Beyond 5G Multi-Tenant Private Networks Integrating Cellular, Wi-Fi, and LiFi, Powered by Artificial Intelligence and Intent Based Policy

# 5G-CLARITY Deliverable D4.3

# Evaluation of E2E 5G Infrastructure and Service Slices, and of the Developed Self-Learning ML Algorithms

| | |
|---|---|
| **Contractual Date of Delivery:** | **May 31, 2022** |
| **Actual Date of Delivery:** | **October 26, 2022** |
| **Editor(s):** | **Joseph McNamara (LMI)** |
| **Author(s):** | **Anil Yesilkaya, Ardimas Purwita (USTRATH),**<br>**Carlos Colman Meixner, Haiyuan Li, Hilary Frank, Shuangyi Yan, Xueqing Zhou (UNIVBRIS),**<br>**Daniel Camps Mur (i2CAT),**<br>**Daniel González Sánchez, Ignacio Soto Campos, David Fernández Cambronero (UPM),**<br>**Jonathan Prados Garzón, Juan José Ramos Muñoz, Lorena Chinchilla Romero, Pablo Muñoz Luengo (UGR),**<br>**Jordi Pérez-Romero, Oriol Sallent, Irene Vilà (UPC),**<br>**Jose Antonio Ordonez Lucena, Ignacio Dominguez Martinez-Casanueva (TID),**<br>**Meysam Goodarzi (IHP),**<br>**Tezcan Cogalan (IDCC),**<br>**Mir Ghoraishi (GIGASYS)** |
| **Work Package:** | **WP4** |
| **Target Dissemination Level:** | **Public** |

This document has been produced in the course of 5G-CLARITY Project. The research leading to these results received funding from the European Commission H2020 Programme under grant agreement No. H2020-871428. All information in this document is provided “as is", there is no guarantee that the information is fit for any particular purpose. The user thereof uses the information at its own risk and liability. For the avoidance of all doubts, the European Commission has no liability in respect of this document, which is merely representing the authors view.

## Revision History

| Revision | Date | Editor /Commentator | Description of Edits |
|---|---|---|---|
| 0.1 | 29/10/21 | Joseph McNamara (LMI) | Creation of document. Rough draft of ToC. |
| 0.2 | 10/12/21 | Daniel González Sánchez (UPM), Jose Antonio Ordonez Lucena (TID) | Updates to Section 3 & 6 |
| 0.3 | 17/01/22 | Daniel González Sánchez (UPM), Daniel Camps Mur (i2CAT), Lorena Chinchilla Romero (UGR), Ignacio Dominguez Martinez-Casanueva (TID), Carlos Colman Meixner (UNIVBRIS), Meysam Goodarzi (IHP) | Content added to Section 2 & 5, Updates to Section 3 |
| 0.4 | 16/02/22 | Daniel Camps Mur (i2CAT), Meysam Goodarzi (IHP), Jordi Perez Romero (UPC), Carlos Colman Meixner (UNIVBRIS), Ignacio Dominguez Martinez-Casanueva (TID), Daniel González Sánchez (UPM), Jose Antonio Ordonez Lucena (TID), Lorena Chinchilla Romero (UGR), Jonathan Prados Garzón (UGR), | Text added to Section 2.1, Section 3.1.1 added, Content added to 4.2, 4.3, 4.4 and 4.5, Updates to Section 5 and additional Sections 5.2.3 and Section 5.3, Content added to Section 6.1 |
| 0.5 | 17/03/22 | Daniel Camps Mur (i2CAT), Juan José Ramos Muñoz (UGR), Tezcan Cogalan (IDCC), Lorena Chinchilla Romero (UGR), Jordi Perez Romero (UPC), Jonathan Prados Garzón (UGR), Pablo Muñoz Luengo (UGR), Meysam Goodarzi (IHP), Ignacio Dominguez Martinez-Casanueva (TID), Daniel González Sánchez (UPM), Jose Antonio Ordonez Lucena (TID), Joseph McNamara (LMI), | New Figures added to Section 2, Content added to Section 2.3, Content added Section 3.2 and Section 3.3, Updates to Section 4.5.1, New content for Section 4.5.2, New content Section 4.6.1 and Section 4.6.2, Table added to Section 4.6.3, Update to Section 5.2.3, Content added to Section 6.2.3. |
| 0.6 | 08/04/22 | Lorena Chinchilla Romero (UGR), Meysam Goodarzi (IHP), Jonathan Prados Garzón (UGR), Joseph McNamara (LMI), Daniel Camps Mur (i2CAT), Carlos Colman Meixner (UNIVBRIS), Jose Antonio Ordonez Lucena (TID), Anil Yesilkaya (USTRATH), Tezcan Cogalan (IDCC), Daniel González Sánchez (UPM), Ignacio Dominguez Martinez-Casanueva (TID), | Content added to Section 3.1.1, Figures added to Section 3.1.2, output added to 3.1.4, Content added 3.2.2 and 3.2.3, Introduction added to Section 4, Content added 4.1.1 and Section 4.1.2, Content and results added 4.5.2, Content added to Section 4.6 and Section 4.7, Section 5.3.2 added, Content added to Section 6.2.1 and Section 6.2.2, Structure outlined for Section 6.3 |
| 0.7 | 29/04/22 | Meysam Goodarzi (IHP), Daniel González Sánchez (UPM), Joseph McNamara (LMI), Ignacio Dominguez Martinez-Casanueva (TID), Anil Yesilkaya (USTRATH), Carlos Colman Meixner (UNIVBRIS), Daniel Camps Mur (i2CAT), | Content added Section 1, Comments on Section 2, Content added 3.2.2, Section 3 reviewed and comments provided, Updates to Section 4 Introduction, Section 6 reviewed and comments provided. |
| 0.8 | 18/05/22 | Daniel Camps Mur (i2CAT), Tezcan Cogalan (IDCC), | Section 2 complete and reviewed |
| 0.81 | 18/05/22 | Meysam Goodarzi (IHP), Tezcan Cogalan (IDCC), Joseph McNamara (LMI), | Section 5 complete and reviewed |
| 0.82 | 04/06/22 | Daniel González Sánchez (UPM), Daniel Camps Mur (i2CAT), Tezcan Cogalan (IDCC), Meysam Goodarzi (IHP), Juan José Ramos Muñoz (UGR), Shuangyi Yan (UNIVBRIS), Ignacio Dominguez Martinez-Casanueva (TID), | Section 3 complete and reviewed |
| 0.83 | 04/06/22 | Meysam Goodarzi (IHP), Ardimas Purwita (USTRATH), Carlos Colman Meixner (UNIVBRIS), Jonathan Prados Garzón (UGR), Tezcan Cogalan (IDCC), Jordi Perez Romero (UPC), Lorena Chinchilla Romero (UGR), Daniel Camps Mur (i2CAT), Haiyuan Li (UNIVBRIS), Xueqing Zhou (UNIVBRIS), Shuangyi Yan (UNIVBRIS), Hilary | Section 4 complete and reviewed |

| | | | |
|---|---|---|---|
| | | Frank (UNIVBRIS), | |
| 0.84 | 15/06/22 | Joseph McNamara (LMI), Daniel Camps Mur (i2CAT), | Section 1 complete and reviewed |
| 0.85 | 15/06/22 | Joseph McNamara (LMI), Daniel Camps Mur (i2CAT), | Section 7 complete and reviewed |
| 0.86 | 25/06/22 | Carlos Colman Meixner (UNIVBRIS), Joseph McNamara (LMI), Jose Antonio Ordonez Lucena (TID), Daniel Camps Mur (i2CAT), | Section 6 complete and reviewed |
| 0.9 | 28/06/22 | Joseph McNamara (LMI) | Review of Figures, Tables and ToC |
| 1.0 | 30/06/22 | Jesús Gutiérrez (IHP), Mir Ghoraishi (GIGASYS) | Style confirmation and submission |
| 1.1 | 30/09/22 | ALL | Revision of the document according to the Review Report comments |
| 2.0 | 26/10/22 | Jesús Gutiérrez (IHP), Mir Ghoraishi (GIGASYS) | Style confirmation, final revision and submission |

# Table of Contents

# List of Figures

# List of Tables

# List of Acronyms

| | |
|---|---|
| 3GPP | 3rd Generation Partnership Project |
| 5G | Fifth Generation |
| 5GNR | 5G New Radio |
| 5G-PPP | 5G Infrastructure Public Private Partnership |
| AEF | API Exposing Function |
| AGV | Automated Guided Vehicle |
| AI | Artificial Intelligence |
| AMBR | Aggregate Maximum Bit Rate |
| AMD | Advanced Micro Devices |
| AMF | API Management Function |
| AP | Access Point |
| APF | API Publishing Function |
| API | Application Programming Interface |
| APN | Access Point Name |
| ATS | Asynchronous Traffic Scheduler |
| ATSSS | Access Traffic Steering Switching & Splitting |
| AWS | Amazon Web Services |
| BLER | Block Error Rate |
| CAPIF | Common API Framework |
| CDF | Cumulative Density Function |
| CIR | Channel Impulse Response |
| CLI | Command Line Interface |
| CPE | Customer Premises Equipment |
| CPU | Central Processing Unit |
| CQI | Channel Quality Indicator |
| CR | Compute Requirements Model |
| CSP | Communication Service Provider |
| CU-CP/UP | Centralized Unit Control Plane and User Plane |
| CUOM | Centralized Unit Orchestration and Management |
| DDA | Delay/jitter Distribution Agent |
| DLT | Distributed Ledger Technology |
| DNN | Deep Neural Network |
| DP | Data Plane |
| DP&M | Data Processing and Management |
| DQN | Deep Q-Network |
| DRL | Deep Reinforcement Learning |
| DSF | Data Semantic Fabric |
| DU | Distributed Unit |
| E2E | End-to-End |
| eMBB | enhanced Mobile Broadband |
| EPC | Evolved Packet Core |
| ETL | Extract Transform Load |
| ETSI | European Telecommunications Standards Institute |
| EuCNC | European Conference on Networks and Communications |

| | |
|---|---|
| FPS | Frames Per Second |
| FQDN | Fully Qualified Domain Name |
| GB | Gigabyte |
| GNB | gNodeB |
| gNMI | gRPC Network Management Interface |
| GSM | Global System for Mobile Communications |
| GST | Generic Slice Templates |
| GUI | Graphical User Interface |
| HTTP | Hypertext Transfer Protocol |
| ICMP | Internet Control Message Protocol |
| IEEE | Institute of Electrical and Electronics Engineers |
| IETF | Internet Engineering Task Force |
| IMSI | International Mobile Subscriber Identity |
| IP | Internet Protocol |
| IS | Intelligence Stratum |
| JSON | JavaScript Object Notation |
| KPI | Key Performance Indicator |
| LiFi | Light Fidelity |
| LOS | Line Of Sight |
| LSTM | Long short-term memory |
| LTE | Long-Term Evolution |
| LWIP | Lightweight IP |
| MAC | Media Access Control |
| MANO | Management and Orchestration |
| MARL | Multi Agent Reinforcement Learning |
| MCSs | Modulation Coding Schemes |
| MDP | Markov Decision Process |
| MEC | Multi-access Edge Computing |
| MF | Management Function |
| ML | Machine Learning |
| MNO | Mobile Network Operator |
| MOS | Management and Orchestration Stratum |
| MPQUIC | Multipath QUIC |
| MPTCP | MultiPath TCP |
| MSS | Maximum Segment Size |
| MTU | Maximum Transmission Unit |
| NEST | NEtwork Slice Type |
| NETCONF | Network Configuration Protocol |
| NFV | Network Functions Virtualization |
| NFVI | Network Function Virtualization Instance |
| NFVIaaS | NFVI as a Service |
| NFVO | NFV Orchestrator |
| NGSI-LD | Next Generation Service Interfaces - Linked Data |
| NS | Network Service |
| NS3 | Network Simulator 3 |
| NSD | Network Service Descriptor |

| | |
|---|---|
| NSMF | Network Slice Management Function |
| NSSO | Network Slice and Service Orchestration |
| ORAN | Open Radio Access Network |
| OSM | Open Source MANO |
| PCP | Priority Code Point |
| PERs | Packet Error Ratios |
| PLMN | Public Land Mobile Network |
| PoC | Proof of Concept |
| PRB | Physical Resource Block |
| QCI | QoS Class Identifier |
| QoS | Quality of Service |
| RAM | Random-Access Memory |
| RAN | Radio Access Network |
| RBAC | Role Based Access Control |
| REST | Representational State Transfer |
| RFC | Request for Comments |
| RIC | RAN Intelligent Controller |
| RL | Reinforcement Learning |
| RNS | Radio Node Selection Model |
| RSI | RAN Slice Instance |
| RSRP | Reference Signal Received Power |
| RSRQ | Reference Signal Received Quality |
| RSSI | Received Signal Strength Indicator |
| RT | Real Time |
| RTT | Round Trip Time |
| RU | Radio Unit |
| SAC | Soft Actor-Critic |
| SBMA | Service Based Management Architecture |
| SCW | Slice Creation Workflow Model |
| SDR | Software-Defined Radio |
| SINR | signal-to-interference-noise ratio |
| SLA | Service Level Agreement |
| SlaaS | Slice as a Service |
| SNPN | Standalone Non-Public Network |
| SNSSAI | Single Network Slice Selection Assistance Information |
| SQS | Slice QoS Model |
| SSID | Service Set IDentifier |
| TC | Traffic Class |
| TCP | Transmission Control Protocol |
| TMF | Tele Management Forum |
| TN | Transport Network |
| TN-C | TN Controller |
| TSN | Time-Sensitive Networking |
| UE | User Equipment |
| UPF | User Plane Function |
| URI | Uniform Resource Identifier |

| | |
|---|---|
| URL | Uniform Resource Locator |
| URLLC | Ultra-Reliable Low Latency Communication |
| USRP | Universal Software Radio Peripheral |
| UTF | Unicode Transformation Format |
| VIM | Virtualized Infrastructure Manager |
| VLAN | Virtual LAN |
| VRF | Virtual Routing and Forwarding |
| VSB | Vertical Service Blueprints |
| VSMF | Vertical Service Management Function |
| WAT | Wireless Access Technology |
| Wi-Fi | Wireless Fidelity |
| XML | Extensible Markup Language |
| YANG | Yet Another Next Generation |
| ZSM | Zero-Touch Service Management |

## Executive Summary

This document, 5G-CLARITY D4.3, constitutes the third deliverable of "WP4: Management Plane" and provides the latest developments of WP4, showcasing how the WP objectives have been successfully accomplished. D4.3 describes the evaluation of two main aspects of WP4. These are the evaluation of the E2E 5G Infrastructure and Service Slices and the evaluation of the developed self-learning ML algorithms. This is achieved through the final implementation of the 5G-CLARITY Service and Slice Provisioning System with an experimental demonstration of intent-driven Slice as a Service capabilities. The implementation and integration of the data management and processing subsystems are validated through a Proof-of-Concept experimental scenario. The self-learning ML algorithms are validated through execution in several scenarios, providing a variety of network functionalities to the system. These two aspects are presented in an integrated experiment showcasing the coordination of ML models within the AI Engine, fed by data accessible through the Data Lake and the results exposed through communication with the Intent Engine.

The ML models used in the system retrieve data for their algorithms from the 5G-CLARITY data management and processing subsystem. The system is broken into three parts. These describe the integration of a variety of telemetry data in the data lake, the collection of transport data within the data semantic fabric and the integration of the data lake and data semantic fabric. Each of these are presented and validated through relevant experimental scenarios. A collection of ML models is also implemented and presented in real-world scenarios. These models, previously described in 5G-CLARITY's D4.2 [1], provide a wide array of network functionalities. Each model is described highlighting the scenario and the impact of the model. The primary learning and conclusions of these experiences are also detailed in deliverable.

Private-public network integration is one of the main distinguished features of the 5G-CLARITY system. This feature represents the ability to make 5G-CLARITY capabilities interwork with MNO's managed capabilities seamlessly. This deliverable reports on relevant application scenarios related to enablers such as the Mediation Function and Service Delivery Models. The final solution design of the Mediation Function is showcased in a use-case based approach highlighting the applicability in a private-public network environment. Service Delivery Models are presented in two scenarios, these are NFVI as a Service and Slice as a Service. These scenarios are presented through two distinct use cases. These include the instantiation of a NFVI in a Smart Internet Lab environment and the provisioning of a network slice through an intent-based interface informed by smart models.

# 1 Introduction

This deliverable presents the evaluation of E2E 5G Infrastructure and Service Slices, and of the Developed Self-Learning ML Algorithms. It expands on the implementation and validation of several systems detailed in 5G-CLARITY D4.2 [1]. These include the final implementation of the 5G-CLARITY Service and Slice Provisioning System with an experimental demonstration of intent-driven Slice as a Service capabilities. The implementation and integration of the data management and processing subsystems validated through a Proof-of-Concept experimental scenario. The extension and validation of predefined Machine Learning Algorithms through several scenarios, providing a variety of network functionalities to the system. The evaluation of an integrated experiment showcasing the coordination of models within the AI Engine, the querying of data from the Data Lake and the triggering and exposer of model results through the Intent Engine. Finally, the 5G-CLARITY Mediation Function is presented along with two experimental demonstrations of 5G-CLARITY Service Delivery Models for Network Function Virtualisation Instances and Slice provisioning.

## 1.1 Objective and scope of this document

5G-CLARITY D4.3 is the third deliverable of Work Package 4: Management Plane. This document details the evaluation of systems and models described in previous deliverable D4.2 [1]. The aim of the document is to provide an evaluation of end-to-end 5G infrastructure, service slices and developed self-learning ML models.

This specific objective of this deliverable are as follows:

OBJ-1 The final implementation of the Service and Slice provisioning Subsystem.

OBJ-2 The Data Lake and Data Semantic Fabric implementation.

OBJ-3 Presenting the ML models in real world scenarios.

OBJ-4 Present the experimental evaluation of Intelligence Stratum, Data Lake and Indoor and non-LOS identification.

OBJ-5 Present the private-public network integration through experimental demonstration.

## 1.2 Document Structure

This document is organised as follows:

Section 2 describes the final implementation of the 5G-CLARITY Service and Slice Provisioning Subsystem and demonstrates the management system utilising both public and private network resources.

Section 3 describes the data lake and data semantics fabric implementation of the 5G-CLARITY data management and processing subsystem.

Section 4 expands on the ML algorithms described in 5G-CLARITY D4.2 [1] presenting Reinforcement Learning models and their application in real-world scenarios.

Section 5 details the integration of the NLoS identification algorithm into the AI Engine and the triggering of the model through the Intent Engine interface.

Section 6 presents the mediation function, experimental demonstrations of service delivery models and various test scenarios to validate different AI engines data and offloading tasks.

Section 7 presents the conclusion, highlighting the contributions of the document

## 1.3 On the fulfilment of 5G-CLARITY management plane requirements and KPIs

In this section we address the requirements and KPIs of the Management and Orchestration Stratum (MOS) and Intelligence Stratum (IS). These requirements and KPIs, first identified in D2.2 [2], are discussed in previous deliverable [1] with reference to how they are validated. This approach is repeated with respect to this deliverable, showing the contribution towards requirements and KPIs. The Management and Orchestration Stratum requirements and KPIs are shown in Table 1-1 and the Intelligence Stratum requirements and KPIs are shown in Table 1-2.

**Table 1-1. Management and Orchestration Stratum - Functional Requirements and KPIs**

| Requirement ID | Requirement Description | Component | Means of Verification [D4.3 section] |
|---|---|---|---|
| **CLARITY-MOS-R1** | The 5G-CLARITY management and orchestration stratum shall be architected following the Service Based Management Architecture (SBMA) principles, with a set of MFs providing/consuming management services through a service bus. | ALL | A design adhering to the SBMA principles was already presented in D2.2. In D4.3 we present the final evaluation of the architecture introduced in D2.2. |
| **CLARITY-MOS-R2** | The 5G-CLARITY management and orchestration stratum shall allow for the provisioning of 5G-CLARITY resource-facing services (i.e., 5G-CLARITY wireless, compute and transport services). | Service and Slice Provisioning subsystem | Transport services are described in Section 2.2 |
| **CLARITY-MOS-R3** | The 5G-CLARITY management and orchestration stratum shall keep a resource inventory, with information on the on-premises resources that can be used for the provision of 5G-CLARITY resource-facing services. This includes information on: *i)* the resource capacity of deployed wireless access nodes, including Wi-Fi/LiFi APs and physical gNBs; *ii)* the compute nodes available in the clustered NFVI (RAN cluster and edge cluster), and related computing/storage/networking resources; *iii)* the capacity and topology of deployed transport network. | Slice Manager, multi-WAT non-RT RIC | The telemetry subsystem demonstrated in Section 3 integrates data sources from wireless and transport nodes. This telemetry enables to keep track of resources available in each domain. In the compute domain resource-based telemetry has not been explicitly shown, as this is a standard feature available in OSM. |
| **CLARITY-MOS-R4** | The 5G-CLARITY management and orchestration stratum shall store a catalog of VxFs/NSDs. | NFVO | Use of OSM 11 [Section 2.2] |
| **CLARITY-MOS-R5** | The 5G-CLARITY management and orchestration stratum shall support to create, retrieve, update and delete VxFDs/NSDs | NFVO | Use of OSM 11 [Section 2.2] |
| **CLARITY-MOS-R6** | The 5G-CLARITY management and orchestration stratum shall allow to create several instances of the same VxF/NFV service. | NFVO | Use of OSM 11 [Section 2.2] |
| **CLARITY-MOS-R7** | The 5G-CLARITY management and orchestration stratum shall allow VxF / NFV service scaling. This scaling includes the scaling-in and scaling-out the resources of deployed VxF / NFV service instances. | NFVO | Use of OSM 11 [Section 2.2] |

| | | | |
|---|---|---|---|
| **CLARITY-MOS-R8** | The 5G-CLARITY management and orchestration stratum shall allow for the provisioning of 5G-CLARITY slices, by defining separate resource quotas when allocating individual 5G-CLARITY resource-facing services. | Slice Manager | Slice provision is described in both Section 2.2 and Section 2.3 |
| **CLARITY-MOS-R9** | The 5G-CLARITY management and orchestration stratum shall maintain information regarding the mapping between 5G-CLARITY slices, constituent 5G-CLARITY resource-facing services and allocated resources. | Slice Manager | Compute resources are allocated in Section 2.1 |
| **CLARITY-MOS-R10** | The 5G-CLARITY management and orchestration stratum shall allow resource elasticity and AI-assisted placement optimization as part of the 5G-CLARITY slice lifecycle management. | Slice Manager | Section 4.2 describes RAN slicing for multi-tenant<br>Section 4.3 describes the optimal access simulation extension |
| **CLARITY-MOS-R11** | The 5G-CLARITY management and orchestration stratum shall provide means for model-based data aggregation, with the ability to collect and process management data (e.g., performance measurements, fault alarms) from different sources in an automated and scalable manner. | Near-RT RIC | Interfaces are described in Section 3.1 |
| | | Data Processing and Management Subsystem | Section 3.2 shows the processing of data in Data Semantic Fabric |
| **CLARITY-MOS-R12** | The 5G-CLARITY management and orchestration stratum shall be able to correlate aggregated data with deployed 5G-CLARITY slices and services instances, providing input to the intelligence engine for AI assisted operation of these instances. | Data Processing and Management Subsystem | The use of AI module to provide prediction based on network data shown in Section 5 |
| **CLARITY-MOS-R13** | The 5G-CLARITY management and orchestration stratum shall provide necessary cloud-native capabilities for MF service production/consumption across the entire stratum. | Cloud Native Support Subsystem | The telemetry subsystem described in Section 3 (e.g. Data Lake in AWS) and the AI engine (OpenFaaS) in Section 5 are developed using cloud native principles. Other subsystems, like the service and slice provisioning in Section 2, could be implemented using cloud native approach (e.g. K8s), but we have not done so to keep the implementation simple. |
| **CLARITY-MOS-R14** | The 5G-CLARITY management and orchestration stratum shall allow individual 5G-CLARITY customers (e.g. MNOs) to securely access and consume MF services. | Mediation Function | Section 6.1.2 described CAPIF which addresses authentication functionality |
| **CLARITY-MOS-R15** | The 5G-CLARITY management and orchestration stratum shall provide the means to expose capabilities with appropriate abstraction levels to individual 5G-CLARITY customers | Mediation Function | Section 6.1.1 and Section 6.1.2 describe the API for customers |

| | | | |
|---|---|---|---|
| **CLARITY-MOS-R16** | The 5G-CLARITY management and orchestration stratum shall provide isolation among customers' workflows and request | Mediation Function | Section 6.1.3 shows the workflows for triggered requests |
| **KPI** | **KPI description** | **Component** | **Means of verification [D4.3 section]** |
| **CLARITY-MOS-KPI1** | According to OBJ-TECH-6, the 5G-CLARITY management and orchestration stratum shall provision a service less than 5 minutes, while providing security and isolation to infrastructure and service slices. | Service and Slice Provisioning Subsystem | Section 2.2 |
| **CLARITY-MOS-KPI2** | According to OBJ-TECH-7, the 5G-CLARITY management and orchestration stratum shall provision an E2E 5G slice integrating compute and transport resources of an MNO in less than 10 minutes | Mediation Function, Slice Manager | Section 2.3 |

**Table 1-2. Intelligence Stratum - Functional Requirements and KPIs**

| Requirement ID | Description | Component | Means of verification [D4.3 section] |
|---|---|---|---|
| **CLARITY-INTS-R1** | The 5G-CLARITY intelligence stratum shall leverage machine learning (ML) models to support intelligent management of network functions. | AI Engine | Section 5 describes the creation and execution of an ML model |
| **CLARITY-INTS-R2** | The 5G-CLARITY intelligence stratum shall host ML models and offer them as services that are accessible outside of the intelligence stratum. Consumers of the ML services are either the network operator or other network functions. | AI Engine& Intent Engine | Section 6.3 describes an AI Engine use case outside of the Intelligence Stratum |
| **CLARITY-INTS-R3** | The 5G-CLARITY intelligence stratum shall provide a point of access for ML services to consume data from the network and forward recommended configurations to suitable network functions. | AI Engine & Intent Engine | Section 6.2.2 shows the coordination of ML models in AI Engine and the Intent Engine to dynamically create slices based on network information |
| **CLARITY-INTS-R4** | The 5G-CLARITY intelligence stratum shall provide ML designers a process or interface to manage the lifecycle of ML models, including the deployment as services. | AI Engine | Section 5 details the complete ML designer experience from creation to deployment to execution of ML Models in AI Engine |
| **CLARITY-INTS-R5** | The 5G-CLARITY intelligence stratum shall expose a communication interface towards the end user that simplifies the management of the 5G-CLARITY platform using intents, including intent-based network configuration and intent-based usage of available ML services. | Intent Engine | The interface of Intent Engine is described in Section 5 with example of an intent message |
| **CLARITY-INTS-R6** | The 5G-CLARITY intelligence stratum shall expose an intent management interface through which the intent lifecycle can be controlled, including creation and removal. | Intent Engine | The interface of Intent Engine is described in Section 5 with examples of successful response |
| **KPI** | **KPI description** | **Component** | **Means of Verification [D4.3 section]** |

| | | | |
|---|---|---|---|
| **CLARITY-INT-KPI1** | According to OBJ-TECH-8, the 5G-CLARITY intelligence stratum shall demonstrate how the AI engine can reduce both manual and semi-automated intervention in at least 2 relevant use cases. | AI Engine and Intent Engine | These use cases are described in Section 6.2.1 and Section 6.2.2 |

# 2 Service and Slice Provisioning Subsystem

This section describes the final implementation of the 5G-CLARITY service and slice provisioning subsystem. The section is organized in the following four subsections:

Section 2.1 provides a high-level overview of all the modules composing the service and slice provisioning subsystem and the extensions carried out within 5G-CLARITY.

