' - timeout,
  'L' - labeled output interface, 'B' - unlabeled output interface,
  'D' - DS Map mismatch, 'F' - no FEC mapping, 'f' - FEC mismatch,
  'M' - malformed request, 'm' - unsupported tlvs, 'N' - no rx label,
  'P' - no rx intf label prot, 'p' - premature termination of LSP,
  'R' - transit router, 'I' - unknown upstream index,
  'X' - unknown return code, 'x' - return code 0

Type escape sequence to abort.

  0 192.168.24.4 MRU 1500 [Labels: 19009 Exp: 0]
L 1 192.168.24.2 MRU 1500 [Labels: implicit-null Exp: 0] 20 ms
! 2 192.168.25.5 14 ms
```

FIGURE 47. HOP NUMBER IN TUNNEL 2

3.2.3. Tunnel path with VLAN 102

```
RP/0/RP0/CPU0:ios#traceroute mpls traffic-eng tunnel-te 3
Tue Apr  8 11:21:11.435 UTC

Tracing MPLS TE Label Switched Path on tunnel-te3, timeout is 2 seconds

Codes: '!' - success, 'Q' - request not sent, '.' - timeout,
  'L' - labeled output interface, 'B' - unlabeled output interface,
  'D' - DS Map mismatch, 'F' - no FEC mapping, 'f' - FEC mismatch,
  'M' - malformed request, 'm' - unsupported tlvs, 'N' - no rx label,
  'P' - no rx intf label prot, 'p' - premature termination of LSP,
  'R' - transit router, 'I' - unknown upstream index,
  'X' - unknown return code, 'x' - return code 0

Type escape sequence to abort.

