

Grant Agreement No.: 101192650

Type of action: HORIZON JU Research and Innovation

Topic: HORIZON-JU-SNS-2024-STREAM-B-01-01

Call: HORIZON-JU-SNS-2024

D3.1 INTERMEDIATE REPORT ON NETWORK IMPLEMENTATION AND EXPOSURE INCLUDING OPEN RAN, XHAUL, SATELLITE, RIC COMPONENTS AND DIGITAL TWIN APPLICATION SPECIFICATIONS AND DESIGN

Work package	WP3
Task	T3.1, T3.2 & T3.3
Due date	30/05/2026
Submission date	31/05/2026
Deliverable lead	SRS
Version	1.0
Dissemination Level	Public
Editor	SRS
Authors	Oriol Font (SRS), Efi Dvir (CER), Javier Velázquez Martínez (TID), Luis Miguel Contreras Murillo (TID), Purva Joshi (DLR), Jorge Baranda (CTTC), Albert Bel (CTTC), Engin Zeydan (CTTC), Luis Blanco (CTTC), Husnain Shahid (CTTC), Berk Buzcu (HES-SO), Jetmir Haxhibeqiri (IMEC), Vasilis Maglogiannis (IMEC), Dries Naudts (IMEC), Jeroen Hoebeke (IMEC),

	Sotiris Spantideas (FDI), Anastasios Giannopoulos (FDI), Marilina Bartsioka (FDI), Nikolaos Kanakaris(IPTO), Nikolaos Stamatiou (IPTO), Stefanos Plastras (NCSRD), Harilaos Koumaras (NCSRD), Pawel Kryszkiewicz (PUT), Marcin Pakuła (PUT), Muhamad Luthfi (DLR)
Reviewers	Internal: Tomaso de Cola (DLR), Gil Kedar (CER), Adrian Kliks (PUT), Pawel Kryszkiewicz (PUT) External: Matteo Pagin (KEY), Selva Vía (CTTC)

Abstract	It reports on the development plans and progress of the UNITY-6G components, network exposure, supporting the POC and the specifications and planned implementation of the Digital Twin application. It will present the proposal of the inference, structured data models for information exchange and the overall concept of the DT “what-if” modification.
Keywords	O-RAN, xHaul, Satellite, RIC, non-3GPP RAN, TSN, Digital Twin, network exposure, API management, implementation, early integration, AI-driven models, datasets

Document Revision History

Version	Date	Description of change	List of contributor(s)
V0.0	16/02/2026	Document creation	Oriol Font (SRS)
V0.1	26/03/2026	ToC and initial section definition	Oriol Font (SRS), Maria Serrano (NBC), Pawel Kryszkiewicz (PUT), Jetmir Haxhibeqiri (IMEC)
V0.2	05/05/2026	First complete draft	All
V0.9	22/05/2026	Revision of the document according to	All
V1.0	31/05/2026	Final revised document	All

DISCLAIMER



Co-funded by
the European Union



Project funded by



Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Swiss Confederation

Federal Department of Economic Affairs,
Education and Research EAER
State Secretariat for Education,
Research and Innovation SERI

The **unity-6G** project has received funding from the [Smart Networks and Services Joint Undertaking \(SNS JU\)](#) under the European Union's [Horizon Europe research and innovation programme](#) under Grant Agreement No 101192650. This work has received funding from the [Swiss State Secretariat for Education, Research, and Innovation \(SERI\)](#).

Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.



Co-funded by
the European Union

COPYRIGHT NOTICE

© 2025 - 2027 UNITY-6G

Project funded by the European Commission in the Horizon Europe Programme		
Nature of the deliverable:	Report	
Dissemination Level		
PU	Public, fully open, e.g. web (Deliverables flagged as public will be automatically published in CORDIS project's page)	✓
SEN	Sensitive, limited under the conditions of the Grant Agreement	
Classified R-UE/ EU-R	EU RESTRICTED under the Commission Decision No2015/ 444	
Classified C-UE/ EU-C	EU CONFIDENTIAL under the Commission Decision No2015/ 444	
Classified S-UE/ EU-S	EU SECRET under the Commission Decision No2015/ 444	

* R: Document, report (excluding the periodic and final reports)

DEM: Demonstrator, pilot, prototype, plan designs

DEC: Websites, patents filing, press & media actions, videos, etc.

DATA: Data sets, microdata, etc.

DMP: Data management plan

ETHICS: Deliverables related to ethics issues.

SECURITY: Deliverables related to security issues

OTHER: Software, technical diagram, algorithms, models, etc.

EXECUTIVE SUMMARY

This deliverable presents the design, implementation progress and early integration of the main components that shape the UNITY-6G architecture. It summarizes the work carried out in the first phase of Work Package (WP)3 and explains how these activities support the project's objectives, while preparing the ground for the upcoming deployment of the described components in the project's proof-of-concept (PoC) implementations. The document covers the architectural blueprint, implementation strategies, integration plans, and the data- and AI-driven models that will underpin the UNITY-6G system. Particular attention is given to network exposure, RAN and core evolution, transport/xHaul enhancements, satellite integration, and the first version of the UNITY-6G Digital Twin (DT) application.

The deliverable starts by reporting the progress made on Open Radio Access Network (O-RAN). This includes updates to the O-CU/O-DU implementations based on the OCUDU platform, improvements to monitoring and control via the RAN Intelligent Controller (RIC), and enablement for relevant O-RAN and 3GPP interfaces (e.g., E2, F1, O1). The work also advances AI-driven RAN functionalities that enhance automation and performance. Additional developments include support for TN/NTN coexistence, transport-aware multi-homing, TSN capabilities, and low-latency uplink procedures. Progress on cloud-native deployment - particularly for the near-Real Time (RT) RIC - simplifies multi-site experimentation and supports scalable infrastructure for upcoming PoCs.

The deliverable also reports on the implementation of the core network using a modular Service Based Architecture (SBA) based on Open5GS. UNITY-6G extends this baseline with cloud-native artefacts that allow disaggregated deployment of control-plane and user-plane functions. These capabilities enable per-slice instantiation, multi-site PoC support, and low-latency paths via edge-deployed User Plane Functions (UPFs), ensuring the core network can meet the requirements of the RAN experiments and the DT-driven analysis planned for later phases.

Transport and xHaul evolution form another important part of the work reported here. We outline the design and validation plans for transport-network enhancements, including integration with TeraFlowSDN, intent-based configuration, and preliminary energy- or slice-aware features. Although some components are still under development, their design and integration roadmap are presented here.

Satellite and NTN integration complement the terrestrial side of the architecture. The deliverable covers the NTN design adopted in UNITY-6G, aligned with 3GPP Release 17 and upcoming Release 18 features. It also assesses options for TN/NTN coexistence and examines split-gNB deployment and fronthaul constraints for Split 2 and Split 7.2x, which are key considerations for PoC definition and DT modelling.

The document also presents the UNITY-6G network exposure framework, covering exposure capabilities across O-RAN, core, Wi-Fi RAN, and transport, as well as vertical-application interfaces. It describes the adoption of Network Exposure Function (NEF), Open Common API Framework (OpenCAPIF), CARMARA APIs, as well as cross-domain API management,

forming a unified exposure layer that supports third-party services and promotes interoperability across heterogeneous networks.

Finally, the deliverable introduces the initial version of the UNITY-6G DT. It outlines the DT architecture, lifecycle, internal components, and the role of AI-based data-generation models. It also specifies the datasets, simulators, monitoring resources, and generative models that will feed the DT through WP3 and WP4. These foundations contribute directly to milestones MS7 and MS8 by defining reusable datasets, structured data models, and the basis for DT-driven prediction and analysis.

Overall, this deliverable fulfils MS6, MS7, and MS8 by establishing the architectural and implementation foundation for UNITY-6G and by preparing the integration and validation activities planned for the next phases. The work reported here will be further detailed in Deliverables D3.2 and D3.3.

TABLE OF CONTENTS

1	INTRODUCTION	13
1.1	Background and motivation.....	13
1.2	Deliverable structure	14
1.3	Relation to other WPs.....	14
2	NETWORK IMPLEMENTATION	15
2.1	O-RAN.....	15
2.1.1	O-RAN architecture and components	15
2.1.2	Automated O-RAN deployment.....	18
2.1.3	Near-RT O-RAN monitoring and control	21
2.2	Core network.....	24
2.2.1	Core network implementation	24
2.2.2	Cloud-native artefacts for disaggregated deployment	25
2.2.3	Edge deployed UPF for reduced user plane latency	26
2.3	Non-3GPP RAN	27
2.3.1	O-RAN Enabled IEEE 802.11 AP architecture	27
2.3.2	Deterministic Features over Wi-Fi.....	31
2.4	Transport / xhaul.....	38
2.4.1	Transport architecture and components.....	38
2.4.2	Intent-based transport configuration	44
2.5	Satellite / ntn	47
2.5.1	NTN network architecture and components	50
2.5.2	TN/NTN coexistence and split gNB Deployment.....	51
3	NETWORK EXPOSURE	55
3.1	O-RAN.....	57
3.1.1	E2.....	57
3.1.2	Monitoring information exposure.....	59
3.1.3	API for remote xApp management.....	61
3.1.4	AI driven resource allocation API	62
3.2	Core network.....	64
3.2.1	Monitoring information exposure.....	65
3.2.2	3GPP NEF APIs Network Exposure	66
3.3	Non-3gpp RAN	72
3.3.1	Internal openwifi APIs	72
3.3.2	Exposure of Monitoring Metrics.....	73

3.3.3	Exposure of Controlling Knobs.....	75
3.4	Transport / xhaul.....	75
3.4.1	Transport Network Exposure API.....	75
3.5	Vertical App Domain (NBC).....	77
3.5.1	CAMARA API Families	78
3.5.2	Role of CAMARA in the UNITY-6G Framework	79
3.5.3	CAMARA Edge Application Management (EAM) API	79
3.5.4	Integration with the NearbyOne Platform	82
3.6	API management.....	83
3.6.1	Nearby HUB module: API manager	84
3.6.2	OpenCAPIF	85
3.7	Network Exposure consumers	86
4	DIGITAL TWIN	88
4.1	Digital Twin: architecture and specification	88
4.1.1	Digital Twin in the Unity 6G architecture	88
4.2	DT design: tools, models, datasets	91
4.2.1	Generative AI data generation model.....	91
4.2.2	Multi-Layer Perceptron (MLP)-based architectures for generating “what-if” scenarios	95
4.2.3	Network Emulation using NS3 simulator	97
4.2.4	Network topology model	101
4.2.5	Model and Dataset of Transport network power consumption.....	104
4.2.6	Modelling and a dataset of End-to-End 5G delays	108
4.2.7	Monitoring system.....	111
4.2.8	Power Grid Representation Model	116
4.2.9	Dataset of IEEE 802.11 Multi Link Operation (MLO) at 2.4 GHz and 5 GHz	117
5	CONCLUSIONS	119
6	REFERENCES	120

LIST OF FIGURES

FIGURE 2-1: OVERVIEW OF THE O-CU/O-DU IMPLEMENTATION (HIGHLIGHTED IN BLUE), PART OF THE OPEN-SOURCE OCUDU SOLUTION [5].	16
FIGURE 2-2: SYSTEM ARCHITECTURE UNDER DEMONSTRATION. NEAR-RT RIC IS A CLOUD-NATIVE ENTITY DEPLOYED IN CLUSTER C	20
FIGURE 2-3: MODIFICATION OF PRB ALLOCATION RATIO OF SLICE SST:1 USING THE PROPOSED REMOTE XAPP MANAGEMENT API.	20
FIGURE 2-4: ARCHITECTURAL CONSIDERATIONS FOR THE XAPP CONTROL LOOP.	23
FIGURE 2-5: SEQUENCE DIAGRAM OF THE NEAR-RT XAPP CONTROL LOOP GUIDED BY RAPP POLICY CONFIGURATION.	23
FIGURE 2-6: EDGE- AND CLOUD CORE-BASED UPF DEPLOYMENTS.	27
FIGURE 2-7: O-RAN ENABLED WIFI AP AND ITS INTEGRATION WITH RIC	28
FIGURE 2-8: NEAR-RT INTIALIZATION WITH OPENWIFI TELEMETRY SM	30
FIGURE 2-9: E2 OPENWIFI AP NODE REGISTRATION	31
FIGURE 2-10: XAPP PRINTINGS OF WI-FI STATISTICS	31
FIGURE 2-11: DISTRIBUTED MLD ARCHITECTURE:	32
FIGURE 2-12: MULTI-LINK OPERATION SYSTEM PERSPECTIVE	33
FIGURE 2-13: COMMUNICATION LATENCY FOR DIFFERENT SCENARIOS	36
FIGURE 2-14: COMMUNICATION RELIABILITY FOR DIFFERENT SCENARIOS	37
FIGURE 2-15: LINK RELIABILITY AT 5 GHZ	37
FIGURE 2-16: LINK RELIABILITY AT 2 GHZ	38
FIGURE 2-17: NSC ARCHITECTURE	39
FIGURE 2-18: TRANSPORT SERVICE CREATION WORKFLOW	41
FIGURE 2-19: TERAFLWSDN CONTROLLER ARCHITECTURE	42
FIGURE 2-20: OVERVIEW OF THE DIFFERENT GNB SPLITS AND SATELLITE PAYLOAD ARCHITECTURES CONSIDERED FOR UNITY-6G.	49
FIGURE 2-21: OVERVIEW OF THE NTN 5G RAN SOLUTION (SATELLITE-SPECIFIC BLOCKS HIGHLIGHTED IN PURPLE).	51
FIGURE 2-22: GENERAL ARCHITECTURE OF SPLIT 2 OPTION	53
FIGURE 2-23: GENERAL ARCHITECTURE OF SPLIT 7.2 OPTION	54
FIGURE 3-1. HIGH-LEVEL OVERVIEW OF THE UNITY-6G NETWORK EXPOSURE LAYER.	56
FIGURE 3-2: UNITY-6G NETWORK EXPOSURE FRAMEWORK.	57
FIGURE 3-3: OCUDU RAN METRICS DASHBOARD	60
FIGURE 3-4: OCUDU PERFORMANCE METRICS DASHBOARD	60
FIGURE 3-5: SWAGGER TRAINING FUNCTIONALITY	64
FIGURE 3-6: ACTIVE UE PROFILE COLLECTED BY PROMETHEUS FROM SMF INSTANCE	65
FIGURE 3-7: HIGH LEVEL OVERVIEW OF NEF EVENT EXPOSURE IN 5G CORE.	67
FIGURE 3-8: BACK-END ARCHITECTURE FOR AMF LOG AGENT	69

FIGURE 3-9:A) EXAMPLE SUBSCRIPTION POST REQUEST - CURRENT LOCATION B) EXAMPLE SUBSCRIPTION POST REQUEST - LAST KNOWN LOCATION	70
FIGURE 3-10: ASSESSIONWITHQOS API SEQUENCE DIAGRAMM	71
FIGURE 3-11:NETWORK SLICE YANG MODEL	76
FIGURE 3-12. CAMARA EAM - NEARBYONE INTEGRATION ARCHITECTURE.	83
FIGURE 3-13. NEARBY HUB API MANAGER.....	84
FIGURE 3-14: OPENCAPIF SEQUENCE DIAGRAMM	86
FIGURE 4-1: OVERALL DIGITAL ARCHITECTURE AND CORRESPONDING COMPONENTS INTERACTIONS. NORTHBOUND INTERFACES REPRESENT HOW THE APPLICATIONS INTERACT WITH THE DT WHEREAS SOUTHBOUND INTERFACES REPRESENT WHERE THE TWIN DATA COMES FROM.....	88
FIGURE 4-2: EXAMPLE OF A SINGLE INPUT-SINGLE OUTPUT DIGITAL TWIN	90
FIGURE 4-3: EXAMPLE OF COMPOSABLE DIGITAL TWIN	91
FIGURE 4-4: TTS-CGAN ARCHITECTURE.....	93
FIGURE 4-5: MTS-CGAN ARCHITECTURE.	94
FIGURE 4-6: ARCHITECTURE OF THE GENERATOR IN LTTS-GAN.....	94
FIGURE 4-7: TIMEPRISM ARCHITECTURE.....	96
FIGURE 4-8: WM-MS3M AND MPC/CEM PLANNER SYSTEM MODEL.	97
FIGURE 4-9: NS-3 AS A SCENARIO GENERATOR IN THE DIGITAL TWIN	98
FIGURE 4-10: BASE NETWORK MODEL	102
FIGURE 4-11: TOPOLOGY MODEL.....	103
FIGURE 4-12: INSTANTANEOUS DEVICE ENERGY MODEL	105
FIGURE 4-13: EXAMPLE OF AND INSTANTANEOUS DEVICE RECORD	106
FIGURE 4-14: STATIC DEVICE ENERGY MODEL	106
FIGURE 4-15: STATIC DEVICE ENERGY RECORD.....	107
FIGURE 4-16: HIERARCHICAL DEPLOYMENT OF MONITORING SYSTEMS.	113
FIGURE 4-17: CIGRE HV BENCHMARK MODEL	117

LIST OF TABLES

TABLE 2-1. DU CONFIGURATION EXCERPT	21
TABLE 2-2: SCENARIOS AND INTRODUCED TRAFFIC LOAD ON EACH LINK	35
TABLE 3-1: OVERVIEW OF THE E2SM-KPM SUPPORT OF THE SRS O-CU/O-DU SOLUTION, PART OF THE OPEN-SOURCE OCUDU PROJECT.....	58
TABLE 3-2: OVERVIEW OF THE E2SM-RC SUPPORT OF THE SRS O-CU/O-DU SOLUTION, PART OF THE OPEN-SOURCE OCUDU PROJECT.....	58
TABLE 3-3: OVERVIEW OF THE E2SM-CCC SUPPORT OF THE SRS O-CU/O-DU SOLUTION, PART OF THE OPEN-SOURCE OCUDU PROJECT.....	58
TABLE 3-4. XAPP REMOTE LIFECYCLE MANAGEMENT API DESCRIPTION	61
TABLE 3-5: END TO END RESOURCE ALLOCATION API ENPOINTS AND DESCRIPTIONS	63
TABLE 3-6: NEF MONITORING EVENT API REST-SUPPORTED METHODS	70
TABLE 3-7:3GPP TECHNICAL PROPERTIES FOR ASSESSMENTWITHQOS API	71
TABLE 3-8: EXPOSED PHYSICAL AND MAC LAYER PARAMETERS	73
TABLE 3-9: PHYSICAL AND MAC LAYER STATISTICS	73
TABLE 3-10: OPENWIFI CONFIGURABLE PHY AND MAC PARAMETERS	75
TABLE 3-11: CAMARA API FAMILIES	78
TABLE 3-12: CAMARA EAM API ENDPOINTS.....	80
TABLE 3-13: EAM API CORE DATA MODEL	81
TABLE 4-1: SIMULATION PARAMETERS USED FOR DATASET GENERATION.....	110

ABBREVIATIONS

3GPP	3 rd Generation Project Partnership
AF	Application Function
AI	Artificial Intelligence
AMF	Access and Mobility Management Function
API	Application Programming Interface
CQI	Channel Quality Indicator
CSI	Channel State Information
DSCP	Differentiated Services Code Point
DMO	Domain Management Orchestrator
ETSI	European Telecommunication Standardization Institute
FRR	Free Range Routing
GEO	Geostationary Earth Orbit
HARQ	Hybrid Automatic Repeat Request
HO	HandOver
HTTP	Hyper Text Transfer Protocol
IEEE	Institute of Electrical and Electronic Engineers
IETF	Internet Engineering Task Force
IMSI	International Mobile Subscriber Identity
IP	Internet Protocol
JSON	Javascript Object Notation
JWT	JSON Web Token
KPI	Key Performance Indicator
KVI	Key Value Indicator
L2SM	Layer 2 Service Model
L3SM	Layer 3 Service Model
LEO	Low Earth Orbit
MANO	Management and Orchestration
MCS	Modulation Coding Scheme
ML	Machine Learning
MLO	Multi-link Operation
NBI	Northbound Interface
NEF	Network Exposure Function
NSC	Network Slice Controller

NTN	Non- Terrestrial Network
OpenCAPIF	Open Common Application Programming Interface Framework
PDU	Protocol Data Unit
PoC	Proof of Concept
PRB	Physical Resource Block
QoS	Quality of Service
RAN	Radio Access Network
REST	Representational State Transfer
RFC	Request For Comments
RIC	RAN Intelligent Controller
SDN	Software-Defined Network
SINR	Signal-to-Interference-plus-Noise Ration
SMF	Session Management Function
SNR	Signal-to-Noise Ratio
SSID	Service Set Identifier
TCP	Transmission Control Protocol
TFS	TeraflowSDN
TLE	Two-Line Element
TN	Terrestrial Network
TS	Technical Specification
TSN	Time-Sensitive Networking
TTI	Transmission Time Interval
TVR	Time-Variant Routing
UDP	User Datagram Protocol
UE	User Equipment
UPF	User Plane Function
Wi-Fi	Wireless Fidelity

1 INTRODUCTION

This deliverable introduces the design, the ongoing implementation and planned integration of the different components of the UNITY-6G architecture. Namely, the Open Radio Access Network (O-RAN), including its near-real time (RT) management and control by means of the RAN Intelligent Controller (RIC), the core network, the non-3GPP RAN (IEEE 802.11 RAN) and its O-RAN enabled access points, the transport (xHaul) network and, finally, the satellite-based communication components, which rely on the Non Terrestrial Networks (NTN) features provided by the latest revisions of the 5G specifications (starting in release 17).

The network exposure is also detailed herein, following the same per-domain structure of the design and implementation description (namely, O-RAN, core network, IEEE 802.11 RAN, and transport), while also covering the vertical application domain, the management of the required Application Program Interfaces (APIs) and the consumers of the exposed metrics.

Finally, the deliverable presents the architecture and specifications of the Digital Twin application of UNITY-6G. Its design is also widely detailed, delving into the related tools, models and datasets.

It must be highlighted that the contents of this deliverable represent the results (mostly at design and specification level) and development and integration plans obtained from the execution of the first phase of Work Package (WP) 3. As such, the information included in this deliverable aims at setting a blueprint upon which the architecture of UNITY-6G will be developed and integrated into both a DT and different PoCs, with the objective of validating the advances proposed by the project. The basis presented herein will thus be iteratively refined and the initial results shown here will be extended in posterior phases of the execution of WP3 and presented in posterior deliverables. Namely, D3.2 will present the final implementation results, and D3.3 will present the integration of the developed components into the project's PoCs.

It is also worth stressing out that this deliverable addresses the following milestones: MS6 ("Define the Open RAN, xHaul Satellite, RIC components and network exposure specifications"), MS7 ("MS7 Release initial dataset and definition of data generation models using generative AI for Digital Twin") and MS8 ("Define the DT application specifications").

1.1 BACKGROUND AND MOTIVATION

Attaining the ambitious goals of UNITY-6G requires enhancing the current communication systems in several fronts towards 6G and beyond. This implies that each individual component (domain) comprising and end-to-end (E2E) communication network needs to be separately analysed and enhanced, while also considering the joint strengthening of the resulting combined network. In this deliverable we present the plans for implementing advances across all network components. The work covers the full stack of the UNITY-6G architecture, including:

- **RAN:** evolution toward a more architecture, enabling improved programmability and easier integration of heterogeneous RAN elements
- **Core network:** focusing on supporting the evolved RAN and provide efficient connectivity between the network and end users
- **Integration of non-3GPP RAN solutions:** incorporation of IEEE 802.11 and other non-3GPP technologies into the evolved and open RAN, while meeting UNITY-6G's low-latency and joint-management goals
- **Transport network evolution:** advancements toward enhanced inter-domain connectivity using Software Defined Networking (SDN), forming an essential foundation for 6G systems.
- **Extension beyond terrestrial networks (TN):** inclusion and coordinated management and optimization of TNs and NTN, enabling new and relevant 6G use cases that are not achievable with today's 5G systems.

1.2 DELIVERABLE STRUCTURE

This first section (Section 1) is focused on providing a brief introduction to the contents of the deliverable, as well as the relation of its contents to the tasks of other WPs. Section 2 describes the network implementation, that is the different components that are being developed in order to compose the UNITY-6G system and provides some initial results. The network exposure of the previously introduced components is detailed in Section 3, while providing details on the utilized interfaces, APIs, vertical application and final consumers. Section 4 introduces the DT, including its architecture, specifications and design. Finally, Section 5 closes the document by presenting the conclusions.

1.3 RELATION TO OTHER WPS

The work presented in this deliverable takes inputs from WP2, where the definition of the UNITY-6G architecture is taking place, jointly with the specification of the project's goals – in terms of KPIs and KVis. The inputs from Deliverables D2.1 [74] and D2.3 [66] are of special interest for the work presented here. Moreover, the implementation of the components takes into account the outcomes of WP4 (AI-driven distributed intelligence components) and WP5 (services and applications). The initial results of WP4 and WP5 are incorporated into the implementation of the network components and DT, while also having a major impact on the design of the network exposure solutions. Finally, the results presented here (of WP3) will form the basis of the PoCs that are being developed in WP6.

2 NETWORK IMPLEMENTATION

This section provides a detailed description of the RAN (O-RAN, IEEE 802.11) transport/xHaul, core network, and satellite layers of the UNITY-6G architecture, aiming at fulfilling MS6, while focusing on the implementation aspects of the different utilized components and its posterior integration.

2.1 O-RAN

The Open Radio Access Network (O-RAN) initiative is a global endeavour that aims at overcoming the limitations of traditional RAN solutions by adopting an open, interoperable and software-based architecture. By using open interfaces and common standards, which facilitates the collaboration of different vendors, O-RAN aims at breaking the proprietary model that has dominated the networks in previous decades. The key principles of the paradigm proposed by O-RAN can be summarized in Section 2.1.1.

2.1.1 O-RAN architecture and components

2.1.1.1 Overview of the O-RAN architecture

The O-RAN paradigm has provided 5G systems with a significant advance from the traditional RAN approach, by promoting an open, disaggregated, and intelligent architecture. Moving away from a monolithic implementation, O-RAN facilitates the distribution of the RAN functions by dividing the gNB into three main components: namely the O-RAN Central Unit (O-CU), O-RAN Distributed Unit (O-DU), and O-RAN Radio Unit (O-RU) [1], and by defining standard (open) interfaces, defined by both the 3GPP and O-RAN Alliance. Moreover, different functional splits have also been defined, such as the 7.2 which relies on the open fronthaul (OFH) specifications [2]. Disaggregation greatly enhances scalability, flexibility, and vendor interoperability, while facilitating RAN function virtualization (vRAN). Moreover, RAN softwarisation and intelligence are at the heart of O-RAN, which has also introduced the RAN Intelligent Controller (RIC) to support the integration of third-party AI-based applications for dynamic network control and optimization. The E2 interface connects the O-CU and O-DU to the near-real-time (RT) RIC to deploy xApps and implement control loops, with latencies between 10 milliseconds and 1 second [3]. Similarly, the O1 interface connects the gNB to the non-RT RIC to deploy rApps and implement non-time-sensitive tasks aimed at Service and Management Orchestration (SMO) automation [4]. Management of the RU is possible using the M-plane over the OFH or through the O1 interface (using helper elements). The O-RAN management capabilities are further expanded by the O2 interface, which connects the O-Cloud with the SMO framework, and the A1 interface which interconnects the non-RT and near-RT RICs (see Figure 2-1).

Whilst the paradigm shift promoted by O-RAN serves as a solid base upon which design the future 6G systems, several key enhancements and extensions will be analysed, designed and implemented as part of UNITY-6G to enable the envisioned UCs and fulfil newly identified requirements.

2.1.1.2 O-RAN implementation

The O-CU/O-DU solution used in UNITY-6G will be based on a software-defined radio (SDR)-based DU implementation, which is part of OCUDU [5] (previously srsRAN [6]), a carrier-grade open-source 5G O-CU/O-DU developed by SRS. OCUDU provides a complete RAN solution compliant with the 3GPP and the O-RAN Alliance specifications and includes the full L1/2/3 stack with minimal external dependencies. The software is portable across processor architectures and scalable from low-power embedded systems to cloudRAN, providing a powerful platform for mobile wireless research and development. In UNITY-6G, OCUDU constitutes the RAN platform enabling the project's PoCs. Figure 2-1 shows the main elements of the O-CU/O-DU solution provided by OCUDU.

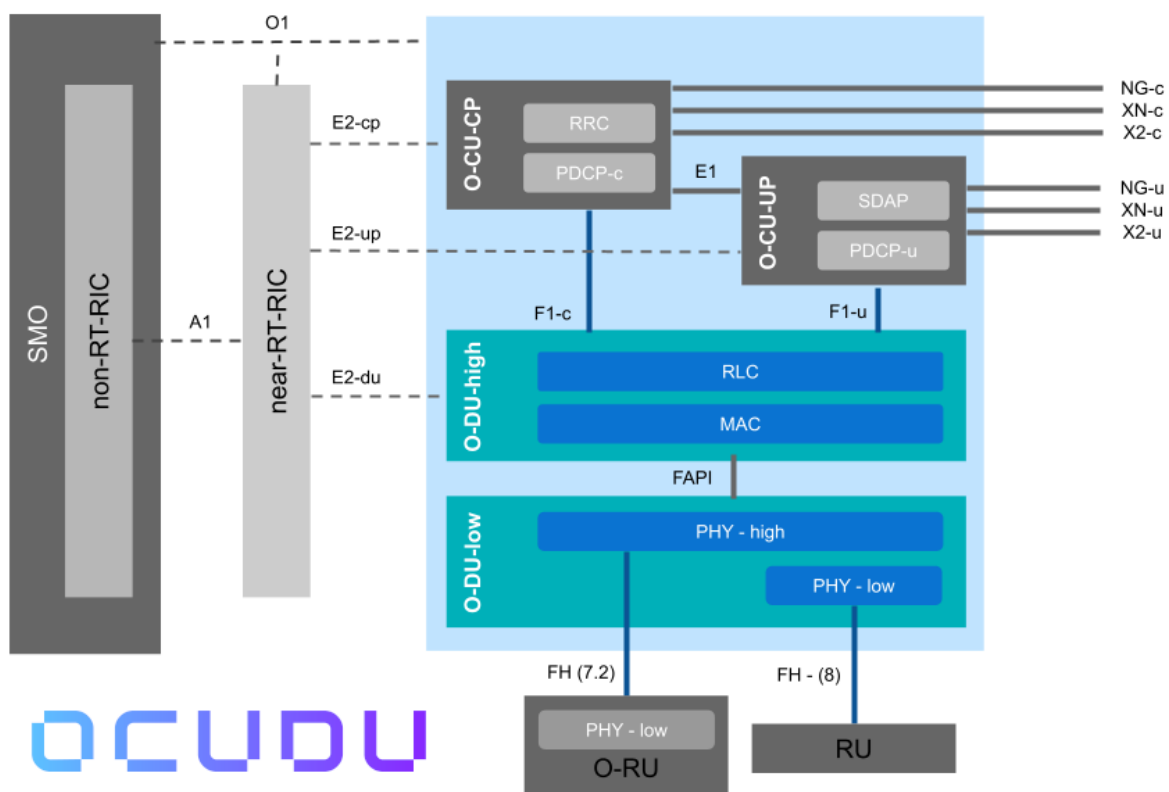


Figure 2-1: Overview of the O-CU/O-DU implementation (highlighted in blue), part of the open-source OCUDU solution [5].

The O-CU/O-DU implemented by OCUDU includes the required 3GPP and O-RAN interfaces to enable disaggregated deployments and interact with the other (third-party) O-RAN and network components of the UNITY-6G architecture. In more detail, the following interfaces are included:

- E1 interface connecting the O-CU-CP with the O-CU-UP.
- F1-c interface connecting the O-DU with the O-CU-CP.
- F1-u interface connecting the O-DU with the O-CU-UP.
- NG interface connecting the O-CU-CP/UP with the AMF/UPF.
- Xn interface enabling connection to other gNBs.
- FAPI+ interface connecting the O-DU-high with the O-DU-low.

- O-RAN Open Fronthaul (OFH) interface connecting the O-DU with the O-RU.
- E2 interface connecting the O-CU/O-DU with the near-RT RIC.
- O1 interface connecting the O-CU/O-DU with the SMO and the non-RT RIC.

The most relevant features for UNITY-6G of the O-CU/O-DU solution developed by SRS, at the time of writing and according to the release 26.04, available at [7], are listed below:

- Support for all FR1 and FR2 bands (Time Division Duplex – TDD – and Full Duplex Division – FDD – support), with up to 100 MHz of signal bandwidth in FR1 and 400 MHz in FR2.
- All PHY channels, RLC and MAC procedures.
- 4x4 MIMO in the Downlink (DL) and Uplink (UL).
- O-CU/O-DU and O-CU-CP/O-CU-UP separation.
- Slicing.
- Intra-O-DU/Intra-O-CU and Intra-O-CU handover (HO) over the NG and Xn interface (including Conditional HO).
- 7.2 Cat-A O-RU support, using the in-house OFH library.
- Near-RT RIC support over E2, including the E2 service model (E2SM) support for KPI monitoring, RAN control and cell configuration and control.
- Non-RT RIC and SMO support over the O1 interface, including cell and O-RU configuration support (the latter via the M-plane of the O-RU).

It must be underlined that the O-CU/O-DU implementation follows the current standard specifications for both components and interfaces, aligned to 3GPP release 18, and that the provided features are being augmented as part of the UNITY-6G development, adding the necessary modifications and extensions to enable the goals of the project. In this regard, strong adherence and alignment to current 3GPP and O-RAN standards is a key aspect when defining the enhancements and extensions required by UNITY-6G. Consequently, the enhancements of the O-CU/O-DU are being designed and developed with that principle in mind.

In the context of UNITY-6G, the O-CU and O-DU of OCUDU is being refined, enhanced and extended as detailed below:

- Enhanced integration to the O-RU by enabling AI-driven control over the M-plane (released under OCUDU 26.04 on March 31, 2026).
- Enhanced transport support, including load balancing, mobility and multi-homing. The latter includes adding Differentiated Services Code Point (DSCP)-based traffic-slice differentiation mechanisms (released under OCUDU 26.04 on March 31, 2026), as detailed in Section 2.1.2.
- Add low-latency support in the RAN to operate in a Time-Sensitive Networking (TSN) context, as introduced in Section 2.3. These extensions include Ethernet Protocol Data Unit (PDU) support and various enhancements to bound the round-trip time (RTT). The latter is under development at the time of writing and will include different features such as UL prescheduling, UL configured grant or Transmission Time Interval (TTI) bundling.

- Enhanced support for dynamic RAN monitoring and management, as part of the O-RAN exposure introduced in Section 3.1:
 - Implementation of the O1 interface for cell configuration and O-RU control (through the M-plane; released under OCUDU 26.04 on March 31, 2026).
 - Although other possible extensions of the E2 interface are under discussion, the development for conditional HO control has been completed (released under OCUDU 26.04 on March 31, 2026). The complete list of features currently supported under OCUDU are detailed in Section 3.1.1.
 - The RAN implementation includes a comprehensive logging system and metrics plane for instrumentation purposes. Both can be configured on a per-layer basis and are generated at run-time [8]. The metrics can be reported over websocket using JSON format, as discussed in Section 3.1.2.
- Enhanced and extended NTN features focusing on the TN/NTN integration (the previously mentioned conditional HandOver (HO) is very relevant in this context), as introduced in Section 2.5. The bulk of the NTN extensions are currently under development, with plans to first address 3GPP release 17-aligned features for both GEO and LEO support and later introduce relevant release 18 functionalities.

2.1.2 Automated O-RAN deployment

One important aspect of the O-RAN vision is the transition from monolithic hardware to cloud-native software. This approach provides flexibility, dynamicity and portability aspects which are essential for agile deployments of modern RAN. The following sub-sections dig into this aspect by presenting the solution proposed within UNITY-6G project to automate the deployment of the Near-RT RIC and its associated E2 nodes (CU and DU) as containerized microservices which could be managed within a virtualised, cloud-native infrastructure.

2.1.2.1 Automated Deployment of the O-RAN Near-RT RIC using cloud-native artefacts

Within the O-RAN architecture, the Near-RT RIC plays a central role by hosting xApps, which implement Artificial Intelligence (AI)-enabled or policy-driven RAN control logic operating at timescales below 1 second. Using the defined E2 interface, xApps can receive RAN Key Performance Measurements (KPMs) from E2 nodes (i.e., CUs and DUs) and issue control actions, enabling closed-loop optimization for mobility management, load balancing, or energy efficiency [1].

However, deploying and operating the Near-RT RIC component is complex in practice. Although the O-RAN Software Community (OSC) provides a reference implementation [9], its installation requires advanced Kubernetes expertise and intricate configuration. Additionally, xApp lifecycle management requires direct platform access and detailed knowledge of E2 procedures and the various components of the Near-RT RIC platform. These factors hinder agile and portable deployments for experimentation in multi-site testbeds emulating different locations (e.g., edge, regional or central) of available points of presence (PoPs). Each deployment still requires manual adjustments to scripts and network configurations. As a result, reproducing environments that resemble real operational setups - where flexibility and

dynamicity are essential, as envisioned in O-RAN specifications [10], becomes time-consuming, error-prone, and difficult to scale. These limitations slow down innovation in AI-enabled RAN optimisation, especially in multi-site 5G/6G testbeds where dynamic and automated orchestration is needed.

To address these limitations, within UNITY-6G project, a purpose-built deployment blueprint has been developed [11]. This blueprint consists of a Helm chart embedded into an ETSI NFV Network Service (NS) package, which can be deployed with a management and orchestration (MANO) platform, like Open Source MANO (OSM), an example of the ones identified for the Domain Management Orchestration (DMO) component of the UNITY-6G architecture. Taking as a reference and enhancing the work in [12], the main features of the designed blueprint are:

- Blueprint design adapted to allow the deployment of different instances of the Near-RT RIC in any PoP without requiring shell access to the servers, as is currently required by [12] and the original OSC Near-RT RIC software [9].
- Inclusion of deployment-time adaptations to the original code implementation in [12] to fit the target Kubernetes environment (e.g., network configuration) where it is executed.
- Enhanced code to dynamically configure and manage the communication ports between Near-RT RIC pods and running xApps to automate the simultaneous execution of the same and different xApp instances.
- The pod responsible for executing the xApps, referred to as the “xApp executor pod”, is equipped with a Python-based REST Application Programming Interface (API) and a sidecar database container to enable remote xApp lifecycle management. Specifically, the defined API exposes endpoints for file management, process execution control, and data logging retrieval. More on this in Section 3.1.3.

Figure 2-2 shows the experimental setup that was used to validate the designed blueprint for the OSC Near-RT RIC and its automated deployment. This setup is distributed across four commercial off-the-shelf servers. One server acts as a DMO with capabilities to orchestrate cloud-native mobile core and O-RAN artifacts based on open-source software (i.e., Open5Gs and OCUDU). This DMO is powered by an instance of ETSI Open-Source MANO (OSM). Three servers act as PoPs, each running a single-node Kubernetes cluster. This setup represents the different locations managed by Mobile Network Operators (MNOs): central, regional, and edge cloud PoPs, where entities of mobile core and RAN (CU/DU, Near-RT RIC) can be distributed as required according to the O-RAN scenarios defined in [10]. The RAN equipment is connected to the edge cloud, where the Falcon RX/G O-RAN switch and Precision Time Protocol (PTP) grandmaster provide synchronization between the edge cloud site and the Radio Unit (RU) LITEON FlexFi FF-RFI078I4. Two smartphones (Realme 9 and OnePlus 8 Pro), accessed via a remote desktop tool (e.g., AnyDesk), are used to generate traffic in the different slices.

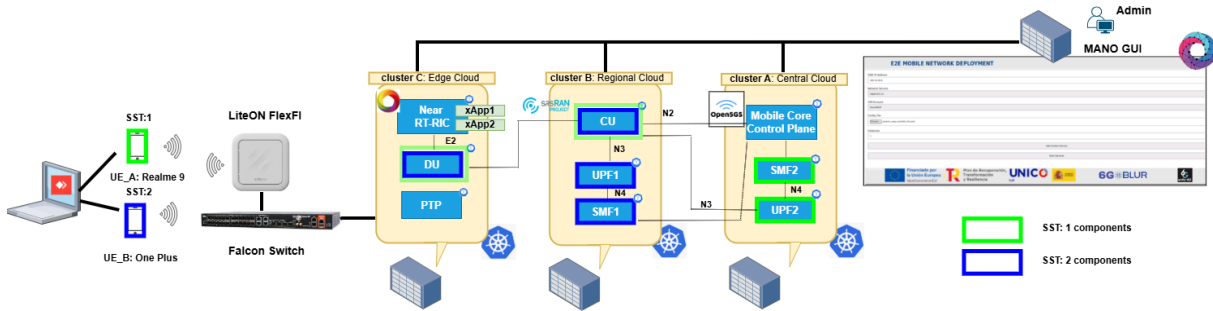


Figure 2-2: System Architecture under Demonstration. Near-RT RIC is a cloud-native entity deployed in Cluster C

Figure 2-3 shows an example of the use of the remote management xApp API enabled after the cloud-native deployment of the OSC Near-RT RIC and based on available capabilities of OCUDU software, as detailed in Section 3.1.1. After establishing a downlink iperf session between the UPF of Slice 1 and UE attached of slice 1, the remote management xApp API allows the execution of a Cell Configuration Control (CCC) xApp capable of controlling the number of Physical Resource Blocks (PRBs) assigned in the configuration of the DU. In particular, the observed performance variation corresponds to changes in the PRB ratio, from 25% to 75%, issued through calls to the designed API calls that triggers the execution of the designed CCC xApp.