Section 2.2 describes the integration of the Accelleran ORAN-based 5G small cell into the 5G-CLARITY service and slice provisioning subsystem through the implementation of a new management function that we call CUOM (Centralized Unit Orchestration and Management). Recall that the integration of Wi-Fi and LiFi technologies was already described in D3.2 [3].

Section 2.3 contains a benchmarking of the 5G-CLARITY slice provisioning time using the private network infrastructure, demonstrating that private network slices can be provisioned in less than 5 minutes. This section addresses 5G-CLARITY OBJ-TECH-6

Section 2.4 describes how the service and slice provisioning subsystem developed in 5G-CLARITY can be integrated with the management system of a public network to deliver end-to-end slices. This section also benchmarks the overall end-to-end slice provisioning time, demonstrating that end-to-end network slices can be provisioned in less than 10 minutes, which addresses 5G-CLARITY OBJ-TECH-7.

## 2.1 Overview of required implementation and integrations

Figure 2-1 describes the architecture of the 5G-CLARITY service and slice provisioning subsystem, where we identify several modules that need to work together to manage the lifecycle of 5G-CLARITY infrastructure slices.

This section reports the final implementation of the 5G-CLARITY service and slice provisioning subsystem that will be used in the pilots demonstrated in WP5. Developing this subsystem required the use of open-source modules available in the state of the art, along with other background assets provided by partners that have been extended in the project, as well as other modules that have been developed from scratch. Table 2-1 provides a detailed overview of all the involved modules and highlights the module integrations that are experimentally validated through the experiments described in this section.

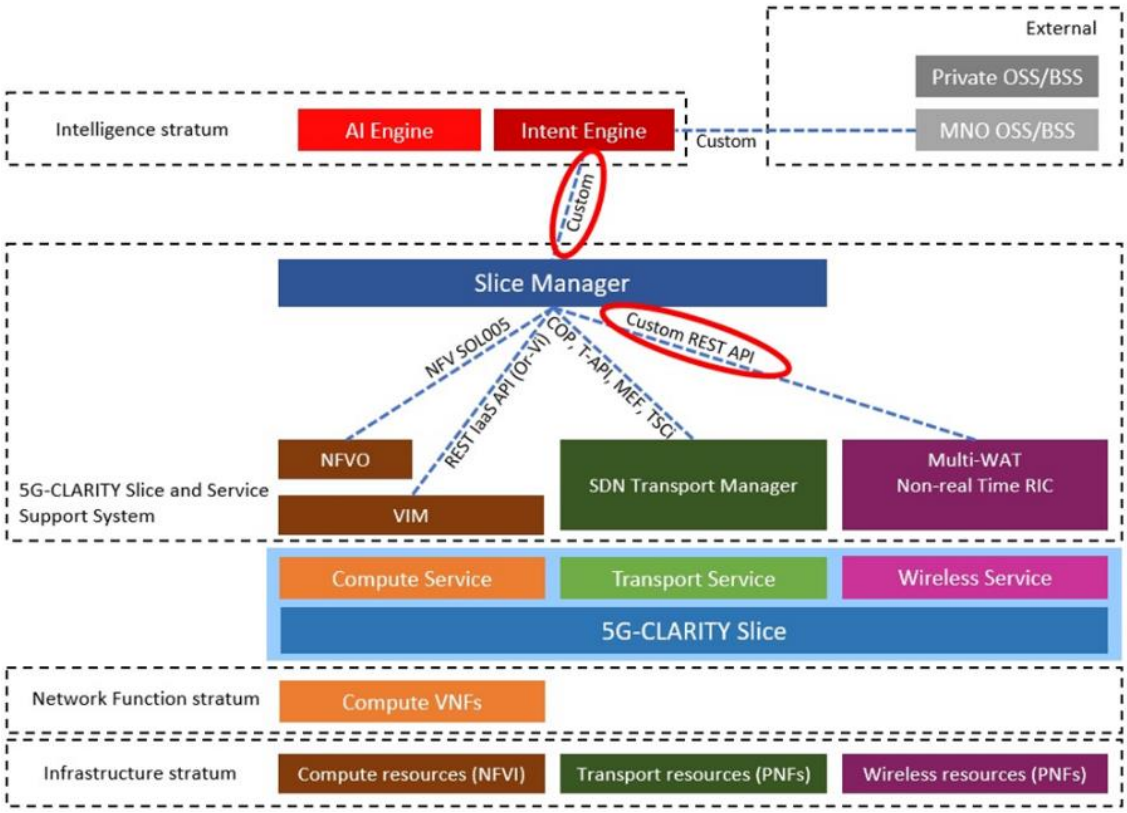



**Figure 2-1. 5G-CLARITY service and slice provisioning subsystem: highlighted in red are customized interfaces developed within the scope of 5G-CLARITY**

**Table 2-1. Overview of modules composing the 5G-CLARITY service and slice provisioning subsystem**

| Module | Background | Extensions in 5G-CLARITY | Responsible partner | Module integrations validated in this section |
|---|---|---|---|---|
| NFVO | OSM 10.0.2 (opensource project [4]) | None | I2CAT | Slice Manager |
| VIM | Open Stack Victoria (opensource project [5] | None | I2CAT | Slice Manager |
| Slice Manager | Initial implementation from 5GCity project [6] | Extension to expose 5GNR and LiFi services<br>Extension to modify dynamically compute resource assigned to a chunk | I2CAT | Multi-WAT non-RT RIC |
| Multi-WAT non RT RIC | Initial implementation from 5GCity project [6] | Southbound clients to manage LiFi and ORAN 5GNR gNB | I2CAT | NETCONF servers in all physical wireless functions |
| NETCONF server in Wi-Fi AP | Initial implementation from 5G-PICTURE project [7] | Extended to support WiFi6 | I2CAT | Multi-WAT Non-RT RIC |
| NETCONF server in LiFi AP | N/A | Built from scratch | PLF | Multi-WAT Non-RT RIC |
| NETCONF server in ORAN 5GNR gNB | Baseline product from ACC | Extended with support for MOCN | ACC | Multi-WAT Non-RT RIC<br>Open5gs (5GC) |

A video demonstrating the integration of the previous software modules to provision 5G-CLARITY infrastructure slices is available in [8].

## 2.2 Integration of ACC 5GNR with the 5G-CLARITY service and slice provisioning subsystem

Deliverable D4.2 [1] describes how the 5G-CLARITY service and slice provisioning subsystem can be used to manage wireless physical and virtual network functions. Figure describes the architecture of the 5G-CLARITY service and slice provisioning subsystem where the Multi-WAT Non-real Time RIC is the entity in charge of managing wireless devices using NETCONF/YANG [9] [10]. A detailed description of how NETCONF/YANG was used to control 5G-CLARITY Wi-Fi and Li-Fi devices was provided in D4.2 [1].

In this deliverable, we describe the approach that has been followed to integrate the Accelleran 5GNR small cell devices into the 5G-CLARITY service and slice provisioning subsystem. A key feature of this technology is its disaggregated nature, aligned with the ORAN architecture, whereby the following physical and virtual network functions are considered:

- Physical functions: 3.5GHz Radio Unit (RU) with 40MHz carrier bandwidth, and bare metal compute server hosting the virtual network functions (VNFs)
- Virtual Network Functions instantiated in bare metal server:
    - Software-based Distributed Unit (DU) provided by Phluido[1]
    - Centralized Unit Control Plane and User Plane (CU-CP/UP) VNFs provided by Accelleran
    - Near-real Time RAN Intelligent Controller provided by Accelleran's dRAX

The integration approach used in 5G-CLARITY consists in integrating the CU-CP and CU-UP components with the multi-WAT Non-real time RIC using NETCONF/YANG, while it was not possible to integrate the DU and RU due to NETCONF support not being available in the DU and RU vendors. This implementation presents certain limitations as low-level radio parameters cannot be controlled by the 5G-CLARITY management plane, which is considered as future work. However, the achieved implementation is sufficient to verify the two 5G-CLARITY slicing models presented in D4.2 [1], namely the "PLMNID-based slicing" and the "PLMNID+SNSSAI-based slicing", which is the main objective of the 5G-CLARITY service and slice provisioning subsystem.

Figure 2-2 contains a description of the YANG models of both the CU-CP (GNB-CU-CP) and the CU-UP (GNB-CU-UP) components. Recall that in the 5G RAN architecture, a single CU-CP function can be in charge of multiple CU-UPs. The following principles are applied:

- Each CU-UP function connects to the CU-CP through the E1 link. IP connectivity is required for that, and the IP address of the CU-CP can be provisioned in the CU-UP function using NETCONF/YANG. Thus, the 5G-CLARITY service and slice provisioning subsystem can for example have a common CU-CP function for all slices and deploy a dedicated CU-UP function for each infrastructure slice, allocating dedicated compute resources to the CU-UP function as described in 5G-CLARITY D4.2 [1]. The service and slice provisioning subsystem could deploy the CU-UP function using the NFVO component and configure the CU-CP IP address using the multi-WAT Non-real Time RIC component.
- The CU-CP function contains a list of operators. For each operator the NG-C link to its respective core network can be configured. This is required by the MOCN functionality used in 5G-CLARITY to

[1] https://www.phluido.net/

implement “PLMNID-based slicing” (c.f. 5G-CLARITY D4.2 [1]).

- Each CU-UP function contains a list of PLMN slices, i.e. PLMNID + S-NSSAI, being accessible through that CU-UP. Notice that the list of radiated PLMNID and SNASSI needs also be configured in the DU, which however needs to be configured in a static manner in our current implementation.

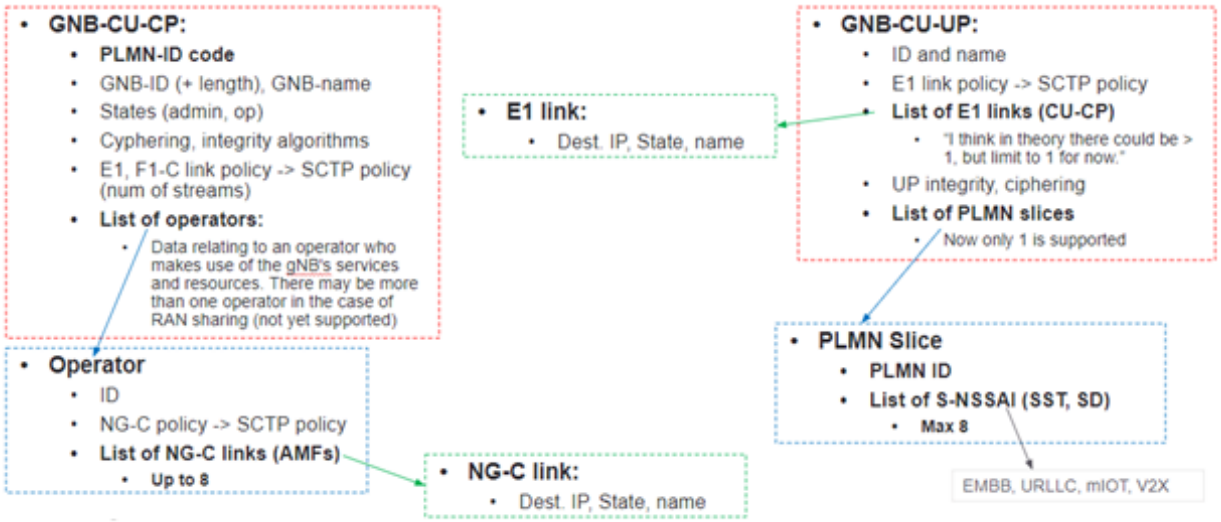



**Figure 2-2. ACC 5GNR YANG model**

Based on the previous description, Figure 2-3 provides an example deployment model showing how CU-UP and CU-CP functions can be mapped to 5G-CLARITY slices. In the figure, we can see three slices deployed in the edge cluster, one represented by the “pink” 5GC control plane corresponding to PLMNID 00103, giving access to data network DNN21. A second slice is represented by “green” and corresponds to PLMNID 00102 giving access to DNN1 and DNN2, and a third slice is represented by “blue” and corresponds also to PLMNID 00102 giving access to DNN12. Notice that DNN2, reachable from the “green” slice is coloured orange in Figure 2-3 to highlight that a separate compute chunk[2] is allocated for this slice, thus highlighting the 1:N mapping between 5G-CLARITY slices and compute chunks in the edge cluster (c.f. deliverable D4.2 [1] for additional details).

Focusing now on the RAN cluster, a corresponding slice instantiation in the RAN cluster could consist of a common CU-CP function, and a dedicated CU-UP for each of the three existing slices, where each CU-UP function would have guaranteed compute resources (compute chunk) in the RAN cluster. Notice though that whether a CU-UP is dedicated to a 5G-CLARITY slice or shared across multiple slices is implementation dependent and ultimately something that can be controlled through the 5G-CLARITY service and slice provisioning subsystem. For example, following the 5G-CLARITY architecture, one could imagine an implementation where the decisions about whether CU-UPs are dedicated or shared across slices is dynamic and is offloaded to a Machine Learning model deployed in the AI engine.

Finally, following the 5G-CLARITY model, each slice would only be reachable from a subset of DUs, depending how the PLMN and SNSSAI lists are configured in each DU.

[2] Meaning a dedicated OpenStack project with a set of CPU, RAM and storage dedicated resources

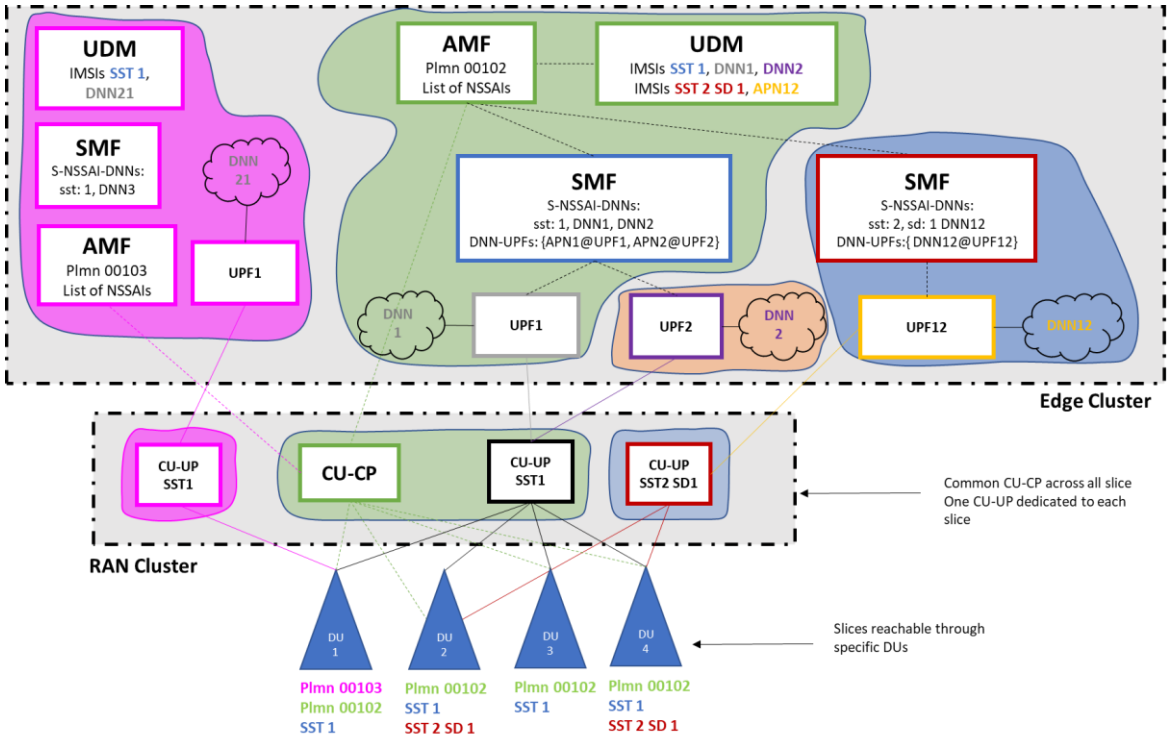



**Figure 2-3. Exemplary CU deployment in RAN cluster enabling 5G-CLARITY slicing**

To manage the complexity introduced by the disaggregated O-RAN architecture, i.e. the modelling of the CU related network functions, we have developed a new management entity in the 5G-CLARITY management plane that we call CUOM (CU Orchestration and Management) function. CUOM can be seen as logically belonging to the Multi-WAT Non-real Time RIC and is used to manage the relations between the CU-UP and CU-CP functions. CUOM is in charge of the following functions:

- CU-UP and CU-CP registration. Figure 2-4 depicts the workflow used to register CU-UP or CU-CP functions in CUOM, where CU-UP and CU-CP are assumed to be already deployed in the RAN cluster. Each virtual CU function is registered individually by providing the IP address of the NETCONF server representing the CU function, and in the case of the CU-UP providing the identifier of the CU-CP that it should be connected to. CUOM then proceeds to configure appropriately each CU function.

- DU registration. Individual DUs are also registered in CUOM specifying the CU-CP that is in charge of controlling that DU function. After DU registration CUOM can solve the potential relations between the CU-CP, CU-UP and DU functional element. The DU component is also registered in the core logic Multi-WAT Non real-Time RIC. Thus, the Multi-WAT Non rt-RIC treats DUs as if they were cells, which can for example be selected to be part of a 5G-CLARITY slice, and offloads all management operations involving the CU to CUOM.

- 5GNR service provisioning workflow. Depicted in Figure 2-5. Once a new 5G-CLARITY slice needs to be provisioned on a given DU, the Multi-WAT Non-real Time RIC triggers a slice provisioning on CUOM indicating the target AMF IP address, the PLMNID and S-NSSAI parameters and the involved DUs that will be part of the 5G-CLARITY slice (refer to 5G-CLARITY D4.2 [1] for an example of 5G-CLARITY slice definition). Given the DU, CUOM identifies the CU-CP function that needs to be configured for that service. Regarding the CU-UP, CUOM analyses the CU-UPs connected to the CU-CP and selects one to serve the requested PLMNID+SNSSAI. In our implementation, CU-UPs are always exclusively allocated to a single slice if possible, but other policies could be considered.

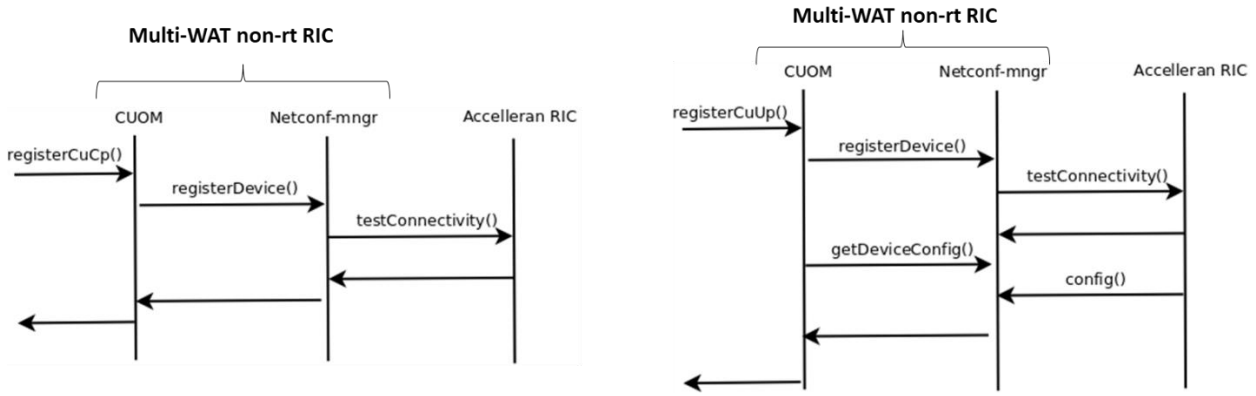



**Figure 2-4. dRAX CU-CP and CU-UP registration workflows with multi-WAT non-rt RIC**

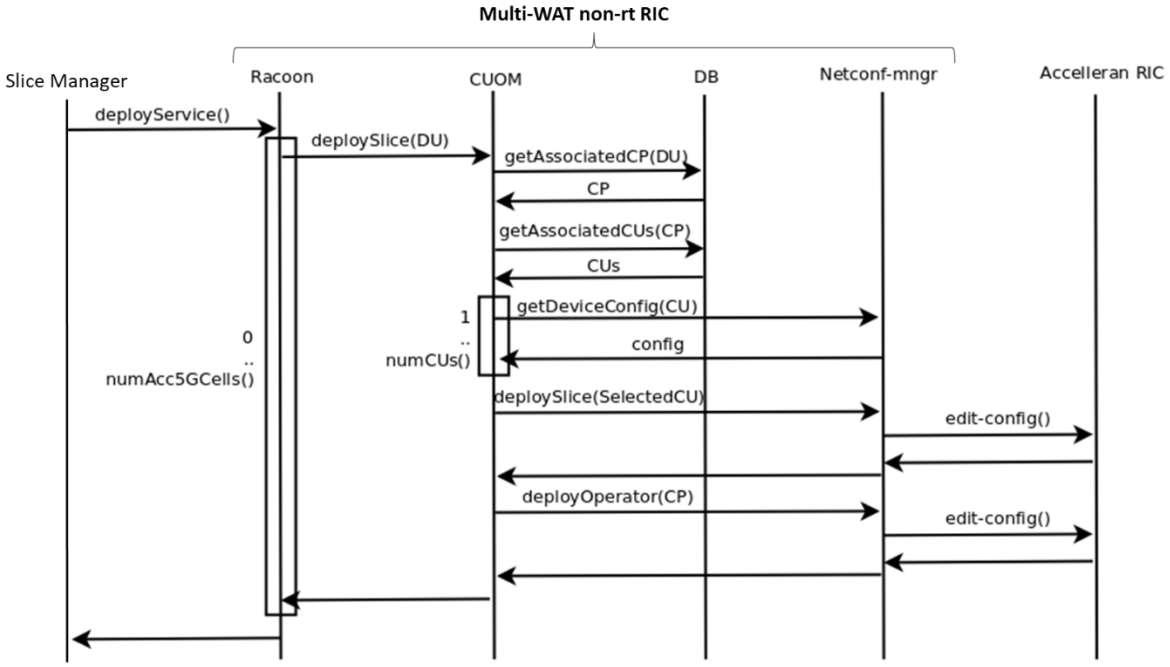



**Figure 2-5. 5GNR service provisioning workflow**

Next, we show some functional evidence about the configuration of an 5GNR service using the multi-WAT non-rt RIC and CUOM.