  0 192.168.24.4 MRU 1500 [Labels: 19007 Exp: 0]
L 1 192.168.24.2 MRU 1500 [Labels: 19000 Exp: 0] 14 ms
L 2 192.168.23.3 MRU 1500 [Labels: implicit-null Exp: 0] 6 ms
! 3 192.168.35.5 19 ms
```

FIGURE 48. HOP NUMBER IN TUNNEL 3

Figures above show that path with VLAN 101 have two hops, while the others have three hops, proving that path configuration is correct.

4. VLAN tagging

The following tests prove that traffic is tagged at the ingress of the transport network. The untagging process is the reverse operation and it is not included here to avoid redundant duplicated images, but it is also validated. To test the tagging operation, several flows of packets are sent to the Openflow switches:

- GTP traffic with destination IP 10.60.60.106 (N3 user plane traffic between gNB/CU and UPF slice#1): VLAN 102 tag added (Figure 49 and Figure 50)
- GTP traffic with destination IP 10.60.10.6 (N3 user plane traffic between gNB/CU and UPF slice#2): VLAN 101 tag added (Figure 51 and Figure 52)
- SCTP traffic (N2 control plane traffic between gNB/DU – AMF): VLAN 100 tag added (Figure 53 and Figure 54)

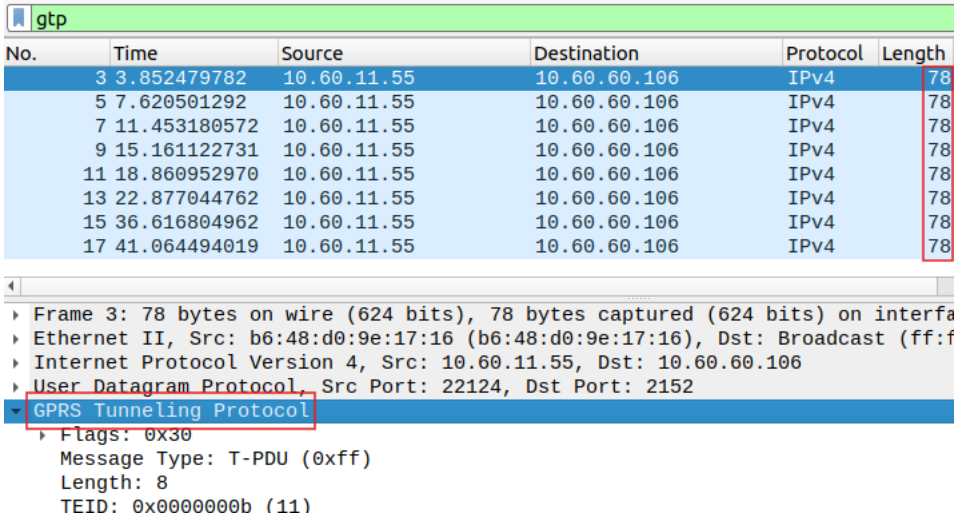


FIGURE 49. GTP TRAFFIC WITH DESTINATION IP 10.60.60.106

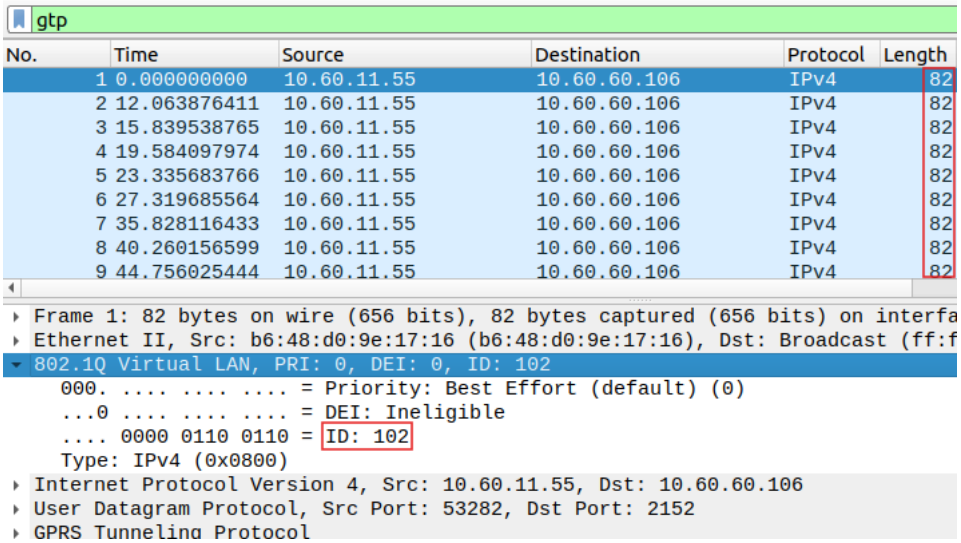


FIGURE 50. GTP TRAFFIC TAGGED WITH VLAN 102

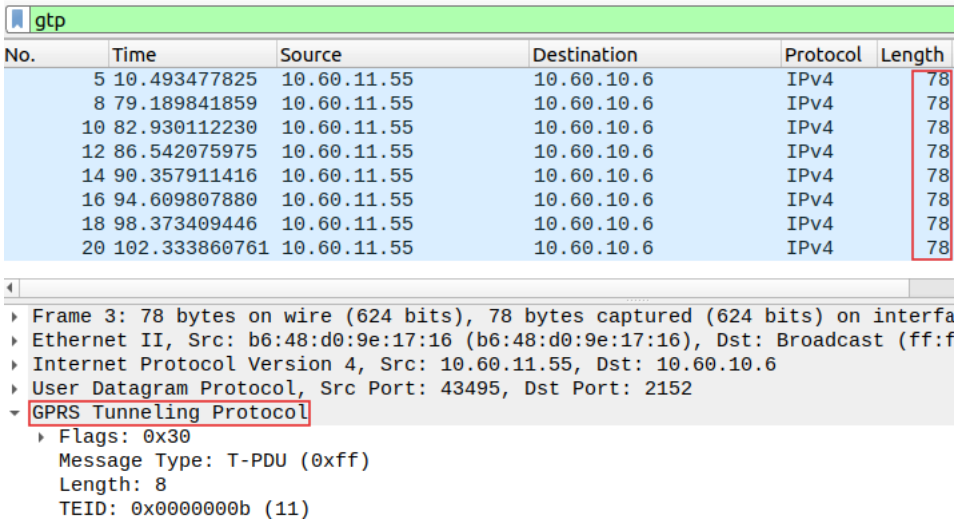


FIGURE 51. GTP TRAFFIC WITH DESTINATION IP 10.60.10.6

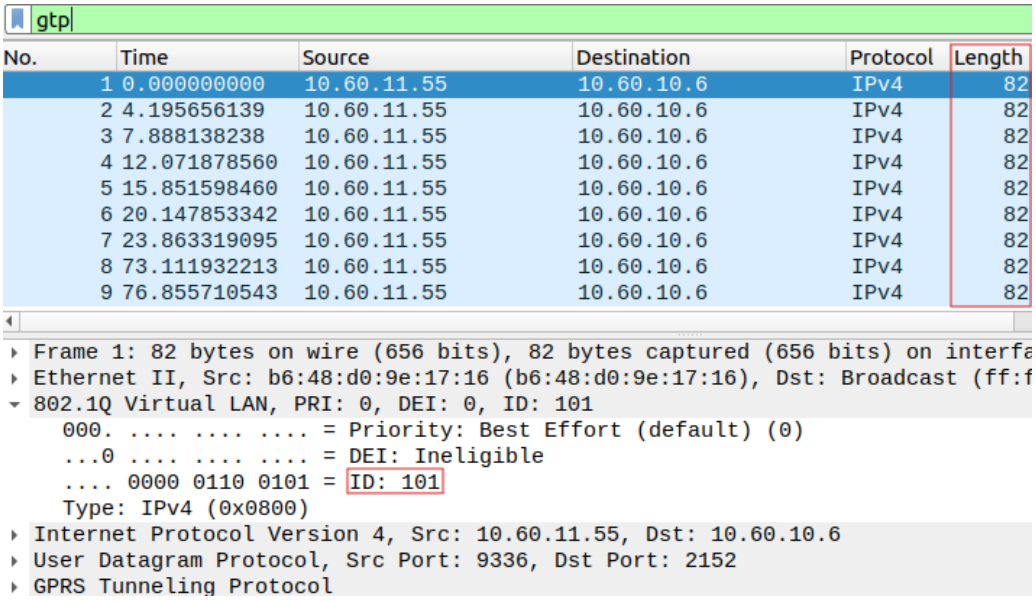


FIGURE 52. GTP TRAFFIC TAGGED WITH VLAN 101

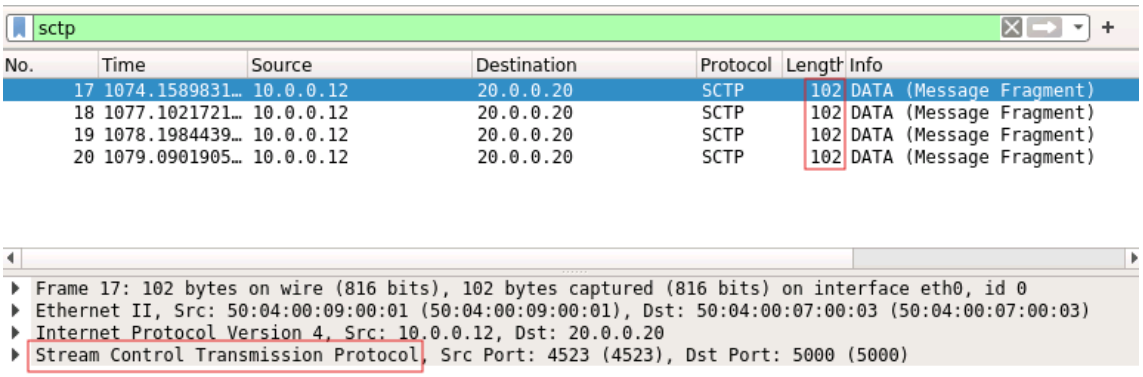


FIGURE 53. SCTP TRAFFIC

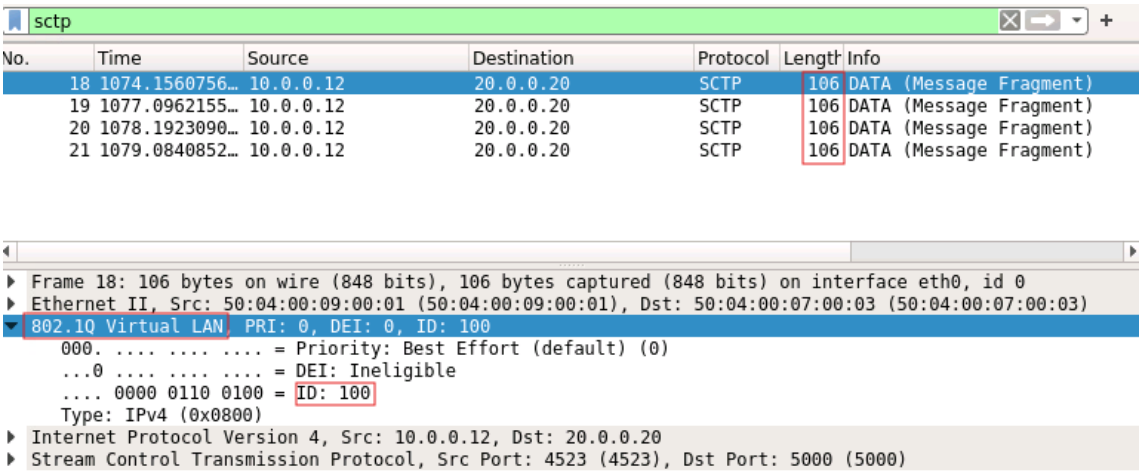


FIGURE 54. SCTP TRAFFIC TAGGED WITH VLAN 100

2.5.4 Conclusions

To sum up, this key concept presented a new element, the network slice controller, to optimize the transport network focusing on an O-RAN-based mobile network. Furthermore, several intensive tests were performed at every point of the system to confirm that the whole implementation was working in conjunction. All tests were successful and confirm the viability of this solution to be implemented into an O-RAN scenario considering multiple end-to-end slices.

2.6 JOINT-K2.2: Entities' placement decisions

Today's networks are characterized by having a massive and constantly growing variety of network functions that, along with the use of proprietary devices, lead to network ossification, hindering network management and service provisioning. Network function virtualization aims to change this situation by decoupling network functions from hardware, performing them in software, at non-specialized computing nodes. Such decoupling mechanism introduces many benefits, including the reduction of both Capital Expenditure (CAPEX) and Operational Expenditure (OPEX), as well as an increased flexibility in service provisioning [34]. The application of this technology is based on algorithms that can instantiate virtualized networks on existing computing infrastructure, aimed to optimize metrics relevant to the particular services. This class of algorithms is commonly known as Virtual Network Embedding (VNE) ones [35]. This key concept focuses on the application of VNE solutions to the disaggregated RAN segment.

The relationship between VNE and disaggregated networks, such as O-RAN, is of utmost relevance since they can help to intelligently manage network resources, by allocating processing, storage, and bandwidth capacities, as well as traffic demands, according to the requirements of applications and services.

The main problem of VNE is its computational complexity, as it usually boils down to NP-hard problems. This means that finding the optimal allocation is a computationally expensive problem and does not have an appropriate general solution in a reasonable time, especially for large instances [36]. In addition, different scenarios present multiple criteria and constraints, such as QoS, processing capacity, bandwidth, latency, and geographic location, adding complexity to the already difficult problem. To address these challenges different VNE solutions have been proposed, based on both heuristic and metaheuristic techniques with a trade-off between optimality and performance [37]. Recently, machine learning algorithms have been also used, which are categorized depending on how they approach the problem and the particular search strategies they employ [38].

2.6.1 System design

VNE is defined as the problem of efficiently assigning virtual networks to physical infrastructures. Specifically, VNE refers to the process of mapping virtual nodes and virtual links of a virtual network to physical nodes and physical links of an underlying network infrastructure (physical substrate) [5]. The virtual networks are typically defined as a set of Virtual Network Request (VNR), each having specific requirements, such as a minimum data rate, a maximum tolerable latency, and a lifespan in

terms of time interval [38]. For each VNR the assignment problem is split into two main subproblems [35], which can be considered jointly or separately:

- Virtual Node Mapping (VNoM): the objective of this subproblem is to find the best match between virtual nodes and available physical resources (processing capacity, bandwidth, memory, network topology) to ensure an optimal performance of the virtual network.
- Virtual Link Mapping (VLiM): In this subproblem, the task is to map the virtual links that connect the assigned virtual nodes in the virtual network through real links in the physical network infrastructure. The challenge lies in finding the optimal paths that meet the bandwidth, latency, and other quality of service requirements.

To tackle this task a variety of solutions are used that can be categorized as:

- **Exact solutions**: exact solutions are those that formulate virtual network embedding problems as Integer Linear Programming (ILP), Mixed Integer Linear Programming (MILP) [39], [40], non-linear Mixed Integer Programming (MIP) [41], or casting VNE to known problems such as Multiple Knapsack Problem (MKP) [42]. This type of solutions leads to an exponential computational cost, which may impose limitations in terms of adaptability to future network changes and scalability in dynamic environments.
- **Metaheuristic solutions**: These are characterized by their ability to explore and exploit the solution search space efficiently, offering a good approximation to the optimal solution in a reasonable time, even if it does not guarantee yielding the best one [37], [43].
- **Heuristic solutions**: These algorithms aim to find acceptable solutions within a reasonable time frame, although they do not guarantee optimality. As a consequence, they often face the problem of getting trapped in local optima, which may be far from the global optimum. To address this limitation, various heuristic alternatives have been proposed, such as the greedy approach and those based on Markov chains [40]. Among these algorithms, ViNEYard stands out, since it improves the coordination between node and link assignment phases [44].
- **AIML solutions**: Machine learning and reinforcement learning algorithms have emerged as promising solutions to address the inherent challenges of efficient resource allocation. In this regard, solutions based on reinforcement allow the collaboration between multiple agents and the hierarchical division of the problem into subproblems [45], [36]. In addition, some recent approaches use Graph Neural Networks (GNN) to automatically extract network features and optimize resource allocation in complex environments [45], [46], [47], [48]. In the same line, Habibi et al. [49] propose GraphVine, a solution that exploits GNN to cluster servers to guide the VNE decision making. Very recently, some works have proposed combining the embedding capabilities of GNNs in general (GCN, GAT, etc.) with deep

reinforcement learning (DRL) architectures. In this sense, in [46] Yan et al. propose a solution exploiting GCN in an Asynchronous Advantage Actor-Critic (A3C) DRL scheme. The same idea using different network definitions and DRL schemes has been adopted [50], [51].

While many solutions for VNE exist, its application to disaggregated RAN has not been tackled so far. In this regard, the combination of GCN and DRL may help on the development of automation systems able to adapt to different topologies and deployment strategies.

In the following, we define the VNE problem and describe the GCN-DRL solution to tackle it. Afterwards, we analyse the performance of the proposed scheme against other approaches.

We represent the physical network as a non-directional graph $G^p = (V^p, E^p)$, where the network nodes correspond to the graph vertices, V^p, and the connections between the network nodes correspond to the graph edges, E^p. It is worth noting that physical nodes can embed both CU/DU virtual entities or RUs. Thus, each physical node $c_i^p \in C^p$ is characterized by the triplet $\{\omega_i^p, m_i^p, r_i^p\}$, where ω_i^p represents the amount of computational resources (i.e. CPU), m_i^p the amount of memory , and r_i^p holds for the amount of RF resources, such as antennas or RF chains. Then, each physical link between two nodes $\{i, j\} \in V^p, l_{i,j}^p \in E^p$, is characterized by its transmission capacity, β_{ij}^p. It is worth remarking that the capacity for a given path or route between two nodes $\{a, b\} \in V^p$, $\mathrm{B}_{\mathrm{ab}}^p$, is defined as $\mathrm{B}_{a,b}^p = \min_{l_{ij} \in R_{ab}} \beta^p{}_{ij}$, where R_{ab} is the set of links that belong to the route. Additional parameters of the physical links, such as propagation, could be also included.

As for the virtual networks, they are defined in a similar manner. Let $G_i^v = (V^v, E^v)$ be the graph of the i-th VNR. Each VNR is made of 3 nodes that represent the CU, DU and RU of the disaggregated base station. Similar to the physical nodes, each virtual node $v_i^v \in V^v$ is characterized by the triplet $\{\omega_i^v, m_i^v, r_i^v\}$, which represents the amount of computational resources, memory and RF resources requested. Note that only the virtual node representing a RU requires RF resources, while this parameter equals zero for the CU/DU virtual nodes. As for the virtual links, they are also defined based on the capacity, β_i^v.

2.6.2 Implementation

The objective is to minimize the cost of embedding each VNR into the physical network while maximizing the performance of the embedding. Similar to other works [51] we define the revenue to cost ratio (R2C) of an VNR G^v as follows:

$$R2C(G^v) = \frac{\phi\ REV(G^v)}{COST\ (G^v)}$$

where ϕ is a binary variable that indicates the feasibility of the solution, $REV()$ holds for the embedding revenue that is computed as a function of VNR resources, and $COST\ ()$ is the cost of the embedding, which is defined as the summation of the used resources. The computation of the function applied to the revenue allows fostering different policies, such as the affinity for some nodes or centralization.

The GCN-DRL follows the design presented in [29] for service function chains and is represented in Figure 55. It follows an Actor Critic scheme where a single node of the current VNR is embedded at each time. As can be observed, the physical network is encoded by a GCN while a Multi-layer Perceptron (MLP) is used for the VNR. The system state is then obtained by a linear combination of both representations. The proposed scheme has been evaluated over FLAG-VNE, which is a variation of VIRNE (https://github.com/GeminiLight/virne), which in turn uses PyTorch libraries.

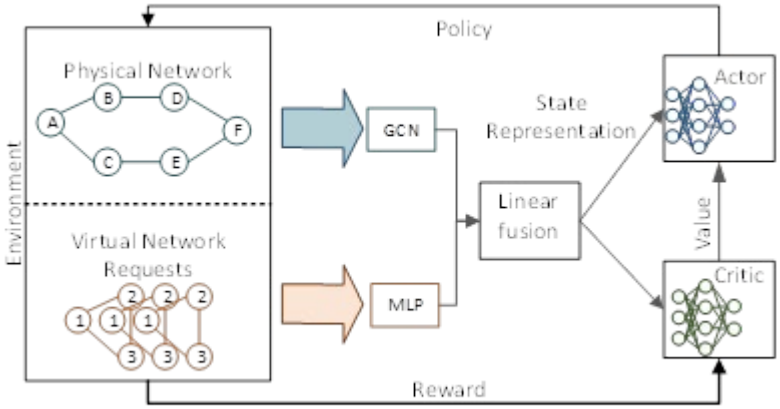


FIGURE 55. GCN-DRL SCHEME

Similar to most works in the literature, the proposed scheme focuses on the VNoM subproblem, while the VLiM one is solved afterwards based on the allocation of nodes. In particular, after allocations the nodes of the VNR the k-shortest path algorithm is used to decide the routes between virtual nodes.

2.6.3 Results

The proposed solution has been analysed over a network with Waxman topology composed of 50 nodes. For simplicity, physical nodes have been configured with enough RAM memory to host all VNRs, while CPU computation capacity is modified to analyse the embedding adaptability of the solution. Among the physical nodes only a few ones are configured with RF capabilities (in particular the last three ones) and with enough capacity to host all the RUs. On the other hand, each VNR is made of three nodes (mimicking the RU/DU/CU chain) where one of them requires 1 unit of RF capacity (i.e. one antenna). Besides, the CPU requirement follows a uniform random distribution between 100 and 1000 generic units. As for the links, enough capacity has been configured in the physical links so that the obtained shortest paths satisfy the requirements of the virtual links. The results are shown by applying the obtained model to a sequence of 100 VNRs. In turn, the model training consists of 10 epochs, each of them embracing 100 VNRs.

Figure 56 shows the evolution of the total revenue for different configurations of the processing capacity of the physical nodes. In particular, the processing capacity is defined as a random uniform variable whose minimum and maximum values are indicated in the legend. As can be observed, the total revenue increases as more computational resources are available within the network.

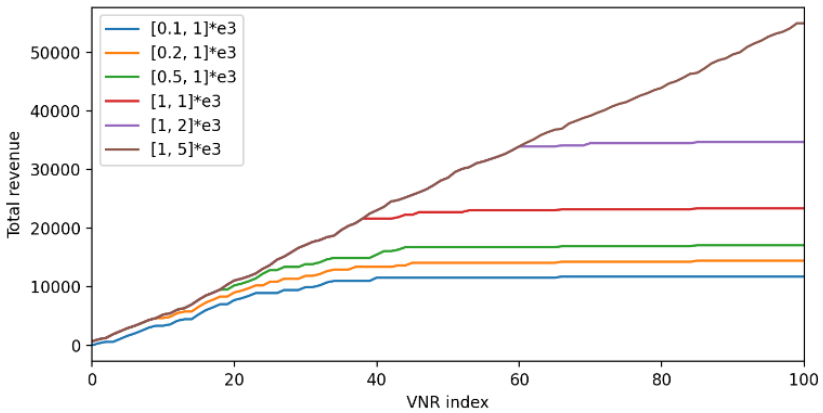


FIGURE 56. EVOLUTION OF THE TOTAL REVENUE AS VNRS ARRIVE

Figure 57 complements the previous one by depicting how the embedding success rate evolves as more physical resources are available. Considering that each DU/CU node has an average computation requirement of 550 units, the total required computational capacity is around 110000 units. It explains that only with the maximum capacity of the physical nodes (around 2500 units in

average) all the VNRs are allocated. In turn, it demonstrates that the proposed model provides a sensible behaviour and is able to exploit all resources when necessary.

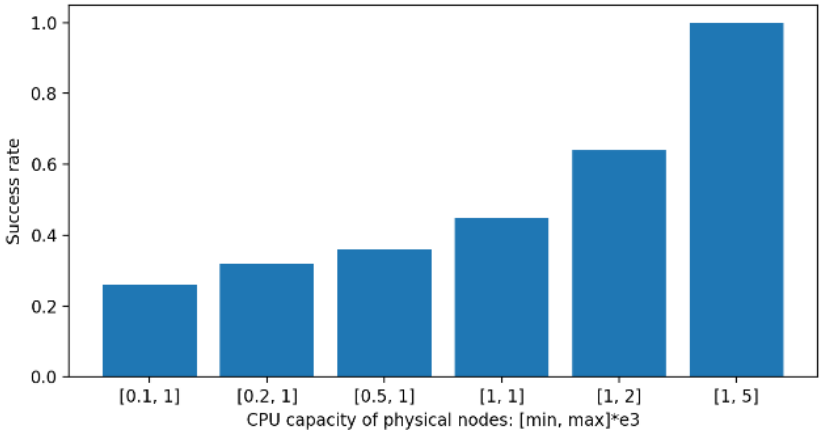


FIGURE 57. EVOLUTION OF THE EMBEDDING SUCCESS RATE

Finally, Figure 58 depicts the distribution of the reward and revenue obtained by the different VNRs upon the variety of physical nodes configurations. As expected, the maximum reward is only reached with the last configuration. Importantly all VNRs obtained the maximum reward, again evincing that as soon as it is possible the proposed scheme optimizes the allocation. As for the revenue, it can be observed that when the available resources are too low, most VNRs obtain almost zero revenue, while a few of them reach values as high as those obtained with large amount of resources. In practice, it would lead to some VNRs (i.e. disaggregated base stations) being efficiently deployed, while others entirely discarded.

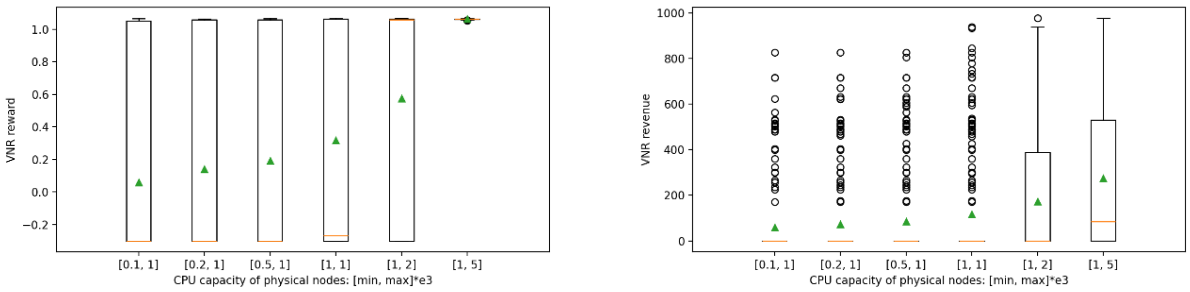


FIGURE 58. DISTRIBUTION OF THE VNR REWARD (LEFT) AND REVENUE (RIGHT)

2.6.4 Conclusions

The disaggregated RAN architectures, such as the one fostered by O-RAN alliance, naturally lead to the virtualization of various RAN elements. In this regard, the proper placement of the virtualized elements can bring relevant benefits in terms of performance and cost. However, the resulting VNE problems present high complexity, making it difficult to find an exact solution due to curse of dimensionality issues. This complexity can be tackled leveraging ML solutions such as deep reinforcement learning. The solution described in the KC exploits DRL and the advanced capabilities of GNN to work with graph-structured data. As has been observed, the described solution is able to effectively exploit all the system capacity and provide reasonable solutions upon system changes. All in all, the described scheme can be customized to foster other policies and according to different criteria, such as centralization or prioritizing physical nodes, or types of VNRs.

2.7 JOINT-K2.3: QoS policies and scheduling

2.7.1 System Design

5G and Beyond (B5G) cellular networks are advancing towards an Open Radio Access Network (O-RAN) architecture, designed to support a flexible, disaggregated, and open framework. This evolution enables operators to integrate components from different vendors seamlessly, allowing for more customizable network deployments [52].

A core feature of O-RAN is the 7.2x functional split, which divides the processing tasks between the Open Distributed Unit (O-DU) and the Open Radio Unit (O-RU). In this split, the O-DU is responsible for higher-level processing functions, including the high-PHY (Physical layer), MAC (Medium Access Control), and RLC (Radio Link Control) layers. Meanwhile, the O-RU handles the lower-level processing, specifically the low-PHY and radio frequency (RF) functions. This division enables enhanced flexibility and scalability within the network, optimizing the distribution of workloads and promoting efficient management of network resources.

However, the O-RAN architecture presents several challenges, particularly due to the O-DU's capability to manage a pool of O-RUs with a certain degree of centralization. This centralization introduces complexities, as it requires the O-DU to handle O-FH (Open Fronthaul) traffic competing with other traffic types, such as backhaul. Additionally, the network's inherent time-sensitive characteristics amplify these challenges, as strict latency requirements must be met to ensure seamless communication between distributed units and maintain consistent network performance.

In addition, properly dimensioning a fronthaul network setup requires a thorough study of its performance under various conditions, as delay variability is a critical factor in preventing Time Alignment Errors (TAE). TAE can significantly degrade network performance, so minimizing variability in latency is essential for maintaining precise synchronization across the fronthaul. Figure 59 illustrates the architecture considered, showing that the transmission and reception windows must be properly aligned. The maximum dispersion, denoted as T12, which represents the highest delay variation experienced by Open Fronthaul traffic, must not exceed certain limits. Furthermore, this traffic may coexist with traffic from legacy technologies, necessitating the application of policies to prioritize O-FH traffic over the others.

To meet these stringent requirements, it is essential to implement HQoS-aware (Hierarchical Quality of Service) policies within the nodes that form the fronthaul segment. These HQoS policies ensure that high-priority traffic, such as control and time-sensitive data (i.e., Open Fronthaul traffic), is managed effectively, enabling a more predictable and stable flow across the fronthaul. This approach helps achieve the reliability and low latency required in 5G and beyond, where fronthaul quality directly impacts the end-to-end performance of the network [53], [54], [55].

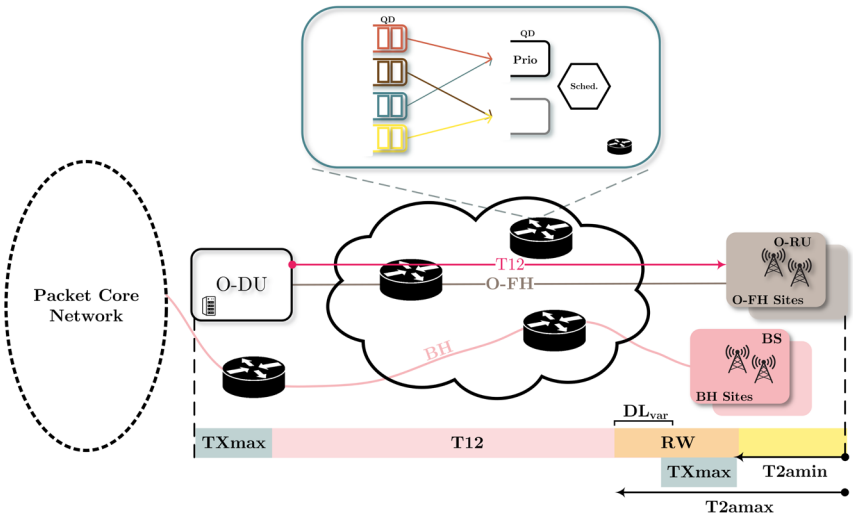


FIGURE 59. FRONTHAUL NETWORK ARCHITECTURE

To analyse end-to-end performance in fronthaul networks aiming for increased centralization—where latency is a critical factor—the following steps were undertaken:

1. **Traffic Characterization**:

Open fronthaul traffic patterns were characterized and validated using the srsRAN software stack, enabling realistic traffic generation aligned with 5G NR specifications.

2. **Performance Evaluation under Different Network Scenarios**:

The evaluation of open fronthaul traffic was conducted over a realistic network topology considering two main deployment scenarios:

- **Dedicated Network for Open Fronthaul Traffic**:
  - **Constant Bit Rate Traffic:** Analysis was performed assuming a constant bit rate, both in the presence and absence of network fragmentation (e.g., enabling/disabling jumbo frames).
  - **Variable Bit Rate Traffic**: Traffic variability, due to dynamic RU load, was modelled and analysed. The E2E delay was studied using an Open Jackson Network approach, under both synthetic and more realistic traffic models that reflect open fronthaul characteristics.
- **Shared Network with Coexisting Technologies**:

  In this scenario, open fronthaul traffic coexists with other types of traffic. Scheduling policies were employed to evaluate performance under mixed conditions.

  The most congested network node was modelled as an M/G/1 queue with priority scheduling to assess the impact of different service levels on the average delay experienced by open fronthaul traffic.

2.7.2 Implementation

To comprehensively analyse the performance of the one-way end-to-end (E2E) delay distribution in O-FH traffic, the implementation has been structured into a series of methodical steps. These include: (i) characterizing the traffic patterns specific to O-FH, (ii) evaluating the one-way E2E delay of O-FH traffic under varying levels of centralization, and (iii) validating HQoS-aware scheduling policies using the ns-3 event-driven simulator. Table 5 summarizes the main contributions in terms of implementation. All source code developed as part of this work is publicly available in the project's repository [2].

Analyse Open-Fronthaul Traffic Patterns:

To analyse Open-Fronthaul (O-FH) traffic patterns under various configurations of 5G NR Frequency Ranges (FR), we have developed a comprehensive approach that integrates both real-world implementation and traffic modelling. Initially, leveraging the 5G NR FR and specified compression methods, we developed a Python script designed to accurately generate key O-FH traffic characteristics. This includes determining the corresponding rates for both the User Plane and

Control Plane, while considering different conditions such as bandwidth (BW), subcarrier spacing (SCS), compression techniques, the number of cells per site, and the number of layers. This approach ensures an in-depth analysis of how these factors influence O-FH traffic behaviour.

For validation, we deployed srsRAN, an open-source 5G software suite, on an Ubuntu 22.04.1 LTS server to run a real Open Distributed Unit (O-DU) instance. This configuration enabled us to transmit O-FH traffic to a loopback interface, with Wireshark capturing the traffic in real-time. To extract key performance metrics such as throughput and packet size, we developed custom Juniper scripts to properly dissect the *.pcap* files captured by Wireshark. This ensured an accurate analysis of the traffic, as illustrated in Figure 60, which outlines the workflow followed during the process.

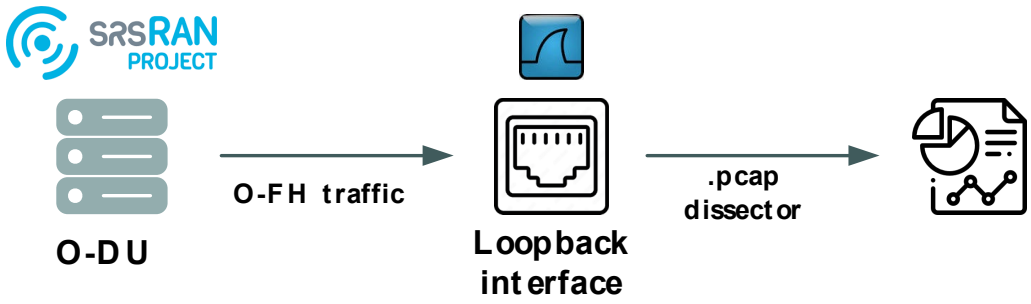


FIGURE 60. METHODOLOGY FOR CHARACTERISING O-FH TRAFFIC FROM REAL DEPLOYMENT

Evaluate One-Way End-to-End (E2E) Delay of Open Fronthaul Traffic with Different Centralization Levels
Using ns-3, implemented in C++, several simulations were conducted to analyse the one-way end-to-end (E2E) delay, assuming the network setup as either a dedicated network or a slice. To achieve this, we first examined the ns-3 queue architecture and introduced an input queue to simulate the processing delay of routers, as determined by their switching matrix. Specifically, within the point-to-point module, the *net-device* was modified to include an input queue associated with the receive interface, as well as a processing time for each packet. This processing time reflects the delay experienced as packets traverse from the input queue to the output queue. Notably, while the original implementation already accounted for transmission capacity, this modification now allows us to adjust both the switching capacity and the transmission capacity, further enhancing the simulation's realism by capturing both processing and transmission dynamics as depicted in Figure 61.

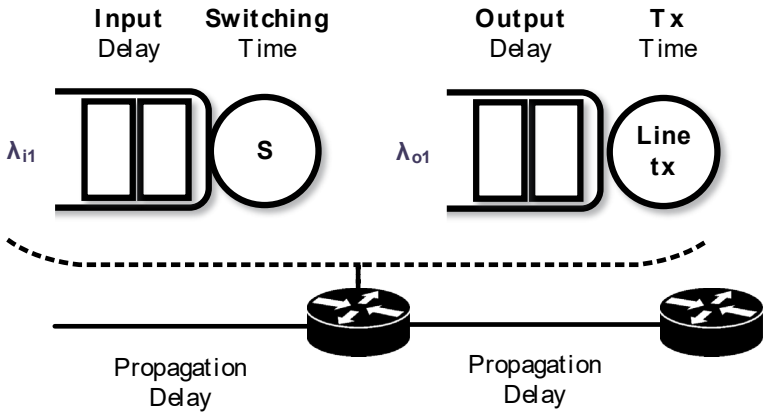


FIGURE 61. SOURCES OF DELAY IN A NETWORK ARCHITECTURE

To analyse the one-way E2E delay performance, we deployed several applications to simulate traffic under various conditions in the *applications* module:

1. **OFH-Application**: This application generates two traffic flows corresponding to the User Plane (U-Plane) and Control Plane (C-Plane). Several parameters can be configured, including enabling or disabling both planes, adjusting the interval between packets, and choosing between constant or exponential distribution for the packet interval. In this specific case, the application assumes constant packet sizes for both planes.
2. **Poisson-Application**: In addition to the OFH-Application, we developed a Poisson-based application to model traffic. This application allows for the definition of exponential inter-arrival times, and the packet size distribution can be configured as either constant or exponential, offering flexibility in traffic generation.

To carry out the simulation, a common scenario has been created with a spine-leaf topology. This topology includes different hierarchical levels that represent various degrees of centralization, as depicted in the following Table 4. The configuration parameters for this scenario are specified via a JSON file, allowing for easy customization and flexibility. The key configuration settings include:

TABLE 4. SCENARIO SETUP AND CONFIGURATION PARAMETERS DEVELOPED IN NS-3

Scenario Setup	Configuration

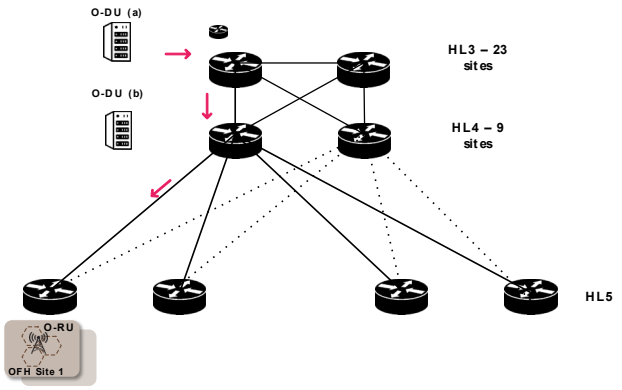


FIGURE 62. NETWORK TOPOLOGY WITH DIFFERENT CENTRALIZATION DEGREES

Site

- # Cells
- Per cell
  - C&UPlane o Uplane
  - Plane rate & Packet Size

Link

- Delay

Router

- Port
  - Transmission rate
    - MTU
    - Buffer
  - Switching capacity
  - HQoS policies
    - Strict priority (SP)
    - WRR
    - SP + WRR

Validate HQoS-Aware Scheduling Policies with ns-3 Event-Driven Simulator

With the aim of testing different HQoS, aware scheduling policies, prioritizing Open Fronthaul (O-FH) traffic as the highest priority, and properly validating their performance in a realistic scenario, we have developed several utilities within the *traffic-control-layer* module.
First, a marking capability has been developed to classify packets according to the port, by leveraging the Differentiated Services Code Point (DSCP) field in the IP packet header, *marker-queue-disc.cc*, as depicted in Figure 63. To enable this, the traffic control policies of the ns-3 framework have been adapted to properly classify packets based on the defined traffic profile.

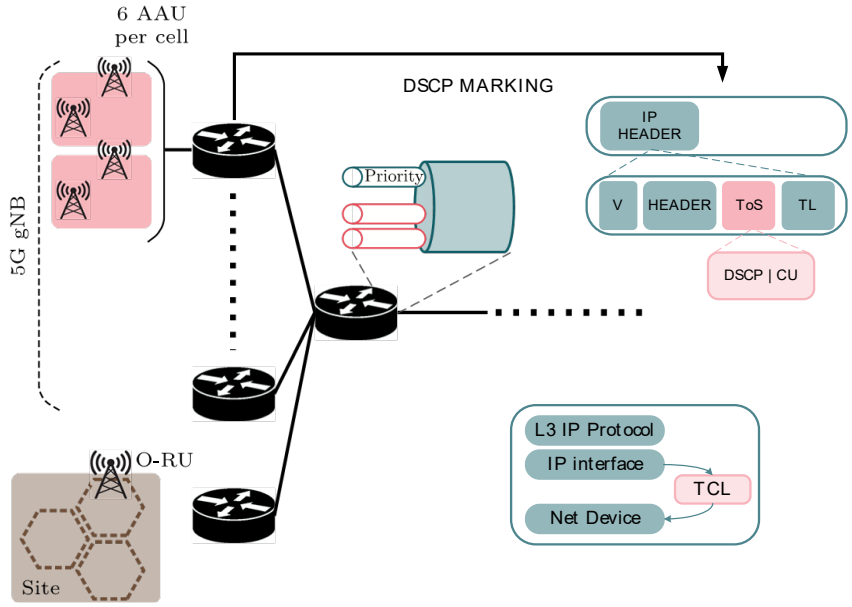


FIGURE 63. DIFF SERV ARCHITECTURE IMPLEMENTATION FOR FRONTHAUL NETWORKS

In addition, the implementation enables the combination of different scheduling policies by leveraging the marking capabilities defined in the scenario configuration. A hierarchical scheduler has been implemented, in which Open Fronthaul (O-FH) traffic is assigned to a Strict Priority (SP) queue, while backhaul traffic is placed in a secondary queue using Weighted Round Robin (WRR) scheduling, based on the DSCP field, as depicted in Figure 64.

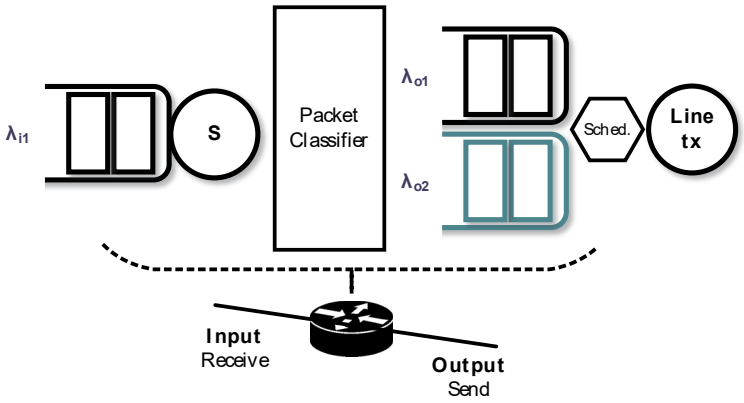


FIGURE 64. QUEUE MODELING FOR A DIFFSERV ARCHITECTURE IN NS-3

Modelling Performance

To accurately emulate network performance, the ns-3 network device, *net-device*, was configured to adjust the service time according to the queuing models, omitting transport and IP layer headers. This refinement ensures a more precise alignment between simulated packet sizes and those considered in analytical models.

In parallel, several queuing models were developed in Python to avoid the longer simulation times caused by the high load of Open Fronthaul traffic. These models serve as a complementary tool for analysing the one-way average delay in the fronthaul network and extracting benchmark values for different priority levels as traffic traverses a node in specific segments of the network. Additionally, open Jackson networks were used to characterize the one-way delay between the O-DU and the O-RU. These tools support both the validation of analytical models and the comparison with simulation results under realistic traffic conditions.

TABLE 5. FUNCTIONALITIES DEVELOPED

Code		Purpose
ns-3	Scenarios	• Realistic scenario based on IP Fusion Network (NW) architecture.
	Applications	• **Open Fronthaul (OFH) Application** to simulate U-plane and C-plane traffic. • **Distribution-app** to model a synthetic traffic
	net-device	• Enhanced with **switching capabilities** to reflect realistic forwarding behaviour. • Enable modelling (avoid counting headers)
	Traffic-Control Layer	• **DSCP marking support per port** for fine-grained QoS control. • **DSCP-based classification** to map incoming packets to traffic classes (queues). • Hierarchical QoS scheduling combining: ○ Strict Priority (SP) for high-priority, low-latency traffic.

		○ Weighted Round Robin (WRR) for fair bandwidth allocation among lower-priority flows.
Python-based Post-processing & Modelling	Traffic Capture and Analysis	• **Custom dissector** to extract throughput and traffic characteristics of user/control plane data. • Based on parsing of **real .pcap traces** captured from **srsRAN** software stack.
	Queueing Theory Models	**M/G/1 Model:** Applied to characterize: • Average delay benchmarks for OFH traffic. • Impact of HQoS on delay performance across different priority levels. **Open Jackson Network (OJN):** • Utilized to estimate end-to-end one-way delay between the O-DU (Distributed Unit) and O-RU (Remote Unit). • Accounts for cumulative queuing and service delays across network elements.as different priority levels when HQoS has been applied.

2.7.3 Results

Analyse Open-Fronthaul Traffic Patterns:

To accurately evaluate the end-to-end (E2E) delay experienced by open-fronthaul traffic, a comprehensive analytical modelling of the downlink (DL) traffic generation process was conducted. The modelling traces the data path starting at the Distributed Unit (DU), considering its traversal through the MAC and high-PHY layers, which directly influence the structure and periodicity of the C/U-Plane traffic transmitted over the fronthaul interface.

Key parameters influencing this modelling include:

- **Protocol Stack Considerations and Packetization Effects**:

The size of the fronthaul packets is determined by how MAC-PHY payloads are encapsulated. Factors such as maximum transmission unit (MTU), the use of jumbo frames, and fragmentation policies impact the number of packets generated and their inter-arrival times.

- **Numerology Configuration**:
  Subcarrier Spacing (SCS) and **Bandwidth (BW)** determine the time-frequency grid used in the physical layer. The number of **Physical Resource Blocks (PRBs)** per Transmission Time Interval (TTI) increases with bandwidth and affects how much data is allocated per user.
- **MIMO Layering and Sectorization**:
  The number of spatial layers configured (e.g., 1 to 8 layers in 5G NR) per transmission sector significantly impacts the overall data volume. Multiple layers result in parallel data streams, increasing the aggregate throughput and, consequently, the fronthaul load.
- **IQ Sample Compression**:
  Compression algorithms (e.g., block floating point, μ-law) are applied to reduce the volume of IQ samples transmitted over the fronthaul. The compression ratio influences the final packet size and must be accurately modelled to reflect realistic traffic conditions. The compression ratio significantly reduces the bandwidth demand on the fronthaul, which affects queuing and delay behaviour.

So given the maximum transmission bandwidth configuration NRB for each UE channel bandwidth and subcarrier spacing defined in Table 5.3.2-1 in TS 38.101 is possible to obtain the maximum data rate [56]. First, we have to establish the packet size of the User Plane, we adopt eCPRI over Ethernet (without IP/UDP), using fronthaul block compression techniques, like Block Floating Point (BFP). The Physical Resource Block PRB_{size} is defined as the payload data width (9 bits by applying BFP9 compression) multiplied by the number of subcarriers, accounting for both the I and Q components, with an additional 8 bits for overhead:

$$PRB_{\text{size}} = \frac{\text{data}_{\text{width}} \cdot N_{\text{subcarriers}} \cdot 2 + 8}{8}$$

Then, considering the number of available RBs within the given BW and SCS defined in [56], we can calculate the total number of bits allocated to user data (iq_{data}), taking into account the Physical Resource Block (PRB) size, as follows:

$$iq_{data} = N_{RB} \cdot PRB_{\text{size}}$$

Besides, the packet size ($Packet_{size}$) encompasses both user data and packet header overhead:

$$Packet_{size} = iq_{data} + Packet_{header}$$

Finally, the data rate in each configuration is estimated by considering the packet size, the number of symbols per slot (i.e., 12), the modulation order (2μ), and converting it to bits per second:

$$\text{Rate} = \text{Packet}_{\text{size}} \cdot 12 \cdot 2^{\mu} \cdot 8$$

Next Figure 65 shows the analytical results and experimentation with srsRAN project software – downlink mode – using SCS of 30kHz across various bandwidths and TDD configuration in a 1 SISO carrier mode, when packets are using all the available PRBs. As depicted in the Figure 65, the error in the analytical approach is around 10%.

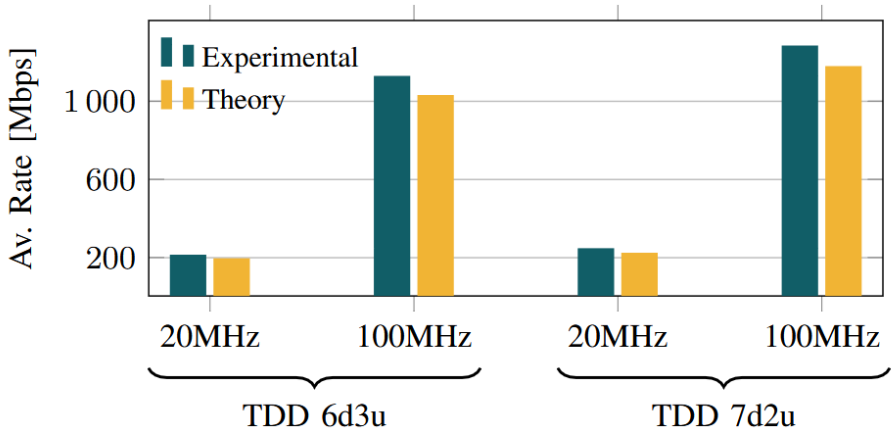


FIGURE 65. COMPARATIVE OF ANALITICAL AND EXPERIMENTAL RATE OF O-FH TRAFFIC IN A SISO MODE

Evaluate One-Way End-to-End (E2E) Delay of Open Fronthaul Traffic with Different Centralization Levels

Once the Open Fronthaul traffic was characterized, its performance was evaluated over a realistic network setup, taking into account all relevant sources of delay, including switching, queuing, transmission, and propagation. In Figure 66, the topology based on the IP Fusion network at Telefónica is shown. Additionally, Table 4 summarizes the corresponding features addressed in the scenario.

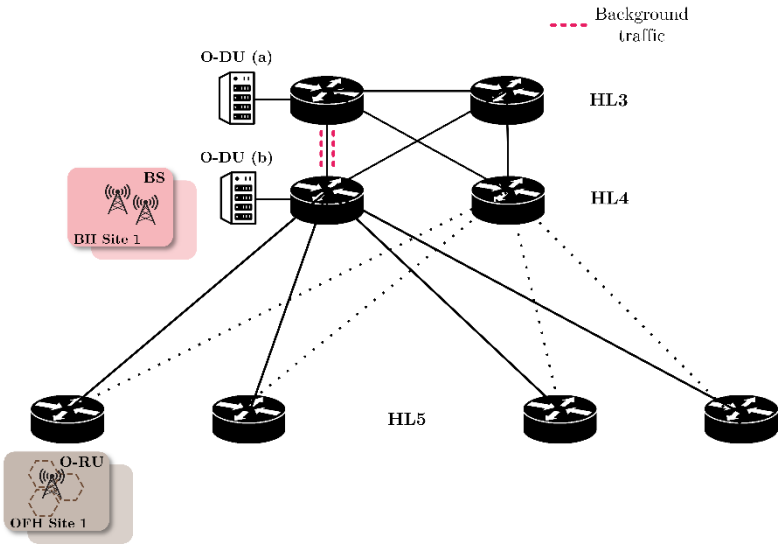


FIGURE 66. TOPOLOGY OF IP NW FUSIÓN TELEFÓNICA WITH DIFFERENT DEGREES OF CENTRALIZATION AND BACKGROUND TRAFFIC FROM LEGACY TECHNOLOGIES

First, the O-DU/O-RU end-to-end delay has been studied through simulations performed with the ns-3 event-driven simulator. Both scenarios, O-DU at HL3 and HL4, have been considered using the different centralization levels. The variability induced by the network was evaluated by analysing the one-way delay, assuming only the downlink direction, with each site aggregating traffic owing to a tri-sectional configuration, as previously commented. In this sense, the impact of network impairments, along with the underlying propagation delay, is analysed over a realistic scenario involving fragmentation, with an MTU set at 1500B.

The simulation setup is configured according to the network scenario outline in Figure 62. Furthermore, the delay induced by routers is accounted using two different approaches. In the first one, each node is assumed to have a constant processing delay, average value mentioned above (10 µs), as depicted in Fig. 2 by HL3-Cte and HL4-Cte. Each boxplot in Figure 67 represents the O-FH delay variability, which considers both the propagation time and the delay induced by the routers, in which the queuing contribution is negligible (in the order of ns).

In the second approach, the delay at each node is modelled as a shifted exponential distribution, with minimum, average, and maximum delays of 5 µs, 10 µs, and 35 µs, respectively, to analyse the variability of the routers delay. These values were obtained from the HL4 testbed depicted above. As shown in the Figure below, indicated by HL3-Var and HL4-Var, the accumulation of variability across multiple nodes results in a dispersion of approximately ±20 µs (lower and upper whiskers) around the average value.

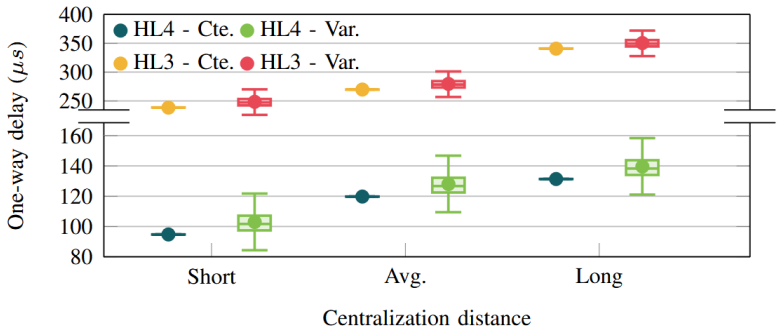


FIGURE 67. ONE-WAY DELAY FROM O-DU TO O-RU UNDER DIFFERENT CENTRALIZATION DISTANCES

After characterizing the overall performance, we focused on the variability stemming from the load dynamics of the sites. This variability leads to fluctuations in packet sizes and data rates, which in turn affect the behaviour of fronthaul traffic. In this context, we specifically concentrate on the delay introduced by transmission and, when present, queuing. The following Figure 68 illustrates the one-way delay experienced under different levels of PRB utilization (indicated by markers as depicted in legend), that is, under varying site loads, along with the validation of the open Jackson network model as a benchmarking tool for estimating average delay. Figure 68 represents the average delay and its dispersion across different degrees of saturation, while also validating the theoretical model.

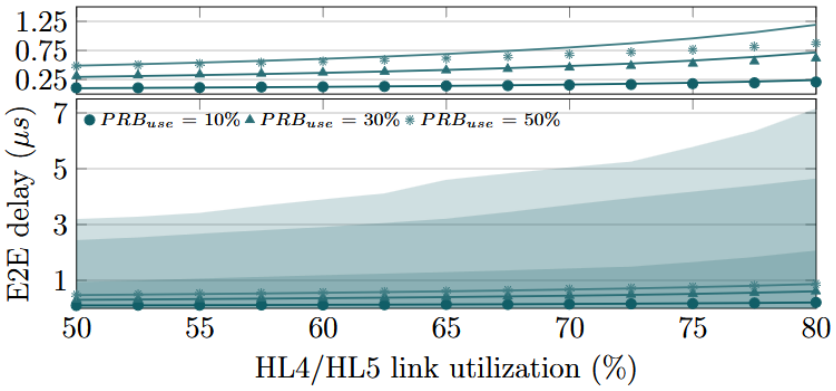


FIGURE 68. ONE-WAY DELAY FOR NON-DETERMINISTIC TRAFFIC UNDER DIFFERENT LOADS OF TRAFFIC GIVEN BY O-FH SITE. COMPARATIVE OF THE QUEUE MODEL BASED ON JACKSON NETWORKS WITH SIMULATION BY NS-3

Validate HQoS-Aware Scheduling Policies with ns-3 Event-Driven Simulator

Once the Open Fronthaul performance was evaluated under a dedicated network, its behaviour was further analysed in scenarios where it shares network segments with other technologies. To model this, the M/G/1 queueing system with multiple priority levels was applied, providing benchmark values for the average delay experienced by each priority class.

To validate the model, traffic profiles corresponding to URLLC and eMBB services were considered. The scheduling policy implemented combined Strict Priority (SP) and Weighted Round Robin (WRR), with four priority levels—each appropriately configured with its respective weights. Note that Open Fronthaul traffic was assigned the highest priority.

Figure 69 presents the benchmark delay values introduced by the node where resource contention occurs, across various saturation levels. Subsequent figures illustrate the system's performance under different network load conditions, specifically 94.7% saturation (23 active sites) and 90.6% saturation (22 active sites), Figure 70 and Figure 71, respectively, for different traffic profiles.

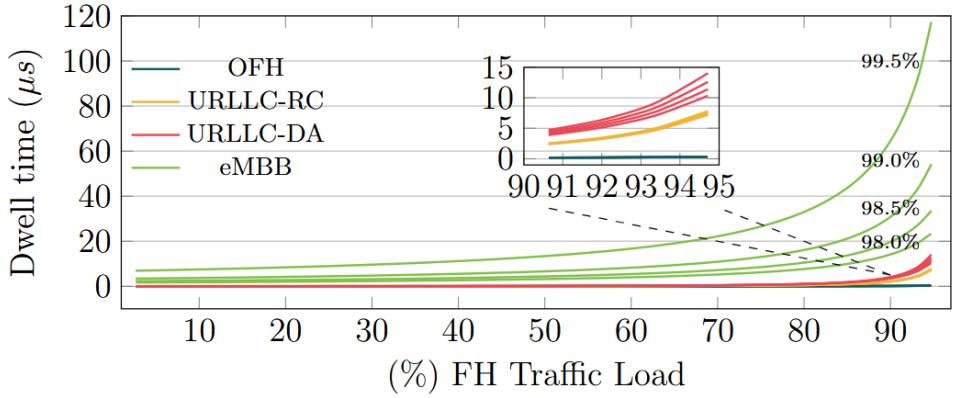


FIGURE 69. BENCHMARK VALUES FOR AVERAGE DELAY OF DIFFERENT PRIORITY LEVELS BY USING A M/G/1 MODEL

The results clearly show that reducing the load of the highest-priority queue (i.e., the open-fronthaul traffic) leads to a significant decrease in the overall average delay across all queues. As the demand for high-priority traffic is reduced, network congestion is alleviated, improving performance even for lower-priority traffic. Moreover, the model demonstrates a good approximation of the observed system behaviour, validating its suitability as a benchmarking tool.

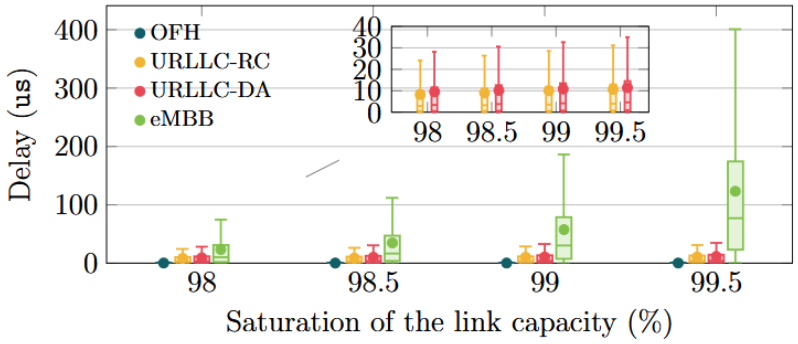


FIGURE 70. SIMULATION PERFORMANCE OF DELAY FOR DIFFERENT SERVICES WITH O-FH TRAFFIC THE HIGHEST PRIORITY. TRAFFIC LOAD OF O-FH TRAFFIC GIVEN BY 23 SITES ACTIVE

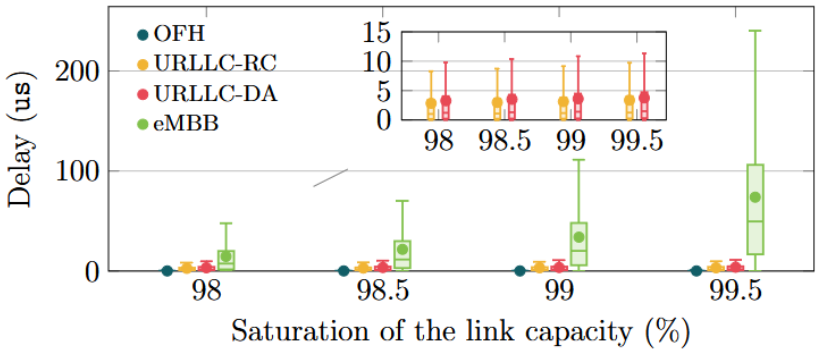


FIGURE 71. SIMULATION PERFORMANCE OF DELAY FOR DIFFERENT SERVICES WITH O-FH TRAFFIC THE HIGHEST PRIORITY. TRAFFIC LOAD OF O-FH TRAFFIC GIVEN BY 22 SITES ACTIVE

2.7.4 Conclusions

This work has analysed the open fronthaul delay (from O-DU to O-RU) under various conditions with the help of the ns-3 simulator and the required enhancements. First, we accurately model the open fronthaul traffic, which is strictly dependent on the radio configuration, to include it in the analysis. This model is also validated using the *srsRAN* software.

To isolate and understand the impact of the network itself, we evaluate the delay performance in a controlled environment, free from the influence of other types of traffic. This allows us to establish a baseline for comparison. In this initial step, we assume that the generated traffic follows a constant bit rate pattern, thereby focusing exclusively on the network's behaviour, particularly the packet processing performance. In addition, we provide a mathematical framework to analyse the end-to-end (E2E) delay on the fronthaul based on Jackson networks.

Once the baseline behaviour has been established, we consider a scenario in which this network segment is shared with other technologies through the use of DiffServ-based policies. The open fronthaul traffic is assigned the highest priority when configuring these policies across the network nodes. The results show that the open fronthaul traffic does not experience any degradation when coexisting with other traffic, even under saturation conditions. Additionally, a mathematical framework is provided to derive benchmark values for different priority levels under a given scheduling policy configuration.

Finally, we observe that the impact of delay dispersion, measured using the ns-3 event-driven simulator, introduced by the open fronthaul traffic is minimal, which is a key consideration when planning and designing the network. This is attributed to both the inherent characteristics of the traffic patterns and the design and configuration of the fronthaul network.

2.8 JOINT-K3.1: Monitoring platform

In deliverable E3 of 6G BLUR-JOINT, two different frameworks for RAN monitoring platforms were presented. The first framework consists of a RAN monitoring platform based on an xApp that performs KPM monitoring over the E2 O-RAN interface. The second framework is an adaptive, intent-based monitoring system based on the deployment of a swarm of LLM agents. We discuss the outcomes obtained with both frameworks next.

2.8.1 KPM Monitoring xApp

2.8.1.1 System Design

The KPM monitoring platform over E2 interface implemented follows the structure shown in Figure 72. The figure also shows integration of the monitoring xApp, deployed in the open-source FlexRIC [57] platform, with Keysight's tool RICtest [58]. RICtest emulates O-RAN network nodes and traffic profiles on different RAN Intelligent Controller's (RIC's) interfaces, allowing testing of the Near Real-Time RIC (and of xApps it hosts) and of the Non-Real-Time RIC (and of rApps it hosts). It is capable of emulating O-RAN nodes with E2 and O1 interfaces serving large quantities of UEs deployed in a multi-cell environment. RICtest is connected to FlexRIC over the E2 interface and used to generate the data and KPMs that the KPM monitoring xApp observes.

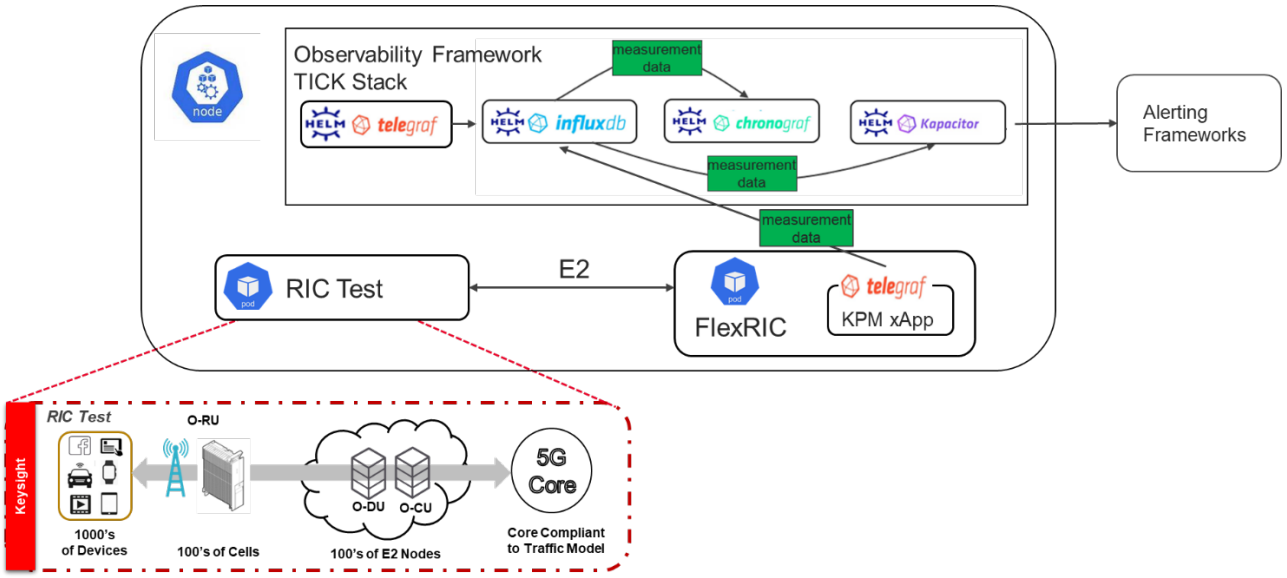


FIGURE 72. ARCHITECTURE OF A TESTING FRAMEWORK FOR AN E2 MONITORING XAPP

2.8.1.2 Implementation

For data handling, the open-source framework TICK Stack [59] by InfluxData is leveraged, due to its ability to handle time-series data. The framework is made of four primary components: Telegraf, which acts as an agent with the role of collecting data from different sources and, after optional preprocessing, forwarding them to the database; InfluxDB is the time-series database used to efficiently store and query data; Chronograph acts as the visualization and user interface component; finally, Kapacitor is used as a real-time data processing engine.

The above proposed monitoring framework can be used to collect KPM data over the E2 interface. In turn, such data can be used to train AI models for different RAN optimization and analysis tasks, as well as to run inference on such models after being trained.

The integration of the proposed KPM monitoring solution with the PoCs of 6G BLUR and, in particular, in the testbed of UC2-PoC1, was not attempted as the testbed implemented a single cell and was primarily focused on the study of the O-FH interface between O-RU and O-DU and the consequences of having a longer distance between these two nodes than the one that is considered as the standard distance. Since no xApp was used in this context, the testbed did not include a RIC component. Therefore, this PoC did not represent a suitable use-case for the implementation and testing of the designed monitoring platform. Nonetheless, in cooperation with other research projects, Keysight has continued developing the proposed KPM monitoring platform to show its

potential to perform anomaly detection over an O-RAN network. We shortly refer to the results obtained next.

2.8.1.3 Results

Keysight developed a KPM monitoring XApp in order to collect data to train and run an AI model performing anomaly detection over an O-RAN network. This xApp was developed within the FlexRIC open-source community framework [9], and has been published as open-source software in [60]. Table 6 presents the performance metrics monitored using the developed xApp.

TABLE 6. KPMS MONITORED BY THE PROPOSED XAPP (NUMBER IN BRACKETS INDICATE SECTIONS IN 3GPP TS 28.5529 WHERE THE CORRESPONDING KPI DEFINITION IS PROVIDED)

KPI
MM.HoExeInterFail (5.1.1.6.1.9)
MM.HoExeInterReq (5.1.1.6.1.7)
MM.HoExeInterSucc (5.1.1.6.1.8)
MM.HoExeIntraReq (5.1.1.6.2.1)
MM.HoExeIntraSucc (5.1.1.6.2.2)
MM.HoPrepInterFail (5.1.1.6.1.3)
MM.HoPrepInterReq (5.1.1.6.1.1)
MM.HoPrepInterSucc (5.1.1.6.1.2)
MM.HoResAlloInterFail (5.1.1.6.1.6)
MM.HoResAlloInterReq (5.1.1.6.1.4)
MM.HoResAlloInterSucc (5.1.1.6.1.5)
RRC.ConnMax (5.1.1.4.2)
RRC.ReEstabAtt (5.1.1.17.1)
RRC.ReEstabSuccWithUeContext (5.1.1.17.2)
RRC.ReEstabSuccWithoutUeContext (5.1.1.17.3)
RRU.PrbUsedDl (5.1.1.2.5)
RRU.PrbUsedUl (5.1.1.2.7)
SM.PDUSessionSetupReq (5.1.1.5.1)
SM.PDUSessionSetupSucc (5.1.1.5.2)

With the collected data, an autoencoder was trained to identify Physical Cell Identity (PCI) conflicts in the network. A PCI conflict occurs when one or more neighbouring cells have mistakenly been configured with the same PCI, and it is a problem that operators encounter frequently when managing their networks. When two neighbouring cells have identical PCI, a UE transitioning between both cells will not be able to detect the second cell and collect the necessary measurements to perform a smooth handover. PCI conflicts can lead to intercell handover execution failures, resulting in increased values for the "*RRC.ReEstabAtt*" and "*RRC.ReEstabSuccWithUEContext*" performance counters. Additionally, the "*MM.HoExeInterFail.Unspecified*" counter may also show elevated activity. Hence, all these three KPMs can be used to diagnose a possible PCI conflict.

The above use case has been documented in a Keysight's Application Note publicly available in [61]. There, it is highlighted how the autoencoder trained with the data collected by the proposed monitoring method is successful in detecting most of the targeted anomalies (PCI conflict). However, the results also unveil that the anomaly detection model can fail in identifying rare anomalies which only manifest themselves seldom in the training dataset. This underlines that, while collection of large amount of data is a necessary step for AI model training, curation of the datasets to eliminate biases in them is also an important task to boost AI-based operations such as anomaly detection, network optimization, etc.

2.8.1.4 Conclusions

The preliminary results discussed above show that the proposed monitoring xApp is useful to monitor E2 interface and create datasets that can be used for AI model training. While the xApp has so far only been tested under Keysight's emulator of E2 nodes, such an emulator is fully O-RAN compliant and, as such, it is expected that it should be used will E2 nodes coming from any vendor that follows the E2 interface specification. Next steps in this work are to implement the proposed platform in a network including actual E2 nodes (possibly in addition to emulated ones) and to perform live anomaly detection in a network setup closer to reality.

2.8.2 Agentic Framework in the Monitoring Platform

2.8.2.1 System Design

The system architecture of the proposed agentic framework for RAN monitoring purposes is divided into four main blocks, as depicted in Figure 73:

1) O-RAN Setup:
    - Includes O-RAN components such as the Software Radio Systems (SRS) gNB, and the O-RAN SC RIC (RAN Intelligent Controller), which work together with an Open5Gs core network instance.
    - Deployment leverages Docker Compose for flexibility.
    - The xApp communicates with the RIC and logs are centralized for monitoring.
2) LLM Agents:
    - A swarm of agents is deployed to manage O-RAN components, each operating as a Docker container. Agents include:

- Doorkeeper Agent: Routes user queries to relevant agents.
- Docker Agent: Monitors container performance.
- Log Analyzer Agents: Parse and analyse logs from Open5GS, RIC, and other components.
- Control Agents: Manage and modify configurations (e.g., SRS gNB slicing parameters).

3) Log Center:
   - Centralized log collection and analysis powered by Retrieval-Augmented Generation (RAG) capabilities.
   - Logs from various components are aggregated, indexed, and searched efficiently.
4) User Interface (UI):
   - A Gradio-based GUI enables users to input queries, review analysis results, and monitor system performance.
   - Fixed and custom query options streamline interaction.

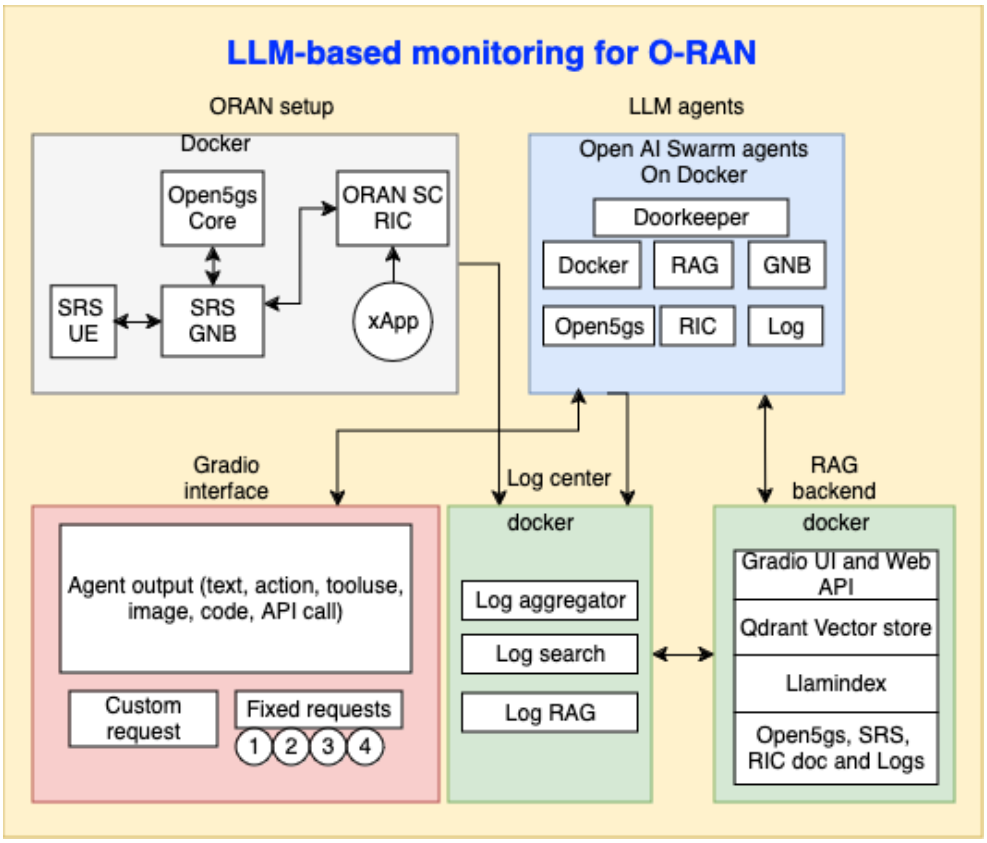


FIGURE 73. SYSTEM ARCHITECTURE FOR LLM-BASED O-RAN MONITORING THE ARCHITECTURE INTEGRATES O-RAN COMPONENTS, LLM-DRIVEN AGENTS, A CENTRALIZED LOG COLLECTION SYSTEM, AND A USER-FRIENDLY GRADIO INTERFACE. THE MODULAR DESIGN FEATURES SPECIALIZED AGENTS FOR MONITORING, LOG ANALYSIS, AND CONTROL, SUPPORTED BY A RETRIEVAL-AUGMENTED GENERATION (RAG) BACKEND FOR CONTEXT-AWARE RESPONSES AND EFFICIENT DATA RETRIEVAL.

2.8.2.2 Implementation

Agents are autonomous, modular entities that execute specific tasks within a system. In the context of O-RAN monitoring, agents represent a lightweight, scalable, and dynamic solution for handling complex workflows such as log analysis, monitoring, and control of network configurations. These agents operate independently yet collaboratively, relying on defined workflows and orchestration mechanisms to ensure task completion.

Agents in this framework leverage Large Language Models (LLMs) to enhance decision-making, adapt to dynamic requirements, and provide intelligent, context-aware responses to user queries. Each agent encapsulates specific functionalities and can collaborate with others through a defined handoff process.

LLM-based Agents

- An LLM agentic uses an LLM at its core to handle a given task, similar to how a human would. The agentic framework has several important components and functionalities, as outlined below:
- **Memory:** Write down experiences and refer to them later.
- **Planning:** Use the same or different LLM for creating plans.
- **Execution:** Execute output and receive feedback from external entities like digital twins, code execution environments, or through self-reflection.
- Tools usage: API call to other tools such as search engine, RAG, optimization engine, database and operating system.

System Message and Prompting Techniques:

- Utilize sophisticated system prompts and techniques (chain of thought, tree of thought).
- Employ short examples and in-context learning.

Knowledge Base:

- Access a set of documents, texts, graphs, etc.
- Store knowledge as an adapter (Lora adapter), modifying a small portion of LLM's weights (0.1% - 1%).
- Run adapters simultaneously or via a router network with minimal overhead.
- Create adapters for different topics, leveraging the RAG framework.

The LLM agent can use one or multiple LLMs as its "brain," making one or several calls to its core LLM to complete tasks. The agent can reflect on its outputs and improve its own performance. This is an emerging field that is becoming of significant importance in the future for automating tasks in real-world applications, including mobile networks.

OpenAI Swarm Agents

The OpenAI Swarm framework provides the foundation for lightweight and dynamic multi-agent orchestration in this project. Below, there is a detailed explanation of the Swarm agents and their role in the O-RAN monitoring system.

Key Features

1. **Lightweight Design**:
    - Swarm agents operate with minimal resource overhead, enabling efficient deployment in research and experimental settings.
2. **Dynamic Handoff Mechanisms**:

- Tasks can be reassigned dynamically between agents, ensuring smooth coordination and flexibility.

3. **Stateless Operations**:
   - Agents do not accumulate internal state, making them adaptable and modular for highly dynamic workflows.
4. **Research and Scalability Focus**:
   - The Swarm framework is tailored for exploratory applications, making it ideal for developing scalable and intelligent multi-agent solutions.

Swarm Agent Roles in O-RAN Monitoring

1. **Coordination**:
   - The Swarm framework manages agent interactions, ensuring seamless communication and task delegation.
2. **Specialization**:
   - Agents are designed to handle specific tasks, such as monitoring Docker containers or analyzing logs, enabling focused and efficient execution.

Comparison to Other Frameworks

The OpenAI Swarm agents differ from other agentic frameworks, such as Microsoft AutoGen and CrewAI, as covered in Table 7, by focusing on lightweight design and stateless interactions, making them particularly well-suited for research environments that require exploration of multi-agent dynamics and scalable solutions for tasks like O-RAN monitoring.

TABLE 7. SUMMARY TABLE OF AGENT FRAMEWORK FEATURES

Framework	Key Features	Applications
AutoGen (Microsoft)	Modular, scalable design, asynchronous messaging	Autonomous task execution, multi-agent collaboration
CrewAI	Role-based collaboration, customizable agent behaviors	Smart assistants, automated customer service
Swarm (OpenAI)	Lightweight, stateless, dynamic handoffs	Research, scalable AI solutions

The developed framework employs a variety of specialized agents. Each agent tailors to specific tasks in the LLM-based O-RAN monitoring system:

1. **Doorkeeper Agent**:

- **Role**: Acts as the central coordinator for the system, receiving user queries and routing them to the appropriate specialized agents.
- **Functionality**:
  - Evaluates query context and assigns tasks to agents.
  - Ensures seamless communication and task handoffs.

2. **Docker Agent**:
   - **Role**: Monitors the Docker environment hosting O-RAN components.
   - **Functionality**:
     - Lists running Docker containers.
     - Fetches performance metrics and resource utilization.
     - Classifies containers by their associated O-RAN components.
3. **Open5GS Log Analyzer Agent**:
   - **Role**: Analyses logs from the Open5GS core to identify errors, performance issues, and operational statuses.
   - **Functionality**:
     - Extracts logs from containers.
     - Summarizes system health and provides actionable insights.
4. **RIC Log Analyzer Agent**:
   - **Role**: Analyses logs generated by the RAN Intelligent Controller (RIC).
   - **Functionality**:
     - Identifies anomalies and errors in RIC operations.
     - Provides summaries for performance monitoring.
5. **Specialized Log Analyzer Agents**:
   - **Role**: Handle advanced queries requiring detailed log analysis and context-aware responses.
   - **Functionality**:
     - Leverage the RAG framework to incorporate contextual information from documentation, source code, and logs.
     - Perform complex log parsing to address user queries.
6. **SRS gNB Control Agent**:
   - **Role**: Manages configurations for the SRS gNB, including slicing and resource allocation.
   - **Functionality**:
     - Adjusts Physical Resource Blocks (PRBs) and other parameters.
     - Validates configuration changes to prevent errors.

2.8.2.3 Deployment of the LLM-based O-RAN Monitoring Platform

Overview

The LLM-based O-RAN Monitoring Platform depicted in Figure 74 integrates various components to provide comprehensive monitoring and management of O-RAN environments. This section outlines the general concepts and strategies employed to set up the platform effectively.

Deployment Architecture

- **Containerization with Docker Compose:**
  - Encapsulates all components—including O-RAN elements, LLM-driven agents, the log collection system, and the user interface—in Docker containers.
  - Utilizes Docker Compose for orchestration, simplifying deployment and ensuring consistency.
- **O-RAN Stack Deployment with ZeroMQ (ZMQ):**
  - **ZeroMQ Protocol:**
    - Facilitates efficient, low-latency communication between O-RAN components.
  - **Open5GS Core Network:**
    - Deployed in Docker containers to provide core network functionalities.
  - **SRS gNB and SRS UE:**