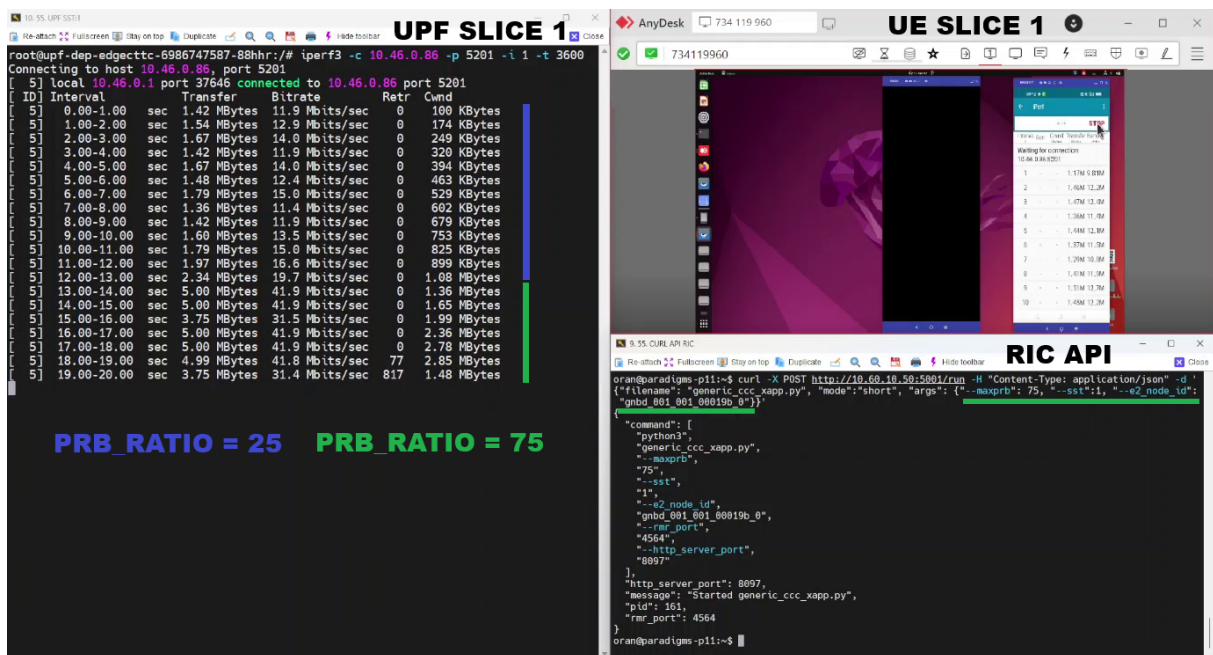


Figure 2-3: Modification of PRB allocation ratio of Slice SST:1 using the proposed remote xApp management API.

Summarising, this contribution lowers the operational barrier for using the OSC Near-RT RIC and accelerates research on AI-driven RAN control, as proposed by UNITY-6G project. It enables rapid prototyping of closed-loop control algorithms in the O-RAN domain, seamless multi-site experimentation, scalable deployment of xApps in cloud-native RAN environments. Overall, this work contributes significantly to enabling practical adoption of the O-RAN architecture in real testbeds and research projects, like the O-RAN domain managed and orchestrated in the UNITY-6G architecture.

2.1.2.2 Automated Deployment of cloud-native independent CU and DU components enabling per-slice DSCP-based midhaul traffic separation

As covered in Section 2.1.1, the O-RAN architecture proposes the separation of the gNB into CU and DU entities. Such separation opens the door to multiple deployment scenarios, like those presented in [10]. Multiplexing gains and deployment flexibility can be obtained when a single CU can manage multiple DU instances connected to different Radio Units. Indeed, the deployment flexibility can be exploited using independent cloud-native artefacts for CU and DU, as depicted in Figure 2-2. Such artefacts were proposed in [14] and [15], and are publicly available in [16]. These artefacts embed containerised implementation of OCUDU software into Helm charts and ETSI NFV-compliant MANO Network service descriptors, delivering the possibility of creating reproducible, portable, and dynamically reconfigurable instances of CU and DU entities. Through parametrisation capabilities built into both the Helm templates and the OSM descriptors, deployment characteristics, IP address configuration of the CU or DU, connection to different core instances or near-RT RIC, definition of network slice identifiers can be customised at instantiation time, so multiple instances of the same software belonging to different service request can coexist in the same infrastructure.

Within UNITY-6G project, the CU and DU artefacts have been enhanced to exploit a new feature of OCUDU code allowing the use of differentiated sockets for the F1u interface between CU and DUs for each slice served by the DU. Based on the use of Differentiated Services Code Point (DSCP) values, a 6-bit field in IP packets, DSCP codes can provide per-slice differentiation of midhaul traffic. Thus, in deployments where DU and CU instances are deployed in different PoPs, different transport policies (e.g. different path) could be applied for the traffic of different slices for a more fine-grained control required in complex scenarios. Table 2-1 shows an excerpt of the configuration template used in the DU artefacts to configure multiple slices, with different PRB allocation and different DSCP codes. This configuration is then later formatted into the appropriate configuration to run OCUDU CU or DU components.

Table 2-1. DU Configuration excerpt

```
snssais:
- sst: '1'
  sched_cfg:
    min_prb_policy_ratio: 0
    max_prb_policy_ratio: 30
  five_qi: 9
  dscp: 6
- sst: '2'
  sched_cfg:
    min_prb_policy_ratio: 0
    max_prb_policy_ratio: 70
  five_qi: 9
  dscp: 7
```

2.1.3 Near-RT O-RAN monitoring and control

In the O-RAN architecture, Near-RT RIC constitutes a fundamental module equipped with intelligent capabilities to enable real-time optimization and resource allocation. As illustrated in

the Figure 2-4, it interfaces with distributed RAN components through the E2 interface, collecting continuous network metrics and enforcing decisions. The key components hosted in Near-RT RIC are [22] :

- *List of xApps*: It consists of repository of all available xApps developed by multiple vendors, which can be dynamically activated or instantiated based on network conditions and operator objectives. For example, a power control or a bandwidth allocation xApp may be active at a certain period for purposes of maximizing the users' throughput and system EE, respectively.
- *Active xApps*: These xApps are active and use RAN metrics and reports stored in the RIC Database with the aim to provide optimization suggestions or control decisions (such as resource block allocation, RU power control, or interference mitigation). For this purpose, they may leverage supervised, unsupervised, or DRL algorithms.
- *RIC Database*: Here the RAN measurements and potential historical performance data gathered from the network through the E2 Service Model (E2SM) interface are maintained. These data are accessible as E2SM Reports to the active xApps.
- *RIC Databus*: Communication between internal components of the Near-RT RIC –usually concerning data provisioning- is facilitated by this databus. Specifically, the active xApps are subscribed to the RIC Databus for ensuring and securing efficient data streaming of the RAN metrics.
- *Action Taker*: Based on the decisions suggested by active xApps (or the conflict resolution outcome for conflicting xApps), the action taker module translates these outputs in applicable actions and transmits them via E2SM Control messages to the RAN nodes.

The radio environment that the Near RT RIC interacts with, comprises of multi-vendor E2 RAN nodes (UEs, RUs, and DUs) of the Open Front Haul interface, continuously producing measurements related to various network conditions, such as traffic demand, mobility, channel losses, UE handovers, transmitting power levels and PRBs. These RAN metrics are fed back to the RIC Database and collected as E2SM Report messages over E2. This closed-loop operation enables continuous monitoring of xApps, and real-time optimization of network performance in a disaggregated O-RAN ecosystem.

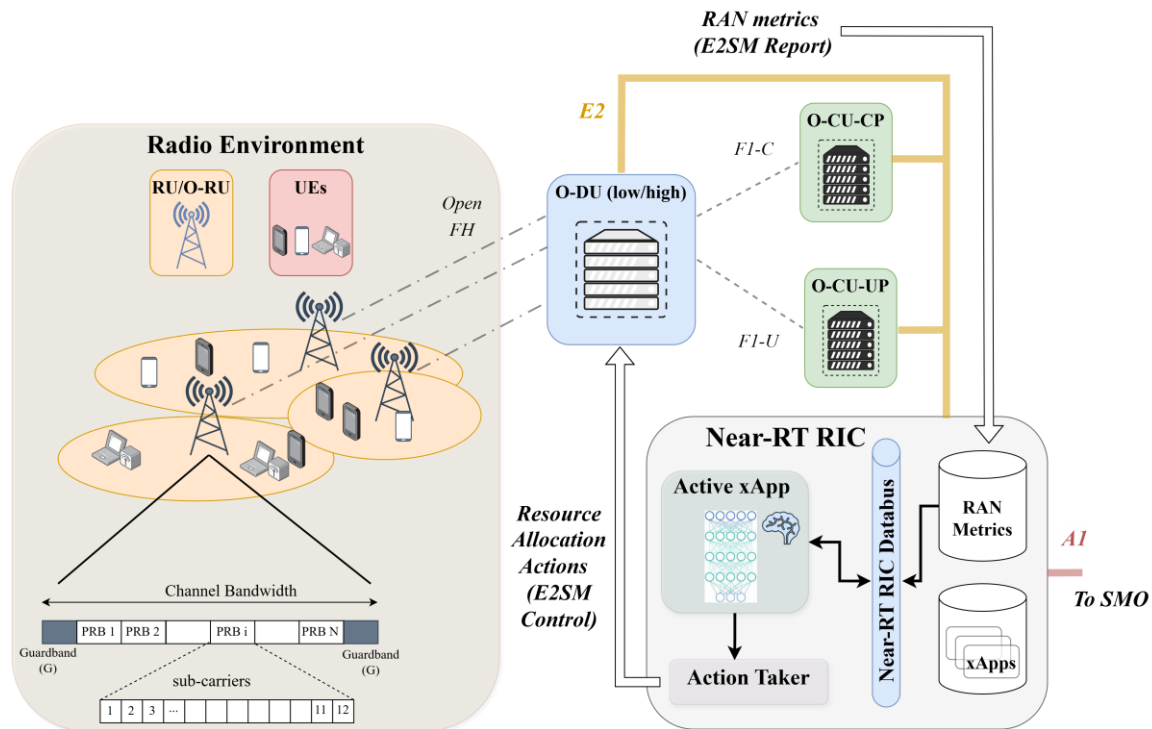


Figure 2-4: Architectural considerations for the xApp control loop.

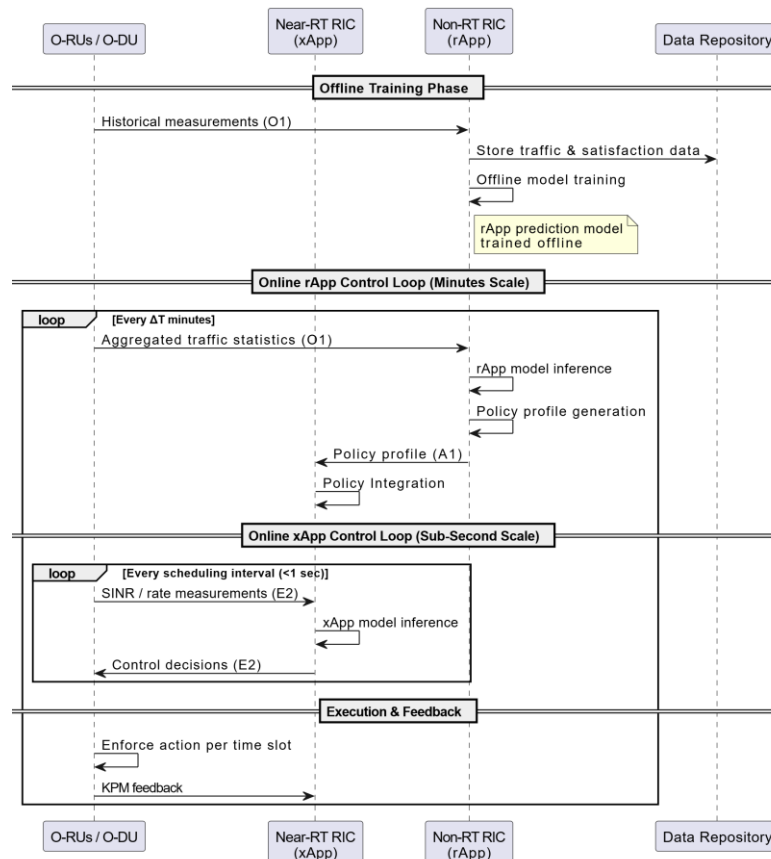


Figure 2-5: Sequence diagram of the Near-RT xApp control loop guided by rApp policy configuration.

The overall Near-RT RIC control architecture functions as a hierarchical closed-loop system across multiple timescales, combining the long-horizon intelligence of the rApp with the near-

real-time decision-making of the xApp. Figure 2-5 presents the associated sequence diagram and distinguishes the offline and online stages of operation. The process typically begins with an offline training stage at the Non-RT RIC, where the rApp gathers historical measurements through the O1 interface. These data are stored and exploited to train a predictive model offline, such as a traffic forecasting model that can later support proactive tuning of xApp policies. Because this stage is completely separated from live RAN operation, it does not affect real-time network responsiveness. During actual network operation, the rApp runs an online control loop on a minutes-scale basis. Using newly acquired measurements, it applies the trained model to infer future traffic behaviour. The resulting predictions, together with SLA-driven policies, are then used to generate the policy profile for the xApp. This profile is transmitted to the Near-RT RIC over the A1 interface and serves to parameterize the xApp logic, rather than issuing direct scheduling commands [23].

Configured by these rApp-provided policy settings, the xApp operates at a sub-second timescale and constitutes the inner loop of the control framework. At every scheduling interval, it retrieves KPMs through the E2 interface, such as instantaneous SINR values and achievable data rates for active UEs. Based on this real-time information, the xApp performs inference to determine the appropriate control actions. For example, it may produce PRB-to-UE scheduling decisions. These actions are then sent to the O-DU via the E2 interface. The O-DU subsequently applies the received decisions at each time slot and enforces them across the associated O-RUs. At the same time, execution-related KPM feedback, including achieved throughput and user satisfaction metrics, is continuously returned to the RIC layers, thereby completing the control loop. In this way, the nested rApp/xApp architecture enables fast adaptation to short-term network fluctuations, such as channel dynamics, while also benefiting from longer-term traffic awareness within the O-RAN system.

2.2 CORE NETWORK

The core network plays a central role in the UNITY-6G architecture, as it enables the service-based connectivity, while providing low-latency and high-speed. Its implementation follows the open and SDN-based approach of the RAN, as it will be detailed below.

2.2.1 Core network implementation

Open5Gs [24] is an open-source implementation of the 5G core network, aligned to 3GPP release 17 and actively maintained by a large developer community. It provides a full set of 5G Core Network Functions (NFs), making it suitable for both research environments and real testbed deployments. In UNITY-6G, Open5GS is used as the baseline core, as it fits naturally with the project's NFV-oriented approach: its components can be easily containerised (e.g., via Docker) and orchestrated in distributed setups and is already widely adopted in combination with OCUDU.

A key feature of Open5Gs is its implementation of the 5G Service Based Architecture (SBA). In SBA, core network functions interact through well-defined service interfaces rather than via static point-to-point connections. This design makes the core more modular, scalable, and flexible – allowing control-plane NFs to discover and communicate with each other

dynamically, while the user-plane ones can be deployed closer to the edge when low latency is required. This architectural model is especially relevant for UNITY-6G, as it supports distributed deployment across multiple sites and aligns with the project's goals for experimentation, openness and programmability.

Open5Gs, thus, includes both the user plane and the control plane. The user plane comprises a single function:

- **User Plane Function (UPF):** carries user data packets between the gNB and the cabled network.

The most relevant functions of the control plane include:

- **Network Repository Function (NRF):** register all control plane functions and help them discover the other available core functions.
- **Access and Mobility Management Function (AMF):** handles user registration, reachability and mobility management.
- **Session Management Function (SMF):** manages active PDU sessions, including establishment and release.
- **Network Slice Selection (NSSF):** enables selecting network slices for users, allowing to optimize the usage of resources.
- **Policy Control Function (PCF):** provides a unified framework for controlling network behaviour, including Quality of Service (QoS) policies.
- **Authentication Server Function (AUSF), Unified Data Management (UDM) and Unified Data Repository (UDR):** management of user authentication and subscriber profile storage.

2.2.2 Cloud-native artefacts for disaggregated deployment

The introduction of cloud-native principles into the management and orchestration of next-generation mobile networks, together with the SBA of mobile core architecture, allows the design of mobile core deployment strategies with a high degree of modularity, automation, and flexibility. Within UNITY-6G project, and more specifically, the mobile core domain will use the artefacts developed in [13], [14], [15] (publicly available at [16]) to perform the modular deployment of mobile core instances based on the open-source software Open5GS, explained in previous Section 2.2.1. Similarly to the approach followed for O-RAN components, these artefacts embed containerised implementation of Open5GS into Helm charts and ETSI NFV-compliant MANO Network service descriptors, delivering the possibility of creating reproducible, portable, and dynamically reconfigurable mobile core instances suitable for multi-site research environments and scenarios with dedicated instances of 5G/6G networks, such as those explored by UNITY-6G project. The resulting approach achieves a level of automation and flexibility beyond the static behaviour typical of many cloud-native core deployments in the literature, as described in [15].

More specifically, these are the set of produced Helm chart allowing a modular deployment of the mobile core based on the number of configured slices. For the control plane, a dedicated blueprint was created to encapsulate functions such as the AMF, AUSF, BSF, NRF, UDM,

UDR, and PCF, with each function executed as an independent Kubernetes pod. This blueprint provides the centralised management point that can be deployed, for example, at a cloud PoP. Among other parameters, the configuration of each different slice is provided in the *values.yaml* file of the Helm Chart, and this configuration is later injected into the corresponding network functions, like the AMF. In parallel, the Session Management Function (SMF) was decoupled from the control-plane blueprint to enable per-slice instantiation and flexible configuration of slice-specific behaviour, offering a granularity impossible to achieve in monolithic deployments. Finally, the User-Plane Function (UPF) was developed as a fully independent blueprint, capable of being deployed on-demand coupled with the corresponding SMF instance. This architecture enables the dynamic creation, removal, and relocation of SMF/UPF instances based on the amount of configured/pre-provisioned slices, which can be particularly advantageous in non-public networks with edge Points of presence. All blueprints expose the standard 3GPP interfaces necessary for inter-NF connectivity (such as N2 and N3/N4 interfaces) via Kubernetes LoadBalancer services to ensure compatibility with the RAN and support distributed multi-cluster or multi-domain operation, as considered in UNITY-6G architecture. Each Helm chart was then encapsulated within ETSI OSM Virtual Network Function (VNF) and Network Service (NS) packages. This integration allows the mobile core to be instantiated, modified, or terminated through automated MANO workflows without requiring a direct interaction with Kubernetes clusters. Through parametrisation capabilities built into both the Helm templates and the OSM descriptors, deployment characteristics, IP address configuration for the network functions, definition of network slice identifiers, can be customised at instantiation time. This approach enables the coexistence of multiple mobile core network instances in the same PoP, each one, serving different configurations originated from different intents. Moreover, the Service Adapter module of the mobile core Domain will need to consider this artefact structure and the configuration of requested mobile core instance to issue the corresponding requests to the associated Domain Management Orchestrator. That is, for a request demanding a single slice core instance, it translates into the instantiation of three artefacts (i.e., control plane, SMF and UPF instance). However, if an additional mobile core instance with two slices is needed, a total of five requests are done (i.e., control plane, one SMF and one UPF instances for slice 1, and another SMF and UPF instances for slice 2).

2.2.3 Edge deployed UPF for reduced user plane latency

In traditional 5G deployments, the UPF is co-located with the rest of the core network functions in a centralised cloud environment. While this simplifies management, it introduces additional latency on the data path, as all user traffic must traverse the network to and from the cloud before reaching its destination. To address this limitation and to reduce the user plane latency for time sensitive traffic flows in NPN deployments, we adopted a split UPF architecture, placing a dedicated UPF instance directly at the edge network, physically close to the end devices. For that, the open source open5GS platform has been used.

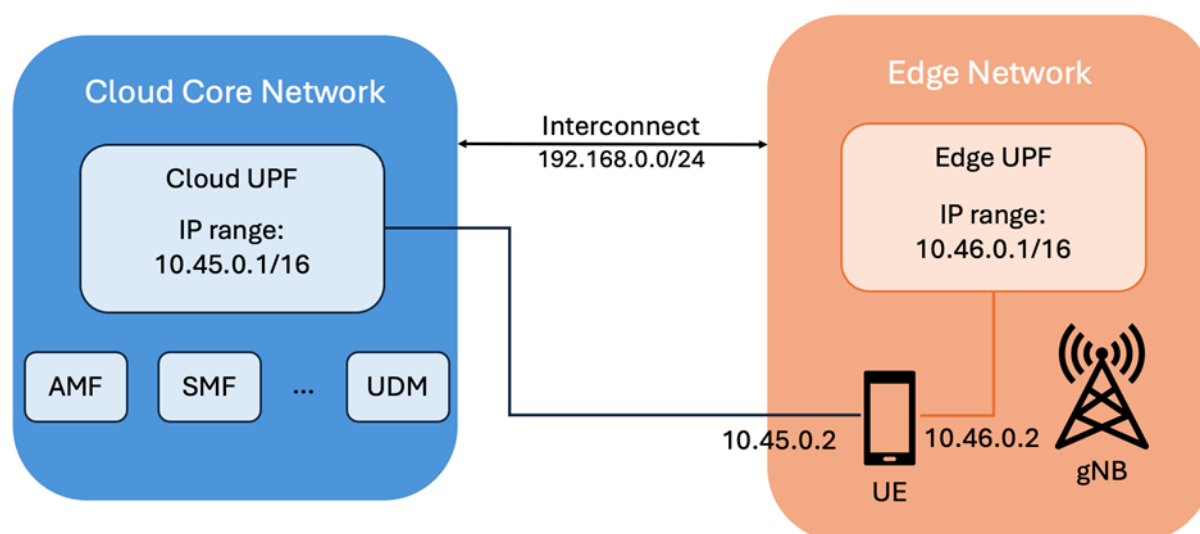


Figure 2-6: Edge- and Cloud Core-based UPF deployments

As shown in Figure 2-6, this architecture comprises two distinct segments. The Cloud Core Network hosts the standard 5G control core functions such as AMF, SMF, UDM, etc., along with a Cloud UPF serving subnet 10.45.0.1/16. The Edge Network, located closer to the radio access equipment and the users, contains a dedicated Edge UPF that serves subnet 10.46.0.1/16. By directing relevant traffic through the Edge UPF rather than the Cloud UPF, the data plane path is significantly shortened: packets no longer need to travel to the centralised cloud and back but are instead processed locally at the edge. This split UPF deployment contributes to cellular NPN meeting the stringent latency requirements imposed by the TSN context.

2.3 NON-3GPP RAN

In this section we will focus on the key innovations that UNITY-6G project brings to non-3GPP radio access network (RAN) for future 6G networks. First, we show the O-RAN enabled IEEE 802.11 access points (APs) architecture. This is achieved utilizing openwifi [25] platform that supports fine-grained exposure of monitoring information (down to physical layer) as well as controlling knobs for management of radio access technologie(s) (RATs) behaviour. Next to this, we describe the design of the features that support deterministic low-latency communication over non-3GPP RAN. The focus will be given to Multi-Link Operation (MLO) for IEEE 802.11 RAT, as one of the most promising feature that is designed in UNITY-6G.

2.3.1 O-RAN Enabled IEEE 802.11 AP architecture

The O-RAN-enabled IEEE 802.11 AP is shown in Figure 2-7. The architecture is composed of three main blocks: the E2 agent that terminates the E2 Service Model (E2SM) and E2 Access Point (E2AP) protocols, the Distributed Applications (dApps) that reside in the APs, and the defined Wi-Fi service models, while xApps reside in edge or cloud in the near-RT RIC. The novel part of this architecture in UNITY-6G is E2 agent that connects to the IEEE 802.11 AP, and the E2SM for openwifi.

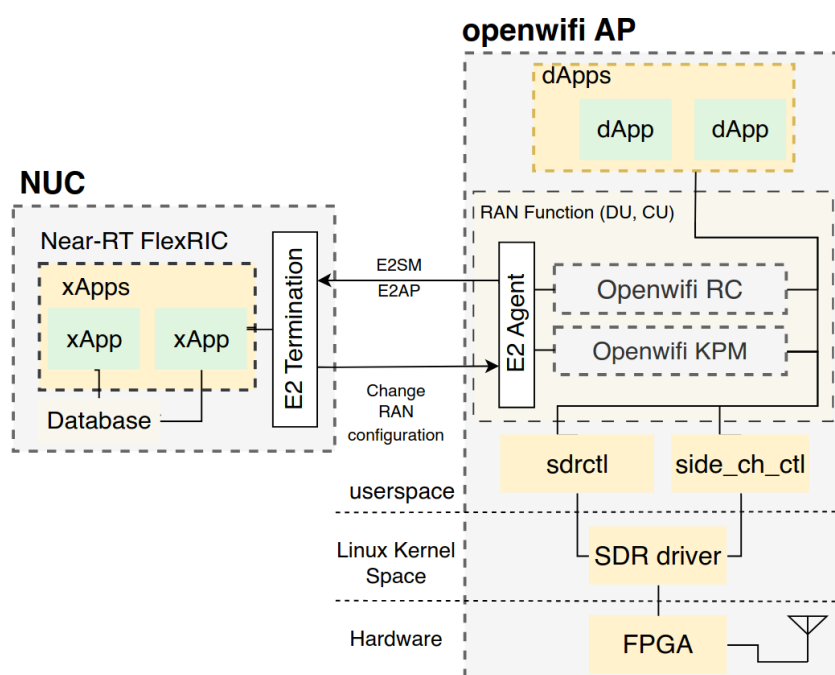


Figure 2-7: O-RAN Enabled WiFi AP and its integration with RIC

2.3.1.1 E2 Agent

The E2 Agent serves as the primary interface between the Near-RT FlexRIC and the openwififi AP hardware. By terminating the E2SM and E2AP protocols, the agent abstracts the complexity of the underlying MAC and PHY layers. Specifically, it facilitates O-RAN compliance for non-3GPP hardware by mapping openwififi KPM and Radio Control (RC) modules to standard E2 service models, enabling seamless telemetry and configuration changes. The integration leverages two specialized modules within the RAN function: the openwififi Key Performance Metric (KPM) and openwififi RC. The KPM module serves as the telemetry interface, reading performance variables from the hardware to support O-RAN observability. Conversely, the RC module facilitates closed-loop control by executing configuration updates directed by the Near-RT RIC. Together, they map the openwififi-specific userspace and kernel operations to standardized E2SM definitions, enabling vendor-agnostic RAN management.

2.3.1.2 xApps and dApps

In the proposed architecture, the Near-RT FlexRIC serves as the hosting platform for xApps, which are microservices designed to manage radio resources in near-real-time. These xApps communicate with the E2 termination agent to exchange information with the underlying RAN. A critical feature of this specific implementation is the support for dApps, which reside inside the openwififi AP. The system maintains an internal interface that allows dApps to interact with the E2 agent. This internal coupling enables dApps to perform rapid, localized decision-making while staying synchronized with the broader network policies defined by the xApps in the RIC. The ecosystem supports a diverse range of application types tailored for different network objectives. xApps typically handle near-real-time optimizations, such as mobility management, load balancing, and interference coordination across multiple access points. In contrast, dApps are designed for high-frequency tasks that require immediate execution within the user space

of the AP. Examples include per-packet scheduling, rapid link adaptation updating Modulation and Coding Scheme (MCS), Channel State Information (CSI) compression, localized MAC-layer security filtering, and more. By splitting logic between xApps and dApps, the system achieves a balance between global network intelligence and the low-latency responsiveness required for high-performance wireless environments. Since openwifi supports TSN features, the integration with TSN is facilitated through the coordination of the openwifi telemetry monitoring SM and the underlying hardware abstraction. Within this framework, specialized xApps or dApps can enforce deterministic latency and jitter requirements by interacting with the *side_ch_ctl* and *sdrctl* modules to set transmission schedules. These control paths allow the system to prioritize time-critical traffic at the Linux Kernel Space and SDR driver levels. By mapping TSN configuration parameters to the E2SM service models, the RIC can orchestrate precise resource slicing and hardware-level synchronization across the MAC and PHY layers, ensuring that the openwifi platform meets the rigorous demands of industrial or mission-critical applications

2.3.1.3 IEEE 802.11 Service Models

Aligned with the O-RAN SM, openwifi facilitates the collection of KPM telemetry data and the implementation of RC mechanisms. The following sections describe the integration and capabilities of these models in the context of openwifi.

E2SM-KPM-openwifi: The E2SM-KPM-openwifi service model provides fine-grained performance and MAC and PHY layer measurements. It exposes detailed CSI information on a per-packet basis, capturing physical link characteristics through subcarrier pilot sequences. Transmission statistics include the total number of transmitted data packets and the corresponding number of failures, defined as packets for which no Acknowledgment (ACK) is received. In addition, retransmission statistics categorize transmitted packets and failures according to the number of retransmission attempts. The model reports real-time TX and RX MCS values for data, management, and ACK packets, typically represented in 10×Mbps units. Management frame statistics are also collected, including total transmissions and failure rates for frames requiring acknowledgements. On the reception side, packet counters provide the total number of received data, management, and ACK frames, further distinguishing between successful receptions and those failing the Frame Check Sequence (FCS) check. Furthermore, Automatic Gain Control (AGC) measurements are available for different packet types (data, management, and ACK), enabling analysis of receiver behaviour. Queue-level metrics are provided for both MAC-priority queues and PHY-layer hardware queues, including packet counts, interrupt triggers, and queue stop and wake-up events. Finally, the Packet Error Rate (PER) can be derived as a high-level metric, computed as the ratio of failed packet deliveries to the total number of transmitted packets.

E2SM-RC-openwifi: The E2SM-RC-openwifi service model enables runtime control over multiple MAC and PHY-layer parameters. It allows manual configuration of the maximum number of hardware retransmissions for transmitted packets. Layer two ACK behaviour can also be controlled, including enabling or disabling automatic ACK transmission, which is particularly relevant for monitor or packet injection modes. Low-level MAC timing parameters, such as Network Allocation Vector (NAV), Distributed Coordination Function Inter-Frame Space (DIFS), Extended IFS (EIFS), and Short IFS (SIFS) durations, can be tuned to modify

channel access behaviour. The contention window configuration supports adjustment of the minimum (CWmin) and maximum (CWmax) values for the Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) backoff algorithm across the four access categories. At the PHY layer, the model supports manual override of the receiver gain by disabling the automatic gain and setting a fixed gain value in dB. Transmit power can be controlled through fine-grained attenuation settings, typically expressed in units of 0.001 dB. Antenna selection is software-defined, enabling switching between available radio frequency (RF) ports. Additional controls include setting the receiver sensitivity threshold in dBm to filter weak signals, directly configuring the RF center frequency to override standard channel tuning, and manually selecting the transmission bitrate and modulation scheme for injected or generated traffic.

2.3.1.4 Initial results for O-RAN enabled AP

To test our implementation in a real-world scenario, we created a small setup consisting of one RIC node and two openwifi nodes: an O-RAN enabled AP and a station. For the RIC, we extended FlexRic [58] near-RT RIC with Wi-Fi SM introduced in Section 2.3.1.3. We implemented a monitoring xApp and the E2 openwifi agent to retrieve Wi-Fi-specific monitoring parameters. At this stage, only the E2SM-KPM- openwifi SM is implemented. RIC and xApp run in the same physical node, an Intel Next Unit of Computing (NUC) mini-PC with Intel Core i7-1260P CPU. The E2 termination agent in openwifi runs on the ZCU-102 board, which runs the Raspbian 11.2 distribution in a 32-bit host architecture. Some modifications regarding the host architecture need to be made to the original code from FlexRic, in addition to implementing the E2 openwifi agent.

The evaluation of the implementation is done as follows. First, we start the near-RT RIC in the NUC with IP 10.42.0.103, which supports the openwifi telemetry SM (circled in red in Figure 2-8). Next, the openwifi E2 termination agent registers with the near-RT RIC as node ID 100 with RAN type openwifi (circled in red in Figure 2-9). Once registration is complete, we launch the xApp that subscribes to the monitoring data from the openwifi O-RAN-enabled AP. In the background, traffic is generated between the openwifi AP and the openwifi station. The monitoring xApp periodically receives updates every 10 ms on the transmitted data, including layer 2 ACK, layer 2 ACK failures, received data in total, and failed received data. These metrics allow us to continuously observe IEEE 802.11 data exchange behaviour through the O-RAN telemetry pipeline (circled in red in Figure 2-10).

```
bigmaster@bigmaster:~/ric_controller/flexric/build$ ./examples/ric/nearRT-RIC -a 10.42.0.103
[UTIL]: Setting the config -c file to /usr/local/etc/flexric/flexric.conf
[UTIL]: Setting path -p for the shared libraries to /usr/local/lib/flexric/
[NEAR-RIC]: nearRT-RIC IP Address = 10.42.0.103, PORT = 36421
[NEAR-RIC]: Initializing
[NEAR-RIC]: Loading SM ID = 2 with def = ORAN-E2SM-KPM
[NEAR-RIC]: Loading SM ID = 142 with def = MAC_STATS_V0
[NEAR-RIC]: Loading SM ID = 146 with def = TC_STATS_V0
[NEAR-RIC]: Loading SM ID = 149 with def = WIFI_TELEMETRY_V1
[NEAR-RIC]: Loading SM ID = 145 with def = SLICE_STATS_V0
[NEAR-RIC]: Loading SM ID = 148 with def = GTP_STATS_V0
[NEAR-RIC]: Loading SM ID = 143 with def = RLC_STATS_V0
[NEAR-RIC]: Loading SM ID = 144 with def = PDCP_STATS_V0
[NEAR-RIC]: Loading SM ID = 3 with def = ORAN-E2SM-RC
[iApp]: Initializing ...
[iApp]: nearRT-RIC IP Address = 10.42.0.103, PORT = 36422
```

Figure 2-8: Near-RT initialization with openwifi telemetry SM

```
[NEAR-RIC]: Initializing Task Manager with 2 threads
[E2AP]: E2 SETUP-REQUEST rx from PLMN 505. 1 Node ID 100 RAN type ran_openwifi
[NEAR-RIC]: Accepting RAN function ID 2 with def = ORAN-E2SM-KPM
[NEAR-RIC]: Accepting RAN function ID 3 with def = ORAN-E2SM-RC
[NEAR-RIC]: Accepting RAN function ID 142 with def = MAC_STATS_V0
[NEAR-RIC]: Accepting RAN function ID 143 with def = RLC_STATS_V0
[NEAR-RIC]: Accepting RAN function ID 144 with def = PDCP_STATS_V0
[NEAR-RIC]: Accepting RAN function ID 145 with def = SLICE_STATS_V0
[NEAR-RIC]: Accepting RAN function ID 146 with def = TC_STATS_V0
[NEAR-RIC]: Accepting RAN function ID 148 with def = GTP_STATS_V0
[NEAR-RIC]: Accepting RAN function ID 149 with def = WIFI_TELEMETRY_V1
```

Figure 2-9: E2 openwifi AP node registration

```
[xApp][DEBUG]: raw RIC_INDICATION RAN_FUNC_ID=149 RIC_REQ_ID=1
[xApp][Wi-Fi]: INDICATION rx device_id=12345 tx_total=39498 tx_fail=358 rx_total=39694 rx_fail=0 uptime=660 RIC_REQ_ID=1
Wi-Fi sqlite not implemented
[Wi-Fi xAPP][DEBUG]: Callback triggered. ind.type=8
[Wi-Fi xAPP]: Received Wi-Fi indication
[Wi-Fi xAPP]: Device ID: 12345
[Wi-Fi xAPP]: tx_data_need_ack_total: 39498
[Wi-Fi xAPP]: tx_data_need_ack_fail: 358
[Wi-Fi xAPP]: rx_data_total: 39694
[Wi-Fi xAPP]: rx_data_fail: 0
[Wi-Fi xAPP]: Uptime: 660 seconds

[xApp][DEBUG]: raw RIC_INDICATION RAN_FUNC_ID=149 RIC_REQ_ID=1
[xApp][Wi-Fi]: INDICATION rx device_id=12345 tx_total=39505 tx_fail=358 rx_total=39701 rx_fail=0 uptime=660 RIC_REQ_ID=1
[Wi-Fi xAPP][DEBUG]: Callback triggered. ind.type=8
[Wi-Fi xAPP]: Received Wi-Fi indication
[Wi-Fi xAPP]: Device ID: 12345
[Wi-Fi xAPP]: tx_data_need_ack_total: 39505
[Wi-Fi xAPP]: tx_data_need_ack_fail: 358
[Wi-Fi xAPP]: rx_data_total: 39701
[Wi-Fi xAPP]: rx_data_fail: 0
[Wi-Fi xAPP]: Uptime: 660 seconds
```

Figure 2-10: xApp printings of Wi-Fi statistics

2.3.2 Deterministic Features over Wi-Fi

In this section we will describe the features that we developed to achieve deterministic communication over Wi-Fi. The main innovative feature (in addition to support for time synchronization and scheduling) is MLO that utilizes multiple links to maintain latency as well as reliability over wireless links. We achieved MLO by utilizing multiple openwifi nodes interconnected via a time-sensitive switch. In the following subsections we will describe the distributed MLO device (d-MLD) architecture and the results achieved utilizing MLO as a mechanism to achieve frame replication and elimination for reliability (FRER) over wireless.

2.3.2.1 Distributed Multi-Link Operation Device Architecture

To support MLO operation and end-to-end FRER, a distributed MLD (d-MLD) architecture is designed. The control functions are split among wireless nodes and the interconnected TSN switch. The low MAC is implemented in FPGA in the openwifi platform and manages the low-level parameters such as channel access mechanism (back-off counter, layer two retransmissions), wireless TSN (WTSN) time-based scheduling, and over-the-air time synchronization [26]. Wireless nodes can operate either in access point (AP) or station (STA) mode. Each wireless device is managed and monitored by device monitoring manager logic implemented in user space utilizing *netlink* communication with the driver. The monitoring information that is collected in each operational link is: channel state information (CSI), packet error rate (PER), and signal-to-noise and interference ratio (SINR). Further, this information is shared with the traffic steering control in the TSN switch for performing traffic steering based on real-time monitoring information.

Distributed nodes are tightly time-synchronized via Ethernet. When the wireless nodes operate in AP mode, they are time-synchronized to the TSN switch. When the wireless nodes operate in STA mode, they are time-synchronized over wireless to their respective APs. In such a case, the TSN switch time is subordinated to the STA(s) time. The time synchronization master clock always resides in the wired network connected to the d-MLD operating in AP mode. The proposed d-MLD architecture can support simultaneous transmit and receive (STR) on both links without any self-interference issues. Self-interference happens due to simultaneous reception and transmission in closely spaced frequencies or cross-talk between antennas in the same radio frequency front-end. Figure 2-11 shows the interconnection between wireless devices and their building blocks to perform MLO, while Figure 2-12 shows the system perspective of the testbed setup. The d-MLD is interconnected with the central network controller (CNC) to retrieve information on e.g., WTSN scheduling and bootstrapping time synchronization. End devices that are connected to the central user controller (CUC) support application requirement sharing with the controller to enable automatic scheduling and management.

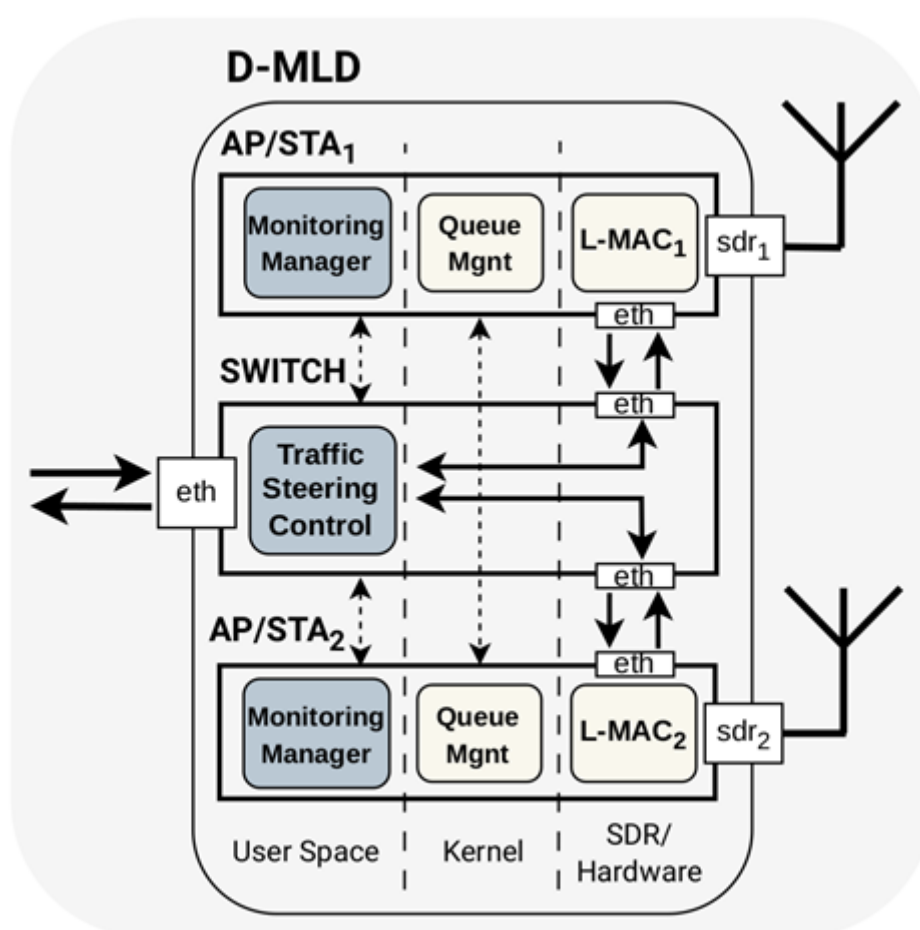


Figure 2-11: Distributed MLD architecture:

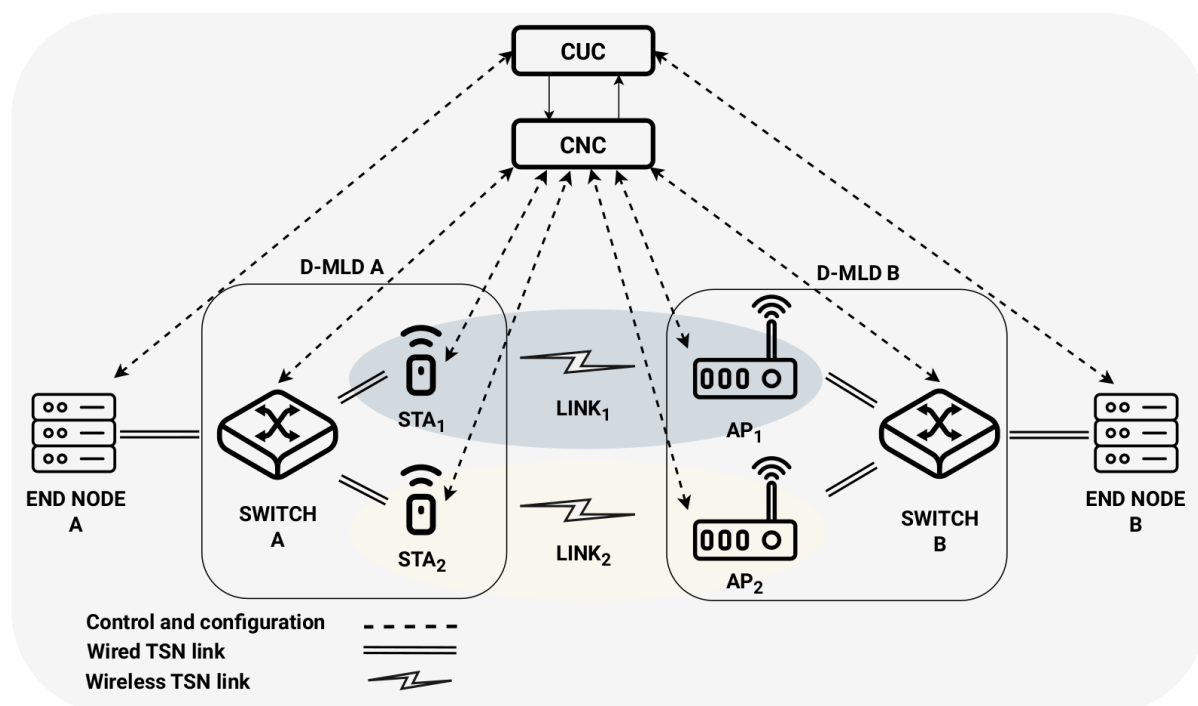


Figure 2-12: Multi-link operation system perspective

Link and Flow Management

Traffic steering is performed by traffic steering control in the TSN switch by allowing traffic duplication on both links or the selection of certain link based on wireless channel measurements. As the decision happens at the TSN switch, the TSN switch should be aware of the monitored information on each link: channel state information (CSI), packet error rate (PER), and signal-to-noise ratio (SNR) at fixed time intervals. Link management is handled independently for each link. This means that each wireless device, when in STA mode, independently associates with the corresponding AP device. Similarly, each wireless device when in AP mode broadcasts the same Service Set Identifier (SSID). Device monitoring manager tracks the established connections of the wireless devices and informs traffic steering control logic. As such, traffic steering control receives the list of all associated stations to the wireless device operating in AP mode, or it receives information if link is active for the wireless device operating in STA mode. If a link is not fully established with the AP or does not have a specific STA connected, the traffic steering control disables that link.