- Figure 2-6 depicts the configuration of the CU-CP and CU-UP modules provisioned in the dRAX server before instantiating any 5GNR service (c.f. Figure 2-5). Notice that a dummy operator "operator-1" and plmnid "99999" values are configured in the CU-CP and CU-UP.
- Figure 2-7 depicts the JSON body included in a 5GNR service request issued from the Slice Manager to the Multi-WAT non-rt RIC. The body includes the following fields:
  - "selectedPhys" field indicates the cells that need to be part of the 5G-CLARITY slice
  - "vlanId" indicates the transport service used for this slice
  - "cellularConfig" indicates the cellular parameters related to this slice assuming a PLMNID-based slicing model, namely:
  - "plmnId" field equal to a valid private plmnid, i.e. "00109"
  - "coreAddress" and "corePort" fields equal to the IP address in the 5G-CLARITY edge cluster where the 5G Core deployed for this slice is reachable.

- Figure 2-8 depicts the CU-CP and CU-UP internal configuration after the Multi-WAT non-rt RIC has completed the 5GNR service configuration, where a new "operator" and "plmn-slice" element corresponding to the configuration requested by the Slice Manager are added to the respective lists in the CU-CP and CU-UP. Notice that in our current implementation, we only support PLMNID-based slicing and so no information on the S-NSSAI is added to the request. Therefore, CUOM assumes by default an eMBB slice type with SST=1 and default value for SSD.
- Figure 2-9 shows evidence in the dRAX dashboard of the new PLMNID being configured in the radios.

CU-CP

```
...
    "operator-list": [
      {
        "gnb-cu-cp": {
          "operator-id": "operator-1",
          "ng-c-link-policy": {
            "sctp-policy": {
              "in-streams": 1,
              "out-streams": 1
            }
          },
          "ng-c-link-list": [
            {
              "ng-c-link": {
                "dest-address": "192.168.50.158"
              }
            }
          ]
        }
      }
    ]
...
```

CU-UP

```
...
    "supported-plmn-slice-list": [
      {
        "supported-plmn-slice": {
          "plmn-id": "99999",
          "s-nssai-list": [
            {
              "S-NSSAI": {
                "sst": "embb",
                "sd": 16777215
              }
            }
          ]
        }
```

**Figure 2-6. dRAX CU-UP and CU-CP configurations prior to 5GNR service configuration**

```
{
  "selectedPhys": [
    "09f30358-fed6-49dc-93d7-79fa0c2174c2"
  ],
  "vlanId": 100,
  "serviceConfig": {
    "cellularConfig": {
      "plmnId": "00109",
      "accelleranConfig": {
        "coreAddress": "192.168.100.25",
        "corePort": 3344
      }
    }
  }
}
```

**Figure 2-7. Slice manager request to Multi-WAT non-rt RIC for 5GNR service configuration**

**CU-CP**

```
...
    "operator-list": [
      {
        "gnb-cu-cp": {
          "operator-id": "192.168.100.25",
          "ng-c-link-policy": {
            "sctp-policy": {
              "in-streams": 1,
              "out-streams": 1
            }
          },
          "ng-c-link-list": [
            {
              "ng-c-link": {
                "dest-address": "192.168.100.25"
              }
            }
          ]
        }
      },
      {
        "gnb-cu-cp": {
          "operator-id": "operator-1",
          "ng-c-link-policy": {
            "sctp-policy": {
              "in-streams": 1,
              "out-streams": 1
            }
          },
          "ng-c-link-list": [
            {
              "ng-c-link": {
                "dest-address": "192.168.50.158"
              }
            }
          ]
        }
      }
    ]
...
```

**CU-UP**

```
...
    "supported-plmn-slice-list": [
      {
        "supported-plmn-slice": {
          "plmn-id": "99999",
          "s-nssai-list": [
            {
              "S-NSSAI": {
                "sst": "embb",
                "sd": 16777215
              }
            }
          ]
        }
      },
      {
        "supported-plmn-slice": {
          "plmn-id": "00109",
          "s-nssai-list": [
            {
              "S-NSSAI": {
                "sst": "embb",
                "sd": 16777215
              }
            }
          ]
        }
      }
    ]
...
```

**Figure 2-8. dRAX CU-UP and CU-CP configurations after 5GNR service configuration**

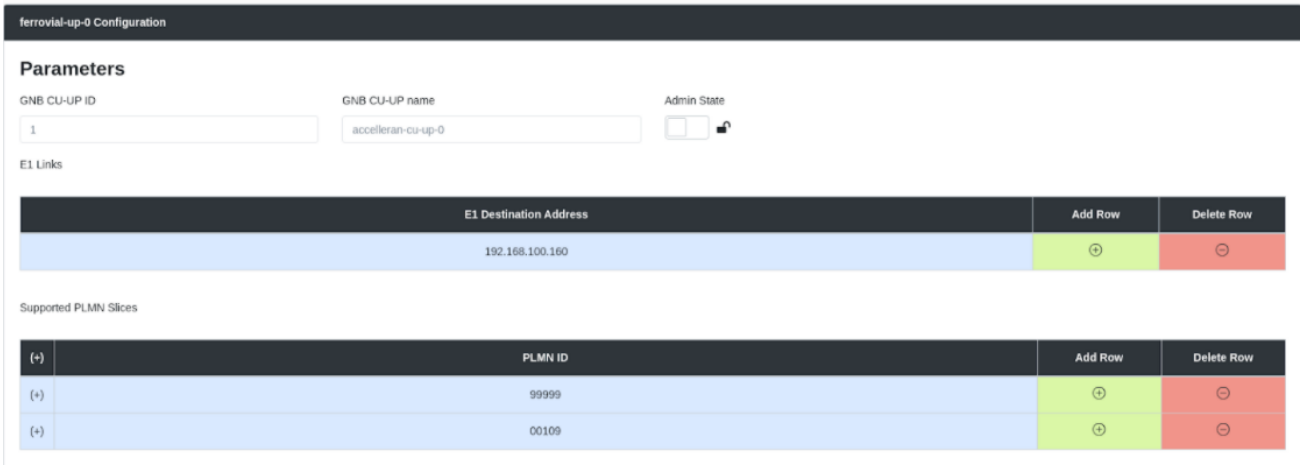



**Figure 2-9. dRAX dashboard indicating configured PLMNIDs**

In the next section we evaluate the time required to deploy a 5G-CLARITY slice including the three types of wireless access networks and all required virtual network functions considered in the project.

## 2.3 Private venue slice provisioning benchmarking

We evaluate in this section the time required to provision a 5G-CLARITY infrastructure slice. This KPI is aligned with 5G-CLARITY OBJ-TECH-6, which targets a slice provisioning time within the private venue of less than 5 minutes.

We note that this KPI was initially investigated in D4.2 [1], section 2.1.3. The evaluation presented therein considered a 4G radio access network and evaluated the wireless service provisioning times using each wireless technology separately. The benchmark provided in this section expands our previous work by: i) including an ORAN compliant 5GNR radio, which is aligned with the reference 5G-CLARITY architecture, and ii) by integrating the service provisioning of all wireless access networks in the Multi-WAT Non-real Time RIC, so that a single API end-point triggers the wireless service provisioning in all the access technologies.

Figure Figure 2-10 depicts the experimental testbed we have deployed at i2CAT to execute this benchmark. The testbed is composed of the following physical infrastructure:

- A 5G-CLARITY edge server, featuring OpenStack Victoria. The server supports the VNFs deployed as part of the instantiation process of the 5G-CLARITY slice, including a monolithic 5GSA core based on open5gs and an MPTCP proxy VNF acting as AT3S user plane function.
- A Pure LiFi-XC Access Point representing the LiFi access network.
- A custom Wi-Fi 6 Access Point, provided by i2CAT.
- The Accelleran's ORAN 5G radio, deployed in a separate vRAN server featuring the Near-real Time RIC (dRAX), the CU and the DU components. A USRP B210 radio is used as RU.

The management plane components of the testbed consist of:

- The Multi-WAT Non-real Time RIC, which is the management element used to configure all wireless access technologies.
- An NFVO based on OSM 11, used to instantiate network services.
- The Slice Manager component that orchestrates the lifecycle of the 5G-CLARITY slices.

To carry out our benchmark, we are interested in the overall slice provisioning time, which is the time from the moment the slice request is sent to the Slice Manager until all VNFs are up and running in the edge server, and all the wireless network functions are appropriately configured. This time is marked as T1 in Figure 2-10. In addition, we also measure the time required to configure only the wireless part, which is marked as T2 in Figure 2-10, where T2 is triggered by the Slice Manager interacting with the Multi-WAT Non real-time RIC to configure the wireless access networks. To collect our measurements, we run the configuration end-point 20 different times, and plot the results as empirical cumulative distribution functions (CDF).

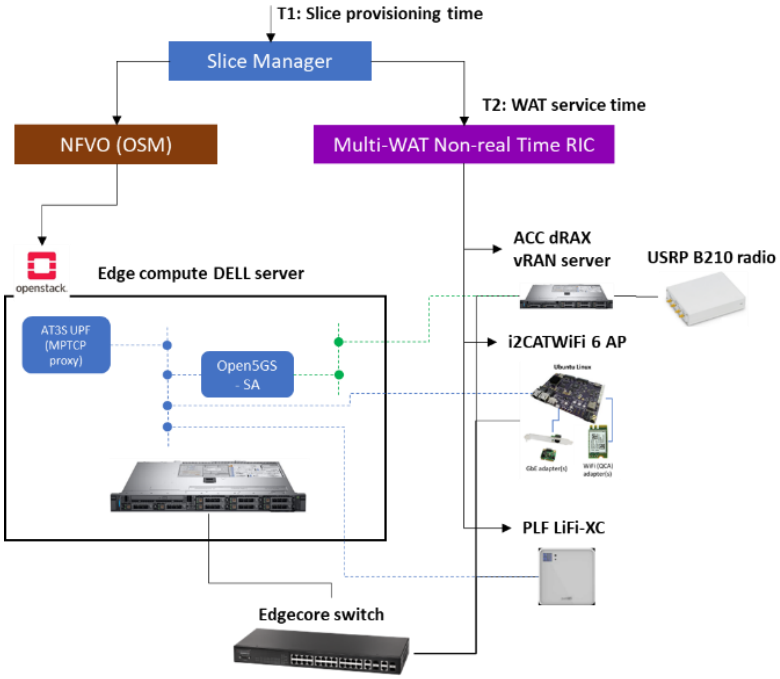



**Figure 2-10. I2CAT testbed to benchmark 5G-CLARITY slice provisioning times**

Figure 2-11 depicts the measured slice provisioning and deletion times (T1 in Figure 2-10) plotted as an empirical CDF. The slice provisioning times are broken up in their different component times, namely:

Slice_creation: Time required to create the data structure in the slice manager that includes the compute and radio chunks. This time is measured to be below 10 seconds.

Slice_activation: Time required to deploy a virtualised mobile core function (open5gs), a dhcp and to configure the radio service including Wi-Fi, Accelleran 5G and LiFi. This time is the largest taking between 35 and 40 seconds.

Service_instantiation: Time required to instantiate the NFV network services associated with the slice, which in the case of our network include the ATSSS proxy VNF. We note that the network service was previously onboarded to the NFVO. This time is measured to be around 37 seconds.

Based on the three previous measurements we conclude that the total time required to provision a 5G-CLARITY slice in the private venue is below 10+40+37=87 seconds, well below the project KPI of 5 minutes.

Figure 2-11 also depicts the slice removal times, broken up into:

Service_removal: Time to deactivate the radio service and remove the service VNFs. This time is measured to be between 20 and 25 seconds.

Slice_removal: Time required to delete the slice data structure in the slice manager and remove the core related network functions. This time is measured to be between 25 and 30 seconds.

Based on these measurements we can see that a 5G-CLARITY slice can be removed in less than 55 seconds, also well below the project KPI of 5 minutes.

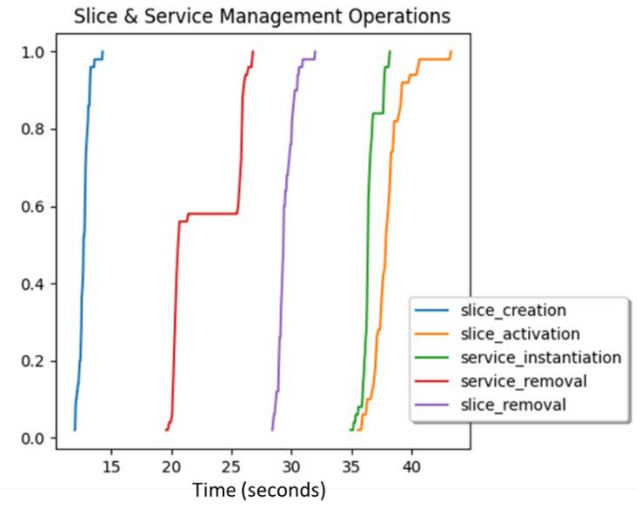



**Figure 2-11. Experimental CDF of 5G-CLARITY slice provisioning and deletion times**

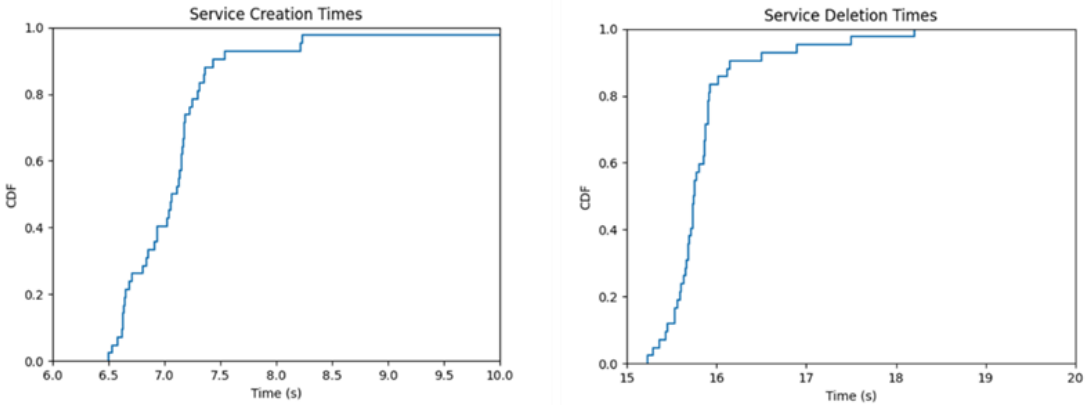



**Figure 2-12. Experimental CDF of WAT service creation and deletion times**

Now we take a closer look at the provisioning time of the radio service, which was included within the slice_activation time in Figure 2-11. Unlike in D4.2 [1], where we showed service provisioning times split by type of wireless access technology, we show a single curve here because our current implementation makes it possible to configure all the technologies at once with a single service call from Slice Manager, which was not possible at the time of writing D4.2. Figure 2-12 depicts the measured wireless service provisioning times, showed as T2 in Figure 2-10. We can clearly see in the left part of Figure 2-12 how the time required to configure all the wireless access nodes in our testbed according to the service parameters provided in the slice request is below 10 seconds in all our trials. These times show how the wireless configuration is a minor component of the overall slice provisioning time, due for example to an optimized implementation of the Multi-WAT Non-real Time RIC that configures all radios in a request in parallel. Correspondingly, the right part of Figure 2-12 depicts service deletion parts, which reaches up to 19 seconds in the worst-case. The reason why service deletion takes longer, is due to retransmission attempts of the Multi-WAT Non-real Time controller in case an interface is blocked when trying to delete.

## 2.4 E2E slice provisioning benchmarking

The goal of this section is twofold. First, we want to demonstrate how the 5G-CLARITY management system for private networks developed in WP4 can be integrated with the management system of a public network to provision an end-to-end network slice, comprising both resources from the private network and resources

from the public network. Our second goal is to demonstrate that the end-to-end network slice can be provisioned in less than 10 minutes, thus fulfilling 5G-CLARITY OBJ-TECH-7.

To achieve the previous goals, we need a public network domain and a public network management system that can be integrated with the 5G-CLARITY private management system. Considering that multi-domain management is out of the scope of 5G-CLARITY, we have decided to partner with the 5G-PPP 5GZORRO project [11] to achieve our goal. The joint work between the 5G-CLARITY and 5G-ZORRO projects has resulted in a joint PoC demonstration showcased at EuCNC 2022 and delivered to the ETSI Zero-Touch Service Management (ZSM) working group [12]. A video with the functional demonstration of the end-to-end network slice provisioning based on the setup described in this section has been uploaded to the 5G-CLARITY YouTube channel [13].

Next, we briefly introduce the scope of the 5G-ZORRO project, we describe the selected evaluation scenario and finally provide our benchmarking results on the provisioning times of the end-to-end network slice.

### 2.4.1 Brief overview of 5G-ZORRO

The 5GZORRO project[3] aims at facilitating multi-party collaboration in dynamic 5G environments where operators and service providers often need to employ 3rd party resources to satisfy a contract. To achieve this, resource providers make their resource offers available for sharing by advertising them through a distributed 5G Marketplace. In general terms, the proposed Marketplace, formed by a mesh of decentralized Distributed Ledger Technology (DLT)-anchored Catalogue instances, enables the creation and acquisition of offers that represent a variety of exposed telco digital assets. These offers include individual resources such as infrastructure components and VNF/CNF, RAN elements, spectrum, edge/core resources; as well as composed bundles in the form of services and slices [14].

In relation to ETSI ZSM, 5GZORRO embraces the coexistence of multiple management domains representing the different stakeholders involved in the platform that offer their own management services that will enable the lifecycle management of provisioned services. 5GZORRO platform complements solutions for zero-touch automation with the cross-domain DLT-based Marketplace, which is used to ensure trust among parties and to establish automated service and slice resource sharing between different domains, and enriched with data services supported by means of sharing the operational data produced by different domains to the involved stakeholders through cross-domain monitoring and analytics AIOps services [15]. Thus, a 5G-CLARITY private network domain can connect to the 5G-ZORRO platform and publish its offerings, e.g. 5G-CLARITY infrastructure slices, using the 5G-ZORRO Marketplace.

The 5GZORRO architecture is designed to offer network operators and service providers the needed mechanisms to automatically negotiate network slice requests and resource composition with external providers based on the availability and capabilities of the services and resources offered on the Marketplace to support multi-domain network slice orchestration with zero-touch lifecycle management. Marketplace offers are modelled following standard open interfaces and information models from the TM Forum suite [16]. Essentially, 5GZORRO stakeholders acting as offer providers consolidate resources and/or services by abstracting the features and characteristics from their technical specifications [17]. In particular, technical specifications of slice-type offers are defined by means of GSM NEtwork Slice Type (NEST) containing the specific values of Generic Slice Templates (GST) to be provisioned for the concrete offered slice [18].

In the 5GZORRO platform, the Network Slice and Service Orchestration (NSSO) service, responsible for the automated lifecycle management of requested network services and network slices acquired from the Marketplace, is implemented both at the inter-domain (denoted as Vertical Service Management Function

[3] https://www.5gzorro.eu/

(VSMF) in Figure 2-13) and the intra-domain layers (denoted as Network Slice Management Function (NSMF) in Figure 2-13). At the inter-domain layer, this service manages the lifecycle of the vertical services and end-to-end network slices, and its split into slice subnets, which will be provisioned by the different domains (e.g. the 5G-CLARITY private network domain). At the intra-domain layer, this service triggers the lifecycle management actions of network slices or network slice subnets to be provisioned completely intra-domain, interacting with the different resource managers for the provisioning of the resources [19]. In the case of a 5G-CLARITY private network domain the involved resource manager is the Slice Manager component of the Management and Orchestration stratum.

In terms of implementation, vertical services are defined using "templates" called Vertical Service Blueprints (VSBs). An offer, containing a reference VSB, can then be requested for orchestration, which the NSSO automatically translates into a specific network slices and network services, indicating the corresponding NEST and/or Network Service Descriptor (NSD) that are relayed to underlying slice and service Management and Orchestration (MANO) systems. Likewise, the NSSO, automatically translates the vertical service lifecycle management actions into network slice level actions.

Following the modular principle of the 5GZORRO architecture, the expected flexibility for the lifecycle management of network slices instances is enforced by supporting several components acting as Network Slice Management Function (NSMF) at the intra-domain level. To achieve this, the NSSO is also integrated with the i2CAT 5G-CLARITY Slice Manager (denoted as Infrastructure Slice Management Function in Figure 2-13), which handles the deployment of 5G-CLARITY infrastructure slices at the intra-domain level, supporting the management of services and resources to ensure slicing principles such as resource allocation, isolation and dedicated connectivity establishment. For this integration, the 5G-CLARITY Slice Manager has been extended to support a NEST-based network slice provisioning.

### 2.4.2 Design of end-to-end network slice

The setup used to demonstrate end-to-end network slicing in WP4 is inspired by the PNI-NPN slice use case developed as part of 5G-CLARITY UC 2.1 in the BOSCH factory in Aranjuez and reported in D5.2 [20]. The use case story consists of the deployment of a PNI-NPN slice where all 3GPP network functions are deployed within the private network, and a computer vision application function is deployed in the edge cloud of the public network. Thus, this use case features two domains in terms of ETSI ZSM. The 5G-CLARITY private network domain, deploying all 3GPP network functions and an AGV mounted camera in the factory floor that transmits pictures every time that it is blocked by an object, and the public network domain, deploying the computer vision application used to identify the objects blocking the progress of the AGV in the factory floor. The interested reader can find additional details about the rationale and business motivations for this use case in deliverable D5.2 [20].

Figure 2-13 depicts the network setup used in the ETSI PoC highlighting the elements provided by the 5G-CLARITY project and the elements provided by the 5G-ZORRO project, as well as the two independent management domains:

**Private network infrastructure**: Hosted at i2CAT laboratory. A 5GNR gNB, based on Amarisoft Callbox Pro, a custom Wi-Fi AP, and an edge compute cluster based on OpenStack deploying a 5GCore and an AT3S user plane function VNFs. In addition, a 5G-CLARITY CPE with a connected camera is also considered. This is the same testbed used in D3.3 section 3 for the MPTCP latency evaluation [21]. This infrastructure represents the private network domain in the ETSI ZSM PoC.

**Private network management plane**: Consisting of the service and slice provisioning subsystem of the 5G-CLARITY management stratum. This subsystem features three management functions, namely the Multi-WAT non-rt Controller (referred to as Multi-access controller in the ETSI PoC figure), an NFVO based

on OSM and a VIM based on OpenStack (referred to as MANO in the ETSI PoC figure), and the 5G-CLARITY Slice Manager (referred to as Infrastructure Slice MF in the ETSI PoC figure).

**The public network infrastructure**: Consisting of an NFVI implemented in the 5TONIC network from Telefonica [22], which represents the public network domain in the ETSI ZSM PoC. The 5G-CLARITY obstacle detection application function is provisioned in this domain. The interested reader is referred to D5.2 [20] for a detailed description and evaluation of this obstacle detection function.

**Public network management plane**: Consisting of a MANO, i.e. NFVO and VIM functions, and a Network Slice Subnet Management Function (NSMF). These functions are contributed to the testbed by the 5G-ZORRO project.

**E2E management domain**: Consisting of a network service Catalogue and a Vertical Service management function (VSMF). These functions are provided by the 5G-ZORRO project. The VSMF component is the one that will interact with the resource managers of each domain, i.e. Slice Manager at i2CAT lab, and the NSMF at 5TONIC, to trigger the end-to-end network slice.

A key aspect to be able to easily provision end-to-end network services across private and public domains is to provision the WAN connectivity. WAN connectivity should address the following requirements:

Automation: WAN connectivity provisioning should be automated as part of the slice deployment without requiring any additional manual provisioning step by the private or public network operators. To this end, the private network is assumed to have Internet connectivity and the public network is assumed to have a pre-installed VPN service endpoint to enable connection from remote domains.

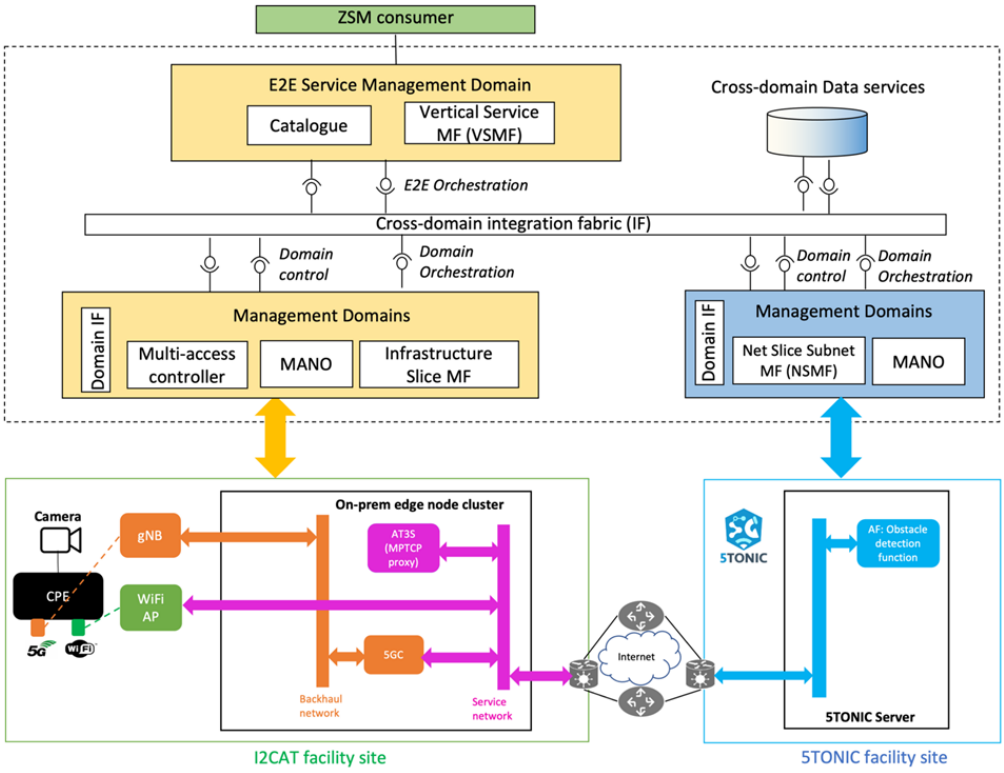



**Figure 2-13. Network setup for end-to-end network slice provisioning used at ETSI ZSM PoC [23]**

Isolation: Deploying an end-to-end slice across the private and public network domains should result in an isolated connectivity service whereby the network functions in the private network that are part of the slice can only access the corresponding functions in the public network, and vice versa.