    - **SRS gNB:** Acts as the base station, communicating with the 5G core using standard interfaces (i.e., N2, N3).
    - **SRS UE:** Simulated user equipment from the SRS 4G RAN project, communicating with the gNB through ZMQ.
  - **O-RAN SC RIC:**
    - The RAN Intelligent Controller sourced from the O-RAN Community RIC platform in srsRAN github repository[3].
- **xApp Deployment:**
  - **Python-based xApps:**
    - Developed as Python functions and deployed within the RIC framework to extend its capabilities.

Deployment Steps

a) Deploy O-RAN Components Using Docker Compose:

- **Open5GS Core Network:**
  - Launch core network services in Docker containers.
- **SRS gNB and SRS UE:**
  - Run SRS gNB configured to communicate with the Open5GS core over ZMQ.

[3] https://github.com/srsran/oran-sc-ric/tree/main

- Run SRS UE to simulate user devices for testing and monitoring.
- **O-RAN SC RIC:**
  - Deploy the RIC platform from the O-RAN Community.

b) Deploy xApps Within the RIC Framework:

- Deploy Python-based xApps inside the RIC to provide specialized monitoring, analytics, and control functionalities.
- xApps are integrated as functions within the RIC framework, leveraging its services and APIs.

c) Deploy LLM Agents and Supporting Services:

- Use Docker Compose to deploy agents (e.g., Doorkeeper Agent, Log Analyzer Agents) as Docker containers.
- Set up the centralized **Log Collection System** to aggregate logs from all components.
- Deploy the **Retrieval-Augmented Generation (RAG) Backend** to integrate contextual information from documentation, source code, and logs.

d) Deploy the User Interface:

- Deploy the **Gradio-based GUI** as a Docker container to provide a user-friendly interface.
- The UI allows users to input queries, view analysis results, and monitor system performance.

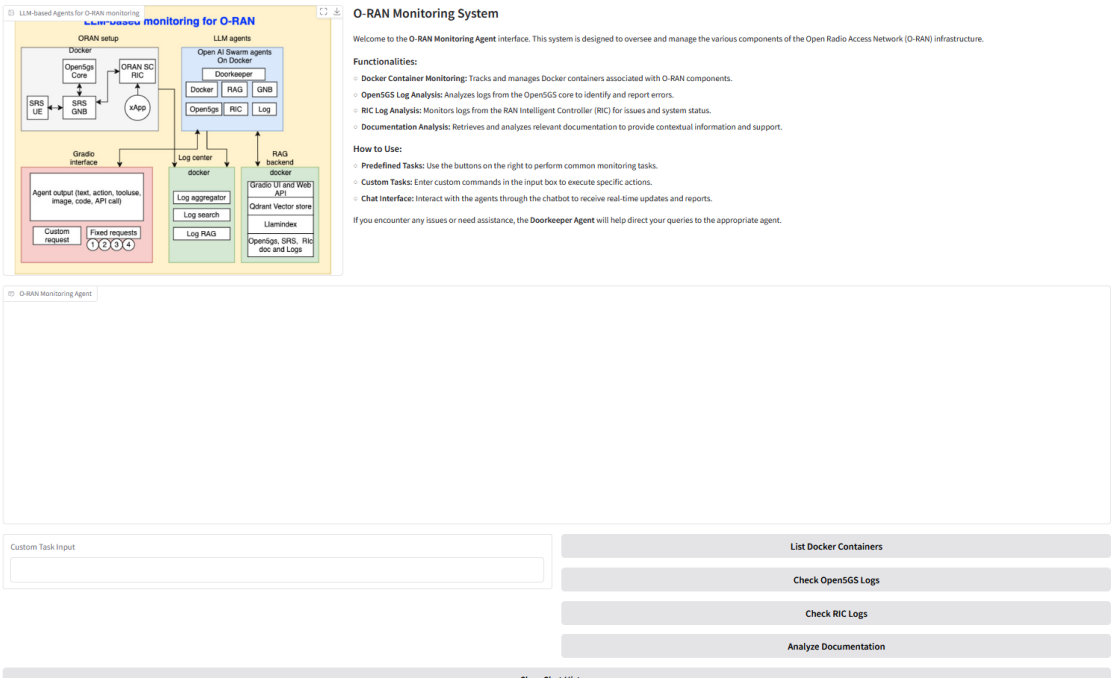


FIGURE 74. RAG FRAMEWORK DEVELOPED

2.8.2.4 Results

The development of the agentic framework has demonstrated a reliable and scalable approach to automating the supervision of containerized environments. The framework is designed to interface directly with Docker infrastructures, enabling it to continuously assess the status of active containers and identify deviations from expected operational behaviour. Through systematic retrieval and inspection of container logs, the system provides timely insights into errors, performance irregularities, and other conditions that may affect service stability, as depicted in Figure 75. This automated log analysis reduces the need for manual oversight and ensures that operational issues can be detected at an early stage.

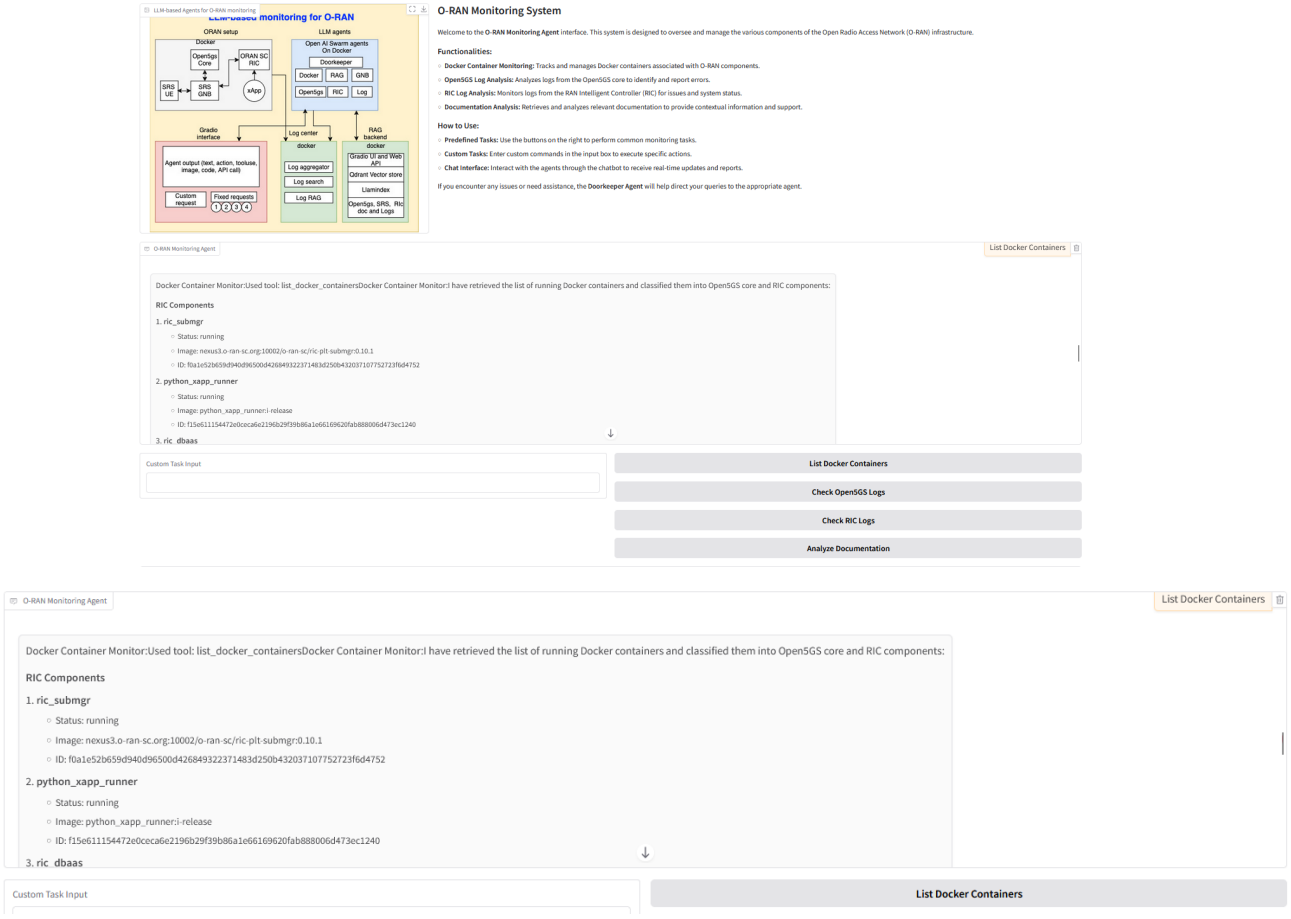


FIGURE 75. CHECK OF THE AVAILABLE DOCKER CONTAINERS DONE THROUGH THE AGENTIC PLATFORM

A notable capability of the framework is its ability to initiate and manage xApps that support specialized monitoring tasks. These xApps can be deployed dynamically based on system conditions or predefined triggers, allowing the monitoring process to be extended or customized without

interrupting ongoing operations. This dynamic integration enhances the overall observability of the environment and supports more precise diagnostic and analytical activities.

Overall, the performed tests indicates that the capabilities of the developed agentic framework [4] significantly improves the efficiency and reliability of container monitoring. By combining automated status checks, structured log analysis, and on-demand deployment of monitoring xApps, the system establishes a robust mechanism for maintaining operational awareness in distributed and containerized infrastructures. This approach reduces manual workload, supports faster issue resolution, and provides a solid foundation for future extensions in intelligent system management.

2.8.2.5 Conclusions

By utilizing Docker for containerization and integrating LLM-driven agents and Python-based xApps within the RIC framework, the O-RAN Monitoring Platform delivers comprehensive monitoring and management capabilities. This deployment strategy ensures a flexible, scalable, and robust system suitable for various O-RAN environments, simplifying deployment while allowing for customization and expansion to meet evolving requirements.

2.9 JOINT-K3.2: AI/ML Platform (AIMLP)

2.9.1 System design

AI/ML Platform is implemented as a Python-based microservice within the Network Digital Twin depicted in Figure 76. It serves as the analytical backbone of the system, leveraging machine learning models for predictive analytics and performance optimization.

The main features of the AI/ML platform are:

- Real-time KPI analysis
- Predictive maintenance algorithms
- Configuration optimization models

[4] https://gitlab.cttc.es/mdi-6gblur/joint/-/tree/main/KeyConcepts/JOINT%20K3.1%20Agentic%20Monitoring%20Platform?ref_type=heads

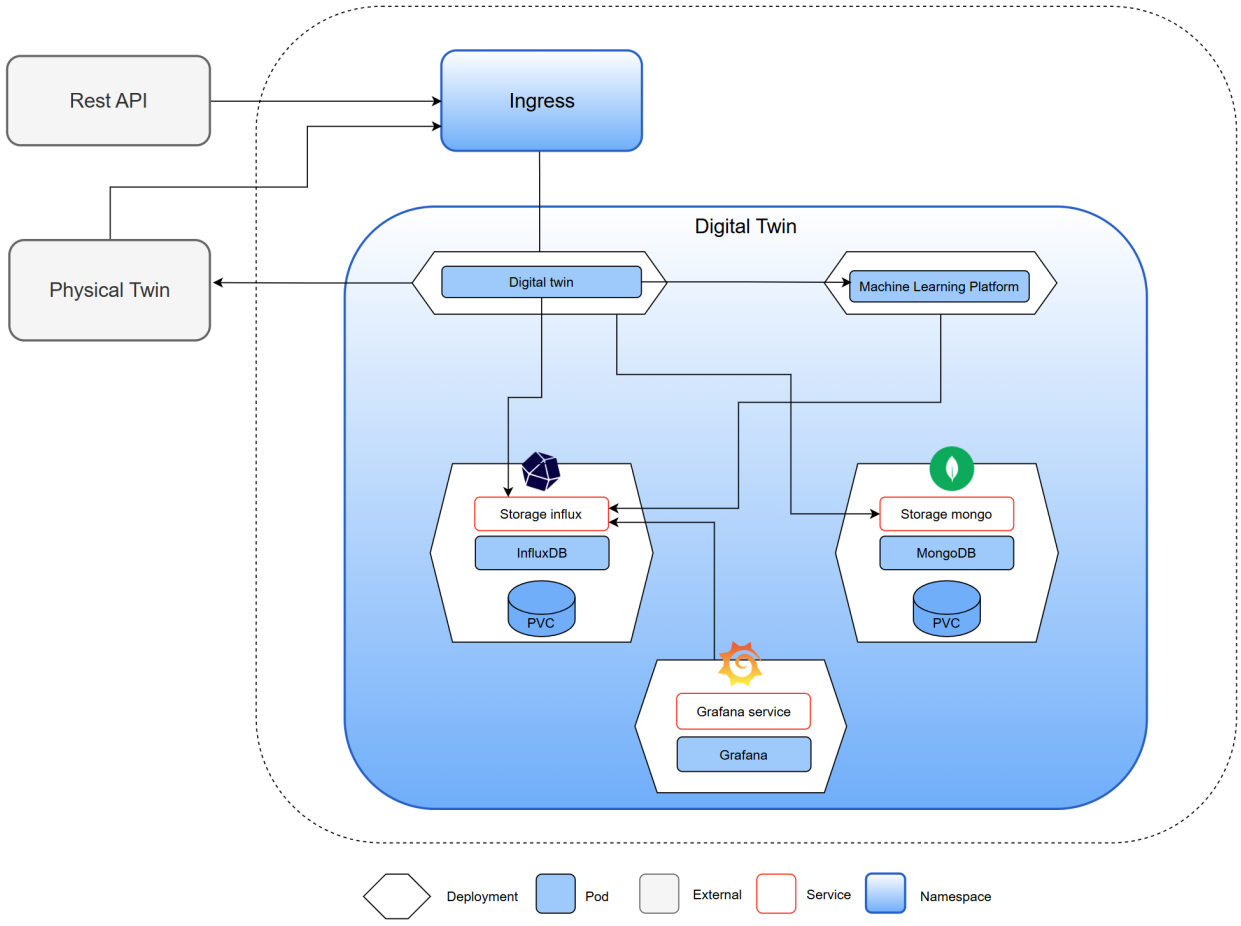


FIGURE 76. MACHINE LEARNING PLATFORM

2.9.2 Implementation

Data Exploration and Analysis:

The collected data originates from NPN Characterization, various experiments, and real-world use case experiments conducted on the NDT, each lasting a different duration. These experiments were facilitated by APIs (Refer to the Section 2.4.2) specifically developed for the automatic execution and collection of the necessary data through the configuration and execution of a battery of experiments. This automated approach ensured the creation of a comprehensive database, which serves as the foundation for all subsequent analysis and predictive solution development using machine learning. Each experiment involved the execution of varied parameter configurations for the NDT settings and a traffic model, ensuring a diverse and robust dataset suitable for training and evaluating predictive models.

The raw dataset comprises 40 unique parameter configurations, with multiple traffic models executed for each, resulting in diverse performance samples. Key performance metrics include energy consumption, one-way delay (OWD), and traffic received. The dataset, free of missing values, includes both categorical and numerical variables, making it well-suited for exploratory analysis and predictive modelling.
Data preprocessing involved cleaning, variable selection, and transformations like One-Hot Encoding for categorical variables and scaling for numerical features to ensure equitable model training. A pipeline integrates these preprocessing stages with model training, enhancing consistency and preventing data leakage.
Exploratory Data Analysis (EDA) revealed important correlations: traffic received is strongly correlated with traffic injected, while energy consumption shows moderate correlations with these traffic metrics. Delays, however, are influenced by categorical variables like direction and protocol, suggesting the need for complex modelling techniques. Descriptive statistics highlighted distribution patterns and the impact of categorical variables on target metrics, emphasizing their significance in model development.

Raw Data:

The Table 8 below contains the collected raw data samples for different types of use case executions at the NPN system.

TABLE 8. RAW DATA SAMPLE.

id	start_time	stop_time	duration	configuration_cell_name	configuration_bandwidth	configuration_power	configuration_antenna	configuration_tdd_pattern	energy	owd	traffic_model	traffic_received	traffic_sent
006321cc-633b-4df9-b73d-8c7e5e84684f	2024-06-26T16:51:31.525361523Z	2024-06-26T16:52:32.757544522Z	60	PELICAN_5G_DOT-1	40	1004	RD4479B78L	DDDSU(10:2:2)	0.74,0.47,0.63,0.8,0.37,0.73,0.47,0.63,0.03,0.13	296181908, 296301779, 296539951, 295315519, 295729536, 294187097, 296468752, 295639050, 294707463, 296092280	uplink_UDP_120M	27.940169564986405,27.955986159093 94,27.930081376845 2,28.52385829713195,28.00731695247287 7,27.931272916411377,27.938538186285808,27.945326234199385,27.932300358656203,27.932300358656203	123.43762172647875,123.51269851864329,123.45158621082237,123.49115464246913,123.46752685825786,123.45280680109232,123.48809042153728,123.47376729 9753,123.49366738604661,123.47132656794848
00f9d464-071d-46f2-b98f-99919b637ca5	2024-06-26T15:36:54.487597718Z	2024-06-26T15:37:55.983240647Z	60	PELICAN_5G_DOT-1	20	1004	RD4479B78L	DDDSUDDSUU(10:2:2)	0.37,0.73,0.47,0.63,0.03,0.13,0.73,0.47,0.63,0.8	519666315, 539733649, 497439809, 519619573, 516947830, 509527009, 501709014, 486166412, 484560798, 482452863	uplink_UDP_150M	15.905741121522565,15.1740725187158 34,15.977648047088252,16.760465176369998,16.250121813285 39,15.479174125405743,16.545358722659824,16.708693434993428,16.781539803342 16,17.042650128 03471	154.35542387062077,154.32637335651947,154.34422969788184,154.32247496330123,154.36649507484617,154.35016831532 86,154.31300843524377,154.34426726158404,154.36187350680 18,154.32894079485223

As we can see in the example of the attached table, there are 14 columns, the contents of which are detailed below:

- id: Unique identifier of the experiment for each record.
- start_time: Start timestamp of the measurement of the experiment.
- stop_time: Stop timestamp of the measurement of the experiment.
- duration: Duration of the measurement of the experiment (in seconds).

- configuration_cell_name: Name of the cell configuration.
- configuration_bandwidth: Bandwidth configuration.
- configuration_power: Power configuration.
- configuration_antenna: Antenna type.
- configuration_tdd_pattern: TDD pattern configuration.
- energy: Energy measurement values for the duration of the experiment, presented as a series of comma-separated numbers, with each number representing a measurement taken every 6 seconds.
- owd: One-way delay (OWD) values as a series of comma-separated numbers. Each number is the measurement gotten every 6 seconds.
- traffic_model: Type of traffic model.
- traffic_received: Received traffic values as a series of comma-separated numbers. Each number is the measurement gotten every 6 seconds.
- energy_total: Total energy consumption measured for the duration of the experiment.

Predictive Model Development

Data Splitting
The objective for data splitting is to ensure robust model training and evaluation by dividing the dataset into training and testing subsets.

Steps Taken:
For each target variable (traffic_received_mean, energy_total, owd_mean):

1. **Dataset Division**:
    - The dataset was split into:
        - **80% Training Data**: Used for training the model and optimizing its parameters.
        - **20% Test Data**: Reserved for evaluating the model's performance on unseen data.
2. **Stratification of Target Variables**:
    - The splits ensured a good balance of the target variable values across both training and testing subsets.

Justifications for Each Action:
Why Split the Data?

- Splitting the dataset is a critical step in model development to prevent overfitting and ensure that the model generalizes well to new, unseen data.
- **Training Data**: By using 80% of the data, the model has sufficient information to learn patterns, relationships, and interactions within the dataset.

- **Test Data**: The remaining 20% provides an unbiased evaluation of the model's performance, simulating how it would perform on real-world data.

Why 80/20 Split?

- This ratio strikes a balance between providing enough data for training while reserving a substantial portion for reliable testing.
- It is a commonly accepted standard in machine learning practice, ensuring neither undertraining nor insufficient testing.

Why Ensure Balance in Splits?

- **Target Variable Distribution**:
    - For numerical targets like traffic_received_mean, energy_total, and owd_mean, ensuring a balanced distribution in training and testing sets prevents skewed predictions.
    - This balance avoids bias introduced by underrepresented scenarios in either split.
- **Dataset Integrity**:
    - The dataset includes diverse configurations (configuration_bandwidth, direction, protocol, configuration_tdd_pattern). Stratification ensures these features are proportionately represented in both subsets, maintaining the integrity of their impact on target variables.

Implementation Details:

- The dataset was split using a random seed for reproducibility, ensuring consistent results across multiple runs.
- Libraries such as sci-kit-learn[11] were used for splitting, leveraging built-in functionality to maintain balance in the target variable distributions.

Benefits of This Approach:

1. **Reduced Overfitting**:
    - By testing on unseen data, the model is less likely to memorize patterns from the training set, enhancing generalization.
2. **Reliable Performance Metrics**:
    - Metrics like Mean Squared Error (MSE), Mean Absolute Error (MAE), and R^2 Score calculated on the test set provide a realistic evaluation of the model's performance.
3. **Balanced Representations**:
    - Proportional representation ensures that both high and low values of target variables are included in both subsets, reflecting real-world diversity.

- Following the visualization of the target variable *traffic_received_mean* distribution for the training and test sets, illustrating the balance achieved during the data splitting process:

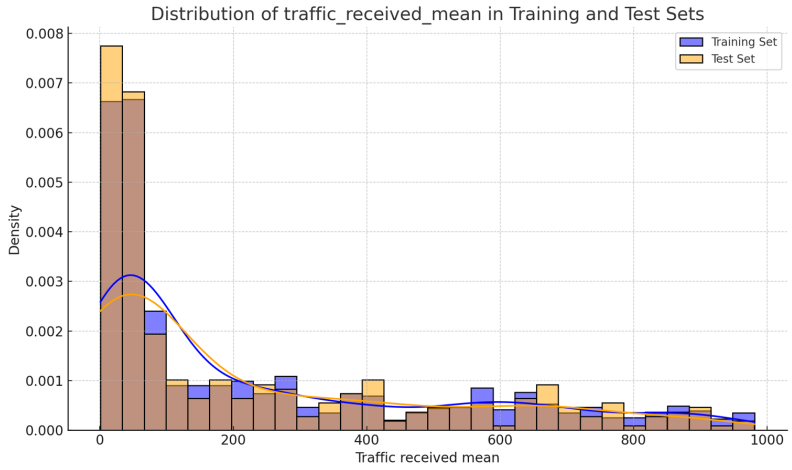


FIGURE 77. TRAINING VS TEST SET

Model Creation

Models Tested

The following predictive models were tested to evaluate their effectiveness in predicting the target variables (traffic_received_mean, energy_total, owd_mean). Each model was chosen for its unique advantages and its potential to leverage the characteristics of the dataset.

Tree-Based Ensemble Models

1. **RandomForestRegressor:**
   - **Strengths**: Captures non-linear relationships, robust to overfitting when properly tuned, and provides feature importance.
   - **Why Suitable**: Handles both categorical (e.g., direction, protocol) and numerical features (e.g., configuration_bandwidth, traffic_injected) effectively. Its ensemble nature makes it resilient to noise in the dataset.
   - **Use Case**: Ideal for scenarios requiring interpretability of feature importance and moderate computational efficiency.
2. **GradientBoostingRegressor:**
   - **Strengths**: Performs well on smaller datasets, focuses on reducing errors iteratively, and captures complex patterns.
   - **Why Suitable**: Effective for modeling non-linear interactions, such as the relationship between configuration_bandwidth and traffic_received_mean.

- **Use Case**: Suitable for datasets with moderately high dimensions and when high predictive accuracy is needed.

3. **XGBoost:**
   - **Strengths**: Highly optimized for speed and accuracy, includes regularization to prevent overfitting.
   - **Why Suitable**: Can handle missing data and complex interdependencies between features like configuration_tdd_pattern and target variables.
   - **Use Case**: Best for achieving high accuracy in datasets with mixed data types and higher computational budgets.
4. **CatBoostRegressor:**
   - **Strengths**: Handles categorical features directly, reducing preprocessing effort, and is robust to overfitting.
   - **Why Suitable**: The dataset includes significant categorical variables (direction, protocol), making CatBoost highly efficient and effective.
   - **Use Case**: Ideal when categorical variables significantly impact target predictions.
5. **LightGBM:**
   - **Strengths**: Highly efficient with large datasets and features, handles categorical data natively.
   - **Why Suitable**: Handles the mix of categorical and numerical data effectively, scales well for high-dimensional datasets.
   - **Use Case**: Recommended for large datasets with multiple categorical variables.
6. **ExtraTreesRegressor:**
   - **Strengths**: Robust to outliers, captures non-linear relationships, and provides feature importance insights.
   - **Why Suitable**: Handles high-dimensional data efficiently and performs well on datasets with imbalanced features.
   - **Use Case**: Selected for training as it combines robustness with computational efficiency.
7. **AdaBoostRegressor:**
   - **Strengths**: Combines weak learners iteratively to reduce bias and variance.
   - **Why Suitable**: Captures patterns overlooked by simpler models, suitable for moderate dataset sizes.
   - **Use Case**: Effective when dealing with challenging target variable distributions.

Linear Models

1. **LinearRegression:**

- **Strengths**: Simple and interpretable, ideal for linear relationships.
- **Why Suitable**: Serves as a baseline model to evaluate other complex models.
- **Use Case**: Effective for identifying direct correlations in features like configuration_bandwidth and traffic_received_mean.

2. **Ridge, Lasso, and ElasticNet:**

- **Strengths**: Introduce regularization to address multicollinearity and enhance generalization.
- **Why Suitable**: Useful for feature selection and handling collinear features like traffic_injected and traffic_received_mean.
- **Use Case**: Ridge for simple regularization, Lasso for feature selection, ElasticNet for combining the two.

Support Vector Machines

1. **Support Vector Regressor (SVR)**:

- **Strengths**: Captures non-linear relationships using kernel functions.
- **Why Suitable**: Useful for predicting owd_mean, which may depend on complex interactions.
- **Use Case**: Best for smaller datasets with non-linear patterns.

Instance-Based Models

1. **KNeighborsRegressor**:

- **Strengths**: Makes predictions based on proximity to neighbors, requires minimal assumptions.
- **Why Suitable**: Effective for local patterns, such as the effect of similar configuration_tdd_pattern values.
- **Use Case**: Useful for small datasets or when interpretability of local decisions is needed.

Other Robust Models

1. **HuberRegressor:**

- **Strengths**: Robust to outliers, combines linear regression with resistance to extreme deviations.
- **Why Suitable**: Ideal for owd_mean, which includes significant outliers.
- **Use Case**: Effective for targets with heavy-tailed distributions.

2. **BaggingRegressor (with Decision Trees):**

- **Strengths**: Reduces variance by aggregating predictions of multiple decision trees.

- **Why Suitable**: Provides stability and robustness in datasets with mixed feature types.
- **Use Case**: Useful for datasets with high variance in target variables.

Model Training

For each target variable (traffic_received_mean, energy_total, owd_mean), **15 models** were trained, leveraging their unique strengths to evaluate their predictive capabilities. The training process was carefully designed to ensure fair comparison and optimal performance for each model. The models trained included a mix of ensemble methods, linear regressors, and robust algorithms, as previously detailed.

1. **Data Preparation**

The training process began with preprocessing steps outlined earlier:

- **Feature Encoding**: Categorical features (direction, protocol, configuration_tdd_pattern) were encoded using One-Hot Encoding.
- **Scaling**: Numerical features (configuration_bandwidth, traffic_injected) were standardized using *"StandardScaler" [12]* to ensure uniform input across all models.
- **Data Splitting**: The dataset was split into an 80/20 ratio for training and testing, ensuring stratified splits for target variable distributions.

2. **Model Training Process**

Each of the **15 models** was trained for every target variable using the following procedure:

1. **Baseline Training**:
   - All models were initially trained using default hyperparameters to establish a baseline performance.
   - Models evaluated included tree-based methods (e.g., RandomForest, GradientBoosting, CatBoost), linear regressors (e.g., Ridge, Lasso), and others (e.g., SVR, Huber).

2. **Hyperparameter Optimization**:
   - To enhance performance, hyperparameters were tuned using *GridSearchCV[13]* or equivalent tuning methods, optimizing parameters such as:
     - **Tree-Based Models**: Number of estimators, maximum depth, learning rate.
     - **Linear Models**: Regularization strength (alpha).
     - **SVR**: Kernel type and regularization parameters.
   - This process minimized overfitting while ensuring robust performance on test data.

3. **Cross-Validation**:

- 5-fold cross-validation was used during training to evaluate model generalizability and prevent overfitting.

4. **Iterative Training**:
   - Performance metrics were monitored during training to identify underperforming models, refine hyperparameters, and iteratively improve.

5. **Evaluation of Training Data**:

Training accuracy and metrics were computed to identify potential underfitting or overfitting before moving to test data evaluation.

2.9.3 Results

Model Evaluation and Performance

To generate the required data to perform the full characterization of the NPN system, it is required an exhaustive measurement campaign. Indeed, in this case, multiple measurement campaigns have been done: some when the NPN was in preparation phase at 5TONIC lab and then, a second battery when deployed at CTTC premises to ensure the NPN system is operating with high efficiency. The generated experiment datasets are automatically collected and fed to ML models to update the training and prediction datasets to improve the AI/ML platform intelligence and accuracy of the prediction and recommendation capabilities.

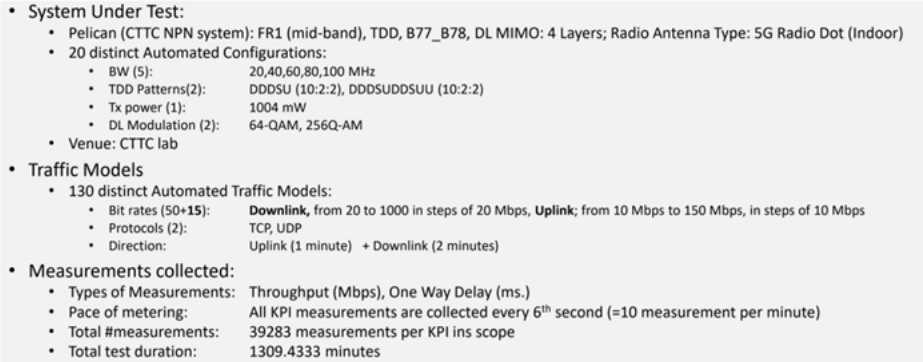


FIGURE 78. DESCRIPTION OF A MEASUREMENT CAMPAIGN DRIVEN IN CUSTOMER PREMISES TO CHARACTERIZE THE NPN SYSTEM AND FEED AI/ML MODELS

Model Evaluation

Models were evaluated on the test set using the following metrics:

- **Mean Absolute Error (MAE)**: Reflects average prediction error magnitude, interpretable in real-world units.
- **Mean Squared Error (MSE)**: Penalizes large errors, offering a comprehensive performance measure.
- **Root Mean Squared Error (RMSE)**: A direct interpretation of MSE in the same unit as the target variable.
- **R^2 Score**: Indicates the proportion of variance explained by the model, with higher values reflecting better fit.

Performance Results by Target Variable

Following is a summary of the performance of all evaluated models across the three target variables (traffic_received_mean, owd_mean, energy_total).

Traffic Throughput [Mbps] (traffic_received_mean):

TABLE 9. TRAFFIC THROUGHPUT [MBPS]

Model	MSE	RMSE	MAE	R^2 Score
RandomForest	139.37	11.81	5.82	0.9980
GradientBoosting	226.95	15.06	9.89	0.9967
XGBoost	140.92	11.87	6.14	0.9979
CatBoost	136.15	11.67	7.02	0.9980
LightGBM	128.85	11.35	6.60	0.9981
AdaBoost	1228.85	35.05	26.78	0.9823
ExtraTrees	133.86	11.57	5.41	0.9981
KNeighbors	315.69	17.77	10.56	0.9955
LinearRegression	3124.42	55.90	40.05	0.9550
Ridge	3121.57	55.87	40.01	0.9551
Lasso	3102.51	55.70	39.70	0.9554
ElasticNet	8767.76	93.64	71.38	0.8738

Model	MSE	RMSE	MAE	R^2 Score
SVR	30578.10	174.87	105.78	0.5600
Huber	3645.68	60.38	33.86	0.9475
Bagging	149.70	12.24	5.99	0.9978

One-Way Delay [Nanoseconds] (owd_mean):

TABLE 10. ONE WAY DELAY [NANOSECONDS]

Model	MSE	RMSE	MAE	R^2 Score
RandomForest	3.37E+15	58031814	24956319	0.9091
GradientBoosting	6.16E+15	78517138	44123287	0.8336
XGBoost	3.12E+15	55831297	25358265	0.9159
CatBoost	3.75E+15	61268104	28164047	0.8987
LightGBM	4.68E+15	68420891	34533585	0.8736
AdaBoost	1.25E+16	111708094	84456887	0.6632
ExtraTrees	3.99E+15	63141272	25794088	0.8924
KNeighbors	6.67E+15	81654066	35754860	0.8200
LinearRegression	1.88E+16	137004185	102075989	0.4934
Ridge	1.88E+16	137155254	102044055	0.4923
Lasso	1.88E+16	137160099	102073598	0.4922
ElasticNet	2.38E+16	154237749	116004265	0.3579
SVR	5.02E+16	224114516	138087177	-0.3557
Huber	3.93E+16	198152622	139343773	-0.0598
Bagging	3.72E+15	60989798	26241144	0.8996

According to the obtained values, the score for some evaluation factors is not very accurate, thus, this was not considered for the final decision and there is a need to carefully select the metrics. This is further elaborated in next sections.

Energy Consumption [W/h] (energy_total):

TABLE 11. CONSUMED ENERGY [WATTS PER HOUR]

Model	MSE	RMSE	MAE	R^2 Score
RandomForest	2917.96	54.02	38.86	0.5253
GradientBoosting	2279.86	47.75	36.67	0.6291
XGBoost	2813.53	53.04	38.43	0.5423
CatBoost	2171.94	46.60	35.20	0.6467
LightGBM	2219.00	47.11	35.57	0.6390
AdaBoost	2596.94	50.96	42.72	0.5775
ExtraTrees	3330.27	57.71	40.88	0.4583
KNeighbors	2665.66	51.63	37.75	0.5664
LinearRegression	3326.79	57.68	45.28	0.4588
Ridge	3314.11	57.57	45.12	0.4609
Lasso	3317.70	57.60	45.06	0.4603
ElasticNet	3818.01	61.79	51.04	0.3789
SVR	3671.49	60.59	45.01	0.4027
Huber	3561.76	59.68	43.76	0.4206
Bagging	3113.51	55.80	40.28	0.4935

Key Observations

After evaluating all the models across the three target variables (*traffic_received_mean, energy_total, and owd_mean*), the following key observations were made:

General Observations

1. **Ensemble Models Outperformed Others**:
   - Tree-based ensemble models (e.g., RandomForest, ExtraTrees, GradientBoosting, CatBoost, and LightGBM) consistently outperformed simpler linear models and distance-based methods like KNeighbors and SVR.
   - These models were better at capturing non-linear relationships and interactions between features, especially for traffic_received_mean and energy_total.
2. **Impact of Outliers**:
   - Robust models like HuberRegressor demonstrated resilience to outliers, particularly for owd_mean, which has a highly skewed distribution and extreme values.
3. **Feature Dependencies**:
   - The categorical features (direction, protocol, configuration_tdd_pattern) played a significant role in predicting all target variables, as shown in ensemble-based feature importance scores.
   - Numerical features such as traffic_injected were crucial for throughput prediction (traffic_received_mean), while configuration_bandwidth was significant for energy_total.
4. **Performance Trade-offs**:
   - While simpler models like LinearRegression and Ridge were computationally efficient, their inability to capture non-linear relationships limited their accuracy.
   - Models like SVR and AdaBoost struggled with large datasets, leading to poor performance and computational inefficiency.

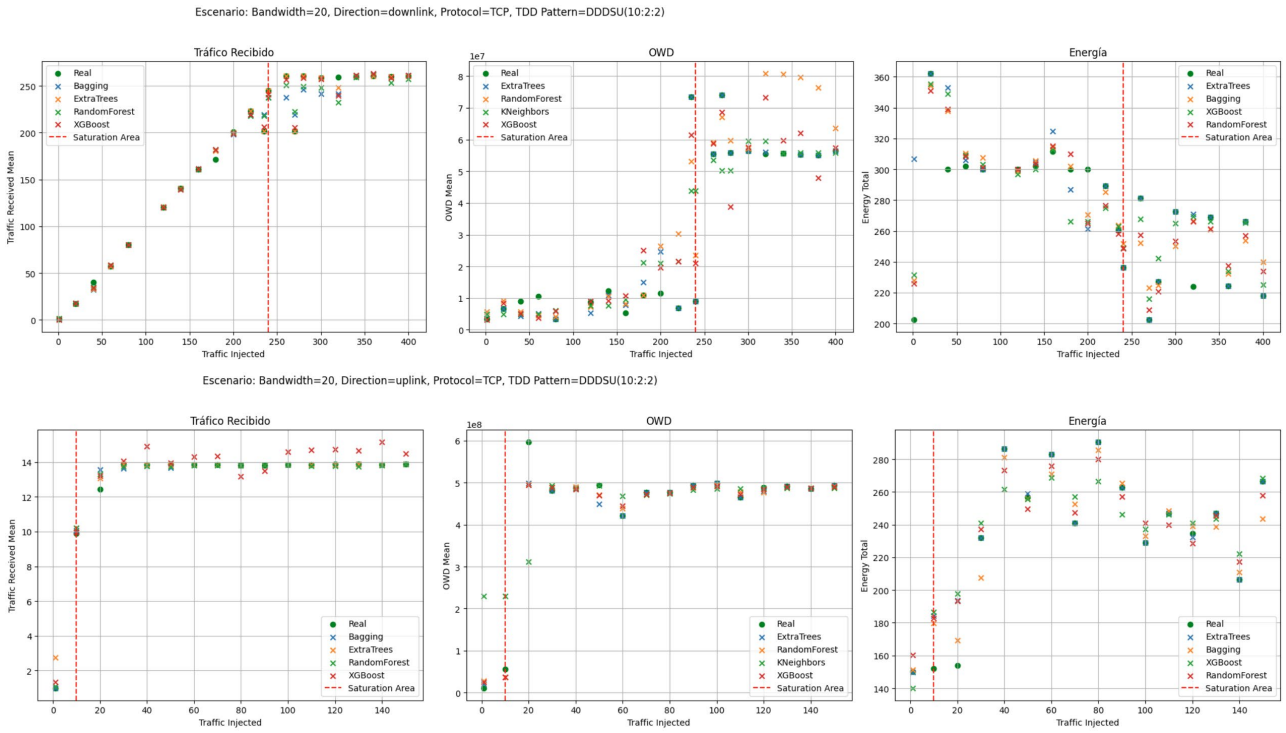
Target-Specific Observations

1. **Throughput (traffic_received_mean)**:
   - LightGBM, CatBoost, and ExtraTrees emerged as the top-performing models with high accuracy (R^2 > 0.998) and low RMSE (<12).
   - These models effectively captured the strong dependency on traffic_injected.
2. **One-Way Delay (owd_mean)**:
   - XGBoost, RandomForest, and ExtraTrees performed well, handling the categorical dependencies and mitigating outliers' effects.
   - Linear models (e.g., Ridge, Lasso) and SVR failed to capture the complex patterns in delay predictions, leading to significantly lower R^2 scores.
3. **Energy Consumption (energy_total)**:

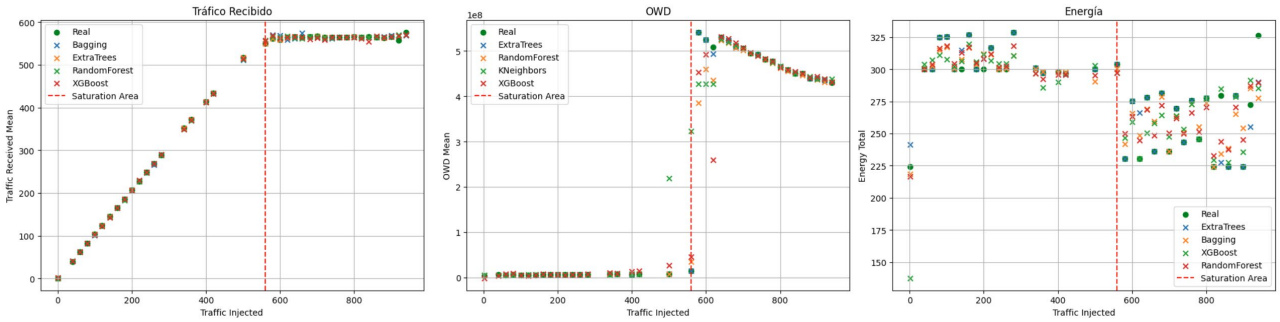
- CatBoost and LightGBM achieved the best balance of accuracy and efficiency, with low RMSE (<50) and MAE values.
- Ensemble models excelled in capturing the interplay between configuration_bandwidth and categorical features.

Final models selection

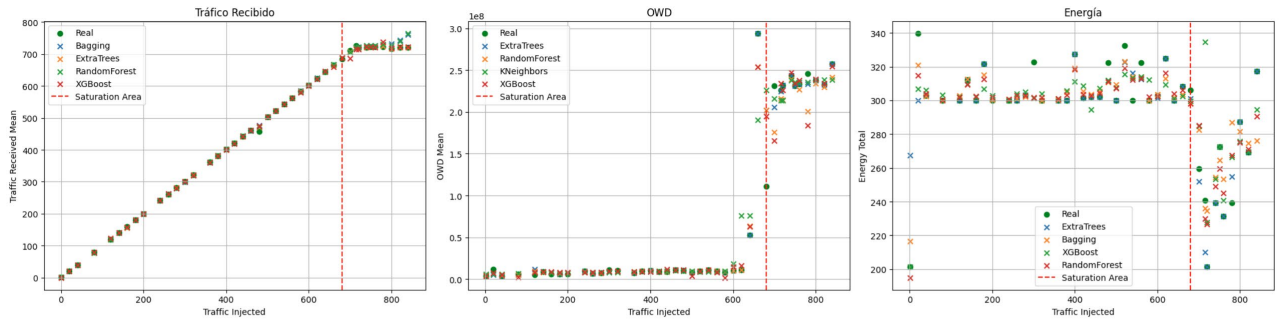
Below some example graphs are attached for comparison, showcasing the models that we have visually identified as performing better in predictions for samples before reaching the saturation threshold.



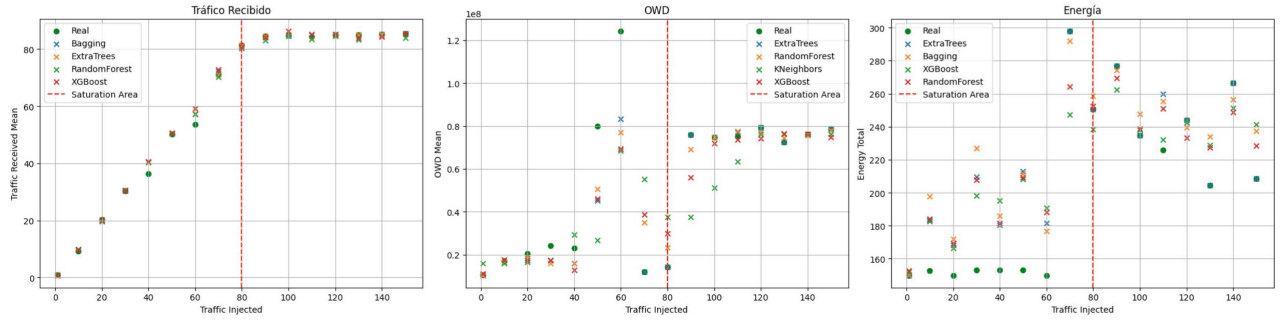
Escenario: Bandwidth=60, Direction=downlink, Protocol=UDP, TDD Pattern=DDDSUDDSUU(10:2:2)



Escenario: Bandwidth=80, Direction=downlink, Protocol=TCP, TDD Pattern=DDDSUDDSUU(10:2:2)



Escenario: Bandwidth=80, Direction=uplink, Protocol=TCP, TDD Pattern=DDDSUDDSUU(10:2:2)



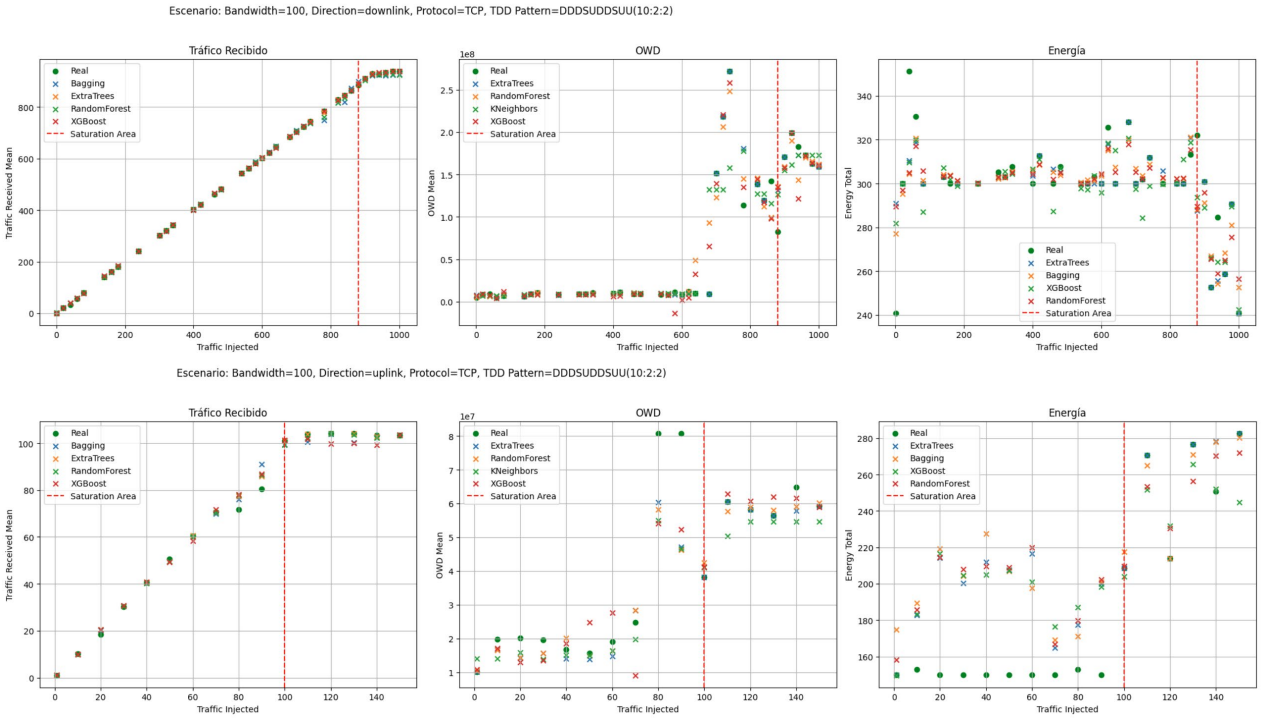


FIGURE 79. PREDICTION GRAPHS

In this analysis, the performance of five predictive models (**ExtraTrees**, **RandomForest**, **XGBoost**, **Bagging**, and **KNeighbors**) was evaluated to predict three key network performance variables: **Traffic Received**, **OWD (One-Way Delay)**, and **Power Consumption**. Each variable was analyzed independently. Below, the results for each variable are described:

Model Behaviour for Traffic Received:

- The **ExtraTrees** model excelled in predicting both the linear growth before the threshold and the stabilization beyond it, showing an almost perfect fit to the real data.
- **RandomForest** and **XGBoost** also delivered highly competitive performance, with predictions nearly identical to ExtraTrees, albeit with slight differences in specific points within the pre-saturation region.
- **Bagging** performed well overall but exhibited minor deviations at lower traffic injection levels, making it slightly less reliable.
- **KNeighbors** showed scattered predictions, especially in the low traffic range, making it less suitable for this variable.

Model Behavior for OWD:

- **ExtraTrees** demonstrated outstanding performance by accurately predicting the stability of OWD in the low-load region and the sharp increase near the saturation threshold. This model consistently matched the real data in all phases.
- **RandomForest** also performed excellently, though slight deviations were observed near the threshold.
- **XGBoost** performed well overall but underestimated the behaviour in the OWD increase range in certain cases.
- **Bagging** was less reliable, with greater dispersion in predictions in the low-load region.
- **KNeighbors**, as with traffic received, showed inconsistent predictions, with values far from the real data.

Model Behavior for Power Consumption:

- **ExtraTrees** delivered outstanding performance, capturing the gradual energy consumption increase in the pre-saturation region and the stabilization near the threshold. Its predictions closely aligned with the real data across all regions.
- **RandomForest** performed very similarly to ExtraTrees but showed slight discrepancies at points of high energy consumption.
- **XGBoost**, while generally accurate, slightly overestimated energy consumption in certain areas, making it less reliable compared to ExtraTrees.
- **Bagging** showed significant dispersion near the saturation threshold, affecting its ability to predict stabilization accurately.
- **KNeighbours** had the worst performance, with scattered and inconsistent predictions.

2.9.4 Conclusions

After independently evaluating each variable, the **ExtraTrees** model was selected as the best option for the following reasons:

1. **High Precision:** ExtraTrees showed exceptional alignment with real data across all variables and regions, both before and after the saturation threshold.
2. **Consistency:** Unlike other models, ExtraTrees provided uniform and reliable predictions for all observed behaviors in each metric.
3. **Versatility:** This model effectively captured both linear patterns (in the case of traffic received) and nonlinear or abrupt changes (in the case of OWD and total energy).
4. **Computational Efficiency:** ExtraTrees offers an efficient implementation that combines fast training (approximately less than 5 seconds on a laptop with an x86 processor, 32 GB RAM, and Intel Iris Xe Graphics GPU with 16 GB of memory) with high predictive power.

In summary, **ExtraTrees** is the ideal choice to independently model each of the key variables in this context, ensuring an optimal balance between accuracy, robustness, and simplicity.

3 Use cases and E2E results

3.1 UC1: 6G Disaggregated Zero-Touch Mobile Network as a Service

3.1.1 PoC1: Zero-Touch distributed 3GPP network for delivering Non-public Networks

3.1.1.1 Introduction

The implementation pathway of this Proof of Concept (PoC) has focused on the practical implementation of the vision concept for Zero-Touch NPN around innovation breakthroughs mentioned in Section 2.3;

- Self-integration of NPN with Public Network,

- Self-integration of NPN with Edge Computing environments,

- Self-configuration of an NPN 5G System according to intent-based declaration of target KPIs

Leveraging both standards such as 3GPP PNI-NPN architectural patterns and major technology trends such as Digital Twin.

3.1.1.2 System Design

With this PoC, we have demonstrated[5] that with Zero-touch capabilities it is extremely easy for the Enterprise to get dedicated 5G connectivity with minimum effort by the CSP. Under such context, the customer will receive a Plug & Play box (carrito/ flight rack) that includes the Radio system and the UPF. A previous pre-integration work (factory configuration) together with self-configured and self-integrated capabilities integrates the private network with the Operator public network, having a fully operational Private 5G Network in few minutes.

The following concepts have been implemented in a practical way in this PoC.

1. Concept 1: **NPN Admin states in Digital Twin**: ready-for-shipment, ready-for-booting, ready-for-service, in-service
2. Concept 2: **NPN Graceful shutdown**; Logic to gracefully power off NPN components to prevent booting issues.

[5] Recording available at: https://gitlab.cttc.es/mdi-6gblur/joint/-/tree/main/PoCs/UC1PoC1%20Zero-Touch%20for%20NPN

3. Concept 3: **NPN Day 0**; NPN system is shipped and delivered with a default pre-verified booting configuration.
4. Concept 4: **NPN Booting Health-check**; Logic that confirms all NPN components are ready for service after booting.
5. Concept 5: **NPN Registration in Digital Twin Platform**; Procedure by which NDT applies final config (Day 1) to NPN
6. Concept 6: **NPN Day 1**: NPN system (Carrito) final config applied by Digital Twin as per Recommender API results
7. Concept 7: **Zero-Touch NPN Start-up Time (KPI)**

3.1.1.3 Implementation

The following picture shows how we understand the delivery flow of an NPN system, within the PNI-NPN concept. The steps it describes are as follows:

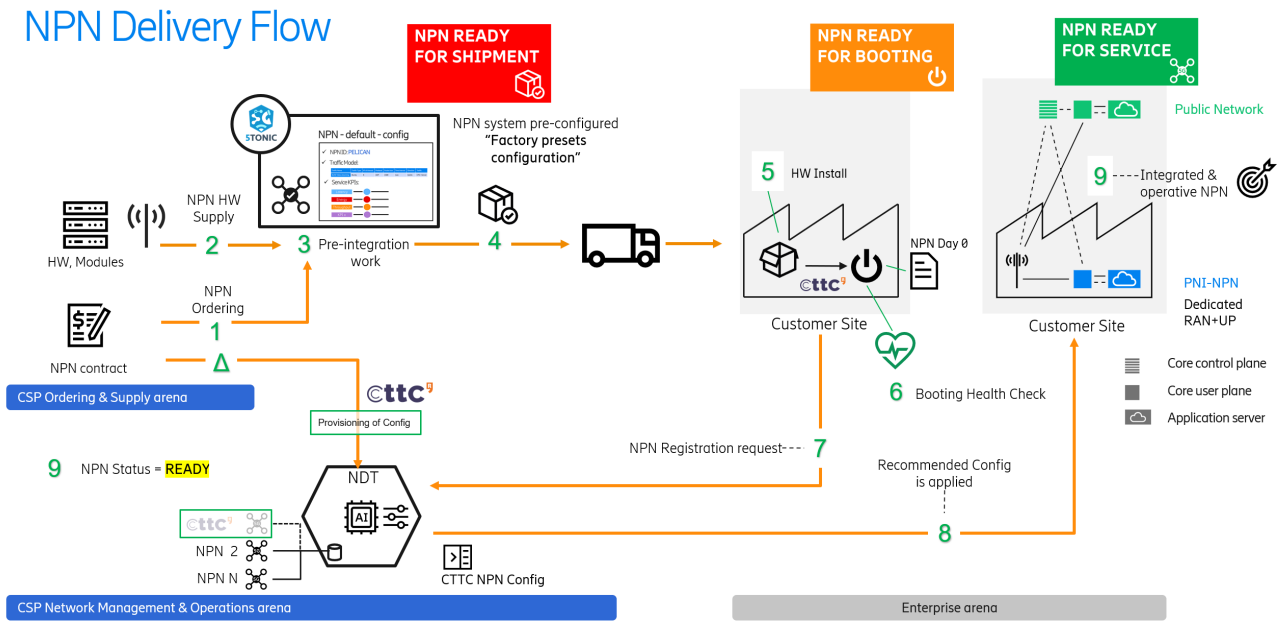


FIGURE 80. NPN DELIVERY FLOW

a) Customer makes a purchase of an NPN system.
b) The HW of the NPN system is provided to the CSP.

c) The system is pre-configured with so-called Factory-presets; All the NPN systems will have the same pre-configuration, which will provide a quick response to the customers and minimize the pre-integration services effort for the CSP.

- All NPN systems will have a so-called Day 0 with a few data specific to the customer/environment where it will be deployed, e.g., the IP Plan (with IP addresses specific to the customer's environment).
- The system is in administrative ***Ready-for-Shipment*** state.

d) Sending the preconfigured NPN system to destination

e) HW installation

f) CSP proceeds to provision in Digital Twin the final configuration of the NPN system according to the customer's request.

In order to promote the NPN system to administrative ***Ready-for-Booting***, following conditions are required (before step e):

- ✓ Authorised Service Provider certified installation (with pictures) of the *"carrito"* including GPS antenna
- ✓ Transport layer between 5GC & NPN Site is operative
- ✓ Site router Ports/IP routes/Firewall rules configured to/from NPN
- ✓ NPN internal cabling has been checked according to installation guide
- ✓ NPN external cabling has been checked (NPN router --- Site Router)
- ✓ NPN connected to power supply
- ✓ Spectrum license issued

g) Power-On

- The logic developed for this PoC includes a Booting Health Check (HC) script; this script is one of the key elements of the Zero Touch process.
- An element-by-element check of the NPN system (Base-Band, LPG, Router, MiniPC, CPE) is done, verifying that all of them start up properly.
- Finally, if the Health check indicates that all the critical elements that allow the NPN to provide services are OK, proceed to the next step.

h) Registration of the NPN system in the Digital Twin platform; NDT is informed that one of the NPNs has "woken up".

i) NDT will check the administrative status of the NPN (which should be Ready-for-Booting) and then apply the final configuration and change its administrative status to ***Ready-for-Service***.

j) Once the customer confirms that the NPN system is serving its subscribers, it will inform the CSP that the NPN is in In-Service administrative status, at which point it will start monitoring the SLAs.

3.1.1.4 Results

PoC Proof Points; Booting sequence + registration in NDT

Here below it is included an example of trace-log of the Booting Health check sequence of a NPN system under test for this PoC.

Most relevant printouts in this log are:

"System uptime 128 seconds": As the Health check script is running in miniPC, it takes 128 seconds to the miniPC to bootup and invoke HC script.

"Waiting 420 seconds for starting": As it takes different time to boot up to the different NPN components, HC script will give time (420 seconds) to all the NPN components to boot up; this way we avoid "false" booting errors (i.e. checking status while some components have not yet completed the booting sequence)

"HC suites": Once the waiting time has concluded, HC script will perform the corresponding checks of the NPN system and remote peers availability (DNS, NDT), such:

- Local miniPC OAM connectivity
- DNS connectivity
- DNS resolution
- NDT connectivity
- NPN Day 0 availability
- CPE status
- Remote connectivity to the NPN components (LPG, R6K, APP Server, Base Band)
- Base band status
- Cell or Cells status

If HC scripts considers that overall status of the NPN system is enough to provide service, it will register in NDT ("calling register API"), informing NDT that NPN is "awake", in order to receive final configuration and changing its administrative state to "ready-for-service".

Finally, the total NPN bootup time is registered (in the logs showed in Figure 81, 606 seconds). This obtained time is quite regular across multiple repetitions of the booting process. Most of this time is consumed by the **"Waiting 420 seconds for starting"** step**.** In particular, it is the LPG component the one being more demanding in terms of bootup time as it needs to recreate a K8s cluster with the corresponding UPF pods.

The former version of this PN system, developed in a pre-6G BLUR project, therefore without ZT capabilities, required few days (3 weeks) to have it fully operative, i.e., configured and fully integrated with the core network in 5TONIC premises. So the reduction time provided by ZT concept will help to increase the scale of deployment of PN networks.

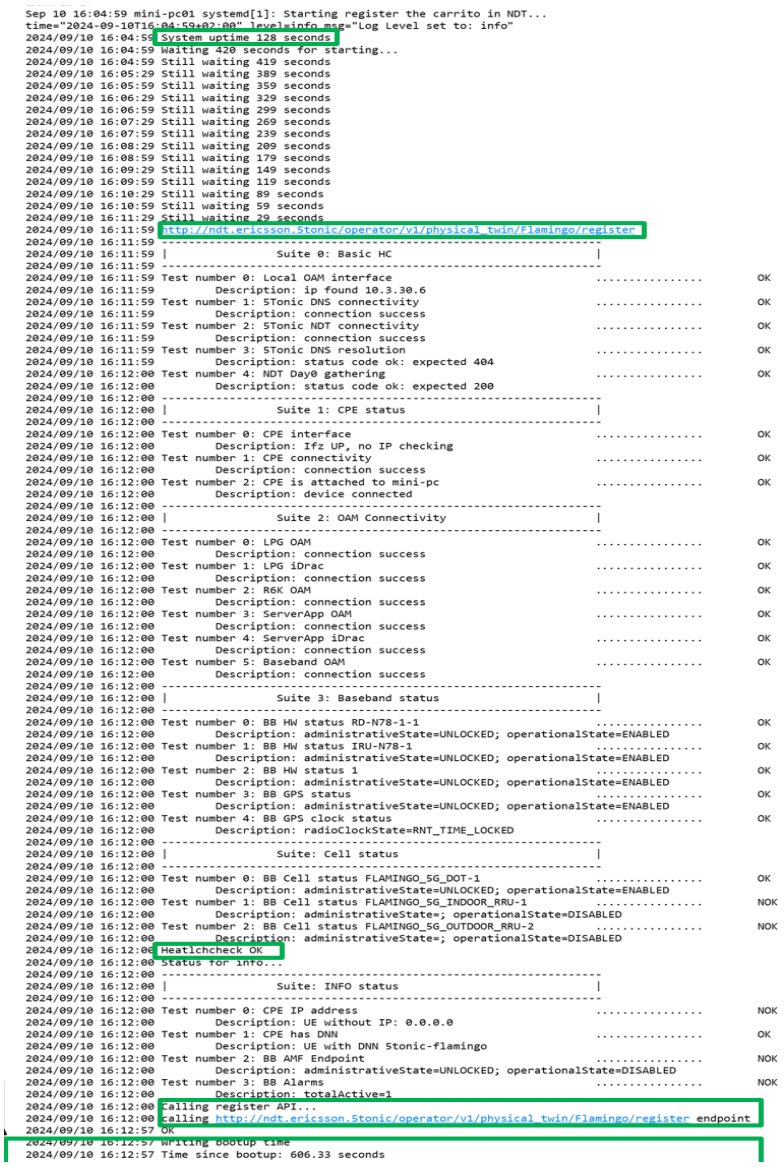


FIGURE 81. NPN BOOTING HEALTH CHECK SEQUENCE + REGISTRATION IN NDT

3.1.1.5 Conclusions and Summary

UC1 PoC1: Zero Touch for delivering Non-public Networks was delivered and showcased in the last 10th October 6G BLUR Plenary meeting[6]. We started from a totally powered off Pelican NPN. After

[6] Recording available for intended project partners (CTTC, Ericsson) at: https://cttcbarcelona.sharepoint.com/sites/PelicanCTTC/_layouts/15/stream.aspx?id=%2Fsites%2FPelicanCTT

provisioning final configuration in NDT and setting-up admin state as ready-for-booting we proceed to power-on the NPN system. After few minutes, NPN system was fully operative and providing 5G connectivity to the CTTC subscribers.

Since then, Zero Touch capabilities has been incorporated and validated in all NPNs ("carritos") being sent to our partners (Flamingo, Parrot, Penguin, Eagle & Pelican).

PoC KPIs

The KPI associated with this PoC was ***Zero-Touch NPN Start-up Time*****:** This KPI measures the time (minutes) it takes for a 5G NPN to auto initialize/auto-configure, auto-integrate with the public network, auto-integrate with the edge network functions, and become fully operational after being started or rebooted, as depicted in Figure 82.

Currently the Zero-Touch NPN Start-up-time ≈ 600 seconds (10 minutes):

It includes booting + health check + registration + final config applied + Integration with 5G Core network for:

- 1. Radio Baseband
- 2. IRU
- 3. LPG (Local Packet Gateway - UPF)
- 4. Router R6K
- 5. Application Server
- 6. NPN Testing-support-CPE

After Zero-Touch NPN Start-up-time Customer has a fully operational Private 5G Network.

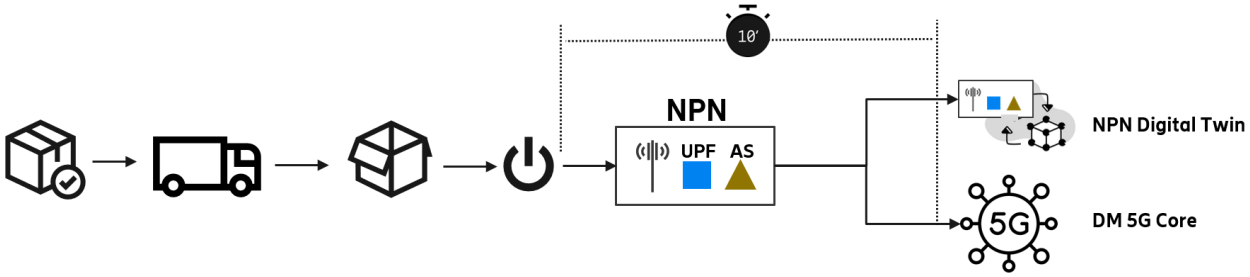


FIGURE 82. ZERO-TOUCH NPN START-UP TIME

C%2FShared%20Documents%2FGeneral%2F6GBLUR%5FUC1%2DPoC1%5FZERO%20TOUCH%20demo%20with%20CTTC%20and%20NPN%20Pelican%2Emp4&referrer=StreamWebApp%2EWeb&referrerScenario=AddressBarCopied%2Eview%2Ed7ea7f29%2D7f2f%2D486a%2D8b9b%2D6ef79a3d0675

3.1.2 PoC2: Distributed O-RAN based mobile network

3.1.2.1 Introduction

This PoC demonstrates the deployment of a flexible and distributed end-to-end cloud-native mobile network based on the O-RAN architecture. Specifically, the PoC involves the deployment of a complete mobile network, including the configuration of network slices to provide differentiated connectivity services to users.

This work builds upon several of the key concepts described previously. More specifically, the involved key concepts are:

- **SMART-K1.1 – CU and DU Implementation**: Implementation of O-RAN Central Unit (CU) and Distributed Unit (DU) components using the srsRAN software.
- **JOINT-K1.1 – Data Plane Reconfiguration and Zero-Touch:** Distributed and Disaggregated cloud-native deployment of mobile core network entities (control plane and data plane).
- **JOINT-K1.2 – Distributed Deployment of End-to-End Slices, Including O-RAN**: Distributed and Disaggregated Cloud-native deployment of all RAN network elements.
- **JOINT-K2.1 – Transport Network Optimization**: Deployment and optimization of transport network flows to support the configured slices in the deployed mobile infrastructure.

3.1.2.2 System Design

The architecture of this PoC is resumed in Figure 83. As the figure depicts, the setup has a distributed deployment based on two locations: the xG EXTREME Testbed located in CTTC (Castelldefels, Spain) and the Telefonica Innovation Lab (Madrid, Spain) which are interconnected through a Virtual Private Network.

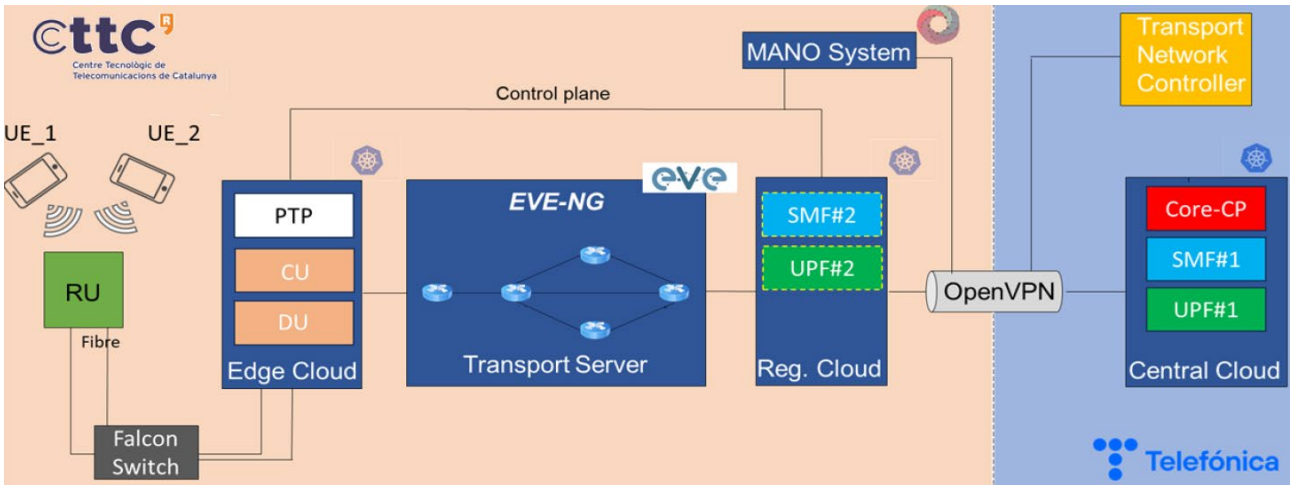


FIGURE 83. SYSTEM ARCHITECTURE

This setup can be divided into three segments: computing hardware for mobile Core and RAN software, RAN hardware, and transport network. The computing hardware consists of four commercial off-the-shelf (COTS) servers. One server executes a designed MANO system powered by an instance of ETSI Open Source MANO (OSM). This MANO system is built as a Flask-based web application designed to interact with an OSM platform instance to **deploy, monitor, and manage network services** implementing the different entities **of the mobile network (core, RAN software)** across the different available points of presence (PoPs). More specifically, this application allows the user to configure the instantiation/termination of a single of multiple network service instances implementing the mobile network entities (e.g., the control-plane part of a mobile core and the user-plane part) by selecting the its associated descriptor, the PoP and a configuration file containing the configuration required to deploy this network service in the specified PoP through a Graphical User Interface (GUI). This request is then sent to the Flask server process, which launches the appropriate backend scripts interacting with the underlying OSM instance for the instantiation and termination of the selected network services. The GUI additionally counts with a diagram of the underlying setup, which is dynamically updated whenever a new instantiation/termination operation occurs. In this way the user can track which and where the different deployed network services. This diagram is interactive, so when a user passes the mouse over the network service name, the GUI presents a detail of the configuration applied for this network instance. It is worth noting that each type of network service is represented with a different colour to ease the visualization of the mobile network entity distribution, as depicted in Figure 83. Moreover, once the network service descriptors are onboarded and the PoPs registered in OSM, the MANO system can be easily updated offline so this MANO system can only see the desired elements (e.g., network service descriptors, PoPs) from OSM.

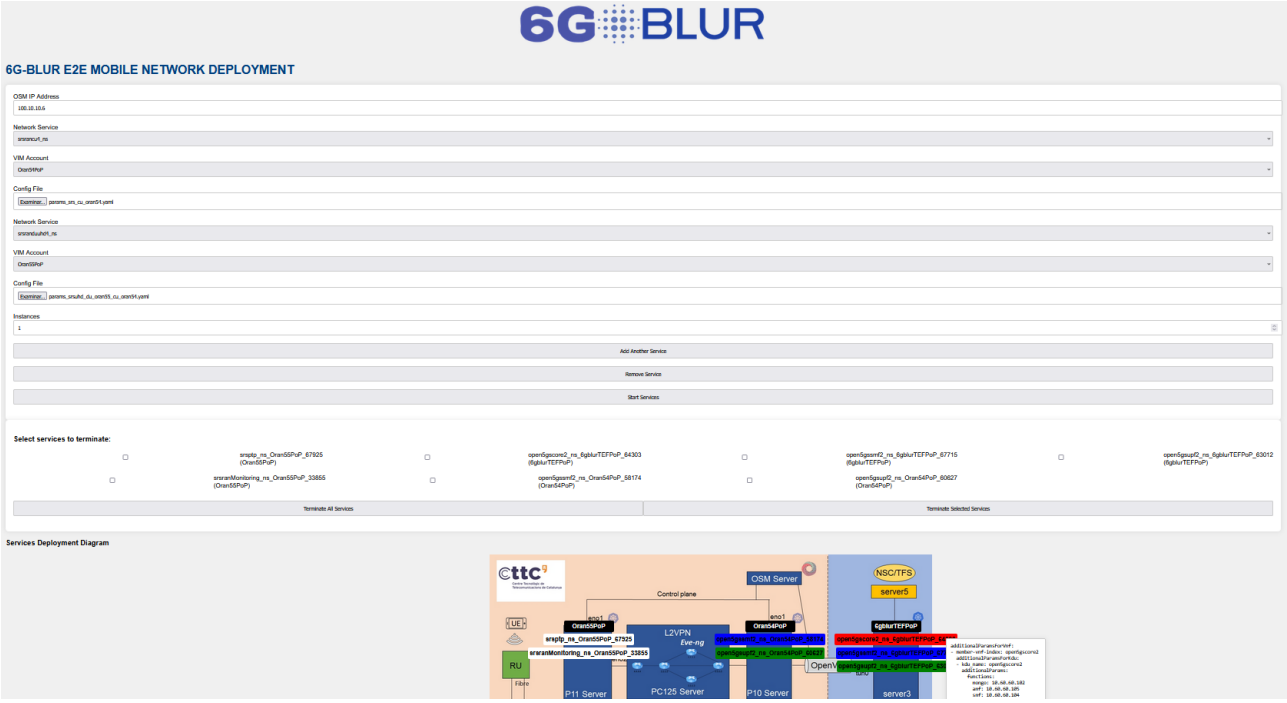


FIGURE 84. MANO SYSTEM GUI. NOTICE THE CAPABILITIES OF THE GUI TO CONFIGURE THE INSTANTIATION/TERMINATION OF MULTIPLE NETWORK SERVICES AND THE CONTEXTUAL INFORMATION PROVIDED BY THE DEPLOYMENT DIAGRAM.

The remaining three servers of the setup act as PoPs, each running a single-node Kubernetes cluster. These three clusters mimic the tiers available in an MNO deployment, i.e., central, regional and edge clouds, where the NSs associated with the mobile network entities are dynamically deployed. The RAN-related hardware consists of an O-RAN switch and Precision Time Protocol (PTP) grandmaster, an O-Radio Unit (RU), and two smartphones for over-the-air (OTA) transmissions. More specifically, we employ a Falcon RX/812/G O-RAN switch, a LITEON FlexFiFF-RFI078I4 O-RU and, a Realme 9 and a OnePlus 8 Pro smartphones, which are handled via a remote desktop tool (e.g., AnyDesk).

Regarding the transport network, it is composed of two COTS servers:

- The **first server** runs the stack of transport network SDN controllers:
  - **Network Slice Controller (NSC)**: orchestrates the request, instantiation, and lifecycle management of transport network slices.
  - **TeraFlowSDN Controller**: acts as the wide area network controller, configuring the transport routers in the setup.
  - **Ryu Controller**: manages the configuration of two OpenFlow switches located at the edges of the transport domain, enabling traffic VLAN tagging and untagging.

- The **second server**, which hosts the **EVE-NG** software, functions as an emulator for the transport network routers (see Figure 83), as well as for the two OpenFlow switches positioned at both endpoints of the server.

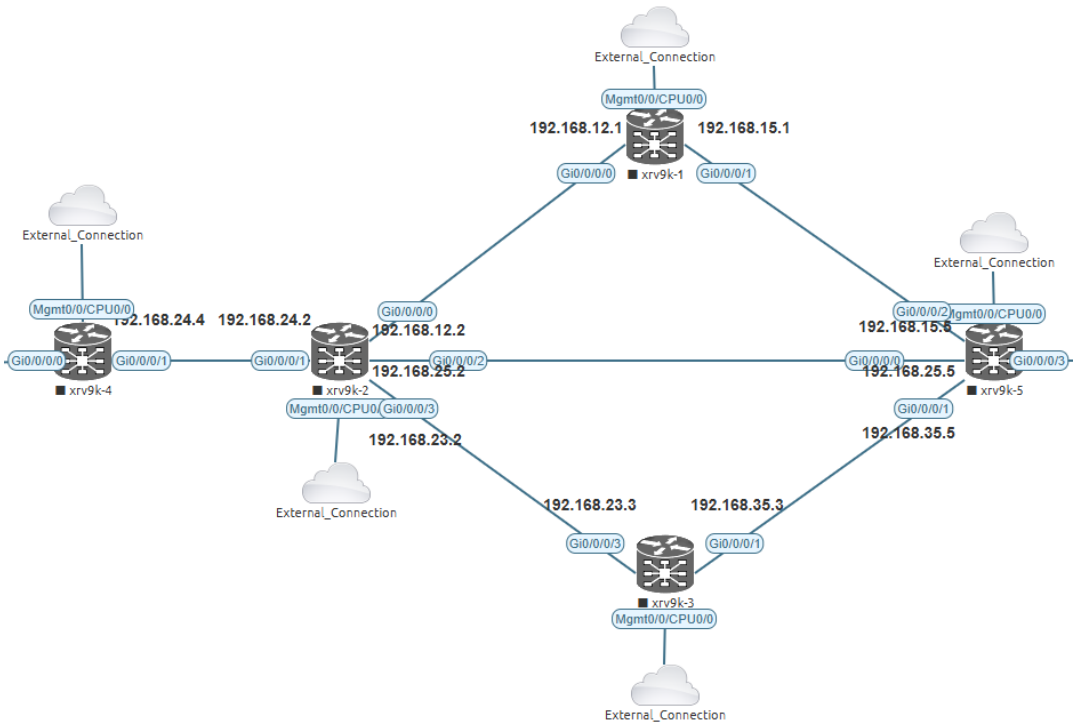


FIGURE 85. TRANSPORT NETWORK TOPOLOGY EMULATED ON EVE-NG

In this PoC, we show the capabilities of this setup to perform the automated orchestration of a fully distributed 5G cloud-native mobile network presenting multiple slices, including O-RAN entities using Split 7.2 and OTA transmission.

In the transport control part, the orchestration is conducted by our implementation of the Network Slice Controller. As described in JOINT K2.1, this component processes intents following 3GPP format and converts them to IETF slicing intents to be consumed by the TeraFlowSDN transport controller. In particular, transport slices are realized into Layer 2 VPNs, setting up a pseudo-wire associated with one of the tunnels previously established in the topology to route mobile network control and user-plane (CP and UP) traffic flows belonging to different network slices.

Regarding the entities of the mobile network, the setup is based on the blueprints (i.e., Helm Charts) described in [14]. Namely, the 5G Core part is Open5GS v2.7.1, and it is deployed through three different blueprints for higher flexibility. One blueprint deploys the CP network functions, the second one allows the deployment of an SMF instance, and the third one allows the deployment of an UPF

instance for UP function. The SMF-UPF decoupling allows a flexible activation, upon demand, of dedicated slice components. In the RAN part, we count with two blueprints, one for the Central Unit (CU) and one for the Distributed Unit (DU) using srsRAN Release 24.12.

Specifically, srsRAN is the open-source RAN solution developed by SRS (one of the 6G BLUR project partners), which is fully compliant with the 3GPP and O-RAN specifications (L1, L2 and L3 stack) and which runs on a variety of platforms and allow disaggregated deployments such as the one envisioned by the PoC. Also, given the distributed nature of the set-up, we opted for the version of the srsRAN suite that runs CU and DU as separate executables It is worth mentioning here that a number of improvements and new features have been added to srsRAN driven by the requirements of the PoC, such as the new slice-aware RAN scheduler and an improved management of the interfaces of the CU-UP, namely F1-U and N3 (see Key Concept SMART K1.1 for further details).The DU manifest is configured to connect to the LITEON FlexFi O-RU hardware through Split 7.2 open fronthaul interface and to launch OTA transmission. Beyond the mobile network entities, an additional blueprint relying on Linux ptp4l and phc2sys processes is executed to acquire the PTP synchronization provided by the Falcon Switch. The flexible templating capabilities of Helm Charts facilitate the parametrization of blueprints for the mobile core entities, CU, and DU, allowing the deployment of a mobile network with a configurable number of slices and physical RAN-allocation configuration for these slices (e.g., ratio of Physical Resource Blocks (PRBs)). As previously mentioned in the description of associated key concepts, these blueprints are further encapsulated in OSM descriptors (VNF and NS packages), so the mentioned parametrization can be exploited through OSM instantiation commands, enabling the on-demand and flexible creation of instances of these entities to fit the needs of given scenarios.

3.1.2.3 Implementation

As previously mentioned, this PoC showcases the orchestration of a fully disaggregated cloud-native open-source 5G mobile network considering Core, transport and RAN segments. The orchestrated network employs Split 7.2 between DU and O-RU and allows the on-demand activation of NFs handling the different slices and associated transport flows, thus enabling the deployment of scenarios where the distribution of the mobile network entities allow covering different requirements in terms of latency and bandwidth, such as those envisioned in [5]. As explained in the Key Concepts section, the involved mobile NFs are packaged as different NSs. Initially, these packages are onboarded into the OSM MANO stack so they can be invoked from the developed MANO system and the transport network flow intent templates (CP and UP) are defined. In order to build the complete setup depicted in Figure 83, we follow the next steps:

1. We request our MANO system to deploy the PTP artefact in the edge cloud to provide the required synchronization between DU and RU to the network interfaces of edge cloud connected to the Falcon O-RAN switch.
2. OSM receives the request from our MANO system to deploy the Open5GS CP NS in the central cloud. We configure two slices for this deployment (i.e., SST:1 and SST:2). Once deployed, the information of the multiple UEs and their slice association are included in the UDR database. Then, we deploy the additional Open5GS NSs (i.e., SMF1, UPF1) corresponding to slice #1 (i.e., SST:1) also in the central cloud. This slice is devoted to low bandwidth and non-latency sensitive communications, e.g., machine-type communications.
3. Later, we send a request to the NSC to configure the transport network paths to allow the F1 communication for slice#1 (CP interface flow, i.e., F1-c, is common for all slices but UP interface flow, i.e., F1-u1, belongs to a given slice).
4. After that, we deploy the disaggregated RAN components (i.e., CU and DU) in the edge cloud. As with core entities, these CU and DU instances are configured to serve two slices. Additionally, we specify the resources (e.g., ratio of PRBs) assigned to each of the requested slices in the DU to fit the requirements of the different slices.
5. Then, we register the UE 1 in the network. After leaving flight mode, the UE 1 receives an IP address in the range of the data network associated with slice#1, thanks to the configured CP flow in the transport network. UP communication flows through the path configured for the F1-u1 flow.
6. We request our MANO system to deploy the set of Open5GS NSs for slice#2 (SST:2) (i.e., SMF2, UPF2) in the regional cloud, devoted to high bandwidth and latency sensitive communications.
7. After this, we send a request to the NSC to configure the transport network path to allow the F1-u for slice#2, i.e., F1-u2. Once this flow is configured in the transport network, we register UE 2 in the network and its UP traffic will flow through a disjoint transport network path, experiencing different performance according to its different requirements.

Throughout this demonstration, each step is visualised using the Graphical User Interface (GUI) of our MANO system, a remote desktop session displaying smartphone activity, and live wireshark captures at key routers showing the packets flowing through each of the three configured transport paths.

3.1.2.4 Results

In this section, we present results derived from the execution of this PoC based on the steps described in previous section.

With respect to the initial Steps 1 and 2, Figure 86 presents the time required to deploy the blueprints corresponding to the PTP synchronization and the different mobile core entities artefacts implementing a slice: the SMF and UPF. This experiment has been repeated 50 times.

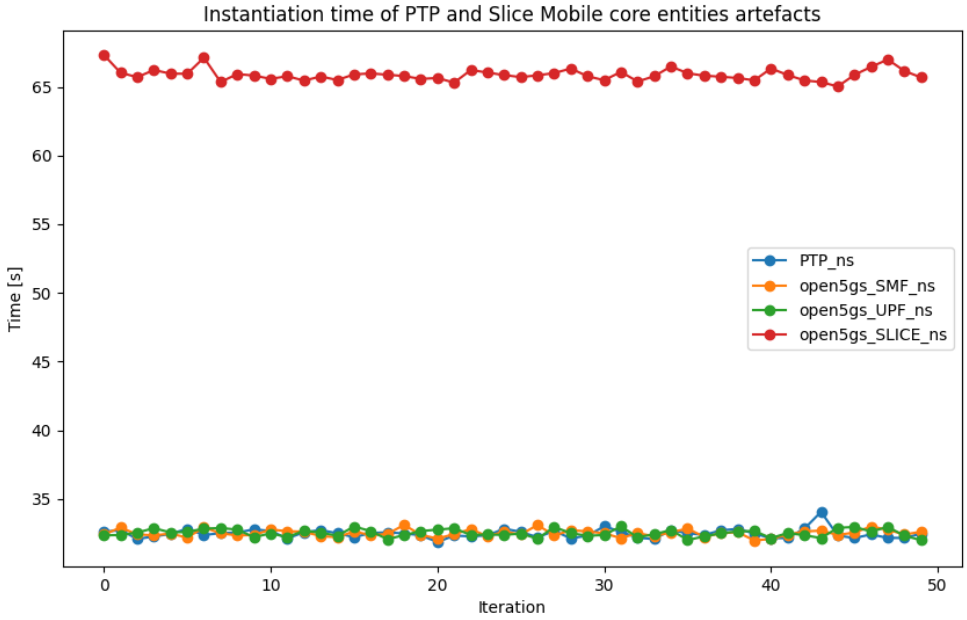


FIGURE 86. EVOLUTION OF INSTANTIATION TIME ACROSS ITERATIONS FOR THE PTP AND OPENGS SLICE ARTEFACTS

As we can see, the time required to instantiate each of the artefacts is around 32 seconds, in line with the trend observed in Section 2.1 and 2.2 , when using OSM software to perform the operation. This is the expected time that a user would experience when using the GUI provided by the MANO system previously described. This graph also shows the time required to jointly instantiate both mobile core components to process slice traffic (i.e., SMF and UPF), which is around 65 seconds, as expected, observing the regularity of the operations. Overall, the performed disaggregation in the mobile core CP artefact incurs in additional time for the whole process of deploying all the mobile core (CP and UP), however it provides further flexibility in the deployment process and more isolation capabilities in the treatment of slices, since there is a devoted SMF network function instance processing operations for the corresponding slice. We consider this impact negligible because the overall deployment time still keeps in the order of minutes, in contrast to current manual procedures that could last for hours, even days. This disaggregation has been possible due to the update in the version of Open5GS software, passing from v2.6.4, used in the experiments of KC1.1 (Section 2.1) to version v2.7.1. Since this new version adds additional features, the resulting container images passes

from 566 MB to 712 MB, but as noted in KC1.1 and KC1.2 experimentation, once a container is cached in a Kubernetes cluster, its deployment time is not altered. Moreover, although this container image needs to be downloaded again, the instantiation process still keeps in the order of minutes, as depicted in Figure 87. In particular, we repeated the previous experiment but every 5 repetitions we removed the container image corresponding to open5Gs, so it needs to be downloaded from the container repository (i.e., Docker Hub), as mentioned in KC1.2 in Section 2.2. A final about remark about Figure 87 is that when deploying the slice components (i.e., SMF and UPF), this caching issue would only affect to the deployment of SMF, because once SMF is deployed, the artefact corresponding to the UPF employs the same container image.

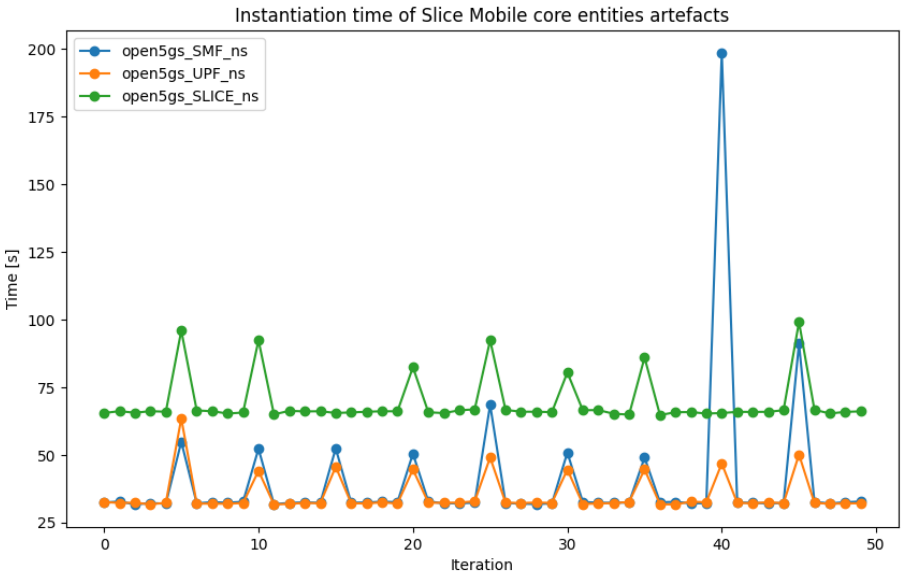


FIGURE 87. EVOLUTION OF INSTANTIATION TIME ACROSS ITERATION FOR OPEN5GS RELATED ARTEFACTS WHEN REMOVING CACHED CONTAINER IMAGES

Regarding the PTP artefact, it is worth mentioning that although the associated containers have been successfully instantiated in around 32 seconds, the Linux *ptp4l* and *phc2sys* processes running inside to acquire the PTP synchronization provided by the Falcon Switch need of about 2 minutes until they reach the synchronization state explained in [15].

In step 3, the NSC receives a 2-flow request (Annex 6.1.1 and 6.1.2) to configure two transport flows in the transport network. A *best-effort flow* for the control plane traffic, with not very demanding requirements in term of latency and throughput, and a user plane traffic flow for slice#1, devoted for

low bandwidth and non-latency sensitive communications. The NSC provides the following answer to the request after performing the required configurations