Next to link management, traffic flow management must be performed in real-time to achieve FRER similar to wired networks. Currently, flow management for MLO is performed on per-traffic identifiers (TID), by mapping certain TIDs at certain links, or dynamically shifting different TIDs between links based on specific parameters (e.g., link load). As the wireless link can change fast over time, to enable FRER utilizing MLO, traffic steering is performed on a per-packet level. Different packets can be steered on different links or can be duplicated in multiple links based on the latest link information. By enabling traffic flow management on a per-packet basis, wireless channel dynamics can be considered.

Frame Replication and Elimination for Reliability (FRER)

To be able to detect duplicates and eliminate them at the endpoint, IEEE 802.11CB [27] specifies a redundancy tag (R-Tag) that utilizes a 16-bit sequence number that is placed in the Ethernet header similar to a VLAN tag. In wireless links, such sequence numbers should be mapped to a sequence number part of the IEEE 802.11 layer two header. This requires a dedicated mapping function when the packet switches from the wired network to the wireless and back, increasing the processing time for each packet. In addition, it will require a new information element on the IEEE 802.11 layer two header.

To streamline the process, we utilize an elimination mechanism based on hash values. When the packet is received at the endpoint, the elimination function produces the hash value of the packet and saves it in a register, adding a timer. Once the duplicates are received, their hash values are calculated and compared with the values saved in the register. If the hash value is the same and still active in the register, the packet is eliminated. To avoid any negative impact of the in-band network telemetry (INT) information on the hash value, the hash value is calculated over layer 2 and the IP payload of the packet only, excluding the IP header. The INT monitoring collects monitoring data on each network hop and adds them as an extension header in the layer three header. As such, the INT monitoring [28] makes the IP header of packet replicas different due to the monitored information on different paths.

2.3.2.2 Measurement Test-Bed and Setup Implementation

The measurement setup is shown in Figure 2-12 and is composed of two openwifi APs running on ZCU102 boards and two openwifi stations running on ZedBoard. All the wireless nodes use FMCOMMS3 radio frontends that can operate between 70 MHz and 6 GHz. Both APs and STAs are connected to their respective Linux-enabled TSN switches utilizing a 10 Gbps link. Two end devices running NUCs are connected to the TSN switches. All the network devices are controlled by a centralized network controller (CNC), which collects the telemetry information and plots the monitored information in real time on a dashboard. The setup operates two links, one at 5 GHz and the other at the 2.4 GHz band.

Traffic Steering

We follow an agent-based architecture [29], similar to the one used for communication between CNC and network nodes. However, for local information sharing we support direct communication between agents and modules inside the d-MLD. The device monitoring manager monitors information regarding the link statistics: CSI and PER on fixed intervals and shares this periodically with traffic steering control agent in the TSN switch. In addition, information regarding the link activity are shared, e.g. if the STA node is associated with AP or not. Such information is collected from *wpa_supplicant_cli* (the command-line interface for the station side Linux daemon) and *hostapd_cli* (the command-line interface for the AP side Linux daemon), respectively. Based on monitored information, traffic steering control decide to enable/disable certain paths.

Bridges between certain interfaces in the TSN switch are implemented using the click router framework [30]. Packets are *colored* with a tag value based on which link(s) they are allowed to follow. Such *color* depends on the Differentiated Services Code Point (DSCP) value of the packet, and traffic steering control decides which packets can be steered/duplicated on each link based on DSCP value selection. As such, each packet can be dynamically steered on

each link based on its color. Similarly, CNC can have full control over such traffic steering, by installing certain mapping in traffic steering control remotely.

Scenarios

To demonstrate the benefits of MLO in enabling FRER in WTSN, we conducted initial measurements to assess the achieved latency and reliability of time-sensitive applications under different levels of traffic load on dedicated links. In these measurements, packet duplication was performed at the TSN switches, while duplicate elimination was handled by the other TSN switch. We evaluated end-to-end traffic flow performance in terms of reliability, latency, and packet loss rate for each communication link separately.

Table 2-2 summarizes the test scenarios. Traffic load on each link was introduced by injecting UDP traffic from another station. First, we measure the maximum achievable UDP throughput for a single station connected to the AP on the respective link. Based on this, traffic load levels are set as fraction of the maximum achievable throughput. All traffic load level [0 %, 25 %, 50 %, 75 %] combinations at both links are tested. The time-sensitive traffic flow (the main traffic flow in the system) is represented as periodic traffic with a period of 50 ms and is transmitted from the end device connected to the d-MLD operating as AP to the end device connected to the d-MLD operating as a STA.

Table 2-2: Scenarios and introduced traffic load on each link

Scenario ID	Links	Traffic load
1	2.4 GHz – 5 GHz	0% - 0%
2	2.4 GHz – 5 GHz	25% - 0%
3	2.4 GHz – 5 GHz	50% - 0%
4	2.4 GHz – 5 GHz	75% - 0%
5	2.4 GHz – 5 GHz	0% - 25%
6	2.4 GHz – 5 GHz	25% - 25%
7	2.4 GHz – 5 GHz	50% - 25%
8	2.4 GHz – 5 GHz	75% - 25%
9	2.4 GHz – 5 GHz	0% - 50%
10	2.4 GHz – 5 GHz	25% - 50%
11	2.4 GHz – 5 GHz	50% - 50%
12	2.4 GHz – 5 GHz	75% - 50%
13	2.4 GHz – 5 GHz	0% - 75%
14	2.4 GHz – 5 GHz	25% - 75%
15	2.4 GHz – 5 GHz	50% - 75%
16	2.4 GHz – 5 GHz	75% - 75%

2.3.2.3 Results

In this section, we will present the achieved results. We show communication latency achieved for time-sensitive traffic flow under different traffic loads on each communication link. In

addition, we also show the reliability of time-sensitive traffic flow as well as the achieved reliability of each link separately.

Achieved communication latency for the time-sensitive traffic flow is shown in Figure 2-13. Due to FRER enabled by MLO, the end-to-end median latency is below ~ 5 ms for all scenarios, except scenario 16 where the traffic load on both links is 75 %. What can be noticed is that the latency is impacted by the traffic load on the 5 GHz link. When the traffic load on that link is increased on average, the latency is increased as well (cfr. scenarios 13 to 16). The presence of outliers come from the fact that there is also traffic load not coordinated from the measurement setup. So even in case of scenario 0, there exists certain channel load due to presence of other APs in the environment.

Communication reliability is shown in Figure 2-14 for all measured scenarios. In all scenarios, the reliability is above 99\% except for scenario 16 (scenario with the highest traffic load on both links), where the reliability dropped down to 98.25\%. To compare the reliability of the time-sensitive flow with the actual reliability provided on each link, we also collected the packet loss rate every second on each link utilizing device monitoring manager. For the 5 GHz link (Figure 2-15), we see that the average reliability decreases with an increase in traffic load and drops down to below 85 % for scenarios 13 to 16. Similarly, for the 2.4 GHz link (Figure 2-16), the reliability drops below 95 % (cfr. scenario 8 and 12) or even below 90\% (cfr. scenario 4 and 16).

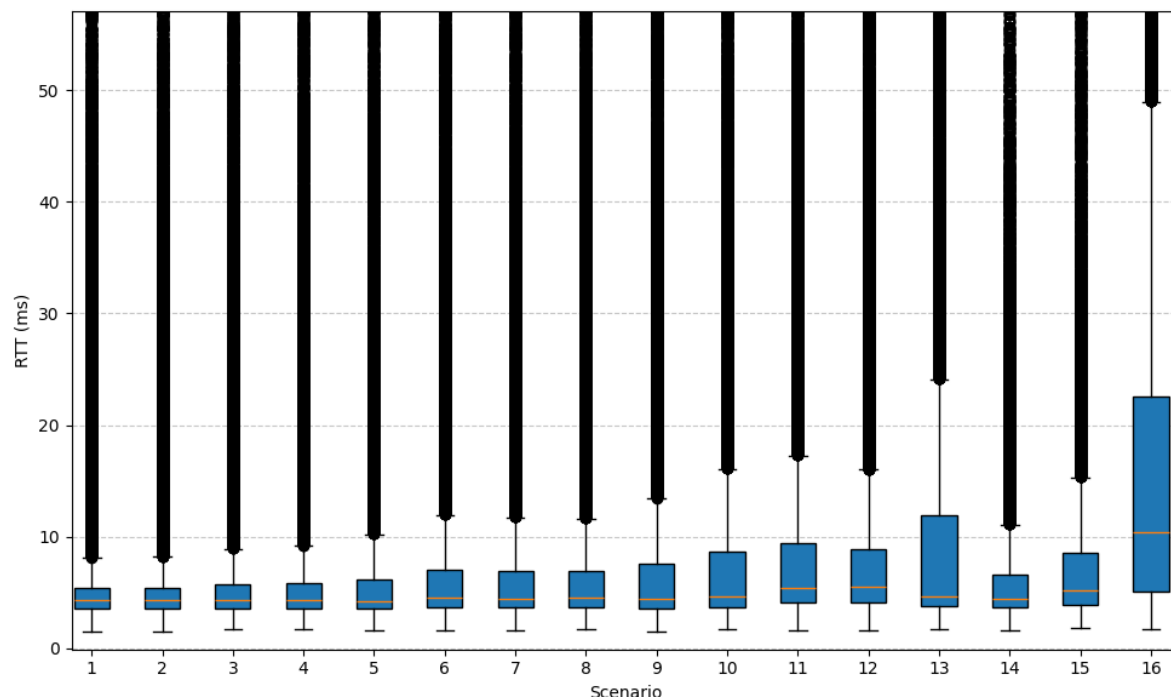


Figure 2-13: Communication latency for different scenarios

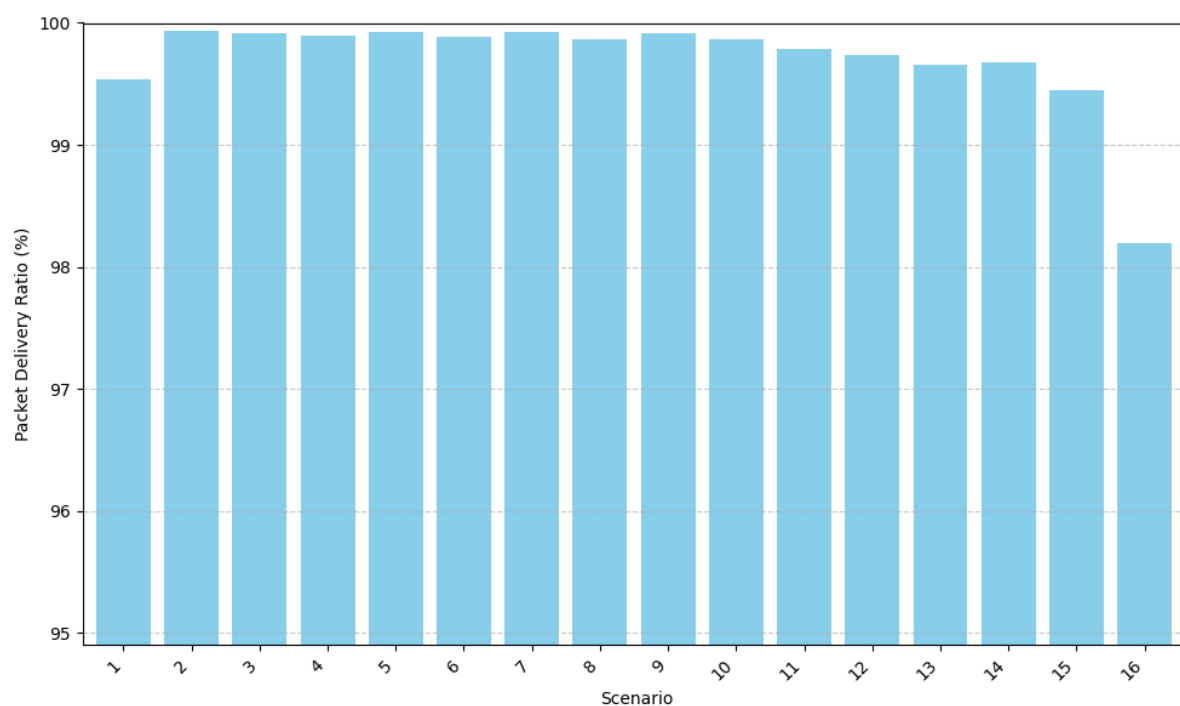


Figure 2-14: Communication reliability for different scenarios

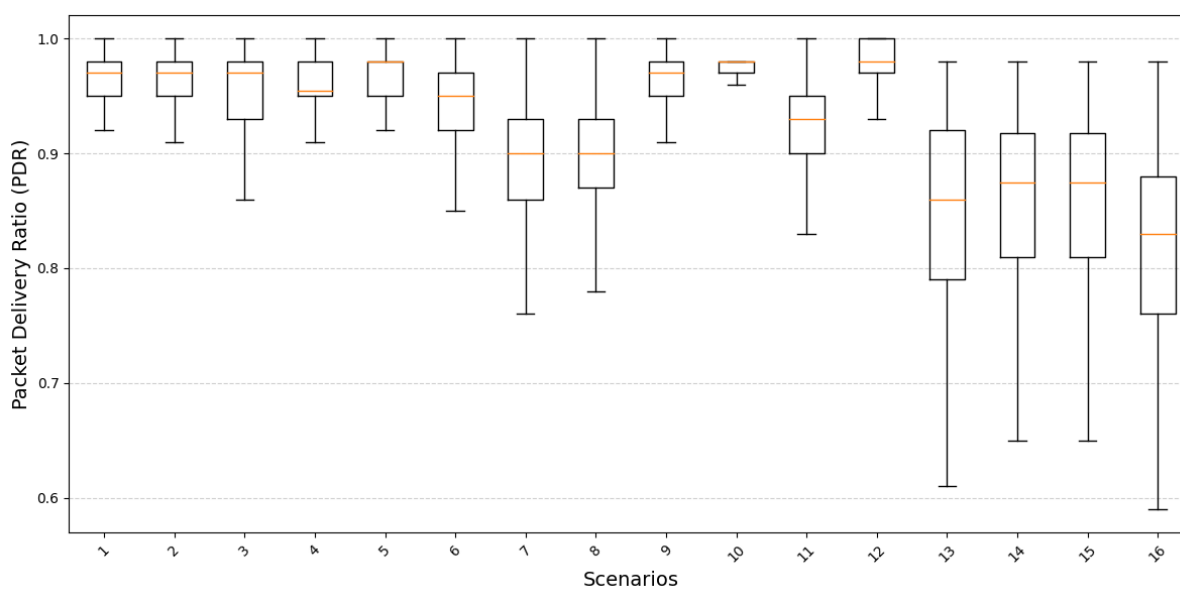


Figure 2-15: Link reliability at 5 GHz

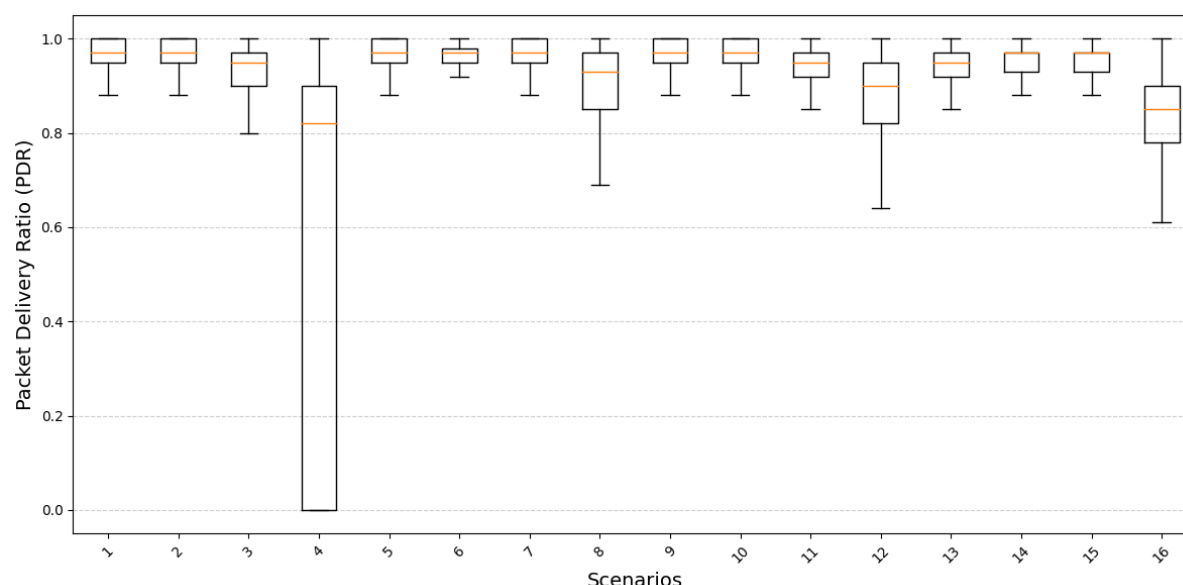


Figure 2-16: Link reliability at 2 GHz

2.4 TRANSPORT / XHAUL

The transport domain is responsible for managing all wired and wireless infrastructure that enables inter-domain connectivity, including transport segments connecting RAN, CN, NTN and NPN [66](for more details, please refer to D2.3 [66]). The management and orchestration of this domain is handled by the Network Slice Controller (NSC), which handles slice lifecycle and connectivity orchestration, assuming the Domain Management Orchestrator (DMO) role. Then, to manage the underlying transport infrastructure, a dedicated Software-Defined Networking (SDN) controller is required. In this implementation, the TeraflowSDN (TFS) [31] controller is used. TFS provides control and abstraction capabilities for heterogeneous transport technologies, supporting both wired IP networking devices and wireless transport equipment. Accordingly, this section is structured around the three main layers that compose the Transport Domain: the NSC, handling the orchestration, the TFS controller for handling the underlay infrastructure and finally the transport infrastructure devices.

2.4.1 Transport architecture and components

2.4.1.1 Network Slice Controller

2.4.1.1.1 Overview

The NSC, serving as the DMO, is responsible for the end-to-end lifecycle management of transport network slices. It processes slice requests, enriches them with contextual information, determines the appropriate transport technology (e.g., L2VPN, L3VPN), and oversees their realization and operational control. In this way, it enables seamless connectivity and coordination across multiple network domains, including Core, RAN, NPN, and NTN, ensuring end-to-end slice continuity.

2.4.1.1.2 Architecture

The NSC architecture (Figure 2-17) follows a modular workflow composed of five functional blocks, each implemented by a corresponding internal component and interacting through standard northbound, eastbound, and southbound interfaces.

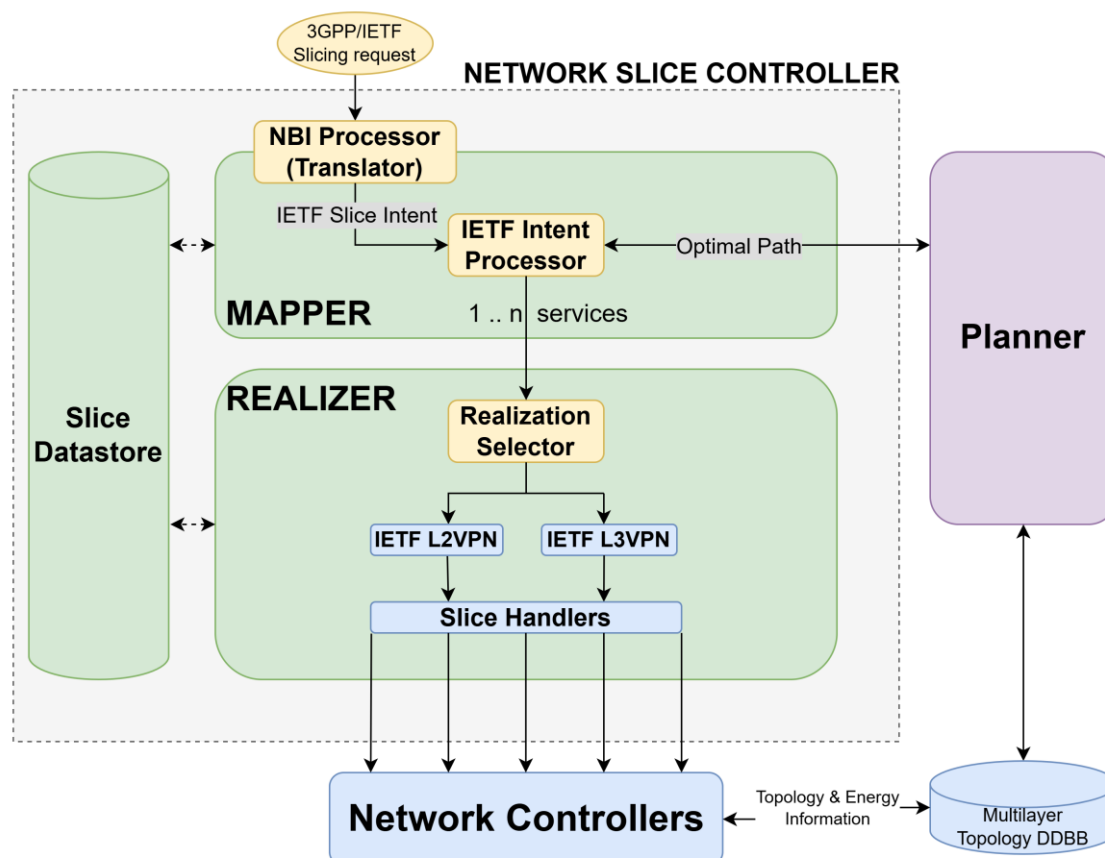


Figure 2-17: NSC Architecture

NBI Processor

This component manages the requests entering the system. There are 4 kinds of requests:

- **Create:** when this request arrives, the Northbound Interface (NBI) processor checks the body of the request to analyze the format. It translates it into an IETF Slice Service Request. This function is able to process each request independently, even if several requests with different formats come together in the same request. Once created, the slice is stored in the slice database.
- **Delete:** this request can be referring to a specific slice or to all slices configured. In any case, a deletion request is sent to the southbound controller, and the referred slices are deleted from the slice database.
- **Get:** this request can be referring to a specific slice or to all slices configured. In any case, the referred slices in the request are returned from the slice database.
- **Modify:** this request allows changing the configuration of an existing slice. It works similarly to the create request. Once processed, a modify request is sent to the controller and the referred slice is updated in the slice database.

Mapper

The mapper receives slices service requests from customers and process them to obtain an overall view of how this new request complements the rest of slices. It extracts the key slice parameters (endpoints, slice constraints, connectivity type and optionally the type of service to deploy) and enriches the request by retrieving missing information from external planning or topology components through the eastbound interface. Based on this enriched context, the Mapper generates as many service instances as required, each with its corresponding set of key slice parameters.

Planner

Although the planner is not a core component of the NSC, it is included here due to its role in supporting certain developments that contribute to the overall functionality of the NSC. The planner, whose role is done by an NDT, operates as a Path Computation Engine (PCE) to determine the optimal path that meets specific requirements. The planner is triggered by the mapper, which relies on it to complete missing information necessary for the realizer to implement the network slice, such as the traffic path.

Realizer

The Realizer converts the abstract slice context into technology-specific service models, which are the IETF Layer 2 Service Model (L2SM) [32] and the IETF Layer 3 Service Model (L3SM) [33] and translates them into controller-specific instructions for the selected southbound entity, the TFS controller, through the slice handlers. The Realizer is responsible for instantiating and modifying the required network resources.

2.4.1.1.3 Interfaces

The NSC contains the following interfaces:

Northbound

The northbound interface of the NSC receives high-level slice requests and adapts them into standardized formats, specifically to the IETF Network Slice Service YANG model [34].

Eastbound

This interface connects to the external components such as Digital Twins. The digital twin interface allows DT components to ingest topology, telemetry, and slice requirement information, enabling predictive analytics and assisting the NSC in validating slice feasibility.

Southbound

The southbound interface is controller specific. In this case, TeraflowSDN controller is selected as the controller to manage the infrastructure, which is compliant with standard interfaces. Specifically, TFS controller NBI supports the L2SM and the L3SM to deploy Layer 2 and Layer 3 VPNs respectively.

Moreover, the NSC contains a telemetry interface for real-time monitoring data, enabling the controller to maintain accurate topology knowledge and support both assurance and closed-loop operations. This interface is further described in Section 3.4.1.

2.4.1.1.4 Workflow

This section describes an example workflow of how a service for the transport network is processed by the NSC, as depicted in Figure 2-18.

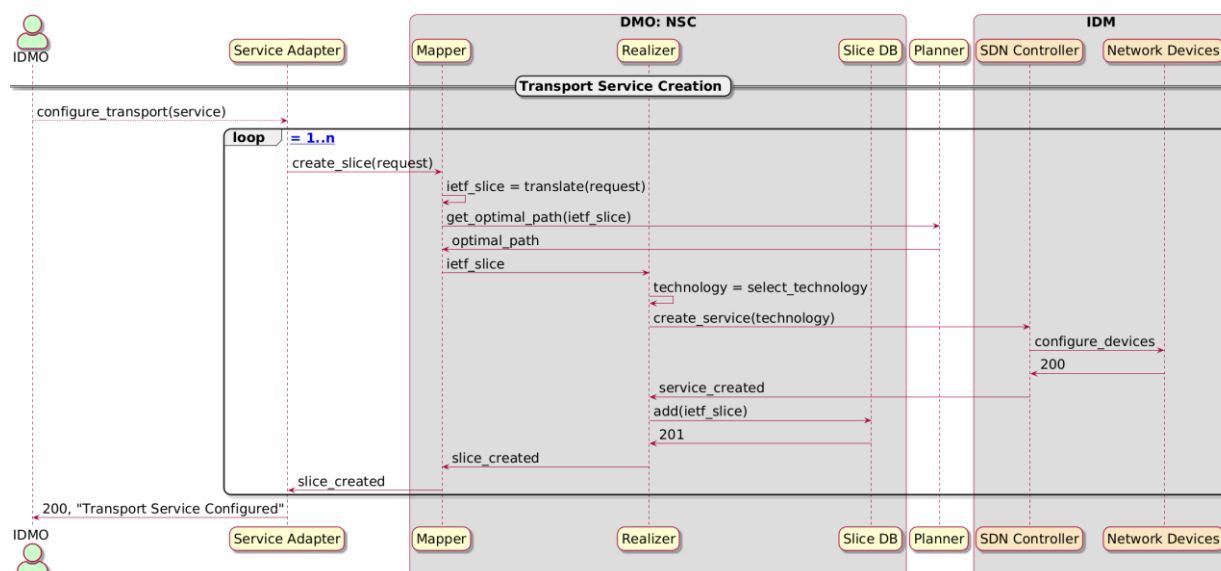


Figure 2-18: Transport Service Creation Workflow

1. The IDMO sends a service request to configure the transport domain
2. The service adapter generates the required slice request for the NSC to process
3. Each request is translated into an IETF Slice
4. The mapper processes the request and interacts with the planner
5. The planner (Digital Twin) processes the request, populates it with the optimal path and returns it to the mapper
6. The mapper sends the request to the realizer, which selects the optimal realization technology for deploying the service
7. The realization module translate the request to the controller specific configuration (L2SM or L3SM)

2.4.1.2 TeraflowSDN Controller

TeraFlowSDN controller is an open-source, cloud-native, carrier-grade Software-Defined Network (SDN) controller developed by the ETSI Software Development Group TFS. It is designed to control and automate transport networks for 5G, beyond-5G, and future 6G, addressing scalability, flexibility, and security limitations of traditional monolithic SDN controllers.

Built on a microservice-based architecture, TFS controller uses standardized northbound and southbound interfaces to enable vendor-agnostic, multi-technology control across IP, optical, and microwave networks. It supports zero-touch automation, network slicing, advanced telemetry, and security features, making it a reference platform for operators, research, and standardization activities.

In the context of the UNITY-6G project, TeraFlowSDN has been selected due to its open-source nature and its modular microservice architecture, which enables the deployment of only the components required by the targeted experiments, thereby improving flexibility and resource efficiency. Furthermore, its adoption is supported by prior experience gained in earlier research projects, which facilitates integration, reduces technical risk, and accelerates experimentation and development activities.

Its architecture, depicted in Figure 2-19, is modular and comprehensive, with each functional block representing a service that can be selectively deployed within the controller. At the northbound interface, the TFS controller employs the L3VPN module to receive service requests from the NSC. On the southbound side, OpenConfig-based¹ modules are used to translate these service requests into concrete device-level configurations. In addition, the telemetry module enables the collection of performance metrics from the network topology (e.g., link latency), which are consumed by the NSC to support connectivity assurance functions. Finally, the inventory module maintains an up-to-date view of the network topology managed by the controller, which is also exposed to and consumed by the NSC.

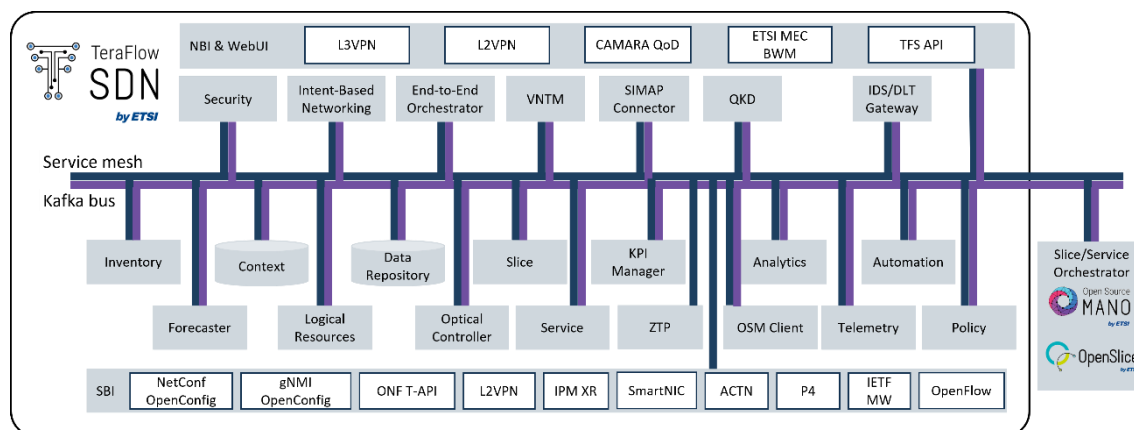


Figure 2-19: TeraflowSDN controller architecture

2.4.1.2.1 Integration with wireless devices

The wireless transport devices are integrated into the 6G architecture transport domain through a layered control and exposure chain which handles service and slice orchestration, TeraFlowSDN provides the programmable transport-control plane, and wireless device integration enables the control and configuration of existing wireless transport solutions within the UNITY-6G framework. Devices interface via REST API, which is translated via adapter to connect to TFS capabilities.

At present, TeraFlowSDN already provides an architectural basis for transport-domain integration, including support for pluggable southbound drivers, device inventory, service orchestration, policy handling, path computation, and transport slicing. In the latest TeraFlowSDN Release, a dedicated MicroWave SBI domain is available. However, that microwave support is presently described as an intermediate-controller integration based on the IETF RFC 8345 topology model, with support focused on L2 radio links and with monitoring explicitly marked as not supported in the published driver description; the publicly referenced

¹ <https://www.openconfig.net/>

tested case is a SIAE intermediate microwave controller. In practical terms, this means that TeraFlowSDN already offers the correct control-plane anchoring point for wireless transport, but it does not yet provide a Ceragon-specific, feature-complete realization for exposing Ceragon transport capabilities, topology, state, and controllable parameters into the TeraFlowSDN environment required by the UNITY-6G transport domain.

The proposed solution is therefore to introduce a Ceragon-specific adapter, acting as a mediation and normalization layer between Ceragon transport devices and controllers on one side, and TeraFlowSDN on the other. This adapter uses Ceragon REST APIs to retrieve and maintain an up-to-date representation of the transport network, including nodes, links, link endpoints, radio capabilities, capacity profiles, protection attributes, operational state, alarms, and selected performance indicators, and would translate these into the abstractions consumed by the TeraFlowSDN Device, Context, Telemetry, Analytics, Policy, and Service-related components. In the opposite direction, the same adapter would translate TeraFlowSDN decisions into Ceragon-understandable actions, allowing transport intents, policy-driven optimization, slice-related constraints, and path-computation outputs to be enforced on the real wireless transport infrastructure. This approach is fully aligned with the TeraFlowSDN architecture, which is explicitly built around multiple SBI plugins for device management, a REST-based northbound exposure toward external systems, and more recent extensions for inventory export, telemetry, analytics, and lifecycle operations on topology, devices, services, and slices.

Within the project, this integration would turn the transport domain from a partially opaque subsystem into an exposed, programmable, and analyzable resource domain inside the broader UNITY-6G architecture. Once Ceragon transport information is continuously synchronized into TeraFlowSDN, the project can use real transport-domain topology, capabilities, inventory, and telemetry as inputs not only for orchestration and optimization, but also for digital-twin synchronization. This is especially important for multi-layer topology handling, inventory management, telemetry, analytics, automation, and path computation; by feeding these transport-domain views into the digital twin, the network control mechanisms can execute realistic what-if analyses on failures, degradations, rerouting strategies, SLA violations, capacity bottlenecks, resilience policies, and alternative configuration plans before applying them to the operational network. The result is a closed-loop workflow in which TeraFlowSDN provides the authoritative transport-domain view, the digital twin evaluates candidate actions under simulated conditions, and validated decisions are returned through the Ceragon adapter for execution in the real transport network.

2.4.1.3 Infrastructure devices

The transport infrastructure devices are the physical elements that realize the transport-domain connectivity between distributed 6G functions and sites. They provide carrier-grade point-to-point microwave and millimetre-wave links, as well as transport-oriented IP devices that provide packet forwarding, aggregation, and interconnection across aggregation, midhaul, and fronthaul segments. and operate as programmable, observable resources under the transport control and orchestration framework.

2.4.1.3.1 IP devices

FRR² devices have been selected due to their flexibility, lightweight footprint, and open-source nature, which make them particularly well suited for research and experimental environments. As a software-based routing solution, FRR enables rapid deployment, easy customization, and reproducible experimentation without the constraints associated with proprietary hardware platforms.

These devices are software-based routing nodes built on Free Range Routing (FRR)² a free and open-source, high-performance IP routing protocol suite for Linux and Unix-based systems. FRR implements a comprehensive set of widely used routing and signalling protocols and Segment Routing extensions, enabling realistic emulation of carrier-grade IP/MPLS transport networks.

FRR follows a modular, daemon-based architecture, where each routing protocol runs as an independent process coordinated by a central routing manager, improving resiliency, extensibility, and fault isolation compared to monolithic routing stacks. In the testbed, FRR devices operate as programmable and SDN-controllable routers, integrating seamlessly with the Linux networking stack and supporting dynamic configuration, routing experimentation, and telemetry collection. Their software-based nature enables rapid deployment, flexible topology definition, and reproducible experiments, making FRR devices well suited for validating control, automation, and assurance mechanisms in advanced transport network scenarios.

2.4.2 Intent-based transport configuration

2.4.2.1 Overview

The increasing heterogeneity and operational dynamicity of 6G transport infrastructures make manual configuration of transport resources progressively inadequate, particularly in scenarios where wired and wireless segments must jointly support stringent service requirements across RAN, CN, and NTN domains. In this context, an intent-based transport configuration capability whose objective is to translate high-level service or client requirements into concrete and enforceable configurations for transport infrastructure elements, with particular focus on wireless transport solutions is contributed. This contribution is aligned with the overall transport-domain framework, and the orchestration role, TeraFlowSDN provides control and abstraction of the underlay, and the transport infrastructure devices execute the resulting configuration actions. Within this layered architecture, the functionality complements the existing orchestration workflow by introducing an intermediate reasoning layer that transforms requested transport objectives into device-realizable configurations while preserving compatibility with the broader UNITY-6G framework.

The main principle of this approach is to allow the requesting entity to specify *what* the transport network is expected to achieve, rather than explicitly defining *how* each transport node shall be configured. Accordingly, the intent can express objectives such as capacity guarantees, resilience targets, low-latency operation, synchronization constraints, energy-efficiency requirements, or slice-aware traffic treatment, without exposing low-level parameters such as modulation profiles, channel widths, protection modes, queue mappings or transmit power

² <https://frrouting.org/>

settings. The purpose of the intent function is therefore to bridge the gap between high-level service objectives and the specific operational capabilities of the underlying transport equipment, enabling more automated, scalable and policy-driven operation of microwave, millimetre-wave and xHaul transport resources. This is especially relevant for the project use case, where there are application-driven scenarios, including those where transport must support demanding SLA profiles such as the ones associated with immersive or holographic services.

2.4.2.2 Functional architecture

From a functional viewpoint, the intent-based transport configuration mechanism is implemented as a modular reasoning pipeline placed between the intent source and the southbound realization functions. This function can be considered as relying on a hybrid architecture that combines strict rule-based validation with AI-assisted interpretation. This design choice is particularly important in transport environments, where requests may be expressed using abstract business or service language, whereas their realization depends on concrete hardware limitations and topology constraints. The ingress stage receives the intent, normalizes it, and parses it so as to identify the targeted transport scope, the intended operational objective, and the relevant constraint category. At the same time, the parser applies validation logic to ensure that the intent is meaningful in the context of the available topology and admissible transport policies, thereby preventing invalid or non-realizable requests from propagating into the control chain.

A key architectural element of the solution is the use of a hardware-aware knowledge base that stores operational envelopes and capability information for the target transport equipment. Technical specifications refer to hardware profiles for node types and associates them with different transport roles. This enables the intent function to reason not only on the requested objective, but also on whether that objective is feasible for the specific node, link, and topology segment involved. In this way, the intent engine acts as a constraint-solving layer that maps the declared objective against actual device and topology capabilities before selecting a suitable realization strategy. Such an approach is essential in wireless transport environments, where the available feature set and operational range may differ significantly from one hardware class to another.

The functional scope of the contribution spans multiple transport intent domains. These include capacity, resilience, QoS or slicing, energy management, and security or synchronization. In operational terms, this means that the system can interpret intents related to preserving or increasing bandwidth resources, enforcing protection and redundancy mechanisms, differentiating transport treatment for latency-sensitive versus best-effort traffic, activating energy-aware configurations during low-load conditions, or ensuring secure and time-synchronized transport operation. Such multi-domain coverage is required in the UNITY-6G context, where the transport domain is not a passive forwarding substrate but an active participant in the fulfilment of end-to-end service requirements.