To deliver on the previous requirements, we design a solution consisting of:

A L2 VPN, based on VXLAN [24], which results in the service network in the private network (pink network in Figure 2-13) being directly connected to the service network in the public network (blue network in Figure 2-13) at layer 2.

To automate the provisioning of the L2 VPN, a dedicated L2 VPN VNF is onboarded on the private network MANO and is included as part of the network service that is used to provision the required functions on the private network. Thus, when deploying the private side of the end-to-end slice, the 5G-CLARITY Slice Manager at i2CAT first provisions the 5GCore (5GC) function directly over the VIM, and then instantiates a network service through MANO that includes the AT3S user plane function and the L2 VPN function.

The L2 VPN deployment on the private side needs to be coordinated with that of the public side, so that the same VXLAN endpoint is used in the two domains. The E2E management domain is in charge of mediating this coordination.

The interested reader can find the details of the WAN connectivity solution used in this PoC in [25].

Figure 2-14 provides a sequence diagram illustrating the interactions between the management functions of the private and public domains. Figure 2-14 highlights in different colours the management domains, i.e. yellow for the private network management domain (ZSM operator #1), blue for the public network management domain (ZSM operator #2), and the two infrastructure domains, grey for the private network domain (Infra domain #1) and orange for the public domain (infra domain #2). For simplicity the sequence diagram in Figure 2-14 is broken in three main domains:

Step 1: Triggering of the E2E slice provisioning request from the catalogue. This translates into a request towards the VSMF, which in turns involves the resource managers of each domain. Notice that for ease of implementation we have deployed the E2E management components (Catalogue and VSMF) together with the private network management plane hosted at i2CAT, whereas in practice these components could be deployed elsewhere. Figure 2-13 illustrates how the E2E slice offering is available from the Catalogue, and how it translates into slice templates in the VSMF.

Step 2: The resource managers in each management domain trigger their respective network provisioning actions. In the case of the private network domain at i2CAT, the Slice Manager function uses the Multi-Access controller to configure the Wi-Fi AP and the gNB, and OSM to instantiate the network service containing the 5GC, the AT3S and the L2 VPN network functions. In the case of 5TONIC, the NSMF triggers OSM to instantiate the obstacle detection function.

Step 3: When all the physical and virtual network functions in each domain are instantiated and the service is ready to operate.

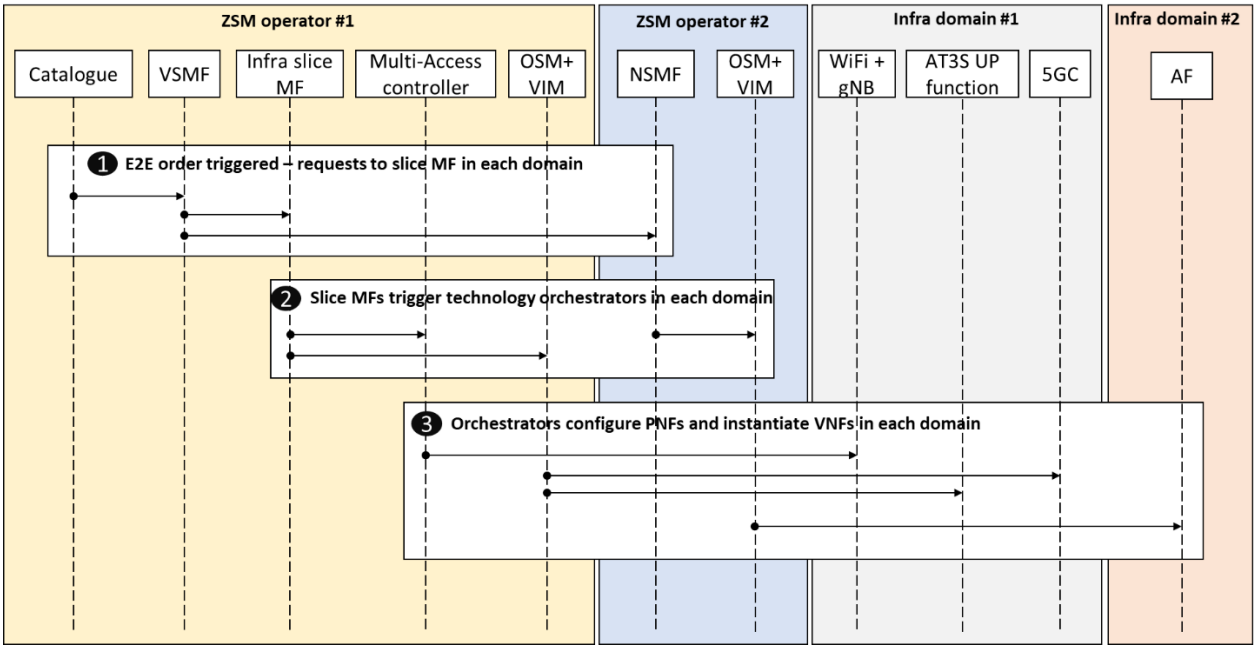



**Figure 2-14. High-level sequence chart describing the interactions between the private and public slices in the considered scenario**

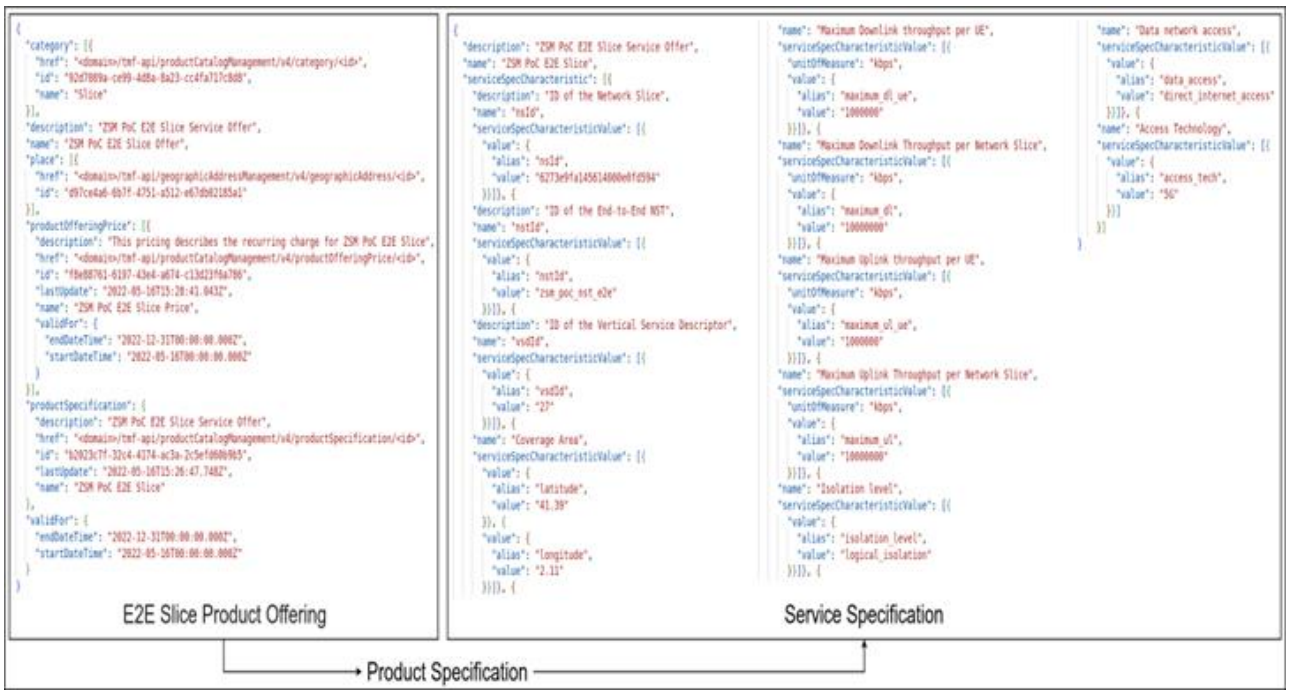



**Figure 2-15. Specification of E2E slice product offering in the catalogue – TM forum compliant (left), and corresponding 3GPP compliant NEST template from the vertical slice management function (right)**

In the next section we benchmark the time required to provision this end-to-end network slice.

### 2.4.3 Benchmarking end-to-end network slice provisioning time

The experiment demonstrating the automated deployment of an end-to-end slice, including a 5G-CLARITY private network domain and an MNO domain, is documented in detail in the public demonstration available in [23]. In this section we collect the evidence of the developed demonstration and justify that the 5G-

CLARITY KPI of deploying an end-to-end slice in less than 10 minutes has been fulfilled.

Figure 2-16 contains a screenshot of the public demonstration [23] in the moment when we are about to trigger the provisioning of the end-to-end slice from the Catalogue (step 1 in Figure 2-14). Looking at the timestamp available in the CLI in the right part of the image, we can see the time being 17:44. Subsequently, Figure 2-17 contains another screenshot of the public demonstration when the instantiation of the end-to-end slice has been completed, including the instantiation of the obstacle detection function at 5TONIC and the provisioning of the 5G-CLARITY private network slice at the i2CAT lab. The CLI on the right part of the image shows that the time when the slice is provisioned is 17:47. Therefore, the end-to-end slice has been provisioned in approximately 3 minutes. We refer the interested reader to the video of the public demonstration to observe all steps involved in the provisioning of the end-to-end slice.

Finally, to illustrate the successful instantiation of the network services in each domain, Figure 2-18 depicts a screenshot with the OSM dashboard in the i2CAT (left) and the OSM dashboard in the 5TONIC domain (right). Both domains indicate that their respective network services are in running state (green check mark). The public demonstration contains additional evidence showing how the 5G-CLARITY CPE can connect to the deployed end-to-end network slice and transmit pictures of obstacles that are successfully detected by the obstacle detection function in the 5TONIC domain.

We conclude highlighting that this experiment demonstrates that the developed 5G-CLARITY management plane can be used to automate the deployment of end-to-end network slices comprising private and public domains in less than 10 minutes. We acknowledge that the evidence we provide is supported by a single experiment, and that the actual instantiation times of a given slice will depend on the network services required in that case. However, we argue that the example we have considered is a representative one, and the fact that we can provision this end-to-end slice in only 3 minutes makes it reasonable to argue that in general provisioning times will be below 10 minutes.

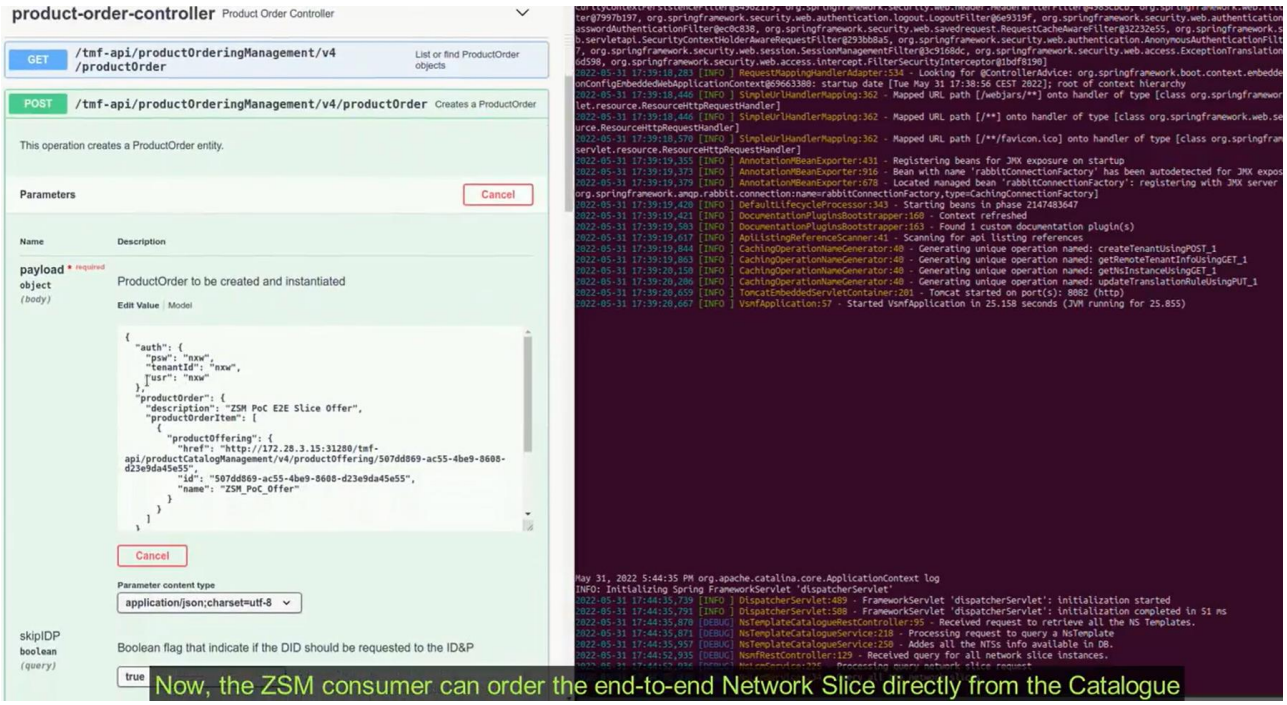



**Figure 2-16. Screenshot from E2E slice provisioning process before triggering the E2E slice provisioning from the catalogue - time: 17:44**

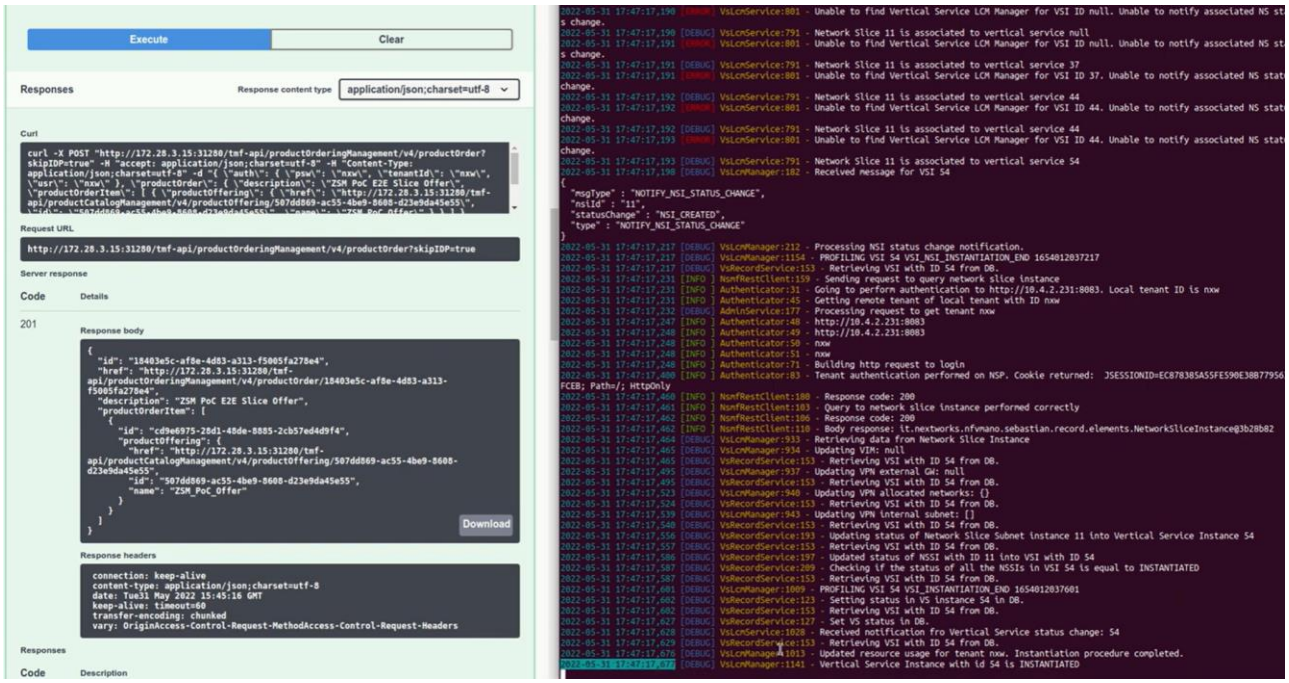



**Figure 2-17. Vertical service instance INSTANTIATED - time: 17:47**

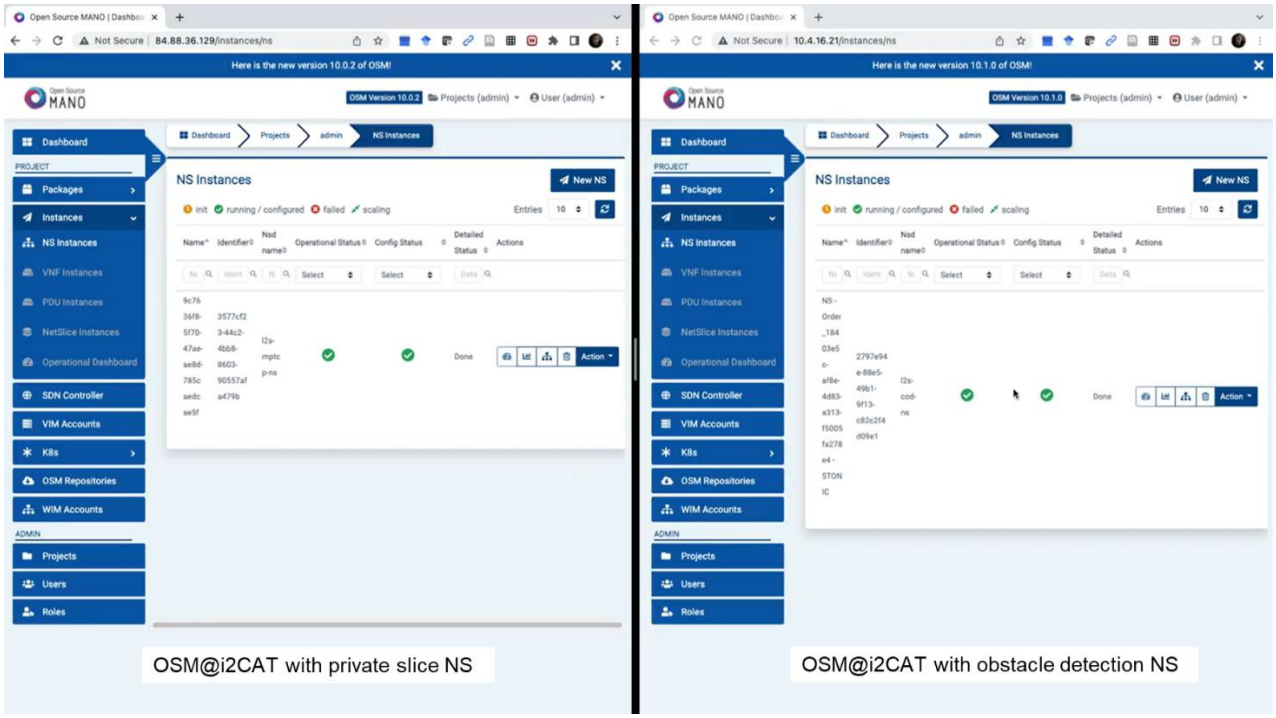



**Figure 2-18. Deployment of the required network services (NS) in the private (left) and public (right) NFVIs**

# 3 Telemetry Subsystem

This section describes the data lake and data semantics fabric implementation of the 5G-CLARITY data management and processing subsystem. The section is organized in the following three subsections:

Section 3.1 provides a high-level overview of the components of the 5G-CLARITY telemetry subsystem and their extensions carried out during the project.

Section 3.2 describes the interfaces that are discussed in D4.2 **[1]** and used to integrate various data sources such as access network telemetry, MPTCP telemetry, transport network telemetry and channel impulse response telemetry to data lake.

Section 3.3 describes the transport network data sources within the data semantics fabric along with experimental scenario details to collect transport network telemetry.

Section 3.4 describes the integration of the two data management and processing subsystems, namely, data lake and data semantics fabric.

Figure 3-1 is used to remind the 5G-CLARITY telemetry framework and existing interfaces captured in D4.2 [1] along with interface-section mapping for this deliverable.

## 3.1 Overview of required implementation and integrations

This section reports on the final implementation of the 5G-CLARITY telemetry subsystem that is composed of two main components, namely Data Lake and Data Semantics Fabric, as well as various resource components that provide telemetry data. Developing this subsystem required the use of software and open-source modules available in the state of the art, along with other background assets provided by partners that have been extended in the project, as well as other modules that have been developed from scratch.

Table 3-1 provides a detailed overview of all the components as part of the 5G-CLARITY telemetry subsystem and highlights the background and extension of the modules during the project. The table also summarizes the experimentally validated modules in this deliverable.

**Table 3-1. Overview of modules composing the 5G-CLARITY telemetry subsystem**

| Module | Background | Extensions in 5G-CLARITY | Responsible partner | Module integrations validated in this section |
|---|---|---|---|---|
| Data Lake | The Data Lake is a cloud-based approach where the cloud computing platform AWS is provided by Amazon. It comprises a multitude of services, including computing, networking, storage, database, analytics and IoT. | Specific AWS services and components are adapted as part of 5G-CLARITY Data Lake solution In order to integrate the Data Lake to the 5G-CLARITY system architecture. Various interfaces are defined to enable data flow from radio access networks, UE to AI engine via Data Lake. | IDCC | API to push/pull telemetry data from various network components.<br>Data storage for specific telemetry data.<br>Data schema details to discover available telemetry data. |
| DSF | The Data Semantic Fabric (DSF) is | DSF monitoring and | UPM & TID | Adding support |

| | | | | |
|---|---|---|---|---|
| | a model-driven monitoring framework that implements a data catalog system based on the ETSI CIM standards and applies streaming telemetry techniques based on YANG data modelling (an opensource contribution available on GitHub[4]). The implementation of the DSF solution evolves from a work done previously on the 5GROWTH project. | operational mechanisms adapted and enhanced to incorporate transport network devices as telemetry data sources for the Data Lake. DSF updates basically include the following:<br>1. Add monitoring mechanisms to collect telemetry data from YANG-based network devices, transform and aggregate the telemetry data to compute new KPIs, and deliver the resulting information to the Data Lake.<br>2. Add operational mechanisms for the registration of YANG-based network devices as data source and the Data Lake as data consumer within the DSF, and the discovery of their capabilities, and also the definition of full data pipeline processes for enable the collection, aggregation, and delivery of the resulting telemetry data. | | for transport network devices as telemetry data sources in the DSF framework and interoperability between the DSF and Data Lake telemetry subsystem modules. |
| Multi-WAT xApp | N/A | Developed from scratch | I2CAT | Near Real Time RIC and Data Lake |
| TCP Telemetry xApp | N/A | Developed from scratch | UGR | Data Lake |
| CIR Telemetry | N/A | Developed from scratch | IHP | Data Lake |

## 3.2 Integration of data sources in Data Lake

There are five main management services introduced for the data lake in 5G-CLARITY D2.2 [2]. These services include:

**Data Lake Ingress Service** that allows ingestion of data (structured or unstructured) to the data lake;

**Data Lake Exposure Service** that exposes data lake storage to authenticated users;

**Data Lake Data Security Service** that maintains data access policies of the data lake;

[4] https://github.com/giros-dit/semantic-data-aggregator

**Data Discovery Service** that allows data discovery queries to the data lake metadata; and

**Data Exploration Service** that allows user to gain access to specified data in the data lake.

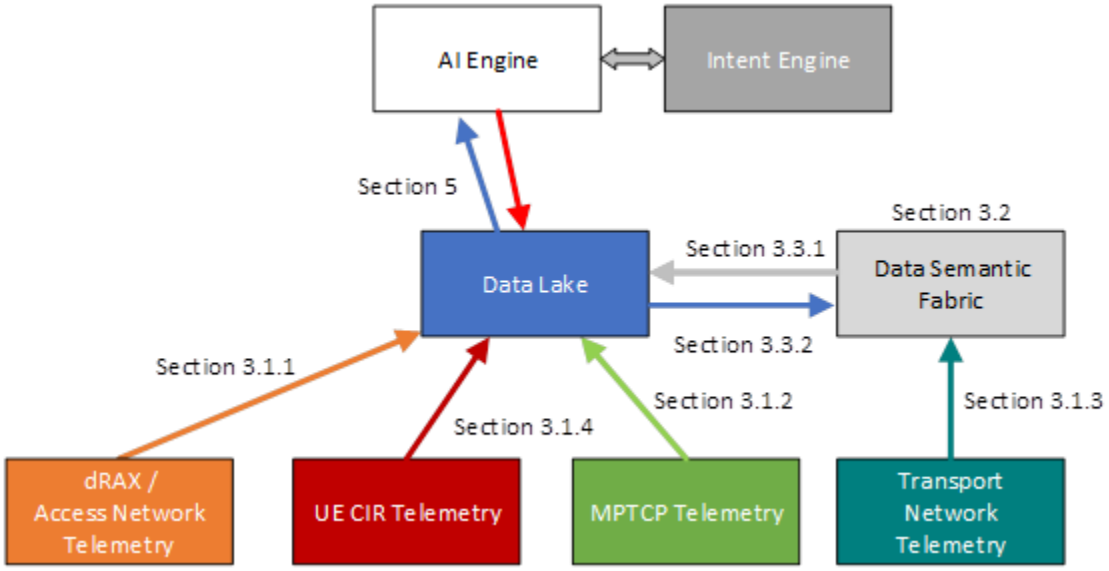



**Figure 3-1. Interface-section mapping for the 5G-CLARITY telemetry framework**

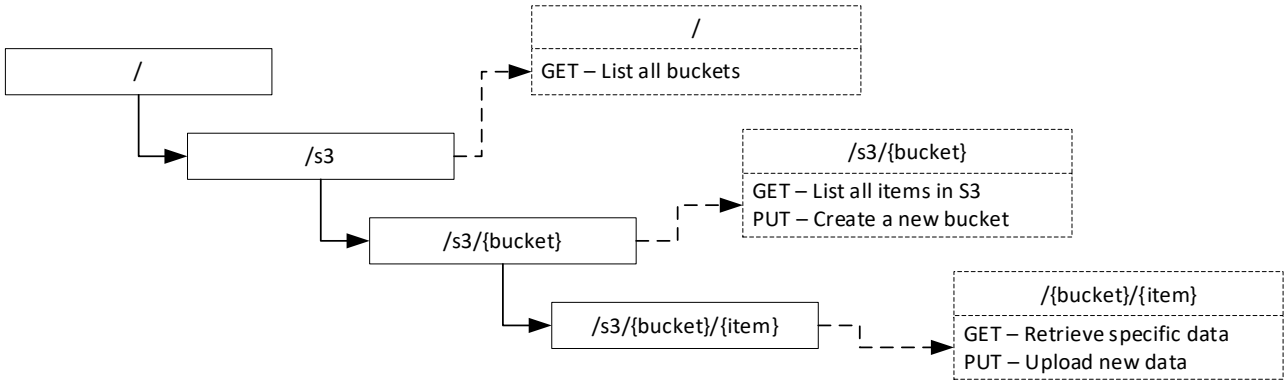



**Figure 3-2. S3 end points used by API Gateway [1]**

As discussed in 5G-CLARITY D4.2 [1], the entry way to the AWS data lake cloud is the AWS API Gateway. The API Gateway enables various system components to push/put data to the data lake as well as enables users and/or system components to fetch/pull any telemetry data that is stored in the data lake. The end points used by the API Gateway within a S3 (Simple Storage Service) storage environment are depicted in Figure 3-2. These end points enable the API to upload and fetch data to/from different buckets as well as different objects in those buckets.