```
{
 “status”: “success”,
 “code”: 200,
 “slices”: [
  {
   “id”: “slice-service-bd6b7c15-944a-4c06-a2ac-d254436402a8”,
   “source”: “10.60.60.105”,
   “destination”: “10.60.11.3”,
   “vlan”: “100”,
   “bandwidth(Mbps)”: 1,
   “latency(ms)”: 20
  },
  {
   “id”: “slice-service-733ee340-c239-4efa-a61e-858640f43b70”,
   “source”: “10.60.60.106”,
   “destination”: “10.60.11.3”,
   “vlan”: “102”,
   “bandwidth(Mbps)”: 10,
   “latency(ms)”: 20
  }
 ],
 “setup_time”: 29.552100002765656
}
```


The trend followed for the deployment of the CU and DU entities have been described in Section 2.2 and are aligned with those previously commented for the slice components (i.e., SMF and UPF NF). Once cached the container in a given cluster, the time required by our MANO system based on OSM to deploy a mobile network entity is around 32 seconds on average. It is worth mentioning that the presented disaggregation allows scalable scenarios since a given CU could handle several DU instances, thus saving computing resources when compared to a monolithic gNB combining CU and

DU functionalities into a single element. Moreover, it is important to highlight that for proper network setup, step 3 is very important to be performed previously. Once the CU instance is up and running, it tries to establish control-plane communication to the AMF entity using N2 protocol encapsulating SCTP traffic. User-plane communication is established between CU and corresponding UPF dictated by AMF using N3 protocol encapsulating UDP traffic once a network attachment request arrives from a UE.

Regarding the transport network, and upon the configuration of the traffic templates requested in step 3, it is observed that SCTP control-plane signalling traffic is routed through the best-effort path, which is tagged with VLAN 100. The figure below presents a Wireshark capture of the interface link between Router 1 and Router 5 in the transport topology. The IPs in the capture correspond to the AMF (10.60.60.105) and the server hosting the CU (10.60.11.55)

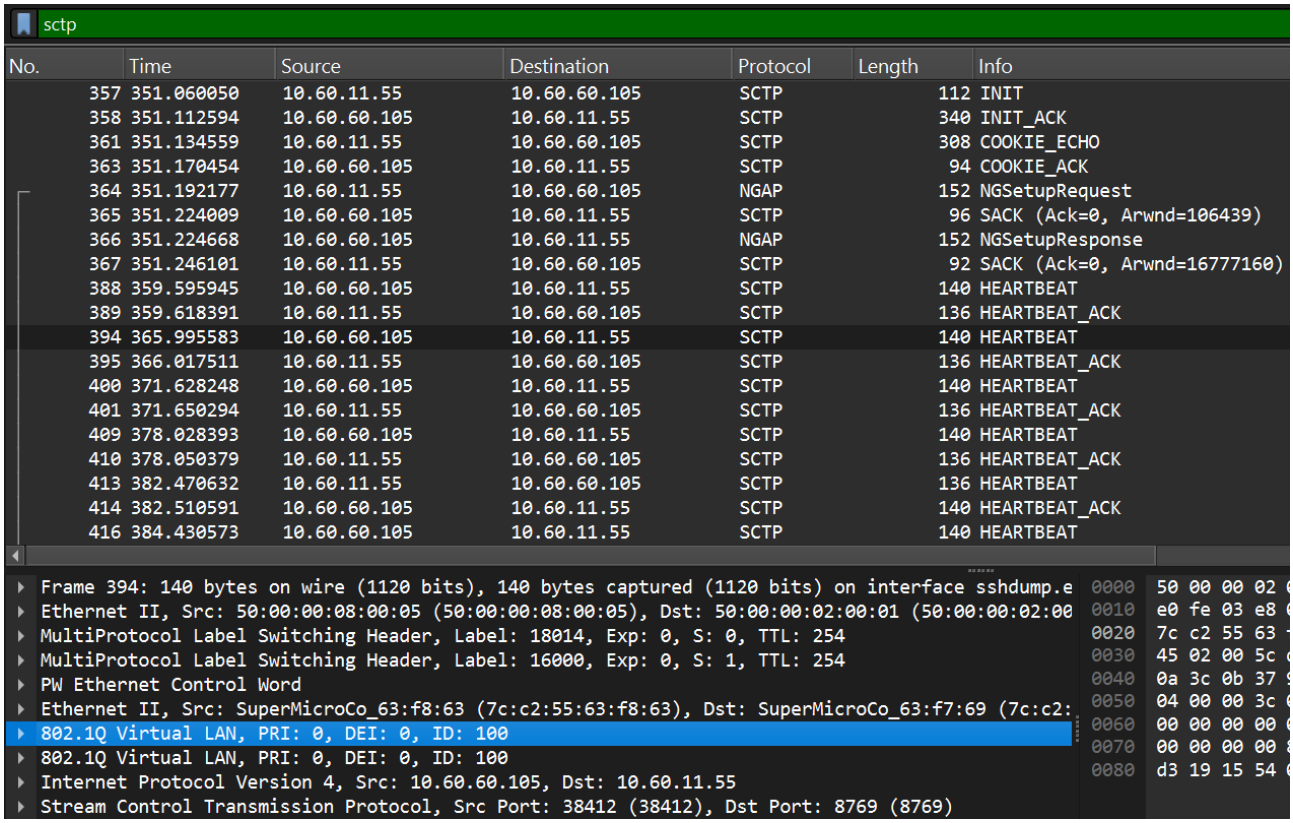


FIGURE 88. CONTROL TRAFFIC CAPTURED IN BACKHAUL

Once UE 1 is registered in the network, the transport network observes user plane traffic traversing the non-latency sensitive path, tagged with VLAN 102. This traffic consists of GTP-U packets with IP destinations corresponding to the UPF located at TID premises (10.60.60.106) and the CU located at

CTTC premises (10.60.11.3). The figure below shows a Wireshark capture of the interface link between Router 3 and Router 5 in the transport topology.

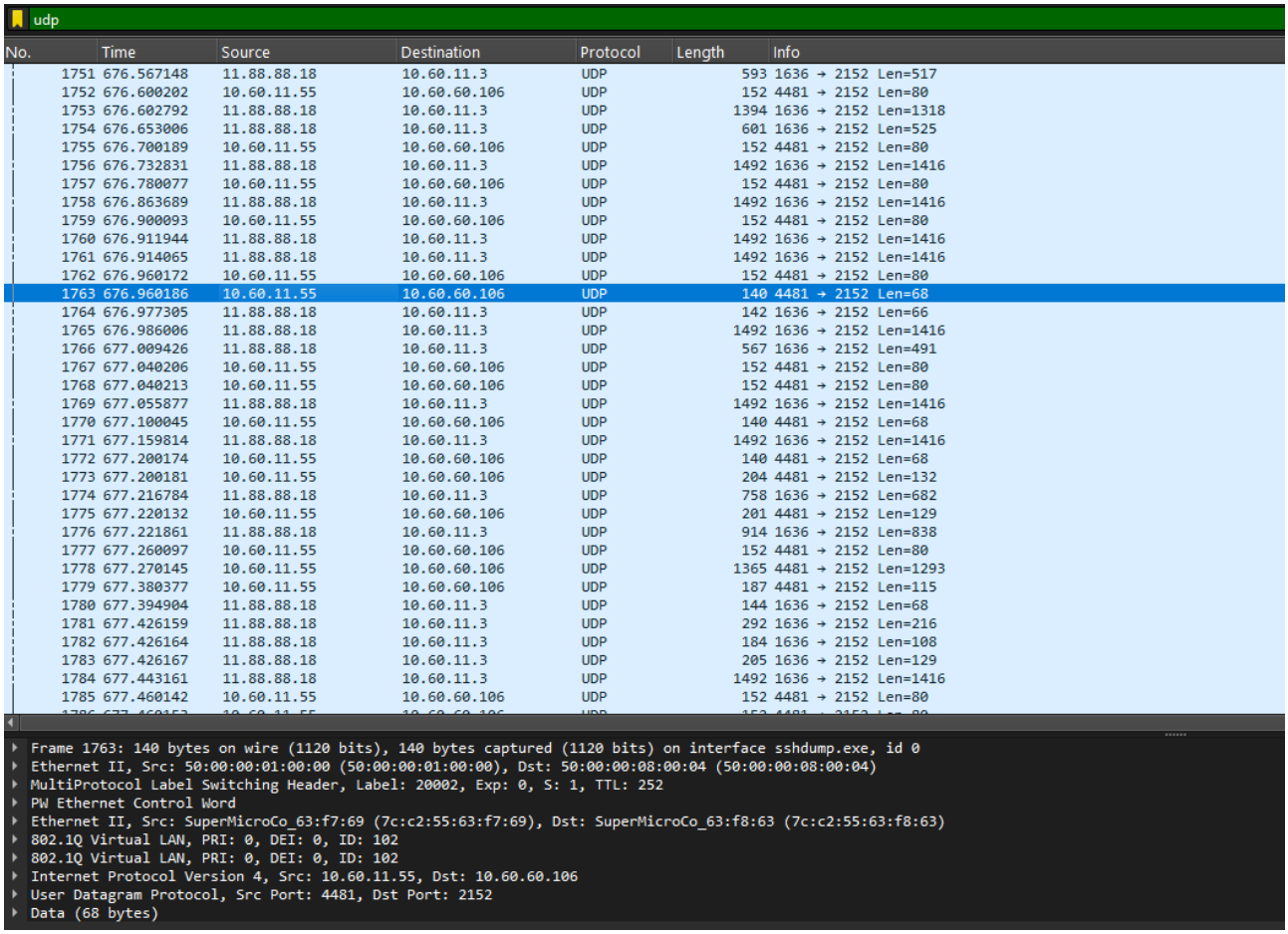


FIGURE 89. SLICE 1 USER PLANE TRAFFIC CAPTURED IN BACKHAUL

With respect to the throughput performance experienced by the slice#1 traffic, Figure 90 presents a comparison between the peak rate experienced for an over-the-air 60 seconds long TCP Iperf session in DL (i.e., between UPF and a UE attached to slice#1) when comparing the deployment of a single slice with 100% of assigned PRBs versus the system with 2 configured slices and only 10% of PRBs assigned to slice#1. As we can see the slice-aware RAN scheduler limits the traffic achievable by slice#1, reaching an average value of 5.5 Mbps. The average peak value for the system configured with a single slice with 100% PRBs is 84Mbps on average (for this experiment, we configured accordingly the transport network flow). In the experiments, the LiteON FlexFI RU is configured to use 20 MHz bandwidth transmission with the default frame configuration DDDDDDSUUU.

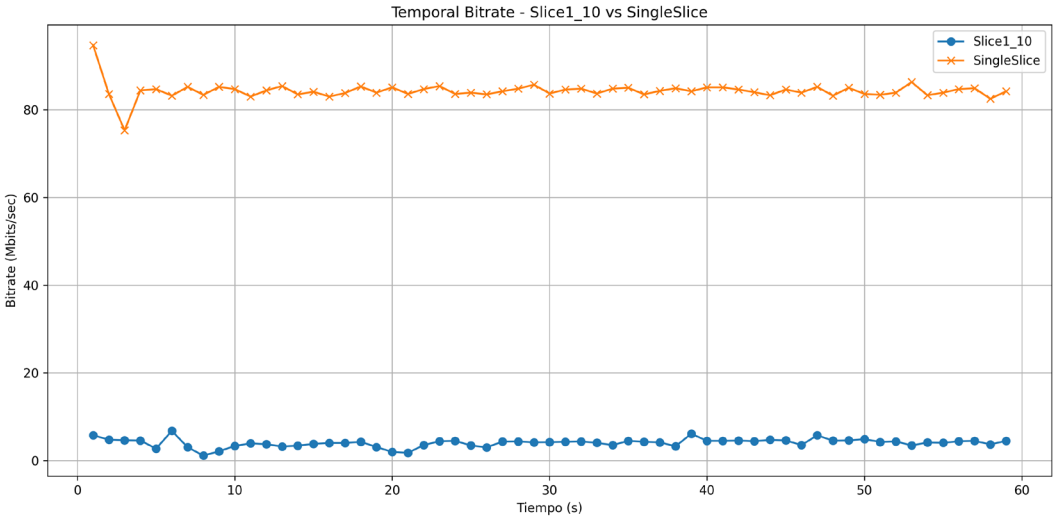


FIGURE 90. COMPARISON OF BIT RATE FOR DIFFERENT PRB RESOURCE ALLOCATION

When required user-plane traffic in slice#2 for high bandwidth and latency-sensitive communications, we instantiate the required mobile core entities (i.e., SMF and UPF) as mentioned in Step2.

In step 7, the NSC receives a flow request (Annex 6.1.3) to configure a user plane traffic for slice#2, intended to high throughput and latency sensitive traffic. The NSC provides the following answer to the request after performing the required configurations.