2.4.2.3 Intent processing and translation workflow

The intent-processing workflow follows a staged logic that can be summarized as parsing, validation, planning, translation and dispatch. First, the system ingests the request through a

northbound entry point. The implementation supports both natural-language directives and structured JSON declarations, thereby enabling interaction by human operators as well as by external automation components. After ingestion, the parser resolves the target scope and the intent category, using hybrid rule-based and AI-assisted reasoning where appropriate. Once the request has been interpreted, it is bound against the hardware knowledge base and against the known physical topology so as to verify whether the requested objective can be directly supported, requires an alternative realization, or must be rejected. This staged decomposition is consistent with the more general transport-domain workflow, where an abstract service request is progressively enriched and translated into a realizable technology-specific configuration. A particularly relevant feature of the contribution is its support for hardware-aware fallback and graceful reconciliation. Traditional transport configuration systems frequently fail when the requested feature is not directly supported by the target hardware. By contrast, the contribution describes a strategy in which the intent engine reasons on the goal of the request and, where possible, derives an alternative configuration strategy that still advances the requested objective. The provided example is a reliability-oriented intent requesting Space Diversity on equipment that does not support it; in such a case, the system may reconcile the request to an alternative protection mode such as 1+1 Hot Standby. This behaviour is important in the UNITY-6G transport context because it increases robustness and operational continuity while avoiding unnecessary service rejection due solely to hardware heterogeneity. Once the intent has been successfully interpreted and reconciled with the available capabilities, the system generates a realization plan and a corresponding set of configuration actions. These actions are represented as structured device configuration objects and can be translated into NETCONF/YANG payloads or controller-consumable instructions. In practice, this means that the function provides the logic that bridges the semantic gap between high-level transport objectives and concrete configuration artefacts, while the downstream controller layer remains responsible for enforcing the resulting configuration on the infrastructure. This positioning is coherent with the overall UNITY-6G transport architecture, in which the southbound realization of services is already delegated to TeraFlowSDN and associated transport controllers. Consequently, the intent function should be understood as a complementary transport-intelligence layer that enriches the existing control chain with hardware-aware reasoning rather than replacing the transport orchestration logic already described. In the proposed intent-based transport solution, the decision logic is structured around skills and tools. A skill denotes a reusable technical capability of the intent engine, i.e., a well-defined function that allows the system to reason on the requested objective and determine an appropriate course of action. Typical skills in this context include intent parsing and classification, topology-aware scope resolution, hardware-capability matching, multi-constraint optimization, fallback selection, conflict detection, and closed-loop assurance against drift between intended and applied state. These skills allow the system to interpret what must be achieved in terms of capacity, resilience, QoS, synchronization, energy or security, and to map these objectives to feasible transport actions over the available infrastructure. The tools available to these skills are the concrete algorithmic and computational mechanisms used to support or refine the decision, such as rule-based validators, hardware knowledge-base queries, optimization routines, heuristic search procedures, telemetry-driven reconciliation logic, large-language-model assisted reasoning for complex or novel intents, and, where relevant, AI/ML or reinforcement-learning models and simulation or what-if evaluation functions that provide additional evidence before selecting the final realization strategy. In this sense, the skills express the operational intelligence of the

intent system, while the tools provide the analytical and computational means through which that intelligence is executed in a safe, explainable and hardware-aware manner.

2.5 SATELLITE / NTN

Satellites in 6G networks expand NTN architectures for ubiquitous coverage, integrating seamlessly with terrestrial systems via advanced payload, and AI driven orchestrations. As 6G networks evolve, satellite-based communication will play a vital role in enhancing connectivity, offering ultra-low latency, fast data transmission, and wide coverage. A key strategy for ensuring network resilience is the integration of satellite systems with terrestrial networks, providing seamless access in rural and remote areas and expanding connectivity to underserved regions.

To overcome the challenges associated with satellite-based communication, innovative technologies such as multi-beam antennas, AI-based resource management, and next-generation satellite constellations are being developed. These advancements will enable more efficient and reliable communication, ultimately improving the overall performance of 6G networks.

Satellites position as native 6G components, forming hybrid TN-NTN topologies with LEO (Low Earth Orbits), MEO (Medium Earth Orbits), GEO (Geostationary Earth Orbit), and HAPS (High Altitude Pseudo Satellites) for flexible coverage. There are several payload types in NTN deployment, from which mainly two types are common: 1) Transparent payload and 2) Regenerative payload [62].

Transparent payload usually also called bent-pipe structure because satellite simply receives signals, shift or amplifies it and retransmits it without interpreting or processing data content. While regenerative payloads perform onboard processing like demodulation, decoding, routing or switching and retransmission.

A regenerative payload with a gNB offers greater flexibility and improved performance, enabling global coverage. Additionally, NTN based on regenerative payloads are not only more flexible, but also more resilient, as they can route traffic and hand over calls directly between satellites with minimal involvement of the core network on the ground. This allows for more efficient and reliable communication, even in areas with limited or no terrestrial connectivity.

Two options are being considered for the regenerative payload:

1. A complete gNB on board the satellite, which offers a more straightforward and efficient solution.
2. A gNB with a Central Unit (CU) on the ground and a Distributed Unit (DU) on board the satellite, which presents challenges in handling the F1 interface between the CU and DU.

Figure 2-20 describes more architectural view of specific payload type, starting from 1) Transparent payload, 2) Regenerative payload with full gNB on board and 3) Regenerative payload with CU-DU split gNB, DU on satellite.

The O-RAN builds upon the 3GPP-defined NG-RAN as a baseline, adding its own specific functionalities. However, this reliance on the 3GPP-defined baseline creates a unique challenge for O-RAN in the context of NTN.

The 3GPP has initially defined a transparent payload architecture in Release 17 but is now considering a regenerative payload in Release 19. As a result, O-RAN-based NTN solutions would need to accommodate multiple types of payloads envisioned by the 3GPP, including different regenerative payload architectures [61].

Furthermore, the O-RAN-based NTN would need to be flexible enough to accommodate different regenerative payload architectures, such as one where the entire gNB is placed on the NTN platform, and another where only the gNB-DU is placed on the NTN platform in a future 3GPP Release. This requires a high degree of adaptability and flexibility in the O-RAN-based NTN solution.

As a result, O-RAN would need to accommodate NTN-specific interfaces to ensure seamless communication between the NTN and the O-RAN architecture. Additionally, depending on the type of NTN architecture (transparent or regenerative), certain information would need to be transmitted through the feeder link to support the use of the SMO and the RIC.

To ensure timely availability of this information, suitable provisioning of feeder link resources would be necessary. This would involve allocating sufficient bandwidth and resources to the feeder link to support the transmission of relevant information between the NTN and the O-RAN architecture [61].

Generally, 3GPP defines functional splits primarily through option 1-8 in its RAN architecture, focusing on dividing gNB into CU and DU via F1 interface, with further PHY-layer splits like 7-1, 7-2 and 7-3 via fronthaul. O-RAN builds directly on these 3GPP splits for example O-DU/O-RU using 7.2x, but extends with open interfaces, adding RIC for AI/ML optimization and SMO for orchestration.

Here mainly, regenerative payload with split gNB framework has been utilized and which is also standardized in O-RAN. Usually, 3GPP defines and also mentioned regenerative split architecture, but there several options for the functional split of RAN. In Section 2.5.2 mainly two split options have been described with OCUDU (previously srsRAN) implementation.

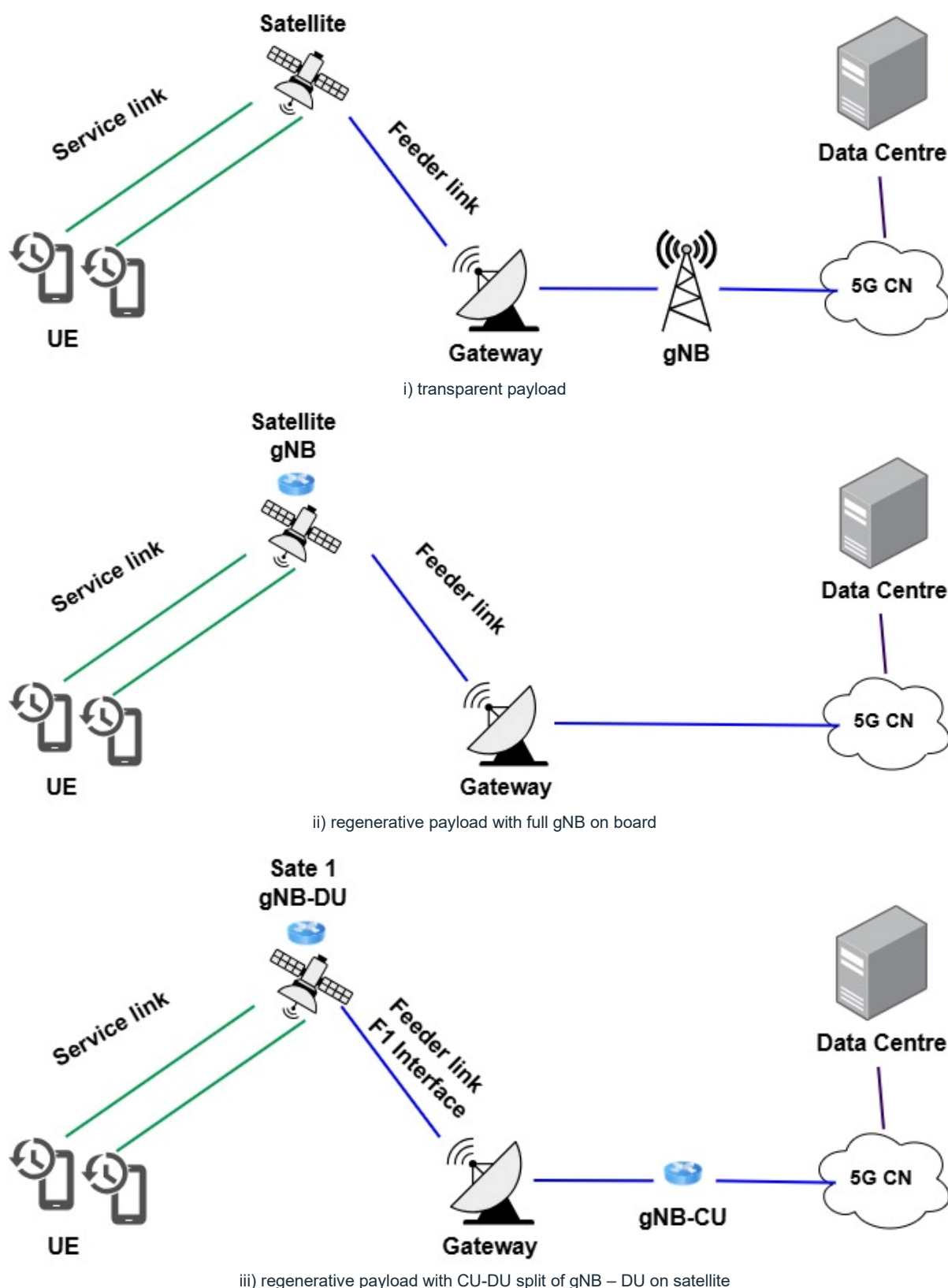


Figure 2-20: Overview of the different gNB splits and satellite payload architectures considered for UNITY-6G.

By moving network functions into space, the O-RAN architecture can be adapted to accommodate different implementation approaches for existing use cases, while also providing an opportunity for the O-RAN community to propose and implement new use cases that may

not have been possible with traditional terrestrial networks. This flexibility and innovation can lead to the development of new and innovative use cases that can take advantage of the unique capabilities of space-based networks.

The proposed use cases within this project for NTN networks should primarily focus on enhancing the sustainability of these networks. Key objectives include reducing energy consumption by both satellites and UE, as well as minimizing signalling overhead in the resource-constrained satellite environment. These goals are essential for ensuring the efficient and sustainable operation of NTN networks.

One promising use case of satellite-enabled 6G networks is sustainable disaster management and emergency communication. Natural disasters like earthquakes, floods, hurricanes and wildfires often damage or break terrestrial communication, where satellite systems provide resilient and rapidly deployable communication services under such conditions.

Real-time holographic communication represents another advanced 6G application that can benefit from satellite assisted networking. Holographic communication requires extremely high data rates, ultra-low latency, synchronization accuracy and reliable end to end connectivity.

2.5.1 NTN network architecture and components

As part of the UNITY-6G extensions, the NTN features of the OCUDU RAN solution will be extended to fulfil the project's (and PoCs) requirements. Figure 2-21 provides an overview of the most relevant and satellite-based communication-specific blocks (highlighted in purple) that are being implemented and/or extended on top of OCUDU's base. A brief description of those follows:

- Support for SIB19 broadcasting, with two options for Ephemeris information (position and velocity vectors or orbital parameters).
- NTN-enabled scheduler, including support for K-offset (accounting for extended satellite propagation latencies) and HARQ with up to 32 processes or disabled mode.
- External SIB19 generator with high accuracy, based on satellite Two-Line Element (TLE) data and the ground location of the UE (access link) and a ground station (feeder link), as well as a rapid updating mechanism in the gNB.
- Feeder-link handling by broadcasting timing advance (TA) information (required by the SIB19) and providing Doppler compensation in both DL (precompensation) and UL (postcompensation).

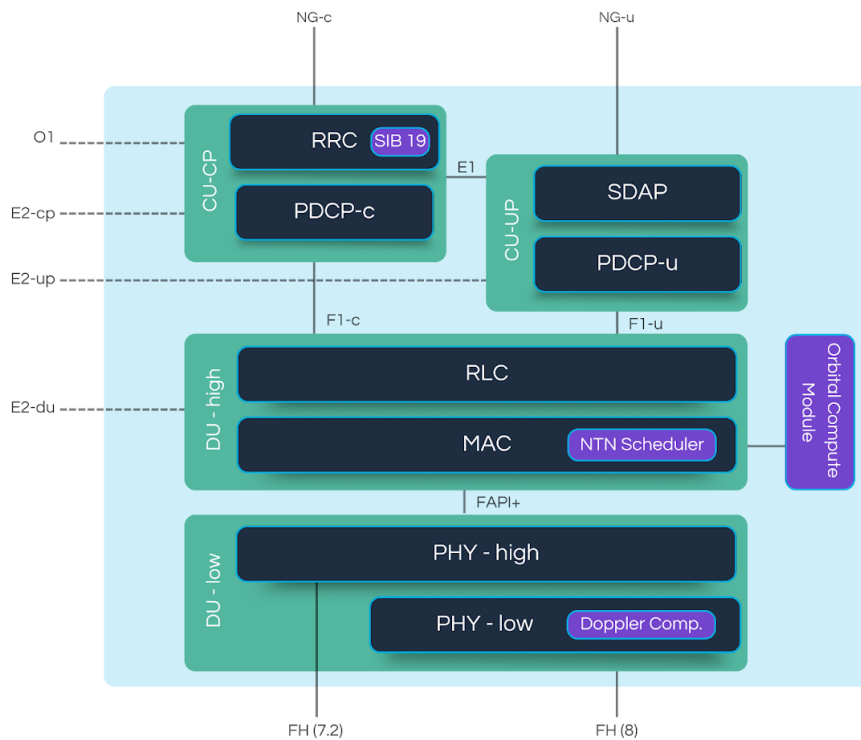


Figure 2-21: Overview of the NTN 5G RAN solution (satellite-specific blocks highlighted in purple).

As detailed in Section 2.1.1.2, in a first phase, the development targets to be aligned to the NR-NTN 3GPP release 17 specifications, supporting both LEO and GEO. Relevant 3GPP release 18 features will be added in a second phase. Moreover, regenerative architecture support will be enabled by leveraging the disaggregated and highly modular architecture of the O-CU/O-DU solution. The enhancements will also focus on the AI-based monitoring and control of the RAN solution in the context of deeply integrated TN/NTN use cases (leveraging both the E2 and O1 interfaces).

2.5.2 TN/NTN coexistence and split gNB Deployment

The coexistence of TTNs and NTN is an important step toward 6G system, where terrestrial infrastructure and satellite connectivity are expected to operate together to provide wider coverage, higher resilience, and service continuity. While TN deployments can also experience high time-varying channel variation, Doppler effects, and temporary link degradation, especially in high-mobility scenarios. However, NTN links introduce these challenges under more severe propagation conditions, including longer propagation distances, larger and time-varying delays, satellite-induced Doppler effects, and feeder/service link dynamics. Therefore, a flexible and softwarized testbed is needed to study how different RAN functional split options behave when part of the gNB architecture is located in space. Here, the objective is to investigate how split gNB architectures can support TN–NTN coexistence by distributing CU, DU, and RU functions between the ground and space segments. The proposed architecture is built mainly using OCUDU (previously srsRAN) as the open source RAN platform. OCUDU provides a 5G CU/DU implementation and supports different deployment modes, including CU/DU split operation using separate srsCU and srsDU applications, as well as O-RAN Split 7.2 operation with compatible Radio Units following the O-RAN disaggregated RAN architecture with O-CU, O-DU, and O-RU functions connected through open interfaces [63].

For NTN-related experimentation, OCUDU also provides a documentation demonstrating 5G NR with NTN extensions, where communication between the monolithic gNB and UE is relayed via non terrestrial components such as satellites. In the general testbed architecture, the Open5GS Core provides the 5G Core Network functions, including AMF, SMF, and UPF. The RAN side is implemented using srsRAN/OCUDU, where the gNB can be deployed either as a split gNB with separate CU and DU functions or as a CU-DU connected to an RU depending on the selected functional split. According to 3GPP TS 38.401, a gNB may consist of a gNB-CU-CP and gNB-CU-UP and one or more gNB-DUs, and the gNB-CU and gNB-DU are connected through the F1 interface. The NTN emulator is inserted at the interface where the satellite-related impairment is intended to be. FlexRIC is used as the near-real-time RIC, where xApps can monitor RAN metrics through the E2 interface and potentially support optimization decisions[66]. Depending on the experiment, the radio/access side can use USRP, ZMQ. Additionally, Amarisoft UE with NTN extensions is used as the UE side component. Here, we proposed to study two relevant split alternatives for TN–NTN coexistence: Split 2, where the NTN impairment is introduced on the CU–DU/F1 transport path, and Split 7.2x, where the NTN impairment is introduced on the DU–RU open fronthaul path. This comparison is important because the placement of the split directly determines which RAN functions are located in the ground segment and which functions are moved to the space NTN segment. As a result, the choice between Split 2 and Split 7.2x affects the required satellite payload processing capability, synchronization requirements and the overall feasibility of TN–NTN integration. The NTN emulator is therefore inserted at the interface where the satellite-related impairment is intended to be studied. In the Split 2 architecture, as shown in Figure 2-22 the functional split is located between the gNB-CU and gNB-DU. The ground segment contains Open5GS Core, srsCU, FlexRIC. The srsCU hosts the higher-layer centralized gNB functions, while the srsDU is placed on the space segment and hosts the lower RAN functions. The NTN transport emulator is inserted between the srsCU and srsDU, affecting the F1-C/F1-U interface. This represents a regenerative NTN scenario where the CU remains on the ground, while the DU is moved closer to the satellite or remote radio access side. In this case, the satellite/NTN link is modeled as an impaired F1 transport network. This split is attractive because F1 is a higher-layer interface than lower-PHY fronthaul and is therefore more suitable for studying CU/DU disaggregation under NTN-like transport impairments. However, it still requires stable transport and reliable handling of F1-C and F1-U traffic. The basis for this architecture is consistent with TS 38.401, where the gNB-CU and gNB-DU are separate logical entities connected through F1 Interface.

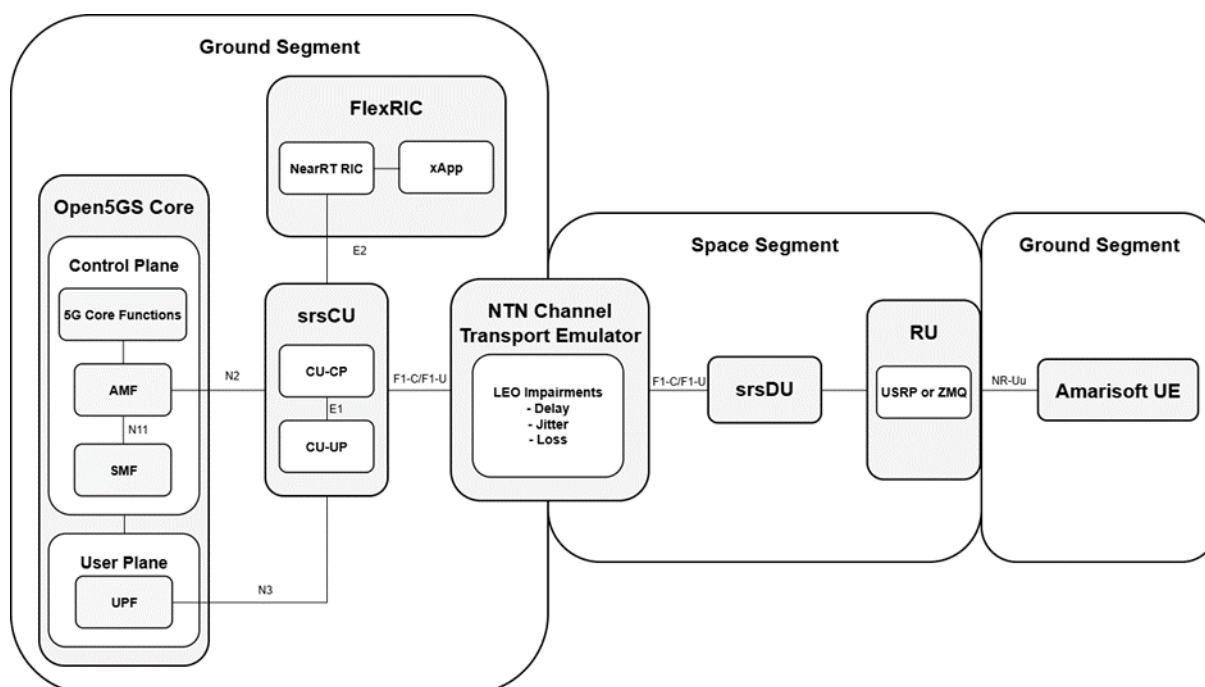


Figure 2-22: General Architecture of Split 2 Option

In the Split 7.2x architecture, as shown in Figure 2-23 the functional split is located between the gNB-DU and the O-RU. In this case, the Open5GS Core, srsCU, srsDU, FlexRIC remain in the ground segment. The space segment contains the O-RU, which hosts the lower-PHY and RF functions. The NTN fronthaul emulator is inserted between the srsDU and the O-RU, affecting the Open Fronthaul 7.2x interface [67]. This represents a scenario where the satellite payload behaves as an O-RU or lower-PHY/RF node, while the DU and CU remain on the ground. Compared with Split 2, this architecture keeps more processing on the ground, which can reduce satellite payload processing complexity. However, it is much more challenging from a transport perspective because Split 7.2x requires a fronthaul link that is much more sensitive to latency, jitter, synchronization, packet loss, and bandwidth limitations than the F1-based CU/DU split.

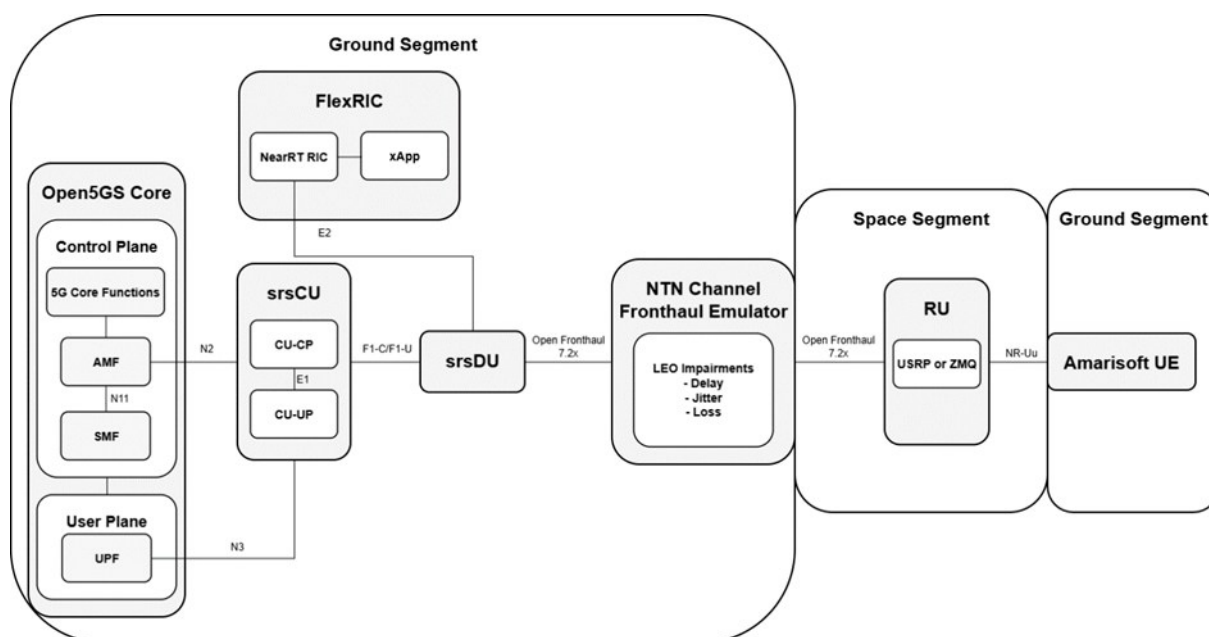


Figure 2-23: General Architecture of Split 7.2 Option

The two split options, therefore provide different trade-offs for TN–NTN coexistence. Split 2 is more feasible for NTN because the F1 interface is a higher-layer interface and allows the DU to be placed on the satellite or remote NTN side. This makes it suitable for studying regenerative payload architectures with onboard DU functionality. Its main challenge is that the satellite or remote NTN node must host more RAN processing, which increases payload complexity and may require more onboard computing resources. Split 7.2x, on the other hand, keeps the DU on the ground and moves only the RU/lower-PHY/RF function to the space segment. This can reduce onboard processing complexity, but it creates a very demanding fronthaul link between DU and RU. As a result, Split 7.2x is useful for studying ambitious O-RAN-based NTN fronthaul concepts, but it is less practical than Split 2 under realistic LEO delay and jitter conditions.

Lastly, in the Unity6G architecture, the building blocks of both functional split can help to simulate how TN-NTN components behave under different split assumptions, where Split 2 focuses on F1-based CU/DU separation and Split 7.2x focuses on O-RAN fronthaul between the DU and RU.

3 NETWORK EXPOSURE

The UNITY-6G architecture includes a Network Exposure Framework designed to provide a unified, secure and interoperable interface between the integrated 6G network infrastructure and a diverse set of application functions, including vertical applications, service orchestrators, network controllers and digital twin platforms.

Acting as a centralized exposure layer, the network exposure framework enables secure and standardized access to monitoring data and control mechanisms across multiple computing and network domains (RAN, transport, core). By exposing network information and fine-grained control mechanisms, it facilitates data-driven, AI-native optimization for service placement, resource management, and end-to-end QoS/QoE satisfaction. To achieve interoperability and alignment with industry practices, the framework adopts standardized technologies and APIs compliant with initiatives such as 3GPP NEF/CAPIF [76] [79], CAMARA [77][80], and O-RAN [78]. Therefore, the network exposure framework serves as a key enabler for the UNITY-6G proof-of-concepts (PoCs), including XR/holographic streaming, Industry 4.0 NPNs, DT-based evaluation, and disaster resilient connectivity.

The Network Exposure Framework implements exposure mechanisms defining two complementary layers, as depicted in Figure 3-1:

- Network exposure Layer-1 (NEL-1), provides telco-specific APIs facilitating direct access to domain functionalities such as xApps/rApps/dApps in the RAN, SDN controllers in the transport, and NEF/CAPIF in the core; and
- Network exposure Layer-2 (NEL-2), for app-developer-oriented APIs founded on CAMARA initiatives, simplifying complexity for external developers and vertical service providers.

Moreover, an API management module provides the governance and operational capabilities required to securely publish, manage, and monitor exposed APIs. This component supports functions such as API onboarding, authentication and authorization, access control, usage monitoring, and lifecycle management, ensuring secure and controlled exposure of network capabilities to internal and external consumers.

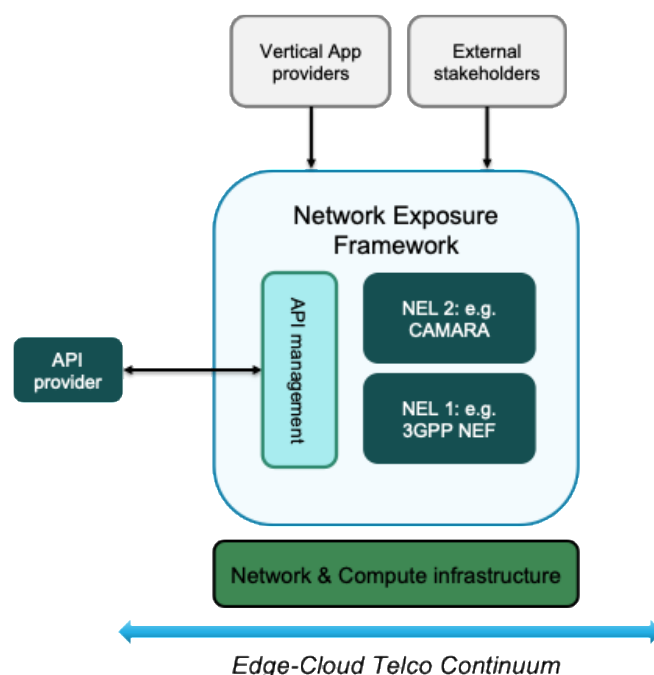


Figure 3-1. High-level overview of the UNITY-6G Network Exposure Layer.

More in detail, Figure 3-2 represents the details of the UNITY-6G Network Exposure Framework. At the bottom of the architecture, the infrastructure domains are represented, including the RAN, transport network, core network, and vertical application domain. Each domain exposes specific capabilities through domain-oriented APIs and control interfaces. In the RAN domain (details in Sections 3.1 and 3.3), the framework integrates Wi-Fi and RAN monitoring data APIs, monitoring information exposure services, and AI-driven resource allocation functions, together with O-RAN Near-RT RIC components supporting xApps and custom APIs. In the transport domain (details in Section 3.4), the framework incorporates a transport exposure API built on top of the Network Slice Controller that aggregates metrics and information from the transport domain, i.e., slice-related information coming from the NSC, as well as transport topology information coming from the SDN controller. Within the core domain (details in Section 3.2), 3GPP NEF functions and monitoring information exposure services provide access to core-network events, QoS management, and operational metrics. The vertical application domain (details in Section 3.5) integrates service orchestration functions and CAMARA-based APIs to expose edge and application lifecycle management capabilities. Supporting functions for API management and monitoring/observability are also included (details in Section 3.6), enabling lifecycle management, policy enforcement, analytics, and operational supervision of exposed APIs and services. Finally, at the top of the architecture, the framework exposes APIs and services toward consumers, external stakeholders, and vertical application providers (see Section 3.7).

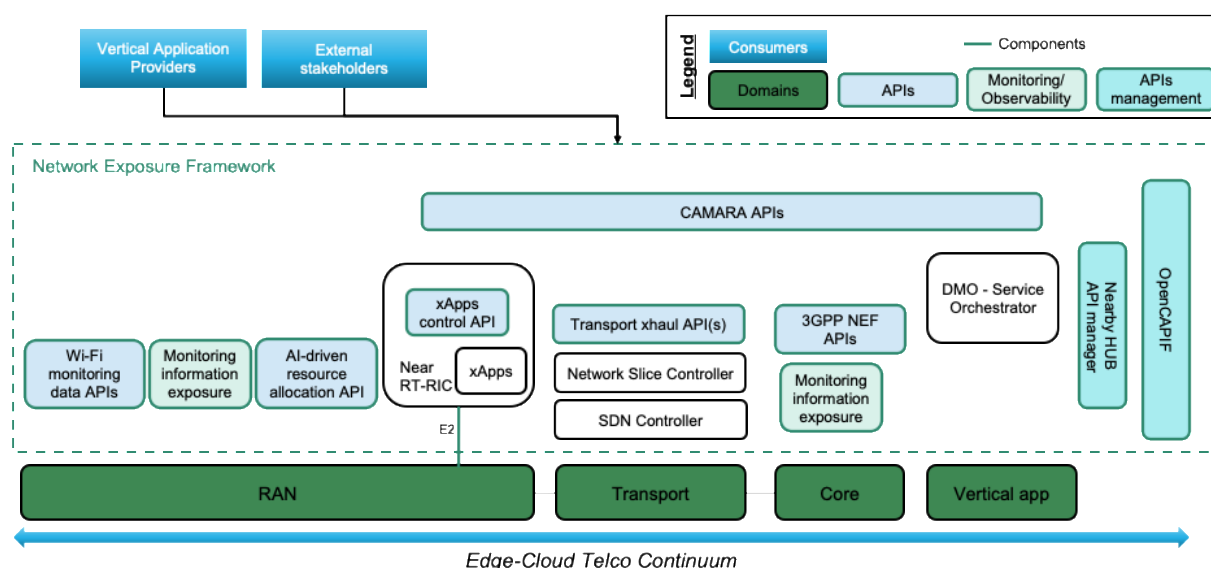


Figure 3-2: UNITY-6G Network Exposure Framework.

3.1 O-RAN

This sub-section deals with UNITY-6G components devoted to the exposure of information from the 3GPP RAN domain, considering also O-RAN interfaces and components. Among such components, a monitoring information tool is considered for the exposition of metrics from the gNB/DU OCUDU software. This software also implements the O-RAN E2 application protocol (E2AP), which handles communication setup, and messaging between the Near-RT RIC and E2 nodes (i.e., CU and DU). These E2 nodes allow the exposure of different metrics and actions of the different service models Key Performance Measurements (KPM), RAN control (RC) and Cell configuration and control (CCC). Regarding O-RAN, UNITY-6G has also explored the definition of a custom application programming interface (API) allowing the remote management of xApps. Finally, an AI-driven API for resource demand forecasting in gNBs (i.e., PRB utilization) is also described.

3.1.1 E2

The E2 interface [35] enables run-time AI-driven RAN profiling and management via the near-RT RIC, as introduced in Section 2.1.1.1. In the context of UNITY-6G, possible E2 extensions are being analysed to be provided on top of OCUDU. At the time of writing, the O-CU/O-DU includes a standard-compliant implementation of the E2 interface, including the E2AP and E2SM, as detailed in Section 5.1. More relevantly to the goals of UNITY-6G, the following features are provided by the E2 interface implementation of OCUDU [36]:

- **E2SM-KPM:** run-time RAN metrics can be obtained through this service. The implementation supports all report service styles (1-5), as well as a wide range of metrics, summarised in Table 3-1. These metrics are very relevant for profiling the performance, correct operation and behaviour of the RAN. Support to additional metrics will be studied as required by UNITY-6G.

Table 3-1: Overview of the E2SM-KPM support of the SRS O-CU/O-DU solution, part of the open-source OCUDU project.

Metric type	Supported metrics	Defined by
Latency	DRB.AirIfDelayUI, DRB.RlcDelayUI, DRB.PdcpReordDelayUI, DRB.RlcSduDelayDI	3GPP
Performance	RRU.PrbUsedDI, RU.PrbUsedUI, DRB.UEThpDI, DRB.UEThpUI, DRB.PacketSuccessRateUlgNBUu, DRB.RlcPacketDropRateDI, RACH.PreambleDedCell	3GPP
	DRB.RlcSduTransmittedVolumeDL, DRB.RlcSduTransmittedVolumeUL, DRB.RlcPacketDropRateDLDist	O-RAN Alliance
Configuration	RRU.PrbAvailDI, RRU.PrbAvailUI, RRU.PrbTotDI, RRU.PrbTotUI	3GPP

- **E2SM-RC:** run-time RAN control and (re)configuration actions can be implemented via this service. The implementation supports control styles 2 and 3, with the control actions summarised in Table 3-2. These RAN control actions are considered quite relevant to UNITY-6G, yet additional ones could be added to enhance the near-RT control-loop if deemed necessary to attain the PoC goals.

Table 3-2: Overview of the E2SM-RC support of the SRS O-CU/O-DU solution, part of the open-source OCUDU project.

Control action name	Control action description
Control Service style 2: cell-level configuration and control	
Slice-level PRB quota	Control the Radio Resource Management (RRM) policy for slice-specific PRB quota allocation
Control Service style 3: Connected mode mobility	
Handover control	Control the handover of the primary cell and subsequent secondary cell reselection
Conditional Handover Control	Control the handover involving target primary cells

- **E2SM-CCC:** run-time cell-level report and control actions can be implemented via this service. The implementation supports control style 2, with the control actions summarised in Table 3-3.

Table 3-3: Overview of the E2SM-CCC support of the SRS O-CU/O-DU solution, part of the open-source OCUDU project.

Control action name	Control action description
Control Service style 2: radio resource allocation control	
O-RRMPolicyRatio	Report or control the slice-specific resource usage quota for a given cell

3.1.2 Monitoring information exposure

The operation of a mobile network requires constant performance monitoring [17] to observe its performance and perform corrective actions to improve the network performance. The importance of the monitoring aspect of the network is reflected also in the O-RAN architecture, as explained in previous section, where xApps hosted by the Near-RT RIC can receive RAN KPM metrics and issue control actions to perform close-loop optimizations. Aligned with the introduction of novel networking paradigms into the management of mobile networks, the provision of such monitoring information capabilities requires the adoption of appropriated MANO procedures to automate its deployment. In the literature, there are several works dealing with RAN monitoring aspects, beyond O-RAN architecture. The authors in [18] focus only on RAN data collection by enabling inline execution of custom codelets based on eBPF, requiring the modification of RAN functions to include such collection capabilities. In [19], the authors present a very comprehensive monitoring system considering RAN performance. However, this system is not orchestrated, that is, it is manually configured and executed and it is not cloud-native, hindering its portability. The monitoring system proposed in [20] is cloud-native, but it is not considering the challenges associated to automating the deployment in a scenario with multiple gNBs.

Taken this into account, the UNITY-6G project is working in the development of a practical scalable cloud-native data collection and monitoring system for the RAN system based on the OCUDU open-source software. This monitoring system works beyond O-RAN procedures able to collect information directly from OCUDU processes. The entities of this monitoring system are included in a Helm chart and later packaged as an ETSI NFV NS package, so an ETSI NFV MANO platform can be used to automate the deployment while providing flexibility, in terms of allowing the support to monitor an arbitrary number of gNBs instances, and portability capabilities to be installed in any PoP. The work in [21] presented the initial version of this monitoring information exposure system. In particular, two different configuration blueprints (i.e., Helm Charts) complementing those devoted to the deployment of the 5G mobile network were created. The first one, enhanced the blueprint used to deploy the srsRAN gNB entity by including an additional pod containing the *metric server* component of the srsRAN suite devoted to collect the generated RAN metrics. Second, the RAN monitoring system blueprint starts an instance of InfluxDBv2 and Grafana to store and visualize the RAN data, respectively. When deploying this blueprint, the user specifies the number of gNBs, prompting an embedded Kubernetes job within the blueprint to dynamically create the corresponding data buckets in the InfluxDBv2. Another embedded Kubernetes job configures multiple data sources and dashboards in Grafana to enable independent data representation for each gNB, visualizing received RAN metrics such as downlink (DL)/ uplink (UL) bitrate, Modulation and Coding Scheme (MCS), Channel Quality Indicator (CQI), and Signal to-Noise Ratio (SNR).

After an update of the srsRAN code (25/10 release), now OCUDU project, the gNB component changed the way of exposing metrics (but also added more metrics) and the *metric server* component was deprecated. Currently, at runtime, the OCUDU gNB/DU process opens a TCP socket and continuously exposes detailed runtime metrics in JSON format. Telegraf component connects to this socket, parses the incoming data, and write it to an InfluxDBv3 database instance, which can then later be visualize in the provided Grafana dashboard, as depicted in Figure 3-3 and Figure 3-4. Metrics are available for several layers, including RU,

DU low, MAC, scheduler, RLC and PDCP, and also the resource consumption of the gNB/DU process in terms of CPU, RAM, power. After this code update, the monitoring exposure system consists of a single blueprint containing Telegraf pods, influxDBv3 pod and the Grafana pod system³. During deployment phase, the available parametrization in this blueprint allows the definition of an arbitrary number of gNBs by specifying a name to identify the gNB/DU instance and the IP address/port where the gNB/DU instance exposes the metrics. Then, for each specified gNB/DU instance a dedicated Telegraf pod is created, and the information is stored under a single data bucket in InfluxDBv3.



Figure 3-3: OCUDU RAN Metrics dashboard



Figure 3-4: OCUDU Performance Metrics dashboard

³ https://gitlab.com/ocudu/ocudu_elements/ocudu_helm/-/tree/main/charts/grafana-ocudu

3.1.3 API for remote xApp management

As part of the work explained in Section 2.1.2.1, on the automation of the cloud-native deployment of OSC Near-RT RIC, a custom REST-based API has been developed to enable the remote lifecycle management and monitoring of xApp compatible with the capabilities of OCUDU software. This API runs in the “xApp Executor pod” of the Near-RT RIC and exposes end points for file management, process execution control, and data logging retrieval. This API allows users to upload new xApps as Python scripts to the “xApp Executor pod”, check their configuration parameters, run them as independent processes according to their lifetime (i.e., short-lived action xApps versus long-lived monitoring process xApps), monitor their status, and access their execution logs. The API is designed to support asynchronous operations, reliability, and persistence across xApp executions, enabling remote and automated closed-loop operations, which is particularly convenient for distributed cloud-based orchestration or the integration of AI- driven pipelines without requiring direct shell access to the host running the components. The “xApp Executor pod” of the Near-RT RIC is exposed as a Load-Balancer Kubernetes Service Type, so external users can have access to this API to remotely control the execution of xApps in the assigned Near-RT RIC instance, created upon-request. Table 3-4 presents a short description of the available endpoints.