More specifically, the API Gateway has a GET and a PUT method execution for both S3 buckets and objects. The method execution flow for a GET request of an object in an S3 bucket is shown in Figure 3-3.

The detailed actions to GET/PUT requests to/from the bucket and objects are available online[5].

The described requests of the API Gateway support the **Data Lake Ingress Service** (PUT request to a specific S3 bucket or item), **Data Lake Exposure Service** (GET request to a specific S3 bucket or item) and **Data**

[5] https://docs.aws.amazon.com/AmazonS3/latest/userguide/RESTAPI.html

**Exploration Service** (GET request to root S3 folder).

Regarding the **Data Lake Data Security Service**, a location constraint and API key are defined along with a set of specific access policies to each existing S3 bucket. These access policies include or exclude specific actions on each S3 bucket and each item within each bucket. For example, deleting an existing object/item or S3 bucket via API calls may not be allowed. In another example, accessing a previous version of an object can be disabled for some buckets/objects.

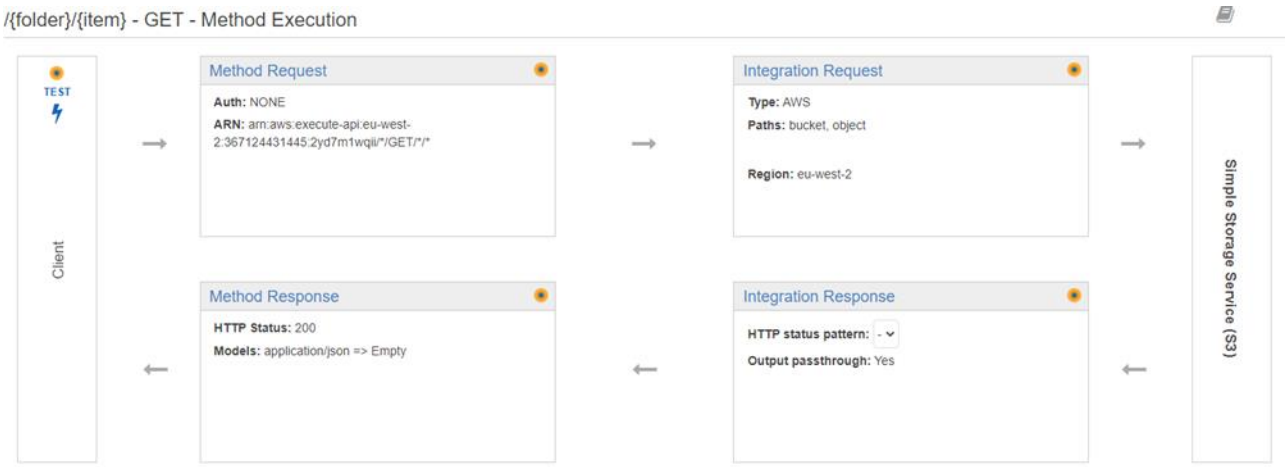



**Figure 3-3. GET request method execution for an object in an S3 bucket**

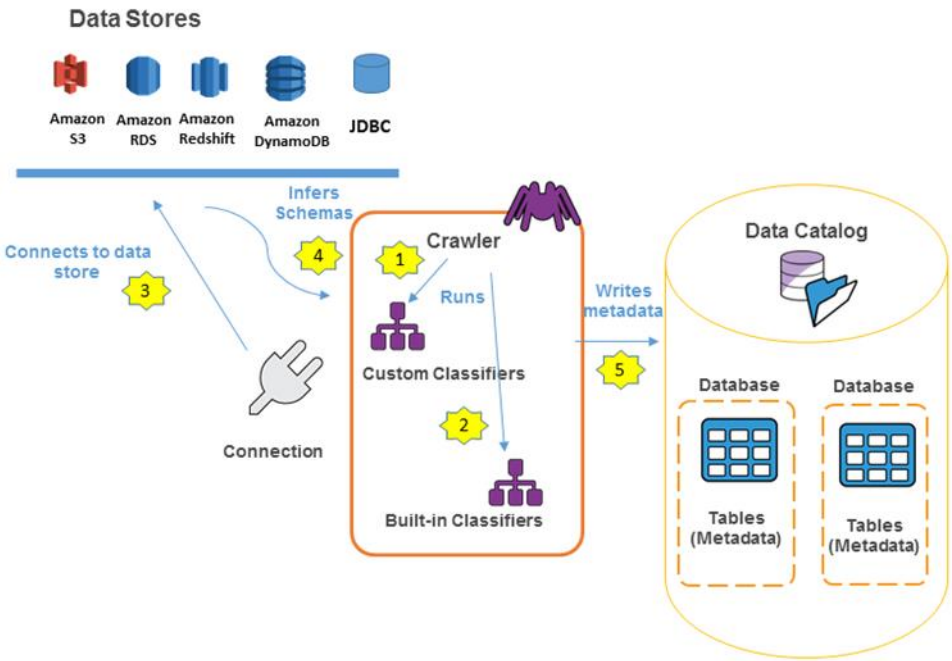



**Figure 3-4 Workflow diagram of AWS Glue Crawlers**[6]

For the **Data Discovery Service**, another AWS tool named AWS Glue is used. AWS Glue is a fully managed ETL (extract, transform, and load) service. Its different capabilities such as Data Catalog, Crawler and Classifier enable the data lake to not only store, annotate but also scan data in all repositories to classify and extract metadata information automatically. Figure 3-4 shows the workflow diagram of AWS Glue Crawler

to populate Data Catalog[6].

In the reminder of this section, we describe how each telemetry source depicted in Figure 3-1 is integrated in the data lake, what telemetry metadata is available in the data lake and how the metadata is structured.

### 3.2.1 Multi-WAT telemetry xApp

In this section we describe our approach to design an xApp running in the near real-time RIC that can extract Multi-WAT telemetry and export it to the data lake. First, in Section 3.2.1.1, we present the design of the xApp. Second, in Section 3.2.1.2, we demonstrate how this xApp has been integrated with the data lake.

#### 3.2.1.1 Multi-WAT xApp design

The main purpose of this xApp (see Figure 3-5) is to extract cellular and Wi-Fi/LiFi telemetry present on the near real-time RIC (ACC's dRAX) and to publish it to the Intelligence Stratum's data lake provided by IDCC, i.e., AWS S3. The xApp can be externally configured by the 5G-CLARITY intelligence stratum, e.g., by the Intent Engine, to specify which topics are to be published, the publication interval and several filters to further select the desired metrics, where this information could be provided by the AI model in the AI engine interested in a certain subset of metrics.

Next, we present our implementation of the 5G-CLARITY xApp for wireless telemetry. Our design is generic to be able to handle all types of wireless technologies considered in 5G-CLARITY.

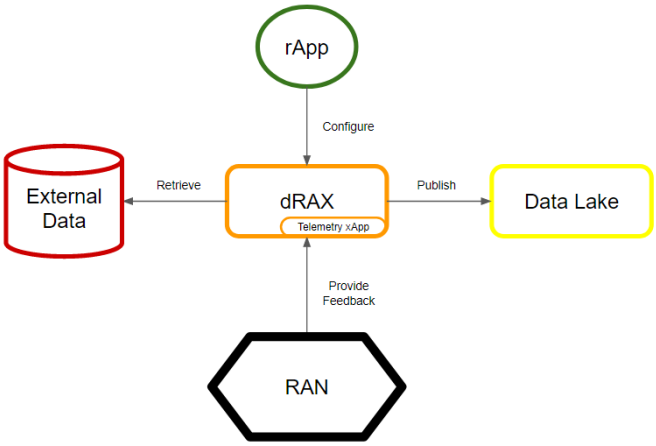



**Figure 3-5. Components involved in the xApp workflow**

However, in our implementation we only demonstrate 4G and Wi-Fi telemetry, due to implementation constraints that we had at the time of writing this deliverable. For example, two versions of the dRAX product are available from ACC's to support 4G or 5G systems. At the time of writing this deliverable the 5G version of dRAX was being used for the service provisioning work of this deliverable described in Section 0, thus we decided to focus our telemetry work on the 4G version of dRAX. The same developed xApp will be migrated to dRAX 5G after the service and slice provisioning work is completed, where additional 5G telemetry topics will be available.

In our current implementation the xApp obtains 4G and Wi-Fi telemetry in the following way:

**4G telemetry**: dRAX has native support for obtaining 4G or 5G telemetry, depending on dRAX's version. Said telemetry is available in the Kafka databus, and any xApp can subscribe to the different topics available.

[6] https://docs.aws.amazon.com/glue/latest/dg/populate-data-catalog.html

**Wi-Fi telemetry**: We have developed another xApp that retrieves Wi-Fi telemetry from an external Prometheus server and publishes said telemetry in the Kafka databus. This architecture was described in D4.2 [1] and can also be used to integrate LiFi telemetry.

Table 3-2 indicates the available 4G and Wi-Fi metrics that can be exported from dRAX to the Data Lake.

A key aspect of the developed xApp is its configuration capabilities. Recall that in the 5G-CLARITY architecture it is the AI/ML models sitting in the AI engine that will request the subset of telemetry that they are interested in to perform their inference predictions.

**Table 3-2. Available 4G and Wi-Fi Telemetry**

| 4G Metrics | Wi-Fi Metrics |
|---|---|
| beaconInfo | AP Frequency (Hz) |
| serviceFound | AP Max Transmission Power (dBm) |
| UE Measurements | AP Number of connected stations |
| UE Throughput | Station Backlog Bytes (Total) |
| CQI | Station Connected Time (Total) |
| BLER | Station Transmitted Bytes (Total) |
| | Station Transmitted Rate (bps) |
| | Station Received Bytes (Total) |
| | Station Received Rate (bps) |
| | Station Signal (dBm) |
| | Station Airtime (Total) |

This request will be expressed by the AI/models as an intent to the Intent Engine, which will then configure our developed Telemetry xApp to provide the requested configuration. To enable the customized Telemetry configuration directly in the xApp we have made use of the concept of ORAN A1 Policies, which allows to customize the reporting behaviour of the xApp directly in the near real-time RIC. Notice that this is much more efficient that a naïve telemetry policy, where the near real-time RIC simply uploads all raw data to the data-lake, which is where the filtering is done. As reported in D2.4 [26] 5G-CLARITY private networks are expected to generate significant amounts of data, hence filtering the relevant data as close as possible to the point where the data is generated is important for efficiency reasons. This is what the developed xApp, which will be deployed in the 5G-CLARITY RAN cluster, allows.

In the 5G-CLARITY system there could be various AI/ML models requesting different subsets of wireless telemetry simultaneously. This is supported in our architecture, by means of deploying a separate Telemetry xApp to serve the data needs of each AI/ML model. Each xApp allows to be configured through an A1 policy that specifies: 1) the data-lake credentials where the data needs to be published, 2) the filtering criteria for the 4G telemetry, and 3) the filtering criteria for the Wi-Fi telemetry. These configuration options are explained in detail next:

**Data Lake configuration**: Currently the only data lake supported by the xApp is AWS S3. We offer two different ways of providing the necessary credentials (see Figure 3-6) to publish in a S3 Bucket, that do not affect the behaviour of the xApp Itself. The first option is to provide a pair of ‘access key’ and ‘secret access key’, which are security credentials that provide access to the whole AWS account to which they are associated, and we only recommend using them for testing purposes. For a production scenario we suggest the use of a pair of “API URL” and “API Key”, which are credentials exclusively related to an S3 bucket, and are much safer to share.

**4G telemetry configuration**: The 4G telemetry parameters that can be configured include the list of topics, out of all the ones listed in Table 3-2, to be published to the data lake; the publication interval, and

the chosen pair of credentials used to access the S3 Bucket, out of the two possible alternatives described in the previous section. Figure 3-7provides an example of A1 policy instance defining a 4G filtering criteria.

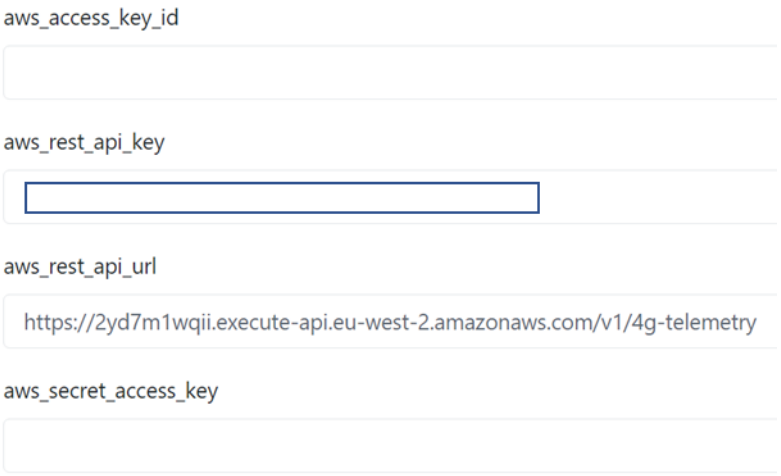



**Figure 3-6. AWS S3 credentials in the xApp configuration**


```
{
  "policy_id": "telemetry-test-2",
  "policytype_id": "1",
  "ric_id": "drax_ric",
  "policy_data": {
      "scope": {
          "telemetry_topics": "serviceFound, beaconInfo",
          "update_interval": 15,
          "aws_access_key_id": "AKIASQJG6UFUL76WIQHK",
          "aws_secret_access_key": "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"

      }
  },
  "service_id": "API",
  "transient": true
}
```


**Figure 3-7. Policy instance for the configuration of 4G telemetry**


```
{
  "policy_id": "telemetry-test-2",
  "policytype_id": "2",
  "ric_id": "drax_ric",
  "policy_data": {
      "scope": {
          "wifi_topics": "All",
          "wifi_update_interval": 15,
          "ap_ips": "192.168.0.123",
          "phy_ids": "",
          "sta_macs": "08:23:70:71:4b:ba",
          "aws_rest_api_url": "https://2yd7m1wqii.execute-api.eu-west-2.amazonaws.com/v1/wifi-telemetry",
          "aws_rest_api_key":

      }
  },
  "service_id": "API",
  "transient": true
}
```


**Figure 3-8. Policy instance for the configuration of Wi-Fi telemetry**

**Wi-Fi Telemetry**: In a similar fashion as described in the previous section, the configuration related to Wi-Fi telemetry has its own A1 policy to control different parameters. Just like previously described

policy, it includes the list of topics to publish (out of all topics listed in Table 3-2), the publication interval and the AWS credentials. The main difference is that the Wi-Fi Telemetry policies also supports a set of filters further select the topics to be published. The filters include the access point IP address, the physical interface ID and the station MAC address involved in each topic. Each of the filters can either be empty, or can consist of one or multiple inputs. The filters are combined to select only the topics to which all three filters can be applied. For instance, in the policy seen in Figure , the xApp will only publish the topics involving both the access point with the IP address '*192.168.0.123*' and involving the station with the MAC address *'08:23:70:71:4b:ba*'. To publish all Wi-Fi topics it is possible to set the value of the '*Wi-Fi_topics*' field to '*All*'.

### 3.2.1.2 Multi-WAT xAPP and Data Lake integration validation

The following figure shows the snapshot of the 4g-telemetry bucket in the data lake. As it can be seen, various telemetry topics are ingested to the data lake successfully.

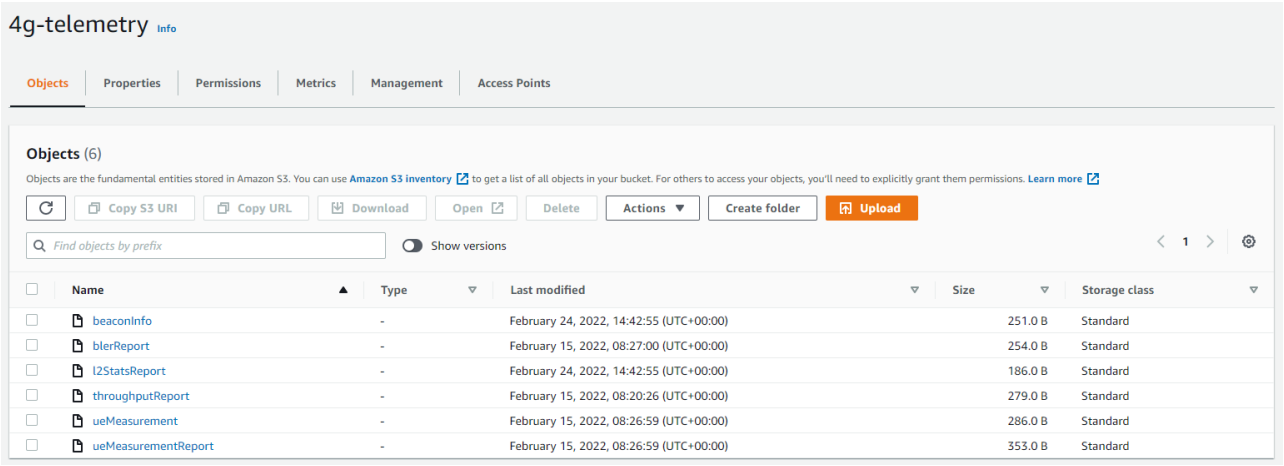



**Figure 3-9. Snapshot of the S3 bucket dedicated to 4G telemetry data**


```
"{\"throughputReport\": {\"cellId\": \"celllab6\", \"dlThroughput\": 44, \"ueDraxId\":
\"cc516426-efba-4fba-b817-7d4876844468\", \"ueRicId\": \"UE_85\", \"ulThroughput\": 7}, \"timestamp\":
1644913224350102259, \"topic\": \"Topic_OPENRAN_DATA\", \"type\": \"THROUGHPUT_REPORT\"}"
```


**Figure 3-10. Snapshot of the GET request for throughputReport object in the 4G telemetry data bucket**

| Name | Database | Location | Classification | Last updated |
|---|---|---|---|---|
| uemeasurementreport | 4g_telemetry_db | s3://4g-telemetry/ueMeasurementReport | json | 20 May 2022 9:33 AM UTC+1 |
| beaconinfo | 4g_telemetry_db | s3://4g-telemetry/beaconInfo | json | 20 May 2022 9:33 AM UTC+1 |
| 4g_telemetry | 4g_telemetry_db | s3://4g-telemetry/ | Unknown | 20 May 2022 9:32 AM UTC+1 |
| l2statsreport | 4g_telemetry_db | s3://4g-telemetry/l2StatsReport | json | 20 May 2022 9:33 AM UTC+1 |
| uemeasurement | 4g_telemetry_db | s3://4g-telemetry/ueMeasurement | json | 20 May 2022 9:33 AM UTC+1 |
| blerreport | 4g_telemetry_db | s3://4g-telemetry/blerReport | json | 20 May 2022 9:33 AM UTC+1 |
| throughputreport | 4g_telemetry_db | s3://4g-telemetry/throughputReport | json | 20 May 2022 9:33 AM UTC+1 |

**Figure 3-11. Output tables of the 4G telemetry-specific crawler. Figure  shows the output of a GET request for the throughput Report object inside the 4G telemetry data bucket in the data lake**

In order to classify and extract metadata information inside the 4G telemetry bucket, a crawler is designed. The crawler generated a set of tables for each object in the 4G telemetry bucket where location, classification and last updated information are obtained, as shown in Figure 3-11.

Figure 3-12 shows a schema of the table generated for l2statsreport object. In addition to that, Figure 3-13 shows schema details generated for l2statsreport, blerreport and throughputreport objects.

| | |
|---|---|
| Name | l2statsreport |
| Description | |
| Database | 4g_telemetry_db |
| Classification | json |
| Location | s3://4g-telemetry/l2StatsReport |
| Connection | |
| Deprecated | No |
| Last updated | Fri May 20 09:33:02 GMT+100 2022 |
| Input format | org.apache.hadoop.mapred.TextInputFormat |
| Output format | org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat |
| Serde serialization lib | org.openx.data.jsonserde.JsonSerDe |
| Serde parameters | paths: l2StatsReport,originator,timestamp,topic,type |
| Table properties | sizeKey: 275; UPDATED_BY_CRAWLER: 4G telemetry Clarity; CrawlerSchemaSerializerVersion: 1.0; recordCount: 1; averageRecordSize: 275; CrawlerSchemaDeserializerVersion: 1.0; compressionType: none; typeOfData: file |

Schema

Showing: 1 - 5 of 5

| | Column name | Data type | Partition key | Comment |
|---|---|---|---|---|
| 1 | l2statsreport | struct | | |
| 2 | originator | string | | |
| 3 | timestamp | bigint | | |
| 4 | topic | string | | |
| 5 | type | string | | |

**Figure 3-12. Table details and schema of l2statsreport object in the 4G telemetry bucket**

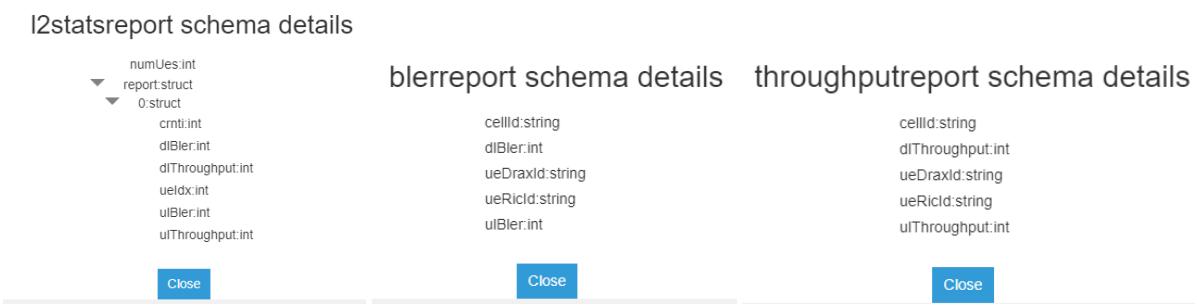



**Figure 3-13. Schema details of l2statsreport (left), blerreport (middle) and throughputreport (right) objects in the 4G telemetry bucket**

**Table 3-3. 4G telemetry data details/semantics**

| Object | Description | Encoding Format | Metadata | Value Type |
|---|---|---|---|---|
| throughputReport | Downlink and Uplink throughputs of the UEs in the network. | UTF-8 | cellId | string |
| | | | dlThroughput | float |
| | | | ueDraxId | string |
| | | | ueRicId | string |
| | | | ulThroughput | float |
| | | | timestamp | string |
| | | | topic | string |
| | | | type | string |
| l2StatsReport | L2 related counters | UTF-8 | numUes | int |
| | | | report | string |
| | | | crnti | int |
| | | | dlBler | float |
| | | | dlThroughput | float |
| | | | ueIdx | Int |
| | | | ulBler | int |
| | | | ulThroughput | Float |

| | | | | |
|---|---|---|---|---|
| | | | originator | string |
| | | | timestamp | float |
| | | | topic | string |
| | | | type | string |
| ueMeasurement | These measurements contain the RSRP and RSRQ value from the UE to its serving cell, as well as a maximum of 8 neighbouring cells | UTF-8 | cellId | string |
| | | | rsrp | int |
| | | | rsrq | int |
| | | | ueCellId | string |
| | | | ueDraxId | string |
| | | | ueRicId | string |
| cqiReport | CQI report of the UE to its serving cell. You can get the CQI value from each subband as well as the wideband CQI value | UTF-8 | cellId | string |
| | | | cqiList | List<int> |
| | | | widebandCqi | int |
| | | | ueRicId | string |
| | | | ueDraxId | string |
| blerReport | BLock Error Rate of the communication between the UE and the serving cell | UTF-8 | cellId | string |
| | | | dlBler | float |
| | | | ulBler | float |
| | | | ueRicId | string |
| | | | ueDraxId | string |

Figure 3-14 shows the snapshot of the Wi-Fi-telemetry bucket in the data lake. A set of access point related telemetry topics are ingested to the data lake successfully.