```
{
 "status": "success",
 "code": 200,
 "slices": [
  {
   "id": "slice-service-ec693d8f-55f5-44b2-a8f8-4755fcfbf39d",
   "source": "10.60.10.6",
   "destination": "10.60.11.3",
   "vlan": "101",
   "bandwidth(Mbps)": 100,
   "latency(ms)": 5
  }
 ],
```

```
  "setup_time": 16.552100002765656
}
```


Once UE 2 is registered in the network, the transport network observes user plane traffic traversing the non-latency-sensitive path, tagged with VLAN 101. This traffic consists of GTP-U packets with IP destinations corresponding to both the UPF (10.60.10.6) and the CU (10.60.11.3), located at the CTTC premises. The figure below presents a Wireshark capture of the interface link between Router 2 and Router 5 in the transport topology. Figure 92 shows that each UEs is associated to a different slice and the associated UPF offers a different internet output. As a consequence, the experience delay by each UE to reach the corresponding UPF, placed one in Madrid (i.e., slice#1) and the other in Castelldefels (i.e., slice#2) is different, as depicted in Figure 93.

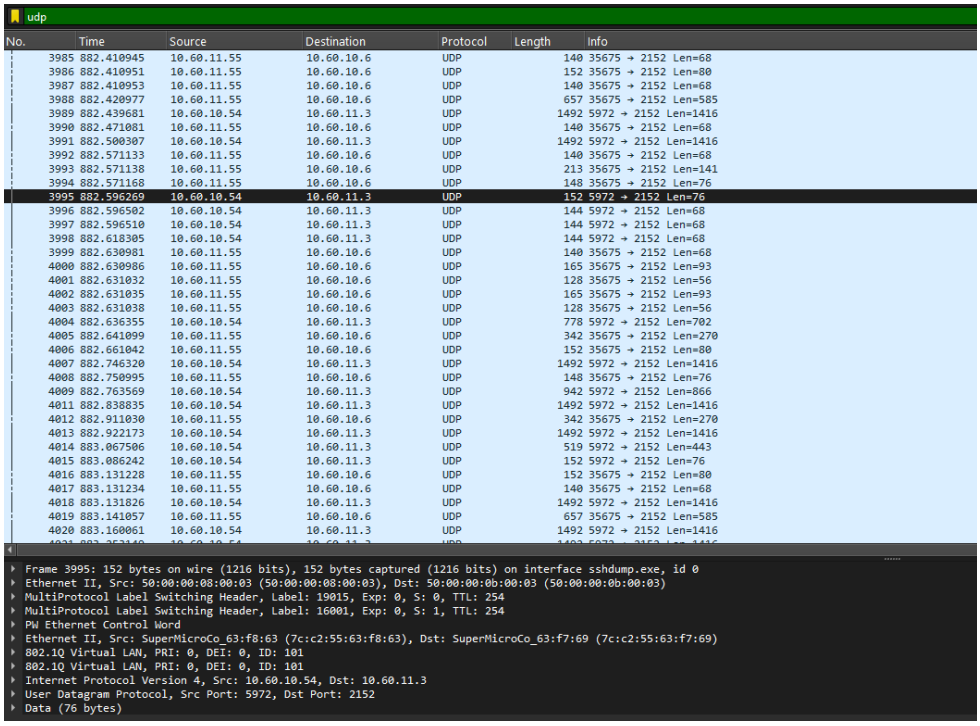


FIGURE 91. SLICE 2 USER PLANE TRAFFIC CAPTURED IN BACKHAUL

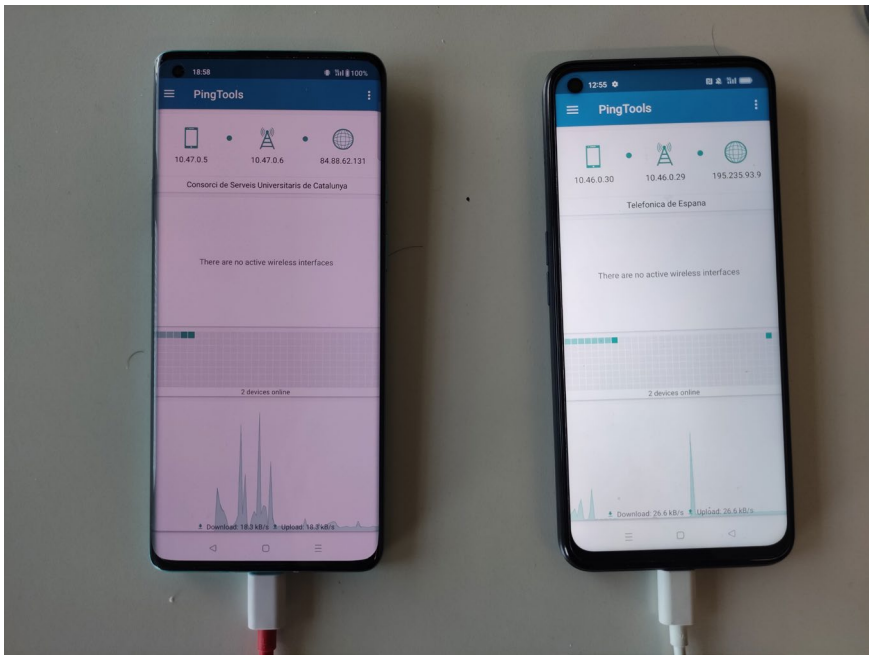


FIGURE 92. UPFS IN DIFFERENT SLICES OFFER DIFFERENT INTERNET OUTPUT TO THE ATTACHED UES (LEFT: SLICE#2 USER, RIGHT: SLICE#1 USER)

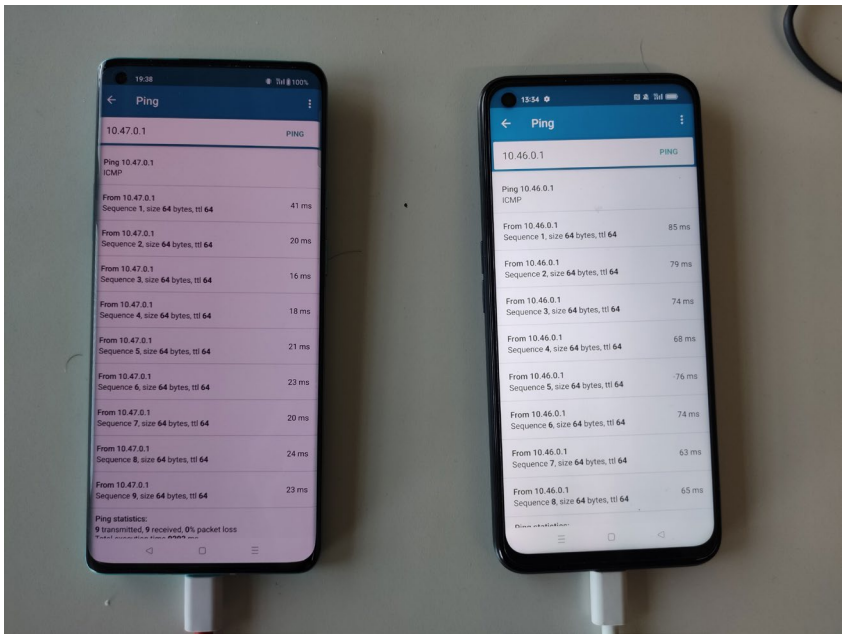


FIGURE 93. UES ASSOCIATED TO DIFFERENT SLICES EXPERIENCE DIFFERENT LATENCY WITH ITS SERVING UPF (LEFT: SLICE#2 USER, RIGHT: SLICE#1 USER)

Regarding the PRB allocation for slice#2 users, it was already established in the DU configuration in Step 4 when launching the corresponding instance. Slice#2 is configured with the complementary PRBs until using the 100% of PRBs. Figure 94 presents the temporal evolution of a 60s DL TCP Iperf session when providing different PRB resource allocation to each slice (transport network required of proper request configuration to perform this comparison). The developed RAN-aware scheduler at the DU, as described in SMART KC1.1, is able to provide the requested PRBs to each slice. In this allocation process there are fluctuations due to varying over-the-air conditions during the experimentation. This variability is represented in Figure 95, which depicts the CDF representation of the experienced throughput between slice#1 and slice#2 when performing five 60 seconds long TCP Iperf session in DL between the UPF assigned to a slice and its corresponding user.

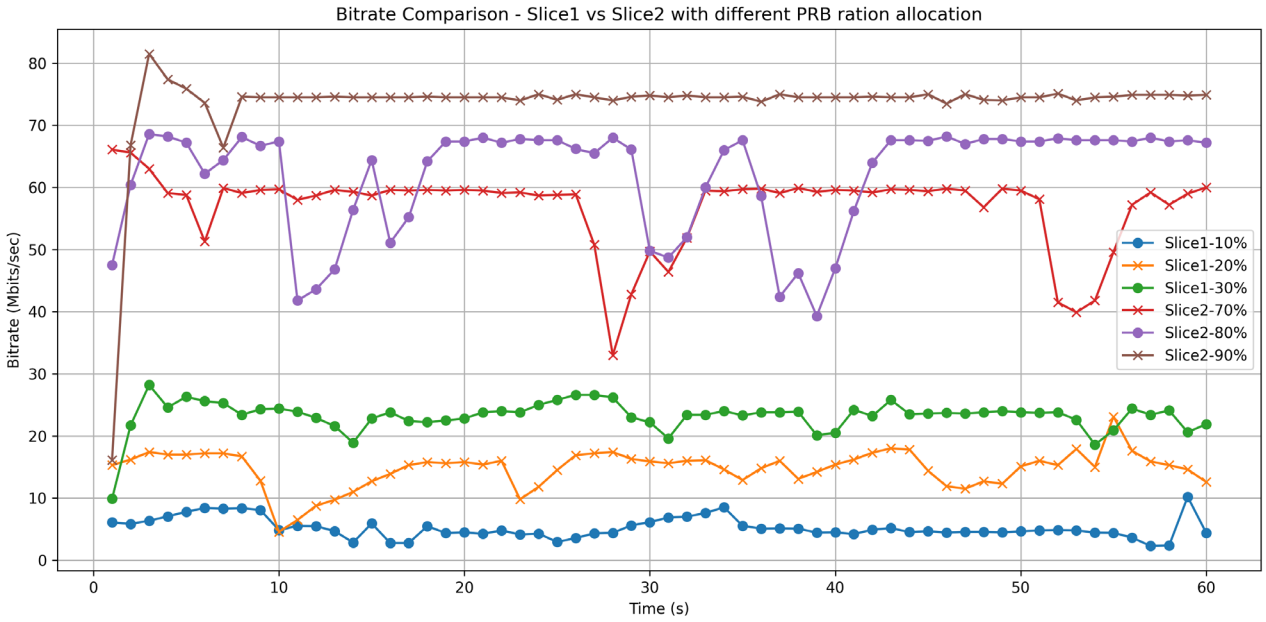


FIGURE 94. COMPARISON OF BIT RATE FOR CONFIGURED SLICE#1 AND SLICE#2 PRB RESOURCE ALLOCATION

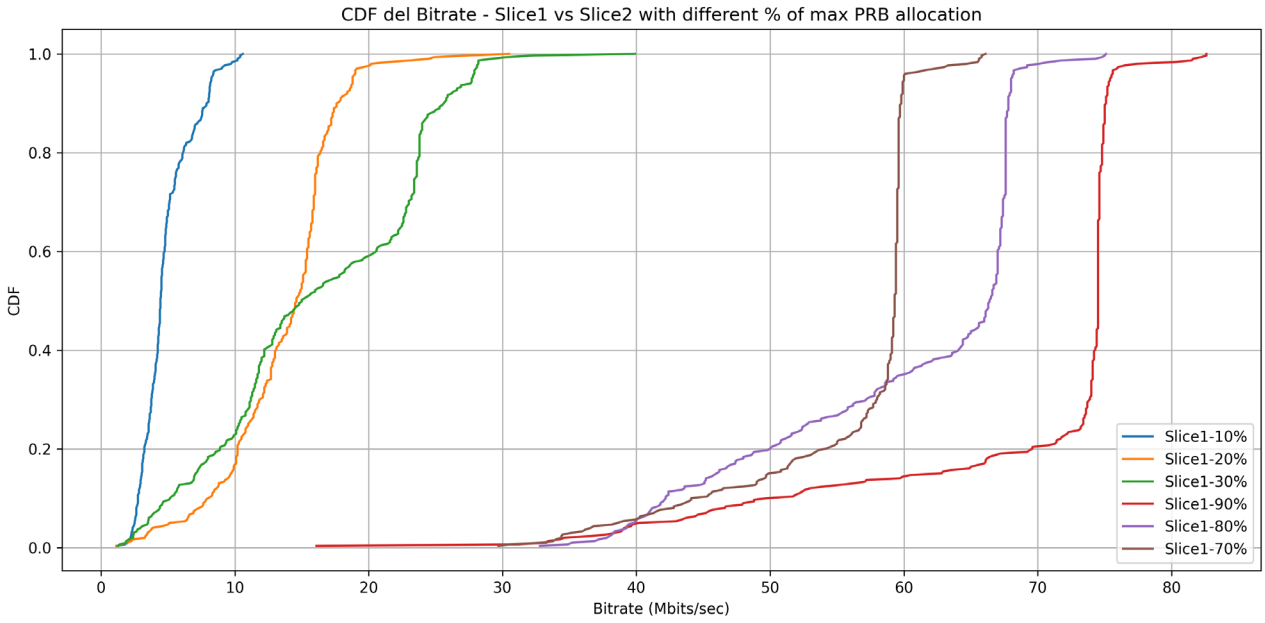


FIGURE 95. CDF OF THE EXPERIENCED THROUGHPUT AT SLICE#1 AND SLICE#2 WHEN MODIFYING PRB SPLIT

To finish the analysis, Figure 96 presents the CDF comparison between the experienced throughput in five 60 seconds long TCP Iperf sessions for slice#1 and slice#2 s when running CU and DU, in charge of slice-aware traffic scheduling, using the cloud-native artefacts developed in JOINT KC1.2 or as a bare-metal process. As observed the cloud-native artefact can offer a similar performance as the bare-metal process, but it provides more automated, flexible, and portability capabilities at the time to perform mobile network deployments.

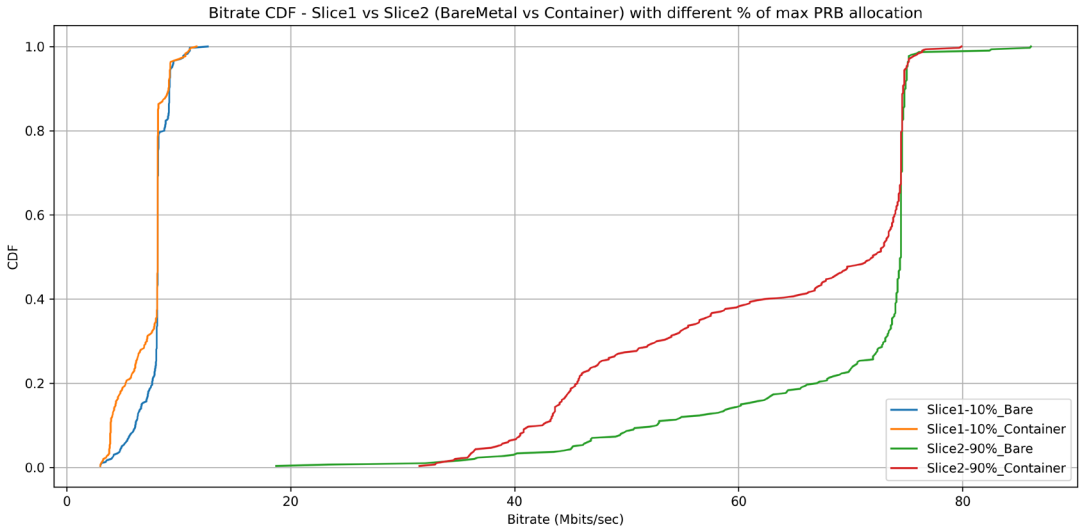


FIGURE 96. COMPARISON BETWEEN SYSTEM PERFORMANCE WHEN RUNNING AS A BARE METAL PROCESS OR AS A CLOUD-NATIVE PROCESS

3.1.2.5 Conclusions

This PoC demonstrates the viability and advantages of deploying a fully cloud-native, open-source, and disaggregated 5G mobile network based on an O-RAN architecture. The PoC highlights the seamless integration of RAN, Core, and transport segments in a distributed setup, showcasing the orchestration capabilities required for a dynamic and flexible mobile network infrastructure.

Through the integration of multiple Key concepts developed within 6G BLUR project, the PoC succeeded in integrating open-source solutions such as srsRAN and Open5GS, leveraging Kubernetes-based deployments using Helm charts embedded into OSM descriptors, so OSM can be leveraged for handling and automate the end-to-end lifecycle management of the E2E mobile network. The open fronthaul library developed by SRS using Split 7.2 showed compatibility with commercial O-RUs, thus allowing over-the-air (OTA) transmissions.

A major milestone achieved in this PoC was the dynamic provisioning of differentiated network slices with specific requirements in terms of latency and bandwidth. The Network Slice Controller (NSC), developed by Telefónica, enabled the automated configuration of the transport network using standard-compliant intents, effectively routing both control and user plane traffic through isolated and optimized Layer 2 VPNs. This slicing mechanism was validated with real UEs and showed the ability to route traffic through distinct paths based on slice requirements.

Moreover, the PoC confirms that disaggregated RAN components can be instantiated flexibly and tailored to per-slice needs via customizable Helm blueprints included in OSM descriptors. The use of

an emulated transport network in EVE-NG enabled extensive testing of path configurations and VLAN-based separation of flows, providing a realistic validation of the orchestration stack under test.

In conclusion, the PoC demonstrates the maturity of open-source and cloud-native solutions for realizing the future of flexible and distributed 5G networks. It also paves the way for future enhancements, such as the integration of AI-driven orchestration, support for additional RAN splits, and further scalability tests in real-world scenarios.

3.2 UC2: Joint RAN and Transport mechanisms for 6G Disaggregated Mobile networks

3.2.1 PoC1: Joint Fronthaul Capacity Control/Placement, QoS management and Flexible Functional Splits

3.2.1.1 Introduction

This proof of concept explores and evaluates the various possibilities that emerge when implementing different distributed approaches within a mobile network infrastructure, specifically incorporating O-RAN components in the RAN domain. Given that centralization is one of the most prominent and beneficial scenarios in Open RAN, offering significant advantages to Mobile Network Operators (MNOs), a key objective of this PoC is to assess the viability of achieving centralization beyond the conventional limits and to explore deployment flexibility. Traditionally, such centralization has been constrained to distances of approximately 10–20 km, relying on dedicated transport solutions to meet strict round-trip time (RTT) requirements in the fronthaul. Instead, this PoC investigates deeper levels of centralization by pushing past these boundaries, aiming to support centralization over distances exceeding 20 km (equivalent to RTTs over 250 µs), by relocating the O-DU away from the cell site.

This works builds upon several key developments:

- **SMART-K1.1** – CU and DU implementation: implementation of O-RAN Central Unit (CU) and Distributed Unit (DU) components using the srsRAN software.
- **SMART-K1.2** – RAN orchestration: enhancement of the FH traffic conditions by integrating various FH control methods.
- **SMART-K2.1** – Development and characterization of FH and MH: extension and improvement of the O-FH library within the srsRAN Project CU/DU solution.
- **SMART-K2.2** – Characterization of the FH and MH transport: study specially on the FH to understand its behaviour under centralized topologies.
- **SMART-K2.3** – Flexible functional split: calculation of the fronthaul requirements that would be needed to support a 6G network and implementation of various functional splits.
- **SMART-K2.4** – Radio stack optimization: adaptations in DU SW to allow for longer FHs.
- **SMART-K2.5** – QoS Management for time-critical immersive applications: design and implementation of generalized QoS MAC schedulers and Lyapunov-based schedulers.
- **SMART-K2.6** - QoS strategies at the transport for multi-service aggregation.

- **JOINT-K2.1** – Transport network optimization: optimization of the transport network within an ORAN-based mobile network.
- **JOINT-K2.2** – Entities' placement decisions: algorithms development commonly known as Virtual Network Embedding (VNE).
- **JOINT-K2.3** – QoS policies and scheduling: implementation of HQoS-aware (Hierarchical Quality of Service) policies.

3.2.1.2 System Design Considerations

As previously seen, UC2 PoC1 combines a great number of key concepts. The following subsections provide details about the interactions among different key concepts by grouping the main contributions of this PoC into different topics.

3.2.1.2.1 FH characterization, Network characterization and O-DU optimization

3.2.1.2.1.1 FH characterization

First, in order to characterize different FH profiles within centralized topologies, we developed a custom software using MATLAB[7]. This tool was designed to, based on a set of input parameters related to the network, the O-RU and the O-DU, determine the most suitable centralization scenario.

By processing these inputs, the software is capable of analyzing multiple configuration possibilities and selecting the optimal deployment strategy. As output, it provides the specific configuration settings required for the O-DU to implement the chosen centralization scheme effectively.

The tool not only automates the evaluation process but also facilitates informed decision-making by offering a clear mapping between the network conditions and the necessary adjustments on the O-DU side. This allows for a systematic and efficient characterization of FH profiles, ensuring that the proposed configuration maximizes performance while aligning with network requirements and constraints. Detailed information about this development can be found in Key Concept K2.2 of the 6GSMART E4 deliverable.

3.2.1.2.1.2 Network characterization

For this work, we have considered Telefónica IP/Fusion network, whose initial aggregation levels are defined as HL5, HL4, and HL3. Typically, the O-DU is deployed directly at the cell site, corresponding to the HL5 level. However, in this study, we have carried out an in-depth analysis of the distribution of routers within the network, as well as their geographical distances and aggregation ratios, in order

[7] Available online at: https://gitlab.cttc.es/mdi-6gblur/smart/-/tree/main/KeyConcepts/SMART-K2.2%20Characterization%20of%20the%20FH%20and%20MH%20transport

to identify the temporal and capacity-related parameters necessary to enable the relocation of the O-DU to higher aggregation levels—specifically, to HL4 (an intermediate scenario) or HL3 (the ideal scenario).

This analysis involved mapping the network topology to understand how traffic is aggregated and propagated through these hierarchical layers. By assessing both the physical distances between nodes and the logical capacity constraints at each level, we were able to estimate critical metrics such as transport latency, bandwidth availability, and the scalability impact associated with centralizing the O-DU functions further upstream.

The ultimate objective of this evaluation is to determine the feasibility and performance trade-offs of shifting the O-DU from the edge (HL5) towards more centralized locations (HL4 or HL3), thereby optimizing resource utilization, simplifying network management, and potentially reducing operational costs, all while ensuring compliance with the stringent latency and capacity requirements inherent to fronthaul networks.

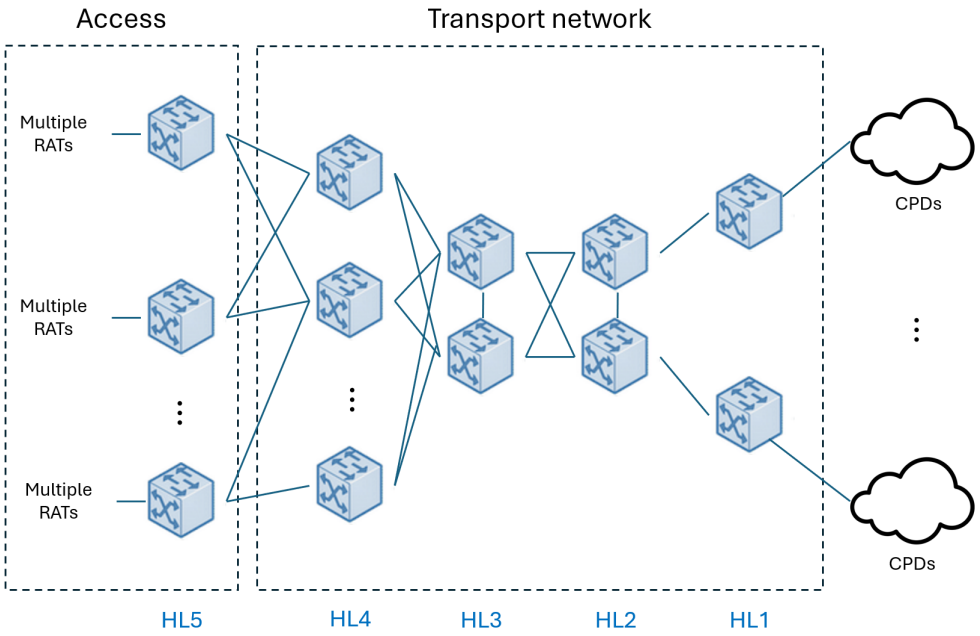


FIGURE 97. TELEFÓNICA´S TRANSPORT NETWORK

In addition, we have identified and dimensioned the necessary link capacities required to handle the estimated traffic in both DL and UL that would arise under this centralization scenario. This involved a detailed assessment of the expected traffic load associated with relocating the O-DU to higher aggregation levels and quantifying the bandwidth demands that such a shift would impose on the underlying transport network.

As part of this process, we analysed a determined cell-site configuration to estimate peak data rates in both DL and UL directions. The reference capacity is based on a tri-sectorial site configuration where one sector operates at peak capacity while the other two sectors run at average capacity, applying C-band (MIMO: 32T32R; BW: 100 MHz) and B1/B3 bands (MIMO: 4T4R; BW: 20 MHz), and leveraging pooling gains from centralization. Combining the three sectors and using BFP9 compression method, each site will gather, in the worst case, a traffic of ≈70 and ≈28 Gbps, in DL and UL, respectively. Based on these projections, we determined the minimum link capacities that would be needed to ensure reliable transport of fronthaul data without congestion or degradation of service quality, with an aggregated capacity of 700 Gbps that was required for the HL5–HL4 links, and 1700 Gbps for the HL4–HL3 and HL3–HL3 links.

Furthermore, we carefully evaluated empirically the timing behaviour of the routers involved under these conditions. This included calculating the end-to-end latency introduced by the additional transport distance, as well as examining processing delays, queuing effects, and jitter at each network node. More information is explained in the 6GSMART E4 deliverable SMART K2.6.

For that purpose, we defined the networks routers including a switching matrix with a maximum capacity of approximately 300 Gbps for HL5, 2 Tbps for HL4 and 4 Tbps for HL3, being commercial numbers available in the network equipment and greater than the worst-case traffic to ensure enough capacity.

In this sense, a testbed was set up to characterize the delay introduced by HLx routers. Specifically, an HL4 setup was implemented using a commercial router. By injecting traffic at capacities of 630, 700, and 701 Gbps, combined with packet sizes of 128, 512, and 1024 bytes, it was observed that average delays remained consistently around 10 µs, with jitter staying below 0.5 µs across all scenarios. These findings suggest that similar performance can be expected at other HLx levels, since increases in aggregate capacity are typically matched by corresponding scaling in the switching matrix.

Furthermore, considering that O-FH traffic will likely share transport with other services (e.g., fixed broadband) over a multi-service network, it is essential to enforce strict prioritization of O-FH packets. Due to their stringent timing constraints, giving O-FH traffic the highest priority ensures that it remains isolated from congestion effects, maintaining stable delay and minimal jitter regardless of surrounding traffic conditions.

These timing considerations are critical in fronthaul scenarios, where strict latency and synchronization requirements must be met to ensure proper coordination between the O-DU and O-RU.

3.2.1.2.1.3 Network impairments simulations

In order to realistically replicate the transport conditions and analyse the delay experienced from the O-DU to the O-RU over a FH deployed within the transport network, we carried out simulations using the ns-3 event-driven simulator. The impairments modelled in ns-3 can be summarized as follows: (i) baseline transport delay, which includes propagation, transmission, and switching times; and (ii) maximum delay, which results from delay variation caused by network load and traffic dynamics. The various capabilities developed in this work are summarized in JOINT K2.3.

To enhance realism, we integrated packet processing delays introduced by each network node, based on measurements obtained from Telefónica's testbed, into the simulations. Additionally, the network topology was configured to reflect the architecture of Spain's IP/Fusion network.

Network-induced variability was evaluated by analysing the one-way delay, focusing solely on the downlink direction. Each site aggregates traffic due to a tri-sector configuration under a defined load level, as previously described. In this context, the impact of network impairments, together with propagation delay, is assessed in a realistic scenario involving fragmentation, with an MTU set to 1500 bytes.

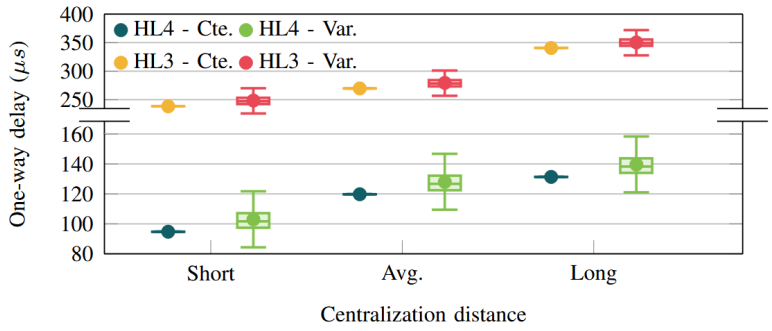


FIGURE 98. ONE-WAY O-FH DELAY PERFORMANCE FOR O-DU AT HL3 AND HL4

Router-induced delays were considered using two approaches. In the first, each node is assigned a constant processing delay, matching the average measured value of 10 µs. This case is shown in Figure 98 as HL3-Cte and HL4-Cte. Each boxplot illustrates the variability of the open fronthaul delay, accounting for both propagation time and router processing delay, with queuing delays being negligible (on the order of nanoseconds).

In the second approach, delays at each node are modeled as a shifted exponential distribution, with minimum, average, and maximum delays of 5 µs, 10 µs, and 35 µs, respectively, based on

measurements from the HL4 testbed. As shown in Figure 98 (HL3-Var and HL4-Var), the accumulation of variability across multiple nodes results in a dispersion of approximately ±20 µs (represented by the lower and upper whiskers) around the mean. As can be seen, the variability introduced by the network interfaces is minimal. However, the processing tasks performed by intermediate routers can significantly contribute to the overall delay variability.

3.2.1.2.1.4 FH characterization analytical results

To ensure accurate calculations of the temporal relationships, the developed tool is based on the CUS-planes specification defined by the O-RAN Alliance. This specification outlines the precise timing relationships that must be maintained between the O-RU and the O-DU, including the corresponding time windows, delay management use cases, and other critical synchronization parameters.

By adhering to the guidelines established in the CUS-planes specification, the tool is able to rigorously model the FH interface and correctly account for the stringent latency and jitter requirements inherent to this link. The specification provides detailed descriptions of the allowable delay budgets, synchronization tolerances, and transport layer behaviours necessary to ensure seamless interaction between the O-RU and O-DU under different deployment scenarios. For example, in Figure 99 timing relations per symbol IQ in DL direction for control planes and user planes are shown.

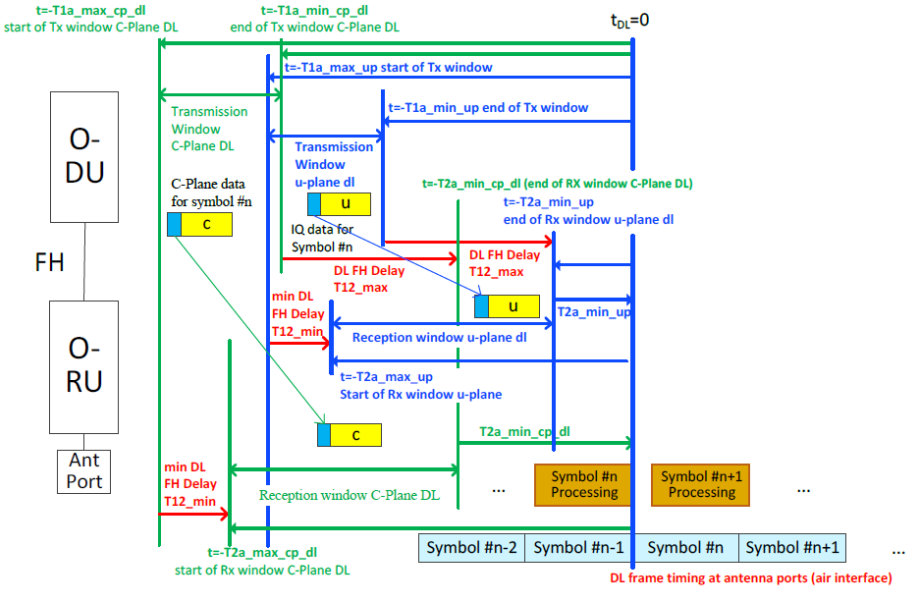


FIGURE 99. DIAGRAM OF TIMING RELATIONS IN FH IN DL DIRECTION

Incorporating these standardized constraints into the tool ensures that all calculated network configurations are not only theoretically optimal but also compliant with industry-accepted performance criteria. This guarantees that any proposed relocation of the O-DU or modification of the network architecture will continue to meet both system requirements and operational demands imposed by O-RAN-based disaggregated radio access networks.

Given the specific centralization conditions of our scenarios, in our case it is possible to extend the degree of centralization simply by relaxing the transport requirements, that is, by loosening the constraints imposed on the network routers. By allowing for slightly less stringent transport conditions, such as increased latency budgets or reduced synchronization precision along the fronthaul links, the O-DU can be relocated to higher aggregation levels more easily, thereby broadening the centralization possibilities.

Visually, the performed changes in the DL can be seen in Figure 100. It can be observed that we have successfully extended T12_max by Δt time units in order to align it with the optimal transmission window (dashed line) of the O-DU. By doing so, we have effectively increased the maximum allowable fronthaul (FH) delay, thereby providing greater flexibility in the transport network design. By adjusting T12_max to match the O-DU's optimal transmission window, we ensure that data transmission remains synchronized and compliant, even as we push the limits of centralization by relocating the O-DU further upstream.

Practically, this means that the fronthaul network can accommodate longer physical distances while still maintaining the performance requirements. The adjustment creates a more forgiving delay budget that can absorb variabilities in transport, such as processing delay and queuing effects, which would otherwise constrain centralization options.

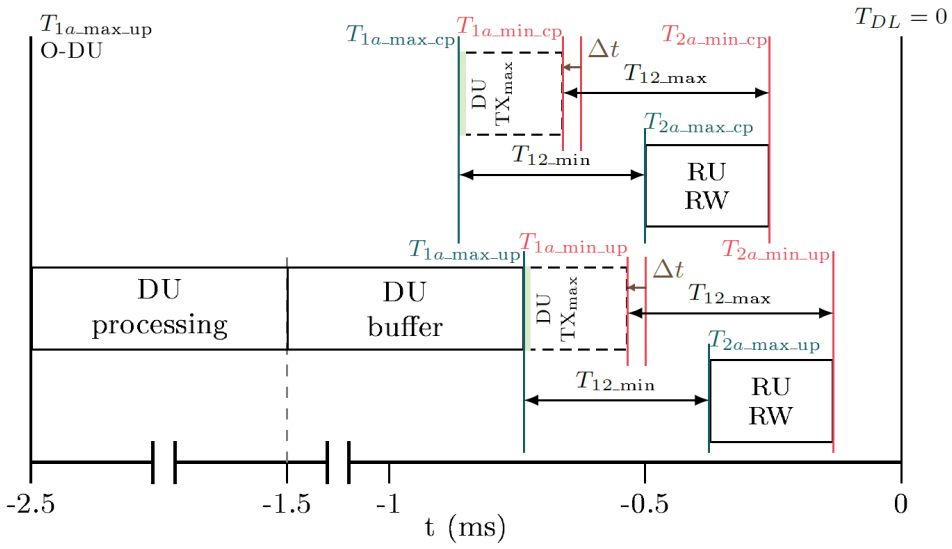


FIGURE 100. FH TIMING SCENARIO IN DL AFTER CHANGES APPLIED

Consequently, when the FH times are modified, these changes also have an impact on the UL, whose diagram with the changes produced is shown in the Figure 101. This implies that the control plane changes in the same way as in DL, and in the user plane the DU RW extends in time Δt time units.

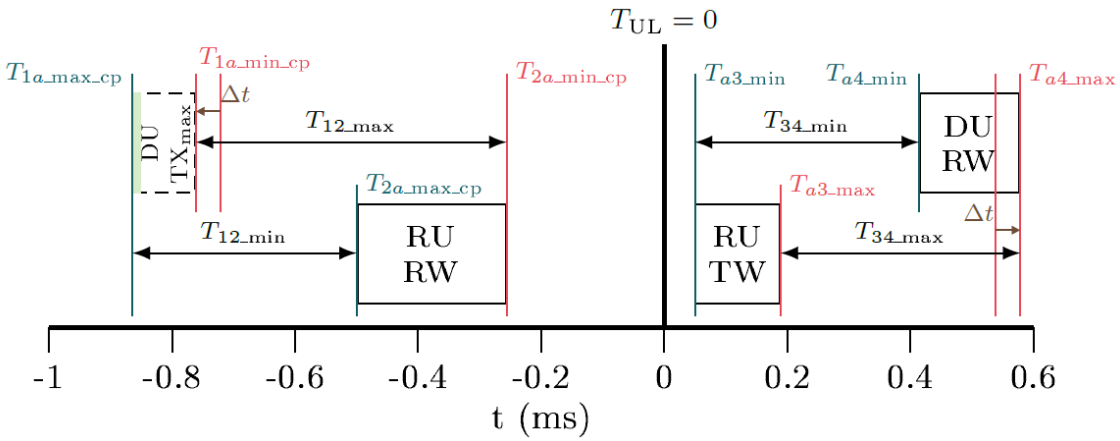


FIGURE 101. FH TIMING SCENARIO IN UL AFTER CHANGES

The implemented tool is capable of generating multiple types of results, both numerical and visual, which facilitate the interpretation and analysis of the evaluated scenario.

On the numerical side, the tool provides precise quantitative outputs such as calculated latency values, delay budgets, and other relevant metrics that characterize the performance and feasibility of the proposed FH configuration.

In parallel, the tool also produces a variety of visual outputs, including graphical representations of network topologies and timing diagrams with the initial and final scenarios. These visualizations are invaluable in offering an intuitive understanding of complex relationships between network parameters and in highlighting the effects of different centralization strategies on overall system performance.

By combining both numerical precision and graphical clarity, the tool enables us to comprehensively interpret the implications of their design decisions. This dual-output approach ensures that technical teams can both validate detailed performance metrics and communicate key insights effectively to broader audiences, supporting informed decision-making throughout the planning and optimization process.

Based on the exportation of results, and by running the program multiple times to analyse the variability of the outcomes according to the network conditions, it becomes possible to thoroughly examine the obtained scenarios and their distribution. For example, Figure 102 shows the FH timings for the O-DU located at HL4 and for the different degrees of centralization. It can be seen how *T12_min* remains constant, while *T12_max* increases to match the optimal value of the O-DU TW.

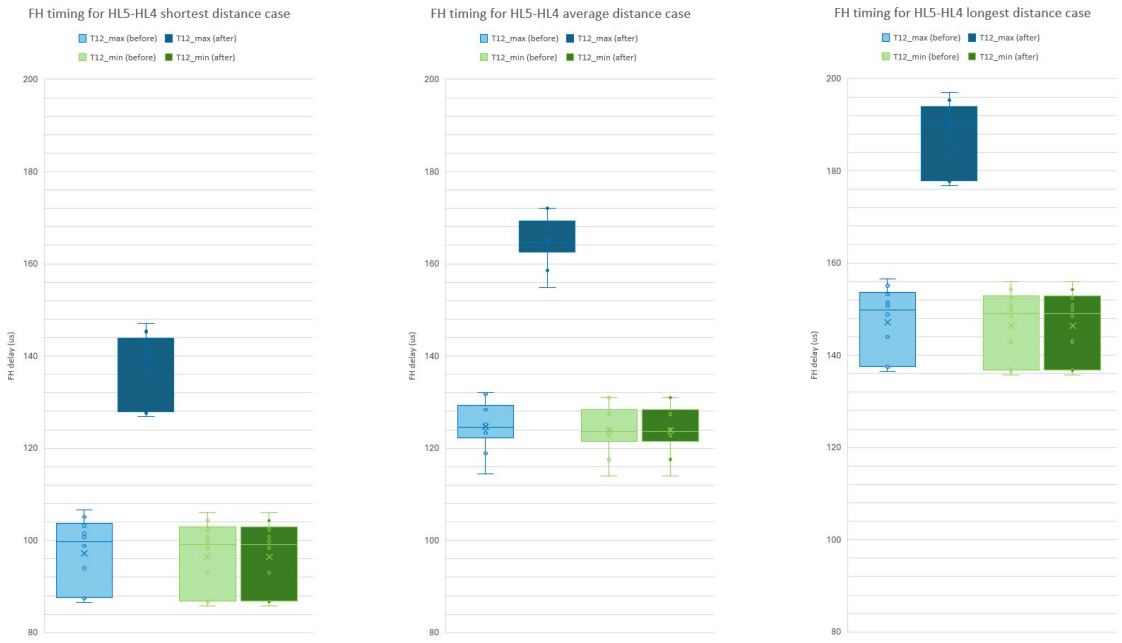


FIGURE 102. FH DELAYS FOR O-DU AT HL4

Likewise, Figure 103 shows the centralization distance achieved in each simulation, going from approximately 10 km to 25 km.

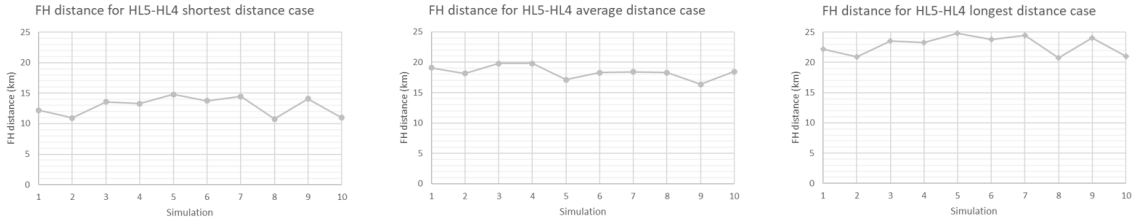


FIGURE 103. FH DISTANCES FOR O-DU AT HL4

Now, for the case in which the O-DU is further centralized and set at HL3, we have the following FH delays plotted in Figure 104, of course longer than in the previous case.

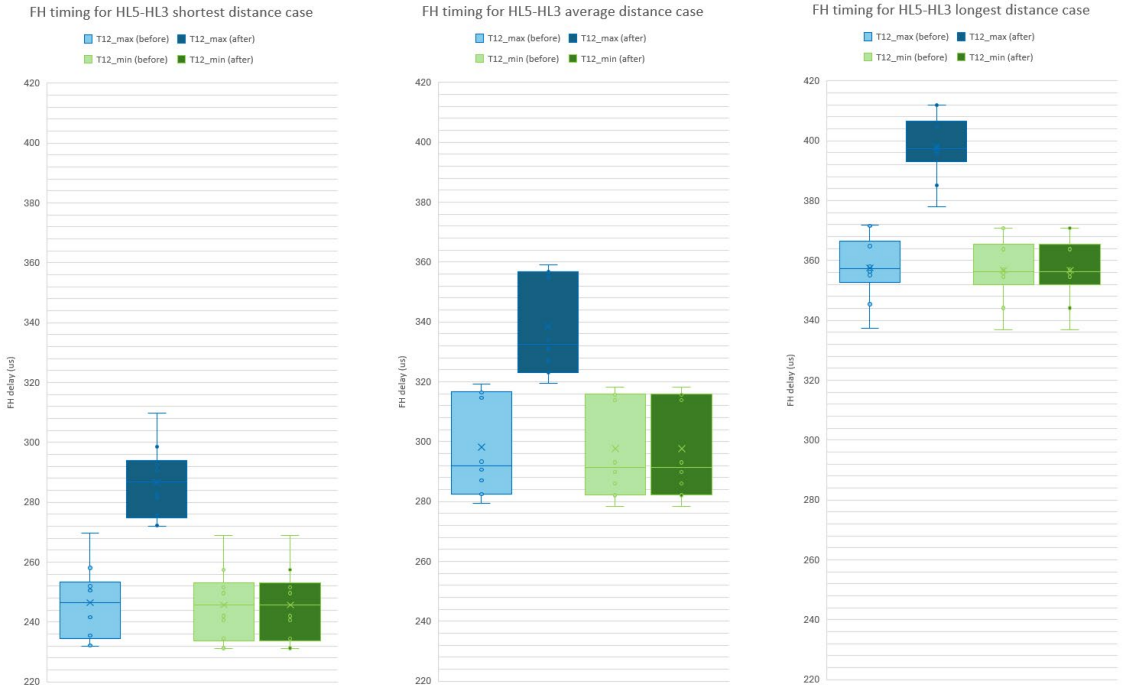


FIGURE 104. FH DELAYS FOR O-DU AT HL3

In turn, it is shown in Figure 105 the centralization distance corresponding with these delays, going from approximately 37 km to 65 km.

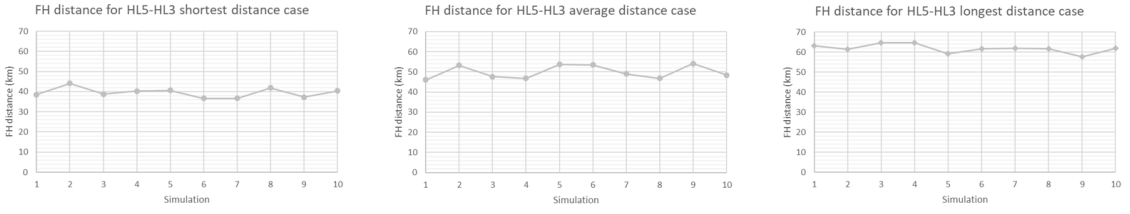


FIGURE 105. FH DISTANCES FOR O-DU AT HL3

In order to offer a clearer and more comprehensive understanding of the results, the following section presents a series of tables detailing the specific modifications implemented at each level of centralization and for each simulation. These tables present values that are not exactly the same that those tested in the testbed and presented at the end of this section, but they are extracted in the same conditions and are presented here intended to illustrate not only the extent of the changes but also the rationale and contextual application behind each adjustment, showing also the variability between simulations due to the inherent variability in network conditions and the specific centralization distance. For this purpose, ten simulations results are presented for each centralization degree.

Table 12 specifically highlights two key aspects for every scenario analysed: the centralization distance obtained both in the preliminary scenario with the inputs provided, and after applying the timing changes to optimize the scenario, and the corresponding study case applied, which outlines the context or methodology adopted in that particular instance. It can be seen that in all the cases, the optimization method applied is the study case 1 (see Smart K2.2 Key Concept in 6G BLUR SMART E4 deliverable), based on adjusting the O-DU TW. So, the applied changes consist in extending *T12_max* (and consequently *T1a_min*) to relax conditions in transport NW routers, thus reducing the O-DU TW to its optimum value, that is six symbol times (*Ts*), i.e. 200 µs. For this reason, the centralization distance does not change since in the preliminary scenario the good conditions in the network already allow us to reach the proposed distances.

TABLE 12. DETAILED RESULTS OF DISTANCE AND STUDY CASE APPLIED FOR EACH SIMULATION

	Simulation	Case study	Centralization distance (before)		Centralization distance (after)	
HL4 – Shortest distance (10-15 km)	1	1	12,19	km	12,20	km
	2	1	10,93	km	10,94	km
	3	1	13,55	km	13,56	km
	4	1	13,28	km	13,28	km
	5	1	14,80	km	14,80	km
	6	1	13,76	km	13,76	km
	7	1	14,45	km	14,46	km

	8	1	10,75	km	10,76	km
	9	1	14,07	km	14,08	km
	10	1	10,98	km	10,98	km
	Avg.		**12,88**	**km**	**12,88**	**km**
HL4 - Average distance (15-20 km)	**Simulation**	**Case study**	**Centralization distance (before)**		**Centralization distance (after)**	
	1	1	19,07	km	19,08	km
	2	1	18,16	km	18,16	km
	3	1	19,79	km	19,80	km
	4	1	19,79	km	19,80	km
	5	1	17,11	km	17,12	km
	6	1	18,28	km	18,28	km
	7	1	18,39	km	18,40	km
	8	1	18,28	km	18,28	km
	9	1	16,38	km	16,38	km
	10	1	18,47	km	18,48	km
	Avg.		**18,37**	**km**	**18,38**	**km**
HL4 - Longest distance (20-25 km)	**Simulation**	**Case study**	**Centralization distance (before)**		**Centralization distance (after)**	
	1	1	22,19	km	22,20	km
	2	1	20,93	km	20,94	km
	3	1	23,55	km	23,56	km
	4	1	23,28	km	23,28	km
	5	1	24,80	km	24,80	km
	6	1	23,76	km	23,76	km
	7	1	24,45	km	24,46	km

	8	1	20,75	km	20,76	km
	9	1	24,07	km	24,08	km
	10	1	20,98	km	20,98	km
	Avg.		**22,88**	**km**	**22,88**	**km**
HL3 – Shortest distance (35-45 km)	**Simulation**	**Case study**	**Centralization distance (before)**		**Centralization distance (after)**	
	1	1	38,52	km	38,52	km
	2	1	44,17	km	44,18	km
	3	1	38,80	km	38,80	km
	4	1	40,31	km	40,32	km
	5	1	40,69	km	40,70	km
	6	1	36,62	km	36,62	km
	7	1	36,66	km	36,66	km
	8	1	41,89	km	41,90	km
	9	1	37,29	km	37,30	km
	10	1	40,38	km	40,38	km
	Avg.		**39,53**	**km**	**39,54**	**km**
HL3 – Average distance (45-55 km)	**Simulation**	**Case study**	**Centralization distance (before)**		**Centralization distance (after)**	
	1	1	46,07	km	46,08	km
	2	1	53,17	km	53,18	km
	3	1	47,60	km	47,60	km
	4	1	46,82	km	46,82	km
	5	1	53,69	km	53,70	km
	6	1	53,53	km	53,54	km

	7	1	49,02	km	49,02	km
	8	1	46,84	km	46,84	km
	9	1	54,03	km	54,04	km
	10	1	48,38	km	48,38	km
	Avg.		**49,92**	**km**	**49,92**	**km**
HL3 - Longest distance (55-65 km)	**Simulation**	**Case study**	**Centralization distance (before)**		**Centralization distance (after)**	
	1	1	63,15	km	63,16	km
	2	1	61,32	km	61,32	km
	3	1	64,58	km	64,58	km
	4	1	64,57	km	64,58	km
	5	1	59,22	km	59,22	km
	6	1	61,56	km	61,56	km
	7	1	61,79	km	61,80	km
	8	1	61,55	km	61,56	km
	9	1	57,77	km	57,78	km
	10	1	61,95	km	61,96	km
	Avg.		**61,75**	**km**	**61,75**	**km**

Regarding the O-DU timing, Table 13 presents the TW boundaries again before and after applying the changes thanks to the MATLAB tool analysis. It can be seen that while *T1a_max* does not change, *T1a_min* is extended (giving more budget for *T12_max*) so the difference between both values is 200 µs, that is, the O-DU TW optimum size. This allows to have more relaxed conditions in the network, what translates to have more margin for possible queueing delays that may appear in the worst conditions. These changes are consequently applied also in UL for the O-DU RW.

TABLE 13. DETAILED RESULTS OF O-DU TW TIMING FOR EACH SIMULATION

HL4 - Shortest distance (10-15 km)	Simulation	E2E delay		
		T1a_max	T1a_min	Difference

		Before		After		Correction		Before		After		Correction		Before		After	
	1	468	us	468	us	0	us	228	us	268	us	40	us	240	us	200	us
	2	461,7	us	461,7	us	0	us	221,5	us	261,7	us	40,2	us	240,2	us	200	us
	3	474,8	us	474,8	us	0	us	234,7	us	274,8	us	40,1	us	240,1	us	200	us
	4	473,4	us	473,4	us	0	us	232,8	us	273,4	us	40,6	us	240,6	us	200	us
	5	481	us	481	us	0	us	240,6	us	281	us	40,4	us	240,4	us	200	us
	6	475,8	us	475,8	us	0	us	235,5	us	275,8	us	40,3	us	240,3	us	200	us
	7	479,3	us	479,3	us	0	us	239,1	us	279,3	us	40,2	us	240,2	us	200	us
	8	460,8	us	460,8	us	0	us	220,5	us	260,8	us	40,3	us	240,3	us	200	us
	9	477,4	us	477,4	us	0	us	237,2	us	277,4	us	40,2	us	240,2	us	200	us
	10	461,9	us	461,9	us	0	us	221,6	us	261,9	us	40,3	us	240,3	us	200	us
	Avg.	**471,41**	**us**	**471,41**	**us**	**0**	**us**	**231,15**	**us**	**271,41**	**us**	**40,26**	**us**	**240,26**	**us**	**200**	**us**
HL4 - Average distance (15-20 km)	**Simulation**	**E2E delay**															
		T1a_max						**T1a_min**						**Difference**			
		Before		**After**		**Correction**		**Before**		**After**		**Correction**		**Before**		**After**	
	1	502,4	us	502,4	us	0	us	262,4	us	302,4	us	40	us	240	us	200	us
	2	497,8	us	497,8	us	0	us	257,3	us	297,8	us	40,5	us	240,5	us	200	us
	3	506	us	506	us	0	us	266	us	306	us	40	us	240	us	200	us
	4	506	us	506	us	0	us	265,7	us	306	us	40,3	us	240,3	us	200	us
	5	492,6	us	492,6	us	0	us	252,9	us	292,6	us	39,7	us	239,7	us	200	us
	6	498,4	us	498,4	us	0	us	258,3	us	298,4	us	40,1	us	240,1	us	200	us
	7	499	us	499	us	0	us	258,9	us	299	us	40,1	us	240,1	us	200	us
	8	498,4	us	498,4	us	0	us	257,9	us	298,4	us	40,5	us	240,5	us	200	us
	9	488,9	us	488,9	us	0	us	248,4	us	288,9	us	40,5	us	240,5	us	200	us
	10	499,4	us	499,4	us	0	us	259,1	us	299,4	us	40,3	us	240,3	us	200	us
	Avg.	**498,89**	**us**	**498,89**	**us**	**0**	**us**	**258,69**	**us**	**298,89**	**us**	**40,2**	**us**	**240,2**	**us**	**200**	**us**
HL4 - Longest distance (20-25 km)	**Simulation**	**E2E delay**															
		T1a_max						**T1a_min**						**Difference**			
		Before		**After**		**Correction**		**Before**		**After**		**Correction**		**Before**		**After**	
	1	518	us	518	us	0	us	278	us	318	us	40	us	240	us	200	us
	2	511,7	us	511,7	us	0	us	271,5	us	311,7	us	40,2	us	240,2	us	200	us
	3	524,8	us	524,8	us	0	us	284,7	us	324,8	us	40,1	us	240,1	us	200	us
	4	523,4	us	523,4	us	0	us	282,8	us	323,4	us	40,6	us	240,6	us	200	us
	5	531	us	531	us	0	us	290,6	us	331	us	40,4	us	240,4	us	200	us
	6	525,8	us	525,8	us	0	us	285,5	us	325,8	us	40,3	us	240,3	us	200	us
	7	529,3	us	529,3	us	0	us	289,1	us	329,3	us	40,2	us	240,2	us	200	us
	8	510,8	us	510,8	us	0	us	270,5	us	310,8	us	40,3	us	240,3	us	200	us
	9	527,4	us	527,4	us	0	us	287,2	us	327,4	us	40,2	us	240,2	us	200	us

	10	511,9	us	511,9	us	0	us	271,6	us	311,9	us	40,3	us	240,3	us	200	us
	Avg.	**521,41**	**us**	**521,41**	**us**	**0**	**us**	**281,15**	**us**	**321,41**	**us**	**40,26**	**us**	**240,26**	**us**	**200**	**us**
HL3 - Shortest distance (35-45 km)	**Simulation**	**E2E delay**															
		T1a_max						**T1a_min**						**Difference**			
		Before		**After**		**Correction**		**Before**		**After**		**Correction**		**Before**		**After**	
	1	615,6	us	615,6	us	0	us	375,6	us	415,6	us	40	us	240	us	200	us
	2	643,9	us	643,9	us	0	us	403,8	us	443,9	us	40,1	us	240,1	us	200	us
	3	617	us	617	us	0	us	376,3	us	417	us	40,7	us	240,7	us	200	us
	4	624,6	us	624,6	us	0	us	384,5	us	424,6	us	40,1	us	240,1	us	200	us
	5	626,5	us	626,5	us	0	us	385,9	us	426,5	us	40,6	us	240,6	us	200	us
	6	606,1	us	606,1	us	0	us	365,9	us	406,1	us	40,2	us	240,2	us	200	us
	7	606,3	us	606,3	us	0	us	366,1	us	406,3	us	40,2	us	240,2	us	200	us
	8	632,5	us	632,5	us	0	us	392,1	us	432,5	us	40,4	us	240,4	us	200	us
	9	609,5	us	609,5	us	0	us	369,4	us	409,5	us	40,1	us	240,1	us	200	us
	10	624,9	us	624,9	us	0	us	384,7	us	424,9	us	40,2	us	240,2	us	200	us
	Avg.	**620,69**	**us**	**620,69**	**us**	**0**	**us**	**380,43**	**us**	**420,69**	**us**	**40,26**	**us**	**240,26**	**us**	**200**	**us**
HL3 - Average distance (45-55 km)	**Simulation**	**E2E delay**															
		T1a_max						**T1a_min**						**Difference**			
		Before		**After**		**Correction**		**Before**		**After**		**Correction**		**Before**		**After**	
	1	653,4	us	653,4	us	0	us	413,3	us	453,4	us	40,1	us	240,1	us	200	us
	2	688,9	us	688,9	us	0	us	448,6	us	488,9	us	40,3	us	240,3	us	200	us
	3	661	us	661	us	0	us	421,1	us	461	us	39,9	us	239,9	us	200	us
	4	657,1	us	657,1	us	0	us	416,4	us	457,1	us	40,7	us	240,7	us	200	us
	5	691,5	us	691,5	us	0	us	451,1	us	491,5	us	40,4	us	240,4	us	200	us
	6	690,7	us	690,7	us	0	us	450,4	us	490,7	us	40,3	us	240,3	us	200	us
	7	668,1	us	668,1	us	0	us	427,3	us	468,1	us	40,8	us	240,8	us	200	us
	8	657,2	us	657,2	us	0	us	416,6	us	457,2	us	40,6	us	240,6	us	200	us
	9	693,2	us	693,2	us	0	us	453,2	us	493,2	us	40	us	240	us	200	us
	10	664,9	us	664,9	us	0	us	424,6	us	464,9	us	40,3	us	240,3	us	200	us
	Avg.	**672,6**	**us**	**672,6**	**us**	**0**	**us**	**432,26**	**us**	**472,6**	**us**	**40,34**	**us**	**240,34**	**us**	**200**	**us**
HL3 - Longest distance (55-65 km)	**Simulation**	**E2E delay**															
		T1a_max						**T1a_min**						**Difference**			
		Before		**After**		**Correction**		**Before**		**After**		**Correction**		**Before**		**After**	
	1	738,8	us	738,8	us	0	us	498,8	us	538,8	us	40	us	240	us	200	us
	2	729,6	us	729,6	us	0	us	489,1	us	529,6	us	40,5	us	240,5	us	200	us
	3	745,9	us	745,9	us	0	us	505,9	us	545,9	us	40	us	240	us	200	us
	4	745,9	us	745,9	us	0	us	505,6	us	545,9	us	40,3	us	240,3	us	200	us
	5	719,1	us	719,1	us	0	us	479,4	us	519,1	us	39,7	us	239,7	us	200	us

	6	730,8	us	730,8	us	0	us	490,7	us	530,8	us	40,1	us	240,1	us	200	us
	7	732	us	732	us	0	us	491,9	us	532	us	40,1	us	240,1	us	200	us
	8	730,8	us	730,8	us	0	us	490,3	us	530,8	us	40,5	us	240,5	us	200	us
	9	711,9	us	711,9	us	0	us	471,4	us	511,9	us	40,5	us	240,5	us	200	us
	10	732,8	us	732,8	us	0	us	492,5	us	532,8	us	40,3	us	240,3	us	200	us
	Avg.	**731,76**	**us**	**731,76**	**us**	**0**	**us**	**491,56**	**us**	**531,76**	**us**	**40,2**	**us**	**240,2**	**us**	**200**	**us**

Likewise, in the case of FH timing *T12_min* does not change and *T12_max* is extended enough to reach O-DU TW optimum size, as can be seen in Table 14.

TABLE 14. DETAILED RESULTS OF FH TIMING FOR EACH SIMULATION

	Simulation	FH delay															
		T12_max						T12_min						Difference			
		Before		After		Correction		Before		After		Correction		Before		After	
HL4 - Shortest distance (10-15 km)	1	94	us	134	us	40	us	93	us	93	us	0	us	1	us	41	us
	2	87,5	us	127,7	us	40,2	us	86,7	us	86,7	us	0	us	0,8	us	41	us
	3	100,7	us	140,8	us	40,1	us	99,8	us	99,8	us	0	us	0,9	us	41	us
	4	98,8	us	139,4	us	40,6	us	98,4	us	98,4	us	0	us	0,4	us	41	us
	5	106,6	us	147	us	40,4	us	106	us	106	us	0	us	0,6	us	41	us
	6	101,5	us	141,8	us	40,3	us	100,8	us	100,8	us	0	us	0,7	us	41	us
	7	105,1	us	145,3	us	40,2	us	104,3	us	104,3	us	0	us	0,8	us	41	us
	8	86,5	us	126,8	us	40,3	us	85,8	us	85,8	us	0	us	0,7	us	41	us
	9	103,2	us	143,4	us	40,2	us	102,4	us	102,4	us	0	us	0,8	us	41	us
	10	87,6	us	127,9	us	40,3	us	86,9	us	86,9	us	0	us	0,7	us	41	us
	Avg.	97,15	us	137,41	us	40,26	us	96,41	us	96,41	us	0	us	0,74	us	41	us
	Simulation	**FH delay**															
		T12_max						**T12_min**						**Difference**			
		Before		**After**		**Correction**		**Before**		**After**		**Correction**		**Before**		**After**	
HL4 - Average distance (15-20 km)	1	128,4	us	168,4	us	40	us	127,4	us	127,4	us	0	us	1	us	41	us
	2	123,3	us	163,8	us	40,5	us	122,8	us	122,8	us	0	us	0,5	us	41	us
	3	132	us	172	us	40	us	131	us	131	us	0	us	1	us	41	us
	4	131,7	us	172	us	40,3	us	131	us	131	us	0	us	0,7	us	41	us
	5	118,9	us	158,6	us	39,7	us	117,6	us	117,6	us	0	us	1,3	us	41	us
	6	124,3	us	164,4	us	40,1	us	123,4	us	123,4	us	0	us	0,9	us	41	us
	7	124,9	us	165	us	40,1	us	124	us	124	us	0	us	0,9	us	41	us
	8	123,9	us	164,4	us	40,5	us	123,4	us	123,4	us	0	us	0,5	us	41	us
	9	114,4	us	154,9	us	40,5	us	113,9	us	113,9	us	0	us	0,5	us	41	us
	10	125,1	us	165,4	us	40,3	us	124,4	us	124,4	us	0	us	0,7	us	41	us

	Avg.	124,69	us	164,89	us	40,2	us	123,89	us	123,89	us	0	us	0,8	us	41	us
HL4 - Longest distance (20-25 km)	**Simulation**	**FH delay**															
		T12_max						**T12_min**						**Difference**			
		Before		**After**		**Correction**		**Before**		**After**		**Correction**		**Before**		**After**	
	1	144	us	184	us	40	us	143	us	143	us	0	us	1	us	41	us
	2	137,5	us	177,7	us	40,2	us	136,7	us	136,7	us	0	us	0,8	us	41	us
	3	150,7	us	190,8	us	40,1	us	149,8	us	149,8	us	0	us	0,9	us	41	us
	4	148,8	us	189,4	us	40,6	us	148,4	us	148,4	us	0	us	0,4	us	41	us
	5	156,6	us	197	us	40,4	us	156	us	156	us	0	us	0,6	us	41	us
	6	151,5	us	191,8	us	40,3	us	150,8	us	150,8	us	0	us	0,7	us	41	us
	7	155,1	us	195,3	us	40,2	us	154,3	us	154,3	us	0	us	0,8	us	41	us
	8	136,5	us	176,8	us	40,3	us	135,8	us	135,8	us	0	us	0,7	us	41	us
	9	153,2	us	193,4	us	40,2	us	152,4	us	152,4	us	0	us	0,8	us	41	us
	10	137,6	us	177,9	us	40,3	us	136,9	us	136,9	us	0	us	0,7	us	41	us
	Avg.	147,15	us	187,41	us	40,26	us	146,41	us	146,41	us	0	us	0,74	us	41	us
HL3 - Shortest distance (35-45 km)	**Simulation**	**FH delay**															
		T12_max						**T12_min**						**Difference**			
		Before		**After**		**Correction**		**Before**		**After**		**Correction**		**Before**		**After**	
	1	241,6	us	281,6	us	40	us	240,6	us	240,6	us	0	us	1	us	41	us
	2	269,8	us	309,9	us	40,1	us	268,9	us	268,9	us	0	us	0,9	us	41	us
	3	242,3	us	283	us	40,7	us	242	us	242	us	0	us	0,3	us	41	us
	4	250,5	us	290,6	us	40,1	us	249,6	us	249,6	us	0	us	0,9	us	41	us
	5	251,9	us	292,5	us	40,6	us	251,5	us	251,5	us	0	us	0,4	us	41	us
	6	231,9	us	272,1	us	40,2	us	231,1	us	231,1	us	0	us	0,8	us	41	us
	7	232,1	us	272,3	us	40,2	us	231,3	us	231,3	us	0	us	0,8	us	41	us
	8	258,1	us	298,5	us	40,4	us	257,5	us	257,5	us	0	us	0,6	us	41	us
	9	235,4	us	275,5	us	40,1	us	234,5	us	234,5	us	0	us	0,9	us	41	us
	10	250,7	us	290,9	us	40,2	us	249,9	us	249,9	us	0	us	0,8	us	41	us
	Avg.	246,43	us	286,69	us	40,26	us	245,69	us	245,69	us	0	us	0,74	us	41	us
HL3 - Average distance (45-55 km)	**Simulation**	**FH delay**															
		T12_max						**T12_min**						**Difference**			
		Before		**After**		**Correction**		**Before**		**After**		**Correction**		**Before**		**After**	
	1	279,3	us	319,4	us	40,1	us	278,4	us	278,4	us	0	us	0,9	us	41	us
	2	314,6	us	354,9	us	40,3	us	313,9	us	313,9	us	0	us	0,7	us	41	us
	3	287,1	us	327	us	39,9	us	286	us	286	us	0	us	1,1	us	41	us
	4	282,4	us	323,1	us	40,7	us	282,1	us	282,1	us	0	us	0,3	us	41	us
	5	317,1	us	357,5	us	40,4	us	316,5	us	316,5	us	0	us	0,6	us	41	us
	6	316,4	us	356,7	us	40,3	us	315,7	us	315,7	us	0	us	0,7	us	41	us

	7	293,3	us	334,1	us	40,8	us	293,1	us	293,1	us	0	us	0,2	us	41	us
	8	282,6	us	323,2	us	40,6	us	282,2	us	282,2	us	0	us	0,4	us	41	us
	9	319,2	us	359,2	us	40	us	318,2	us	318,2	us	0	us	1	us	41	us
	10	290,6	us	330,9	us	40,3	us	289,9	us	289,9	us	0	us	0,7	us	41	us
	Avg.	298,26	us	338,6	us	40,34	us	297,6	us	297,6	us	0	us	0,66	us	41	us
HL3 - Longest distance (55-65 km)	**Simulation**	**FH delay**															
		T12_max						**T12_min**						**Difference**			
		Before		**After**		**Correction**		**Before**		**After**		**Correction**		**Before**		**After**	
	1	364,8	us	404,8	us	40	us	363,8	us	363,8	us	0	us	1	us	41	us
	2	355,1	us	395,6	us	40,5	us	354,6	us	354,6	us	0	us	0,5	us	41	us
	3	371,9	us	411,9	us	40	us	370,9	us	370,9	us	0	us	1	us	41	us
	4	371,6	us	411,9	us	40,3	us	370,9	us	370,9	us	0	us	0,7	us	41	us
	5	345,4	us	385,1	us	39,7	us	344,1	us	344,1	us	0	us	1,3	us	41	us
	6	356,7	us	396,8	us	40,1	us	355,8	us	355,8	us	0	us	0,9	us	41	us
	7	357,9	us	398	us	40,1	us	357	us	357	us	0	us	0,9	us	41	us
	8	356,3	us	396,8	us	40,5	us	355,8	us	355,8	us	0	us	0,5	us	41	us
	9	337,4	us	377,9	us	40,5	us	336,9	us	336,9	us	0	us	0,5	us	41	us
	10	358,5	us	398,8	us	40,3	us	357,8	us	357,8	us	0	us	0,7	us	41	us
	Avg.	**357,56**	**us**	**397,76**	**us**	**40,2**	**us**	**356,76**	**us**	**356,76**	**us**	**0**	**us**	**0,8**	**us**	**41**	**us**

With all this information at hand, and once we had determined the input values required by the MATLAB tool, namely, the O-RU processing timing and the FH conditions, we were able to generate the centralization scenarios for each degree of centralization shown in Table 15. These scenarios allowed us to extract FH latencies and O-DU processing times, which we then applied within the testbed environment. This enabled us to empirically validate the results obtained through simulation, ensuring that the theoretical findings aligned with practical performance under real or emulated network conditions.

TABLE 15. CENTRALIZATION SCENARIOS

Centralization degree	Case study applied		Centralization distance [km]	O-DU timing				FH timing			
	Before [µs]	After [µs]	Before [µs]	After [µs]	Before [µs]	After [µs]	Before [µs]	After [µs]			
				T1a_max		T1a_min		T12_max		T12_min	
HL4-shortest	1		12,8	471,4	471,4	231,2	271,4	97,2	137,4	96,4	96,4
HL4-average	1		18,38	498,9	498,9	258,7	298,9	124,7	164,9	123,9	123,9

HL4-longest	1	22,88	521,4	521,4	281,2	321,4	147,2	187,4	146,4	146,4
HL3-shortest	1	39,54	620,7	620,7	380,4	420,7	246,4	286,7	245,7	245,7
HL3-average	1	49,92	672,6	672,6	432,3	472,6	298,3	338,6	297,6	297,6
HL3-longest	1	61,96	731,8	731,8	491,6	531,8	357,6	397,8	356,8	356,8

3.2.1.2.1.5 O-DU optimization

As it can be observed, in the scenario with the highest degree of centralization, we were able to achieve a distance of nearly 65 km between the O-DU and the O-RU. Along with the corresponding FH delay values derived for this configuration, we utilized the srsRAN software to validate system behaviour under these conditions.

Through these tests, it was verified that, for this level of centralization, it is not necessary to modify or fine-tune the O-DU protocols to accommodate the increased delays introduced by the extended transport distance. This potential adjustment of O-DU protocols remains an open area for future research. As such, the only modification required in our current study was the adjustment of the window parameters on the O-DU side to align with the extended fronthaul conditions. More information about this can be found in Section of KC K2.4 of 6GSMART E4 deliverable.

Specifically, srsRAN Project is the open-source RAN solution developed by SRS (one of the project partners), which is fully compliant with the 3GPP and O-RAN specifications. Given the requirements of the PoC, we opted for the monolithic version of the software (CU and DU running in the same executable), since it is easier to configure for high bandwidth cells with strict latency requirements. All the timing settings discussed above are easily configurable in the "ru_ofh" section of the srsRAN Project yaml configuration file:

```
ru_ofh:
  t1a_max_cp_dl          # T1a maximum value for downlink Control-Plane. Supported: [0 – 1960]
  t1a_min_cp_dl          # T1a minimum value for downlink Control-Plane. Supported: [0 - 1960]
  t1a_max_cp_ul          # T1a maximum value for uplink Control-Plane. Supported: [0 - 1960]
  t1a_min_cp_ul          # T1a minimum value for uplink Control-Plane. Supported: [0 - 1960]
  t1a_max_up             # T1a maximum value for User-Plane. Supported: [0 - 1960]
  t1a_min_up             # T1a minimum value for User-Plane. Supported: [0 - 1960]
  ta4_max                # Ta4 maximum value for User-Plane. Supported: [0 - 1960]
  ta4_min          # Ta4 minimum value for User-Plane. Supported: [0 - 1960]
```

where all values are interpreted as a time interval in microseconds. It is worth remarking that these values are later converted by the software into an integer number of OFDM symbols (approximately 33 μs with a subcarrier spacing of 30 kHz). Note that, when modifying these time windows, the operator may also need to fine tune the following parameter in the "expert_phy" section:

```
expert_phy:
   max_proc_delay   # Maximum allowed DL processing delay in slots. Supported: [1 - 30]
```

Roughly speaking, this delay (expressed in units of slot duration, that is 0.5 ms with a subcarrier spacing of 30 kHz) encompasses all the steps required by a DL slot from its scheduling until its transmission at the antenna port at the RU, thus including the delay introduced by the O-FH protocol when transferring data from the DU to the RU.

The timing results obtained analytically for the FH have been validated in an Open RAN end-to-end (E2E) testbed set up at the 6G-SANDBOX premises. This validation process was carried out to cross-check the simulation outcomes and to extract practical, real-world results within a controlled laboratory environment.

By replicating the centralization scenarios and fronthaul conditions in the testbed, we were able to empirically verify that the analytical models and software tool predictions closely aligned with the observed system behaviour. The testbed allowed us to assess latency metrics under realistic conditions across an Open RAN disaggregated architecture.

3.2.1.2.2 QoS scheduling schemes

We have proposed various MAC schedulers to address QoS requirements of delay-critical XR traffic flows. In particular:

- **QoS scheduler**: considers the resource type (GBR, non-GBR, dc-GBR), the priority level of the flow, the packet delay budget (PDB), the head-of-line (HOL) delay and the PF metric.
- **DPP scheduler**: considers the guaranteed flow bit rate (GFBR) requirement, the resource utilization, and stability of the queues, using Lyapunov control theory.
- **PDU set-aware scheduler**: extends DPP scheduler, using Lyapunov control theory, with PDU set information, like the PDU set delay budget and PDU set size.

They have been implemented in ns-3 5G-LENA and evaluated through system-level simulations using mixed AR/VR/CG scenarios. This work is summarized in SMART K2.5 and has contributed to publications [16], [17] and [18].FH control methods

We have developed various FH control methods to control the downlink data bulks that go through a limited-capacity fronthaul link. In particular:

- Dropping
- Postponing
- OptimizeRbs
- OptimizeMcs
- AdjustRbs

They have been implemented in ns-3 5G-LENA and evaluated through system-level simulations using mixed AR/VR/CG scenarios. This work is summarized in SMART K1.2 (could be found at 6GSMART E4 deliverable) and has contributed to publications [19], [20].

3.2.1.2.3 Flexible Functional Splits

We have derived the capacity and latency requirements that transport networks (fronthaul and midhaul) will need to meet in future 6G deployments, for different functional split options. We have analyzed the impact of different functional split options using the 5G-LENA simulator. This represents the first step toward implementing dynamic Flexible Functional Splits in 5G-LENA. This work is summarized in SMART K2.3 (could be found at 6GSMART E4 deliverable) and has contributed to publication [21].

3.2.1.3 Experimental Implementation and Assessment

3.2.1.3.1 Testbed Description

A structured and modular testbed is designed to validate the theoretical analysis in practical scenarios (see Figure 106). This testbed emulates a scenario in which a O-DU/O-CU is connected to an O-RU through an O-FH that introduces a significant delay. The O-FH delay is introduced by a Keysight instrument, Network Emulator 3 (NE3), which has the ability to introduce a programmable delay to all packets that travers it in either direction. With this, we emulate the operation of (a part of) a system in which O-DU/O-CUs are centralized at a significant distance from the O-RUs, as illustrated in Figure 107.

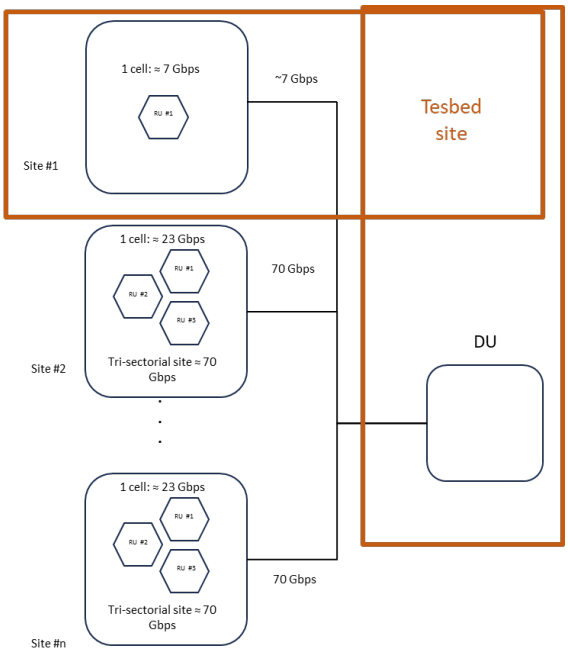


FIGURE 106. CONSIDERED SYSTEM MODEL, WITH THE PART EMULATED BY THE TESTBED HIGHLIGHTED IN THE ORANGE FRAME

3.2.1.3.1.1 Overall testbed

The conceptual architecture and practical implementation of this testbed are shown in Figure 107. It consists of seven components, together forming an end-to-end cellular network emulation environment with automated and centralized control. We explain each component and the corresponding functionalities below in more detail.

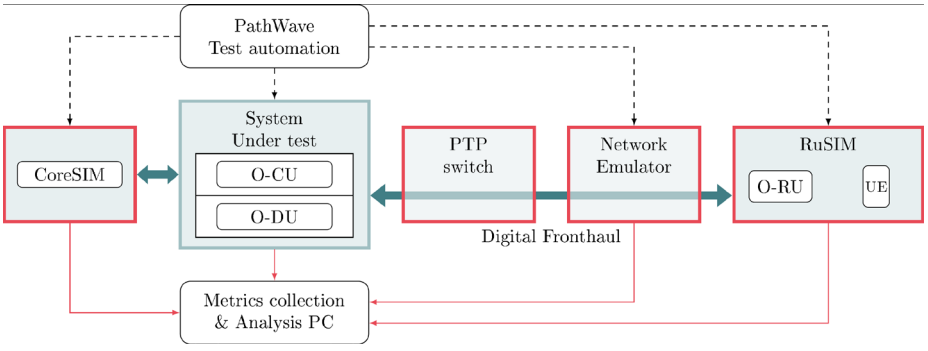


FIGURE 107. DIAGRAM OF THE TESTBED COMPONENTS

3.2.1.3.1.2 RuSIM

RuSIM [62] is a Keysight emulator of O-RUs and UEs. It can emulate multiple O-RUs with configurable cell settings while creating IP traffic and mimicking thousands of devices running real voice and data connections. This makes RuSIM a powerful tool for testing the complete protocol stack for both 4G and 5G (NSA and SA) through the eCPRI interface, and particularly suited to test the performance of the O-FH interface as is the focus in this PoC. The configuration and control of RuSIM is governed by AirMosaic, a Keysight proprietary software that is also used to supervise the generation of user traffic.

The server running RuSIM has the following hardware specifications:

- Manufacturer: Supermicro
- CPU
  - Model: Intel Xeon Gold 6238R CPU @ 2.20GHz
  - Sockets: 2
  - Cores per socket: 28
  - Threads per core: 1
  - CPU max frequency: 4.0 GHz
  - CPU min frequency: 1.0 GHz
- RAM
  - DDR version: DDR4
  - Frequency: 3200 MHz
  - Memory per module: 16 GiB
  - Number of modules: 8
  - Total memory: 128 GiB
- O-FH NIC
  - Type: fiber
  - Link mode: 10000baseCR/Full
  - Speed: 10 Gbps

3.2.1.3.1.3 NE3

To emulate the network impairments in the fronthaul link, we utilize Keysight Network Emulator 3 (NE3) [63]. This instrument enables users to accurately introduce the real conditions that occur over live production networks, such as packet drops, delays, reordering, duplications, etc. In this particular testbed, NE3 is used to emulate the delays that will be present in the O-FH in the case of centralization of O-DUs, due to the large physical length of the line and possible switches and routing devices in its path.

3.2.1.3.1.4 PTP Switch

Given features such as increased radio bandwidth or carrier aggregation, where each aggregated component carrier needs to be accurately time-aligned, one rising design challenge in 5G and 6G networks is achieving precise timing. This becomes even more important with the introduction of functional splits in the O-RAN architecture, since equipment from different vendors may be placed separately at different sites, making it impossible to share the same local clock across equipment. Therefore, tight time synchronization between different network components is necessary in 5G O-RAN deployments. For instance, it is essential to synchronize O-DU and O-RU in Split 7.2. To achieve this, O-RAN Alliance defines various techniques, as specified in [64]. In our testbed, we implement PTP (IEEE 1588 Precision Time Protocol) synchronization between O-RU and O-DU via the S-plane based on the LLS-C3 configuration.

We utilize a timing-aware O-RAN switch, Falcon-RX [65], as the primary PTP source. To generate and transmit the PTP synchronization signal to the secondary nodes, it connects to an external GPS antenna and receives the GPS signal from multiple satellites. Then, the lower layers emulation server (RuSIM, functioning as O-RU) and the system under test (functioning as O-DU) connect to the switch via fibers for both S-plane and C-/U-plane traffic. Note that for the lower layers emulation server, the S-plane and C-/U-plane share the same interface, but for the system under test, as recommended by the srsRAN setup, the S-plane and C-/U-plane are separated into two independent interfaces, connected by different cables.

On the RuSIM side, ptp4l runs on the server, using the received PTP signal to synchronize its network interface card (NIC) clock to the primary PTP source. Then, timing-related values such as system frame number (SFN) are calculated based on the synchronized NIC clock. On the other hand, since the gNB utilizes the system clock on the hosting server, we execute ptp4l and phc2sys on the test server, which use the PTP signal to synchronize the server's system clock to that of the primary clock.

3.2.1.3.1.5 srsRAN gNB

The O-RU emulated by RuSIM is controlled by the srsRAN gNB solution from SRS. The srsRAN Project is the software-based complete RAN (O-CU and O-DU) solution from SRS. It is compliant with 3GPP Release 17 and O-RAN Alliance specifications. The software, distributed as open source, is entirely written by SRS with minimal external dependencies, it includes the full L1/2/3 stack and is portable across processor architectures. The srsRAN features that are most relevant in the context of the PoC are:

- Support for all FR1 bands, both TDD and FDD, and for all bandwidths up to 100 MHz; 4x4 downlink MIMO;
- Support for Split 7.2 enabled by the in-house O-FH library;

- AVX512-accelerated kernels for LDPC encoding/decoding and O-FH IQ samples compression/decompression;
- Exhaustive configuration via yaml files.

More details about srsRAN, and especially on the advances in the O-FH library driven by this PoC, are provided as part of the key concept SMART-K2.1. In this experimental setup, the SRS software is deployed in a server with the following characteristics:

- Manufacturer: Supermicro
- CPU
  - Model: Intel Xeon Platinum 8260 CPU @ 2.40GHz
  - Sockets: 2
  - Cores per socket: 24
  - Threads per core: 2
  - CPU max frequency: 3.9 GHz
  - CPU min frequency: 1.0 GHz
- RAM
  - DDR version: DDR4
  - Frequency: 2400 MHz
  - Memory per module: 64 GiB
  - Number of modules: 8
  - Total memory: 512 GiB
- O-FH NIC
  - Type: fiber
  - Link mode: 10000baseCR/Full
  - Speed: 10 Gbps
- Back-haul NIC
  - Type: fiber
  - Link mode: 10000baseCR/Full
  - Speed: 10 Gbps

In order to facilitate that the software runs in real time and no processing delays impair the operation of the O-DU/O-CU, Data Plane Development Kit (DPDK) is enabled at the server. DPDK [66]bypasses the kernel networking stack and directly accesses the NIC. Thus, it can reduce the overhead of packet processing. Given that the gNB is supposed to process a large number of packets in real-time, packet processing can significantly affect the overall performance. Based on this observation, to achieve high throughput and stable performance when running highly demanding cell scenarios (e.g., a cell with 100 MHz bandwidth using 4 antenna ports at the base station and 4 MIMO layers), we install

DPDK on the test server to improve the packet processing performance of gNB. To utilize and maximize the benefits of DPDK, we further configure the test server for DPDK execution, such as enabling 1 GB HugePages and binding the C-/U-Plane network interface to DPDK. By enabling DPDK, we notice a prominent gNB performance improvement in demanding scenario executions. For example, in the aforementioned 4 × 4, 100 MHz cell, running the gNB without DPDK causes BLER and low throughput to connected users, while this degradation does not happen with DPDK.

3.2.1.3.1.6 CoreSIM

In order to be able to emulate the performance of an end-to-end network, a 5G Core emulator, CoreSIM, is used. CoreSIM is a UPF and AMF Core Network emulator developed by Keysight [67]. It simplifies RAN testing by eliminating Core Network unwanted dependencies, thereby allowing an easily controllable and repeatable testing environment. CoreSIM is highly scalable, as it can handle up to 100 independent test lines in parallel, supporting 10000 UEs and 1000 registrations/sec. It also offers great flexibility through simulators for both EPC NSA and 5G SA configurations, supporting various data, storage, voice, and video protocols.

3.2.1.3.1.7 Open RAN Studio Player and Capture Appliance

As a capturing and debugging device, we use Keysight's S5040A Open RAN Studio Player and Capture Appliance [68]. It is an instrument grade test and measurement appliance designed to operate with Keysight's PathWave Signal Studio and Open RAN Studio to emulate a Distributed Unit (O-DU), capture O-RAN uplink communications, and perform the measurements necessary to validate an O-RU's functional operation and performance.

3.2.1.3.1.8 KS8400 – OpenTAP

To control different equipment and run desired commands, we make use of the Keysight PathWave Test Automation tool (KS8400) [69], which is based on the open-source OpenTAP, to implement and sequence all test steps. The key point is that this tool can execute the test steps and corresponding logic specified in a test plan file (in tapplan format). Using different OpenTAP plugins, KS8400 can orchestrate the operation of the instruments and pieces of software included in the testbed in a synchronous manner, as well as performing some basic recording for a postprocessing of results.

3.2.1.3.2 Scenario description and testbed configuration parameters

In order to facilitate the presentation of results, we describe here the different scenarios that have been emulated using the described testbed, as well as the different configuration parameters for these.

3.2.1.3.2.1 Scenarios

Using the testbed described above, different O-DU/O-CU centralization scenarios can be emulated. The key aspect in the emulation of these scenarios concerns mainly the settings of the NE3 instrument, as it is used to introduce different O-FH delay characteristics that mimic the signal propagation delays and delay variation that is expected to be encountered in such scenarios.

As it will be detailed in the following, we consider three main types of scenarios:

- A baseline scenario, representing the standard distributed deployment where the O-DU is located at the cell site, with no or minimal spatial separation between O-RU and O-DU. The O-FH introduces a minimal delay.
- Centralization scenarios in which pooling of multiple O-DU is performed in a centralized location to jointly serve a set of multiple O-RUs placed at different cell-sites with geographically distributed locations. In this situation, the O-RUs may be at a significant distance of the O-DU, leading to an O-FH that introduces substantial delays in the traffic traversing it.
- Finally, in order to investigate the ultimate limitations in terms of delays that the O-FH can tolerate, we also evaluate some additional scenarios in which larger O-FH delays than those of the centralized case are considered, even if not realistic in current commercial deployments.

Scenarios comprise, first and foremost, NE3 settings, specifically, delay distribution and values. Specifically, the following distributions and delay parameters have been studied in this project:

- Constant distribution: delay
- Uniform uncorrelated distribution: minimum delay, maximum delay
- Uniform saw-toothed distribution: minimum delay, maximum delay

Constant distribution simply adds a constant delay to all packets passing through NE3. An option exists to make this impairment transparent for PTP packets, although it has been disabled for our tests. Disabling it allows to emulate physically long O-FH connection, where the signal propagation delays equally both PTP and non-PTP packets.

Uniform uncorrelated distribution adds a uniformly random delay independently to subsequent packets. It is important to know that no reordering is allowed with this impairment, and thus the real distribution might be shifted towards the larger value, depending on the packet size, pipe capacity, distribution width, etc. Separate impairments for packet reordering exist, but have not been used within the scope of this project.

Uniform saw-toothed distribution deterministically affects packet delay following a sawtooth shape. Delays for consecutive packets start from the maximum value and slowly decrease towards the minimum value. Once this is reached, delays start again from the maximum value, as show in Figure 108.

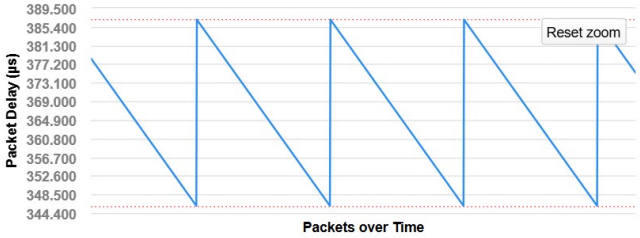


FIGURE 108. UNIFORM SAWTOOTHED DISTRIBUTION

3.2.1.3.2.1.1 Baseline scenario

The Baseline test scenario emulates the standard case in which the O-DU is co-located with the O-RU and, hence, the O-FH introduces negligible delay. The placement of NE3 in between RuSIM and the server running SRS' O-DU software implies, however, that the processing time of packets traversing NE3 introduces a small inherent delay that cannot be avoided. Hence, in the baseline scenario, described in Table 16, the O-FH introduces a constant delay equal to 15 us.

TABLE 16. DESCRIPTION OF THE BASELINE TEST SCENARIO

Test Scenario	Centralization Distance	Minimum O-FH delay	Maximum O-FH delay
BASELINE	0 km	15 us	15 us

3.2.1.3.2.1.2 O-DU Centralization Scenarios

We now turn our attention to the scenarios of interest for this PoC, which represents situations in which a pool of O-DUs is centralized at given location with the goal of serving a number of O-RUs that are deployed in cell sites geographically distributed around that polling location. The number of O-RUs served by the pool of O-DUs depends on the hierarchy level (HL) of the network at which the pooling is affected. In this case, and following the scenarios theorized by Telefónica based on its Spain´s transport network, we consider two different hierarchy levels at which pooling of O-DUs may occur: HL4 and HL3. The former corresponds to the aggregation of 9 cell sites, while the latter corresponds to the aggregation of ~23 cell sites. In addition, for each of these aggregation levels, we can consider different distance ranges for the cell sites, depending on the cell layout. In particular, we consider 3 distance ranges (short, average, and long), thus determining 6 possible scenarios, as described in Table 17.

TABLE 17. DESCRIPTION OF THE CENTRALIZATION TEST SCENARIOS

Test Scenario	Centralization Distance	Minimum O-FH delay	Maximum O-FH delay
HL4-SHORT	14.8 km	98 us	139 us
HL4-AVERAGE	19.8 km	123 us	164 us
HL4-LONG	22.12 km	134 us	175 us
HL3-SHORT	41.56 km	244 us	285 us
HL3-AVERAGE	47.78 km	275 us	316 us
HL3-LONG	61.96 km	346 us	387 us

As can be observed in Table 17, all 6 scenarios are characterized by a range of delays of 41 us (maximum FH delay – minimum FH delay) that the O-FH can introduce. In our testbed, this is implemented in different ways, depending on whether a constant or a variable delay is introduced by NE3:

- In the case of a constant delay, 3 possible subcases are considered for each of the test scenarios, which we coin as MIN, MID, and MAX. As their name indicate, in these three cases NE3 introduces a delay that corresponds to the minimum, mid-point, and maximum delay of the considered O-FH delay range.
- In the case of a variable delay (whether uniformly or saw-toothed distributed), NE3 introduces a delay that varies along the whole range 41 us range of delay introduced by the O-FH in each of the test scenarios.

3.2.1.3.2.1.3 Extra-long O-FH scenarios

Finally, the third set of scenarios is one in which the assumed O-FH delayed is increased beyond that of the scenarios presented in Table 17, which were based on real deployment cases in a real operator network. These scenarios do not correspond to any centralization setup and, therefore, we do not associate any centralization distance to them. They are synthetic scenarios merely meant to investigate how much can the delay introduced at the O-FH be stretched before losing all connectivity between O-DU and O-RU, and their characteristics are displayed in Table 18. As it can be seen, these scenarios are defined by a constant O-FH delay ranging from 0.5 to 1 ms.

TABLE 18. DESCRIPTION OF TEST SCENARIOS WITH EXTRA LONG O-FH

Test Scenario	Centralization Distance	Minimum O-FH delay	Maximum O-FH delay
XLONG-500	N/A	500 us	500 us
XLONG-600	N/A	600 us	600 us
XLONG-700	N/A	700 us	700 us
XLONG-800	N/A	800 us	800 us
XLONG-900	N/A	900 us	900 us
XLONG-1000	N/A	1000 us	1000 us

3.2.1.3.2.2 Delay management parameter settings

In order to adapt to the different O-FH test scenarios described above, in which packet exchanges between O-RU and O-DU are subject to different delays depending on the scenario considered, the O-DU needs to adapt the transmission (TX) and reception (RX) windows of the packets it will transmit / receive over the O-FH. The O-DU TX and RX windows are determined by its delay management parameters, mainly the parameters coined as *T1a* and *Ta4*. In our investigation, we consider two types of setting:

- A baseline setting, inspired by the setting recommended by the O-RAN Fronthaul Inter-Operability Test (IOT) Profile 1, described in [70].
- A modification of this setting that is designed to adapt to longer O-FH delays.

We present both of these settings next.

3.2.1.3.2.2.1 IOT Profile 1 Delay Management Parameters

The delay management parameters for this case are described in Table 19.

TABLE 19. O-FH DELAY MANAGEMENT PARAMETERS IN O-RAN IOT PROFILE 1

Parameter	Value (us)
FH Parameters	
T12_max, T34_max	160
T12_min, T34_min	0
RuSIM Parameters	

Tcp_adv_dl	125
T2a_max_cp_dl	500 (=T2a_max_up + Tcp_adv_dl)
T2a_min_cp_dl	259 (=T2a_min_up + Tcp_adv_dl)
T2a_max_cp_ul	336
T2a_min_cp_ul	125
T2a_max_up	375 (>=345)
T2a_min_up	134 (<=134)
Ta3_max_up	171
Ta3_min_up	50
SRS O-DU Parameters (usec)	
T1a_max_cp_dl	500
T1a_min_cp_dl	419
T1a_max_cp_ul	336
T1a_min_cp_ul	285
T1a_max_up	375
T1a_min_up	294
Ta4_max_up	331
Ta4_min_up	50

3.2.1.3.2.2.2 Scenario-adjusted

When considering the case of O-DU pooling at centralized locations, it is important to adjust the O-DU TX and RX window parameters in order to account for the extra delays introduced by the O-FH in such situation, so that the packets transmitted by the O-DU arrive within the O-RU RX window, and the O-RU packet transmissions arrive within the O-DU RX window. To do so, we adjust the O-DU parameters based on the maximum (**max_FH_delay**) and minimum (**min_FH_delay**) O-FH delays reported in Table 17 and Table 18, as detailed in Table 20. In Table 21, we show comprehensively the value of all delay management parameters resulting from applying the adjustment to the O-DU pooling test scenarios.

TABLE 20. SCENARIO-ADJUSTED O-FH DELAY MANAGEMENT PARAMETERS

Item	Value (us)
FH Parameters	
T12_max, T34_max	**max_FH_delay**
T12_min, T34_min	**min_FH_delay**
RuSIM Parameters	
Tcp_adv_dl	125
T2a_max_cp_dl	500 (=T2a_max_up + Tcp_adv_dl)
T2a_min_cp_dl	259 (=T2a_min_up + Tcp_adv_dl)
T2a_max_cp_ul	336
T2a_min_cp_ul	125
T2a_max_up	375 (>=345)
T2a_min_up	134 (<=134)
Ta3_max_up	171
Ta3_min_up	50
SRS O-DU Parameters (usec)	
T1a_max_cp_dl	375 + **min_FH_delay** + 125 (= T1a_max_up + Tcp_adv_dl = T2a_max_cp_dl + **min_FH_delay**)
T1a_min_cp_dl	134 + **max_FH_delay** + 125 (= T1a_min_up + Tcp_adv_dl = = T2a_min_cp_dl + **max_FH_delay**)
T1a_max_cp_ul	336 + **min_FH_delay** (= T2a_max_cp_ul + T12_min)
T1a_min_cp_ul	125 + **max_FH_delay** (= T2a_min_cp_ul + T12_max)
T1a_max_up	375+ **min_FH_delay** (= T2a_max_up + T12_min)
T1a_min_up	134 + **max_FH_delay** (= T2a_min_up + T12_max)
Ta4_max_up	171 + **max_FH_delay** (= Ta3_max_up + T34_max)
Ta4_min_up	50 + **min_FH_delay** (= Ta3_min_up + T34_min)

TABLE 21. SCENARIO-ADJUSTED DELAY MANAGEMENT PARAMETERS FOR THE O-DU POOLING TEST SCENARIOS

Test Scenario	HL3	HL3	HL3	HL3	HL3	HL3	HL4	HL4	HL4	HL4	HL4	HL4	HL4	HL4	HL4	HL3	HL3	HL3
	Longest	Longest	Longest	Shortest	Shortest	Shortest	Shortest	Shortest	Shortest	Average	Average	Average	Longest	Longest	Longest	Average	Average	Average
	max	mid	min	max	mid	min	max	mid	min	max	mid	min	max	mid	min	max	mid	min
NE3 Delay	387	366	346	285	264	244	139	118	98	164	143	123	175	154	134	316	295	275
max delay	387	387	387	285	285	285	139	139	139	164	164	164	175	175	175	316	316	316
min delay	346	346	346	244	244	244	98	98	98	123	123	123	134	134	134	275	275	275
t1a_max_cp_dl	846	846	846	744	744	744	598	598	598	623	623	623	634	634	634	775	775	775
t1a_min_cp_dl	646	646	646	544	544	544	398	398	398	423	423	423	434	434	434	575	575	575
t1a_max_cp_ul	682	682	682	580	580	580	434	434	434	459	459	459	470	470	470	611	611	611
t1a_min_cp_ul	512	512	512	410	410	410	264	264	264	289	289	289	300	300	300	441	441	441
t1a_max_up	721	721	721	619	619	619	473	473	473	498	498	498	509	509	509	650	650	650
ta1_min_up	521	521	521	419	419	419	273	273	273	298	298	298	309	309	309	450	450	450
ta4_max	558	558	558	456	456	456	310	310	310	335	335	335	346	346	346	487	487	487
ta4_min	396	396	396	294	294	294	148	148	148	173	173	173	184	184	184	325	325	325
t2a_max_cp_dl	500	500	500	500	500	500	500	500	500	500	500	500	500	500	500	500	500	500
t2a_min_cp_dl	259	259	259	259	259	259	259	259	259	259	259	259	259	259	259	259	259	259
t2a_max_cp_ul	336	336	336	336	336	336	336	336	336	336	336	336	336	336	336	336	336	336
t2a_min_cp_ul	125	125	125	125	125	125	125	125	125	125	125	125	125	125	125	125	125	125
t2a_max_up	375	375	375	375	375	375	375	375	375	375	375	375	375	375	375	375	375	375
t2a_min_up	134	134	134	134	134	134	134	134	134	134	134	134	134	134	134	134	134	134
ta3_max_up	171	171	171	171	171	171	171	171	171	171	171	171	171	171	171	171	171	171
ta3_min_up	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50
Tcp_adv_dl	125	125	125	125	125	125	125	125	125	125	125	125	125	125	125	125	125	125

3.2.1.3.2.3 Other testbed configuration parameters

We report next other miscellaneous settings used in the testbed.

3.2.1.3.2.4 Air-interface and O-FH U-Plane Compression Parameters

We start by parameters to be configured in RuSIM, the O-RU emulation. RuSIM emulates the transmission of DL packets by the O-RU over the air interface, as well as the transmission of UE packets to the O-RU in the UL. Hence, the assumed parameters for the air interface emulation need to be determined. In addition to this, different compression options for the packets that the O-RU transmits and receives over the O-FH are possible, as these need to be set in agreement with the O-DU. Both types of parameters, along with the settings we have used, are reported in Table 22.

TABLE 22. AIR-INTERFACE EMULATION AND O-FH COMPRESSION PARAMETERS

Category	Item	Value
General	Radio access technology	NR TDD
	TDD configuration	DDDDDDDSUU
	Nominal sub-carrier spacing	30 kHz
	Total channel bandwidth	100 MHz
	Number of spatial/antenna streams	4
	Fronthaul Ethernet Link	10 Gbps x 1 lane
	RU category	Category A
IQ Compression	U-Plane data compression method	Block floating point
	U-Plane data IQ bitwidth	9
	IQ data frame format not including udCompHdr field	TRUE

3.2.1.3.2.4.1 Traffic characteristics

Traffic is based on Keysight's UDG protocol. This protocol is natively integrated within both RuSIM and CoreSIM, and allows the user to precisely control UL and DL traffic independently. It is possible to have simultaneous UL+DL traffic, or just one of them. If only mono-directional traffic is performed, it is important to know that control messages in the other direction will still be present, thus making traffic in the other direction non-zero.

Two types of traffic definitions have been used throughout all experiments:

1. Default traffic burst.
2. Traffic bursts for average E2E latency estimation.

As shown in Figure 109, the *Default scenario* comprises a single flow type, a single UE with static mobility, performing a set of actions defined in two subsequent sessions. Static mobility allows us to control parameters such as position, speed, frequency offset, CQI, AWGN, RSRP, RI, etc. In fact, UEs are emulated within RuSIM, thus enabling full control of the radio conditions between the UEs and the O-RU. Importantly, given that our setup comprises 4 spatial streams, we set the Rank Indicator (RI) equal to 4, allowing us to obtain the maximum throughput through MIMO multiplexing in DL.

Two sessions have been created: a first session related to registration and PDU session establishment and a second one related to the actual data exchange. This allows us to logically separate the UE connection setup from the actual data exchange. In case multiple UEs are part of the scenario, e.g., in the *Average E2E latency scenario*, the second session does not start until the first one is completed. This allows having all UEs correctly attached and with a PDU Session already established and, thus, starting the data bursts all simultaneously.

In the second session, a set of increasing DL-only traffic is generated from CoreSIM towards the UE. Traffic bursts last 60 seconds, while pauses between bursts last 5 seconds. A longer 20 second pause is performed before starting the UL-only traffic, i.e., traffic created from the RuSIM's UE going towards CoreSIM. Pauses between bursts allow post-processing scripts to easily detect bursts, greatly simplifying the data analysis. The traffic bursts created in this session are illustrated in Figure 110.

Finally, the UE releases the PDU session and deregisters from the network.

Test Scenario: Subscribers: (1), Groups: (1)	
Flows (1)	
Group: FTP Cell1	
Static 5GNr Mobility (Cells: 1)	
Subscriber List (1)	
Session List (2)	
Attach	
0	Registration
1	PDU Session Establish
Test gnb1	
0	Delay
1	Delay
2	UDG Monodirectional Transmission DL
3	Delay
4	UDG Monodirectional Transmission DL
5	Delay
6	UDG Monodirectional Transmission DL
7	Delay
8	UDG Monodirectional Transmission DL
9	Delay
10	UDG Monodirectional Transmission DL
11	Delay
12	UDG Monodirectional Transmission DL
13	Delay
14	UDG Monodirectional Transmission DL
15	Delay
16	UDG Monodirectional Transmission DL
17	Delay
18	UDG Monodirectional Transmission DL
19	Delay
20	UDG Monodirectional Transmission DL
21	Delay
22	Delay
23	UDG Monodirectional Transmission UL
24	Delay
25	UDG Monodirectional Transmission UL
26	Delay
27	UDG Monodirectional Transmission UL
28	Delay
29	UDG Monodirectional Transmission UL
30	Delay
31	UDG Monodirectional Transmission UL
32	Delay
33	UDG Monodirectional Transmission UL
34	Delay
35	UDG Monodirectional Transmission UL
36	Delay
37	PDU Session Release
38	Deregistration

FIGURE 109. DEFAULT TRAFFIC SCENARIO AS CONFIGURED IN THE UDG TRAFFIC GENERATOR

The *Average E2E latency scenario* resembles the default one just described. To more accurately extract E2E latencies, though, a single burst 5-minutes burst is performed during the second session, allowing us to capture more latency data. Furthermore, we found that the number of UEs also affects some results, and this variable is also added to the scenario. When multiple UEs are connected, traffic is considered to be aggregate on all UEs, conversely, each UE experiences $\frac{1}{\#UEs}$ of the total traffic.

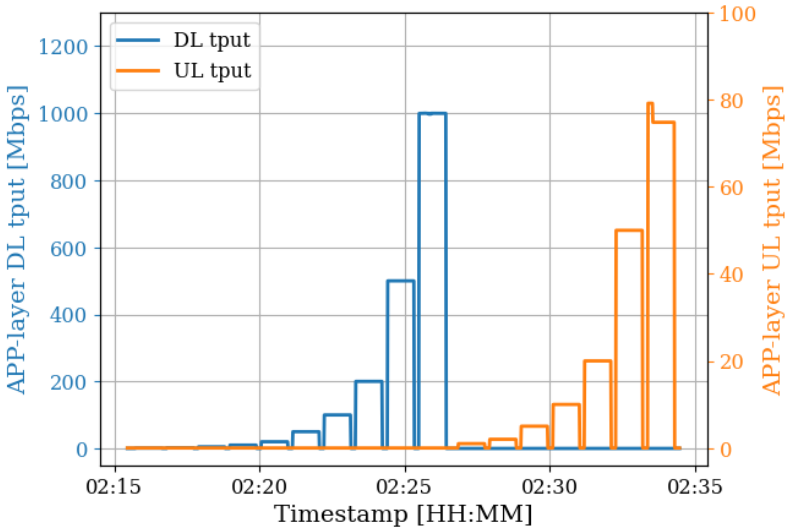


FIGURE 110. ILLUSTRATION OF THE DL (BLUE) AND UL (ORANGE) TRAFFIC BURSTS USED IN THE DEFAULT TRAFFIC SCENARIO

3.2.1.3.2.4.2 UL SNR and DL BLER

RuSIM allows emulating some impairments over the wireless channel. Specifically, it allows adding AWGN noise to UL signals, and BLER (separately or each MCS) to DL signals. Specifically, these settings can be controlled by the mobility model. For simplicity, within this project we only used the *Static 5GNr Mobility*, meaning that a configuration is maintained constant throughout the entire scenario duration.

UL AWGN noise has to be enabled first, then it can be set relative to the UL signal power in dB units. This means that $SNR_{dB} = -AWGN_{dB}$, e.g., $AWGN = -5\ dB \rightarrow SNR = 5\ dB$.

DL BLER, instead, can be set independently for each MCS. This means that, for example, the user can set DL BLER for MCS1=0, for MCS2=0.0001, for MCS3=0.0002, etc.

3.2.1.3.2.4.3 O-DU/O-CU: max_proc_delay (MPD)

One additional srsRAN O-DU parameter used within this project is `expert_phy:max_proc_delay` (MPD). As explained in Sections 3.2.1.2.1.5, ..his parameter controls the maximum allowed DL

processing delay (in number of slots), which defaults to 5 slots (or 2.5 ms considering the 30 kHz SCS used). The idea is that longer MPD trades higher E2E latency to allow longer processing latency, possibly due to an exceptionally long O-FH or to underperforming hardware.

3.2.1.3.2.4.4 SUT load

As the O-DU needs to perform heavy signal processing tasks in real time and respecting the tight timing constraints imposed by the O-FH timing management model, it is expected that the transmission of packets over the O-FH may be affected by the computational load of the server running the O-DU. To test this, we run some tests in which the server running the O-DU is loaded with some extra computing tasks, and we compare the results with those when the server is exclusively running the O-DU and O-CU tasks.

3.2.1.3.3 Results

We now begin reporting the results obtained in the testbed. We will focus mostly on the evaluation of the O-DU pooling scenarios defined in Table 17, and how the performance of the system when using the scenario-adjusted delay management parameters compares with respect to when using the baseline parameters.

3.2.1.3.3.1 Centralization of O-DU/CUs

We start by considering the scenarios of Table 17. First, we inspect the statistics collected by the O-DU and the O-RU in terms of O-FH packets that arrive on-time, early, or late at each of the ends. Then, we focus on the throughput and block-error rate results achieved. Throughout this investigation, we assume an ideal air-interface situation, in which no packet errors exist in the over-the-air transmission. This allows us to isolate the effect of the O-FH in the end-to-end system performance, without this analysis being obscured by the inevitable performance degradation that will be introduced by wireless channels.

3.2.1.3.3.1.1 Ideal air-interface

3.2.1.3.3.1.1.1 Packet timing with different scenarios

Next, we depict the rate of packets that arrive early, on-time, and late in each of the scenarios. For each scenario, two tests are run: one using the baseline delay management parameters in Table 19 (labelled as *IOT profile 1* in the legend of the figures), and another test using the adjusted parameters of Table 20 (labelled as *Adjusted* in the legend of the figures). In both tests, NE3 introduces two types of delay respectively: either a constant delay or a random delay uniformly distributed within ±20 µs around the value in the corresponding scenario.

In all subsequent figures, a green vertical line at 160 µs indicates the timing threshold of the baseline configuration: under an O-FH which incurs delays smaller than 160 µs (i.e., to the left of the green line), all O-FH packets transmitted by O-DU and O-RU in their respective TWs should be received within the RW of the counterpart.

3.2.1.3.3.1.1.1 UL User Plane

For UL user plane traffic, both scenarios exhibit similar behaviour when the O-FH delay is below 160 µs (left of the green line). However, when the O-FH delay exceeds 160 µs, a noticeable divergence emerges. Specifically, in the baseline configuration (blue lines), the late packet ratio grows significantly, whereas it remains stable under the scenarios with adjusted parameters. Correspondingly, on-time packet ratio in the baseline configuration shows a markable decrease, indicating a strong sensitivity to delay. As for the early packet, the ratio remains consistently low across all tests, as expected in theory analysis.

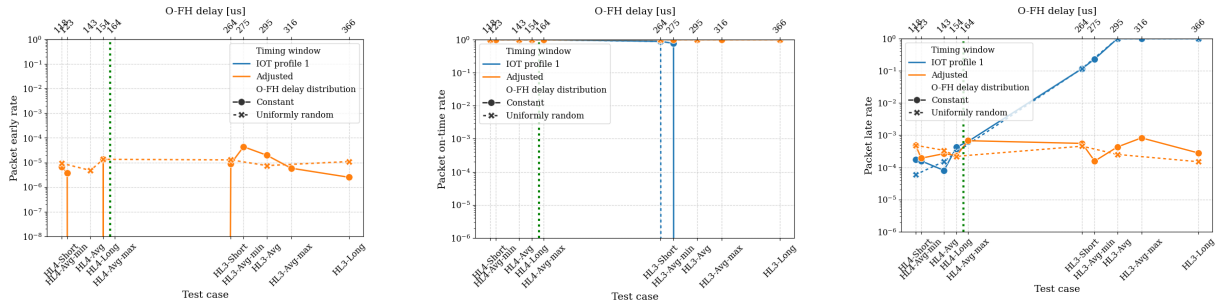


FIGURE 111. RATE OF EARLY (LEFT), ON-TIME (MIDDLE) AND LATE (RIGHT) PACKETS OBSERVED IN THE UL IN USER-PLANE TRAFFIC

3.2.1.3.3.1.1.1.2 UL Control Plane

The timing evaluation of UL control plane packets, conducted at the O-RU, presents results consistent with those observed in the user plane: a significant increase happens in the late packet ratio when the O-FH delay grows in baseline scenarios. In contrast, this is not observed in the scenarios using adjusted parameters, where the late packet ratio remains 0. This behaviour also leads to the different trends for the on-time packet ratio: in baseline scenarios, it decreases sharply once the O-FH delay is larger than 160 µs, while remaining stable in the adjusted scenarios. As for early packets, all tests report a value of 0, aligning with the previous observations.

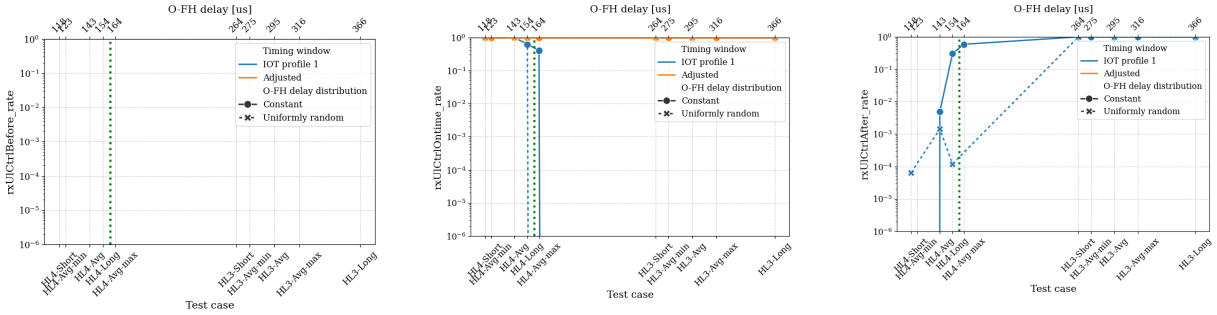


FIGURE 112. RATE OF EARLY (LEFT), ON-TIME (MIDDLE) AND LATE (RIGHT) PACKETS OBSERVED IN THE UL IN CONTROL-PLANE TRAFFIC

3.2.1.3.3.1.1.1.3 DL User Plane

The O-RU also collects packet timing statistics for DL traffic. Specifically, the user plane traffic on DL shows a markable increase in the late packet ratio in baseline scenarios as the O-FH delay increases, while this ratio remains at a low and stable level in tests conducted with adjusted timing window parameters.

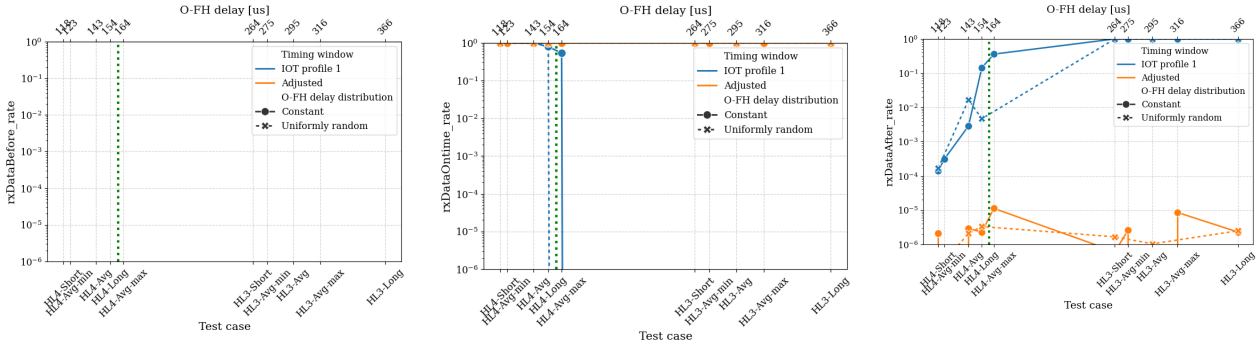


FIGURE 113. RATE OF EARLY (LEFT), ON-TIME (MIDDLE) AND LATE (RIGHT) PACKETS OBSERVED IN THE DL IN USER-PLANE TRAFFIC

3.2.1.3.3.1.1.1.4 DL Control Plane

Similarly, the packet timing analysis for DL control plane traffic reveals trends consistent with those observed in other traffic types. In baseline scenarios, increasing O-FH delay results in a significant impact on the late and on-time packet ratios, while this has minimum effect in scenarios using adjusted parameters.

The above observations clearly demonstrate the strong influence of O-FH delay on packet timing. However, by properly adjusting the timing window parameters, packet timing violation can be substantially reduced even in long O-FH delay cases.

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA
cttc Centre Tecnològic de Telecomunicacions de Catalunya

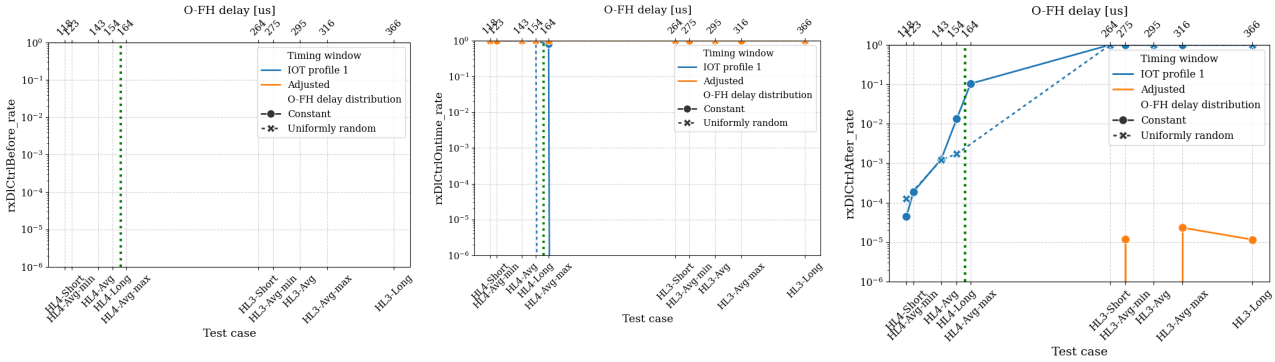


FIGURE 114. RATE OF EARLY (LEFT), ON-TIME (MIDDLE) AND LATE (RIGHT) PACKETS OBSERVED IN THE DL IN CONTROL-PLANE TRAFFIC

3.2.1.3.3.1.1.2 Throughput and BLER results

Next, we report the throughput and BLER achieved in either UL (PUSCH) or DL (PDSCH), depicted in Figure 115 and Figure 116. These figures present consistent results with the previous packet timing analysis. The results indicate that the test cases with O-DU aggregation at the HL4 level can all be accomplished without any noticeable performance degradation with the default TW and RW settings. For the test cases with O-DU aggregation at the HL3 level, though, only the two cases with shorter O-FH delay provide decent performance, even if a small degradation can be observed in the downlink throughput. For the rest of HL3 test cases, which imply longer delays in the OFH, communication between the O-RU and the O-DU along the O-FH interface is not possible with the default TW/RW settings, and no throughput is obtained. When, on the other hand, the TW/RW boundaries are adjusted as described before, we observe that the throughput obtained in all test cases hovers around the offered traffic, and no degradation of the performance can be observed with increasing centralization distance. Remarkably, packet delay variation across the O-FH results in no degradation of the performance. These results confirm that, with careful adjustment of the TW/RW parameters at the O-DU, O-FH of significantly larger length can be deployed with no E2E performance penalty.

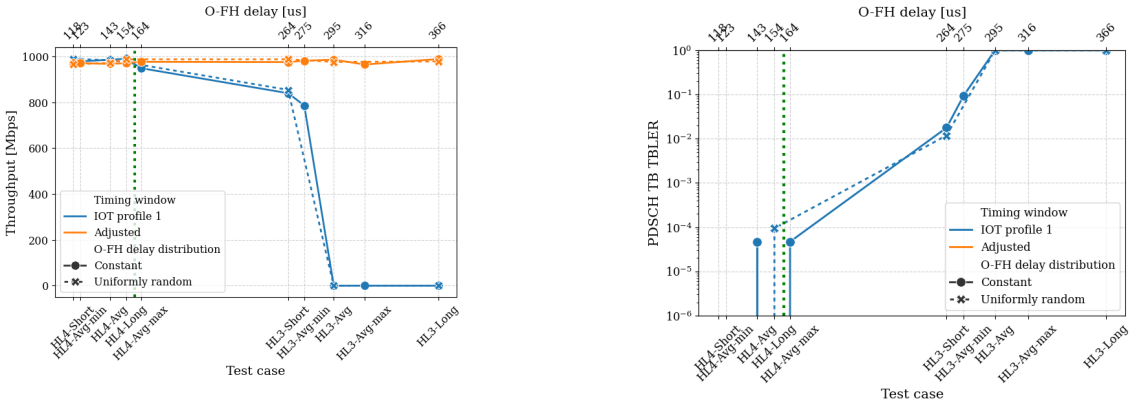


FIGURE 115. THROUGHPUT (LEFT) AND BLER (RIGHT) OBSERVED IN THE DOWNLINK (PDSCH)

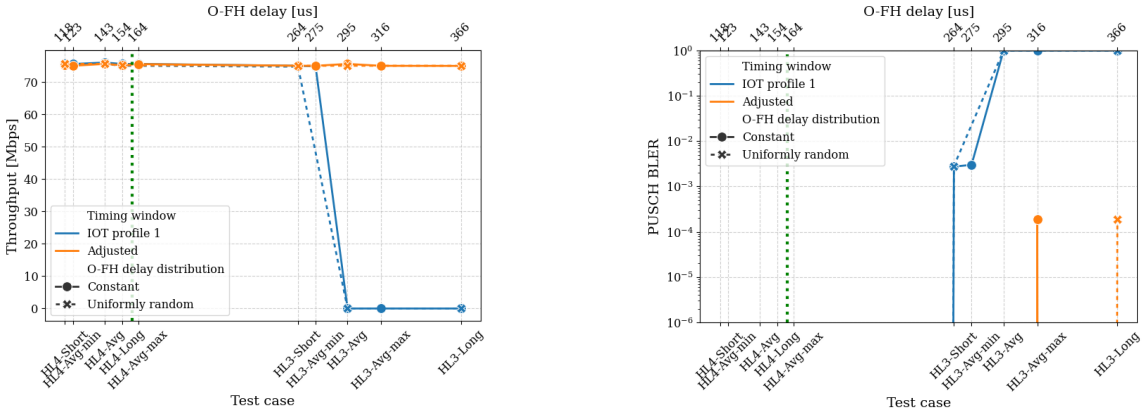


FIGURE 116. THROUGHPUT (LEFT) AND BLER (RIGHT) OBSERVED IN THE UPLINK (PUSCH)

3.2.1.3.3.1.2 Realistic SNR + Multiple UEs

In the following results, the assumption of an idealistic air-interface is dropped, and results assuming and UL SNR of 0 dB are reported.

Under this condition, the results in Figure 117 indicate that the UL MCS is reduced to 4, and PUSCH throughput can no longer reach the configured value of 78 Mbps, but is instead limited to approximately 8.3 Mbps in all O-FH timing configurations. Notably, the UL BLER remains at a relatively stable value of 0.01 across all tests. This may be attributed to the O-DU configuration parameter *olla_target_bler*, which defines the target BLER in Outer-Loop Link Adaptation (OLLA) algorithm.

Although noise is only applied on the UL channel, we notice in Figure 118 that the DL MCS and throughput are also heavily affected, despite the DL BLER remaining at 0. Specifically, the MCS

reduces to 20, while PDSCH throughput decreases from 1 Gbps (under ideal air interface conditions) to approximately 380 Mbps.

Besides, all these figures demonstrate that the impact of introducing a realistic SNR on the UL channel is consistent across different O-FH timing configurations, as both throughput and BLER remain unchanged in tests with different O-FH delays and timing window settings.

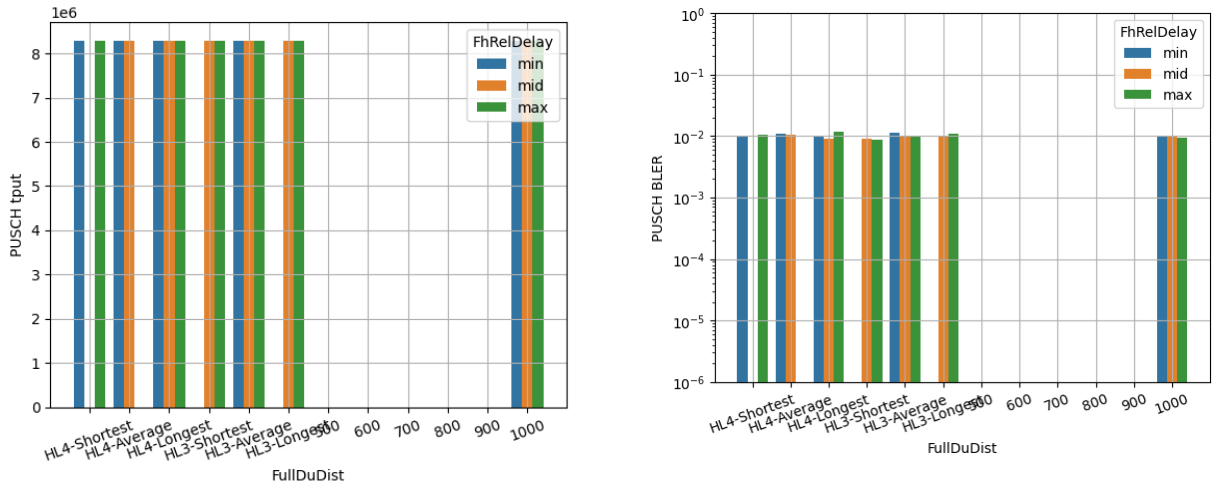


FIGURE 117. THROUGHPUT (LEFT) AND BLER (RIGHT) OBSERVED IN THE UPLINK WITH UL SNR SET TO 0 DB

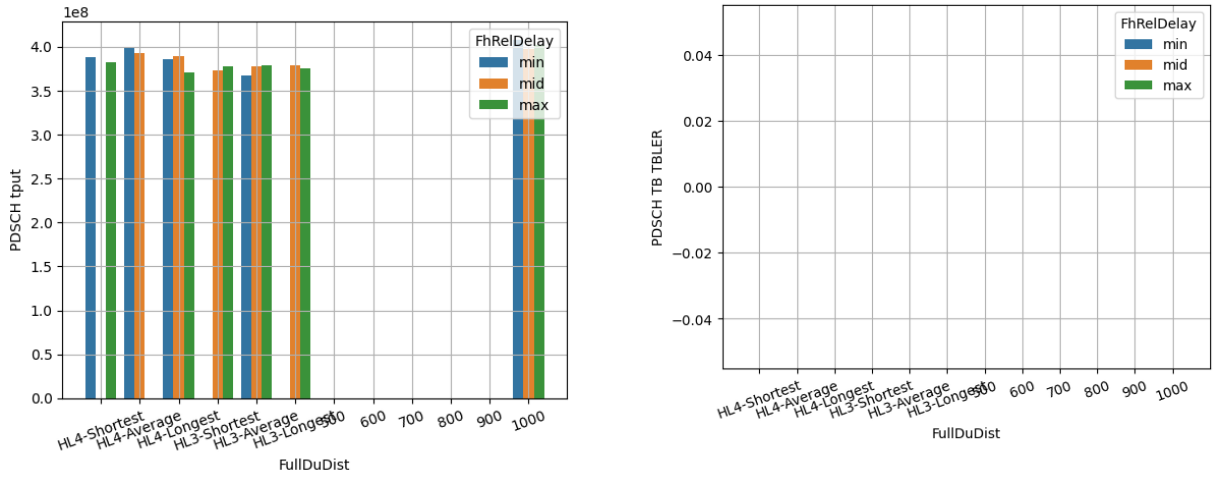


FIGURE 118. THROUGHPUT (LEFT) AND BLER (RIGHT) OBSERVED IN THE DOWNLINK WITH UL SNR SET TO 0 DB