Table 3-4. xApp remote lifecycle management API description

Method	Endpoint	Description
GET	/files	Provides the list of xApp scripts available in the xApp executor pod
GET	/fileinfo/<xapp_file>.py	Provides information about the parameters considered for the execution of the <xapp_script_name>.py file
GET	/db/processes	Shows the list of xApp in execution, specifying the state (running or terminated), the parameters used for the execution and the current PID.
GET	/db/process/<pid>	Filters information of previous endpoint (/db/processes) by PID
GET	/stdout/<pid>	Show the last 50 lines of the logs of the execution of the xApp associated with the given <pid> value
POST	/upload	Allows the user to upload an xApp script into the xApp executor pod. The name of the script is specified in the request (e.g., curl -X POST -F "file=@xapp_script.py" http://<IP_addr>:5001/upload)
POST	/run	Allows the user to execute an xApp. The parameters are included in the requests (e.g., X curl -X POST http://<IP_addr>:5001/run -H "Content-Type: application/json" -d '{"filename": "generic_ccc_xapp.py", "mode": "short", "args": {"sst": 1, "--maxprb": 75, "--e2_node_id": "gnbd_001_001_00019b_0"}'}')

POST	/stop/<pid>	Allows the user to stop the execution on an xApp identified by PID
-------------	-------------	--

3.1.4 AI driven resource allocation API

The multivariate probabilistic forecasting solution for radio resource allocation has been containerized, and AI/ML model information has been exposed to other parts of the networks by developing a REST/API.

Traditional forecasting approaches for network resource allocation are commonly based on deterministic and often univariate models, such as conventional LSTM architectures, which provide a single predicted value for future demand. Although these methods can learn temporal patterns from historical data, they generally analyze each resource stream independently and fail to capture the complex correlations that exist among multiple tenants, cells, or network slices. More importantly, deterministic models do not provide information about the uncertainty associated with future observations, leading to overconfident predictions and potentially inefficient resource allocation decisions in highly dynamic wireless environments. To overcome these limitations, this work adopts a multivariate probabilistic forecasting framework. Unlike traditional single-point estimators, probabilistic forecasting aims to estimate the full probability distribution of future resource demands rather than only their expected values. This paradigm shift enables the model to quantify uncertainty through prediction intervals and confidence ranges, providing a more reliable representation of future network behavior.

Additionally, multivariate probabilistic models jointly process multiple correlated time series and contextual features, such as PRB utilization across tenants, time-of-day patterns, and weekly traffic dynamics. By learning both temporal dependencies and inter-series relationships, models such as GPVAR and TFT can better characterize complex network conditions and improve forecasting robustness. As a result, probabilistic forecasting supports more adaptive and risk-aware resource management strategies. The detailed description of the algorithm tools is omitted in this deliverable and will be presented with further detail in Deliverable D4.1.

The AI-driven resource allocation interface is realized through a containerized radio application (rApp) integrated with a RESTful API framework. Even though it has been initially designed for the Non-Real Time RIC, it can be considered to expose AI-based multivariate probabilistic forecasting to other network entities. This API facilitates the exposure of advanced multivariate probabilistic forecasting models, such as Temporal Fusion Transformer (TFT) and Gaussian Process Vector Autoregression (GPVAR), to the O-RAN ecosystem. By leveraging cloud-native principles, the API bridges the gap between theoretical predictive modelling and real-time network orchestration. To ensure platform independence and consistent behavioural performance across development and production environments, the resource allocation AI/ML is encapsulated using Docker. All dependencies, including specialized libraries (e.g., PyTorch Forecasting), configurations and runtimes, are packaged into a lightweight Docker image. The containerized RRM solution contains analytical engine for AI/ML model training/inference. The API endpoints are illustrated in Table 3-5.

Table 3-5: End to End Resource Allocation API Endpoints and Descriptions

Method	Endpoint	Description
GET	/status	Checks whether the containerized rApp is running and whether the internal model/pipeline is ready.
POST	/ingest	Receives required data made available through authorized data sources
POST	/training	Starts data preprocessing, model training , and validation using historical network data.
POST	/prediction	Internally performs preprocessing, and multivariate probabilistic forecasting
GET	/metrics	Exposes performance indicators (e.g. Quantiles and/or AI model weights) to authorized management/monitoring entities
POST	/retraining	Model retraining using accumulated historical network data
GET	/info	Exposes metadata of the deployed AI/ML model, such as model type, version, etc.

Network Service Exposure via Swagger API: The framework utilizes Swagger (OpenAPI) to provide a structured, user-friendly interface for interacting with the rApp's backend and other network entities. This integration simplifies lifecycle management, including testing, deployment and scaling, without requiring sophisticated external tools. As said, this API can expose critical AI/ML metrics for the RRM analytics (e.g., the predicted quantiles for the PRB utilization in the multi-step ahead horizon, AI-related metrics or performance metrics, such as MSE or CRPS or AI/ML model weights), allowing risk-aware decisions in other network entities and even allowing offline retraining of the model, when needed. By providing immediate feedback on API calls, the system ensures operational.

As an example, two core functional endpoints that handle the training and inference phases of the AI models are illustrated below by Swagger API

1. /Training (POST API):

- **Functionality:** Orchestrates the training and validation of selected multivariate estimators.
- **Input Parameters:** Uses the data ingest through /ingest endpoint for model training and users may specify the estimator (GPVAR, TFT, or LSTM), the number of input series to complete data input for model training.
- **Output:** Upon completion, the API stores the trained model and returns metrics such as total training time or model weights.

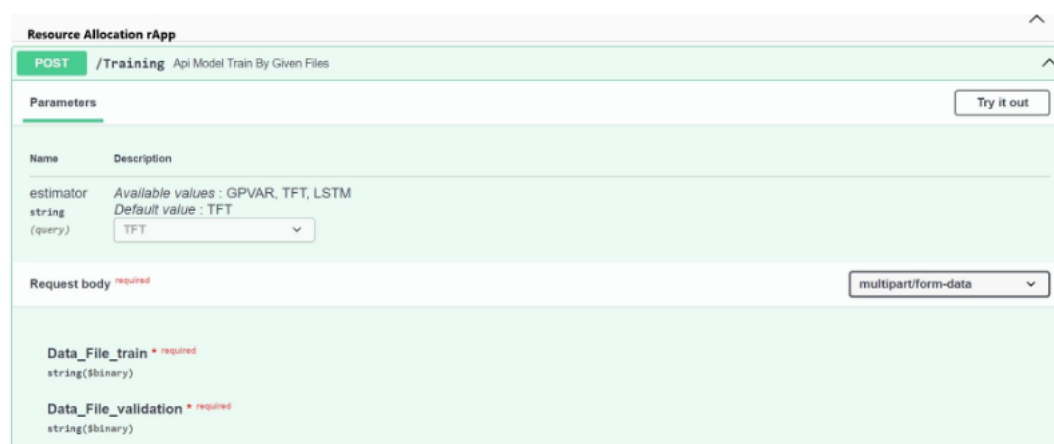


Figure 3-5: Swagger Training Functionality

2. /Prediction (POST API):

- **Functionality:** Executes resource demand forecasting using the previously trained and saved models.
- **Workflow:** The backend container automatically loads the relevant model based on user parameters, processes the test dataset, and generates multivariate forecasts.
- **Performance Metrics:** The response includes prediction time and accuracy metrics, such as Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) or probabilistic metrics (e.g., CRPS), enabling real-time analysis of the estimator's reliability, and expose the necessary metrics using /metrics endpoint to the authorized management/monitoring entities

The integration of a Swagger interface promotes agility and scalability. It allows for parallel execution of training and prediction tasks, enabling to test multiple data combinations and hyperparameter configurations (via AutoML tools like sOptuna) without creating bottlenecks in the network operational behaviour.

3.2 CORE NETWORK

This sub-section deals with UNITY-6G components devoted to the exposure of information from available mobile core network instances. On one hand, a monitoring information tool is considered for the exposition of metrics provided by some network functions (e.g., AMF, SMF, PCF, UPF) in the Open5GS software, which can be exploited locally to create close-loops using the tuples defined in the UNITY-6G architecture. On the other hand, we present the 3GPP NEF network function, which works as a gateway for authorised external application functions allowing the exposition of network events or capabilities, such as QoS control, session management, and analytics, through standardized APIs while hiding the complexity of the mobile core by acting as a mediator, that is, translating external API requests into internal service calls and vice versa. To further enhance this exposure framework, UNITY-6G incorporates the OpenCAPIF (Common API Framework) implementation. The goal is to provide a unified and standardized management plane that governs the consumption of these Northbound APIs. By leveraging OpenCAPIF, the project ensures that the exposure of core

network capabilities whether originating from the NEF or direct monitoring tools is conducted in a secure, authenticated, and controlled manner.

3.2.1 Monitoring information exposure

Open5GS, the open-source mobile core network implementation considered in UNITY-6G, as commented in Section 2.2.2, provides the possibility of activating a small HTTP server exposing performance metrics of relevant network functions, namely AMF, PCF, SMF, UPF, to expose internal state and performance data. When enabled, each network function acts as a Prometheus target, exposing a `/metrics` endpoint on a configurable TCP port so network engineers can have a granular visibility into application-specific and system resource of the instantiated cloud-native mobile core. The application-specific metrics can be useful for monitoring the health of a 5G mobile core instance. Examples of such metrics can be a profile of authentication requests (AMF), the number of available gNBs (AMF), the number of active UEs (SMF), as depicted in Figure 3-6, the number of QoS flows (SMF), number of N4 sessions (UPF), number of active sessions (PCF, SMF, UPF). The system resource metrics provide information about standard process-level metrics like CPU and actual used memory. The collection of such metrics allows the creation of close-loop ensuring to diagnosis problems and trigger changes in the mobile core configuration reducing human intervention.

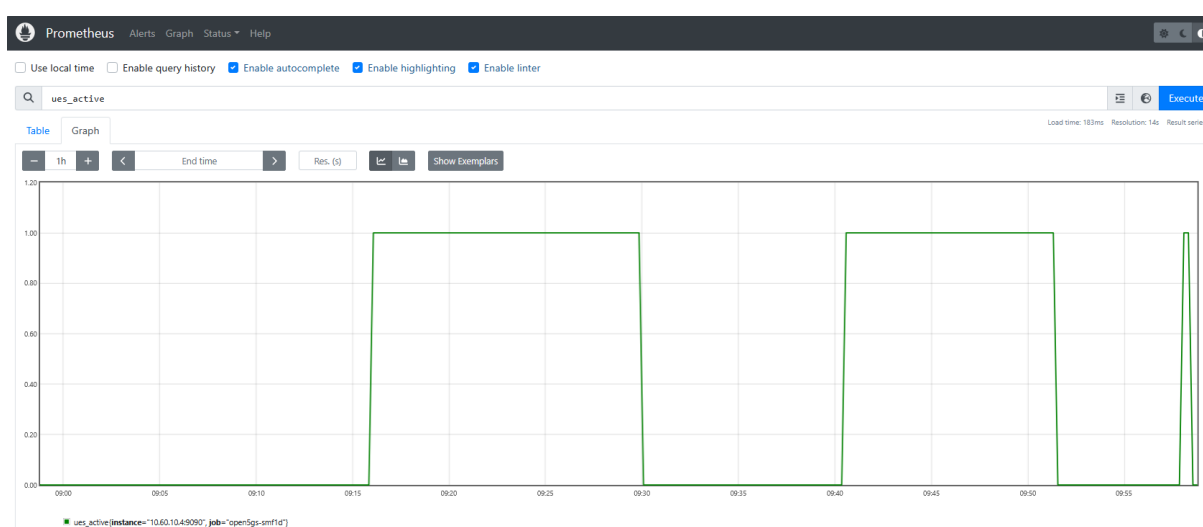


Figure 3-6: Active UE profile collected by Prometheus from SMF instance

In order to exploit such capabilities, the cloud-native artefacts proposed in Section 2.2.2 have been enhanced to activate the metrics servers exposing the available metrics of the different network functions of Open5GS. The artefacts of the mobile core are complemented with an additional artefact to create a specific Prometheus cloud-native instance devoted to collect the data coming from the configured mobile core instance. It is worth to mention that this scraping process is possible by the exposition of an additional port in the pods created for the different network functions, so metrics can be collected remotely. Like the artefact of the mobile core, the Prometheus artefacts are encapsulated as a ETSI OSM NS package, so the parametrisation capabilities built into both the Helm templates and the OSM descriptors, allows the customisation at instantiation time to include the scraping targets (e.g., exposed IP address and port belonging to the network function). This enables the coexistence of multiple Prometheus instances, each one devoted to the corresponding mobile core instance.

3.2.2 3GPP NEF APIs Network Exposure

The Network Exposure Function (NEF) serves as the fundamental gateway in the UNITY-6G architecture, enabling secure and standardized communication between the 3GPP 5G Core (5GC) and external Application Functions (AFs). By abstracting the complexities of the core network into Northbound RESTful APIs, the NEF allows the project to implement service-aware orchestration and real-time network adaptation. This section details the implementation and integration logic of two critical 3GPP NEF APIs, `MonitoringEvent` and `ASSessionWithQoS`, which are essential for achieving the project's Key Performance Indicators (KPIs) in dynamic and high-demand scenarios. The `MonitoringEvent` API is implemented to provide the network with "situational awareness," allowing external applications to subscribe to and receive notifications regarding User Equipment (UE) status, such as location changes, reachability, and connectivity loss. Within the context of UNITY-6G, this is vital for maintaining service continuity in the disaster-recovery and mobility use cases. The `ASSessionWithQoS` API facilitates the active management of the communication fabric by allowing applications to request specific Quality of Service (QoS) levels for active data sessions. This ensures that high-priority flows such as those required for Digital Twin synchronization or critical emergency communications receive the necessary bandwidth and latency guarantees dynamically, based on the real-time intent of the service provider.

3.2.2.1 MonitoringEvent API

For an AF to retrieve the location of a UE, appropriate event reporting parameters must be configured and a `Nnef_eventExposure_Subscribe` request to NEF must be issued. This request initiates a sequence of operations across various Network Functions (NFs) in the 5G Core, ultimately resulting in a location notification being sent from the NEF to the AF. The high-level approach is illustrated in Figure 3-7.

The baseline for our implementation builds upon prior work that introduced NEFSim, an open-source NEF simulator designed to enable testing of the northbound API in a simulated environment [60]. Building on this foundation, our work now extends NEFSim logic to support cloud-native deployment and real-time event handling and integration with AMF-derived data, utilizing the southbound interfaces. This section details the architecture and functional design of the proposed implementation. Figure 3-7, depicts the high level overview of the proposed NEF event exposure in 5GC.

Design Overview and Actions Flow

The system is composed of two main modules: the AMF Log Agent (backend) and the NEF Monitoring Event API (frontend). Together, they implement a fully decoupled monitoring pipeline that remains transparent to the 5GC while still compliant with 3GPP TS 29.522 event monitoring. First, the AMF Log Agent (back-end) monitors UE registration activity by parsing AMF container logs. This agent employs a configurable polling strategy, customizable by the end-user, scanning for registration messages that contain relevant UE identifiers (e.g., IMSI). Upon detecting such an event, the agent parses the logs to extract relevant registration details:

- Raw registration metadata, including timestamps and event context.
- Parsed UE location information, formatted in alignment with 3GPP data models.

These data structures are constructed as JavaScript Object Notation (JSON) objects and stored in an external MongoDB database in two collections: `ue_events` and `location_info`. This separation enables persistent, queryable storage that is accessible by external NEF components. Secondly, the `MonitoringEventSubscription` message, as defined in 3GPP TS 29.122, enables an application server to request monitoring of specific network events for a target subscriber. This subscription is characterized by several critical attributes: `MonitoringType`, which specifies the class of events to be monitored; `LocationType`, indicating the current or last known location of a UE, `msisdn`, uniquely identifying the subscriber of interest, `maximumNumberOfReports`, which constrains the total number of notifications generated for the subscription; `notificationDestination`, specifying the endpoint where monitoring notifications should be delivered; and `repPeriod`, which defines the periodicity for reporting events. Upon occurrence of a subscribed event, the network generates a `MonitoringNotification` message, which encapsulates one or more `MonitoringEventReport` instances.

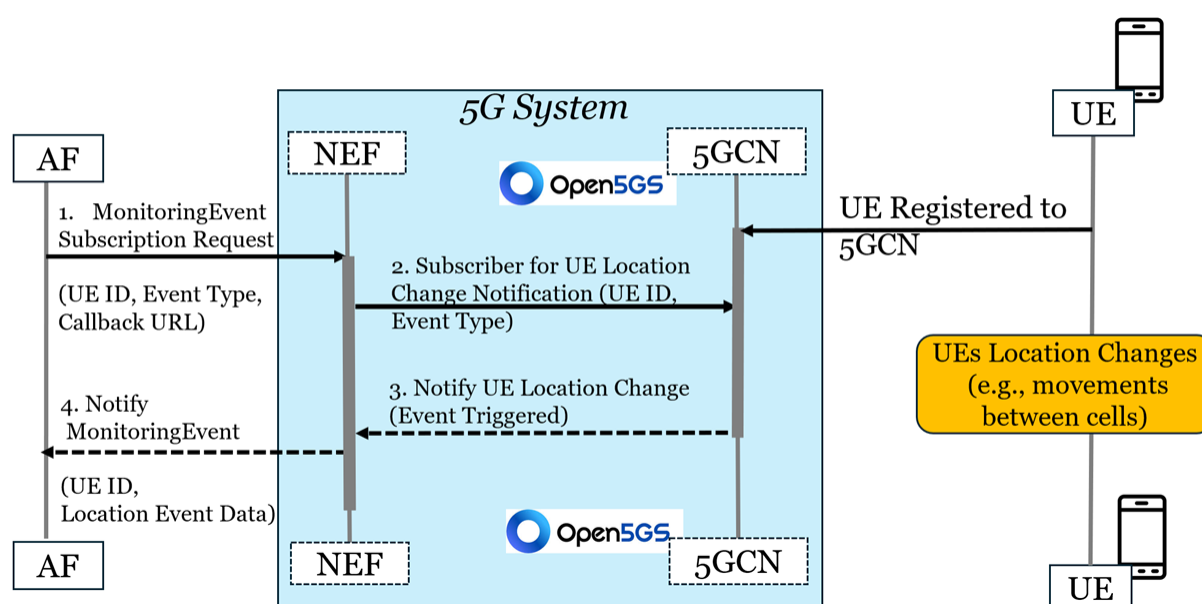


Figure 3-7: High level overview of NEF Event Exposure in 5G Core.

Each report provides detailed information regarding the monitored event and, if location spotted, includes a `LocationInfo` object that conveys the subscriber's location according to the prescribed `LocationType`. This mechanism ensures that applications can obtain timely and structured information about network events along with associated spatial context, enabling sophisticated analytics and context-aware services. In parallel, log monitoring of AMF provides a mechanism to observe UE registration, deregistration, and PDU session establishment events within the 5G core network. By analyzing these logs, it is possible to extract network-derived attributes such as Tracking Area Identity (TAI), Cell ID, or other signaling context, thereby enabling coarse-grained location retrieval of the subscriber. This approach allows applications to correlate UE registration or session events with approximate location information, facilitating tracking of subscriber presence and network attachment patterns. However, the method is inherently limited to events that generate AMF-level signaling; RAN-level mobility, may not trigger additional AMF logs, resulting in potential gaps in location granularity. To address this limitation, log monitoring can be complemented with RAN signaling

events, AMF context updates, or dedicated 5G positioning mechanisms, providing higher-fidelity tracking while maintaining compatibility with Monitoring Event API subscriptions. Thirdly, NEF Monitoring Event API frontend exposes an Asynchronous Representational State Transfer (RESTful) Monitoring Event API interface to AFs using OpenAPI specifications. It supports standard operations for subscription lifecycle management:

- POST: Creates a monitoring subscription with UE target ID and event parameters.
- GET: Retrieves subscription information by AF or subscription ID.
- DELETE: Cancels a monitoring subscription.

When a new POST subscription request is submitted, the NEF performs the following operations: First, it creates and stores an association between the subscription and its corresponding request parameters in a MongoDB database. This is the same database utilized by the AMF Log Agent to support other Hypertext Transfer Protocol (HTTP)-based operations. Subsequently, the NEF asynchronously queries the database for UE identifiers (IMSI) that match the subscription criteria. If a match is identified, the NEF retrieves the corresponding location information and delivers it to the AF via a predefined callback interface. This response is structured in accordance with 3GPP specifications. This decoupled architecture allows the NEF to provide location information without requiring real-time interaction with the 5G Core network. Such an approach simplifies system integration and enhances interoperability with different opensource 5G core network solutions. In terms of the cloud-native deployment, the entire system is designed using state-of-the-art cloud-native technologies and principles. All components are containerized with Docker and orchestrated via Docker Compose. MongoDB serves as the central data store, with endpoints configurable at runtime. This design ensures platform independence, supporting integration with any compliant 5GC deployment without modification to the core network.

Finally, API exposure validation and evaluation setup part, is being presented through practical deployment and integration of the NEF Monitoring Event API within a cloud-native 5G testbed. The implementation is based on opensource components and deployed entirely in a containerized environment including the following elements:

- 5G Core: We employ open5GS as the reference 3GPP-compliant 5G Core implementation. It provides the essential network functions, such as the Access and Mobility Management Function (AMF) and Session Management Function (SMF), enabling realistic simulation of UE registration and mobility procedures.
- RAN Emulator: The Radio Access Network (RAN) is emulated using UERANSIM, an open-source tool capable of simulating both User Equipment (UE) and gNodeB (gNB). It allows for the generation of realistic NAS signalling and interaction with the core network.
- AMF Log Agent: A custom-developed backend component designed to parse logs produced by the AMF. It extracts relevant registration and location data, which is then structured and stored for use by higher-layer services such as the NEF.
- MongoDB: A NoSQL database used to store registration metadata extracted from the AMF logs. This includes UE registration events, IMSI identifiers, and associated location information required by the NEF API.

- NEF Monitoring Event API: A RESTful API endpoint implemented to conform with 3GPP NEF specifications. It supports monitoring event subscriptions and exposes UE location data through a documented Swagger interface for seamless AF integration.

Each component runs in isolated Docker containers, simulating a cloud-native 5G system. All components were developed and deployed using Docker. The environment configuration files enable full reproducibility, and the NEF API is exposed via the Swagger use interface (UI) for ease of interaction. This modularity supports easy redeployment in other cloud or on-premise 5G setups. Moreover, the system workflow unfolds in several stages, as Figure 3-8 depicts. The first step begins with a simulated UE via the UERANSIM initiating a registration request to the Open5GS core. As part of this process, registration logs are generated within the AMF container.

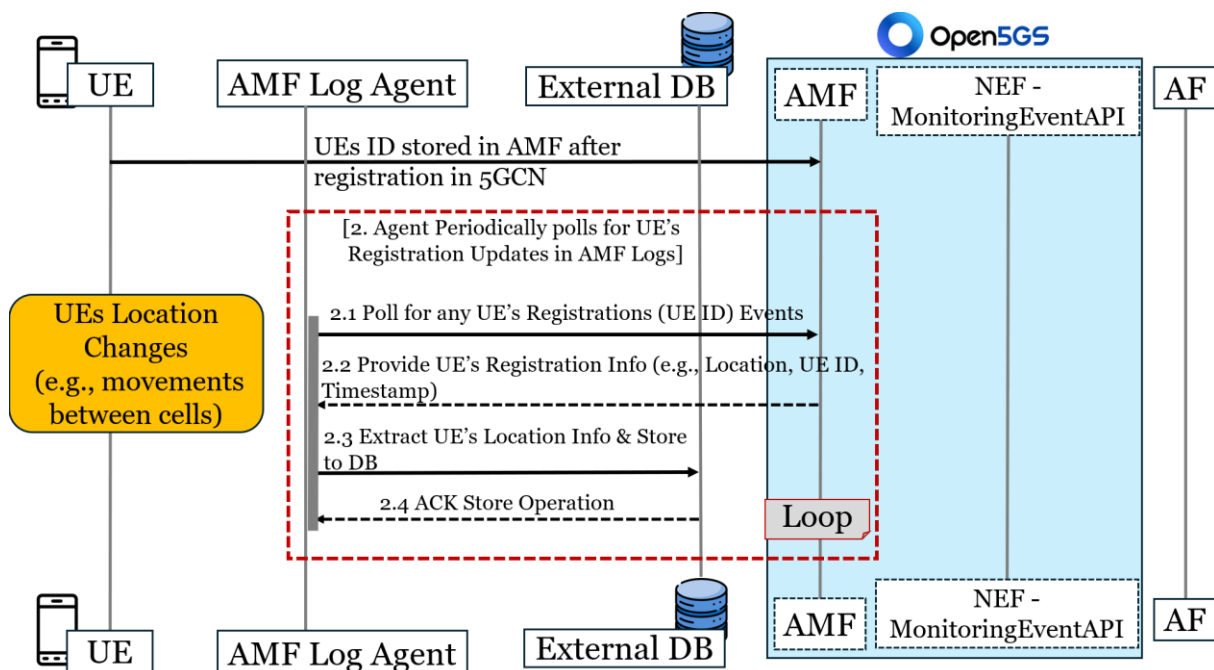


Figure 3-8: Back-end Architecture for AMF Log Agent

Then, step 2 includes the AMF Log Agent, a custom backend component designed to continuously monitor AMF activity, which polls the AMF log files to identify registration events in near real-time. Upon detection, the agent extracts and parses critical data including the IMSI (International Mobile Subscriber Identity), PLMN (Public Land Mobile Network) ID, and Cell ID and formats the information into structured JSON objects. These objects are then persistently stored in a MongoDB instance, specifically within collections dedicated to user events and location information. In addition, in step 3, in parallel, the external AF, integrated via a Swagger-based API interface, submits a POST request to the NEF Monitoring Event API. This request includes the target UE identifier along with subscription parameters such as event type and reporting configuration. The NEF API processes the request, validates the input, and stores the subscription metadata in the shared MongoDB database for future reference and matching against registered UEs. Furthermore, under step 4, once the subscription is stored, the NEF Monitoring API asynchronously checks for matching UE activity by querying the MongoDB collections populated by the AMF Log Agent. If a match is found between a stored subscription and a previously parsed UE registration event, the NEF constructs the location

notification message. This message is then transmitted to the AF via a predefined HTTP callback interface. Finally, in step 5, after receiving the notification, the AF can manage the subscription lifecycle through additional API operations. Specifically, the AF can retrieve subscription details using a GET request or cancel the subscription entirely with a DELETE request. Both operations are executed by the NEF and confirmed through appropriate API responses, thus finalizing the monitoring session.

F. Base Endpoints

The below table depicts 4 different HTTP-based methods and their endpoints as an AF can hit. Also, two POST request examples are given, one for 'current location' and another for 'last known location', in Figure 3-9.

Table 3-6: NEF Monitoring Event API REST-supported Methods

Method	Endpoint	Description
POST	/3gpp-monitoring-event/v1/{scsAsId}/subscriptions	Create a MonitoringEvent subscription (current location)
GET	/3gpp-monitoring-event/v1/{scsAsId}/subscriptions/	Read all of the active subscriptions for the AF
GET	/3gpp-monitoring-event/v1/{scsAsId}/subscriptions/{subscriptionId}	Read an active subscription for the AF and the subscription Id
DELETE	/3gpp-monitoring-event/v1/{scsAsId}/subscriptions/{subscriptionId}	Deletes an already existing subscription

```

{
  "accuracy": "CGI_ECGI",
  "msisdn": "001010143245445",
  "notificationDestination": "http://test_server:8001",
  "monitoringType": "LOCATION_REPORTING",
  "maximumNumberOfReports": 3,
  "locationType": "CURRENT_LOCATION",
  "repPeriod": {
    "duration": 20
  }
}

```

```

{
  "msisdn": "001010143245445",
  "notificationDestination": "http://test_server:8001",
  "monitoringType": "LOCATION_REPORTING",
  "locationType": "LAST_KNOWN_LOCATION",
}

```

Figure 3-9:a) Example Subscription POST Request - CURRENT LOCATION b) Example Subscription POST Request - LAST KNOWN LOCATION

3.2.2.2 AsSessionWithQoS API

The implementation of the AsSessionWithQoS API within UNITY-6G focuses on demonstrating the capability of the Network Exposure Function (NEF) to mediate high-level application requirements into granular 5G core policies. By interacting with the Open5GS Policy Control Function (PCF), the NEF applies Policy and Charging Control (PCC) rules to manage 5G QoS Identifier (5QI) flows for specific subscriber sessions. The primary objective of this implementation is to allow an external Application Function (AF) to establish and manage sessions with guaranteed service characteristics. The AF specifies the required QoS level (via 5QI) and additional policy constraints, such as bandwidth limits. The NEF acts as the critical bridge, translating these Northbound RESTful requests into standard Npcf_PolicyAuthorization calls toward the PCF to enforce policy authorization within the 5G system. Figure 3-10 shows the AsSessionWithQoS API sequence diagram in the 5G

architecture. The operational lifecycle of a QoS session follows a structured sequence of Northbound exposure and Southbound core interaction:

- **Session Establishment:** The process begins when the AF sends a POST request to the NEF to create a QoS subscription. The NEF derives the necessary policy parameters and establishes an Individual Application Session Context on the PCF. Once the PCF confirms the session ID, the NEF responds to the AF with a 201 Created status, confirming the active QoS enforcement.
- **Context Management and Retrieval:** During the session's lifetime, the AF can perform GET operations to retrieve specific subscription details or list all active subscriptions. The NEF maintains the linkage between the external subscription resource and the internal authorized policy state, ensuring consistent visibility of the network resources allocated to the service.
- **Termination and Resource Release:** When the high-priority service concludes, the AF issues a DELETE request. The NEF subsequently instructs the PCF to terminate the Application Session Context. Upon confirmation from the core, the NEF responds with a 204 No Content, ensuring that network policy and QoS resources are released and available for other services.

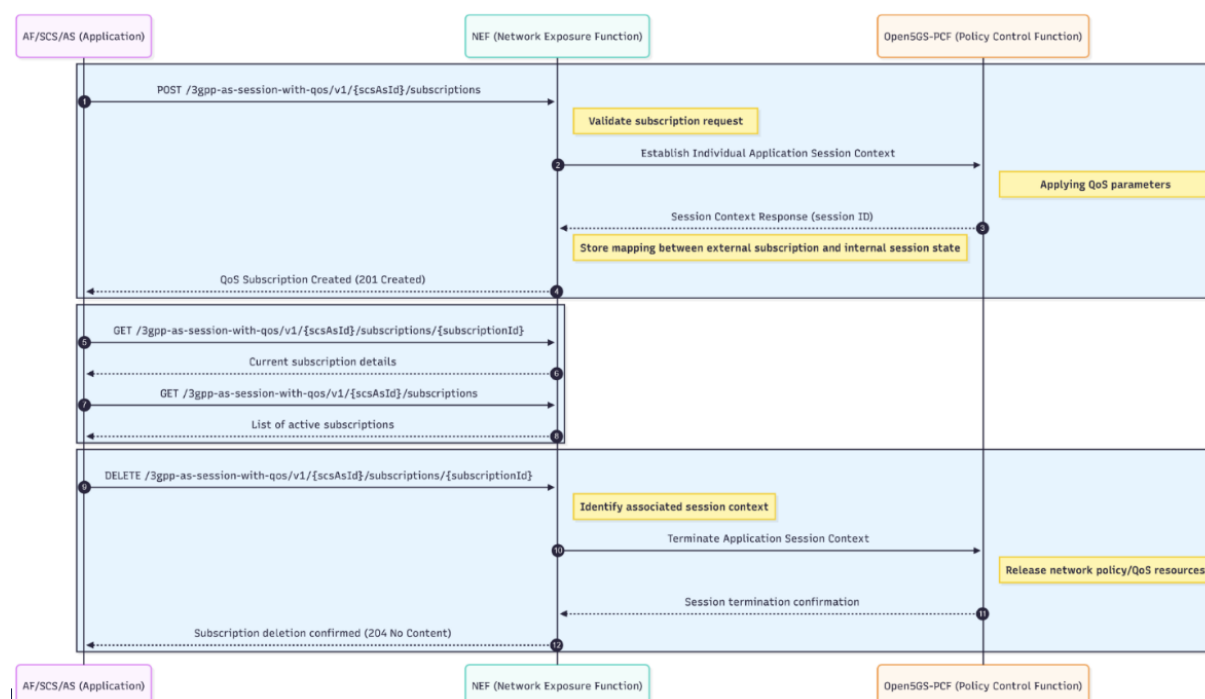


Figure 3-10: AsSessionWithQoS API Sequence Diagramm

Regarding Standards compliance, the following 3GPP Technical Specifications (TS) are used for the implementation of the AsSessionWithQoS API:

Table 3-7:3GPP technical properties for AsSessionWithQoS API

Technical Specification	Title	Version	Logic	Resource
Common Data Types	TS 29.122 Common Data Types	1.2.0-alpha.3	Standard 3GPP data types used in several APIs	3GPP TS.29.122 V17.2.0 T8

				reference point for Northbound API [40]
AsSession with QoS API	3gpp-as-session-with-qos	1.2.0-alpha.1	API to set up an application session with required QoS	[40]
Common Data Types for Service Based Interfaces	Common Data Types	1.3.0-alpha.2	Shared data types for 5G service-based APIs	3PP TS 29.571 Common Data Types for Service Based Interfaces, v17.2.0 [41]
PCF Policy Authorization Service API	Npcf_PolicyAuthorization Service API	1.2.0-alpha.2	Service for policy authorization in 5G networks	3GPP TS 29.514 V17.1.0; 5G System; Policy Authorization Service; Stage 3 [42]

3.3 NON-3GPP RAN

Exposure of metrics and controlling knobs is important to be able to observe the application requirement fulfilment and network behaviour, to manage and control the network impact on applications, as well as to maintain the application requirements. In this section we will describe the initial set of IEEE 802.11 monitoring metrics that we expose towards higher layer APIs, as well as the accessibility of certain controlling knobs for managing behaviour of the IEEE 802.11 RAN. This is just the initial list of possible metrics that can be exposed from the platform that we are using. Based on further details on the needed metrics and knobs (e.g. what parameters can be used to detect channel load in IEEE 802.11, what parameters can be used to detect interference in the channel etc) we will define the exact exposing metrics to higher APIs at a later stage during the project.

3.3.1 Internal openwifi APIs

Each IEEE 802.11 AP is equipped with an E2 agent that is able to talk to the Radio Network Controller (RIC) in both directions, exposing metrics as well as getting control commands. The initial version of the custom service model (SM) for IEEE 8 AP is already described in Section 2.3.1.3. The integration of the E2 interface with the openwifi APs is achieved by utilizing *netlink* interface as well as Linux *sysfs*. While *netlink* is used to expose specific monitoring parameters for the *sdrctl* command, the controlling command for openwifi interface, the *sysfs* is a pseudofile system provided by the Linux kernel to expose certain information about different kernel subsystems including hardware devices. As such, we have extended the *sysfs* to include also monitoring parameters from openwifi interface in both APs and STAs. *Sysfs* is used to read the monitoring parameters, while for the controlling knobs the *sdrctl* is used, which uses *netlink* socket API underneath. For controlling knobs, one needs to have a confirmation

if the actual control command has been applied before you move on. As such, *sdrctl* is chosen compared to *sysfs*.

In case of E2 interface termination the *sysfs* makes it easier to integrate and to collect the monitoring information. The E2 interface agent at the IEEE 802.11 will read a specific file for specific metric and pass it to the SM towards RIC. For controlling commands, the E2 interface just executes a *sdrctl* command to run certain controlling feature directly at the driver or physical layer of openwifi interface.

3.3.2 Exposure of Monitoring Metrics

Openwifi exposes an extensive list of metrics towards the higher layer APIs, in this case towards E2 interface termination agent. The metric lists and their definition is given in Table 3-8 for exposed physical and MAC layer parameters. Table 3-9 shows the exposed physical and MAC layer statistics and their definition.

Table 3-8: Exposed physical and MAC layer parameters

Metric	Definition
state_tx_queue	State of the transmit state machine for transmit queues
tx_ctrl_state	State of the state machine handling ACK and retransmission
BO_state	State of the CSMA/CA backoff state machine
NAV_state	State of the state machine tracking Network Allocation Vector
queue_index	Queue index for the current Tx packet
retx_status	Show if the retransmission is in progress or not
tx_finish_event	A pulse to indicate that ACK is received or the maximum number of retransmissions is reached
PHY_hdr_finish_event	A pulse to indicate that a PHY header is demodulated (checksum not necessarily OK)
rcv_pkt_finish_event	A pulse to indicate that the Rx packet is ended (checksum not necessarily OK)
PHY_tx_start_event	A pulse to indicate that the PHY transmission started
PHY_tx_end_event	A pulse to indicate that the PHY transmitter gives finishing signal
CCA_report	The CCA indicates that the channel is idle (1) or not (0)
pkt_tx_event	A pulse to indicate that a packet in a queue will be transmitted
pkt_retx_event	A pulse to indicate that a retransmission will be started
queue_gate_state	Open (1) / close (0) state of each queue

Table 3-9: Physical and MAC layer statistics

Statistic	Definition
total_tx_pkt	Total number of transmitted data packets reported in <i>openwifi_tx_interrupt()</i> both failed and succeeded
total_tx_pkt_no_ACK	Total number of transmitted data packets reported in <i>openwifi_tx_interrupt()</i> with no ACK received
total_tx_pkt_retx	Total number of transmitted data packets reported in <i>openwifi_tx_interrupt()</i> at different number of retransmission both failed and succeeded

total_tx_pkt_rtx_no_ACK	Total number of transmitted data packets reported in openwifi_tx_interrupt() at different number of retransmission with no ACK received
MCS_tx_pkt	MCS (10*Mbps) of transmitted data packet reported in openwifi_tx_interrupt() both failed and succeeded
MCS_tx_pkt_no_ACK	MCS (10*Mbps) of transmitted data packet reported in openwifi_tx_interrupt() with no ACK received
total_tx_mng_pkt	Total number of transmitted management packets reported in openwifi_tx_interrupt() both failed and succeeded
total_tx_mng_pkt_no_ACK	Total number of transmitted management packets reported in openwifi_tx_interrupt() with no ACK received
total_tx_mng_pkt_retx	Total number of transmitted management packets reported in openwifi_tx_interrupt() at different number of retransmission both failed and succeeded
total_tx_mng_pkt_rtx_no_ACK	Total number of transmitted management packets reported in openwifi_tx_interrupt() at different number of retransmission with no ACK received
MCS_tx_mng_pkt	MCS (10*Mbps) of transmitted management packet reported in openwifi_tx_interrupt() both failed and succeeded
MCS_tx_mng_pkt_no_ACK	MCS (10*Mbps) of transmitted management packet reported in openwifi_tx_interrupt() with no ACK received
total_rx_pkt	Total number of received data packets both with failed and correct FCS
total_rx_pkt_fail_FCS	Total number of received data packet with failed FCS
total_rx_mng_pkt	Total number of received management packets both with failed and correct FCS
total_rx_mng_pkt_fail_FCS	Total number of received management packet with failed FCS
total_rx_ACK	Total number of received ACK packets both with failed and correct FCS
total_rx_ACK_fail_FCS	Total number of received ACK packets with failed FCS
MCS_rx_pkt	MCS (10*Mbps) of received data packets both with failed and correct FCS
MCS_rx_pkt_fail_FCS	MCS (10*Mbps) of received data packet with failed FCS
MCS_rx_mng_pkt	MCS (10*Mbps) of received management packets both with failed and correct FCS
MCS_rx_mng_pkt_fail_FCS	MCS (10*Mbps) of received management packet with failed FCS
MCS_rx_ACK	MCS (10*Mbps) of received ACK packet both with correct and failed FCS

3.3.3 Exposure of Controlling Knobs

Openwifi exposes also several controlling knobs to control and manage the behaviour of the IEEE 802.11 radio access network. Table 3-10 shows all the controlling knobs that openwifi can expose to higher layers APIs.

Table 3-10: Openwifi configurable PHY and MAC parameters

Knob	Definition
CSMA/CA_on_off	Enable/disable CSMA/CA mechanism
LBT_thre	Update the listen-before talk (LBT) threshold per device
NAV_on_off	Network Allocation Vector enable/disable
DIFS_on_off	Enable/disable DIFS before the actual transmission
EIFS_on_off	Enable/disable EIFS before the actual transmission
CW_on_off	Enable/disable contention per device
CW_min_queue	Set CW min per queue
CW_max_queue	Set CW max per queue
retx_nr	Set nr or ReTx per device
ACK_tx_on_off	Enable/disable the ACK request on the transmitted packet
ACK_rx_on_off	Enable/disable response with an ACK on the receiver
rx_sense_thre	Update Rx threshold sensitivity per device
tx_atten_level	Update the Tx digital attenuation at certain values
tx_freq	Update transmitter frequency
rx_freq	Update receiver frequency

3.4 TRANSPORT / XHAUL

3.4.1 Transport Network Exposure API

The transport network exposure API is a Telco-specific API built on top of the Network Slice Controller (Section 2.4). It aggregates metrics and information from the transport domain, slice-related information coming from the NSC, as well as transport topology information coming from the SDN controller handling the network, TeraflowSDN (see Section 2.4.1.2). This section provides a detailed description of the capabilities of this API as well as the different endpoints and its functionalities.

The API is based on RESTCONF, which is a RESTful network management protocol that provides a simple way to access and manipulate network device configuration and operational data. It uses standard HTTP methods (GET, POST, PUT, PATCH, DELETE) and encodes data modelled in YANG using JSON, making it easy to integrate with web-based tools and automation systems. RESTCONF is defined by the IETF to offer a lightweight alternative to NETCONF, focusing on simplicity and ease of use over HTTP [37].

The API is based on the IETF RFC 9543, which defines a YANG model that stores the Network Slice framework defined by IETF. Moreover, this API includes not only network slice-related information, but metrics and endpoints aggregated from the SDN controller handling the network.

A summary of the model is depicted in Figure 3-11. An IETF Network Slice Service is specified in terms of a set of SDPs, the associated connectivity constructs and the service objectives that the customer wishes to see fulfilled. These three key elements are expressed in this model:

- SLO-SLE templates: it expresses the set of Service Level Objectives (SLOs) and Service Level Expectations (SLEs) that apply to one or several network slices. The SLOs are described as Directly Measurable Objectives such as guaranteed minimum bandwidth, guaranteed maximum latency or maximum permissible packet loss ratio. On the other hand, the SLEs are not directly measurable and some examples of them are security, isolation and geographic restrictions.
- Service Demarcation Points (SDPs): the SDPs can be defined as the endpoints at which a Network Slice is delivered by a service provider to a customer.
- Connectivity construct: a set of SDPs together with a communication type (point to point, point to multipoint) that defines how traffic flows are between the SDPs.

```

module: ietf-network-slice-service
  +--rw network-slice-services
    +--rw slo-sle-templates
      | +--rw slo-sle-template* [id]
      |   ...
    +--rw slice-service* [id]
      +--rw id
      +--rw description?
      +--rw service-tags
      |   ...
      +--rw (slo-sle-policy)?
      |   ...
      +--rw test-only?
      +--rw status
      |   ...
      +--rw sdps
      |   ...
      +--rw connection-groups
      |   ...
      +--rw custom-topology
      |   ...