Figure 3-15 shows the output of a GET request for the hostapd_sta_signal_dBm object inside the Wi-Fi telemetry data bucket in the data lake.

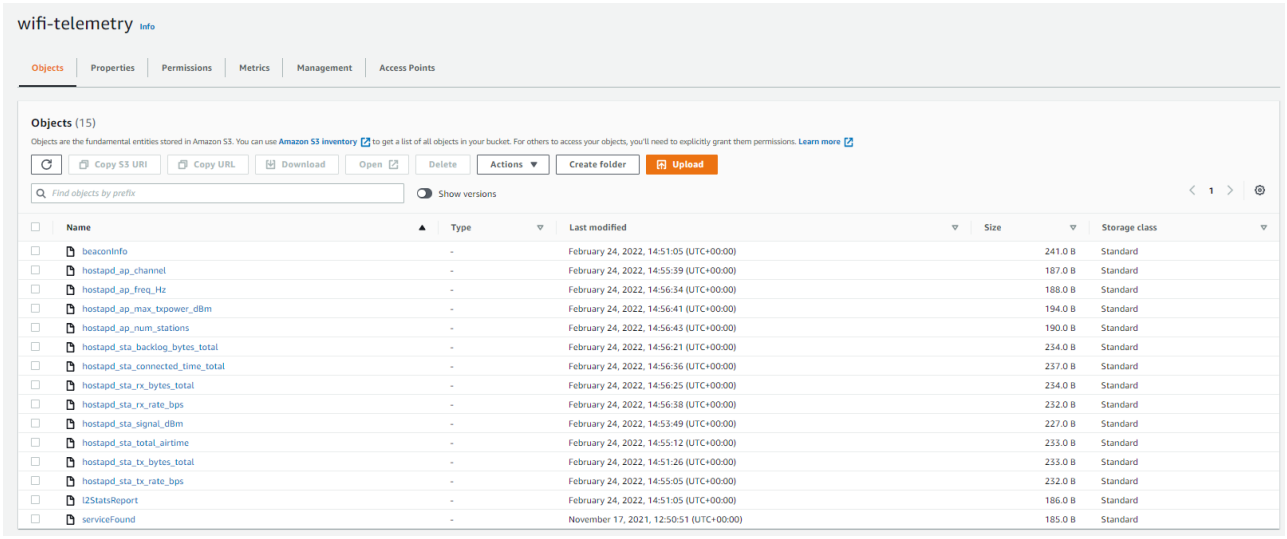



**Figure 3-14. Snapshot of the S3 bucket dedicated to Wi-Fi telemetry data**


```
1  "{\"type\": \"hostapd_sta_signal_dBm\", \"value\": \"-38\", \"metadata\": {\"id\": \"wlp5s0\", \"instance\":
2  \"192.168.40.94:9551\", \"job\": \"racoon_wifi\", \"mac_sta\": \"04:f0:21:39:82:c9\"}, \"timestamp\": 1645714428.226}"
```


**Figure 3-15. Snapshot of the GET request for hostapd_sta_signal_dBm object in the Wi-Fi telemetry data bucket**

**Table 3-4. Wi-Fi Telemetry Data Details/Demantics**

| Object | Description | Encoding Format | Metadata | Value Type |
|---|---|---|---|---|
| hostapd_ap_channel | Frequency channel used by the access point. | UTF-8 | value | int |
| | | | id | string |
| | | | instance | IP |
| | | | job | string |
| | | | timestamp | float |
| hostapd_ap_freq_Hz | Central frequency used by the access point. | UTF-8 | value | int |
| | | | id | string |
| | | | instance | IP |
| | | | job | string |
| | | | timestamp | float |
| hostapd_sta_signal_dBm | Signal intensity received by a station. | UTF-8 | value | int |
| | | | id | string |
| | | | instance | IP |
| | | | job | string |
| | | | timestamp | float |
| | | | mac_sta | MAC |
| hostapd_ap_max_txpower_dBm | Maximum transmission power of an access point. | UTF-8 | value | int |
| | | | id | string |
| | | | instance | IP |
| | | | job | string |
| | | | timestamp | Float |
| hostapd_sta_connected_time_total | Total time that a stations has been connected to the access point. | UTF-8 | value | int |
| | | | id | string |
| | | | instance | IP |
| | | | job | string |
| | | | timestamp | Float |
| | | | mac_sta | MAC |
| hostapd_sta_rx_bytes_total | Total bytes received by a station. | UTF-8 | value | int |
| | | | id | string |
| | | | instance | IP |
| | | | job | string |
| | | | timestamp | float |
| | | | mac_sta | MAC |
| hostapd_sta_rx_rate_bps | Average throughput in bps received by a station. | UTF-8 | value | int |
| | | | id | string |
| | | | instance | IP |
| | | | job | string |
| | | | timestamp | float |

| | | | | |
|---|---|---|---|---|
| | | | mac_sta | MAC |
| hostapd_sta_tx_bytes_total | Total transmitted received by a station. | UTF-8 | value | int |
| | | | id | string |
| | | | instance | IP |
| | | | job | string |
| | | | timestamp | float |
| | | | mac_sta | MAC |
| hostapd_sta_tx_rate_bps | Average throughput in bps transmitted by a station. | UTF-8 | value | int |
| | | | id | string |
| | | | instance | IP |
| | | | job | string |
| | | | timestamp | float |
| | | | mac_sta | MAC |

### 3.2.2 MPTCP-telemetry xApp

MPTCP-telemetry can be obtained from the open MPTCP sockets at the CPE and proxies by means of the Python API developed in D3.2 [3]. In order to expose this information to external agents such as the xApps, a REST API has been developed and described in D3.3 [21]. This REST API is a portable way to provide access to the structured data that the Python API returns, i.e., a JSON document.

Analogously to the Multi-WAT xApp, the MPTCP-Telemetry xApp publish the data gathered into a MPTCP-telemetry bucket in the data lake, as it is shown in Figure 3-16. However, as aforementioned, the xApp retrieves the telemetry from other nodes by submitting the HTTP requests defined as REST API endpoints. CPEs and Proxies use the same REST API and result format. The result of each request includes the telemetry of every open MPTCP socket and connection within the node. As each request is addressed to a specific CPE or Proxy address, the polling procedure could be configured with different periods, or even on demand.

Figure 3-17 shows the snapshot of the MPTCP-telemetry bucket, named MPTCP-CPE1 in the data lake. MPTCP socket related telemetry topics are ingested to the data lake successfully.

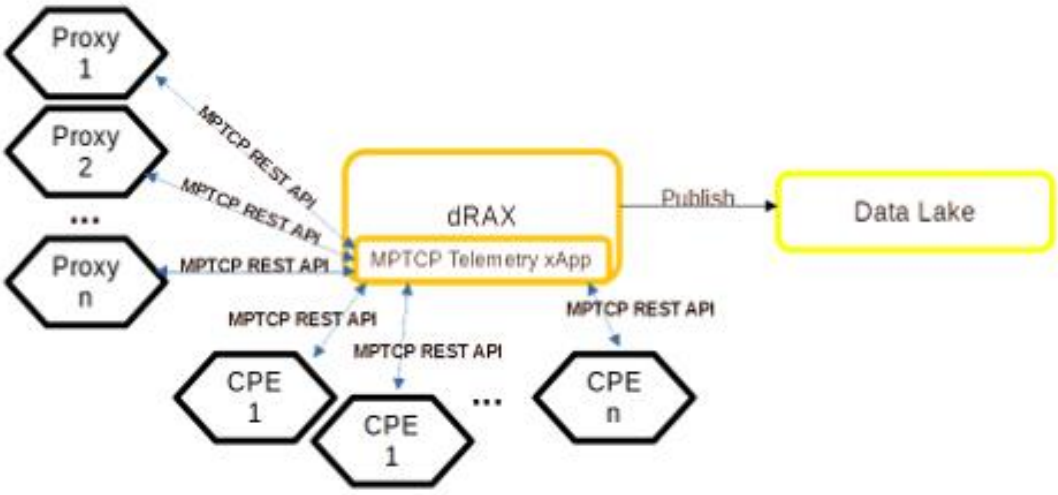



**Figure 3-16. MPTCP-telemetry xApp interaction scheme**

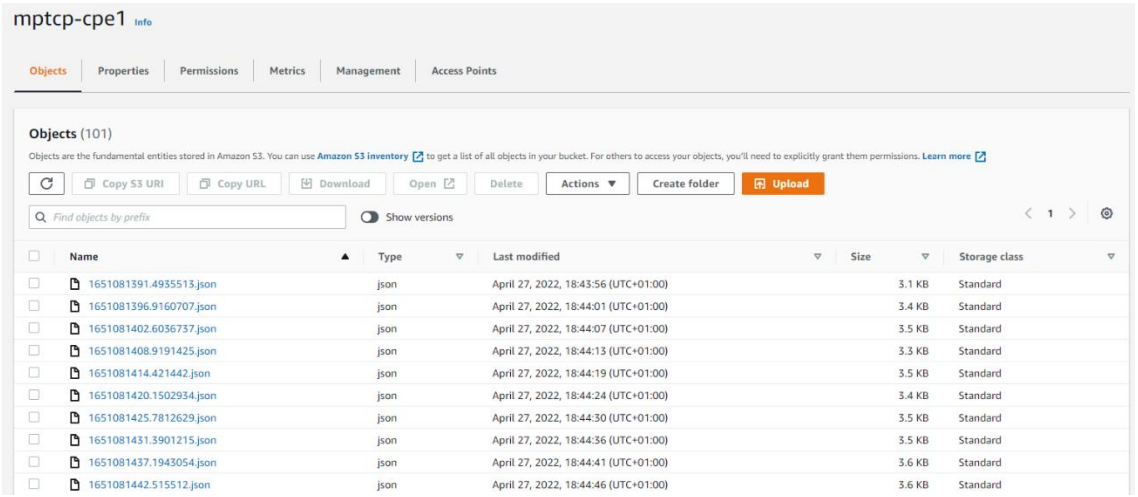



**Figure 3-17. Snapshot of the S3 bucket dedicated to MPTCP telemetry data**

Figure 3-18 shows the output of a GET request for the object named as the timestamp of the telemetry data inside the MPTCP telemetry bucket in the data lake.

In order to classify and extract metadata information inside the MPTCP telemetry bucket, a crawler is designed. The crawler generated a set of tables for each object in the MPTCP telemetry bucket where location, classification and last updated information are obtained, as shown in Figure 3-19.


```
{
    "host": "CPE-1",
    "sockets": [
        {
            "subflows": [
                {
                    "timestamp": 1651081520.9336054,
                    "src_ip": "10.1.2.1",
                    "src_port": 47585,
                    "dst_ip": "10.1.1.2",
                    "dst_port": 5001,
                    "timer": "(on,079ms,0)",
                    "inode": 119818,
                    "sk": 31,
                    "con_alg": "cubic",
                    "wscale": "7,7",
                    "rto": 207,
                    "rtt": 6.067,
                    "rtt_var": 1.115,
                    "mss": 1428,
                    "pmtu": 1500,
                    "rcvmss": 536,
                    "advmss": 1428,
                    "cwnd": 29,
                    "ssthresh": 18,
                    "bytes_sent": 26305536,
                    "bytes_retrans": 35120,
                    "bytes_acked": 26268989,
                    "segs_out": 18475,
                    "segs_in": 9493,
                    "data_segs_out": 18472,
                    "send_rate": 54606230.0,
                    "lastsnd": 129,
                    "lastrcv": 129759,
                    "lastack": 128,
                    "delivered": 18471,
                    "busy": "40710ms",
                    "sndbuf_limited": "582ms(1.4%)",
                    "unacked": 1,
                    "retrans": "0/28",
                    "dsack_dups": 27,
                    "rcv_space": 14280,
                    "rcv_ssthresh": 64108,
                    "minrtt": 0.302
                },
```


**Figure 3-18. Snapshot of the GET request for the timestamped object in the MPTCP telemetry data bucket**

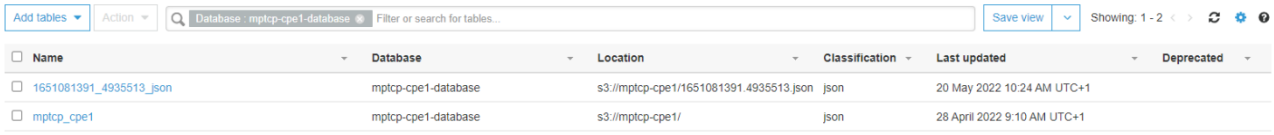



**Figure 3-19. Output tables of the MPTCP-specific crawler**

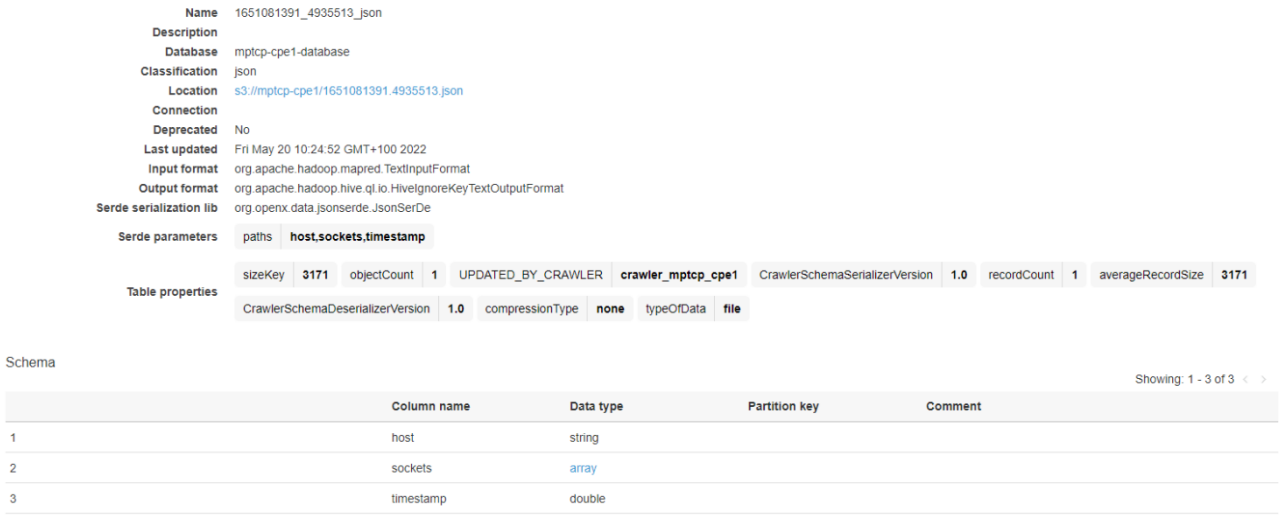



**Figure 3-20. Table details and schema of the timestamped object in the MPTCP telemetry bucket**

Figure 3-20 and Figure 3-21 show a schema of the table and schema details generated for the timestamped object, respectively.


**Figure 3-21. Schema details of the timestamped object in the MPTCP telemetry bucket**

Each object stored in the mptcp_telemetry bucket represents a MPTCP socket. The name of each object is the inode identifier of the socket. Each object is composed of a list of TCP subflows, with the information exposed in the table. Most part of the metadata information is described in the manual pages of iproute2's ss tool [iproute2-ss-tool-ref].

**Table 3-5. MPTCP Telemetry Data Details/Semantics**

| Description | Encoding Format | Metadata | Value Type |
|---|---|---|---|
| Information for subflows belonging to a given MPTCP socket. The subflow identifier is formed as follows: <mptcp-socket-inode>-<src_ip>-<src_port>-<dst-ip>-<dst_port>, where:<br>- <mptcp-socket-inode> is the inode number of the MPTCP socket, which coincides with the name of the bucket object.<br>- <src_ip> and <dst_ip> are the source and destination IP addresses of the TCP subflow respectively, expresed as four 8-bit decimal numbers separated by periods.<br>- <src_port> and <dst_port> represent the source and destination ports, expressed as an | JSON | advmss (advertised maximum segment size, in bytes) | String (integer value) |
| | | busy (Time busy sending data, in ms) | String (integer value, *plus "ms") |
| | | bytes_acked (bytes acked) | String (integer value) |
| | | bytes_sent (bytes sent) | String (integer value) |
| | | con_alg (congestion algorithm name) | string |
| | | cwnd (congestion | String |

| | | | |
|---|---|---|---|
| integer. | | window size, in MSS) | (integer value) |
| | | data_segs_out (The number of segments sent containing data) | String (integer value) |
| | | delivered (segment delivered, including retransmissions?) | String (integer value) |
| | | dst_ip (destination IP address) | string |
| | | dst_port (destination port) | String (integer value) |
| | | inode (inode number of the MPTCP socket) | Integer |
| | | lastack (time since the last ack received, in milliseconds) | String (integer value) |
| | | lastrcv (time since the last packet received, in milliseconds) | String (integer value) |
| | | lastsnd (time since the last packet sent, in milliseconds) | String (integer value) |
| | | minrtt (Minimum RTT) | String (float value) |
| | | mss (max segment size, expressed in bytes) | String (integer value) |
| | | pmtu (path MTU value, expressed in bytes) | String (integer value) |
| | | rcv_space (helper variable for TCP internal auto tuning socket receive buffer) | String (integer value) |
| | | rcv_ssthresh (Current window clamp) | String (integer value) |
| | | rcvmss (maximum segment size announced to peers as acceptable, in bytes) | String (integer value) |
| | | rto (re-transmission timeout value expressed as milliseconds) | String (integer value) |
| | | rtt (average round trip time expressed in milliseconds) | float |
| | | rtt_var | float |

| | | | |
|---|---|---|---|
| | | (mean deviation of rtt, expressed in milliseconds) | |
| | | segs_in (segments received) | String (integer value) |
| | | segs_out (segments sent out) | String (integer value) |
| | | send_rate (egress bps)* | float |
| | | sk (uuid of the socket?) | String (hexadecimal value) |
| | | src_ip (source IP address) | string |
| | | src_port (source port) | String (integer value) |
| | | ssthresh (tcp congestion window slow start threshold) | String (integer value) |
| | | timestamp (sampling instant expressed as in seconds since the Epoch time on January 1st, 1970 at UTC ) | float |
| | | wscale (send scale factor and receive scale factor) | String (two integers separated by a comma) |

### 3.2.3 Transport network telemetry

This section provides information related to the telemetry data from the transport network domain that is provided by the DSF telemetry system and stored in the Data Lake. This telemetry data is calculated in the form of KPIs by the DSF from information collected from network devices. Section 3.2.4 details the process of calculating these KPIs, which is then demonstrated in Section 3.2.5 on an experimental scenario for transport network data sources. The following table represents a description of the data schema for the main metadata content of the S3 objects in which the telemetry KPIs are stored.

**Table 3-6. Transport Network Telemetry Data Details/Semantics**

| Object | Description | Encoding Format | Metadata | Value Type |
|---|---|---|---|---|
| eMBB_Throughput_KPI | The effective data rate calculated as the number of bits per unit of time sent through a specific interface of a network device | JSON | throughput-in | String (bit/second) |
| | | | throughput-out | String (bit/second) |
| | | | duration | Integer (seconds) |
| | | | interface-name | String |
| URLLC_PacketLoss_KPI | The percentage of packets that fail to reach their destination | JSON | packet-loss-in | String (%) |
| | | | packet-loss-out | String (%) |

| | | | | |
|---|---|---|---|---|
| | during a period of time, calculated across the interfaces of network devices | | duration | Integer (seconds) |
| | | | interface-name | String |

Section 3.3.1.2 specifies the particular data model followed by the DSF telemetry system to write the telemetry KPIs into the Data Lake's S3 bucket. According with this particular data model, the Figure 3-22 and Figure 3-23 depict samples about the representation of the telemetry KPIs in each particular object of the transport telemetry bucket.

Once the new object is pushed to the Data Lake, the Data Lake classifies and extracts metadata information inside the object by using specific crawlers designed for the bucket and object. For the transport network telemetry data, a crawler is designed to generate a table for the available telemetry object in the transport telemetry bucket along with the location, classification, and last updated information of the object. Figure 3-24 and Figure 3-25 show a schema of the table and schema details generated for the Packet Loss and Throughput KPI objects in the transport network telemetry data.


```
{
  "ietf-yang-instance-data:instance-data-set": {
    "timestamp": "2022-06-03T13:56:00Z",
    "description": [
      "The effective input data rate calculated as the number of bits per
      unit of time sent through a specific network device interface.",
      "The effective output data rate calculated as the number of bits per
      unit of time sent through a specific network device interface."
    ],
    "name": "eMBB_Throughput_KPI",
    "content-schema": {
      "module": [
        "openconfig-interfaces-kpis@2022-05-13"
      ]
    },
    "content-data": {
      "openconfig-interfaces:interfaces": {
        "interface": [
          {
            "name": "Ethernet1",
            "openconfig-interfaces-kpis:throughput-kpi-notification": {
              "event-time": "2022-06-03T13:56:00Z",
              "throughput-kpi": {
                "duration": 10,
                "throughput-in": "290.0",
                "throughput-out": "300.0"
              }
            }
          }
        ]
      }
    }
  }
}
```


**Figure 3-22. A sample of throughput KPI**


```
{
  "ietf-yang-instance-data:instance-data-set": {
    "timestamp": "2022-06-03T08:48:10Z",
    "description": [
      "The percentage of input packets that fail to reach their destination
      in a period of time, measured across the interfaces of network devices.",
      "The percentage of output packets that fail to reach their destination
      in a period of time, measured across the interfaces of network devices."
    ],
    "name": "URLLC_PacketLoss_KPI",
    "content-schema": {
      "module": [
        "openconfig-interfaces-kpis@2022-05-13"
      ]
    },
    "content-data": {
      "openconfig-interfaces:interfaces": {
        "interface": [
          {
            "name": "Ethernet1",
            "openconfig-interfaces-kpis:packet-loss-kpi-notification": {
              "event-time": "2022-06-03T08:48:10Z",
              "packet-loss-kpi": {
                "duration": 10,
                "packet-loss-in": "3.0",
                "packet-loss-out": "2.5"
              }
            }
          }
        ]
      }
    }
  }
}
```


**Figure 3-23. A sample of packet loss KPI**

| | |
|---|---|
| Name | packet_loss_kpi_2022_06_03t08_48_10z_json |
| Description | |
| Database | transport_database |
| Classification | json |
| Location | s3://transport-telemetry/packet-loss-kpi@2022-06-03T08_48_10Z.json |
| Connection | |
| Deprecated | No |
| Last updated | Tue Jun 07 09:56:19 GMT+100 2022 |
| Input format | org.apache.hadoop.mapred.TextInputFormat |
| Output format | org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat |
| Serde serialization lib | org.openx.data.jsonserde.JsonSerDe |
| Serde parameters | paths **ietf-yang-instance-data:instance-data-set** |
| Table properties | sizeKey **871** UPDATED_BY_CRAWLER **crawler_5gclarity_transport** CrawlerSchemaSerializerVersion **1.0** recordCount **1** averageRecordSize **871** CrawlerSchemaDeserializerVersion **1.0** compressionType **none** typeOfData **file** |

Schema

Showing: 1 - 1 of 1

| | Column name | Data type | Partition key | Comment |
|---|---|---|---|---|
| 1 | ietf-yang-instance-data:instance-data-set | struct | | |

| | |
|---|---|
| Name | throughput_kpi_2022_06_03t13_56_00z_json |
| Description | |
| Database | transport_database |
| Classification | json |
| Location | s3://transport-telemetry/throughput-kpi@2022-06-03T13_56_00Z.json |
| Connection | |
| Deprecated | No |
| Last updated | Tue Jun 07 09:56:19 GMT+100 2022 |
| Input format | org.apache.hadoop.mapred.TextInputFormat |
| Output format | org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat |
| Serde serialization lib | org.openx.data.jsonserde.JsonSerDe |
| Serde parameters | paths **ietf-yang-instance-data:instance-data-set** |
| Table properties | sizeKey **839** UPDATED_BY_CRAWLER **crawler_5gclarity_transport** CrawlerSchemaSerializerVersion **1.0** recordCount **1** averageRecordSize **839** CrawlerSchemaDeserializerVersion **1.0** compressionType **none** typeOfData **file** |

Schema

Showing: 1 - 1 of 1

| | Column name | Data type | Partition key | Comment |
|---|---|---|---|---|
| 1 | ietf-yang-instance-data:instance-data-set | struct | | |

**Figure 3-24. Table details and schema of packet loss and throughput KPI objects in the transport network telemetry bucket**

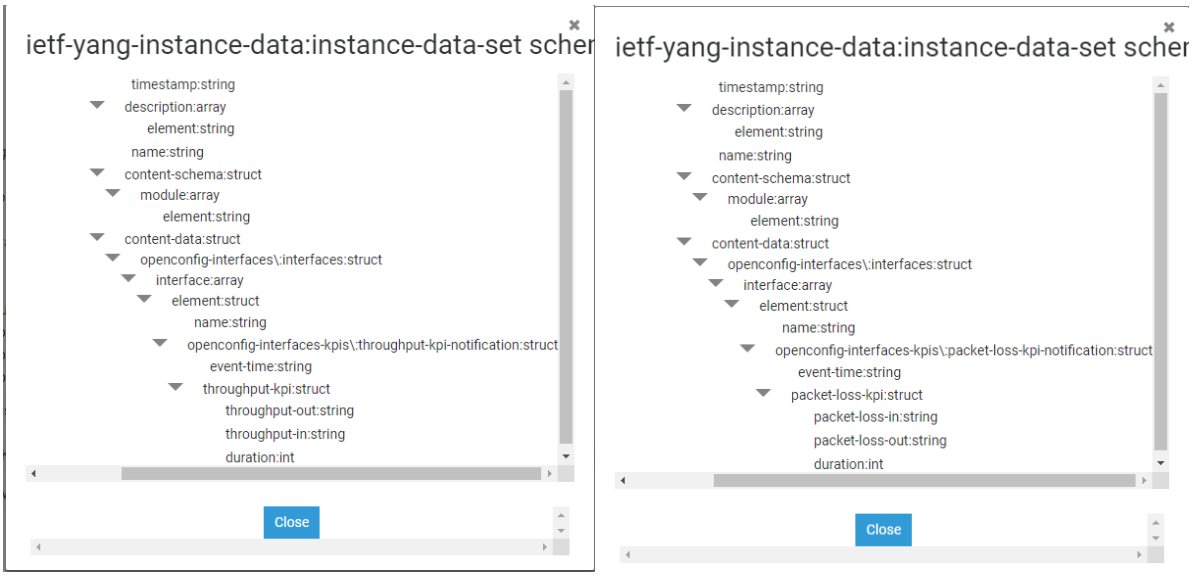



**Figure 3-25. Schema details of YANG instance data for Packet Loss (left) and throughput (right) KPI objects in the transport network telemetry bucket**

### 3.2.4 CIR telemetry

This section provides information on NLOS telemetry data put and get requests along with the telemetry

data details. More specifically, channel impulse response (CIR) data is ingested to the data lake via AWS API. A more detailed use case on utilizing the CIR telemetry data for NLOS identification ML algorithm is provided in Section 5. According to that use case, firstly, CIR telemetry data for a UE is collected by an SDR. Then, a script at SDR calls AWS API with "API URL" and "API Key" credentials exclusively related to the S3 bucket that is named "nlos-telemetry" and dedicated to the CIR telemetry data and pushes the CIR telemetry data to that S3 bucket.