```
          |--------------------DL---------------------|-------------------------UL-------------------------------
pci rnti | cqi  ri  mcs  brate   ok  nok  (%)  dl_bs | pusch  rsrp  mcs  brate   ok  nok  (%)    bsr     ta  phr
  1 4606 |  15 4.0   20   366M  871  679  43%  4.86M |   0.1 -61.2    4   102k   25    0   0%      0      0   38
  1 4606 |  15 4.0   20   393M  934  616  39%  4.78M |   0.0 -61.2    4   106k   26    0   0%      0      0   38
  1 4606 |  15 4.0   20   348M  829  721  46%  4.86M |   0.1 -61.2    4   102k   25    0   0%      0      0   38
  1 4606 |  15 4.0   20   375M  892  658  42%  4.48M |   0.1 -61.2    4   102k   25    0   0%      0      0   38
  1 4606 |  15 4.0   20   367M  889  661  42%  5.16M |   0.1 -61.2    4   102k   25    0   0%      0      0   38
  1 4606 |  15 4.0   20   332M  790  760  49%  5.16M |   0.1 -61.2    4   102k   25    0   0%      0      0   38
  1 4606 |  15 4.0   20   431M 1025  525  33%  5.16M |  -0.0 -61.2    4   106k   26    1   3%      0      0   38
  1 4606 |  15 4.0   20   371M  885  665  42%  4.48M |   0.0 -61.2    4   102k   25    1   3%      0      0   38
  1 4606 |  15 4.0   20   345M  822  728  46%  5.16M |   0.1 -61.2    4   102k   25    0   0%      0      0   38
  1 4606 |  15 4.0   20   336M  801  749  48%  4.78M |   0.1 -61.2    4   102k   25    0   0%      0      0   38
  1 4606 |  15 4.0   20   114M  275  189  40%     0  |  -0.0 -61.2    4    41k   10    0   0%      0      0   38

          |--------------------DL---------------------|-------------------------UL-------------------------------
pci rnti | cqi  ri  mcs  brate   ok  nok  (%)  dl_bs | pusch  rsrp  mcs  brate   ok  nok  (%)    bsr     ta  phr
  1 4606 |  15 4.0   20    61k   37    0   0%      0  |   0.2 -61.2    4   4.0M  198    0   0%    300k      0   38
  1 4606 |  15 4.0   20   121k   75    0   0%      0  |   0.2 -61.2    4   8.2M  396    4   1%    300k      0   38
  1 4606 |  15 4.0   20   119k   74    0   0%      0  |   0.2 -61.2    4   8.2M  396    4   1%    300k      0   38
  1 4606 |  15 4.0   20   119k   74    0   0%      0  |   0.2 -61.2    4   8.2M  396    4   1%    300k      0   38
  1 4606 |  15 4.0   20   121k   74    0   0%      0  |   0.2 -61.2    4   8.2M  396    4   1%    300k      0   38
  1 4606 |  15 4.0   20   119k   74    0   0%      0  |   0.2 -61.2    4   8.2M  396    4   1%    300k      0   38
  1 4606 |  15 4.0   20   126k   74    0   0%      0  |   0.2 -61.2    4   8.2M  396    4   1%    300k      0   38
  1 4606 |  15 4.0   21   128k   74    0   0%      0  |   0.2 -61.2    4   8.2M  396    4   1%    300k      0   38
  1 4606 |  15 4.0   21   130k   75    0   0%      0  |   0.2 -61.2    4   8.2M  396    4   1%    300k      0   38
  1 4606 |  15 4.0   21   128k   74    0   0%      0  |   0.2 -61.2    4   8.2M  396    4   1%    300k      0   38
  1 4606 |  15 4.0   21   131k   74    0   0%      0  |   0.2 -61.2    4   8.2M  396    4   1%    300k      0   38
```

FIGURE 119. SNAPSHOT OF PUSCH AND PDSCH METRICS PROVIDED BY SRSRAN O-DU

The results in Figure 120 illustrate how different levels of UL air-interface noise influence network KPIs under different O-FH configurations. We can observe a strong negative correlation between PHY-layer throughput and noise level. However, this correlation behaves differently in DL and UL. In the UL, the throughput decreases gradually as the SNR drops below 22 dB. However, the DL throughput remains nearly constant down to 4 dB SNR, and then declines sharply below this threshold.

Figure 121 also indicates that noise has no observable effect on PDSCH BLER, which remains zero across different SNR values. In contrast, the PUSCH BLER is sensitive to SNR, as it is 0 for SNR values above 20 dB, but increases to 0.01 once the SNR falls below this threshold. Note that this value is consistent with the threshold observed for the drop in PUSCH throughput.

DL one-way latency is also affected by the noise level, particularly under high throughput or noise conditions. Specifically, in Figure 122 we notice that increased noise leads to higher latency at 1 Gbps throughput, while its impact is minimal at lower throughput levels. Additionally, the DL latency increases sharply as the SNR drops from 4 to 2 dB but remains relatively constant at higher SNR values. Moreover, the figure also demonstrates that the impact of air-interface noise is independent of the O-FH configuration. No matter whether the O-FH delay is low (15 µs), medium (366 µs), or high (1000 µs), the trends are similar.

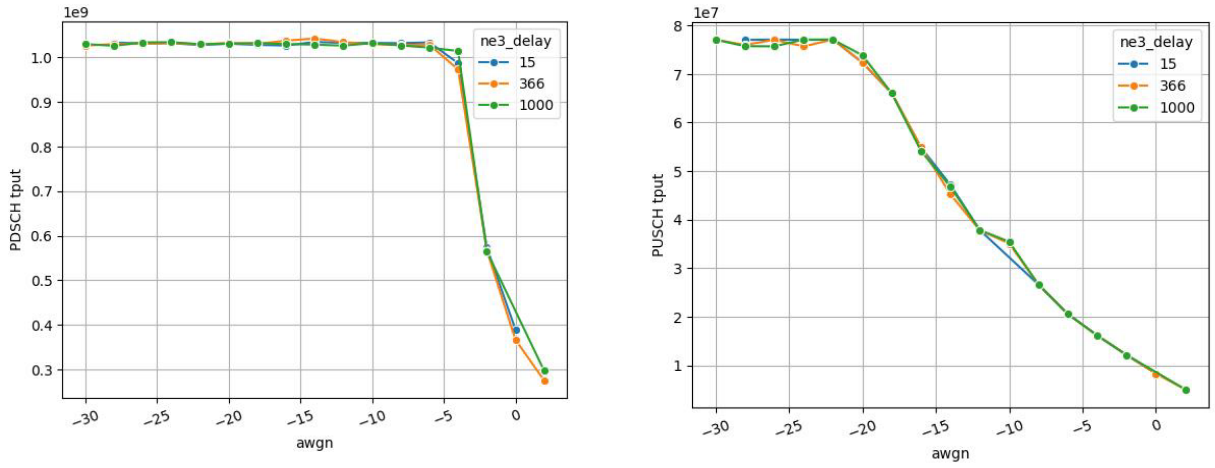


FIGURE 120. THROUGHPUT ACHIEVED AT THE PDSCH (LEFT) AND PUSCH (RIGHT) VS THE UL NOISE LEVEL

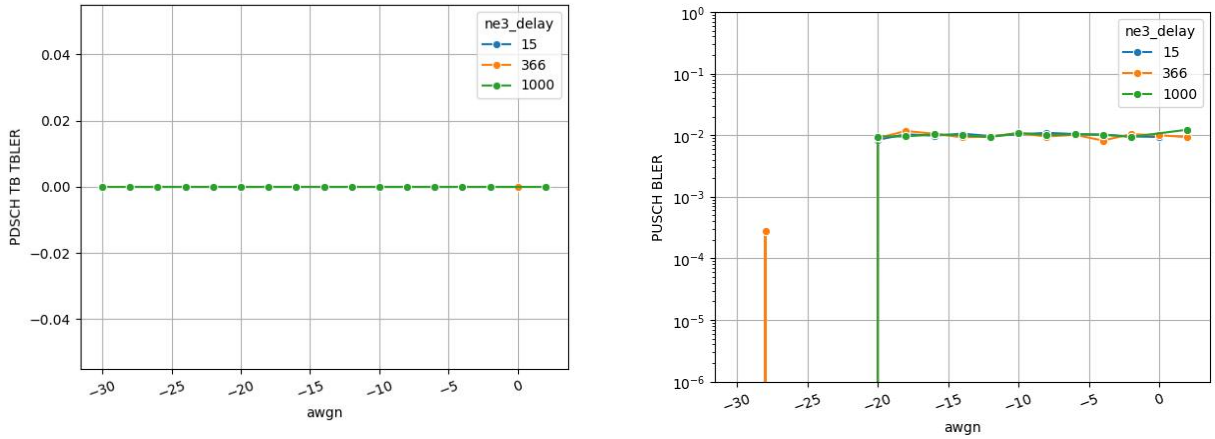


FIGURE 121. BLER ACHIEVED AT THE PDSCH (LEFT) AND PUSCH (RIGHT) VS THE UL NOISE LEVEL

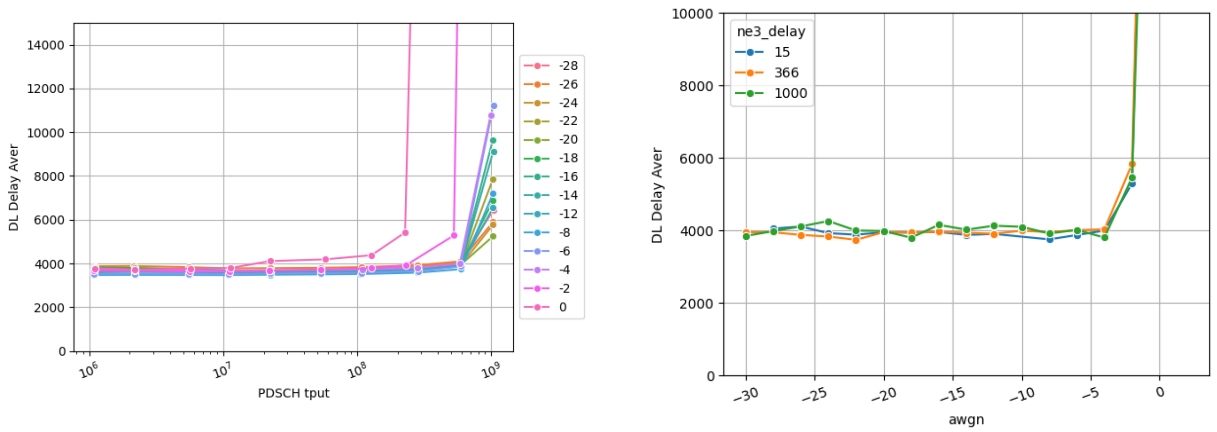


FIGURE 122. LEFT: DL LATENCY ACHIEVED AT DIFFERENT UL NOISE LEVELS WITH AN O-FH DELAY OF 15 US. RIGHT: DL LATENCY ACHIEVED WITH DIFFERENT O-FH DELAYS FOR A FIXED PDSCH THROUGHPUT OF 500 MBPS

3.2.1.3.3.2 Packet timing analysis

Within the scope of the project, some metrics collected from the DU in the uplink, suggested the DU was reporting a portion of UL packets arriving late in time compared to their expected allowed temporal window. Addressing the related investigation involved conducting repeated incremental testing to rule out any random artifact or misbehaviours and after than required in depth root cause analysis.

As any timing mismatch needs to be always observed from both potential failing sides, we tried to first check initially the traffic capturing from the sending and receiving servers. But in doing so we found that direct capture in the servers at such high rates was also impacting the accuracy/precision in the timestamping and could even incorrectly suggest lost packets because of the limitation of the software capture process itself.

Finally, we decided to use the hardware-based capture and analysis tools from Keysight Open RAN Studio and from the Network Emulator 3 hardware equipment. By doing that, and cross comparing with the timing of associated PTP messages, we were able to confidently confirm that the RU was not transmitting packets before the start of the transmission window. In Figure 123 it can be observed that in an explicit initial analysis all packets at a specific subframe and symbol captured at transmission port plane consistently have timestamps above the expected 50 microseconds. Then, a more detailed analysis in Figure 124 using the delay management analysis features in Open RAN studio provided further evidence on the absence of actual early in excess packets.

Thanks to the confidence that these set of analysis tests and tools provided, the consortium partners were able to set higher focus on investigating the uplink receiving part and identify the actual root cause for the initially offending metric.

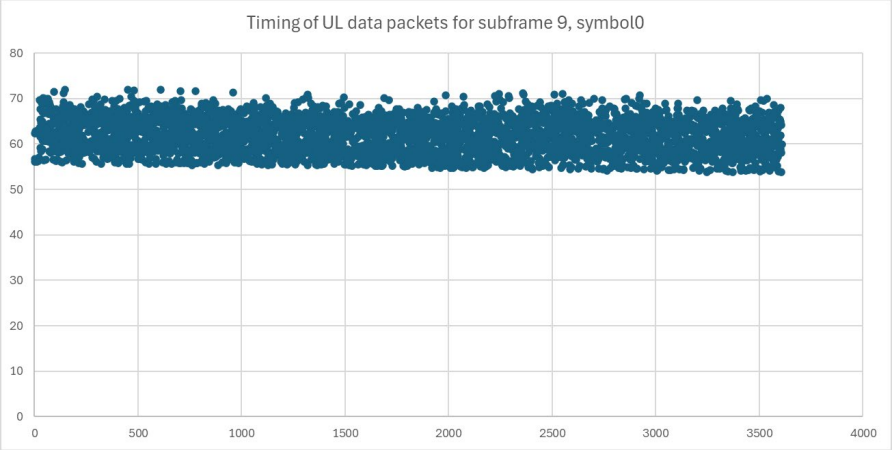


FIGURE 123. TIMING ANALYSIS ON OPEN-RAN STUDIO PCAP

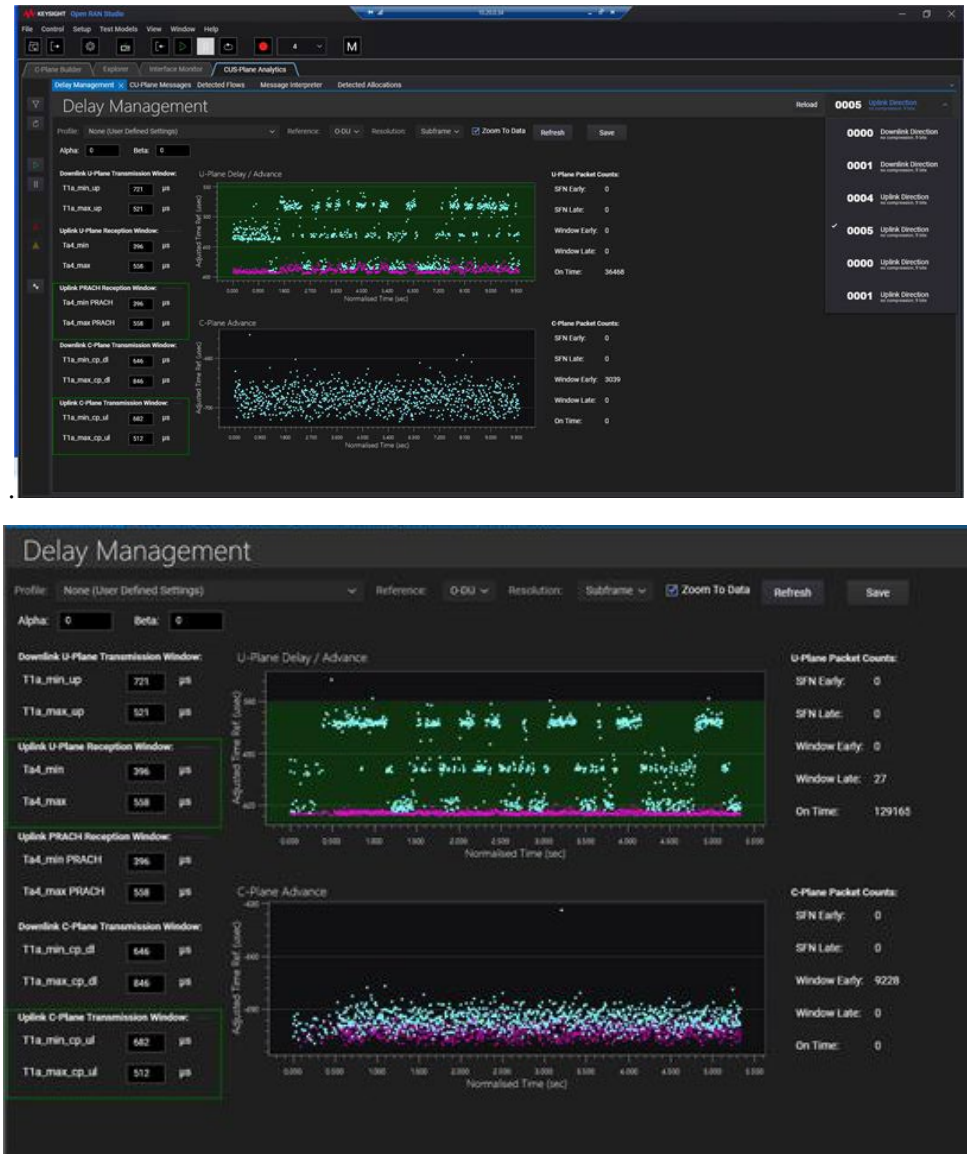


FIGURE 124. TIMING ANALYSIS ON OPEN-RAN STUDIO

3.2.1.3.3.3 E2E latency studies

Below we discuss our experiments on E2E latency evaluations.

3.2.1.3.3.3.1 Challenges in measuring latency

For DL one-way latency, RuSIM only provides histogram buckets with relatively larger intervals (e.g., [0,5 ms), [5,10 ms), etc.) or long-term averages starting from the beginning of the emulation. The

coarse granularity of these latency buckets causes most latency measurements to fall within the first or second bucket across different configurations, making it indistinguishable between different configurations. As for the long-term average, considering that we are emulating multiple bursts with different throughput values in one test, this may mask the difference over averaging. A potential way to alleviate this limitation is to calculate latency over short intervals (e.g., per second) using the available long-term average and the total number of packets. Figure 125 below presents an example of this method, which clearly reveals the impact of O-FH delay on DL latency.

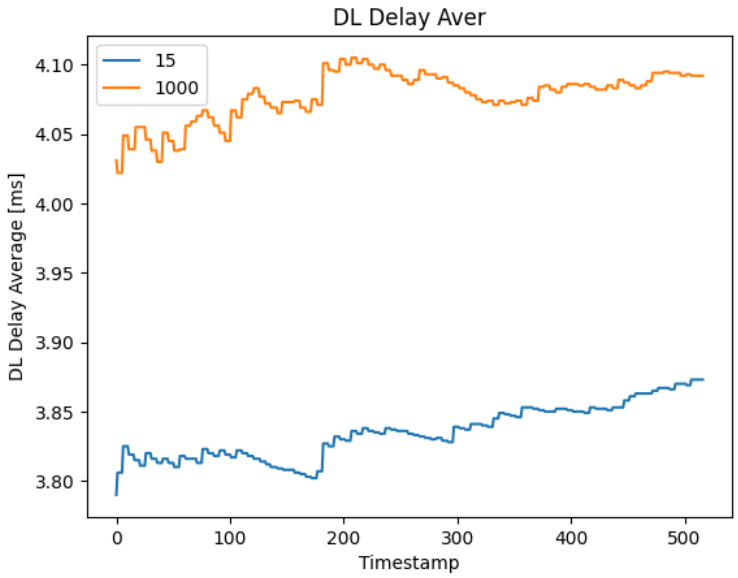


FIGURE 125. ESTIMATED INSTANT LATENCY FOR O-FH DELAYS OF 15 NS (BLUE) AND 1 US (ORANGE)

On the contrary, UL one-way latency measurement is not natively supported in RuSIM. Instead, we can estimate it through manual packet capture and postprocessing. In the figure below, we present the calculated UL latency histogram. However, this approach requires extensive manual effort and complex timing calculations, making it impractical for our large-scale testing.

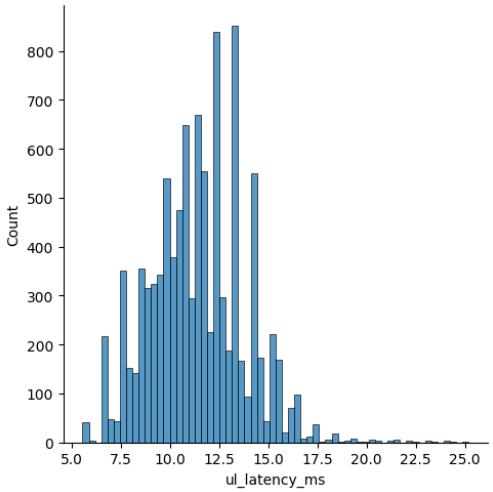


FIGURE 126. HISTOGRAM OF THE UL LATENCY THAT CAN BE OBTAINED MANUALLY VIA PACKET CAPTURES

Therefore, considering all these factors, we only consider the DL latency with a simplified test setup to ensure reliable measurement. In each test with a specific configuration, instead of running a series of bursts with varying throughput values, we conduct a single 5-minute DL throughput burst. This approach allows the long-term average to effectively reflect DL latency under this configuration.

Another important consideration is that the one-way latency measurement relies on synchronization between the UE (RuSIM) and the Core (CoreSIM). RuSIM is synchronized via PTP to the PTP switch, while CoreSIM is synchronized via NTP to the PTP switch. To assess synchronization accuracy, we conduct a one-day drift monitoring for both methods.

From Figure 127, we see that NTP synchronization between CoreSIM and the switch can exhibit drifts of up to 1 ms. In contrast, PTP synchronization for RuSIM generally maintains nanosecond-level accuracy, with only occasional spikes. This means that the one-way latency measurement may be subject to inaccuracies on the order of 1 ms due to the limitations in NTP synchronization.

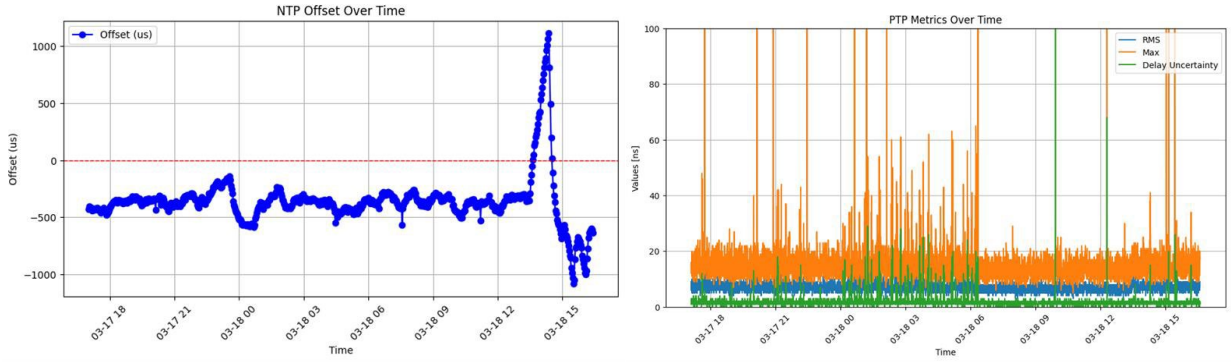


FIGURE 127. TIMING OFFSETS ACHIEVED WITH NTP (LEFT) AND PTP (RIGHT), MEASURED TO ASSESS THE ACCURACY OF THE LATENCY ESTIMATES

3.2.1.3.3.3.2 Interaction of max_proc_delay, data-rates, SNR, O-FH delay

In this section, we evaluate the effect of several scenario parameters on the end-to-end latency that is observed in DL user traffic. The parameters that we explore are described next:

- UL SNR on the air interface: we set the UL SNR to 40, 10, and 0 dB (corresponding to AWGN settings of -40, -10, and 0, respectively) in order to explore how the quality of the UEs uplink connection may impact the system's latency. Degradation on the UL air interface will lead to the loss of ACKs from the UEs, triggering unnecessary retransmissions and increasing latency as a result.
- O-FH delay: with NE3, we emulate O-FHs that introduce different delays in the transmission of packets from O-DU to O-RU and vice versa. In particular, we evaluate O-FH delays of 15, 118, 336, and 1000 us.
- Application-layer DL throughput: we also explore how the overall DL throughput measured at the application layer affects the average DL latency.
- The time allowed for the O-DU to process DL packets: srsRAN O-DU has a configurable parameter that determines the maximum time that the O-DU is allowed to spend processing a certain DL packet before it is forwarded to the O-RU. This parameter, `max_proc_delay`, determines the number of slots prior to the transmission of a packet over the air that the O-DU can start processing the packet. Increasing values of this parameter increase the E2E latency, but settings too small may lead to the O-DU dropping packets in situations of high computational load, as it will not have enough time to carry out all the necessary operations.
- The number of UEs active in the considered cell, which may impact the latency related to signalling for the different users.

The results of different tests sweeping over the above parameters are presented in Figure 128 to Figure 133, which report the average downlink latency observed under different configurations.

Without exhaustively describing each of the figures, we note the following overall observations from the obtained results:

- DL latency is degraded as the UL SNR decreases. This is likely due to packet losses in the UL control information that is related to DL traffic.
- Generally, an increasing number of UEs concurrently accessing the cell leads to an increase in latency. Similarly, a higher DL throughput increases the DL latency as well. Hence, unsurprisingly, the end-to-end latency is negatively impacted by the increasing system load.
- DL latency is increased by larger settings of the O-DU processing time, as expected. However, care has to be taken when trying to lower the value of `max_proc_delay`. Low settings for this parameter can only be successfully used whenever the O-FH does not introduce a large delay; otherwise, the O-DU needs to be provided sufficient processing time to compensate for the O-FH propagation delay. Similarly, larger number of UEs imposes tougher processing tasks for the O-DU (e.g., the scheduler operations), and thus more conservative values of the `max_proc_delay` parameter need to be used in those conditions.

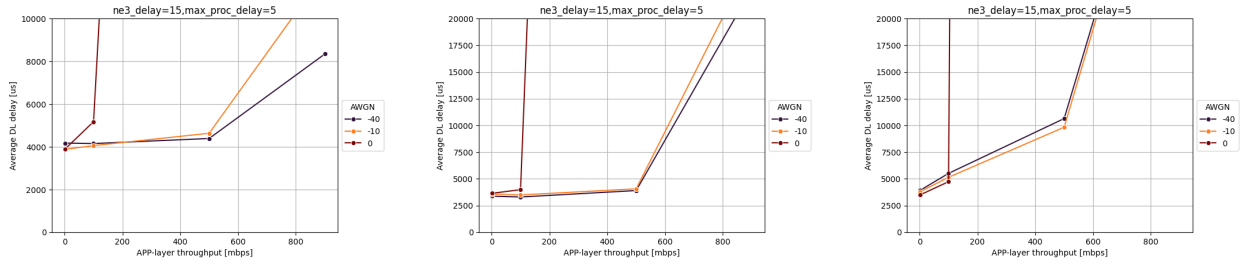


FIGURE 128. AVERAGE DOWNLINK DELAY AS A FUNCTION OF THROUGHPUT AND SNR, FOR DIFFERENT NUMBER OF UES: 1 (LEFT), 10 (CENTER), AND 50 (RIGHT)

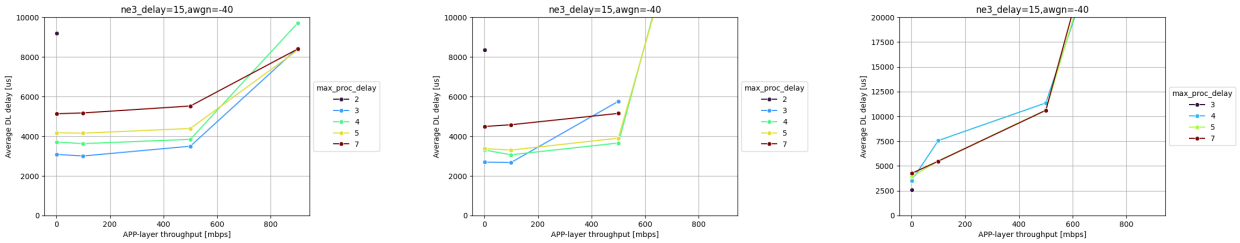


FIGURE 129. AVERAGE DOWNLINK DELAY AS A FUNCTION OF THROUGHPUT AND O-DU PROCESSING TIME, FOR DIFFERENT NUMBER OF UES: 1 (LEFT), 10 (CENTER), AND 50 (RIGHT)

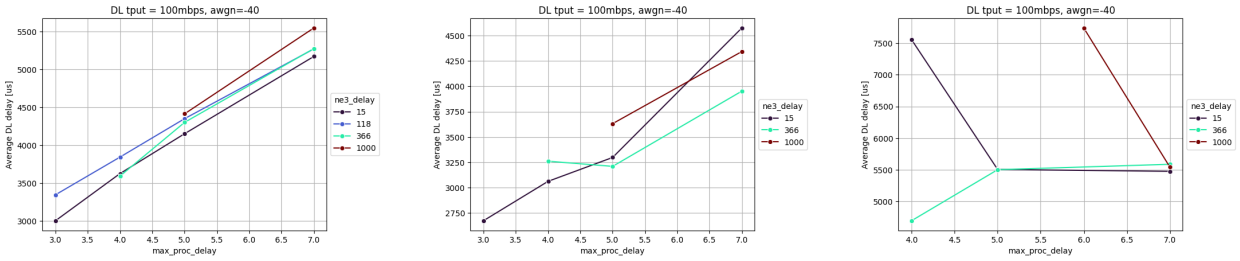


FIGURE 130. AVERAGE DOWNLINK DELAY AS A FUNCTION OF O-DU PROCESSING TIME AND O-FH DELAY, FOR DIFFERENT NUMBER OF UES: 1 (LEFT), 10 (CENTER), AND 50 (RIGHT)

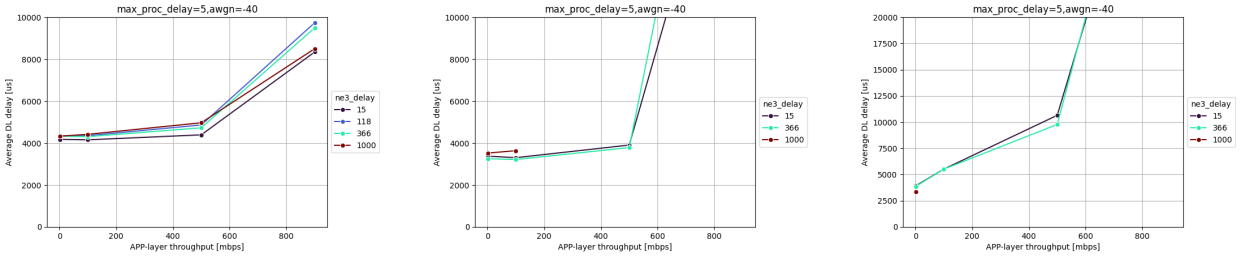


FIGURE 131. AVERAGE DOWNLINK DELAY AS A FUNCTION OF THROUGHPUT AND O-FH DELAY, FOR DIFFERENT NUMBER OF UES: 1 (LEFT), 10 (CENTER), AND 50 (RIGHT)

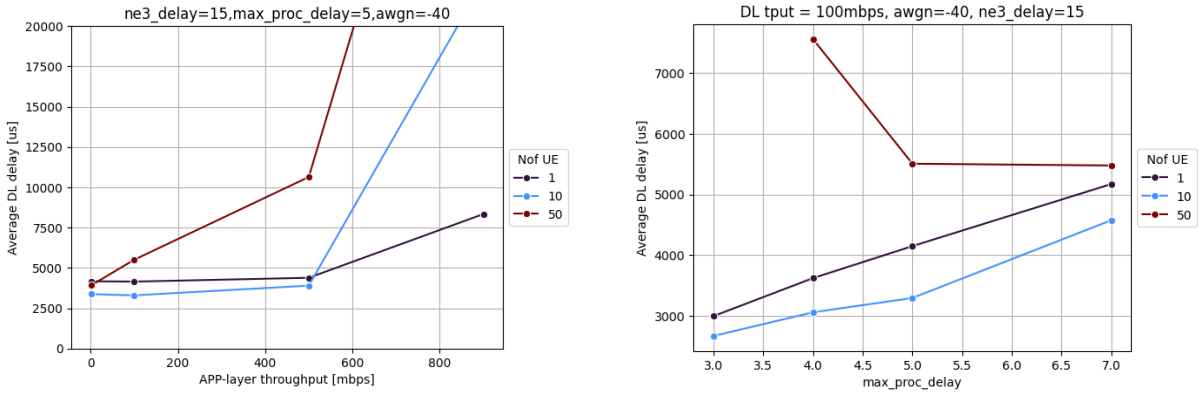


FIGURE 132. AVERAGE DOWNLINK DELAY AS A FUNCTION OF THE NUMBER OF UES AND DL THROUGHPUT (LEFT) AND O-DU PROCESSING TIME (RIGHT)

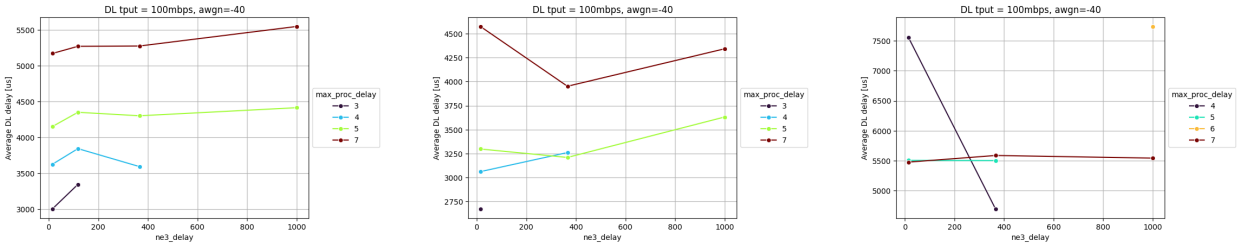


FIGURE 133. AVERAGE DOWNLINK DELAY AS A FUNCTION OF O-FH DELAY AND O-DU PROCESSING TIME, FOR DIFFERENT NUMBER OF UES: 1 (LEFT), 10 (CENTER), AND 50 (RIGHT)

3.2.1.3.3.4 Computational Performance of O-DU/CU server

During exploratory tests, we run several Linux perf tools on the server hosting the O-DU/CU to monitor performance metrics such as CPU and memory usage. However, we observe that the execution of these tools can also interfere the packet timing analysis.

3.2.1.3.3.4.1 Tests with / without perf running

The resulting figures, such as Figure 134 and Figure 135, indicate that running perf tools consistently leads to an increased ratio of late packets compared to tests where no additional processes are active, across all O-FH configurations. An initial explanation is that running perf introduces additional workload on the CPU, which slows down the O-DU/CU processes and causes more late packets. A reduction in the ratio of early packets is also observed when perf is running. However, considering the very small absolute number of early packets, this conclusion may not be statistically significant.

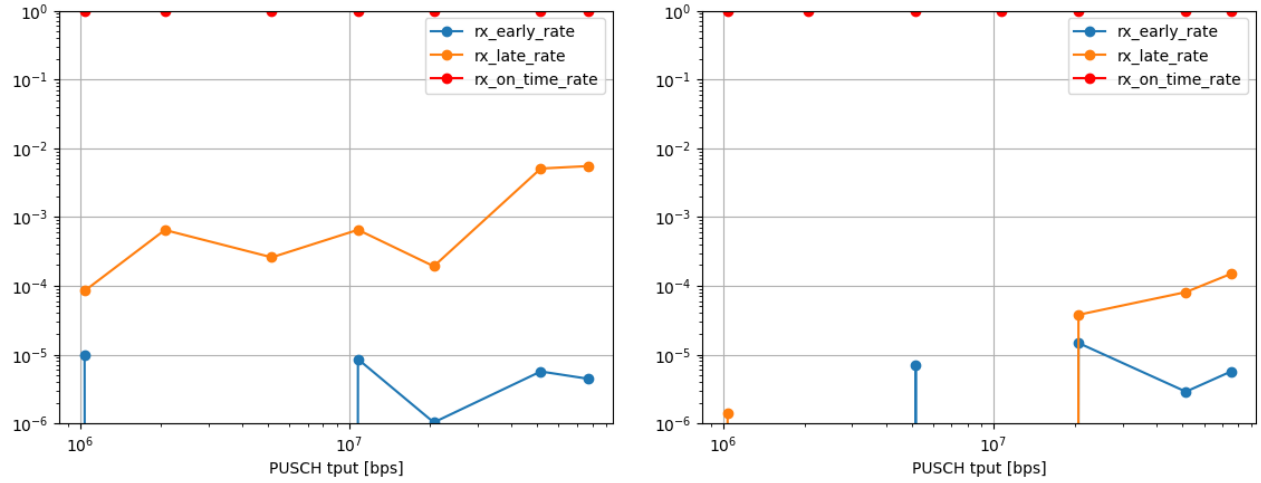


FIGURE 134. MEASURED RATE OF EARLY, LATE AND ON-TIME UL U-PLANE PACKETS IN THE HL3-LONG MID DELAY SCENARIO AND 78 MBPS OF UL TRAFFIC WHEN CONCURRENTLY RUNNING PERF (LEFT) AND WITHOUT IT (RIGHT)

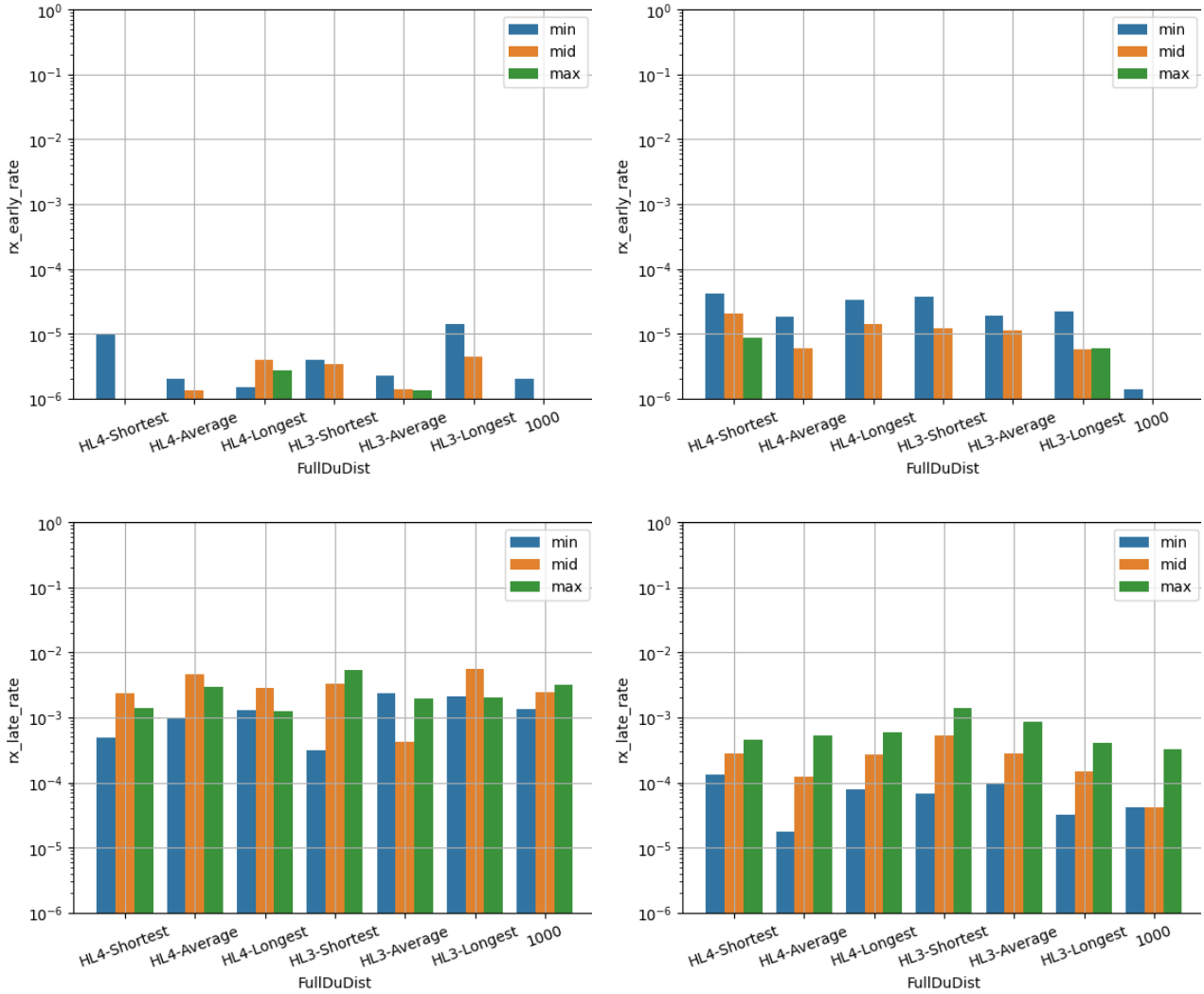


FIGURE 135. UL U-PLANE PACKET TIMING FOR MAX UL THROUGHPUTPUT (77 MBPS) WHEN CONCURRENTLY RUNNING PERF (LEFT) AND WITHOUT IT (RIGHT)

To further validate this observation and explanation, we conduct the same tests while explicitly introducing CPU workload by running the stress-ng command during the emulation. The results, shown in Figure 136, indicate that by introducing 20% CPU load on all CPU cores, there is a significant increase in the late packets detected by the O-DU. Thus, we conclude that the computing resources of the hosting server play a key role in the O-DU/CU packet timing.

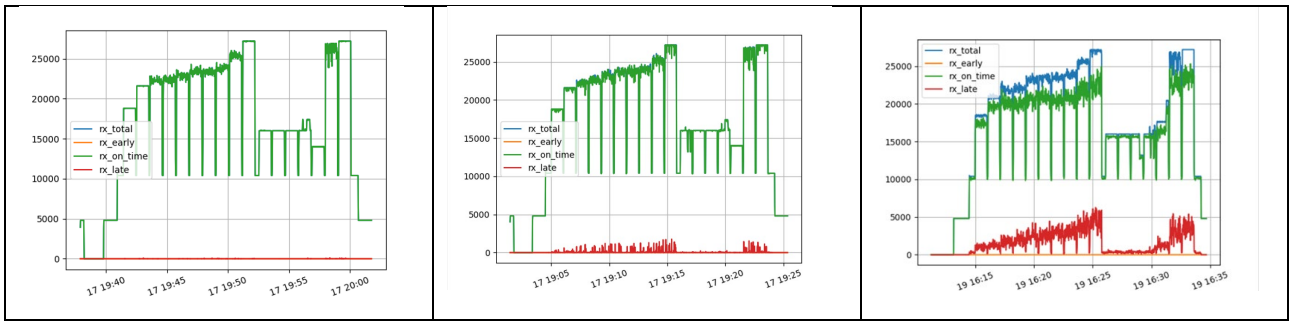


FIGURE 136. UL U-PLANE PACKET TIMING FOR MAX UL THROUGHPUTPUT (77 MBPS) WHEN CONCURRENTLY RUNNING PERF (LEFT) WITHOUT IT (CENTER) AND WHEN INTRODUCING A 20% CPU LOAD VIA STRESS-NG (RIGHT)

3.2.1.3.3.5 O-FH limits

3.2.1.3.3.5.1 Stretching O-FH delay to the limit

By default, the *max_proc_delay* parameter in the SRS O-DU/CU software is set to 5 slot intervals. As shown in previous sections, reducing the value of this parameter can lead to smaller latency. However, doing so also reduces the time margin for packet processing at the O-DU/CU, increasing the risk of failure in scenarios with longer O-FH delays. The figure below illustrates that, for a specific configuration, there exists a threshold on the O-FH delay: as long as the delay is below this threshold, all tests complete successfully, but once it exceeds this value, tests will consistently fail.

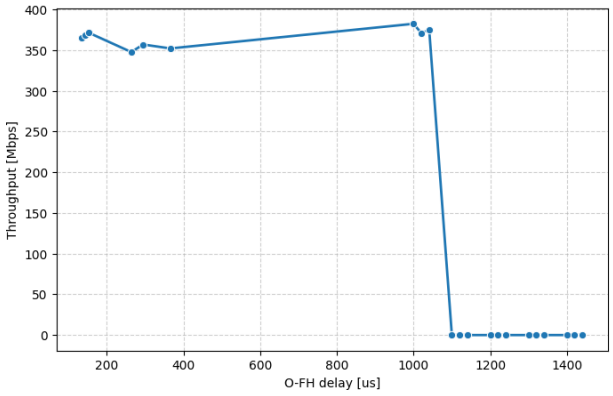


FIGURE 137. AWGN=0, MAX_PROC_DELAY=5, TESTS WITH OFH DELAY LARGER THAN 1100 US ALL FAILED

Therefore, to evaluate this trade-off, we conduct a series of tests to examine the maximum O-FH delay that the system can tolerate when applying different *max_proc_delay* values. Figure 138 shows whether a successful connection (marked as a green dot) was achieved for a given O-FH delay (y-axis, in us), or whether the connection was unsuccessful (marked with a red dot). A clear correlation between the *max_proc_delay* value and the failure threshold for O-FH delay. Obviously, reducing

max_proc_delay lowers the threshold on the O-FH delay that the system can work without failure. This finding aligns with theoretical expectations, as both a decreased *max_proc_delay* and an increased O-FH delay puts more stringent constraints on processing time, making the system more prone to failure. Thus, we can see a trade-off emerges between smaller latency and longer achievable O-FH distance.

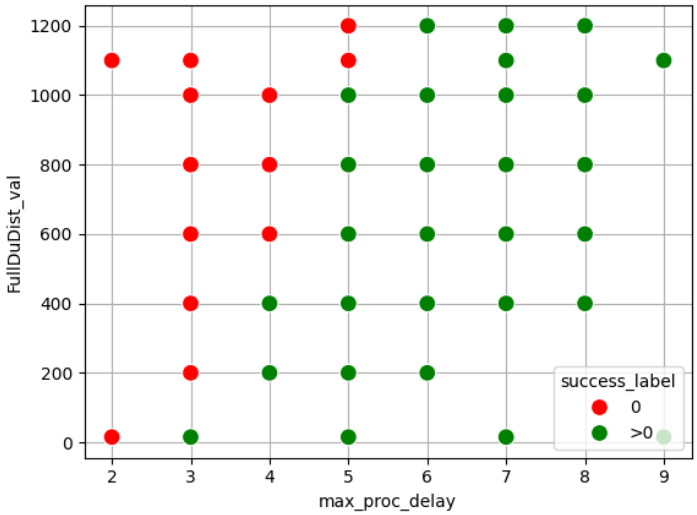


FIGURE 138. RELATIONSHIP BETWEEN MAX_PROC_DELAY AND MAX ACHIEVABLE O-FH DELAY

3.2.1.3.3.5.2 Effects of max_proc_delay, SNR, data-rate, etc.

To extend the previous analysis, we conduct similar tests with different data rates, SNR levels, and numbers of UEs. We can draw the conclusions from the below Figure 139 and Figure 140: increasing the throughput, number of UEs, or air-interface noise level tends to reduce the maximum O-FH delay under which the system can successfully complete the test. In other words, higher throughput, more UEs, or lower SNR values lead to a shift of the O-FH delay threshold toward smaller values. These conclusions align with the expectations, as the increase in all the three factors implies higher computational workload at the O-DU/CU, thus requiring a larger processing time margin. Then to meet this constraint, the system can only tolerate shorter O-FH delays.

FIGURE 139. SUCCESSFUL AND UNSUCCESSFUL CONNECTIONS UNDER DIFFERENT O-FH DELAYS, DL DATA RATES, UL SNRS, AND SETTINGS FOR THE O-DU PROCESSING TIME. THE FIGURE COMPARES THE RESULTS OBTAINED WITH 1 UE VS THOSE OBTAINED WITH 10 UES IN THE NETWORK

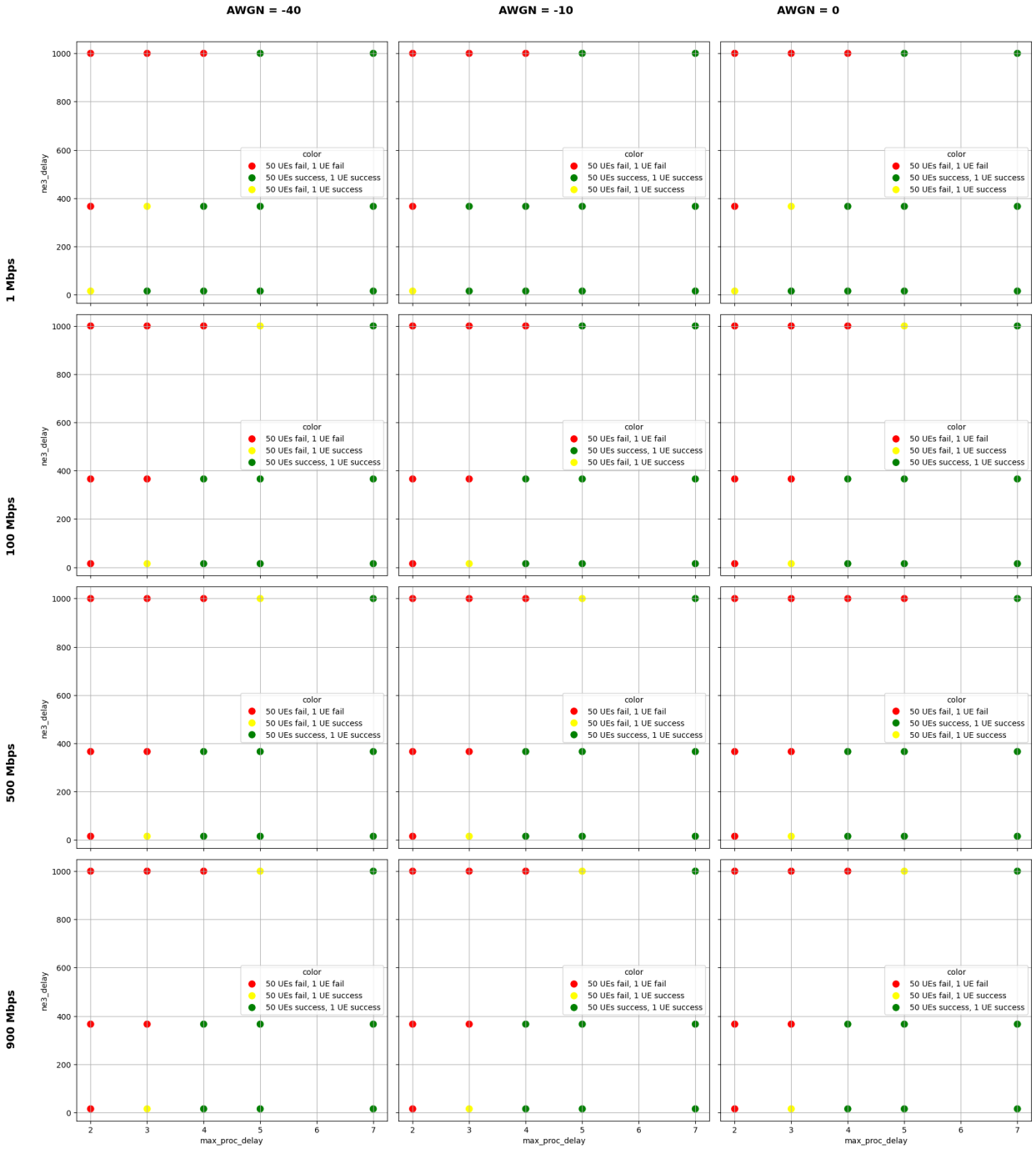


FIGURE 140. SUCCESSFUL AND UNSUCCESSFUL CONNECTIONS UNDER DIFFERENT O-FH DELAYS, DL DATA RATES, UL SNRS, AND SETTINGS FOR THE O-DU PROCESSING TIME. THE FIGURE COMPARES THE RESULTS OBTAINED WITH 1 UE VS THOSE OBTAINED WITH 50 UES IN THE NETWORK

3.2.1.3.4 Conclusions

To conclude, the experimental validation in UC2-PoC1 has shown that the concept of O-DU pooling at centralized locations is indeed feasible with the current O-RAN O-FH specifications, provided that the relevant parameters of the O-DU are appropriately adjusted. From the testbed implementation of the system, we can draw the following conclusions:

- It is crucial to adjust the O-DU TX and RX windows in order to reflect the O-RU timing and the packet delay and jitter introduced by the O-FH. With careful adjustment of these parameters, good end-to-end performance can be attained for the centralization distances targeted by this PoC.
- The E2E latency achieved by a system with a long O-FH is affected by multiple factors: number of registered UEs, computational capabilities of the servers running the RAN functions, SNR at the air interface, etc. Adjusting the time available for processing at the O-DU can be used to optimize the E2E latency, but conservative settings should be prioritized to avoid connectivity problems when the system is under high load conditions – both high traffic loads and high computational loads.
- The results indicate that, with our current testbed emulation of the considered disaggregated system, O-FH delays of up to 1 millisecond can be dealt with if the timing parameters are appropriately adjusted.

Nonetheless, we remark as well that our experimental validation is based on a simplified representation of the system, where the data-rates in the air interface and through the O-FH have to necessarily be scaled down in order to fit the hardware limitations present in a lab setup. While these results are encouraging, corroboration of them through a pilot implementation in a live, commercial network would be the natural next step.

3.2.2 PoC2: NPN X-validation/optimization with Digital Twin

3.2.2.1 Introduction

The Proof of Concept (PoC) for NPN X-validation/optimization using a Digital Twin successfully demonstrated the automated self-configuration capabilities of the NPN system during its initial setup. This ensured that the system achieved the desired performance levels expected by subscribers. This goal was accomplished by leveraging the NPN Network Digital Twin (NDT) platform (refer to Section 2.4), which recommended optimal configurations and employed flexible slicing techniques to seamlessly integrate the NPN with both Edge Computing environments and the Public Network. The NPN system automatically implemented these configuration recommendations from the Network Digital Twin platform, minimizing the need for manual intervention.

Subsequently, the system activated monitoring of actual usage and performance, enabling the collection, reporting, and analysis of patterns via the Digital Twin platform. This process established a knowledge base that supported data-driven optimization of performance and resource utilization, as demonstrated in UC3PoC1 Data-Driven RAN optimization for NPN deployments.

3.2.2.2 System Design

Building upon the results of UC1PoC1, which focused on the Zero-Touch distributed 3GPP network for deploying Non-public Networks, the NPN system was pre-integrated by default. It was then (re)configured in accordance with the recommendations provided by the Network Digital Twin system. These recommendations were based on inputs from NPN service stakeholders, including CSPs and Enterprise/Subscribers, regarding (i) anticipated traffic patterns and the associated network service performance levels required for the NPN, and (ii) any applicable policies or constraints for resource utilization efficiency, such as bandwidth or latency optimization. The Network Digital Twin platform generated and presented a prioritized list of alternative configurations that met these requirements and initiated the implementation process. This meant that (a) a set of configurable NPN parameters was automatically updated, and (b) the initial slicing scheme established at the NPN's startup was adjusted. The alignment between the delivered performance and expected performance for the specified traffic models was tracked, analysed, and demonstrated.

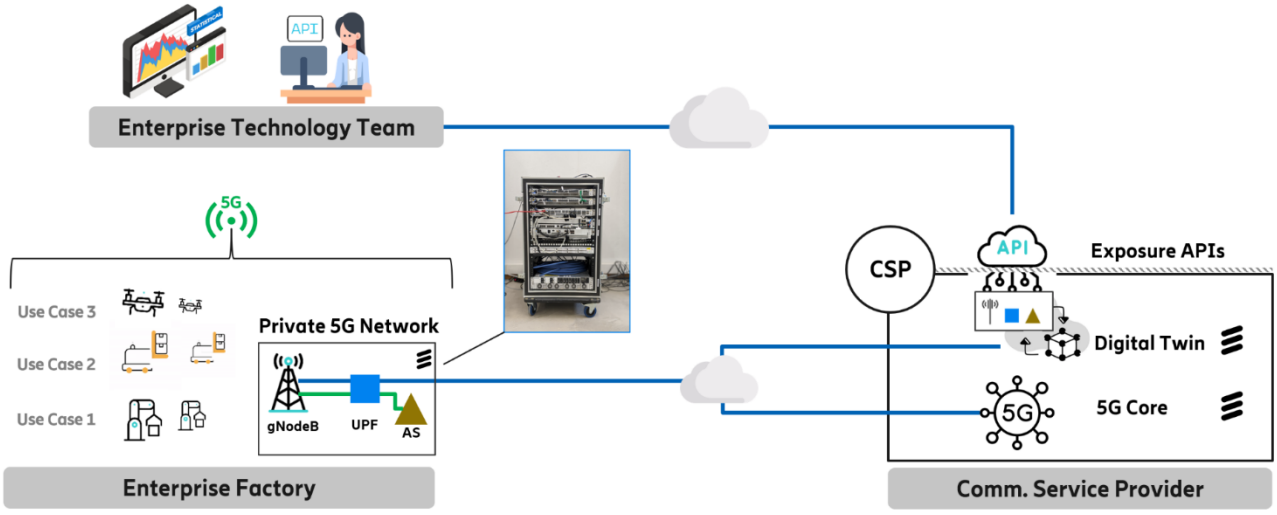


FIGURE 141. PNI-NPN DEPLOYMENT BETWEEN ENTERPRISE FACILITY AND CSP

3.2.2.3 Implementation

The NPN is deployed at CTTC facilities in Barcelona and integrated with the Public Network (advanced 5G Core) in the 5Tonic Lab, Leganes, Madrid, as showed in Figure 141. The Network Digital

Twin helps us to characterize the NPN system for various types of use cases and traffic. We ran different real-world use cases and selected one to explain how the Network Digital Twin platform can be utilized during various phases of the innovation cycle for connectivity use cases and to perform the optimization of the NPN system. We will illustrate how to use the Design Recommendations API described in Section 2.4.2 for an enterprise to deploy new connectivity use cases. Further, we will demonstrate the use of Analytics APIs to identify regular traffic patterns, determine actual traffic requirements, validate the initial recommendation, and identify potential optimizations.

3.2.2.4 Results

Enterprise Use Case: Deploying AGVs

An enterprise wishes to deploy five Automated Guided Vehicles (AGVs) in their factory or warehouse. The traffic model and the number of AGVs will determine the KPIs that must be achieved with the NPN network. The following Figure 142 shows the individual AGVs and the total target traffic KPI requirements. Based on these requirements, we can use the Design Recommendation API to determine the most suitable configuration that meets the traffic KPI requirements for this use case.

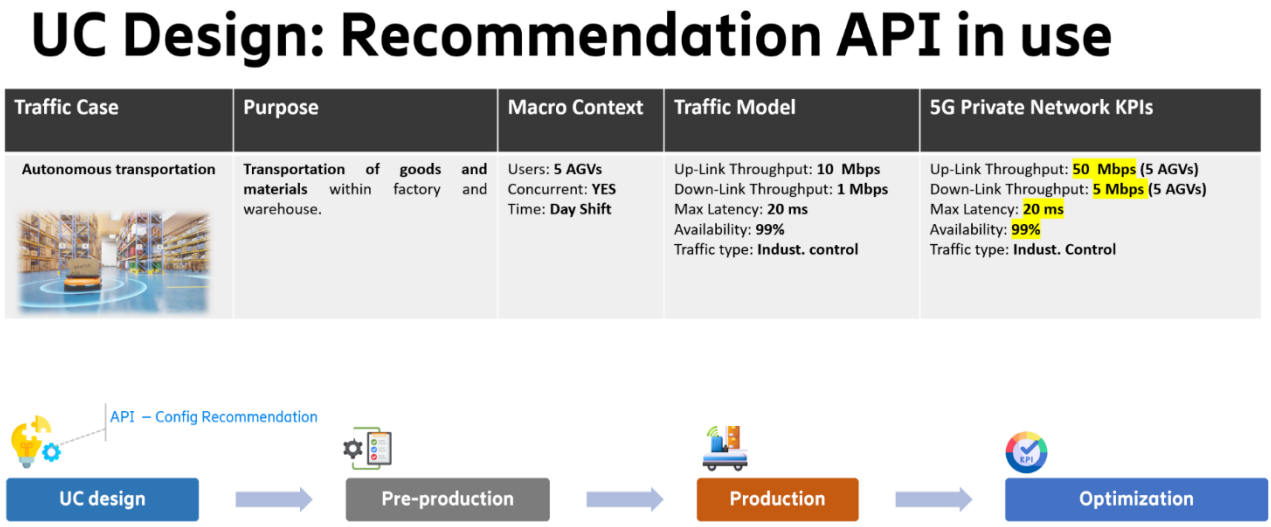


FIGURE 142. DESIGN RECOMMENDATION

Let's query the NDT platform's Design Recommendation API, providing the use case traffic model and expected traffic KPI requirements, and request the platform to suggest the most suitable configurations. Figure 143 presents the response of the Design Recommendation API. Approximately, it takes 20 seconds to receive the response. In this case, the NDT platform suggests three optimal configurations that fulfil the target KPI requirements, recommending the first-priority configuration.

POST https:// {{EXPOSURE_API_IP}} /5tonic-exposure/v1/npn/Pelican/recommend Send

Params Authorization Headers (9) Body Scripts Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify


```
{
  "optimizationInputData": [
    {
      "TrafficModel": {
        "protocol_stack": "UDP",
        "bandwidth": "50M",
        "traffic_direction": "uplink"
      },
      "MinThroughput": "50",
      "MaxOWDMean": "20"
    },
    {
      "TrafficModel": {
        "protocol_stack": "UDP",
        "bandwidth": "5M",
        "traffic_direction": "downlink"
      },
      "MinThroughput": "5",
      "MaxOWDMean": "20"
    }
  ],
  "optimizationType": "THROUGHPUT_OPTIMIZATION"
}
```


Body Cookies Headers (5) Test Results 200 OK 19.60 s 10.33 KB Save Response

Pretty Raw Preview Visualize

Recommend API Inputs: KPI's requirements

Recommendation Criteria	Uplink KPI Tput (Mbps)	Uplink KPI OWD (ms)	Uplink KPI Power (W)	Downlink KPI Tput (Mbps)	Downlink KPI OWD (ms)	Downlink KPI Power (W)
THROUGHPUT_OPTIMIZATION	50	20	N/A	5	20	N/A

Recommend API Outputs: Configurations sorted by THROUGHPUT_OPTIMIZATION

Priority	Bandwidth (MHz)	TDD Pattern	Uplink Max ach. Tput (Mbps)	Uplink Pred. OWD (ms)	Uplink Pred. Power Con. (W)	Downlink Max ach. Tput (Mbps)	Downlink Pred. OWD (ms)	Downlink Pred. Power Con. (W)
1	100	DDDSUDDSUU(10:2:2)	97.081	12.615	180.488	833.713	5.536	262.758
2	80	DDDSUDDSUU(10:2:2)	81.972	12.499	191.728	633.796	6.012	277.404
3	100	DDDSU(10:2:2)	73.013	13.288	174.782	833.945	5.779	286.529

FIGURE 143. DESIGN RECOMMENDATION RESPONSE

The Enterprise can use the Configuration API described in Section 2.4.2 to apply the recommended configurations to the NPN system and utilize the Configuration Retrieval API to validate the applied configuration.

Now, let's assume that the AGVs have been in service for six months. We then ask the Digital Twin platform: What has been the actual throughput demand of the AGVs? Here, we can use the Analytics APIs to identify regular traffic patterns, determine actual traffic requirements, validate the initial recommendation, and identify potential optimizations. The Analytics API is a powerful tool that provides analytical information on various domain KPIs, such as downlink/uplink throughput per user, downlink/uplink cell capacity, latency, and availability. Some of these KPIs are shown below.

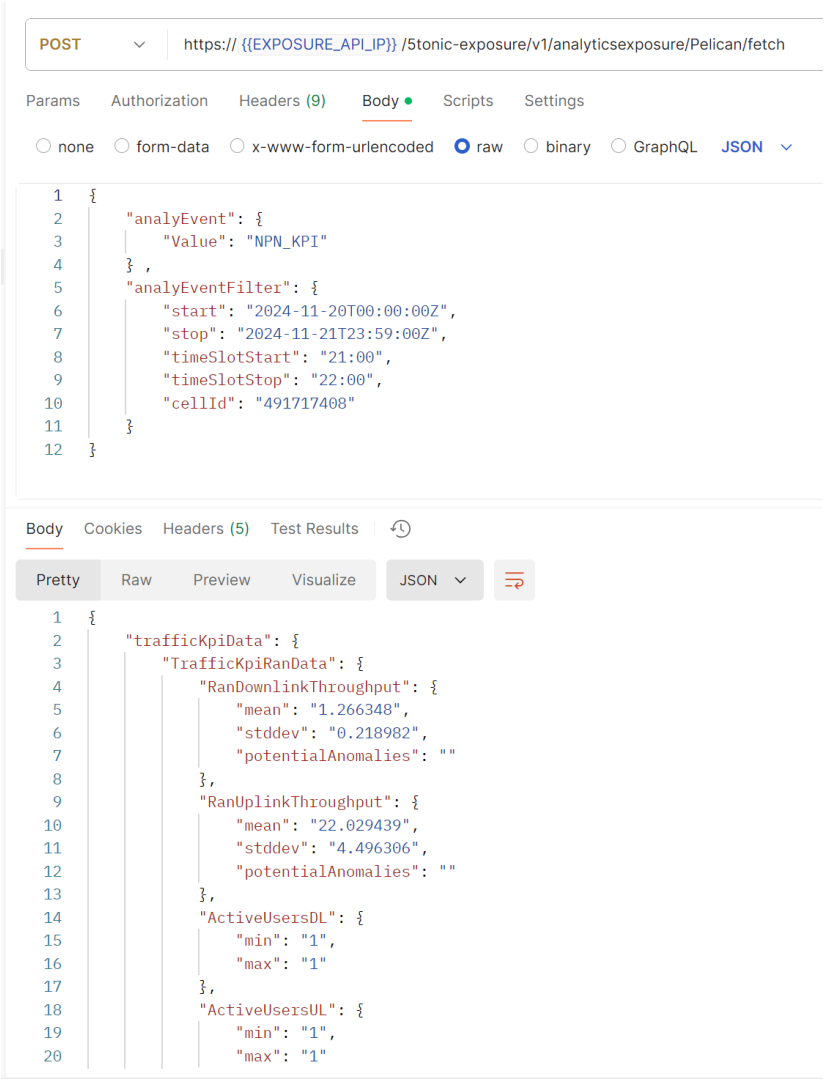


FIGURE 144. ANALYTICAL RESPONSE

POST https:// {{EXPOSURE_API_IP}} /5tonic-exposure/v1/analyticsexposure/Pelican/fetch Send

Params Authorization Headers (9) Body Scripts Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
{
    "analyEvent": {
        "value": "NPN_KPI"
    },
    "analyEventFilter": {
        "temporalGranSize": 600,
        "dnn": "pelican-embb"
    },
    "tgtUe": {
        "Ipv4Addr": "10.3.204.68"
    }
}
```

Body Cookies Headers (5) Test Results 200 OK 2.08 s 1.58 KB Save Response

Pretty Raw Preview Visualize JSON

```
{
    "trafficKpiData": {
        "TrafficKpiUeData": {
            "UeDownlinkThroughput": {
                "mean": "99.528601",
                "stddev": "12.681777",
                "potentialAnomalies": ""
            },
            "UeUplinkThroughput": {
                "mean": "N/A",
                "stddev": "N/A",
                "potentialAnomalies": ""
            }
```

FIGURE 145. THROUGHPUT PER USER KPI

POST https:// {{EXPOSURE_API_IP}} /5tonic-exposure/v1/analyticsexposure/Pelican/fetch

Params Authorization Headers (9) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
{
    "analyEvent": {
        "Value": "NPN_KPI"
    } ,
    "analyEventFilter": {
        "start": "2024-11-20T00:00:00Z",
        "stop": "2024-11-21T23:59:00Z",
        "timeSlotStart": "21:00",
        "timeSlotStop": "22:00",
        "cellId": "491717408"
    }
}
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
{
    "trafficKpiData": {
        "TrafficKpiRanData": {
            "RanDownlinkThroughput": {
                "mean": "1.266348",
                "stddev": "0.218982",
                "potentialAnomalies": ""
            },
            "RanUplinkThroughput": {
                "mean": "22.029439",
                "stddev": "4.496306",
                "potentialAnomalies": ""
            },
            "ActiveUsersDL": {
                "min": "1",
                "max": "1"
            },
            "ActiveUsersUL": {
                "min": "1",
                "max": "1"
```

FIGURE 146. CELL THROUGHPUT KPI

POST https:// {{EXPOSURE_API_IP}} /5tonic-exposure/v1/analyticsexposure/Pelican/fetch Send

Params Authorization Headers (9) Body Scripts Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
{
    "analyEvent": {
        "value": "NPN_KPI"
    },
    "analyEventFilter": {
        "temporalGranSize": 600,
        "dnn": "pelican-embb"
    },
    "tgtUe": {
        "Ipv4Addr": "10.3.204.68"
    }
}
```

Body Cookies Headers (5) Test Results 200 OK 2.08 s 1.58 KB Save Response

Pretty Raw Preview Visualize JSON

```
    },
    "TrafficKpiCoreData": {
        "UpfDataVolume": {
            "value": "853870882"
        }
    },
    "TrafficKpiNpnData": {
        "Latency": {
            "mean": "6.835326",
            "stddev": "1.307527"
        },
        "LinkLocalAvailability": {
            "value": "99.939269"
        },
        "LinkCentralAvailability": {
            "value": "0.000000"
        }
    }
```

FIGURE 147. LATENCY & AVAILABILITY KPI

The following Key Performance Indicators (KPIs) were successfully validated in the PoC for the example use case, as showed in Figure 144-Figure 147, including:

- downlink/uplink throughput per user KPI,
- downlink/uplink cell capacity KPI,
- application-level latency KPI,
- reliability KPI,
- coverage KPI,

- service availability KPI

The Network Digital Twin platform provided a framework to collect these KPIs and expose them via Network Exposure Analytics APIs, as previously demonstrated. All listed KPIs were collected, logged, and made accessible through the Network Exposure Analytics API, except for "KPI Reliability." Although reliability was not explicitly provided in the API output, it was derived from the primary KPI of latency by enriching the NEF information of the average latency value with a standard deviation measurement.

Characterization datasets and graphs were shared with CTTC for internal use and auditing purposes. Experimentation and real-world use case datasets were collected using the Network Exposure Analytics API. A demonstration video of this PoC was presented to partners, and it is available online[8] in the 6G BLUR repository.

3.2.2.5 Conclusions

In conclusion, the Non-Public Network (NPN) X-validation/optimization proof of concept (PoC) successfully demonstrated the potential of using a Network Digital Twin platform for automated self-configuration and optimization in non-public networks. Through the integration of Prediction, recommendation and analytics, we illustrated how enterprises can utilize the Design Recommendations API to deploy new connectivity use cases, such as the AGV deployment in industrial settings, tailored to specific target KPI requirements. The Analytics APIs were employed to identify regular traffic patterns, determine actual traffic needs, and validate the initial configuration recommended by the NDT platform, ensuring it met the use case traffic requirement KPIs. This PoC showcased the NDT platform's capabilities for optimization and the methodology's applicability to various real-world use cases, as detailed in Section 2.9 on the AI/ML Platform. The 'Final Models Selection' subsection highlighted the chosen models' superior performance in predictions and recommendations across different scenarios. The PoC underscored efficiency improvements and reduced the need for manual intervention, establishing a robust framework for ongoing performance enhancement and resource utilization. The Digital Twin platform proved to be a pivotal tool for Communication Service Providers (CSPs) to offer customized connectivity services quickly and at scale, enabling extensive customizations for all types of enterprise customers.

[8] Available online at: https://gitlab.cttc.es/mdi-6gblur/smart/-/tree/main/PoCs/UC2PoC2%20NPN%20Optimization%20with%20NDT

4 Conclusions

This deliverable presents the main outcomes of the algorithms and procedures developed within the different key concepts defined in the 6G-BLUR JOINT subproject. The defined nine key concepts, together with the interaction of those defined in the 6G-BLUR SMART subproject helped to define four PoCs dealing with relevant aspects such as the zero-touch deploying of a non-private network (NPNs) using commercial equipment, the dynamic end-to-end deployment of disaggregated mobile networks using open-source software and orchestration tools, the study of optimization strategies for the fronthaul segment of O-RAN based deployments to exploit the capabilities of this architecture in terms of multiplexing gains by centralising equipment, and the optimization of the deployed NPN by the integration of an NDT equipped with suitable APIs to perform closed-loop optimizations.

More specifically, the main outcomes of such key concepts and proof of concepts are:

- A commercial 5G NPN can initialize/auto-configure, auto-integrate with the public network, auto-integrate with the edge network functions, and become fully operational after 10 minutes.
- A complete end-to-end deployment of a disaggregated over-the-air cloud-native mobile network comprising RAN, core and transport segments and including O- RAN architectural principles can be achieved by integrating open-source solutions both for the mobile network entities itself and the management and orchestration tools. Such deployments can be done in several minutes and the data plane performance is similar to the bare-metal counterpart.
- O-DU pooling at centralized locations is indeed feasible with the current O-RAN O-FH specifications, provided that the relevant parameters of the O-DU are appropriately adjusted.
- NDTs enrich NPN networks by integrating predictive, recommendation and analytics APIs so enterprises can utilize the Design Recommendations API to deploy new connectivity use cases or reconfigure those running to achieve/ensure target KPI requirements within the deployed environment.

All the algorithms/procedures and PoCs developed within 6G-BLUR JOINT project, and complemented with those in 6G-BLUR SMART project has represented a valuable step beyond state of the art that are being exploited in other running SNS projects like 6G-SANDBOX, UNITY-6G and have been the basis for the development of new SNS project proposals dealing with the development of next-generation mobile networks.

5 References

[1] J. Baranda, S. Barrachina-Muñoz, R. Nikbakht, M. Payaró and J. Mangues-Bafalluy, “Disaggregating a 5G Non-Public Network via On-Demand Cloud-Native UPF Deployments,” in 19th International Conference on Network and Service Management (CNSM), 2023, 30 October- 2 November 2023, Niagara Falls, Canada.

[2] 6G BLUR Gitlab repository, [Online] Available at: https://gitlab.cttc.es/mdi-6gblur

[3] J. Baranda, J. Mangues, R. Martínez, L. Vettori, K. Antevski, C.J. Bernardos, X. Li, “Realizing the Network Service Federation Vision: Enabling Automated Multidomain Orchestration of Network Services”, in IEEE Vehicular Technologies Magazine, vol. 15, no. 2, pp. 48-57, 2020.

[4] O-RAN Alliance, ”O-RAN Alliance Specifications”. [Online]. Available at https://www.o-ran.org/specifications.

[5] O-RAN Alliance, “O-RAN Cloud Architecture and Deployment Scenarios for O-RAN Virtualized RAN 6.0,” 2024. [Online]. Available at: https://specifications.o-ran.org/specifications.

[6] J. Baranda, A.Bel, S. Barrachina, M. Payaró, J. Mangues-Bafalluy, "Distributed Sequential Cloud-Native Deployment of an End-to-End 5G Network with O-RAN Functions", in the 15th IEEE International Conference on Network of the Future (NoF), 2024, 2-4 October, Castelldefels, Spain.

[7] J. Baranda, A. Bel, S. Barrachina, M. Payaró, J. Mangues-Bafalluy, "Demo: On-demand Disaggregated Deployment of Cloud-Native Mobile Networks from Core to RAN", in 25th ACM International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing (Mobihoc), 2024, 14-17 October, Athens, Greece.

[8] srsRAN Project - Open Source RAN. [Online]. Available at: https://github.com/srsran.

[9] FlexRIC, O-RAN Alliance compliant nearRT-RIC, xApps. [Online]. Available at: https://gitlab.eurecom.fr/mosaic5g/flexric.

[10] 5G-ACIA "A 5G Traffic Model for Industrial Use Cases". [Online]. Available at : https://5g-acia.org/whitepapers/a-5g-traffic-model-for-industrial-use-cases.

[11] "scikit-learn". [Online]. Available at: https://scikit-learn.org/1.5/index.html.

[12] “StandardScaler”. [Online]. Available at: https://scikit-learn.org/1.5/modules/generated/sklearn.preprocessing.StandardScaler.html

[13] “GridSearchCV”. [Online]. Available at: https://scikit-learn.org/dev/modules/generated/sklearn.model_selection.GridSearchCV.html

[14] J. Baranda, A. Bel, S. Barrachina, M. Payaró, J. Mangues-Bafalluy, “End-to-End Slice Orchestration in a 5G cloud-native Mobile Network with O-RAN Split 7.2”, in the IEEE International Conference on Computer Communications (INFOCOM), 19-22 May 2025, London, UK.

[15] srsRAN Project – O-RAN 7.2 RU Guide. [Online]. Available at: https://docs.srsran.com/projects/project/en/latest/tutorials/source/oranRU/source/index.html

[16] K. Koutlia, B. Bojovic, S. Lagén, X. Zhang, P. Wang, and J. Liu. 2023. System Analysis of QoS Schedulers for XR Traffic in 5G NR. Elsevier Simulation Modelling Practice and Theory 125 (2023), 102745. https://doi.org/10.1016/j.simpat.2023.102745

[17] K. Koutlia, S. Lagen, and B. Bojovic, “Enabling QoS provisioning support for delay-critical traffic and multi-flow handling in ns-3 5G-LENA,” in Proceedings of the 2023 Workshop on ns-3, vol. 1 of WNS3 2023, pp. 45–51, ACM, June 2023.

[18] N. Villegas, A. Larrañaga, L. Diez, K. Koutlia, S. Lagén, R. Agüero, "Optimizing QoS MAC Scheduling in 5G NR: A Lyapunov Approach Evaluated with XR Traffic", in IEEE Transactions on Network and Service Management. Under review.

[19] K. Koutlia, S. Lagen, On the impact of Open RAN Fronthaul Control in scenarios with XR Traffic, Elsevier Computer Networks, Volume 253, 2024.

[20] A. Larrañaga, N. Villegas, K. Koutlia, L. Diez, R. Aguero, S. Lagen, "Novel Fronthaul Control Method to Address the Fronthaul/Air Interface Tradeoff", IEEE Wireless Commun. and Networking Conf. Workshops, Milan (Italy), Mar. 2025.

[21] N. Villegas, A. Larrañaga, L. Diez, K. Koutlia, S. Lagén, and R. Agüero, “Extending QoS-aware scheduling in ns-3 5G-LENA: A Lyapunov based solution,” in Proceedings of the 2024 Workshop on ns-3, ser. WNS3 ’24. New York, NY, USA: Association for Computing Machinery, Jun. 2024, pp. 54–59. [Online]. Available: https://dl.acm.org/doi/10.1145/3659111.3659118

[22] ‘Eve-ng’. Accessed: May. 15, 2025. [Online]. Available at: https://www.eve-ng.net/

[23] ‘Cisco IOS XRv 9000 Router’, Cisco. Accessed: Oct. 31, 2024. [Online]. Available: https://www.cisco.com/c/en/us/support/routers/ios-xrv-9000-router/series.html

[24] J. Moy, ‘OSPF Version 2’, Internet Engineering Task Force, Request for Comments RFC 2328, Apr. 1998. doi: 10.17487/RFC2328.

[25] B. Thomas, L. Andersson, and I. Minei, ‘LDP Specification’, Internet Engineering Task Force, Request for Comments RFC 5036, Oct. 2007. doi: 10.17487/RFC5036.

[26] A. Viswanathan, E. C. Rosen, and R. Callon, ‘Multiprotocol Label Switching Architecture’, Internet Engineering Task Force, Request for Comments RFC 3031, Jan. 2001. doi: 10.17487/RFC3031.

[27] R. T. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, ‘Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification’, Internet Engineering Task Force, Request for Comments RFC 2205, Sep. 1997. doi: 10.17487/RFC2205.

[28] ‘TeraFlowSDN | TeraFlow’. Accessed: May. 15, 2025. [Online]. Available: https://www.teraflow-h2020.eu/

[29] ‘Open vSwitch’. Accessed: May. 15, 2025. [Online]. Available: https://www.openvswitch.org/

[30] ONF, ‘OpenFlow Switch Specification Version 1.5.1 (Protocol version 0x06)’, 2015.

[31] ‘Ryu SDN Framework’. Accessed: May. 15, 2025. [Online]. Available: https://ryu-sdn.org/

[32] 3GPP, «5G Network Resource Model (NRM)», 3GPP TS 28541 V17111, 2024.

[33] B. Wu, D. Dhody, R. Rokui, T. Saad, y J. Mullooly, «A YANG Data Model for the RFC 9543 Network Slice Service», Internet Engineering Task Force, Internet-Draft draft-ietf-teas-ietf-network-slice-nbi-yang-22, may. 2025. [Online]. Available at: https://datatracker.ietf.org/doc/draft-ietf-teas-ietf-network-slice-nbi-yang/

[34] Yi, X. Wang, K. Li, S. k. Das, and M. Huang, "A comprehensive survey of network function virtualization," Computer Networks, vol. 133, pp. 212–262, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128618300306

[35] Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach, "Virtual Network Embedding: A Survey," IEEE Communications Sur- veys & Tutorials, vol. 15, no. 4, pp. 1888–1906, 2013.

[36] Wang, R. S. Batth, P. Zhang, G. S. Aujla, Y. Duan, and L. Ren, "VNE Solution for Network Differentiated QoS and Security Requirements: From the Perspective of Deep Reinforcement Learning," 2022.

[37] Fajjari, N. Aitsaadi, G. Pujolle, and H. Zimmermann, "VNE-AC: Virtual Network Embedding Algorithm Based on Ant Colony Meta- heuristic," in 2011 IEEE International Conference on Communications (ICC), 2011, pp. 1–6.

[38] H. Cao, S. Wu, Y. Hu, Y. Liu, and L. Yang, "A survey of embedding algorithm for virtual network embedding," China Communications, vol. 16, no. 12, pp. 1–33, 2019.

[39] Q. Lu and C. Huang, "Performance modeling and joint resource allocation algorithms for online virtual network embedding," IEEE Transactions on Network and Service Management, vol. 21, no. 1, pp. 1048–1066, 2024.

[40] H. Cao, L. Yang, Z. Liu, and M. Wu, "Exact solutions of VNE: A survey," China Communications, vol. 13, no. 6, pp. 48–62, 2016.

[41] H. Wu, F. Zhou, Y. Chen, and R. Zhang, "On virtual network embedding: Paths and cycles," in 2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), 2019, pp. 179–188.

[42] X. Li, H. Lu, W. Zhou, and P. Hong, "VNE-RFD: Virtual network embedding with resource fragmentation consideration," in 2014 IEEE Global Communications Conference, 2014, pp. 1842–1847.

[43] P. Zhang, Y. Hong, X. Pang, and C. Jiang, "Vne-hpso: Virtual network embedding algorithm based on hybrid particle swarm optimization," IEEE Access, vol. 8, pp. 213 389–213 400, 2020.

[44] H. Kim and S.-H. Lee, "An evaluation method for secure virtual network embedding algorithms," Journal of Computer Virology and Hacking Techniques, vol. 13, 11 2017.

[45] P. Zhang, Z. Luo, N. Kumar, M. Guizani, H. Zhang, and J. Wang, "CE-VNE: Constraint escalation virtual network embedding algorithm assisted by graph convolutional networks," Journal of Network and Computer Applications, vol. 221, p. 103736, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804523001558

[46] P. Zhang, C. Wang, N. Kumar, W. Zhang, and L. Liu, "Dynamic virtual network embedding algorithm based on graph convolution neural network and reinforcement learning," IEEE Internet of Things Journal, vol. 9, no. 12, pp. 9389–9398, 2022.

[47] S. Ma et al., "Graph Convolutional Network Aided Virtual Network Embedding for Internet of Thing," in IEEE Transactions on Network Science and Engineering, vol. 10, no. 1, pp. 265-274, 1 Jan.-Feb. 2023, doi: 10.1109/TNSE.2022.3207205.

[48] H. Liu, "Research on Virtual Network Embedding Model and Algorithm Based on Graph Attention Network and Multi-Layer Perceptron," 2023 5th International Conference on Artificial Intelligence and Computer Applications (ICAICA), Dalian, China, 2023, pp. 812-818, doi: 10.1109/ICAICA58456.2023.10405441.

[49] F. Habibi, M. Dolati, A. Khonsari and M. Ghaderi, "Accelerating Virtual Network Embedding with Graph Neural Networks," 2020 16th International Conference on Network and Service Management (CNSM), Izmir, Turkey, 2020, pp. 1-9, doi: 10.23919/CNSM50824.2020.9269128.

[50] Ning Chen, Peiying Zhang, Neeraj Kumar, Ching-Hsien Hsu, Laith Abualigah, Hailong Zhu, Spectral graph theory-based virtual network embedding for vehicular fog computing: A deep reinforcement learning architecture, Knowledge-Based Systems, Volume 257, 2022, 109931, ISSN 0950-7051, https://doi.org/10.1016/j.knosys.2022.109931.

[51] P. Zhang, C. Wang, N. Kumar, W. Zhang and L. Liu, "Dynamic Virtual Network Embedding Algorithm Based on Graph Convolution Neural Network and Reinforcement Learning," in IEEE Internet of Things Journal, vol. 9, no. 12, pp. 9389-9398, 15 June15, 2022, doi: 10.1109/JIOT.2021.3095094.

[52] ORAN. O-RAN Working Group 1 (Use Cases and Over-all Architecture WG). O-RAN Architecture Description (V11.00). TS. 2024.

[53] ORAN. O-RAN Working Group 9 (Open X-haul Transport WG). O-RAN Xhaul Packet Switched Architectures and Solutions (V7.00). TS. 2024

[54] ORAN. O-RAN Working Group 4 (Open Fronthaul Interfaces WG). Control, User and Synchronization Plane Specification (V14.00). TS. 2024.

[55] Luis M. Contreras, Ivan Bykov, and Krzysztof Grzegorz Szarkowicz, "5QI to DiffServ DSCP Mapping Example for Enforcement of 5G End-to-End Network Slice QoS", Internet-Draft draft-cbs-teas-5qi-to-dscp-mapping-00. Work in Progress. Internet Engineering Task Force, Mar. 2024.

[56] 3GPP. TS 138 101-1 - V16.5.0 - 5G; NR; User Equipment (UE) radio transmission and reception; Part 1: Range 1 Standalone (3GPP TS 38.101-1 version 16.5.0 Release 16). TS. 2020.

[57] R. Schmidt, M. Irazabal and N. Nikaein, "FlexRIC: an SDK for next-generation SD-RANs," in Proceedings of the 17th International Conference on Emerging Networking Experiments and Technologies CoNEXT '21, New York, NY, USA, 2021.

[58] Keysight Technologies, "P8828S RICtest — RAN Intelligent Controller Test Solutions | Keysight," [Online]. Available: "https://www.keysight.com/be/en/product/P8828S/rictest-ran-intelligent-controller-test-solutions.html". [Accessed: 06/2025].

[59] InfluxData, “Introduction to InfluxData's InfluxDB and TICK Stack,” [Online]. Available: https://www.influxdata.com/blog/introduction-to-influxdatas-influxdb-and-tick-stack/ . [Accessed June 2025].

[60] Gitlab, “examples/xApp/c/keysight · master · mosaic5g / Flexric · GitLab,” [Online]. Available: https://gitlab.eurecom.fr/mosaic5g/flexric/-/tree/master/examples/xApp/c/keysight . [Accessed June 2025].

[61] Keysight Technologies, “An O-RAN Digital Twin to train AI/ML in Anomaly Detection,” [Online]. Available: https://www.keysight.com/be/en/assets/3125-1308/application-notes/An-O-RAN-Digital-Twin-to-Train-AI-ML-in-Anomaly-Detection.pdf . [Accessed June 2025].

[62] Keysight Technologies, “P8822S RuSIM RAN Testing Toolset | Keysight,” [Online]. Available: https://www.keysight.com/be/en/product/P8822S/rusim-over-o-ran-fronthaul.html . [Accessed June 2025].

[63] Keysight Technologies, “Network Emulator 3 | Keysight,” [Online]. Available: https://www.keysight.com/be/en/products/network-test/network-test-hardware/network-emulator-3.html . [Accessed June 2025].

[64] O-RAN Working Group 4, “O-RAN control, user and synchronization plane specification 16.01,” O-RAN Alliance, 2024.

[65] FibroLAN, “Falcon-RX/G,” [Online]. Available: https://www.fibrolan.com/Falcon-RX . [Accessed 2025].

[66] The LINUX Foundation Projects, “DPDK -- The Open Source Data Plane Development Kit Accelerating Network Performance,” [Online]. Available: https://www.dpdk.org/ . [Accessed 2025].

[67] Keysight Technologies, “P8850S CoreSIM – Core Simulation RAN Solutions | Keysight,” [Online]. Available: https://www.keysight.com/be/en/product/P8850S/coresim-core-simulation-ran-solutions.html . [Accessed 2025].

[68] Keysight Technologies, “S5040A Open RAN Studio Player and Capture Appliance | Keysight,” [Online]. Available: https://www.keysight.com/be/en/product/S5040A/open-ran-studio-player-and-capture-appliance.html . [Accessed 2025].

[69] Keysight Technologies, “KS8400A PathWave Test Automation - Keysight,” [Online]. Available: https://www.keysight.com/be/en/product/KS8400A/pathwave-test-automation-developers-system.html . [Accessed 2025].

[70] O-RAN Fronthaul Working Group, “O-RAN.WG4.IOT.0-R004-v12.00 - Fronthaul Interoperability Test Specification (IOT),” O-RAN Alliance, 2024.

6 Annex

6.1 3GPP Slice Requests (NRM TS 28.541 [33])

6.1.1 Backhaul Control Plane Request


```
{
  "NetworkSlice1": {
    "operationalState": "",
    "administrativeState": "",
    "serviceProfileList": [],
    "networkSliceSubnetRef": "TopSliceSubnet1"
  },
  "TopSliceSubnet1": {
    "operationalState": "",
    "administrativeState": "",
    "nsInfo": {},
    "managedFunctionRef": [],
    "networkSliceSubnetType": "TOP_SLICESUBNET",
    "SliceProfileList": [
      {
        "sliceProfileId": "TopId",
        "pLMNInfoList": null,
        "TopSliceSubnetProfile": {
          "dLThptPerSliceSubnet": {
            "GuaThpt": 310,
            "MaxThpt": 620
          },
          "uLThptPerSliceSubnet": {
            "GuaThpt": 160,
            "MaxThpt": 320
          },
          "dLLatency": 20,
          "uLLatency": 20
        }
      }
    ],
    "networkSliceSubnetRef": [
      "RANSliceSubnet1"
```

```
    ]
  },
  "RANSliceSubnet1": {
    "operationalState": "",
    "administrativeState": "",
    "nsInfo": {},
    "managedFunctionRef": [],
    "networkSliceSubnetType": "RAN_SLICESUBNET",
    "SliceProfileList": [
      {
        "sliceProfileId": "RANId",
        "pLMNInfoList": null,
        "RANSliceSubnetProfile": {
          "dLThptPerSliceSubnet": {
            "GuaThpt": 310,
            "MaxThpt": 620
          },
          "uLThptPerSliceSubnet": {
            "GuaThpt": 160,
            "MaxThpt": 320
          },
          "dLLatency": 20,
          "uLLatency": 20
        }
      }
    ],
    "networkSliceSubnetRef": [
      "BackhaulSliceSubnetN2"
    ]
  },
  "BackhaulSliceSubnetN2": {
    "operationalState": "",
    "administrativeState": "",
    "nsInfo": {},
    "managedFunctionRef": [],
    "networkSliceSubnetType": "RAN_SLICESUBNET",
    "SliceProfileList": [
      {
        "sliceProfileId": "BackhaulId",
        "pLMNInfoList": null,
        "RANSliceSubnetProfile": {
```

```
        "dLThptPerSliceSubnet": {
          "GuaThpt": 1,
          "MaxThpt": 2
        },
        "uLThptPerSliceSubnet": {
          "GuaThpt": 1,
          "MaxThpt": 2
        },
        "dLLatency": 20,
        "uLLatency": 20
      }
    }
  ],
  "EpTransport": [
    "EpTransport CU-N2",
    "EpTransport AMF-N2"
  ]
},
"EpTransport CU-N2": {
  "IpAddress": "10.60.11.3",
  "logicalInterfaceInfo": {
    "logicalInterfaceType": "VLAN",
    "logicalInterfaceId": "100"
  },
  "NextHopInfo": "4.4.4.4",
  "qosProfile": "A",
  "EpApplicationRef": [
    "EP_N2 CU-N2"
  ]
},
"EP_N2 CU-N2": {
  "localAddress": "10.60.11.3",
  "remoteAddress": "10.60.60.105",
  "epTransportRef": [
    "EpTransport CU-N2"
  ]
},
"EpTransport AMF-N2": {
  "IpAddress": "10.60.60.105",
  "logicalInterfaceInfo": {
    "logicalInterfaceType": "VLAN",
```

```
      "logicalInterfaceId": "100"
    },
    "NextHopInfo": "5.5.5.5",
    "qosProfile": "A",
    "EpApplicationRef": [
      "EP_N2 AMF-N2"
    ]
  },
  "EP_N2 AMF-N2": {
    "localAddress": "10.60.60.105",
    "remoteAddress": "10.60.11.3",
    "epTransportRef": [
      "EpTransport UPF-N2"
    ]
  }
}
```


6.1.2 Backhaul User Plane Slice 1 Request


```
{
  "NetworkSlice1": {
    "operationalState": "",
    "administrativeState": "",
    "serviceProfileList": [],
    "networkSliceSubnetRef": "TopSliceSubnet1"
  },
  "TopSliceSubnet1": {
    "operationalState": "",
    "administrativeState": "",
    "nsInfo": {},
    "managedFunctionRef": [],
    "networkSliceSubnetType": "TOP_SLICESUBNET",
    "SliceProfileList": [
      {
        "sliceProfileId": "TopId",
        "pLMNInfoList": null,
        "TopSliceSubnetProfile": {
          "dLThptPerSliceSubnet": {
            "GuaThpt": 310,
            "MaxThpt": 620
```

```
        },
        "uLThptPerSliceSubnet": {
          "GuaThpt": 160,
          "MaxThpt": 320
        },
        "dLLatency": 20,
        "uLLatency": 20
      }
    }
  ],
  "networkSliceSubnetRef": [
    "RANSliceSubnet1"
  ]
},
"RANSliceSubnet1": {
  "operationalState": "",
  "administrativeState": "",
  "nsInfo": {},
  "managedFunctionRef": [],
  "networkSliceSubnetType": "RAN_SLICESUBNET",
  "SliceProfileList": [
    {
      "sliceProfileId": "RANId",
      "pLMNInfoList": null,
      "RANSliceSubnetProfile": {
        "dLThptPerSliceSubnet": {
          "GuaThpt": 310,
          "MaxThpt": 620
        },
        "uLThptPerSliceSubnet": {
          "GuaThpt": 160,
          "MaxThpt": 320
        },
        "dLLatency": 20,
        "uLLatency": 20
      }
    }
  ],
  "networkSliceSubnetRef": [
  "BackhaulSliceSubnetN31"
  ]
```

```
},
"BackhaulSliceSubnetN31": {
  "operationalState": "",
  "administrativeState": "",
  "nsInfo": {},
  "managedFunctionRef": [],
  "networkSliceSubnetType": "RAN_SLICESUBNET",
  "SliceProfileList": [
    {
      "sliceProfileId": "BackhaulId",
      "pLMNInfoList": null,
      "RANSliceSubnetProfile": {
        "dLThptPerSliceSubnet": {
          "GuaThpt": 10,
          "MaxThpt": 20
        },
        "uLThptPerSliceSubnet": {
          "GuaThpt": 10,
          "MaxThpt": 20
        },
        "dLLatency": 20,
        "uLLatency": 20
      }
    }
  ],
  "EpTransport": [
    "EpTransport CU-N31",
    "EpTransport UPF-N31"
  ]
},
"EpTransport CU-N31": {
  "IpAddress": "10.60.11.3",
  "logicalInterfaceInfo": {
    "logicalInterfaceType": "VLAN",
    "logicalInterfaceId": "102"
  },
  "NextHopInfo": "4.4.4.4",
  "qosProfile": "C",
  "EpApplicationRef": [
    "EP_N3 CU-N31"
  ]
```

```
  },
  "EP_N3 CU-N31": {
    "localAddress": "10.60.11.3",
    "remoteAddress": "10.60.60.106",
    "epTransportRef": [
      "EpTransport CU-N31"
    ]
  },
  "EpTransport UPF-N31": {
    "IpAddress": "10.60.60.106",
    "logicalInterfaceInfo": {
      "logicalInterfaceType": "VLAN",
      "logicalInterfaceId": "102"
    },
    "NextHopInfo": "5.5.5.5",
    "qosProfile": "C",
    "EpApplicationRef": [
      "EP_N3 UPF-N31"
    ]
  },
  "EP_N3 UPF-N31": {
    "localAddress": "10.60.60.106",
    "remoteAddress": "10.60.11.3",
    "epTransportRef": [
      "EpTransport UPF-N31"
    ]
  }
}
```


6.1.3 Backhaul User Plane Slice 2 Request


```
{
  "NetworkSlice1": {
    "operationalState": "",
    "administrativeState": "",
    "serviceProfileList": [],
    "networkSliceSubnetRef": "TopSliceSubnet1"
  },
  "TopSliceSubnet1": {
    "operationalState": "",
```

```
    "administrativeState": "",
    "nsInfo": {},
    "managedFunctionRef": [],
    "networkSliceSubnetType": "TOP_SLICESUBNET",
    "SliceProfileList": [
      {
        "sliceProfileId": "TopId",
        "pLMNInfoList": null,
        "TopSliceSubnetProfile": {
          "dLThptPerSliceSubnet": {
            "GuaThpt": 310,
            "MaxThpt": 620
          },
          "uLThptPerSliceSubnet": {
            "GuaThpt": 160,
            "MaxThpt": 320
          },
          "dLLatency": 10,
          "uLLatency": 10
        }
      }
    ],
    "networkSliceSubnetRef": [
      "RANSliceSubnet1"
    ]
  },
  "RANSliceSubnet1": {
    "operationalState": "",
    "administrativeState": "",
    "nsInfo": {},
    "managedFunctionRef": [],
    "networkSliceSubnetType": "RAN_SLICESUBNET",
    "SliceProfileList": [
      {
        "sliceProfileId": "RANId",
        "pLMNInfoList": null,
        "RANSliceSubnetProfile": {
          "dLThptPerSliceSubnet": {
            "GuaThpt": 310,
            "MaxThpt": 620
          },
```

```
          "uLThptPerSliceSubnet": {
            "GuaThpt": 160,
            "MaxThpt": 320
          },
          "dLLatency": 20,
          "uLLatency": 20
        }
      }
    ],
    "networkSliceSubnetRef": [
    "BackhaulSliceSubnetN32"
    ]
  },
  "BackhaulSliceSubnetN32": {
    "operationalState": "",
    "administrativeState": "",
    "nsInfo": {},
    "managedFunctionRef": [],
    "networkSliceSubnetType": "RAN_SLICESUBNET",
    "SliceProfileList": [
      {
        "sliceProfileId": "BackhaulId",
        "pLMNInfoList": null,
        "RANSliceSubnetProfile": {
          "dLThptPerSliceSubnet": {
            "GuaThpt": 100,
            "MaxThpt": 200
          },
          "uLThptPerSliceSubnet": {
            "GuaThpt": 20,
            "MaxThpt": 40
          },
          "dLLatency": 5,
          "uLLatency": 5
        }
      }
    ],
    "EpTransport": [
      "EpTransport CU-N32",
      "EpTransport UPF-N32"
    ]
```

```
  },
  "EpTransport CU-N32": {
    "IpAddress": "10.60.11.3",
    "logicalInterfaceInfo": {
      "logicalInterfaceType": "VLAN",
      "logicalInterfaceId": "101"
    },
    "NextHopInfo": "4.4.4.4",
    "qosProfile": "B",
    "EpApplicationRef": [
      "EP_N3 CU-N32"
    ]
  },
  "EP_N3 CU-N32": {
    "localAddress": "10.60.11.3",
    "remoteAddress": "10.60.10.6",
    "epTransportRef": [
      "EpTransport CU-N32"
    ]
  },
  "EpTransport UPF-N32": {
    "IpAddress": "10.60.10.6",
    "logicalInterfaceInfo": {
      "logicalInterfaceType": "VLAN",
      "logicalInterfaceId": "101"
    },
    "NextHopInfo": "5.5.5.5",
    "qosProfile": "B",
    "EpApplicationRef": [
      "EP_N3 UPF-N32"
    ]
  },
  "EP_N3 UPF-N32": {
    "localAddress": "10.60.10.6",
    "remoteAddress": "10.60.11.3",
    "epTransportRef": [
      "EpTransport UPF-N32"
    ]
  }
}
```

6.2 TeraflowSDN L2VPN Service Descriptor


```
{
  "services": [
    {
      "service_id": {
        "context_id": {
          "context_uuid": {
            "uuid": "admin"
          }
        },
        "service_uuid": {
          "uuid": "l2-acl-svc-17429923732568250"
        }
      },
      "service_type": 2,
      "service_status": {
        "service_status": 1
      },
      "service_endpoint_ids": [
        {
          "device_id": {
            "device_uuid": {
              "uuid": "4.4.4.4"
            }
          },
          "endpoint_uuid": {
            "uuid": "0/0/0-GigabitEthernet0/0/0/0"
          }
        },
        {
          "device_id": {
            "device_uuid": {
              "uuid": "5.5.5.5"
            }
          },
          "endpoint_uuid": {
            "uuid": "0/0/3-GigabitEthernet0/0/0/3"
          }
        }
      ],
```

```
      "service_constraints": [
        {
          "custom": {
            "constraint_type": "bandwidth[kbps]",
            "constraint_value": "20"
          }
        },
        {
          "custom": {
            "constraint_type": "latency[ms]",
            "constraint_value": "20"
          }
        }
      ],
      "service_config": {
        "config_rules": [
          {
            "action": 1,
            "custom": {
              "resource_key": "/settings",
              "resource_value": {}
            }
          },
          {
            "action": 1,
            "custom": {
              "resource_key": "/device[4.4.4.4]/endpoint[0/0/0-GigabitEther-
net0/0/0/0]/settings",
              "resource_value": {
                "sub_interface_index": 0,
                "vlan_id": 100,
                "circuit_id": "100",
                "remote_router": "5.5.5.5",
                "ni_name": "ELAN100"
              }
            }
          },
          {
            "action": 1,
            "custom": {
```

```
              "resource_key": "/device[5.5.5.5]/endpoint[0/0/3-GigabitEther-
net0/0/0/3]/settings",
              "resource_value": {
                "sub_interface_index": 0,
                "vlan_id": 100,
                "circuit_id": "100",
                "remote_router": "4.4.4.4",
                "ni_name": "ELAN100"
              }
            }
          }
        ]
      }
    },
    {
      "service_id": {
        "context_id": {
          "context_uuid": {
            "uuid": "admin"
          }
        },
        "service_uuid": {
          "uuid": "l2-acl-svc-17429923733224160"
        }
      },
      "service_type": 2,
      "service_status": {
        "service_status": 1
      },
      "service_endpoint_ids": [
        {
          "device_id": {
            "device_uuid": {
              "uuid": "4.4.4.4"
            }
          },
          "endpoint_uuid": {
            "uuid": "0/0/0-GigabitEthernet0/0/0/0"
          }
        },
        {
```

```
          "device_id": {
            "device_uuid": {
              "uuid": "5.5.5.5"
            }
          },
          "endpoint_uuid": {
            "uuid": "0/0/3-GigabitEthernet0/0/0/3"
          }
        }
      ],
      "service_constraints": [
        {
          "custom": {
            "constraint_type": "bandwidth[kbps]",
            "constraint_value": "200"
          }
        },
        {
          "custom": {
            "constraint_type": "latency[ms]",
            "constraint_value": "5"
          }
        }
      ],
      "service_config": {
        "config_rules": [
          {
            "action": 1,
            "custom": {
              "resource_key": "/settings",
              "resource_value": {}
            }
          },
          {
            "action": 1,
            "custom": {
              "resource_key": "/device[4.4.4.4]/endpoint[0/0/0-GigabitEther-
net0/0/0/0]/settings",
              "resource_value": {
                "sub_interface_index": 0,
                "vlan_id": 101,
```

```
                "circuit_id": "101",
                "remote_router": "5.5.5.5",
                "ni_name": "ELAN101"
              }
            }
          },
          {
            "action": 1,
            "custom": {
              "resource_key": "/device[5.5.5.5]/endpoint[0/0/3-GigabitEther-
net0/0/0/3]/settings",
              "resource_value": {
                "sub_interface_index": 0,
                "vlan_id": 101,
                "circuit_id": "101",
                "remote_router": "4.4.4.4",
                "ni_name": "ELAN101"
              }
            }
          }
        ]
      }
    },
    {
      "service_id": {
        "context_id": {
          "context_uuid": {
            "uuid": "admin"
          }
        },
        "service_uuid": {
          "uuid": "l2-acl-svc-17429923733753200"
        }
      },
      "service_type": 2,
      "service_status": {
        "service_status": 1
      },
      "service_endpoint_ids": [
        {
          "device_id": {
```

```
            "device_uuid": {
              "uuid": "4.4.4.4"
            }
          },
          "endpoint_uuid": {
            "uuid": "0/0/0-GigabitEthernet0/0/0/0"
          }
        },
        {
          "device_id": {
            "device_uuid": {
              "uuid": "5.5.5.5"
            }
          },
          "endpoint_uuid": {
            "uuid": "0/0/3-GigabitEthernet0/0/0/3"
          }
        }
      ],
      "service_constraints": [
        {
          "custom": {
            "constraint_type": "bandwidth[kbps]",
            "constraint_value": "200"
          }
        },
        {
          "custom": {
            "constraint_type": "latency[ms]",
            "constraint_value": "10"
          }
        }
      ],
      "service_config": {
        "config_rules": [
          {
            "action": 1,
            "custom": {
              "resource_key": "/settings",
              "resource_value": {}
            }
```

```
          },
          {
            "action": 1,
            "custom": {
              "resource_key": "/device[4.4.4.4]/endpoint[0/0/0-GigabitEther-
net0/0/0/0]/settings",
              "resource_value": {
                "sub_interface_index": 0,
                "vlan_id": 102,
                "circuit_id": "102",
                "remote_router": "5.5.5.5",
                "ni_name": "ELAN102"
              }
            }
          },
          {
            "action": 1,
            "custom": {
              "resource_key": "/device[5.5.5.5]/endpoint[0/0/3-GigabitEther-
net0/0/0/3]/settings",
              "resource_value": {
                "sub_interface_index": 0,
                "vlan_id": 102,
                "circuit_id": "102",
                "remote_router": "4.4.4.4",
                "ni_name": "ELAN102"
              }
            }
          }
        ]
      }
    }
  ]
}
```