```

Figure 3-11: Network Slice YANG model

It is important to note that the slo-sle templates are stored in an independent object, while the rest of information is stored over the “slice-service” object, that includes other relevant information such as the id of slices or its description.

This key information is exposed through several endpoints which all share the same root: /restconf/data/ietf-network-slice-service:network-slice-services. All endpoints allow the GET, POST, PUT and DELETE operations. The following endpoints are exposed:

- `/restconf/data/ietf-network-slice-service:network-slice-services` à It operates over the whole model, being able to create, modify, retrieve and delete several slices at the same time, as well as slo-sle templates
- `/restconf/data/ietf-network-slice-service:network-slice-services/slo-sle-templates/slo-sle-template` à it allows creating, modifying, retrieving and deleting slo-sle templates
- `/restconf/data/ietf-network-slice-service:network-slice-services/slice-service` à it allows creating, modifying, retrieving and deleting the active slice services
- `/restconf/data/ietf-network-slice-service:network-slice-services/slice-service={slice_service_id}/sdps` à it allows creating, modifying, retrieving and deleting SDPs from a given slice

Moreover, this API allows subscribing to YANG notifications of the IETF Network Slice Model through the endpoint `/restconf/operations/ietf-subscribed-notifications:establish-subscription`. This endpoint allows customers to retrieve slice telemetry from the network and check that the slice requirements are being met. In this way, the SDN controller is continuously monitoring the network and when a client subscribe to slice telemetry, the NSC aggregates the end-to-end path telemetry data to provide notification on the compliance of slice SLOs and SLEs.

Finally, this API offers endpoints exposing network slice topology. This is done by exposing the IETF Network Model [38] which is a YANG data model to represent networks and topologies, augmented with slice-related parameters coming from the existing slice YANG model already presented. This work aligns with the ongoing in draft-ietf-teas-network-slice-topology-yang [39], which specifies a Network Slice Topology YANG data model that enables the representation of slice-specific intent topologies on top of the generic IETF network topology framework. The model allows a network slice customer to express resource reservation intentions, connectivity scope, and underlay path visibility within a provider network, extending the network slicing framework defined in RFC 9543. In particular, it introduces YANG augmentations to describe slice-specific nodes, links, and underlay paths, as well as their associated constraints (e.g., bandwidth, latency, and resource partitioning), enabling finer control and partial exposure of provider topology. This model and its endpoints will be further developed for the final report of UNITY-6G developments.

3.5 VERTICAL APP DOMAIN (NBC)

CAMARA is a global, open-source initiative hosted by the Linux Foundation [43], established in collaboration with the GSMA and leading telecommunications operators worldwide [44]. The project aims to define and publish a unified set of standardised, operator-agnostic APIs that expose mobile network capabilities to application developers, enabling innovation at the network edge without requiring deep telecom domain expertise.

The CAMARA initiative addresses a long-standing fragmentation problem in the telecoms industry: each operator historically exposed proprietary APIs with different data models, authentication mechanisms, and endpoint structures [44], [45]. CAMARA resolves this by providing a single, interoperable API catalogue that any operator can implement, and any

application provider can consume, regardless of the underlying network vendor or infrastructure.

The CAMARA project pursues the following objectives:

- **Standardised network APIs:** Define a common REST/OpenAPI-based API surface for exposing network capabilities such as Quality-on-Demand (QoD), device location, number verification, network slicing, and edge application management.
- **Vendor neutrality:** APIs are defined independently of the underlying network implementation, allowing operators to adopt them on top of diverse technology stacks (5G Core, MEC platforms, SDN controllers, etc.).
- **Developer accessibility:** Lower the barrier for application developers to leverage telecom capabilities, using familiar REST/JSON interfaces, OAuth-based authentication, and standard HTTP semantics.
- **Industry alignment:** CAMARA works in coordination with the GSMA Open Gateway initiative [44], 3GPP [45], ETSI MEC [46], and other standards bodies to ensure coherence across the ecosystem.

The project is governed through a set of Working Groups, each responsible for a specific API family. Contributions are made through open pull requests and design reviews, following a meritocratic, consensus-driven model. Nearby Computing (NBC), as a partner in the UNITY-6G project, actively participates in the CAMARA initiative, contributing in particular to the Edge Cloud sub-project and the Computing Services API family [47].

3.5.1 CAMARA API Families

CAMARA organises its APIs into functional domains. Table 3-11 summarises the major API families.

Table 3-11: CAMARA API Families

API Family	Description	Representative APIs
Authentication & Fraud Prevention	Identity verification and fraud detection capabilities	Number Verification, SIM Swap, OTP Validation
Location Services	Exposure of device location and geofencing capabilities	Location Retrieval, Location Verification, Geofencing
Communication Services	APIs related to messaging, voice, and communication control	WebRTC Call Handling, Call Forwarding
Communication Quality	APIs to manage and guarantee network performance	Quality on Demand (QoD), QoS Provisioning
Device Information	Access to device status and connectivity information	Device Reachability, Device Status, Device Identifier

Computing Services	Exposure of edge/cloud capabilities and MEC interaction	Edge Application Management (EAM), Edge Discovery, Traffic Influence (CAMARA Edge / Computing APIs CAMARA Project, Edge Cloud APIs (Computing Services Sub-Project), 2026.[Online].Available: https://github.com/camaraproject/EdgeCloud)
Payments & Charging	Monetisation and billing capabilities via operators	Carrier Billing, Carrier Billing Refund
Service Management	APIs for capability exposure control and governance	Consent Management, Capability Restrictions

Within the UNITY-6G Network Exposure Framework, the **Edge Application Management (EAM) API** is the primary CAMARA contribution, as it directly addresses the challenge of deploying and managing vertical applications across distributed Telco Edge Cloud zones.

3.5.2 Role of CAMARA in the UNITY-6G Framework

In the context of the UNITY-6G Network Exposure Framework architecture, CAMARA and EAM API occupy the Vertical App Domain layer, exposing the northbound interface through which vertical application providers interact with the underlying network and edge infrastructure. Specifically, the EAM CAMARA API provides the standardized entry point for:

- **Application lifecycle management:** Vertical application providers register, deploy, monitor, and terminate their applications through CAMARA APIs, without knowledge of the operator's internal platform.
- **Infrastructure discovery:** Application providers query available edge sites, their geographic regions, operational status, and associated Kubernetes clusters through a unified discovery interface.
- **Multi-site orchestration:** Deployments can target multiple Edge Cloud Zones simultaneously, with the platform handling per-zone provisioning and reconciliation.

The CAMARA API layer abstracts the complexity of the underlying orchestration platform (in this case, the NearbyOne platform), presenting a clean, standardised interface to external consumers while delegating all state management and workload orchestration to the platform's internal services.

3.5.3 CAMARA Edge Application Management (EAM) API

The CAMARA Edge Application Management API provides a RESTful interface for managing the complete lifecycle of edge-native applications across distributed Telco Edge Cloud zones. It is defined as an OpenAPI 3.0.3 specification and covers four functional areas [48]:

- **Application Management:** Register application metadata (name, version, provider, packaging format, resource requirements, component specifications) and manage the catalogue of registered applications.

- **Instance Management:** Deploy application instances to specific Edge Cloud Zones, monitor their operational status, and terminate them when no longer needed.
- **Deployment Management:** Orchestrate multi-zone deployments that maintain application instances across multiple Edge Cloud Zones simultaneously, with support for dynamic zone addition and removal.
- **Edge Discovery:** Query available Edge Cloud Zones and their associated Kubernetes clusters, filtered by geographic region and operational status.

3.5.3.1 API Endpoints and Data Model

The EAM API defines 13 endpoints across the four functional areas. The complete endpoint inventory is presented in Table 3-12.

Table 3-12: CAMARA EAM API Endpoints

Endpoint	Method	Description	Response
/apps	POST	Register an application manifest with metadata, packaging and resource specifications	201 — App registered, returns appld
/apps	GET	List all registered applications for the current organisation	200 — Array of AppManifest
/apps/{appld}	GET	Retrieve a specific registered application by identifier	200 — Single AppManifest
/apps/{appld}	DELETE	Remove an application registration (409 if active instances exist)	204 — App removed
/appinstances	POST	Deploy an application instance to a specific Edge Cloud Zone	202 — Instance creation initiated
/appinstances	GET	List application instances, optionally filtered by app or region	200 — Array of AppInstanceInfo
/appinstances/{id}	DELETE	Terminate a running application instance	204 — Instance terminated
/deployments	POST	Create a multi-zone deployment targeting one or more zones	202 — Deployment created
/deployments	GET	List deployments for a given application	200 — Array of AppDeploymentInfo
/deployments/{id}	DELETE	Delete a deployment, cascade-terminating all instances	202 — Deletion initiated
/deployments/{id}	PATCH	Update a deployment's target zones (add/remove)	200 — Updated deployment
/edge-cloud-zones	GET	Discover available Edge Cloud Zones by region/status	200 — Array of EdgeCloudZone
/clusters	GET	List Kubernetes clusters backing the zones	200 — Array of ClusterInfo

The core data model of the EAM API comprises the entities listed in Table 3-13.

Table 3-13: EAM API Core Data Model

Entity	Description	Key Fields
AppManifest	Application registration record with metadata and deployment specs	name, version, appProvider, packageType, appRepo, requiredResources, componentSpec
AppInstanceInfo	Runtime information about a deployed application instance	appInstanceId, appld, name, edgeCloudZoneId, status, accessEndpoint
AppDeploymentInfo	Multi-zone deployment record with references to instances	appDeploymentId, appld, name, edgeCloudZoneIds, appInstanceIds
EdgeCloudZone	An available MEC site with geographic and operational metadata	edgeCloudZoneId, name, edgeCloudProvider, edgeCloudRegion, status
ClusterInfo	A Kubernetes cluster associated with an Edge Cloud Zone	kubernetesClusterRef, edgeCloudZoneId, region
RequiredResources	Resource specification supporting multiple infrastructure kinds	infraKind (kubernetes/vm/container), CPU, memory, storage, GPU

3.5.3.2 Application Lifecycle

The EAM API supports a well-defined application lifecycle that proceeds through the following phases:

- 1. Registration:** The application provider submits an application manifest via POST /apps, which stores the metadata (name, version, packaging format, resource requirements) and returns an appld, which is a unique handle for all subsequent operations. No workloads are deployed at this stage.
- 2. Discovery:** The application provider queries available Edge Cloud Zones via GET /edge-cloud-zones and GET /clusters to identify suitable deployment targets based on geographic region, operational status, and cluster capabilities.
- 3. Instantiation:** Using the appld and a selected edgeCloudZoneId, the application provider deploys an instance via POST /appinstances. The platform provisions the workload on the target zone and returns an appInstanceId with an initial "instantiating" status. Alternatively, POST /deployments deploys instances to multiple zones in one operation.
- 4. Monitoring:** The application provider retrieves instance status via GET /appinstances, observing transitions through instantiating → ready → running. Access endpoints (IP, port, FQDN) become available once the instance is ready.
- 5. Update:** For multi-zone deployments, PATCH /deployments/{id} enables dynamic scaling, like adding or removing zones at runtime. The platform reconciles the desired state by creating new instances or terminating removed ones.
- 6. Termination:** The application provider deletes instances via DELETE /appinstances/{id} or entire deployments via DELETE /deployments/{id} (which cascades to all associated

instances). Application registrations are removed via DELETE /apps/{appId} (which fails if active instances remain).

3.5.4 Integration with the NearbyOne Platform

The implementation of the EAM CAMARA API in UNITY-6G is via integration with the NearbyOne orchestration platform (presented in Deliverable D2.2 [75]). Concretely, the CAMARA EAM API is implemented as a translation layer between the standardised CAMARA REST interface and the NearbyOne platform's internal services [49]. The EAM API service itself is stateless, which means that it holds no application or instance state. All state is stored and managed within the NearbyOne platform as declarative resource bundles called NearbyBlocks.

The architecture comprises the following layers (see Figure 3-12):

1. **CAMARA EAM API Service:** A Go-based REST API server implementing the CAMARA EAM specification (OpenAPI 3.0.3). It receives HTTP requests from application providers, validates them against the API schema, and translates them into NearbyOne platform operations.
2. **NearbyOne Core API Server:** The platform's internal API server, which the EAM service communicates with to manage application state, query infrastructure, and trigger deployments.
3. **NearbyOne Service Manager:** Manages the lifecycle of NearbyBlocks chains. When the EAM API requests a new application registration or instance deployment, the Service Manager instantiates the appropriate NearbyBlock chain with the provided configuration values.
4. **NearbyOne Orchestrator:** The NearbyOne declarative orchestration engine. It reconciles the desired state expressed in NearbyBlock resources (such as Connections and ChartDeployments) with the actual state of the target Kubernetes clusters, creating, updating, or removing workloads as needed.
5. **Edge Cloud Zones:** The target Kubernetes clusters where application workloads are deployed [50]. Each zone is registered as a Cloud Site in the NearbyOne platform and is accessible to the Orchestrator for workload provisioning.

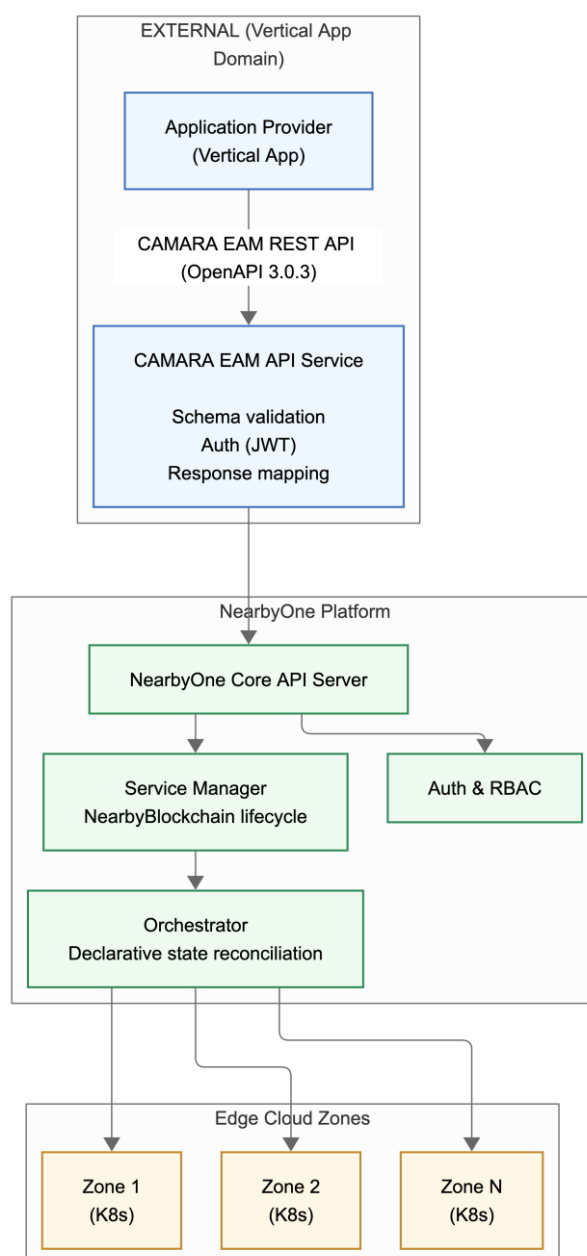


Figure 3-12. CAMARA EAM - NearbyOne Integration Architecture.

The implementation of CAMARA EAM API, and its integration with the NearbyOne orchestrator is work in progress. The final implementation will be reported in Deliverable D3.2.

3.6 API MANAGEMENT

The API Management layer in UNITY-6G serves as the orchestration and governance plane for the project's exposure capabilities. As the network transitions toward 6G, the ability to securely expose network functions to third-party developers and vertical industries becomes a critical requirement. This section details the two-fold approach adopted by UNITY-6G to manage this exposure: the integration of a high-level, developer-centric catalog through the Nearby HUB and the implementation of the standardized 3GPP OpenCAPIF framework. The

objective of this dual-layer management strategy is to bridge the gap between complex telecommunications standards and the operational needs of application developers. While the Nearby HUB focuses on the lifecycle management, versioning, and monitoring of diverse API types (including TM Forum and CAMARA), OpenCAPIF ensures that these interactions remain strictly compliant with 3GPP security and discovery protocols. Together, these components provide a unified approach for API governance, ensuring that network assets, ranging from core monitoring to QoS control, are discoverable, secure, and consistently managed across the heterogeneous UNITY-6G infrastructure.

3.6.1 Nearby HUB module: API manager

NearbyOne (introduced in Deliverable D2.2 [75]) delivers a modular architecture, with transversal capabilities across different operational domains. Among them, the Nearby HUB module provides an API Catalog (see Figure 3-13) that enables users to onboard, manage, and monitor network APIs within the UNITY-6G network exposure ecosystem. The main capabilities of the Nearby HUB module are:

- Cataloging and versioning of REST, TM Forum, CAMARA, and GraphQL endpoints.
- Policy application (authentication, authorization, quota) and usage analytics.
- Developer-centric discovery and documentation surfaces.
- Event and telemetry integration for API performance and reliability insights.

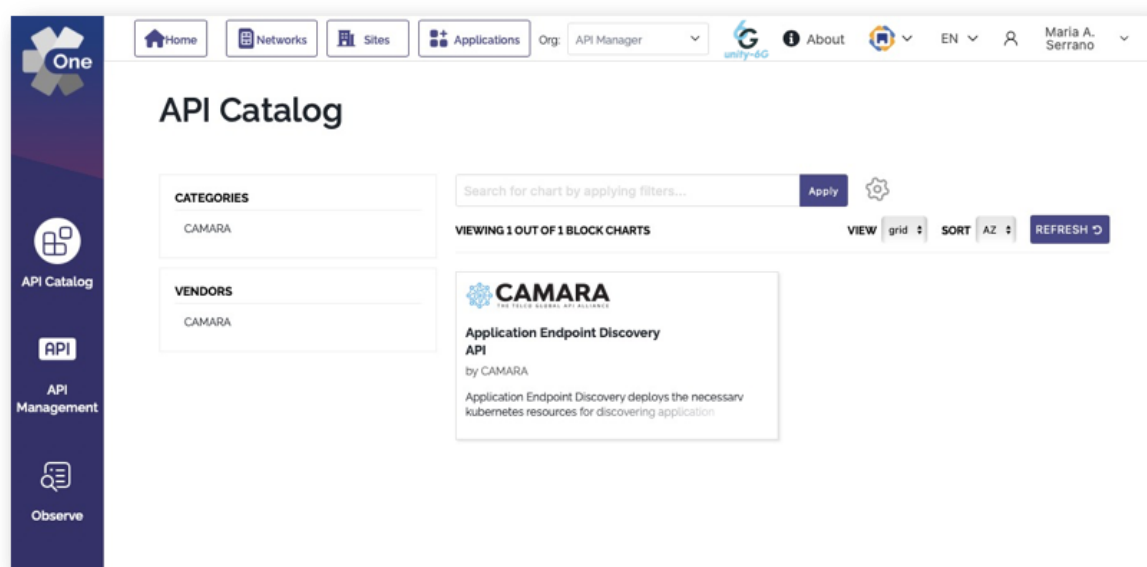


Figure 3-13. Nearby HUB API manager.

Concretely, the API Management Functions implemented in NearbyOne are the following:

API Onboarding and Publishing: Nearby Hub streamlines the process of onboarding new APIs in the UNITY-6G network exposure ecosystem and publishing them for internal or external consumption. It provides tools for registering, documenting, and versioning APIs, ensuring that services are discoverable and accessible to authorized users and systems. By centralizing API onboarding, UNITY-6G can maintain consistency, enforce standards, and accelerate the rollout of new integrations and services.

Security and Access Control: Security is a core function of Nearby Hub. The module enables administrators to define authentication and authorization policies for APIs, including support for API keys, OAuth, and role-based access control. This ensures that only trusted users and applications can access sensitive data and services. Nearby Hub also provides mechanisms for rate limiting, threat detection, and audit logging, helping UNITY-6G network exposure framework to comply with security and regulatory requirements.

Monitoring and Analytics: To ensure high performance and reliability, Nearby Hub continuously monitors API usage and health. It collects metrics such as request rates, error rates, and latency, providing real-time dashboards and historical analytics. These insights help API providers to identify bottlenecks, troubleshoot issues, and optimize API performance, ensuring a seamless experience for consumers and partners.

Developer Portal: To access API documentation, usage examples, and interactive testing tools.

The Nearby Hub module in NearbyOne will be integrated in the UNITY-6G Network Exposure framework to deliver end-to-end API management capabilities, from onboarding and securing APIs to monitoring usage and providing developer resources. As an example, the Edge Application Management CAMARA API will be published in the Nearby HUB module to expose its services to external partners or applications.

3.6.2 OpenCAPIF

The UNITY-6G project utilizes the OpenCAPIF (Common API Framework) to provide a standardized management plane for the secure exposure of telecom APIs. By integrating the OpenCAPIF SDK [51], the project implements a robust authorization and discovery layer that sits between API Providers (such as the NEF) and API Invokers (such as xApps). This implementation ensures that network capabilities whether originating from the core or direct monitoring tools are governed by a unified security framework that handles onboarding, authentication, and service discovery in a zero-trust environment [52]. The system is built on a modular FastAPI architecture where the API Provider publishes its service characteristics using standard OpenAPI YAML schemas. To ensure the integrity of security credentials across distributed environments, the framework incorporates a dedicated SFTP server for the secure transfer and persistent storage of CAPIF-issued certificates. This setup abstracts the complexity of 3GPP security protocols, allowing for automated deployment and granular control over the API lifecycle within the UNITY-6G ecosystem. The integration follows a structured sequence of actions to ensure authorized communication between the network functions and external applications as Figure 3-14 depicts:

1. **User Registration and Onboarding:** A user profile is created within the CAPIF core using a registration script. This user can then be onboarded as a Provider, an Invoker, or both. During this phase, the Provider receives a unique server certificate (capif_cert_server.pem), which is stored securely via the SFTP server for persistent identity verification.
2. **Service Publication:** The Provider publishes its available APIs (e.g., the NEF MonitoringEvent service) to the CAPIF core. This action includes uploading the OpenAPI

specification, which makes the service discoverable to other authorized entities in the network.

3. **Service Discovery and Token Acquisition:** The Invoker application performs a discovery request to identify available services that match its requirements. Once the desired service is located, the Invoker retrieves a JWT (JSON Web Token) from the CAPIF core. This token acts as the primary authorization credential for subsequent API calls.
4. **Authorized Service Consumption:** The Invoker sends a request to the Provider's API endpoint, including the JWT in the header. The Provider verifies the token against its CAPIF-issued certificate. Upon successful validation, the Provider processes the request and streams the requested network data back to the Invoker, which then provides the information to the end-user application (xApp).
5. **Lifecycle Management:** The framework supports robust offboarding and cleanup procedures. When a service or participant is no longer required, the offboarding script removes the entity from the CAPIF core and revokes associated certificates, ensuring a clean and secure network state.

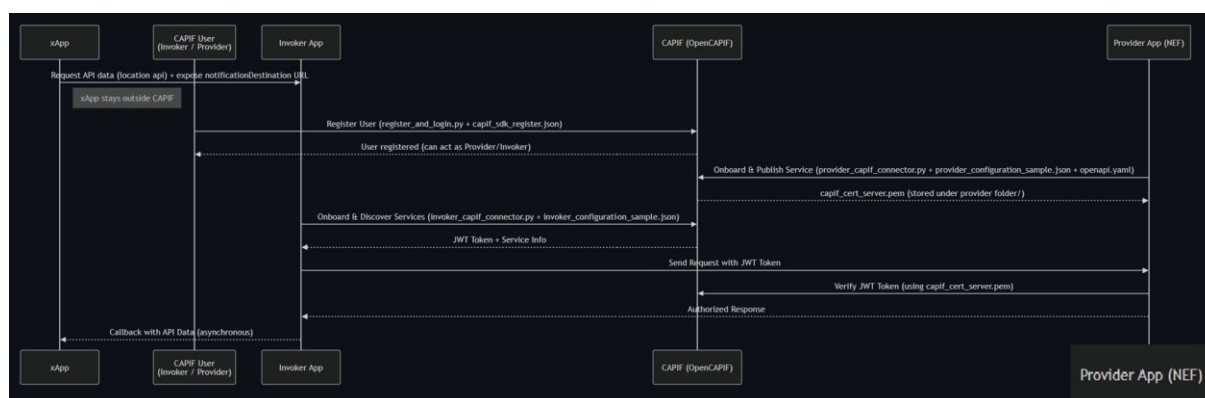


Figure 3-14: OpenCAPIF Sequence Diagramm

3.7 NETWORK EXPOSURE CONSUMERS

The consumers of the UNITY-6G Network Exposure framework represent the entities that interact with it to access monitoring information, analytics, and programmable network capabilities. These consumers can belong to different operational and application domains and constitute the primary users of the exposed APIs and services. The framework is designed to transform heterogeneous network functionalities and operational data into programmable services that can be securely consumed by internal automation entities as well as external applications and third-party service providers. In this way, the framework enables the transition from traditional closed telecom infrastructures toward an open, service-oriented, and API-driven 6G ecosystem.

For external stakeholders, the framework acts as a secure mediation layer that abstracts the complexity of telecom infrastructure and exposes network capabilities through interoperable APIs. Such stakeholders may include service providers, infrastructure operators, research platforms, public authorities, industrial partners, or third-party software developers interested in integrating advanced connectivity capabilities into their services. Through standardized interfaces based on technologies such as CAMARA, 3GPP NEF/CAPIF, and O-RAN, external

entities can access network information and request specific network behaviours without requiring detailed knowledge of the underlying infrastructure implementation.

The consumers of the exposed metrics from the different domains of the UNITY-6G architecture can also be intelligent functions designed to transform raw network data into actionable insights and automated decisions. These functions typically operate in an individual domain, ingesting data streams from RAN, core, Wi-Fi, transport, and non-terrestrial segments through standardized APIs, though can also operate across domains. In this context, these intelligent consumers incorporate AI-assisted algorithms (developed in WP4) for analyzing network metrics such as radio resource utilization (e.g., PRB usage), core signalling load, transport slice performance, and Wi-Fi channel conditions. The AI-assisted data analysis based on the network visibility enables these intelligent functions to detect anomalies, identify performance bottlenecks, and understand the relationship between network conditions and application-level requirements.

A key role of these intelligent consumers is to enable closed-loop automation. Leveraging techniques from ML and data analytics, they can infer patterns, forecast future states (e.g., traffic demand or congestion), and recommend or directly trigger control actions via the exposed control interfaces (such as O-RAN xApps, core network functions, or SDN controllers for the transport domain). For instance, an AI model may predict an increase in PRB utilization in specific cells and proactively adjust resource allocation, or detect degradation in a network slice and reconfigure routing policies. These feedback loops operate at different timescales, from near real-time optimization in the RAN to longer-term policy adjustments in the core and transport domains, ensuring adaptive and efficient network management.

4 DIGITAL TWIN

4.1 DIGITAL TWIN: ARCHITECTURE AND SPECIFICATION

This section proves we achieved Milestone “Define the DT application specifications”

4.1.1 Digital Twin in the Unity 6G architecture

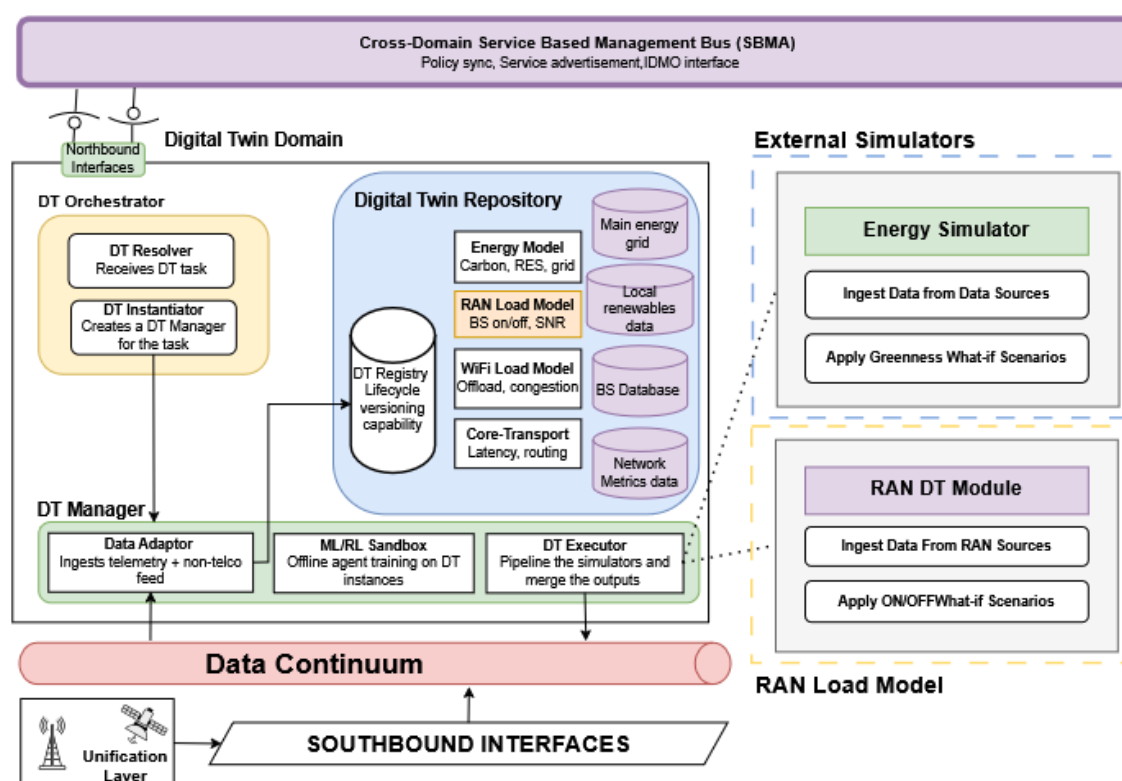


Figure 4-1: Overall digital architecture and corresponding components interactions. Northbound interfaces represent how the applications interact with the DT whereas Southbound interfaces represent where the twin data comes from

Figure 4-1 illustrates the internal organisation of the DT domain and its interactions with the UNITY-6G multi-domain architecture. At the top, the Cross-Domain Service-Based Management Architecture (SBMA) bus serves as the northbound communication plane. Through this bus, other domains forward "what-if" tasks and receive the DT's responses. A task may be predictive (e.g., what happens if I install 25 solar panels at specific locations?), to which the service replies with evaluation results, or prescriptive (e.g., which configuration maximizes greenness under a fixed panel budget?), to which the DT recommends an optimal configuration. In either case, enforcement authority remains with the requesting entity. To address the challenge of scaling the DT concept from domain-specific functions to whole-service coverage across heterogeneous environments (e.g., TN, NTN, and NPN), the architecture relies on dynamic composition. When a task spans multiple domains, the Inter-Domain Management and Orchestration (IDMO) entity requests the DT domain to create a comprehensive twin of the entire service chain.

The DT Orchestrator acts as the control entry point. Its DT Resolver receives incoming tasks from the SBMA, while the DT Instantiator creates a dedicated, query-isolated DT Manager instance. This ensures concurrent twin instances run without contention. The DT Manager serves as the central coordination unit. Its Data Adaptor ingests both standard network telemetry (e.g., traffic flows, user associations) and non-telco contextual feeds (e.g., real-time renewable energy availability, grid carbon intensity) via the data continuum. Crucially, the DT Executor fulfils the IDMO's inter-domain requests by generating a composite DT service. At query time, it pipelines external simulators (e.g., an Energy Simulator) with various DT Modules (e.g., a RAN DT Module). This approach enables multi-domain what-if reasoning without requiring a monolithic simulation model, consolidating the respective outputs into a unified, actionable result. Furthermore, the ML/RL Sandbox provides an isolated environment for offline training of reinforcement learning agents on these synthesised composite instances, decoupling the model development cycle from live network operations. To manage the deployment of these applications across specific service chains, the DT Repository acts as a global storage hub. It maintains versioned virtual models and ML applications spanning the energy, RAN, and transport domains, coordinated by a DT Registry that governs lifecycle and versioning. When a localized DT application (such as an end-to-end delay estimating xApp) must be transferred to a specific domain (e.g., an O-RAN RIC), the domain requests it from the global repository. The data manager then transfers the application over the Data Continuum, seamlessly mapping it to the target domain's interfaces. Finally, the Data Continuum and its Southbound Interfaces expose a unification layer that continuously synchronises the physical infrastructure's state into the DT Repository, sustaining the fidelity of the virtual models over time.

4.1.1.1 DT construction principles

A fundamental principle in designing Digital Twins for communication networks is that each DT must be created with a “clearly defined task”. The intended purpose determines the modelling approach, and the computational resources required. For example, a DT whose objective is to estimate the power consumption of the RAN or Core Network under a given configuration does not require a full-stack network emulator. Power consumption models for radio access nodes are well established and depend primarily on load, making a high-fidelity simulation unnecessary and inefficient for such a task. A different class of applications requires a much more detailed representation of network behaviour. For example, estimating packet-level transmission delay or end-to-end latency distributions may necessitate a Digital Twin capable of reproducing protocol interactions, queueing dynamics, scheduling behaviour, and transport network characteristics. In such cases, a full-stack network emulator or a high-fidelity simulator becomes essential. It is important to note, however, that although such complex DTs are theoretically capable of producing simple outputs, such as a single KPI, they are rarely the right tool for that purpose. Their computational cost, resource consumption, and typically longer execution delay make them unsuitable for tasks that could be handled by a lightweight analytical or machine-learning-based model. Using a heavy DT to answer a simple question is technically possible, but operationally unsustainable, especially in scenarios requiring real-time or near-real-time responses. These observations reinforce the conclusion that no single Digital Twin model can satisfy all possible requirements. Different tasks demand different levels of abstraction, accuracy, and computational efficiency. For this reason, the U6G architecture

adopts a repository of DT models-approach, each optimized for a specific task and selectable by the DT Manager according to the performance, latency, and fidelity constraints of the use case.

The literature reflects this diversity. The term Network Digital Twin encompasses a wide range of modelling approaches, from simple look-up tables, through analytical models and machine-learning-based predictors (including Graph Neural Networks), up to full network emulation platforms. This variety reinforces the need for a structured repository in which each model is stored together with a clear specification of its capabilities and limitations.

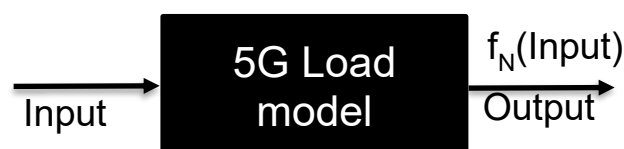


Figure 4-2: Example of a single input-single output Digital Twin

To specify such a Digital Twin, we should think of it as of a mathematical function with defined inputs and outputs, as shown in the example in Figure 4-2. While the input may include the entire network configuration and state, in many cases only a subset is necessary. For example, a power-consumption DT may require only the current load expressed as the percentage of occupied resource blocks. Similarly, the output may be a single KPI, a vector of performance indicators, or even a probabilistic output when the DT models uncertainty or environmental mismatch.

To support effective model selection, each DT module must be described by a set of operational properties, such as:

- computational complexity and resource consumption
- execution speed (e.g., number of outputs per second)
- update delay relative to the real network
- accuracy and error bounds
- required inputs and provided outputs

These characteristics allow the DT Manager to choose the most appropriate model for a given optimization or prediction task.

4.1.1.2 Composable Digital Twins

In heterogeneous networks, different elements of the end-to-end chain, such as base stations, transport segments, and edge computing platforms, may originate from different vendors and rely on different modelling assumptions. Nevertheless, many use cases require a Digital Twin that represents the behaviour of the entire chain. This necessitates the ability to compose multiple DT modules into a coherent system-level Digital Twin.

The complexity of composition varies. In some cases, it is straightforward: for example, estimating the total power consumption of a sequence of network elements can be achieved by simply adding the outputs of individual DT modules. However, in more complex scenarios the dependencies between modules are non-trivial. Consider a system composed of RAN,

Transport Network, and Edge Computing DTs. Although each module can be represented as an input-output function, the Transport Network DT may depend not only on the global input (such as traffic demand) but also on the output of the RAN DT, which determines traffic burstiness or packet arrival characteristics. The Edge Computing DT may depend on both previous outputs and additional contextual information such as compute load or service placement. Such a dependence is presented graphically in Figure 4-3. As a result, composition requires explicit modelling of inter-module dependencies and a clear definition of how information flows between them.

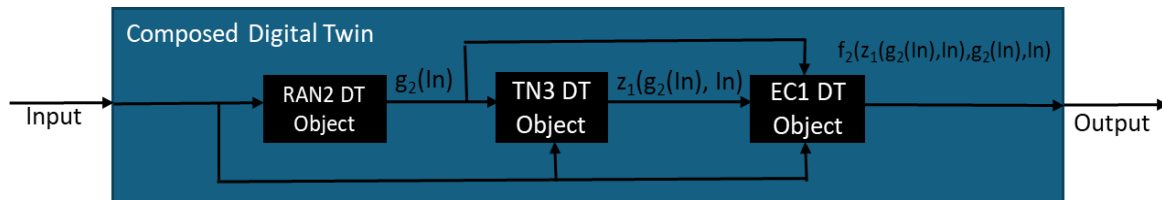


Figure 4-3: Example of Composable Digital Twin

When DT modules are implemented as simulators, composability also imposes significant requirements. All simulators must probably expose a common API, support compatible input and output formats, and operate on aligned time scales. For example, one simulator may operate on a per-packet basis while another uses per-frame or per-second time steps. Without harmonization, meaningful composition is impossible. Synchronization mechanisms are also necessary to ensure consistent state propagation across modules. Furthermore, the semantics of KPIs and events must be standardized to avoid ambiguity when combining outputs from different simulators.

These considerations highlight the importance but also difficulties of a composable Digital Twin architecture, in which each DT module is designed not only to perform its specific task but also to interoperate with other modules when required. The repository-based approach can support this by storing, alongside each model, metadata describing its composability constraints, required inputs, provided outputs, and assumptions about time scales and execution environments. This enables the DT Manager to construct end-to-end Digital Twins dynamically, selecting and connecting modules in a manner consistent with both the task requirements and the underlying modelling assumptions.

4.2 DT DESIGN: TOOLS, MODELS, DATASETS

This section proves and reports achievement of Milestone “Release initial dataset and definition of data generation models using generative AI for Digital Twin”

4.2.1 Generative AI data generation model

The effectiveness of AI-driven network optimization strongly depends on the availability of large-scale, diverse, and high-quality datasets representing realistic network conditions, user behaviors, traffic patterns, and radio environments. In Open RAN scenarios, these datasets typically include Key Performance Indicators (KPIs) collected from different network components and interfaces, such as the E2 interface, including metrics like Reference Signal

Received Power (RSRP), Reference Signal Received Quality (RSRQ), Signal-to-Interference-plus-Noise Ratio (SINR), downlink/uplink throughput, and Physical Resource Block (PRB) utilization. These KPIs provide critical information about radio conditions, network load, spectrum efficiency, and Quality of Service (QoS), and are fundamental for training AI/ML models aimed at network optimization, anomaly detection, traffic prediction, and resource management.

In this context, Generative Artificial Intelligence (GenAI) models have emerged as a promising solution for synthetic data generation and data augmentation in wireless communications research. Techniques such as Generative Adversarial Networks (GANs) and Diffusion Models can learn the statistical properties and latent representations of complex datasets, enabling the generation of realistic synthetic samples that preserve the characteristics of original network data while expanding dataset diversity.

The use of generative models in 6G Open RAN environments is twofold. Firstly, synthetic data generation can mitigate data scarcity issues by augmenting limited datasets and improving the robustness and generalization capabilities of AI/ML models. Secondly, GenAI techniques can support the creation of balanced datasets, particularly for underrepresented network conditions or rare failure scenarios that are difficult to observe in operational systems.

4.2.1.1 Generative Adversarial Networks (GANs)

A possible framework for synthetic data generation could be Transformer-based GANs. These models feature a Transformer Encoder for both the generator and the discriminator. The principle on which GANs learn the data distribution is an adversarial game in which the generator objective is to generate new data samples able to fool the discriminator. Therefore, the objective of the latter instead is to correctly classify true (that is, coming from the true data distribution) samples as real and generated samples as fake. GANs offer relatively good representation power with fast inference speed, but they could not scale well when data presents a high number of features. Moreover, training GANs can be tricky due to the fact that the state the generator has to reach in order to be capable of generating new data indistinguishable from the real data is what is known as a Nash equilibrium, which can be difficult to achieve.

Already existing architectures are:

- **Transformer-based Time-Series Generative Adversarial Network (TTS-GAN):** a GAN for learning unimodal data
- **Transformer-based Time-Series Conditional Generative Adversarial Network (TTS-CGAN):** a conditional GAN, evolution of TTS-GAN, with the generator conditioned on categorical input
- **Multivariate Time Series Conditional Generative Adversarial Network (MTS-CGAN):** a conditional GAN, with the generation phase conditioned by both a categorical input and other time series
- **Long-Term Time Series Generative Adversarial Network (LTTS-GAN):** a GAN for long-term time series generation, with the generator conditioned on time series

Subsequently, a detailed description of the GAN architectures listed above.

4.2.1.1.1 TTS-GAN and TTS-CGAN

The only difference between TTS-GAN and TTS-CGAN is that the latter allows to condition the Generator output on a categorical variable, so that the Generator can produce time series belonging to different classes. Therefore, for practical matters, only the conditional variant is explained.