Figure 3-26 shows the snapshot of the NLOS telemetry bucket in the data lake. A set of access point related telemetry topics are ingested to the data lake successfully.

Figure 3-27 shows the output of a GET request for the CIR telemetry data object inside the NLOS telemetry data bucket in the data lake.

In order to classify and extract metadata information inside the NLOS telemetry bucket, a crawler is designed. The crawler generated a table for the CIR telemetry object in the NLOS telemetry bucket where location, classification and last updated information are obtained. Figure 3-28 and Figure 3-29 show a schema of the table and schema details generated for CIR telemetry data object, respectively.

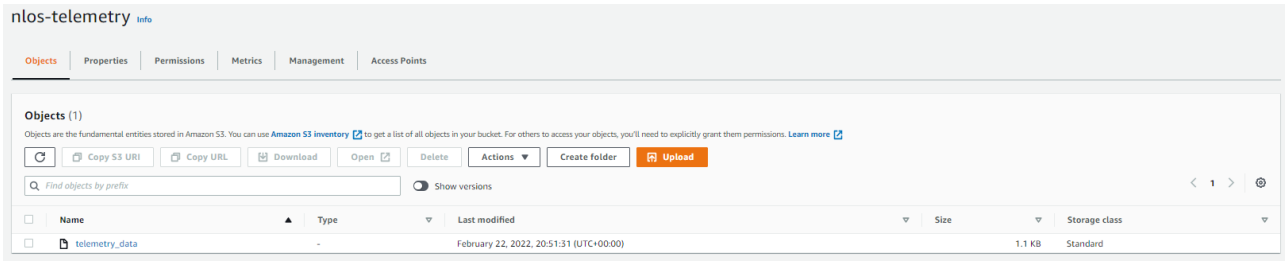



**Figure 3-26. Snapshot of the S3 bucket dedicated to NLOS/CIR telemetry data**


```
{"cir": [0.45496916849501023, 0.42763404874302086, 0.49140397719283196, 0.5439353794266972, 0.5936191547932792,
0.6285042105309225, 0.5300220325383228, 0.48770322737379407, 0.44746769351384885, 0.3760396242637832,
0.22599898519045827, 0.07291540720912874, 0.004050921072960863, 0.002523760344034917, 0.0013123534792575358,
0.0021680981646380907, 0.0020983685328779846, 0.001749939914956326, 0.0018711455048468488, 0.0016382249115236575,
0.0017715441678081223, 0.0015287570616852513, 0.0012806782997154228, 0.002123138638503306, 0.0014392156974232068,
0.0013346455793401281, 0.0013348288032528775, 0.0017178602007618222, 0.0016514564451280973, 0.002153660969737777,
0.00213803301860538, 0.002549461217047433, 0.0019216463676471 49, 0.0025708525390631316, 0.0022024643192271546,
0.002885022827132193, 0.003342916727553222, 0.0035047066345861126, 0.012450987350130983, 0.1537612673306062,
0.5127069375374423, 0.8428670614592373, 1.0, 0.9971995896050985, 0.9798621061752552, 0.9649957354810158,
0.8739697838576744, 0.6982632265109557, 0.44642594699766264, 0.3553681696883677], "link_condition": "nlos"}
```


**Figure 3-27. Snapshot of the GET request for CIR telemetry data in the NLOS telemetry data bucket**

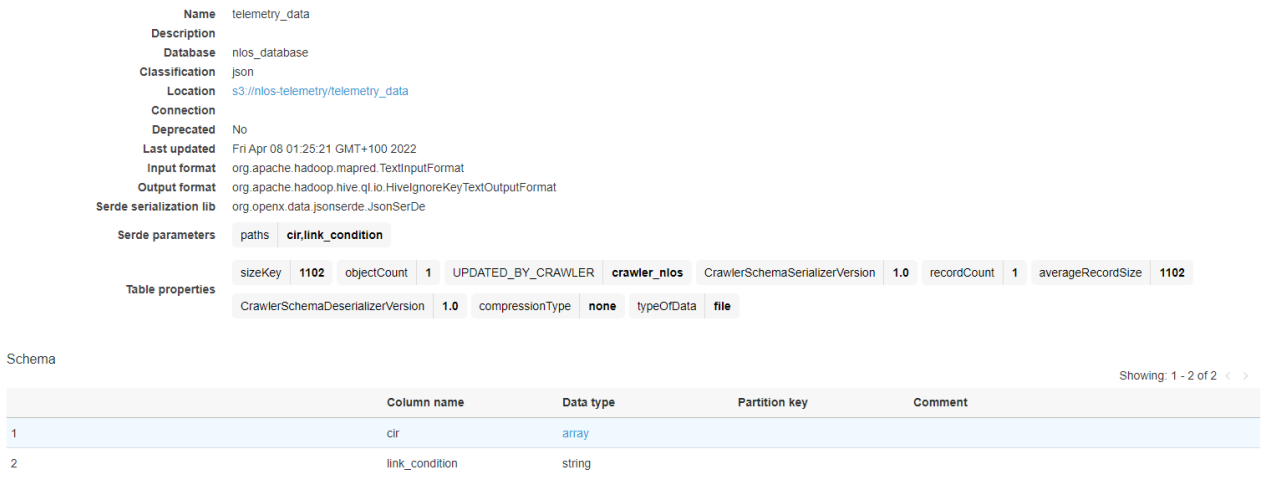



**Figure 3-28. Table details and schema of CIR telemetry data object in the NLOS telemetry bucket**

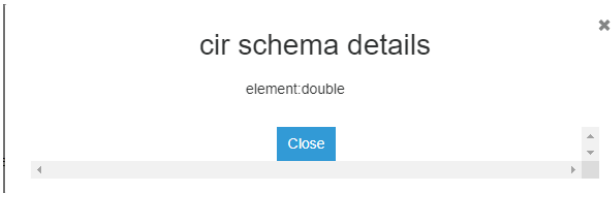



**Figure 3-29. Schema details of CIR in the NLOS telemetry bucket**

**Table 3-7. UE CIR Telemetry Data Details/Semantics**

| Object | Description | Encoding Format | Metadata | Value Type |
|---|---|---|---|---|
| Telemetry_data | The latest measured CIR performed by AP and the true link condition (for the pupose of evaluation) | JSON | cir | String (float value) |
| | | | link_condition | string |

## 3.3 Transport network data sources in DSF

An important type of data source identified in the network infrastructure domain is the telemetry-based network devices. As introduced in previous deliverables, in networking world model-based streaming telemetry provides a mechanism to collect data of interest from remote data sources (e.g., configuration and operational data from network devices) whose information is structured according to formal data models, and to transmit it in a structured format to remote destinations for monitoring. This mechanism is specifically tailored for the automatic tuning of the network based on real-time data, and crucial for network seamless operation. A higher frequency of fine-grained data collection available through telemetry enables better monitoring performance and, therefore, better troubleshooting. It can help achieve better performance across the whole network infrastructure, such as more efficient bandwidth utilization, comprehensive risk assessment and control, and greater scalability, among other things. Thus, streaming telemetry converts the monitoring process into a data analytic proposition that enables a fast extraction and analysis of massive data to improve decision-making. Therefore, model-based streaming telemetry is gaining attention as a monitoring mechanism for network devices, mainly relying on YANG data models and management protocols such as gNMI or NETCONF.

In the scope of 5G-CLARITY, we consider the telemetry-based network devices as potential data sources for the transport network domain. In this sense, the main goal of this section is to showcase the capabilities of the 5G-CLARITY DSF telemetry system to address the collection, processing and aggregation of telemetry for this type of transport network data sources. There is a developed prototype of the DSF framework, which is available as an open-source project on GitHub[7].

The structure of this section is organized as follows. Subsection 3.2.1 provides insights on how discover the context information of the telemetry-based network devices from the DSF telemetry system in order to expose their related capabilities. Subsection 3.2.2 describes the mechanisms supported by the DSF in order to register network devices as data sources and expose their available capabilities. Subsection 3.2.3 presents the solution supported by the DSF in order to collect the telemetry data from the registered network devices based on the gNMI management protocol and YANG data modelling language. Subsection 3.2.4 describes the mechanisms for aggregating the telemetry data that leads to the calculation of new performance metrics in the form of KPIs. Finally, the subsection 3.2.5 presents a testbed scenario in order to validate how

[7] https://github.com/giros-dit/semantic-data-aggregator

transport network devices can be integrated with DSF and how the telemetry data can be collected and aggregated accordingly.

### 3.3.1 Context information of telemetry-based network devices

As part of the integration of transport network devices as data sources within the DSF telemetry system, they must be registered and expose their capabilities. The operators of the DSF framework require these capabilities in order to discover what data are available in the network devices and how these data can be ingested through the DSF. In this regard, the DSF collects context information that describes the capabilities of devices with support for model-driven telemetry that are present in the transport network. To model this context information, the DSF leverages the NGSI-LD standard [27].

Figure 3-29 depicts the NGSI-LD information model that captures context information that represents the telemetry capabilities associated with a YANG-modelled network device. The *Device* entity captures the main features of the network device such as the vendor, the name of the model, and the software version. Each of the YANG modules supported by the network device is represented by the *Module* entity. This entity includes a set of properties that uniquely identify a YANG module: module name, revision number, and namespace. Additionally, the implementation details of each YANG module by a given device is represented with the relationship *implementedBy*. This relationship may contain information regarding YANG features or deviations that are applied to the linked YANG module.

The context information related to the YANG modules available in the network device can be obtained from the Capability Discovery functionality by following the management protocol specification. When the network device supports the gNMI management protocol, the *Gnmi* entity specifies the address and port of the endpoint associated to the gNMI service, the protocol version as well as the supported encoding formats, e.g., JSON-IETF. If the network device supports the NETCONF management protocol, the *Netconf* entity also specifies the address and port of the endpoint associated to the NETCONF service. In addition, the *Netconf* entity includes information related to additional NETCONF capabilities supported by the network device, such as XPath-based filtering support in protocol operations and the capability to send notifications to subscribers.

Lastly, the NGSI-LD information model includes a *Credentials* entity that represents basic authentication credentials with username and password. Such credentials can be configured for either of the two network management protocols that may be supported by the device. This configuration is expressed in the model by means of the *authenticates* NGSI-LD relationship. Note that this context information represents sensitive data, and therefore, only read access to it must be limited to authorized users. The mechanism that enables access control to certain context information will be implemented in future releases of the DSF.

### 3.3.2 Telemetry-based network device registration and discovery of capabilities

The DSF features a component named the Telemetry Explorer. This component is a microservice that enables, first, the registration of network devices as data sources, and second, exposing the capabilities available in the registered network device. To perform these operations, DSF operators leverage the standard NGSI-LD API to interact with the Scorpio NGSI-LD Context Information Broker [28] as depicted in Figure 3-31.

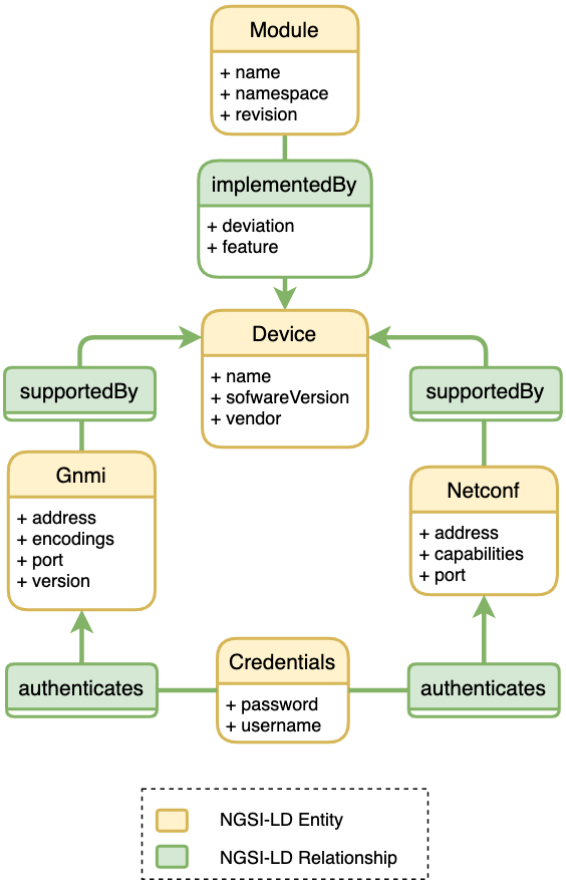



**Figure 3-30. NGSI-LD information model for telemetry-based network device**

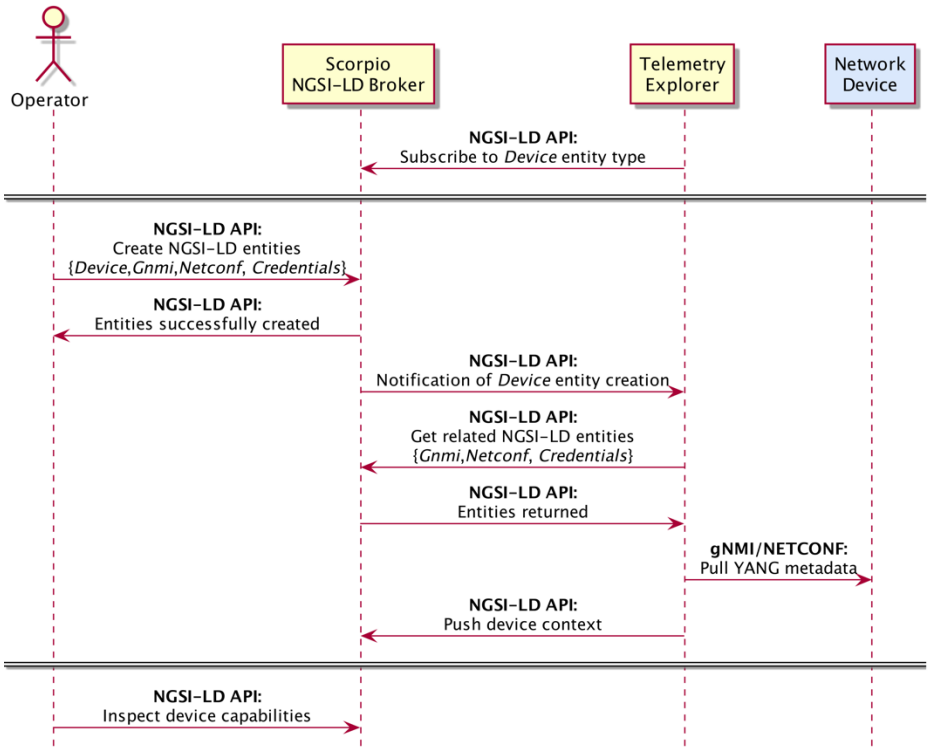



**Figure 3-31. Registration of network device and discovery of capabilities through the NGSI-LD API**

To provide the registration of new devices of a transport network, the Telemetry Explorer first subscribes to

updates on context information related to the *Device* entity type. Next, DSF operators can trigger the registration of a network device by creating the corresponding *Device* entity in the Context Broker. Note that other entity types associated with the network management protocols supported in the device (i.e., *Gnmi*, *Netconf*, and *Credentials*) must be provided as well. Once the creation of the *Device* entity is successful, the Context Broker sends a notification to the Telemetry Explorer indicating that a new network device is available. Consequently, the Telemetry Explorer collects context information about network management protocols to establish a connection with the network device. Telemetry Explorer leverages this connection to collect metadata, such as supported YANG modules or network protocol server details, from the network device. The retrieved metadata is transformed into context information as new *Module* entities and updates on the existing *Netconf* and *Gnmi* entities (e.g., list of encoding formats supported by the gNMI server or the NETCONF-related capabilities).

Once all context information related to the registered network device has been stored in the Context Broker, DSF operators can move on to discover the capabilities of the device. In the same way as with the registration operation, the capability discovery operation is performed by sending queries through the NGSI-LD API to navigate the property graph that was modelled as shown in Section 3.2.1.

### 3.3.3 Collecting telemetry data from the network devices

This subsection addresses how the DSF collects model-based telemetry data from network devices. As described in deliverable D4.2 [1], the DSF leverages Apache NiFi [29] and Apache Flink [30] big data tools to collect, aggregate, and deliver telemetry data. In this sense, to collect telemetry data related to network devices from the DSF framework, two data pipeline steps are chained as depicted in Figure 3-32.

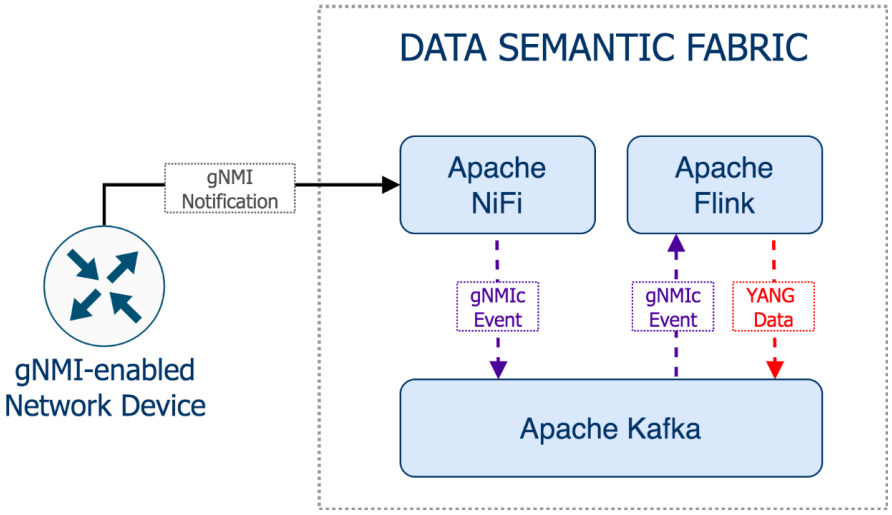



**Figure 3-32. Data pipeline that collects telemetry data from a gNMI-enabled network device**

The DSF collects the telemetry data from the network devices using the gNMI management protocol. To do this, a gNMI CLI (Command Line Interface) client called gNMIc [31] is used. This gNMIc client has full support for gNMI RPC operations, including the operation to subscribe to telemetry data. The subscription operation can be on-change mode or based on sampling interval. Then, the collection process basically consists of subscribing to a specific XPath – either in on-change mode or sample mode – from the telemetry-based network device. This XPath is the selector for the specific YANG data node(s) from the YANG model(s) supported by the target network device. To orchestrate this collection process, an Apache NiFi flow is automated, which creates a gNMI subscription to a specific YANG-based telemetry data from the network device and writes the notification events that are generated into a particular Kafka topic.

Once the notification events are written into Kafka, the second step of the chain comes into play. One of the

principles of the DSF is that all data handled internally must be structured as per YANG data models, taking advantage of the semantics, flexibility, and scalability provided by the YANG-modelled data. In addition, having the data structured according to a YANG data model allows improving interoperability between data sources and consumers. The problem is that although the telemetry collected from network devices is modelled according to YANG, the notification events generated by the subscriptions made by the gNMIc client are not strictly structured according to the original YANG data model. In order to cover this problem, the DSF implements an Apache Flink application for structuring the gNMIc event notification into the original YANG data model, and then wraps the data into a YANG notification using a particular encoding format, following a similar approach to NETCONF and RESTCONF event notifications as defined in RFC 5277 [32]. This application is implemented based on the YANG Tools [33] project provided by OpenDayLight. YANG Tools is a set of libraries and tooling that supports the use of YANG in Java programming language and allows normalizing data according to a specific YANG data model and serializing that YANG-modelled data into a JSON or XML encoding format. For the sake of interoperability, we will use the JSON-IETF format, which is a standardized JSON encoding format for representing YANG-modelled data as defined in RFC 7951 [34]. Then, the DSF executes an Apache Flink application that reads the event notification generated by gNMIc from an internal Kafka topic and structure the collected telemetry data into a YANG notification. Once data have been packaged into the YANG notification, it is sent back to another Kafka topic where interested data consumers can subscribe.

```
module: openconfig-interfaces-notification-wrapper
  +--ro notification
     +--ro event-time?   yang:date-and-time
     +--ro interfaces
        +--ro interface* [name]
           +--ro name              -> ../config/name
           +--ro config
           |  ...
           +--ro state
           |  ...
           +--ro hold-time
           |  ...
           +--ro subinterfaces
           |  ...
```

**Figure 3-33. YANG tree representation on top of the YANG module for wrapping into notifications the state and configuration of network device interfaces**

Figure 3-33 depicts an example for the partial tree representation of the YANG module for wrapping into notifications the state and configuration statistics about the network device interfaces as covered by the OpenConfig YANG module named *openconfig-interfaces* [35].

### 3.3.4 Calculating telemetry KPIs

One of the purposes of the DSF framework is the ability to perform transformations and aggregations over the collected telemetry data. These data transformation and aggregation processes are programmed as Apache Flink applications that, through operations applied to the related telemetry data such as field mapping, filtering or windowed aggregations, calculate new resulting information. These aggregations over the telemetry data enable to compute new performance metrics in form of KPIs. In this regard, the YANG module named *openconfig-interfaces* [36] supported by the Arista's cEOS routers (i.e., the network devices used in the validation scenario in section 3.2.5) provides relevant telemetry data, such as the operational state data of the router interfaces to calculate the following KPIs:

- **Throughput (bit/second)**: The effective data rate calculated as the number of bits per unit of time sent through a specific network device interface. The *openconfig-interfaces* module provides

telemetry data such as the total bytes received and transmitted through a given interface in the form of counters, which is useful for calculating this KPI.

- **Packet Loss Rate (%)**: The percentage of packets that fail to reach their destination in a period of time, measured across the interfaces of network devices. The *openconfig-interfaces* module provides information such as the number of incoming and outgoing packets dropped from a given interface which, combined with the total number of packets received and transmitted through the interfaces during a period of time, is useful for calculating this KPI.