The generator takes as input a 1D Gaussian noise vector and a (embedded) label concatenated together. This is passed to a series of N Transformer Encoder blocks that yield synthetic outputs. The discriminator takes as input either real or synthetic signals. These are divided into patches and to each of them is added a positional encoding value before being passed to a series of N Transformer Encoder blocks. The discriminator features an adversarial classification head (is the signal being analyzed coming from the true distribution or from the generator?) and a categorical classification head (to which class does this signal belong to?). [67]. The building blocks of TTS-CGAN are shown in Figure 4-4.

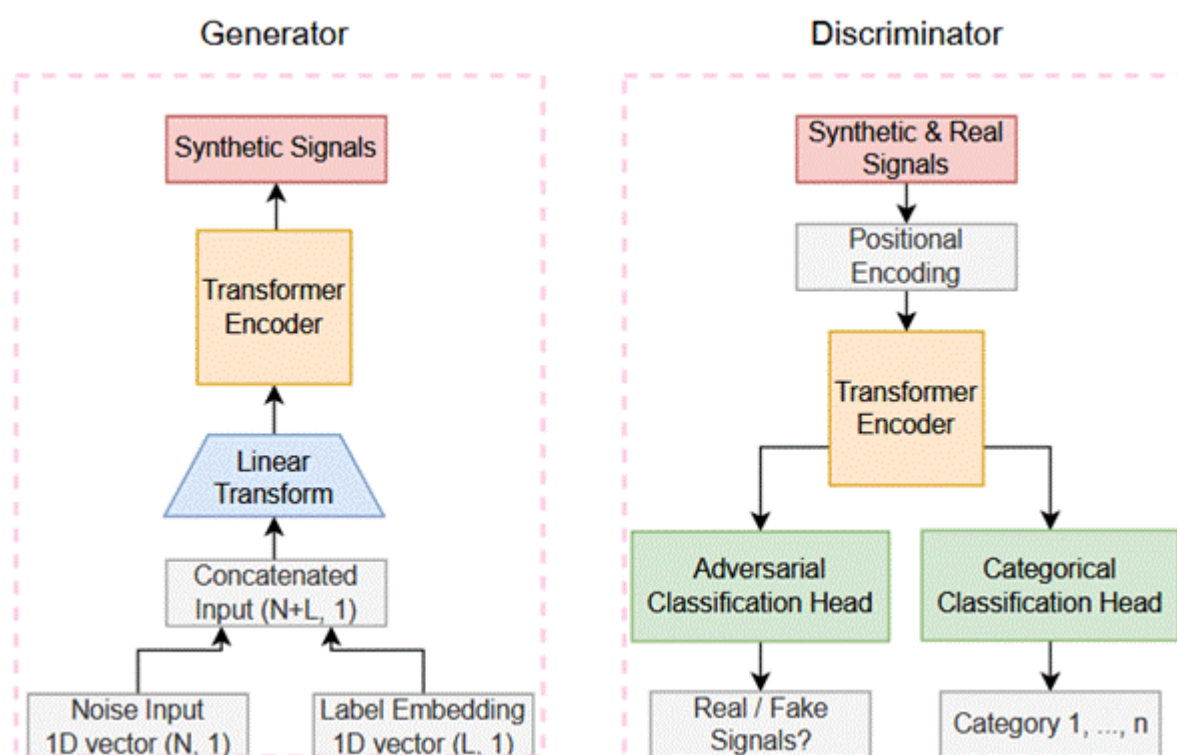


Figure 4-4: TTS-CGAN architecture.

4.2.1.1.2 MTS-CGAN

The architecture shown in Figure 4-5 could be considered to be the evolution of TTS-GAN and TTS-CGAN. In fact, the core of the generator and discriminator is again a series of N Transformer Encoder blocks. However, the generator presents a major enhancement: the generation phase is conditioned on other multi-variate time series, besides the categorical information. The discriminator, on the other hand, is equipped only with an adversarial classification head to tell if the signals it's analyzing are real or fake [68].

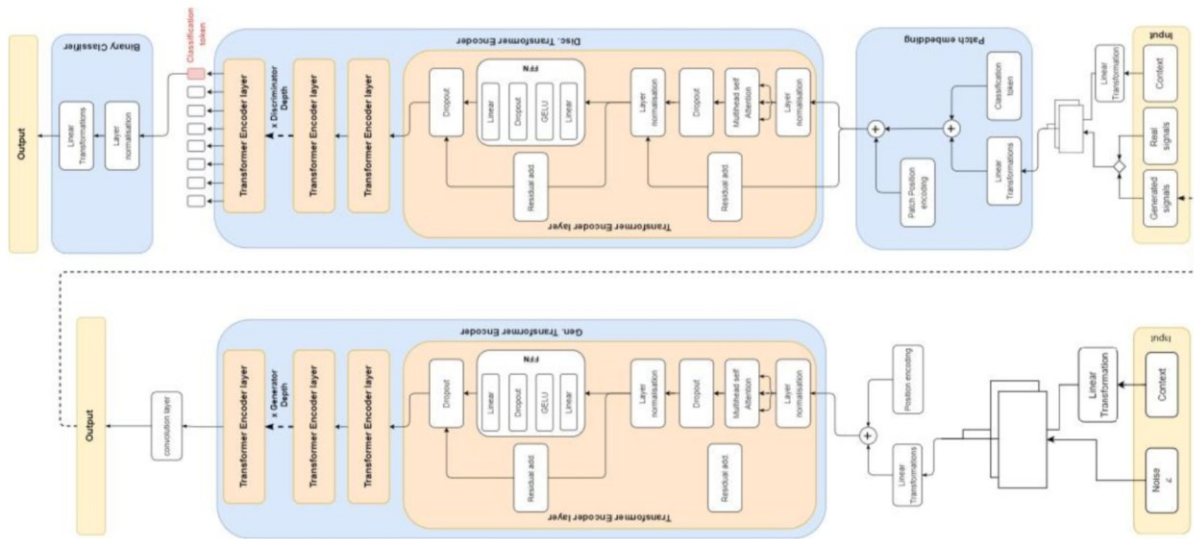


Figure 4-5: MTS-CGAN architecture.

4.2.1.1.3 LTTS-GAN

It is an architecture specifically engineered to address the limitations of RNNs and Transformers-based GANs in generating long-term time series [69]. The main innovations are on the generator side. The discriminator is the same as other GANs previously discussed. The generator architecture, shown in Figure 4-6, consists of:

- A global-local attention mechanism that balances long-range periodic dependencies with fine-grained local temporal dynamics. The global branch utilizes an autocorrelation to capture long-term cyclical trends, while the local branch consists of two 1D Convolutional layers followed by Batch Normalization to retain short-term, neighboring temporal dynamics. The two outputs are combined with a convolution operation to form the output of the whole block.
- A progressive series decomposition block that extracts trend-cyclical components through moving average operations across M channels. The seasonal components are obtained by subtracting the calculated trend-cyclical components from the time series. To obtain the final seasonal output, the M seasonal components go through a channel fusion block.
- A Feed-Forward Network (FFN) for projection and information propagation, which is the same FFN present in a classical Transformer Encoder model.

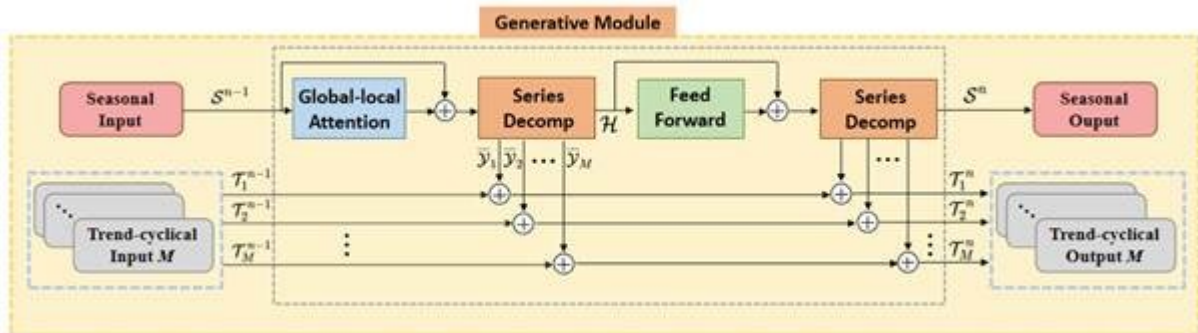


Figure 4-6: Architecture of the generator in LTTS-GAN.

4.2.1.2 Diffusion Models

Today, the state of the art regarding generative models is diffusion models. Initially developed for image generation, they have been adapted for time series generation. The generation process of a diffusion model starts with the *forward diffusion process*: Gaussian noise is progressively added at each time step until the sample resembles a sample drawn from a pure noise distribution. In the *reverse diffusion process* the model tries to reconstruct the original sample by removing the noise step by step. This effectively eliminates the instability problem in GANs training and renders the training quite stable. Additionally, the representation power of diffusion models has been proved to be superior to older generative architecture. However, a major drawback is that the generation process takes a much larger time than the inference time of GANs. Some examples of diffusion models are described below.

4.2.1.2.1 TimeDART

TimeDART is a self-supervised pre-training framework that unifies autoregressive modelling and denoising diffusion processes to capture both long-term global dynamics and local patterns [70]. The architecture consists of:

- **A causal transformer encoder** that takes as input the clean patch embeddings results from the input time series
- **A patch-level diffusion process** using a cosine noise scheduler
- **A masked denoising decoder** acting as the denoising reverse process, where noisy time series patches serve as queries while the encoded clean patch embedding serve as keys and values

4.2.1.2.2 Text-to-Series (T2S)

T2S is a diffusion-based framework for generating time series from natural-language descriptions [71]. The architecture is composed of three main components:

- **A T2S Diffusion Transformer (T2S-DiT)**, employing flow matching for the diffusion process and a “patchified” version of the original DiT module as the denoiser
- **A pretrained Length-Adaptive Variational Autoencoder (LA-VAE)** that compresses variable-length time series into a unified latent space before sending them to the forward diffusion process of the T2S-DiT and expands them back to their original length after the reverse denoising process
- **A text embedder** that embeds captions of the input time series into a context vector, which is injected into the denoising process to condition the generation

4.2.2 Multi-Layer Perceptron (MLP)-based architectures for generating “what-if” scenarios

4.2.2.1 TimePrism

TimePrism represents a fundamental shift away from Monte Carlo sampling paradigms (like diffusion or flow-based models) toward a deterministic, discrete probabilistic forecasting framework. Designed intentionally as a minimalist proof-of-concept, it demonstrates that

directly outputting explicit scenario and probability pairs can outperform complex sampling-based architectures in both accuracy and computational efficiency [72].

The model takes as input a multivariate time series and decomposes into trend and season components. These are passed through two different fully connected networks (FCN) layers that project the trend and season history into M and K future trajectories, respectively. Taking into account all possible combinations, the two FCNs output $N = M \cdot K$ possible future scenarios. Then, to assign a probability to each future scenario, a third FCN takes as input the original historical time series and maps it to a logits vector of size N . At inference time, a softmax function is applied after the logits layer to obtain a valid probability distribution so that each of them represents the probability that a certain scenario occurs. The overall architecture is shown in Figure 4-7.

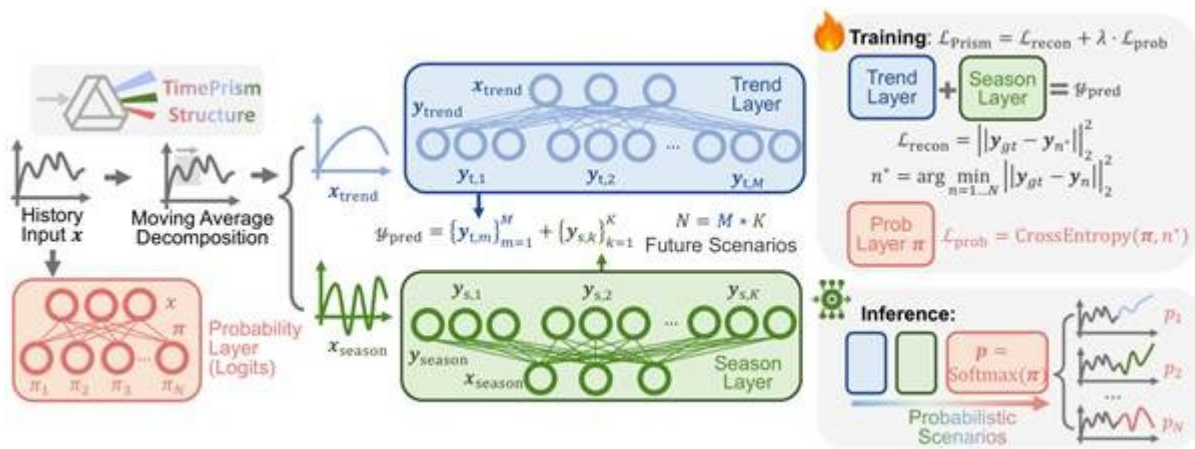


Figure 4-7: TimePrism Architecture.

4.2.2.2 World Model Multi-Scale Structured State-Space Model (WM-MS3M)

It is a strictly causal, action-conditioned state-space paradigm designed for near-real-time O-RAN control. It operates by learning the physical dynamics of network KPIs (the "world") and using an outer-loop planner to select optimal control actions. It consists of two main blocks: a predictive engine and a decision engine [73].

The world model (the predictive engine) takes as input a lookback window of several KPIs and the related actions taken in the network. The architecture is as follows:

- **MS3M backbone:** utilizes depth wise 1D causal convolutions derived from continuous-time HiPPO–LegS dynamics via bilinear (Tustin) discretization to capture temporal dependencies at multiple scales without future data leakage.
- **Gating & Mixing:** employs squeeze-and-excitation gating to dynamically modulate channel importance based on the temporal context, followed by a Gated Linear Unit (GLU) mixer for non-linear feature interactions and a residual LayerNorm connection.
- **Compact Stochastic Latent:** a diagonal-Gaussian latent variable z is inferred to capture unobserved propagation and interference conditions (modelling both aleatoric and epistemic uncertainty). During deployment, it relies strictly on a prior network; during training, it can access a posterior network.

- **Dual Decoders:** a full-frame MSE decoder reconstructs the entire next-step KPI vector for robust representation learning, while a heteroscedastic target decoder predicts both the mean and data-dependent variance for specific goal KPIs, augmented with a bounded autoregressive (AR) skip connection for short-term fidelity.

A model predictive control-based cross entropy method (MPC/CEM) uses the world model to roll out possible KPIs evolutions (the what-if forecasts) given candidate action sequences and chooses the next step associated with the KPI sequence presenting the best cumulated reward.

Figure 4-8 shows how the WM-MS3M and the MPC/CEM Planner interact between themselves in the near-RT RIC and with the external network environment.

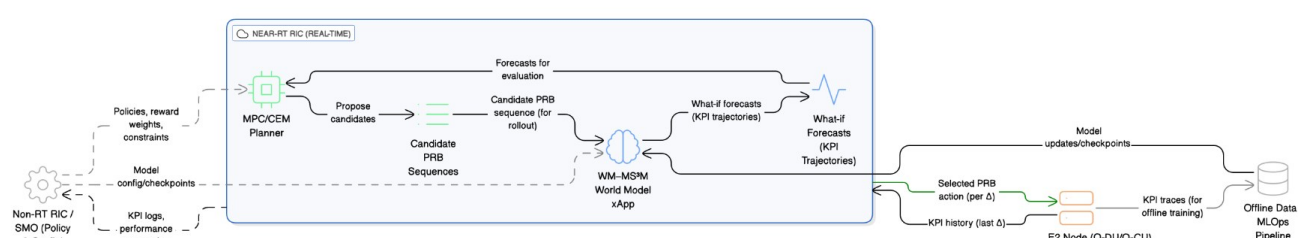


Figure 4-8: WM-MS3M and MPC/CEM Planner system model.

4.2.3 Network Emulation using NS3 simulator

NS-3 will be used in UNITY-6G as a network simulation and emulation building block within the Digital Twin toolset as shown in Figure 4-9. Its role is to provide an executable environment in which selected behaviours of the network can be reproduced, modified, and evaluated under controlled conditions. NS-3 is a discrete-event network simulator designed to model communication systems at packet, protocol, flow, and event level. It supports the representation of network nodes, links, protocol stacks, routing behaviour, traffic-generating applications, wireless channels, mobility, failure events, and performance measurements. This makes it suitable for Digital Twin tasks in which the expected behaviour of a network scenario cannot be derived from topology information alone, but must be evaluated through the interaction between traffic, configuration, protocols, and time-dependent events.

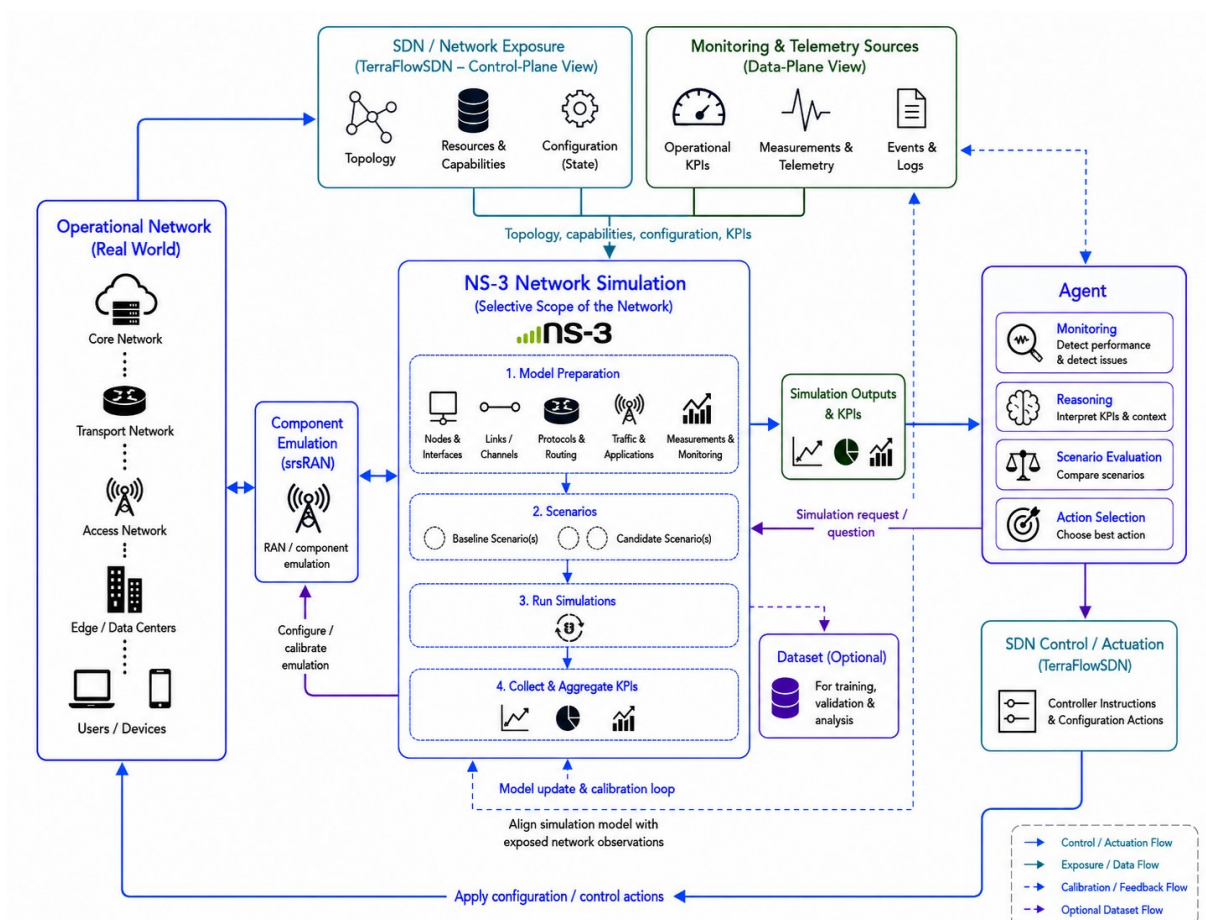


Figure 4-9: NS-3 as a scenario generator in the digital twin

In the context of UNITY-6G, NS-3 is not intended to replicate the full operational network in all its implementation details. A full copy of the deployed system would be impractical, computationally expensive, and unnecessary for most Digital Twin investigations. Instead, NS-3 will be used according to a selective twinning approach. For each investigation, only the part of the real or planned network that is relevant to the technical question is translated into a simulation model. The selected scope may represent a network segment, a service path, a set of traffic flows, a routing condition, a failure case, or a specific access-network scenario. The resulting NS-3 model is therefore a scoped executable representation of the network state, designed to answer a specific question rather than to mirror the entire system continuously.

NS-3 is considered in UNITY-6G as a configurable simulation building block rather than as a complete network replica. Its selection is motivated by its ability to represent network behaviour at a level where packet dynamics, protocol operation, routing, traffic demand, link conditions, and measurements can be evaluated together. Depending on the investigation, the NS-3 model may represent core IP networking, access-network behaviour, wireless propagation, mobility, traffic generation, routing logic, failure and recovery events, or measurement collection. The exact model composition is determined by the purpose of the Digital Twin task and by the level of fidelity required to produce a useful answer.

The main value of NS-3 in this context is its ability to evaluate dynamic network behaviour over time. Static inventory and topology data can indicate which elements exist and how they are connected, but they do not necessarily reveal how the network will behave under changing

load, failure, rerouting, mobility, or altered configuration. NS-3 enables these behaviours to be examined through repeatable simulation runs. It can be used to evaluate a baseline state, apply a controlled modification, and compare the resulting behaviour with the original scenario. The output of such evaluations provides engineering evidence that can support Digital Twin analysis and decision validation.

The main requirement for using NS-3 in the UNITY-6G Digital Twin is the availability of a simulation-ready scenario description. This description must define the relevant network scope, the entities to be represented, the connectivity between them, the link or channel properties, the traffic assumptions, the control assumptions, and the KPIs to be collected. The required level of detail depends on the scenario. A routing-focused scenario requires an accurate graph and forwarding assumptions. A latency-focused scenario requires appropriate traffic, delay, and queueing assumptions. A wireless scenario requires relevant radio and mobility assumptions. A failure or resilience scenario requires a clear definition of the event being tested and the expected restoration logic. Therefore, the NS-3 model is built only after the intended inputs, outputs, and evaluation objective are defined.

The twinning process for NS-3 begins by defining the purpose and scope of the simulation. The Digital Twin task determines which network elements and behaviours must be represented. Physical or logical network elements are mapped into NS-3 nodes. Connectivity relationships are mapped into NS-3 links or channels. Interface and link capabilities are mapped into simulation parameters. Service demand is mapped into traffic-generating applications or flow definitions. Operational changes are mapped into scheduled simulation events. In this way, selected operational network information is converted into a simulation-ready model using the primitives available in NS-3.

In terms of what is modelled, the NS-3 component represents only the aspects of the network that materially affect the required Digital Twin output. Network elements are represented as simulated entities, connectivity is represented as simulated relations, services are represented as traffic demand, and operational changes are represented as events in the simulation timeline. When the investigation concerns radio behaviour, NS-3 can include channel and mobility modelling. When the investigation concerns network behaviour, it can include routing, traffic, queueing, and performance measurement. When the investigation requires interaction with external systems or live traffic, NS-3 emulation capabilities may be considered, provided that the required interfaces and timing assumptions are compatible with the test setup.

Where SDN-controlled network information is available, TerraFlowSDN can support the creation and update of NS-3 scenarios by exposing a structured representation of the managed network. Its contribution in this context is to provide an abstraction layer between heterogeneous operational resources and the simulation environment. Instead of requiring the NS-3 model-generation process to interact directly with individual devices or vendor-specific management systems, TerraFlowSDN can expose a normalized view of the controlled domain through programmable interfaces. This view can then be translated into NS-3 entities, relations, parameters, and events, reducing manual modelling effort and improving alignment between the simulator and the operational network.

This interaction is important because the quality of the simulation depends on the quality and relevance of the model inputs. Topology information can be mapped into simulated nodes and

links. Capability information can be mapped into the main performance parameters of the simulated resources. Connectivity or service information can help define the simulated paths and traffic relations. Telemetry or KPI feedback can be used to calibrate the assumptions used in the model. In this manner, NS-3 scenarios can be grounded in the current or recent state of the network, rather than being based only on static or manually created assumptions.

The outputs of the NS-3 component are selected according to the evaluated scenario. The simulator may produce flow-level statistics, packet-level traces, time-resolved logs, and aggregated KPIs. These outputs can be used to compare a baseline scenario with one or more modified scenarios. In a predictive usage mode, NS-3 can estimate the expected behaviour of a defined scenario. In a comparative usage mode, it can evaluate several candidate scenarios under the same assumptions. In both cases, the simulation output is not treated as an exact prediction of the future, but as structured decision-support evidence derived from a controlled and reproducible model.

In addition to direct what-if evaluation, NS-3 can also support dataset generation for Digital Twin and AI/ML development. Since the simulator can execute controlled scenarios repeatedly, it can generate labelled or semi-labelled data under conditions that are difficult, expensive, or risky to reproduce in the operational network. This is especially relevant for rare events, failure cases, degradation patterns, recovery processes, and controlled variations of traffic or topology. The generated datasets may be used for offline ML training, reinforcement learning, anomaly detection, forecasting, model validation, or benchmarking of decision algorithms. In this role, NS-3 acts not only as an evaluation tool, but also as a controlled data-generation environment for the broader Digital Twin toolchain.

For dataset generation, each simulation run should be accompanied by metadata that describes the scenario assumptions, input parameters, random seeds, topology version, traffic profile, control action, and produced KPIs. This is necessary to ensure that generated datasets are reproducible and interpretable. The dataset should not only contain final aggregate results, but also the intermediate states required by the intended AI/ML task. For example, a learning algorithm may require sequences of network states and corresponding performance outcomes, while a validation process may require paired baseline and modified scenario results. The exact dataset structure should therefore be defined according to the target use case and the model that will consume it.

A central requirement for the NS-3 component is parameter update and calibration. Since the operational network changes over time, the simulator should not rely only on fixed assumptions. Parameters related to topology, connectivity, resource capability, traffic demand, routing state, and observed performance may need to be refreshed when relevant network feedback becomes available. Slowly changing parameters can be updated when configuration changes are detected. Operational parameters can be updated before a simulation is executed or when new telemetry becomes available. Highly dynamic conditions may be represented through recent snapshots, averaged measurements, or statistical assumptions, depending on the fidelity required by the scenario.

The feedback-based update process follows a structured sequence. First, relevant operational evidence is collected from monitoring, telemetry, controller, or exposure sources. Second, the collected information is normalized into a scenario description suitable for simulation. Third,

the scenario description is mapped into NS-3 configuration parameters. Fourth, the simulation is executed and the resulting KPIs are compared with available real-network measurements when validation data exist. If the simulated behaviour deviates significantly from observed behaviour, the calibration assumptions are adjusted. This process does not aim to create a perfect replica of the real network; it aims to maintain sufficient fidelity for the specific Digital Twin question being evaluated

The NS-3 component also has clear modelling boundaries. It does not reproduce proprietary device internals, undocumented hardware behaviour, full vendor-specific management logic, or every protocol and implementation detail of the operational system. These aspects are either abstracted, approximated, represented by external models, or excluded from the simulation scope unless they are essential for the investigated behaviour. Furthermore, higher fidelity increases computational cost. A detailed simulation may be valuable for specific analyses, but it may not be appropriate for every Digital Twin query. Conversely, a simplified model may execute quickly but may hide effects that are important for a specific question. Therefore, the model abstraction must be selected according to the purpose of the scenario.

For UNITY-6G, the planned use of NS-3 is therefore focused on controlled, scenario-based network evaluation and dataset generation. The simulator will support the transformation of selected operational network information into executable models, the execution of baseline and modified scenarios, and the generation of measurable outputs that can assist engineering analysis, Digital Twin validation, and AI/ML development. Its effectiveness will depend on the correctness of the input data, the mapping between real-network information and NS-3 parameters, the calibration process used to align the simulation with network feedback, and the suitability of the chosen abstraction level. Used in this manner, NS-3 provides a practical and extensible Digital Twin tool for evaluating selected network behaviours and producing reusable simulation data before conclusions or decisions are transferred back to the operational environment.

4.2.4 Network topology model

4.2.4.1 Introduction

This contribution presents the IETF Network Topology data model defined in RFC 8345 [38] as the reference model for representing network topologies within the UNITY-6G Digital Twin. The choice of RFC 8345 is motivated by its high level of abstraction, flexibility, and extensibility, which make it suitable for describing heterogeneous networks spanning multiple technological domains and layers. This model is supported by the TeraFlowSDN controller, used in UNITY-6G, which further reinforces its suitability and facilitates its practical adoption within the project architecture.

RFC 8345 defines a generic framework to represent networks, topologies, and their relationships independently of specific technologies. This abstraction capability is particularly relevant in the context of digital twins, where complex real-world infrastructures must be modelled, analysed, and evolved consistently over time. In UNITY-6G, the model enables a unified representation of network resources, services, and their dependencies, facilitating cross-domain analysis and what-if simulations.

This section first introduces the IETF Network Topology model and its main concepts. It then describes how the model can be applied and extended to support the UNITY-6G Digital Twin.

4.2.4.2 IETF Network Topology Model

RFC 8345, “A YANG Data Model for Network Topologies”, defines an abstract and technology-agnostic data model for the representation of networks and topologies using YANG. The model is structured in two main parts:

- The base network model (ietf-network)
- The topology augmentation (ietf-network-topology)

Together, these modules provide a generic framework to describe network inventories (record of all hardware, software, and configuration assets connected to a network), hierarchies, and topological relationships.

4.2.4.2.1 Base Network Model

The base network module enables the representation of networks as high-level abstractions and supports the concept of network hierarchies. Its main characteristics are:

- Definition of networks and network identifiers
- Representation of nodes contained within a network
- Support for supporting networks, allowing a network to depend on or be built upon other networks
- Maintenance of a basic inventory of network nodes

This structure allows the modelling of network stacks, where higher-level networks (e.g., service networks) can reference lower-level networks (e.g., transport or physical networks). As a result, the ietf-network module can serve as a common foundation for multiple topology models, including Layer 1, Layer 2, Layer 3, service, and inventory models.

```

module: ietf-network
  +--rw networks
    +--rw network* [network-id]
      +--rw network-id      network-id
      +--rw network-types
      +--rw supporting-network* [network-ref]
      | +--rw network-ref  -> /networks/network/network-id
      +--rw node* [node-id]
        +--rw node-id      node-id
        +--rw supporting-node* [network-ref node-ref]
        +--rw network-ref
        | -> ../../supporting-network/network-ref
        +--rw node-ref    -> /networks/network/node/node-id
    
```

Figure 4-10: Base Network Model

4.2.4.2.2 Topology Model

The ietf-network-topology module augments the base model to describe explicit topology information. It introduces graph-based constructs to represent how network elements are interconnected. Key elements include:

- Nodes, representing vertices in the topology graph

- Links, representing edges between nodes
- Termination points, representing connection endpoints on nodes

The model supports:

- Multiple topologies per network
- Topologies at different layers
- Overlay topologies
- Explicit relationships and dependencies between topologies

Additionally, links and termination points can reference supporting links and termination points in underlying networks, enabling cross-layer mappings and dependency tracking.

```

module: ietf-network-topology
augment /nw:networks/nw:network:
  +--rw link* [link-id]
    +--rw link-id      link-id
    +--rw source
    | +--rw source-node? -> ../../nw:node/node-id
    | +--rw source-tp?   leafref
    +--rw destination
    | +--rw dest-node? -> ../../nw:node/node-id
    | +--rw dest-tp?   leafref
    +--rw supporting-link* [network-ref link-ref]
      +--rw network-ref
      | -> ../../nw:supporting-network/network-ref
      +--rw link-ref   leafref
augment /nw:networks/nw:network/nw:node:
  +--rw termination-point* [tp-id]
    +--rw tp-id            tp-id
    +--rw supporting-termination-point*
      [network-ref node-ref tp-ref]
    +--rw network-ref
    | -> ../../nw:supporting-node/network-ref
    +--rw node-ref
    | -> ../../nw:supporting-node/node-ref
    +--rw tp-ref          leafref
  
```

Figure 4-11: Topology Model

4.2.4.3 Application of RFC 8345 to the UNITY-6G Digital Twin

Within UNITY-6G, the RFC 8345 model can be used as the core representation of network topologies in the Digital Twin. Its abstraction capabilities allow the Digital Twin to:

- Represent heterogeneous network domains (e.g., transport, RAN, satellite)
- Capture hierarchies and dependencies across technologies and layers
- Provide a common and simplified view of complex infrastructures

By using a unified topology model, the Digital Twin can integrate information from different controllers and management systems while preserving the logical relationships between network components.

The model allows different types of connectivity to be represented in a consistent manner, including wired, wireless and satellite links.

In addition, services in UNITY-6G can be abstracted as higher-level topologies and linked to supporting domain-specific topologies, such as transport or RAN networks. This enables end-to-end visibility from service level down to physical resources, which is a core requirement for digital twin-based analysis and optimization.

The topology model can be further augmented to support dynamic and predictive scenarios, which are central to the Digital Twin concept. In particular, the model can be extended to represent planned or hypothetical topology changes, enabling “what-if” analyses.

In this context, the work of the IETF Time-Variant Routing (TVR) working group is especially relevant. TVR focuses on modelling topological changes over time, where routing decisions depend on when traffic is sent or received. Even with unchanged topology and costs, different routes may be selected at different times. The ietf-tvr-topology model [53] introduces scheduling concepts that allow:

- Nodes and links to be marked as available or unavailable over time
- Periodic or recurring availability patterns
- Explicit modelling of time-dependent topological states

This capability is directly applicable to the UNITY-6G Digital Twin, as it enables the simulation of dynamic behaviours, maintenance windows, failures, and future network evolutions.

4.2.5 Model and Dataset of Transport network power consumption

Telefónica has developed an energy framework that enables the retrieval and analysis of energy-related metrics from a wide range of transport network routers deployed in Telefónica’s network. In previous and ongoing projects such as ALLEGRO⁴ and EXIGENCE⁵, power consumption measurements were reported for a limited set of vendor equipment. These measurements were mainly focused on total equipment power consumption while varying the traffic throughput over time. Power data were obtained by combining measurements from smart power distribution units with energy consumption values reported through the equipment Command Line Interface (CLI).

Within the UNITY-6G project, this dataset will be significantly extended. New measurements will be collected from the same routers as well as from additional transport network equipment. In contrast to previous efforts, this work will not only measure overall energy consumption, but will also take into account additional parameters such as device configuration and the number and type of installed boards and components. To support this enhanced data collection and analysis, two YANG energy models have been defined: “Instantaneous Device Energy” and “Static Device Energy”

The Instantaneous Device Energy model captures energy-related and operational parameters at specific points in time, while the Static Device Energy model processes and aggregates this information over time in order to derive representative values such as typical, maximum, and nominal power consumption.

⁴ <https://www.allegro-he.eu/>

⁵ <https://projectexigence.eu/>

Instantaneous Device Energy Model

<pre> module: instantaneous-device-energy-tid +-- container device +-- leaf name string +-- leaf power decimal64 (W) +-- leaf energy-mode string +-- leaf energy-efficiency decimal64 (W/fps) +-- container configuration +-- leaf configuration string +-- leaf transceivers string +-- leaf boards string +-- container performance-metrics +-- leaf ambient-temperature decimal64 (°C) +-- leaf cpu-load uint8 (0..100) (%) +-- leaf fan-speed union +-- type uint32 +-- type uint8 (0..100) +-- leaf traffic-type string +-- leaf traffic-throughput decimal64 (Gbps) +-- leaf traffic-packet-size uint32 (Bytes) +-- list power-supply +-- key "name" +-- leaf name string +-- leaf rated-power decimal64 (W)- </pre>	<pre> +-- leaf input-current decimal64 (A) +-- leaf input-voltage decimal64 (V) +-- leaf input-power decimal64 (W) +-- leaf output-current decimal64 (A) +-- leaf output-voltage decimal64 (V) +-- leaf output-power decimal64 (W) +-- leaf efficiency uint8 (0..100) (%) +-- list components +-- key "name" +-- leaf name string +-- leaf type string +-- leaf power decimal64 (W) +-- list boards +-- key "name" +-- leaf name string +-- leaf type string +-- leaf power decimal64 (W) +-- list transceivers +-- key "name" +-- leaf name string +-- leaf type string +-- leaf power decimal64 (W) </pre>
---	---

Figure 4-12: Instantaneous device energy model

The instantaneous model is structured around a single top-level container called device, which represents a specific piece of network equipment and includes the following information:

- Name: uniquely identifies the equipment
- Power: total power consumption of the equipment
- Energy-mode: indicates the energy-saving or performance mode currently supported and enabled on the device, which may affect its power consumption behaviour.
- Energy-efficiency, measured in Watts per frame per second (fps)
- Configuration: describes static configuration parameters of the device that may influence energy consumption.
- Performance metrics: includes instantaneous operational data such as temperature, CPU load, fan speed, and traffic processed by the device.
- Power supply: provides per-power supply information, including voltage and current measurements.
- Components: a list of all the components that are in the device and how much power each one consumes in the equipment
- Boards: a list of all boards that are in the device and how much power each one consumes in the equipment
- Transceivers: a list of all transceivers that are in the device and how much power each one consumes in the equipment

An example of this model is shown below:

<pre>{ "instantaneous-device-energy-tid:device": { "name": "HL4_5_1_Huawei", "power": "403.0", "energy-mode": "basic", "configuration": { "configuration": "PIC_1;PIC_2;NPU_10;PIC_5;MPU_11;PSU_1 3;PSU_14", "transceivers": "0x100G", "boards": "PIC_1;PIC_2;NPU_10;PIC_5;MPU_11" }, "performance-metrics": { "cpu-load": 4, "traffic-type": "IDLE", "traffic-throughput": "0.0", "traffic-packet-size": 0 }, "power-supply": [{ "name": "PSU_13", "input-current": "3.91", "input-voltage": "54.66", "input-power": "213.72" }, { "name": "PSU_14", "input-current": "3.46",</pre>	<pre> "input-voltage": "54.72", "input-power": "189.33" }], "boards": [{ "name": "PIC_1", "type": "PIC", "power": "62.40" }, { "name": "PIC_2", "type": "PIC", "power": "62.40" }, { "name": "NPU_10", "type": "NPU", "power": "243.44" }, { "name": "MPU_11", "type": "MPU", "power": "35.00" }], "components": [], "transceivers": [] } }</pre>
---	--

Figure 4-13: Example of and instantaneous device record

Static Device Energy Model

<pre>module: static-device-energy-tid +-- container device +-- leaf name string +-- leaf typical-power decimal64 (W) +-- leaf maximum-traffic decimal64 (Gbps) +-- leaf max-power decimal64 (W) +-- leaf efficiency decimal64 (W/fps) +-- leaf nominal-power decimal64 (W) +-- list power-supply +-- key "name" +-- leaf name string +-- leaf type string +-- leaf capacity decimal64 (W) +-- leaf typical-power decimal64 (W) +-- leaf nominal-power decimal64 (W) +-- list boards</pre>	<pre> +-- leaf type string +-- leaf capacity decimal64 (W) +-- leaf typical-power decimal64 (W) +-- leaf nominal-power decimal64 (W) +-- list components +-- key "name" +-- leaf name string +-- leaf type string +-- leaf capacity decimal64 (W) +-- leaf typical-power decimal64 (W) +-- leaf nominal-power decimal64 (W) +-- list transceivers +-- key "name" +-- leaf name string +-- leaf type string</pre>
--	---

Figure 4-14: Static device energy model

The Static Device Energy model contains essentially the same structural information as the instantaneous model, but with values normalized or aggregated over time to provide stable reference metrics for the device. The main elements captured by this model are:

- Name: uniquely identifies the equipment
- Typical power: average power consumption of the device under representative operating conditions.

- Maximum traffic throughput supported by the device.
- Maximum power consumption registered in the equipment
- Energy-efficiency, measured in Watts per frame per second (fps)
- Nominal power: expected power consumption under normal, vendor-recommended operating conditions.
- Power supply: a list of all power supplies in the device and the capacity, typical and nominal power values it presents
- Components: a list of all components in the device and the capacity, typical and nominal power values it presents
- Boards: a list of all boards in the device and the capacity, typical and nominal power values it presents
- Transceivers: a list of all transceivers in the device and the capacity, typical and nominal power values it presents

An example of this model is shown below:

<pre>{ "device": { "name": "HL4_5_1_Huawei", "typical-power": 58.71, "nominal-power": 550, "max-power": 717.81, "maximum-traffic-throughput": "1Tbps (200Gbps)", "power-supply": [{ "name": "PSU_13", "type": "PSU" }, { "name": "PSU_14", "type": "PSU" }], "boards": [{ "name": "MPU_11", "type": "MPU", "typical-power": 35.0, "nominal-power": 49.0 }, { "name": "PIC_1", "type": "PIC", "capacity": "100G", "typical-power": 62.4, "nominal-power": 70.0 }] } }</pre>	<pre>{ "name": "PIC_2", "type": "PIC", "capacity": "100G", "typical-power": 62.4, "nominal-power": 70.0 }, { "name": "NPU_9", "type": "NPU", "typical-power": 243.0, "nominal-power": 375.0 }, { "name": "NPU_10", "type": "NPU", "typical-power": 240.0, "nominal-power": 375.0 }], "transceivers": [{ "name": "T100G", "type": "100G", "capacity": "100G", "typical-power": 4.0 }] }</pre>
--	---

Figure 4-15: Static device energy record

In the final report, an expanded dataset will be provided, including measurements and derived metrics from multiple types of transport network equipment deployed in Telefónica's network.

4.2.6 Modelling and a dataset of End-to-End 5G delays

An important scenario of DT operation in the project is end-to-end (E2E) latency estimation. Therefore, a dataset of this metric can be an initial part in the DT creation. The dataset used in this work was generated with a custom simulation framework designed to estimate packet-level E2E latency in a 5G New Radio (NR) environment. The adopted modelling approach is inspired by analytical latency models proposed for 5G V2X and V2V communications, where the total latency is decomposed into radio access, transport network, and core network components. In particular, the overall E2E latency can be expressed as: $L_{E2E} = L_{radio} + L_{TN} + L_{CN}$ [54] where L_{radio} denotes the RAN latency, L_{TN} represents the Transport Network (TN) latency, and L_{CN} corresponds to the Core Network (CN) latency. This decomposition follows the modelling philosophy presented in the literature on 5G NR latency analysis for V2X services.

The radio-layer part of the model is motivated by the 5G NR latency framework proposed in [55], where latency depends on OFDM numerology, slot duration, scheduling, retransmissions, modulation and coding, and resource availability. That work shows that radio latency is strongly affected by subcarrier spacing (SCS), slot timing, HARQ operation, and traffic load, especially when the number of active users increases and bandwidth is limited. The transport and core network components are inspired by the end-to-end latency model proposed by Coll-Perales et al.[54] where the E2E latency is extended beyond the radio interface to include propagation, processing, and queuing delays across the transport network and the 5G core.

Although the dataset generator is inspired by these analytical models, it is implemented as a discrete-event simulation tailored to the specific scenarios considered in this study. Therefore, the final dataset should be interpreted as a simulation-based dataset derived from a simplified implementation of the theoretical models, rather than as a direct export of the exact analytical framework presented in the cited papers.

4.2.6.1 Radio latency modelling

In the implemented simulator, packet transmission is performed on a slot-by-slot basis according to the 5G NR numerology. The slot duration is determined from the selected SCS using the standard NR relationship $T_{slot} = 1/2^\mu \text{ ms}$, where $\mu = \log_2 \left(\frac{SCS}{15 \text{ kHz}} \right)$. Hence, increasing the SCS shortens the slot duration and reduces the transmission time granularity, in agreement with the 5G NR numerology design described in the literature [56].

Each user equipment (UE) is associated with a traffic stream characterized by its achieved bitrate and SINR. Packet arrivals are generated stochastically during the simulation, and each packet is placed in the transmission queue of the corresponding UE. For every slot, the simulator identifies active users, sorts them according to channel quality, and allocates radio resources based on the available resource blocks (RBs). The resource grid follows the OFDM structure of 5G NR, where a slot consists of 14 OFDM symbols and the available RBs depend on the selected SCS and bandwidth configuration. The use of RB-based allocation is consistent with the 5G NR radio resource model described in the reference latency studies.

The number of RBs required to transmit a packet is computed from the packet size and the estimated spectral efficiency. The spectral efficiency is derived from the user SINR through a CQI-to-efficiency mapping, which reflects the adaptive modulation and coding principle used in 5G NR. In the analytical model presented by [56] modulation and coding are selected based on CQI and target BLER, with different MCS tables corresponding to low-error-protection and high-error-protection modes. In the implemented generator, this concept is approximated by mapping SINR to CQI and then to a fixed spectral-efficiency table, which is subsequently used to determine the RB demand of each packet.

The radio latency of a successfully delivered packet in the simulator is composed of three main terms: the waiting time in the transmission queue, the processing delay associated with the selected UE capability and SCS, and the HARQ-related delay. UE processing times are parameterized according to the numerology and capability level, following the idea that faster UE processing reduces transmission latency. This is aligned with the analytical treatment of processing delay in the 5G NR latency model, where processing time depends on the numerology and UE processing capability [54].

The simulator also models packet failures and retransmissions through a simplified HARQ mechanism. For every transmission attempt, a packet is marked as successfully received or failed according to a target block error rate (BLER). Two reliability modes are supported by the simulator design: a low-error-protection mode (LEP) and a high-error-protection mode (HEP). In the current dataset instance, only the LEP configuration was used. This design is directly inspired by the distinction between LEP and HEP configurations discussed in the 5G NR analytical model, where one configuration targets a BLER of 0.1 while the more robust one aims at 10^{-5} .

If transmission fails, the simulator schedules a retransmission after the HARQ delay, up to a predefined maximum number of retries. If the packet still cannot be delivered successfully after the maximum number of retransmissions, it is considered lost and assigned a fixed latency value of 200 ms. This value acts as a sentinel indicating transmission failure and is later used to derive the binary success flag included in the dataset. As a result, the dataset not only captures latency values of successfully transmitted packets, but also preserves information about reliability and packet loss.

4.2.6.2 Transport and core network latency

Beyond the radio interface, the simulator augments each packet with latency contributions from the TN and the CN. The TN follows a reference TN topology model used in [54], with three multiplexing nodes (namely: M1, M2, M3) connecting gNB with the core network. This structure, as well as the representative distances and link capacities between these nodes, is directly consistent with the E2E latency model proposed in [54], where the TN is modelled as an optical hierarchical transport network with three multiplexing stages.

The TN latency in the code is composed of two terms: propagation delay and transit delay. Propagation delay depends on the physical distance traversed by the packet and the propagation speed in the transport medium. Transit delay is estimated using queuing theory, where each traversed node is modelled as an M/M/1 system. This choice is consistent with the transport-network modelling approach in the cited E2E latency paper, where TN nodes are

represented as M/M/1 queues to capture the effect of load-dependent congestion. Depending on the selected deployment mode, the simulator can approximate edge, midhaul, or cloud-oriented transport paths, which results in different propagation distances and a different number of queued nodes [54].

The core network latency is modelled separately and activated when the cloud-oriented core mode is used. In this case, the simulator estimates the latency introduced by the UPF and intermediate core nodes using a propagation term and an M/D/1-based queuing model. This is in line with the analytical E2E model, where the core network delay is represented as the sum of propagation, processing, transmission, and queuing effects, and M/D/1 queues are used to describe the behavior of user-plane functions in the 5G core. Consequently, the latency samples generated reflect not only the radio load but also the influence of transport and core aggregation under increasing traffic intensity.

4.2.6.3 Scenario design and dataset structure

The dataset is available in [55] and is organized as a set of scenario-specific CSV files, each corresponding to a distinct simulation configuration. In the experiments conducted in this work, the main scenario dimensions include the number of active UEs, the selected SCS, the UE capability class, the baseline SINR assigned to the UEs, and the offered bitrate per UE. More specifically, the dataset includes 48 simulation scenarios obtained by combining 1 and 10 UEs, SCS values of 15, 30, and 60 kHz, UE capability classes 1 and 2, baseline SINR values of 5 and 15 dB, and offered bitrates of 5 and 10 Mbps per UE. In the current dataset, only LEP configuration was used.

These parameters were chosen because they directly affect the contention for radio resources and the slot duration, which are among the most important determinants of latency in 5G NR systems. The full set of parameters used during dataset generation, together with their values in the current dataset instance, is summarized in Table 4-1.

Table 4-1: Simulation parameters used for dataset generation

Category	Parameter	Field name	Values	Description
Radio	Number of UEs	ue_count	1, 10	Number of active UEs in the cell
Radio	Subcarrier spacing	scs_khz	15, 30, 60 kHz	Defines numerology and slot duration
Radio	UE capability	ue_capabilities	1, 2	Determines UE processing time
Radio	MCS mode	mcs_mode	LEP	Reliability / protection mode used in this dataset instance
Radio	Packet size	packet_size_bits	1200 bits	Application packet size
Radio	Bandwidth	bandwidth_hz	20 MHz	System bandwidth used to determine available RBs
Radio	Scheduler type	scheduler_type	Dynamic	Scheduling strategy assumed in the simulation

Radio	Max HARQ retries	max_retries	3	Maximum number of retransmissions
Channel / traffic	UE SINR	default_sinr_db	5, 15 dB	Baseline SINR assigned to UEs
Channel / traffic	UE bitrate	default_bitrate_bps	5, 10 Mbps	Offered traffic load per UE
TN/CN	Transport mode	tn_mode	mid	Transport topology configuration
TN/CN	Core mode	core_mode	cloud	Core network configuration
TN/CN	Packet arrival intensity	lambda_packets_per_s	2000 pkt/s	Parameter controlling TN/CN load
Simulation	Duration	duration_sec	10 s	Simulation time window
Simulation	Seed	seed	42	Random seed used for reproducibility

The dataset has packet-level granularity, meaning that each row corresponds to one packet observed during the simulation rather than to an aggregate scenario-level measurement. This structure makes the dataset suitable for both descriptive and predictive analysis. At the most basic level, it enables a study of latency distributions, packet loss, and the effect of load on E2E performance. At the same time, since each packet is represented as a separate observation with associated radio-configuration features, the dataset can also be used in supervised learning tasks, such as latency prediction, congestion detection, reliability estimation, or performance-aware scheduling optimization.

4.2.7 Monitoring system

The Monitoring System (MS) constitutes a fundamental enabler of the Digital Twin (DT) framework described in Section 4.1, acting as the southbound interface between the physical network infrastructure and the DT domain. Feeding the DT modules with accurate, real-time Key Performance Indicators (KPIs) from heterogeneous network components is essential to instantiate and operate a functional DT, including the development of Proof-of-Concept (PoC) use cases. The diversity of networking modules in B5G/5G experimental platforms, spanning radio access, transport, and core network elements, makes it impractical to rely on a single standardized telemetry interface. For instance, while the O-RAN E2 interface provides access to RAN-level measurements, it does not cover metrics from other domains such as edge computing nodes, virtualized network functions, or transport segments.

Although existing monitoring solutions such as Prometheus or Grafana offer well-established telemetry capabilities, they are primarily designed for IT-centric environments and assume homogeneous data sources and standardized exporters, which are not always available in heterogeneous 5G/B5G platforms. The MS has therefore been developed as a custom solution, tailored to integrate arbitrary monitoring probes, support heterogeneous interfaces, and adapt to the evolving requirements of experimental 5G deployments, enabling seamless interoperability with the DT framework.

The Monitoring System (MS) has been conceived as a flexible, scalable, and cloud-native framework for the monitoring and analysis of distributed 5G environments. The increasing

complexity of Beyond 5G (B5G) and 5G infrastructures, characterized by virtualization, softwarization, edge computing, and multi-domain deployments, requires advanced monitoring mechanisms capable of operating efficiently across heterogeneous and geographically distributed infrastructures.

In this context, the proposed MS addresses the need for real-time monitoring, data collection, aggregation, and analysis in highly dynamic network environments. The system enables the acquisition of operational metrics from multiple network segments and computational domains while ensuring scalability, modularity, and interoperability. The architecture has been designed following modern cloud-native principles and relies on containerized microservices orchestrated through Kubernetes. This design choice allows the system to dynamically adapt to changing workloads and deployment conditions while simplifying lifecycle management and resource orchestration. Furthermore, the Monitoring System supports both centralized and distributed deployment models, enabling its integration in diverse operational scenarios, including edge computing infrastructures, distributed cloud environments, and multi-domain 5G experimental platforms.

4.2.7.1 Architectural design

The MS architecture follows a distributed microservices-based approach, where each functional component is implemented as an independent containerized service. This modular design improves scalability, maintainability, and extensibility while enabling independent deployment and upgrading of system components. The infrastructure is deployed on top of Kubernetes clusters, which provide automated orchestration, resource allocation, service discovery, fault tolerance, and scalability mechanisms. Kubernetes enables the system to efficiently manage the lifecycle of monitoring services while ensuring high availability and operational resilience. The architecture is organized according to a hierarchical model (Figure 4-16) composed of local monitoring entities and higher-level aggregation components.

Each MS includes a **Manager**, responsible for the lifecycle management and configuration of monitoring tasks; a **Broker** (Apache Kafka), which provides the messaging bus for data exchange; a **Time-Series Database (TSDB)** for storing monitoring metrics; and several **Sampling Functions (SFs)** that collect measurements from monitored resources. Monitoring data can be obtained from different types of network functions, including **Virtual Network Functions (VNFs)**, **Physical Network Functions (PNFs)**, and **Cloud-Native Network Functions (CNFs)**, as well as from service instances. The architecture also incorporates a **Cloud Infrastructure Services API (CIS API)** and a **Service Management Layer (SML)** to interact with the underlying cloud and service-management environment. At higher levels, an **Analytics Engine (AE)** processes the collected monitoring information, while a **Decision Engine (DE)** uses these insights to support orchestration and management actions. Local monitoring entities are responsible for collecting metrics from nearby network elements or computational resources, while upper-level nodes aggregate and process monitoring information obtained from multiple sources through the hierarchical interfaces shown in the figure.

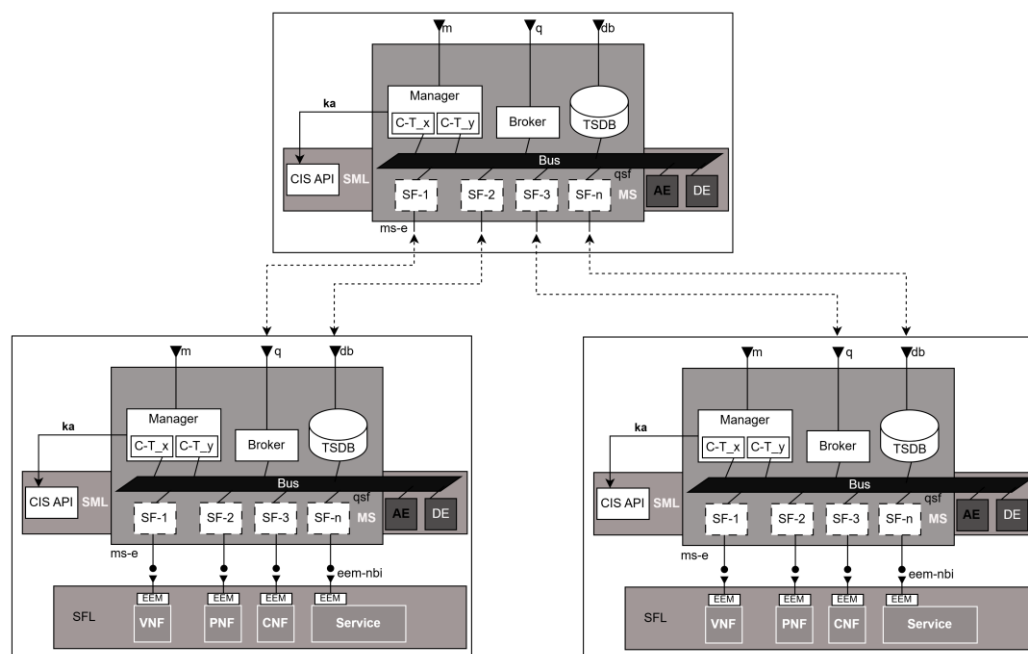


Figure 4-16: Hierarchical deployment of monitoring systems.

This hierarchical organization reduces the amount of raw monitoring traffic transmitted through the network and enables distributed data processing closer to the monitored resources. As a result, the architecture contributes to reducing latency, improving scalability, and optimizing resource utilization in large-scale deployments. The use of microservices also facilitates the integration of new monitoring functionalities and analytics modules without requiring significant modifications to the overall system architecture. Additional monitoring agents, data consumers, or analytics engines can be incorporated dynamically depending on deployment requirements.

4.2.7.2 Messaging and data distribution framework

To support efficient real-time communication between monitoring components, the system integrates Apache Kafka as the main messaging and data distribution platform.

Kafka operates as a distributed publish-subscribe messaging bus that enables reliable and scalable data streaming among producers and consumers. Monitoring data generated by different Sampling Functions (SFs) are published into Kafka topics and can subsequently be consumed by storage systems, visualization tools, analytics modules, or external applications. The adoption of Kafka provides several advantages for distributed monitoring scenarios: a High-throughput data streaming capabilities, a Scalability through distributed brokers and topic partitioning, Fault tolerance and resilience mechanisms, Decoupling between data producers and consumers.

Support for asynchronous communication.

This architecture allows multiple monitoring consumers to simultaneously process the same monitoring streams for different purposes, including visualization, anomaly detection, machine learning analytics, or historical storage.

The use of Kafka also improves interoperability between components and enables the integration of external systems into the monitoring pipeline. Since communication is based on

standardized messaging mechanisms, additional modules can subscribe to monitoring streams without affecting the operation of existing components.

Monitoring Functions and Data Collection

The core monitoring capabilities of the system are implemented through Sampling Functions (SFs). These functions are responsible for collecting metrics from the monitored infrastructure and generating monitoring streams. SFs can be configured to monitor a wide range of metrics, including CPU and memory utilization, network throughput and latency, Packet loss and jitter, Service availability, Radio access network indicators, Edge and cloud resource consumption, Application-level performance indicators.

Each Sampling Function operates independently and can be customized according to the requirements of a specific monitoring task. The functions are generally implemented using lightweight scripts or applications, commonly developed in Python, and deployed as containerized services.

The modular nature of SFs allows the system to easily support new monitoring probes and measurement methodologies. Additional monitoring agents can be integrated dynamically to address evolving monitoring requirements in future deployments. Once collected, monitoring metrics are formatted and transmitted to Kafka topics. This decoupled communication model ensures that data acquisition tasks remain independent from downstream processing tasks. The monitoring framework also supports configurable sampling intervals and execution parameters, allowing users to balance monitoring granularity and system overhead depending on operational requirements.

Monitoring System Manager

The lifecycle management of monitoring tasks is performed by the Monitoring System Manager (MS Manager), which acts as the orchestration and control component of the architecture. The MS Manager provides programmatic interfaces that enable users or external orchestration platforms to configure, deploy, update, and terminate monitoring tasks dynamically.

Through the MS Manager, users can define:

- Monitoring task parameters.
- Sampling intervals.
- Resource allocation policies.
- Kafka topics.
- Container execution configurations.
- Deployment constraints.

After receiving a monitoring request, the MS Manager automatically deploys the required monitoring resources within the Kubernetes infrastructure. This process includes the creation of containers, configuration of communication interfaces, allocation of computational resources, and initialization of monitoring agents.

The automation capabilities provided by the MS Manager significantly reduce operational complexity and facilitate the rapid deployment of monitoring services in dynamic environments.

In addition, the manager supports the removal and reconfiguration of monitoring tasks during runtime, enabling adaptive monitoring strategies based on changing network conditions or operational requirements.

4.2.7.3 Deployment and Cloud-Native Operation

The Monitoring System has been specifically designed to operate in cloud-native environments. All components are containerized and orchestrated using Kubernetes, enabling portable and reproducible deployments across different infrastructures.

The deployment process can be automated through configuration templates and orchestration scripts, minimizing manual intervention and simplifying operational procedures.

Kafka clusters are deployed and managed using Kubernetes operators such as Strimzi, which provide automation for broker deployment, topic management, scaling, and fault recovery.

The use of Kubernetes provides several operational advantages:

- Automated scaling of monitoring components.
- Self-healing mechanisms.
- Dynamic resource management.
- Simplified service discovery.
- Infrastructure portability.
- High availability.

These capabilities are particularly relevant in 5G and B5G environments, where network conditions and workloads may vary dynamically.

Furthermore, the cloud-native design facilitates deployment across edge nodes, distributed data centers, and hybrid cloud infrastructures, supporting highly distributed operational scenarios.

4.2.7.4 Multi-Domain Monitoring Support

One of the main characteristics of the proposed Monitoring System is its support for multi-domain monitoring scenarios.

Modern 5G infrastructures frequently involve multiple administrative and technological domains, including radio access networks, transport networks, edge infrastructures, cloud data centers, and core network components. Monitoring these heterogeneous environments requires mechanisms capable of aggregating and correlating information obtained from multiple distributed sources.

The proposed architecture enables the integration of monitoring streams originating from different domains into a unified monitoring framework. Metrics collected at local domains can be aggregated and processed centrally while preserving the distributed nature of the infrastructure. This capability enables end-to-end monitoring across multiple segments of the network and facilitates advanced analytics involving cross-domain correlations. Additionally,

the hierarchical organization of the monitoring framework reduces bandwidth consumption and improves scalability by limiting the transmission of unnecessary raw monitoring data.

Data Visualization and Analytics

The monitoring data generated by the system can be integrated with external visualization and analytics tools. Metrics streamed through Kafka can be consumed by databases, processing engines, or dashboard platforms such as Grafana, enabling real-time visualization of network and infrastructure status.

These visualization capabilities facilitate:

- Operational monitoring.
- Fault detection.
- Performance analysis.
- Capacity planning.
- SLA verification.
- Detection of anomalous behaviors.

Real-time dashboards allow operators and researchers to observe network conditions dynamically and identify performance degradation or abnormal events.

The monitoring framework can also support advanced analytics modules based on machine learning or artificial intelligence techniques, enabling predictive maintenance and automated decision-making mechanisms.

4.2.7.5 Conclusion

The Monitoring System provides a flexible, scalable, and extensible monitoring framework tailored to the requirements of distributed 5G and Beyond 5G environments.

By combining cloud-native technologies, distributed messaging systems, and modular monitoring functions, the architecture enables efficient real-time monitoring across heterogeneous and multi-domain infrastructures.

The adoption of Kubernetes and Apache Kafka ensures scalability, resilience, and interoperability while facilitating dynamic deployment and lifecycle management of monitoring services.

The hierarchical architecture and distributed monitoring approach contribute to reducing operational overhead and improving scalability in large-scale deployments.

Overall, the proposed Monitoring System constitutes a robust and adaptable solution capable of supporting advanced monitoring, analytics, and decision-making processes in next-generation network infrastructures.

4.2.8 Power Grid Representation Model

Within UNITY-6G the CIGRE benchmark High Voltage (HV) Transmission Network can be used as the foundational architectural representation for the power grid interface. By

integrating this benchmark into the project's Digital Twin (DT) framework we establish a standardized environment for testing 6G domain interactions with the energy market. The abstractions provided by the CIGRE-based DT allow for the abstractions below.

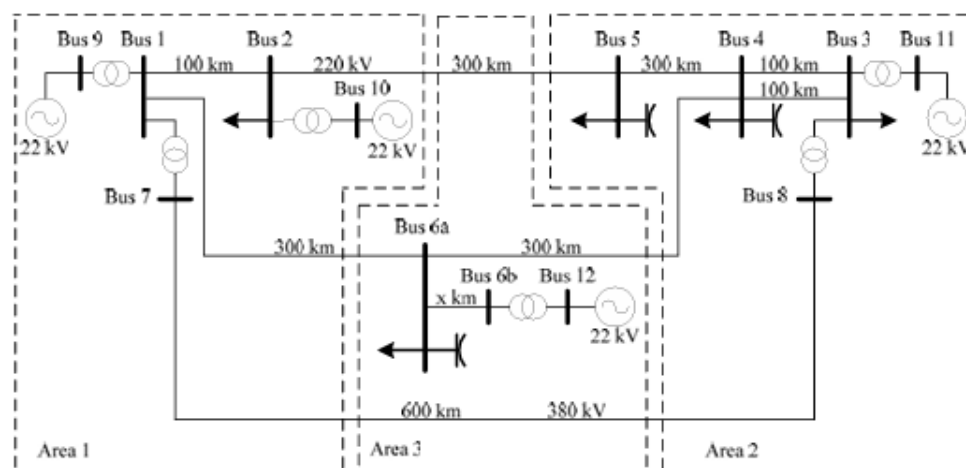


Figure 4-17: Cigre HV benchmark Model

Energy colour Representation, the DT maps the energy mix at the point of interconnection for a specific 6G domain.

The model will be able to perform time series forecasting to predict next-day energy availability and grid stress, allowing for the UNITY-6G architecture to pre-emptively optimize heavy computational tasks.

4.2.9 Dataset of IEEE 802.11 Multi Link Operation (MLO) at 2.4 GHz and 5 GHz

This dataset contains measurements from 16 distinct scenarios of Multi-Link Operation (MLO) in a wireless environment under varying levels of interference and traffic load. The dataset is designed to analyze performance behaviour when dual-band connectivity is exploited under congestion conditions.

MLO operates simultaneously over 2.4 GHz and 5 GHz frequency bands, where application packets from the main traffic flow are duplicated and transmitted across both links to improve reliability and latency performance.

Experimental Setup

Each scenario is defined by a different level of wireless interference generated using UDP traffic load. The interference ranges from 0% to 75% of the link capacity load, applied independently to the respective links.

The experimental setup consists of the following parameters:

- **Frequency bands:** 2.4 GHz and 5 GHz
- **Distance between MLO station and AP:** 5 meters
- **Main traffic flow:** ICMP ping traffic
 - Packet interval: 1 packet every 50 ms

- **Interference traffic:** UDP traffic generated from Single-Link Operation (SLO) stations
- **Interference control:** Separate UDP flows per link targeting the MLO access point

Data Collection

For each scenario, two types of measurements are collected over a 15-minute duration: Channel State Information (CSI) and Packet Error Ratio (PER) Measurements are sampled at **1-second intervals**. In addition, ICMP ping measurements are recorded continuously to evaluate end-to-end latency performance.

5 CONCLUSIONS

This deliverable has presented the design and implementation of the network components, organized on a per-domain basis. The specific pieces providing the implementation of each component in the UNITY-6G architecture has been described for the RAN (and RIC), core network, non-3GPP RAN, transport and satellite. Besides the plans for development, including both enhancements and extensions on top of the presented elements, a few illustrative initial results have also been detailed, highlighting both the progress of WP3 and the alignment of the tasks towards fulfilling the ambitious goals of UNITY-6G.

Similarly, the design and initial deployment of the network exposure solutions has also been covered in this deliverable. The description has not only touched its design and practical realisation aspects, but focused on the definition of the interfaces and APIs that facilitate the interaction and joint management of the different network elements. The design has also considered the utilization of the network exposure solutions by the different actors in the vertical application domain, as well as the consumption of the exposed metrics, with the goal of providing a comprehensive, harmonized and self-sufficient 6G network architecture, where AI-driven monitoring and control processes jointly manage and optimize the different coexisting domains and technologies.

Finally, the document has also unfolded its vision for the realisation of accurate DT implementations, which are deployed in parallel to the physical network and allow the study and validation of advanced solutions to optimize its operation without disrupting it. The deliverable has delved into the definition of the DT architecture proposed by UNITY-6G, while also describing its specifications and design. Moreover, the related tools, models and generation of datasets, which allow to further develop and refine the AI-driven mechanisms that will be deployed to monitor and control the (physical) network. Illustrative initial results have also been included to demonstrate the advances of these tasks during the first phase of the execution of WP3.

6 REFERENCES

- [1] M. Polese et al., "Understanding O-RAN: Architecture, Interfaces, Algorithms, Security, and Research Challenges," IEEE Communications Surveys and Tutorials, vol. 25, no. 2, 2023.
- [2] O-RAN Working Group 3, "O-RAN near-RT RAN intelligent controller near-RT RIC architecture 2.00," O-RAN, Alfter, Germany, document O-RAN.WG3.RICARCH-v02.00, Mar. 2021.
- [3] O-RAN Working Group 4, "O-RAN Fronthaul control, user and synchronization plane specification 7.0," O-RAN, Alfter, Germany, document ORAN-WG4.CUS.0-v07.00 Technical Specification, Jul. 2021.
- [4] O-RAN Working Group 2, "O-RAN non-RT RIC architecture 1.0," O-RAN, Alfter, Germany, document O-RAN.WG2.Non-RT-RIC-ARCH-TS-v01.00 Technical Specification, Jul. 2021.
- [5] The Open Centralized Unit Distributed Unit (OCUDU) project, The Linux Foundation Projects. Last accessed at <https://ocudu.org> on April 21, 2026.
- [6] The srsRAN Project. Last accessed at <https://srsran.com/5g> on April 21, 2026.
- [7] The OCUDU Project GitLab repository. Last accessed at <https://gitlab.com/ocudu/ocudu> on April 21, 2026.
- [8] The OCUDU Project documentation. User manual, Outputs section. Last accessed at http://docs.ocudu.org/user_manual/outputs on April 21, 2026.
- [9] O-RAN Software Community, Near-RT RIC Platform Documentation, Available at: <https://docs.o-ran-sc.org/en/latest/projects.html#near-realtime-ran-intelligent-controller-ric> [Accessed: 30/03/26]
- [10] O-RAN Alliance, "O-RAN Cloud Architecture and Deployment Scenarios for O-RAN Virtualized RAN 8.01," 2025. Available at: <https://specifications.o-ran.org/specifications> [Accessed: 30/03/26].
- [11] Jorge Baranda, Piotr Gawlowicz, Oriol Font-Bach, Engin Zeydan, Josep Mangués-Bafalluy, "Automated Orchestration of O-RAN Near-RT RIC with Integrated Remote xApp Management", in IEEE/IFIP Network Operations and Management Symposium (NOMS'26).
- [12] srsRAN Project - ORAN-SC-RIC, Available at: <https://github.com/srsran/oran-sc-ric> [Accessed: 30/03/26].
- [13] J. Baranda et al, "Disaggregating a 5G Non-Public Network via On-demand Cloud-Native UPF Deployments", in IEEE CNSM, Niagara Falls, Canada, 2023.
- [14] J. Baranda, et al., "End-to-End Slice Orchestration in a 5G cloud-native Mobile Network with O-RAN Split 7.2", in IEEE INFOCOM, London, UK, 2025
- [15] J. Baranda et al, "Design and Evaluation of an Orchestrated E2E Cloud-Native Mobile Network with O-RAN", in IEEE WiMob, Marrakech, Morocco, 2025.
- [16] 6GBLUR project Gitlab repository, [Online] Available at: <https://gitlab.cttc.es/mdi-6gblur>
- [17] L. Bonati, M. Polese, S. d'Oro, P. Brach del Prever, and T. Melodia, "5G-CT: Automated Deployment and Over-the-Air Testing of End-to-End Open Radio Access Networks", in IEEE Communications Magazine, Vol. 63, Issue 1, pp. 155-160, 2025.
- [18] X. Foukas, B. Radunovic, M. Balkwill, and Z. Lai, "Taking 5G RAN Analytics and Control to a New Level", in ACM International Conference on Mobile Computing and

- Networking (MobiCom'23), pp. 1-16, Madrid, Spain, 2023.
- [19] M. Mekki, S. Arora, and A. Ksentini, "A scalable monitoring framework for network slicing in 5G and beyond mobile networks", in *IEEE Transactions on Network and Service Management*, Vol.19, Issue 1, pp. 413 - 423, 2022.
 - [20] S. Barrachina-Muñoz, M. Payaró, and J. Manges-Bafalluy, "Cloud-native 5G experimental platform with over-the-air transmissions and end-to-end monitoring", in *IEEE International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP'22)*, pp. 22-26, Porto, Portugal, 2022.
 - [21] J. Baranda, A. Bel, S. Barrachina-Muñoz, M. Payaró, J. Manges-Bafalluy, "An Automated Configurable Cloud-Native Monitoring System for the Radio Access Network", in *IEEE Wireless Communications and Networking Conference (WCNC'25)*, Milan, Italy, 2025.
 - [22] Giannopoulos, Anastasios, et al. "Supporting intelligence in disaggregated open radio access networks: Architectural principles, AI/ML workflow, and use cases." *IEEE access* 10 (2022): 39580-39595.
 - [23] Giannopoulos, Anastasios, et al. "Interoperable rApp/xApp Control over O-RAN for Mobility-aware Dynamic Spectrum Allocation." *arXiv preprint arXiv:2601.13769* (2026).
 - [24] Open5Gs. Last accessed at <https://open5gs.org/> on April 21, 2026.
 - [25] X. Jiao, W. Liu, M. Mehari, M. Aslam, and I. Moerman, "Openwifi: A free and open-source IEEE802.11 SDR implementation on SoC," *IEEE Vehicular Technology Conference*, vol. 2020-May, 5 2020.
 - [26] Aslam, M., Liu, W., Jiao, X., Haxhibeqiri, J., Miranda, G., Hoebeke, J., ... & Moerman, I. (2021). Hardware efficient clock synchronization across Wi-Fi and Ethernet-based network using PTP. *IEEE Transactions on Industrial Informatics*, 18(6), 3808-3819.
 - [27] "IEEE draft standard for local and metropolitan area networks – frame replication and elimination for reliability - amendment 2: Extended stream identification functions," *IEEE P802.1CBdb/D2.0*, July 2021, pp. 1–93, 2021.
 - [28] J. Haxhibeqiri, P. H. Isolani, J. M. Marquez-Barja, I. Moerman, and J. Hoebeke, "In-band network monitoring technique to support sdn-based wireless networks," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 627–641, 2020.
 - [29] G. Miranda, E. Municio, J. Haxhibeqiri, J. Hoebeke, I. Moerman, and J. M. Marquez-Barja, "Enabling time-sensitive network management over multi-domain wired/wi-fi networks," *IEEE Transactions on Network and Service Management*, vol. 20, no. 3, 2023.
 - [30] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router," *ACM Transactions on Computer Systems (TOCS)*, vol. 18, no. 3, pp. 263–297, 2000..
 - [31] 'ETSI Software Development Group TeraFlowSDN - SDG TFS'. Accessed: Mar. 03, 2026. [Online]. Available: <https://tfs.etsi.org/>.
 - [32] M. Boucadair, O. G. de Dios, S. Barguil, and L. A. Munoz, 'A YANG Network Data Model for Layer 2 VPNs', Internet Engineering Task Force, Request for Comments RFC 9291, Sep. 2022. doi: 10.17487/RFC9291..
 - [33] S. Barguil, O. G. de Dios, M. Boucadair, L. A. Munoz, and A. Aguado, 'A YANG Network Data Model for Layer 3 VPNs', Internet Engineering Task Force, Request for Comments RFC 9182, Feb. 2022. doi: 10.17487/RFC9182..
 - [34] B. Wu, D. Dhody, R. Rokui, T. Saad, and 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-25, May 2025. Accessed: Apr. 14, 2026., [Online]. Available: <https://datatracker.ietf.org/doc/draft-ietf-teas-ietf-network-slice-nbi-yang>.
- [35] O-RAN Working Group 3, "O-RAN nE2 General Aspects and Principles 2.00," O-RAN, Alfter, Germany, document O-RAN.WG3.E2GAP-v02.01, Jul. 2021..
 - [36] The OCUDU Project documentation. User manual, Tutorials, Near-RT RIC section. Last accessed at http://docs.ocudu.org/user_manual/tutorials/near-rt-ric on April 21, 2026..
 - [37] A. Bierman, M. Björklund, and K. Watsen, 'RESTCONF Protocol', no. 8040. RFC Editor, Jan-2017..
 - [38] A. Clemm, J. Medved, R. Varga, N. Bahadur, H. Ananthakrishnan, and X. Liu, 'A YANG Data Model for Network Topologies', no. 8345. RFC Editor, Mar-2018..
 - [39] X. Liu, L. M. Contreras, S. Belotti, A. Guo, and I. Busi, 'IETF Network Slice Topology YANG Data Model', Internet Engineering Task Force, Jan. 2026.
 - [40] 3GPP TS 29.122 V18.5.0 (2024-12), T8 reference point for Northbound APIs; Technical realization of Service Capability Exposure (Release 18). Available: http://www.3gpp.org/ftp/Specs/archive/29_series/29.122/.
 - [41] 3GPP TS 29.571 V17.2.0 (2021-12), 5G System; Common Data Types for Service Based Interfaces; Stage 3 (Release 17). Available: http://www.3gpp.org/ftp/Specs/archive/29_series/29.571/.
 - [42] 3GPP TS 29.514 V17.1.0 (2021-06), 5G System; Policy Authorization Service; Stage 3 (Release 17). Available: http://www.3gpp.org/ftp/Specs/archive/29_series/29.514/.
 - [43] CAMARA (Linux Foundation): CAMARA Project, CAMARA: Open Source API Framework for Telco Services, Linux Foundation, 2026.[Online]. Available: <https://camaraproject.org>.
 - [44] GSMA Open Gateway: GSMA, Open Gateway Initiative, 2024.[Online]. Available: <https://www.gsma.com/open-gateway>.
 - [45] 3GPP (5G Architecture) 3GPP, TS 23.501: System Architecture for the 5G System, Release 17, 2023.[Online]. Available: <https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3144>.
 - [46] ETSI MEC ETSI, Multi-access Edge Computing (MEC); Framework and Reference Architecture, ETSI GS MEC 003 V3.1.1, 2022.[Online]. Available https://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/03.01.01_60/gs_MEC003v030101p.pdf
 - [47] CAMARA Edge / Computing APIs CAMARA Project, Edge Cloud APIs (Computing Services Sub-Project), 2026.[Online].Available: <https://github.com/camaraproject/EdgeCloud>.
 - [48] CAMARA EAM API CAMARA Project, Edge Application Management API, 2026.[Online]. Available <https://github.com/camaraproject/EdgeApplicationManagement>
 - [49] Nearby Computing, NearbyOne Platform Description, 2021.[Online]. Available: <https://www.nearbycomputing.com/wp-content/uploads/2021/08/NearbyOne-Product-Data-Sheet-v2.0.pdf>.
 - [50] The Kubernetes Authors, Kubernetes Documentation, 2024.[Online]. Available: <https://kubernetes.io/docs>.
 - [51] ETSI OpenCAPIF SDK, <https://labs.etsi.org/rep/ocf/sdk>

- [52] ETSI OpenCAPIF Repository, <https://labs.etsi.org/rep/ocf/capif>
- [53] Y. Qu, A. Lindem, E. Kinzie, D. Fedyk, and M. Blanchet, 'YANG Data Model for Scheduled Attributes', Internet Engineering Task Force, Feb. 2026..
- [54] B. Coll-Perales et al., "End-to-End V2X Latency Modeling and Analysis in 5G Networks," in IEEE Transactions on Vehicular Technology, vol. 72, no. 4, pp. 5094-5109, April 2023, doi: 10.1109/TVT.2022.3224614..
- [55] Pakuła, M., & Kryszkiewicz, P. (2026). 5G NR End-to-End Latency Simulation Dataset [Data set]. Zenodo. <https://doi.org/10.5281/zenodo.20035549>.
- [56] M. C. Lucas-Estañ et al., "An Analytical Latency Model and Evaluation of the Capacity of 5G NR to Support V2X Services Using V2N2V Communications," in IEEE Transactions on Vehicular Technology, vol. 72, no. 2, Feb. 2023, doi: 10.1109/TVT.2022.3208306..
- [57] 5G-HUB Project, «D2.2 System Architecture, Specifications and Interfaces,» Nov. 2025.
- [58] 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, 2021, pp. 411–425.
- [59] Cloud Native Computing Foundation (CNCF), Helm: The Kubernetes Package Manager, 2024.[Online]. Available: <https://helm.sh/>.
- [60] V. Pitsilis et al., "B5G Core Network Openness: A Proof of Concept Implementation of Location Reporting via Network Exposure Function," 2025 IEEE Conference on Network Function Virtualization and Software-Defined Networking (NFV-SDN), Athens, Greece, 2025.
- [61] O-RAN white paper "Deployments of O-RAN-based Non-Terrestrial Networks" February 2025
- [62] Giuliano, R., & Innocenti, E. (2023). Machine Learning Techniques for Non-Terrestrial Networks. Electronics, 12(3), 652. <https://doi.org/10.3390/electronics120306523>
- [63] O-RAN Alliance, "O-RAN Architecture Description 16.00," 2026 Available at: <https://specifications.o-ran.org/specifications>
- [64] O-RAN Alliance, "Near-RT RIC Architecture 8.00," 2026. Available at: <https://specifications.o-ran.org/specifications>
- [65] O-RAN Alliance, "O-RAN Control, User and Synchronization Plane Specification 20.00," 2026. Available at: <https://specifications.o-ran.org/specifications>
- [66] UNITY-6G: Deliverable 2.3 1st Release of UNITY-6G Architecture
- [67] Li, X., Ngu, A. H. H., & Metsis, V. (2022). Tts-cgan: A transformer time-series conditional gan for biosignal data augmentation. arXiv preprint arXiv:2206.13676.
- [68] Madane, A., Dilmi, M. D., Forest, F., Azzag, H., Lebbah, M., & Lacaille, J. (2022). Transformer-based conditional generative adversarial network for multivariate time series generation. arXiv preprint arXiv:2210.02089.
- [69] Ma, X., Cui, X., Pun, M. O., & Cheng, Z. (2025). LTTS-GAN: A long-term time series generative adversarial network. Signal Processing, 110161.
- [70] Wang, D., Cheng, M., Liu, Z., & Liu, Q. (2024). Timedart: A diffusion autoregressive transformer for self-supervised time series representation. arXiv preprint arXiv:2410.05711.
- [71] Ge, Y., Li, J., Zhao, Y., Wen, H., Li, Z., Qiu, M., ... & Pan, S. (2025). T2s: High-resolution time series generation with text-to-series diffusion models. arXiv preprint

- arXiv:2505.02417.
- [72] Dai, X., Xu, Z., Cai, W., & Xu, Q. (2025). From Samples to Scenarios: A New Paradigm for Probabilistic Forecasting. arXiv preprint arXiv:2509.19975.
 - [73] Rezazadeh, F., Chergui, H., Debbah, M., Song, H., Niyato, D., & Liu, L. (2025). Agentic World Modeling for 6G: Near-Real-Time Generative State-Space Reasoning. arXiv preprint arXiv:2511.02748.
 - [74] UNITY-6G: Deliverable 2.1 Techno-economic analysis of the 6G environment, use case requirements and KPIs
 - [75] UNITY-6G: Deliverable 2.2 Trust model and trust management approaches
 - [76] 3GPP, "Technical Specification Group Services and System Aspects; Functional architecture and information flows to support Common API Framework for 3GPP Northbound APIs; Stage 2 (Release 19)," 3GPP TS 23.222, V19.0.0, Dec. 2025.
 - [77] GSMA, "GSMA Open Gateway Service Management and Architecture Framework," GSMA Permanent Reference Document NG.134, v1.0, 2023
 - [78] O-RAN Alliance, "O-RAN Use Cases Detailed Specification v19.00," O-RAN.WG1.TS.Use-Cases-Detailed-Specification-R005-v19.00, Technical Specification, Feb. 2026.
 - [79] 3GPP, "Technical Specification Group Services and System Aspects; System architecture for the 5G System (5GS); Stage 2 (Release 19)," 3GPP TS 23.501, V19.5.0, Mar. 2026
(https://www.etsi.org/deliver/etsi_ts/123500_123599/123501/19.05.00_60/ts_123501v190500p.pdf)
 - [80] CAMARA Project, <https://github.com/camaraproject>