In this sense, the semantic information related to the new calculated KPIs must be considered. In this regard, an extension of the original *openconfig-interfaces* YANG model has been proposed taking advantage of the evolutionary capabilities of YANG data modelling. By means of the *augment* statement, a new YANG module called *openconfig-interfaces-kpis* can insert additional information (i.e., the KPIs) into the original YANG model. In this sense, each KPI is defined as a notification container within the related augmented YANG module. Figure 3-34 depicts the YANG tree representation for the augmentation of the original YANG model by extending the information related to a specific device interface with new data nodes in the form of KPI notifications. The notifications represent the metadata corresponding to the throughput and packet loss aggregated KPIs computed from the telemetry data retrieved by the gNMI subscription. Every notification related to KPI is composed by its own value and by the date and time in which it was calculated.

```
module: openconfig-interfaces-kpis

  augment /oc-if:interfaces/oc-if:interface:
    +--ro throughput-kpi-notification
    |  +--ro event-time?       yang:date-and-time
    |  +--ro throughput-kpi
    |     +--ro throughput-in?    per-decimal
    |     +--ro throughput-out?   per-decimal
    |     +--ro duration?         yang:timestamp
    +--ro packet-loss-kpi-notification
       +--ro event-time?        yang:date-and-time
       +--ro packet-loss-kpi
          +--ro packet-loss-in?    per-decimal
          +--ro packet-loss-out?   per-decimal
          +--ro duration?          yang:timestamp
```

**Figure 3-34. YANG tree representation for the augmentation of the openconfig-interfaces YANG model with the aggregated KPIs**

The KPI value includes both the incoming and outgoing calculated value on the corresponding interface, as well as the duration interval in which the KPI was computed.

The Figure 3-35 depicts the data pipeline process for the KPIs calculation in the DSF framework. First, an Apache Flink application reads from an input Kafka topic a gNMI events related to the *openconfig-interfaces* model already structured according to a YANG notification (i.e., the notification normalization process seen in the previous section). Then, the same Flink application reads the fields needed from consecutive gNMI events, which are aggregated according to a time window, and calculates the related KPIs. Finally, the application structure the KPIs according to the *openconfig-interfaces-kpis* model. Once the KPIs have been packaged into the related YANG model, they are sent back to another output Kafka topic where interested data consumers can subscribe.

### 3.3.5 Experimental scenario for transport network data sources

An experimental scenario serving as a *Proof of Concept* (PoC) for transport network data sources has been implemented in order to demonstrate the DSF capabilities to collect, aggregate, and process telemetry data from network devices.

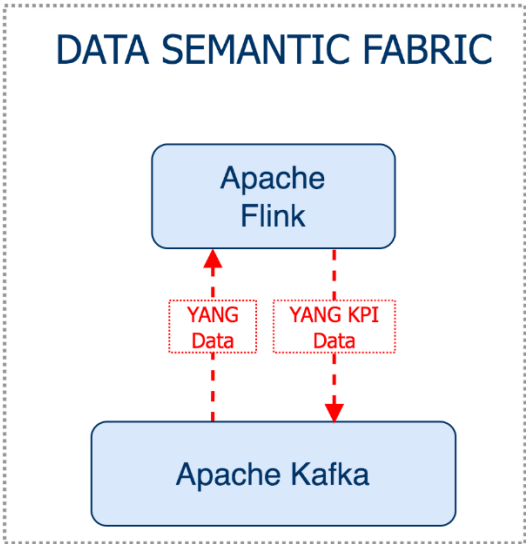



**Figure 3-35. Data pipeline for the aggregation of telemetry KPIs**

The implementation of the testbed scenario is based on a virtual deployment on the OpenStack platform. The prototype design, which is depicted in Figure 3-36, is composed by three main components:

Data Semantic Fabric (DSF) framework for applying streaming telemetry mechanisms to collect telemetry data from transport network data sources, aggregate the information and apply transformations, and deliver the result information to interested data consumers. DSF is a microservice-based framework deployed on a virtual machine of the OpenStack scenario (i.e., VM2 in Figure 3-36).

Network devices that support YANG-model based configuration management and streaming telemetry over network management protocols such as gNMI and NETCONF. For the PoC scenario, a virtualized router model Arista containerized EOS [37] (Arista cEOS) from the vendor Arista Networks has been chosen. This particular Arista model is a containerized router version to be deployed on a container runtime engine, such as Docker. In addition, the Arista's cEOS router model support YANG data modelling language and telemetry based on the gNMI and NETCONF protocols. In the prototype design, two instances of cEOS routers are deployed as Docker containers in the virtual machine where the DSF framework is running (i.e., the VM2 in Figure 3-36). For the experimental scenario, the telemetry data from Arista's cEOS routers is collected via the gNMI management protocol.

Traffic generator service to inject synthetic traffic data on the network devices. The solution *Ixia BreakingPoint* [38] from Keysight is selected. This solution allows generating a multitude of different traffic profiles such as ICMP or HTTP traffic flows. In addition, it allows limiting the data rate generated by the interface in different time intervals. The traffic generator consists of one virtual system controller (i.e., the BreakingPoint vController component in Figure 3-36) and the virtual blades (i.e., the vBlade module within the VM1 in Figure 3-36). The vBlades are the traffic generation modules that send and receive traffic. The vController is the management entity for orchestrating the vBlades. The vController provides a user interface to manage the system. In the validation scenario, a single vBlade module is used to generate traffic in the network devices.

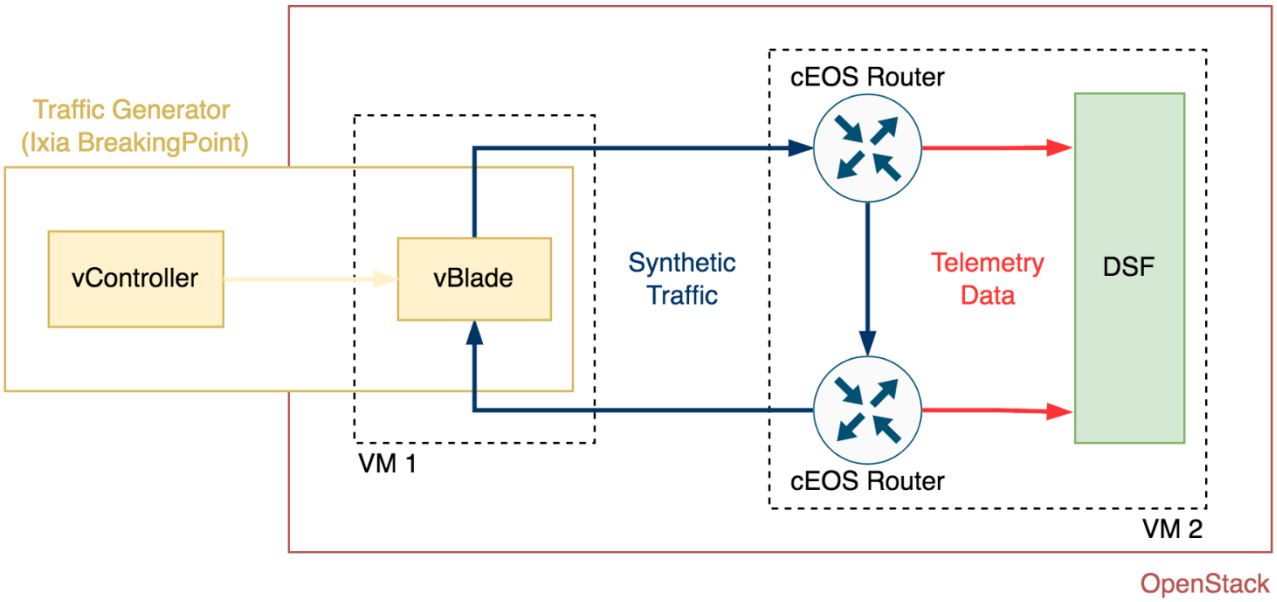



**Figure 3-36. Prototype scenario related to transport network telemetry**

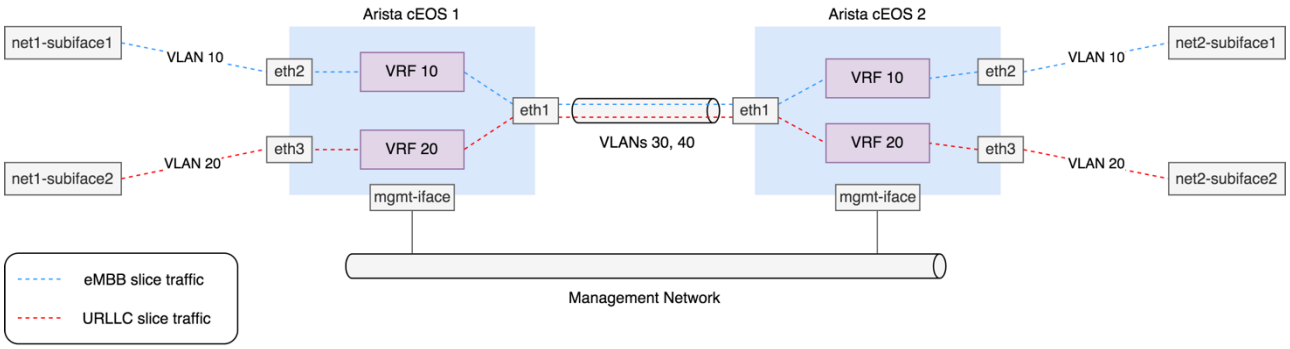



**Figure 3-37. Logical interconnection between the transport network devices in the experimental scenario**

In the validation environment, the capability of the DSF to apply data transformations over the telemetry data is demonstrated in order to compute the two different KPIs introduced in the previous subsection: Throughput and Packet Loss. To calculate these KPIs in the testbed scenario, the following assumptions are taken into account:

- Two different traffic patterns will be generated from the traffic generator. To uniquely identify traffic patterns, each one will be tagged with a specific VLAN ID.
- Each traffic pattern isolated for each particular VLAN corresponds to a particular 5G-CLARITY network slice: eMBB traffic and URLLC traffic.
- For the eMBB traffic, the Throughput KPI is computed.
- For the URLLC traffic, the Packet Loss Rate is computed.

Figure 3-37 depicts a logical diagram about the networking configuration and operation within the forwarding plane of the transport network devices deployed in the experimental scenario.

The following explains the complete cycle of the traffic since it is generated from the source virtual machine (i.e., the VM1 with the traffic generator) until the traffic returns to the same point, passing through all the intermediate points. Going back to the Figure 3-37, the traffic generated from VM1 is forwarded to VM2 through an ingress internal network that interconnects both virtual machines. This traffic is tagged by the

traffic generator for a specific VLAN with ID 10 or ID 20. The traffic with VLAN ID 10 corresponds to the eMBB slice and the traffic with VLAN ID 20 corresponds to URLLC slice. When traffic arrives at VM2 through the incoming interface, it is segmented to the corresponding subinterface specific to the regarding slice (e.g., for the eMBB slice the incoming traffic arrives through the *net1-subiface1* subinterface as show in Figure 3-37). Then, the traffic is forwarded to the first router device (i.e., Arista cEOS 1) working as the next-hop gateway. It's important to highlight that the Arista's cEOS routers work as layer 3 switches, whose interfaces can be enabled as switching ports or routing ports. Then, the traffic arrives to the router through a specific access switching port depending on the VLAN ID. The Arista's cEOS routers implements the VRF (Virtual Routing and Forwarding) mechanism that enables to route the traffic between VLANs. This VRF mechanism is needed for routing the traffic from the VLAN 10 and VLAN 20. Each VRF (i.e., VRFs 10 and VRF 20) creates a specific routing and forwarding information base for each VLAN traffic, allowing to isolate them. Once the traffic is isolated per VRF within the first hop router, the traffic needs to be forwarding to the second hop router (i.e., Arista cEOS 2). In such a situation, the traffic is forwarded from the first router for each specific VRF but through a single output interface. Apart from the VLANs 10 and 20, the routers define two additional VLANs (i.e., VLANs with ID 30 and ID 40) in order to enable the forwarding from the VRFs. Then, the VLAN 30 corresponds to the VRF 10 and the VLAN 40 corresponds to the VRF 20. This is essential since VRF only enables routing between different VLANs and also to isolate traffic between the two routers. All these VLANs are shared between the two routers. The interconnection between both routers is a trunk link that allows to switch both VLAN 30 and VLAN 40 traffic. When the traffic arrives to the second hop router, the router applies the VRF mechanism in order to forward the isolated traffic to the corresponding egress VLAN (i.e., VLAN 10 or VLAN 20). Once the traffic leaves the second hop router, it is forwarded to the corresponding egress subinterface of the VM2 depending on the VLAN tag, in order to finally send the traffic back to VM1 through another egress internal network that interconnects both virtual machines to complete the full traffic generation cycle.

In addition, as the Figure 3-37 shown, the router devices have an additional management interface used to manage the device configuration either by the CLI or by the NETCONF or gNMI management protocols and also to collect the required telemetry data.

Taking the above considerations into account, the steps of the complete telemetry pipeline process are analyzed below, from the injection of traffic in the transport network scenario to obtaining the resulting KPI. The demonstration explanation is particularized for validating the case of the Throughput KPI.

In the *Ixia BreakingPoint* traffic generator we create a simple traffic pattern that simulate the injection of ICMP request packets. In this traffic generation test, a constant data rate of 1 Mbps is configured. The Figure 3-38 displays the aforementioned traffic generation test when it has been running for 340 seconds. It can be seen how the traffic sending data rate (i.e., the green line in the data-time function) coincides with the traffic receiving data rate (i.e., the orange line in the data-time function) during the timeline, fluctuating a little below the threshold of 1 Mbps.

As the demonstration is particularized for compute the Throughput KPI, the traffic has been generated for the eMBB slice. Then, the traffic has been tagged with VLAN ID 10. During the traffic generation, we can check how the traffic is been forwarded throughout the router devices as described in Figure 3-37. In this situation, we can check how the counters corresponding to the network device interfaces are increasing. It can be checked directly by collecting telemetry data from the network devices. In such a case, we can create gNMI subscriptions to the interface-related statistics across the network devices. Figure 3-39 depicts a sample of notification event received from the gNMIc client and written into a Kafka topic after a subscription has been created for the incoming bytes on Ethernet2 interface of the Arista cEOS 1 router. The sample shows an incremental counter with the current number of incoming bytes through the interface. The

subscription has been generated in a sample mode with a sampling interval of 10 seconds.


**Figure 3-38. Partial results of the Ixia BreakingPoint test**


```
[
   {
        "name": "default-1655469998",
        "tags": {
            "interface_name": "Ethernet2",
            "source": "localhost:6030",
            "subscription-name": "default-1655469998"
        },
        "timestamp": 1655470713219757800,
        "values": {
            "/interfaces/interface/state/counters/in-octets": 342803006
        }
    }
]
```


**Figure 3-39. A sample of gNMIc-related event notification for subscription on incoming traffic through a network device interface**

As commented in Section 3.2.3, the DSF framework structure the events received from the gNMIc subscription into a YANG data model that enables wrapping the telemetry data into a notification particularized for the data available on the original data model (i.e., the *openconfig-interfaces* model from where the "in-octets" statistic is retrieved). Figure 3-40 depicts the previous sample of the gNMIc event already normalized according to the notification wrapper YANG data model and written into another Kafka topic. It is important to highlight that the resulting notification, in addition to showing the value of the incoming bytes through the Ethernet2 interface, transforms the timestamp of the event generated by gNMIc into a date and time format.

Once gNMI notifications are normalized according to the notification wrapper YANG model, the computation of the Throughput KPI can be performed. Since a subscription has been created in sample mode with a sampling interval of 10 seconds, the Throughput KPI could be calculated between two consecutive gNMI notifications every 10 seconds of time. Figure 3-41 depicts a sample of notification about the incoming throughput calculated through the Ethernet2 interface during the last 10 seconds. As described in Section 3.2.4, the KPI is computed by an Apache Flink application that enables to aggregate consecutive events from the streaming data read from the input Kafka topic where the notification events are written. When the notification events are aggregated, and the Throughput KPI is computed and normalized according to the KPI notification YANG model, the resulting telemetry KPI notification is written to an output Kafka topic. It is important to highlight that the resulting KPI notification, in addition to showing the value of the current

incoming throughput in bits per second through the Ethernet2 interface and the duration of the calculation interval in seconds, includes the date and time when the KPI was computed.


```
{
  "openconfig-interfaces-notification-wrapper:notification": {
    "event-time": "2022-06-17T14:58:33Z",
    "interfaces": {
      "interface": [
        {
          "name": "Ethernet2",
          "state": {
            "counters": {
              "in-octets": "379440118"
            }
          }
        }
      ]
    }
  }
}
```


**Figure 3-40. gNMIc-related event normalized according to a gNMI notification**


```
{
    "openconfig-interfaces:interfaces": {
        "interface": [
            {
                "name": "Ethernet2",
                "openconfig-interfaces-kpis:throughput-kpi-notification": {
                    "event-time": "2022-06-17T12:59:58Z",
                    "throughput-kpi": {
                        "duration": 10,
                        "throughput-in": "949756.8"
                    }
                }
            }
        ]
    }
}
```


**Figure 3-41. Incoming throughput KPI notification**

Finally, as will be described in the Section 3.3.1.2, the DSF telemetry system provides the possibility to structure the Throughput KPI notification according to a particular YANG data model for storing the telemetry data values as an instance data file format within an external storage system such as a Data Lake. Figure 3-42 depicts the Throughput KPI notification previously computed and showed in Figure 3-41 normalized according to this particular YANG data model. Then, the resulting KPI telemetry data normalized according to this YANG model are written into another Kafka topic where prospective consumer can access them.

These DSF-related operations about how to trigger a data pipeline to calculate the KPIs and how to dispatch and write the resulting KPI telemetry data, which follows the instance data file format shown in Figure 3-42, in the Data Lake system when consuming telemetry from the transport network are detailed in Section 3.3.1.

```
{
  "ietf-yang-instance-data:instance-data-set": {
    "content-schema": {
      "module": [
        "openconfig-interfaces-kpis@2022-05-13"
      ]
    },
    "name": "eMBB_Throughput_KPI",
    "timestamp": "2022-06-17T12:59:58Z",
    "description": [
      "The effective input data rate calculated as the number of bits per unit of
       time sent through a specific network device interface.",
      "The effective output data rate calculated as the number of bits per unit of
       time sent through a specific network device interface."
    ],
    "content-data": {
      "openconfig-interfaces:interfaces": {
        "interface": [
          {
            "name": "Ethernet2",
            "openconfig-interfaces-kpis:throughput-kpi-notification": {
              "event-time": "2022-06-17T12:59:58Z",
              "throughput-kpi": {
                "duration": 10,
                "throughput-in": "949756.8"
              }
            }
          }
        ]
      }
    }
  }
}
```


**Figure 3-42. Throughput KPI notification structured according to the YANG instance data file format**

## 3.4 Integration between DSF and Data Lake

This section details the integration between the DSF and the Data Lake, which has been addressed for the following two possible configurations:

**DSF → Data Lake**: The DSF enables transport network devices to be incorporated as data sources for the Data Lake. This integration has been completed, and details are provided in Section 3.4.1.

**Data Lake → DSF**: The Data Lake platform is registered as a data source from which the DSF can pull AWS S3 objects for further processing. This integration has been partially implemented, and details are provided in Section 3.4.2.

This integration has been validated through the implementation of a prototype, which can be found in GitHub[8].

### 3.4.1 DSF as the Data Source to Data Lake

This section covers the two main aspects of the DSF-to-Data-Lake integration. First, the management of data pipelines within the DSF to collect and aggregate telemetry data from transport network devices, and eventually, store the results in the Data Lake. Second, the implementation details on how the aggregated data are delivered from the DSF to the Data Lake platform by using the Flink and NiFi tools.

#### 3.4.1.1.1 Creation of Data Pipelines in the DSF

The DSF enables operators to create data pipelines by means of the standard NGSI-LD API. Following the same approach as with the definition of NGSI-LD information models for network devices, we devise NGSI-LD information models to represent data pipelines within the DSF. By defining the steps of a data pipeline as

[8] https://github.com/giros-dit/semantic-data-aggregator

NGSI-LD entities, DSF operators can compose in a declarative fashion a property graph that determines the structure of the pipeline. For the use case that the DSF addresses in 5G-CLARITY, we propose the NGSI-LD information model depicted in Figure 3-43.

This information model depicts a data source, a data pipeline, and a data consumer. The *Device* entity represents a data source from the transport network as described in Section 3.2.1. The data consumer is represented by the *DataLake* entity, which renders the entry point to the Data Lake platform, i.e., the API Gateway, including the required parameters to establish a connection, namely, the URI and the region. Lastly, the information model captures a data pipeline that is composed of three main steps: (i) collection of telemetry data from a network device through gNMI protocol; (ii) aggregation of telemetry data to compute network interface KPIs; and (iii) storing aggregated KPIs as new files (i.e., S3 objects) in buckets of the Data Lake platform. To represent each step of the pipeline, the following NGSI-LD entity types are defined:

***GnmiCollector***: Models the configuration of a gNMI subscription to the target network device, which is specified in the graph by setting the *hasInput* relationship to the entity that represents the device. The *GnmiCollector* entity allows to specify values for gNMI subscription parameters as defined in the official specification [31] such as the sampling interval for periodic subscriptions.

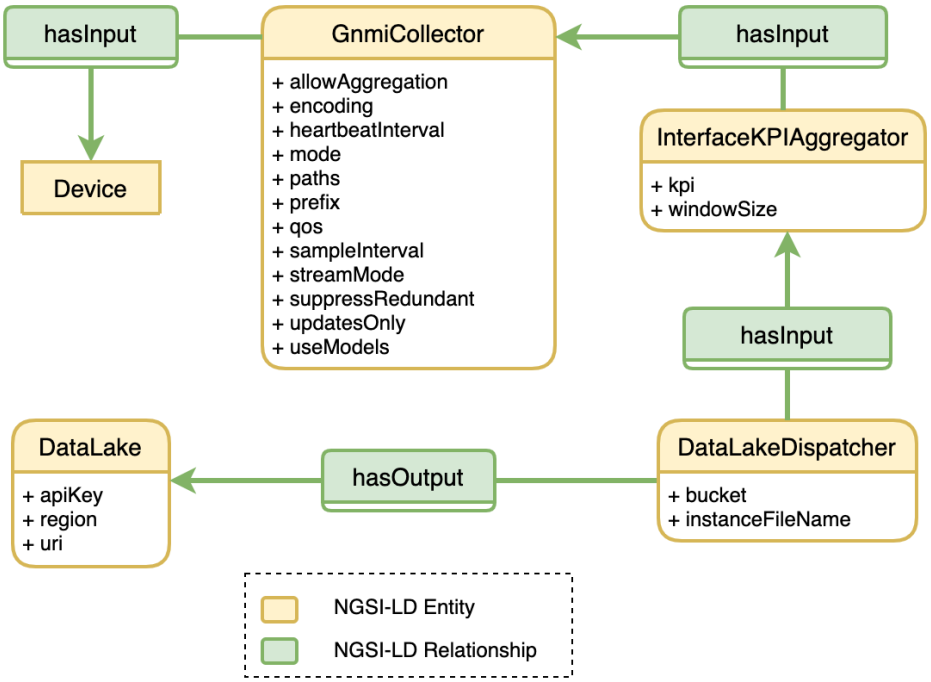



**Figure 3-43. NGSI-LD information model of data pipeline that collects telemetry data from device and stores aggregated KPIs in 5G-CLARITY's Data Lake platform**

***InterfaceKPIAggregator***: Representation of a streaming processing task that computes the specified network interfaces KPI. Thus far, the DSF supports only the aggregation of packet loss and throughput KPIs as detailed in Section 3.2.5. The type of KPI to be aggregate can be configured by means of the *kpi* property. Optionally, the entity enables configuring the size of the time-based aggregation window, which is specified in milliseconds. The task that produces data from which this aggregating task computes the KPIs is determined by the *hasInput* relationship. In the current use case, the target of this relationship must be a *GnmiCollector* entity.

***DataLakeDispatcher***: Entity that models a task that produces aggregated KPI data as new files in the specified S3 bucket in the Data Lake platform. The name that will be given these files can be adjusted with the *instanceFileName* property, which maps to the instance data set name as described in RFC 9195 [39]. Lastly, to determine the Data Lake platform where to store the files, the *hasOutput* relationship must point to an *DataLake* entity.

In summary, the proposed NGSI-LD information model provides an abstraction of the data pipeline that the DSF will build. This enables DSF operators to simply express what to monitor, which KPI to aggregate, and where these data should be stored in the Data Lake. Based on the provided configuration for the data pipeline, the Weaver component of the DSF orchestrates different workflows for each step of the pipeline. In the following, we will go through the details of each workflow that takes place depending on the type of entity created by the DSF operator.

#### 3.4.1.1.2 GnmiCollector

Figure 3-44 depicts the workflow that takes place for the GnmiCollector task. First, the Weaver establishes an NGSI-LD subscription to receive updates on the GnmiCollector entity type. Whenever a DSF operator defines a new GnmiCollector entity through the NGSI-LD API, the Weaver component receives the notification from the Weaver extracts the configuration for a gNMI subscription.

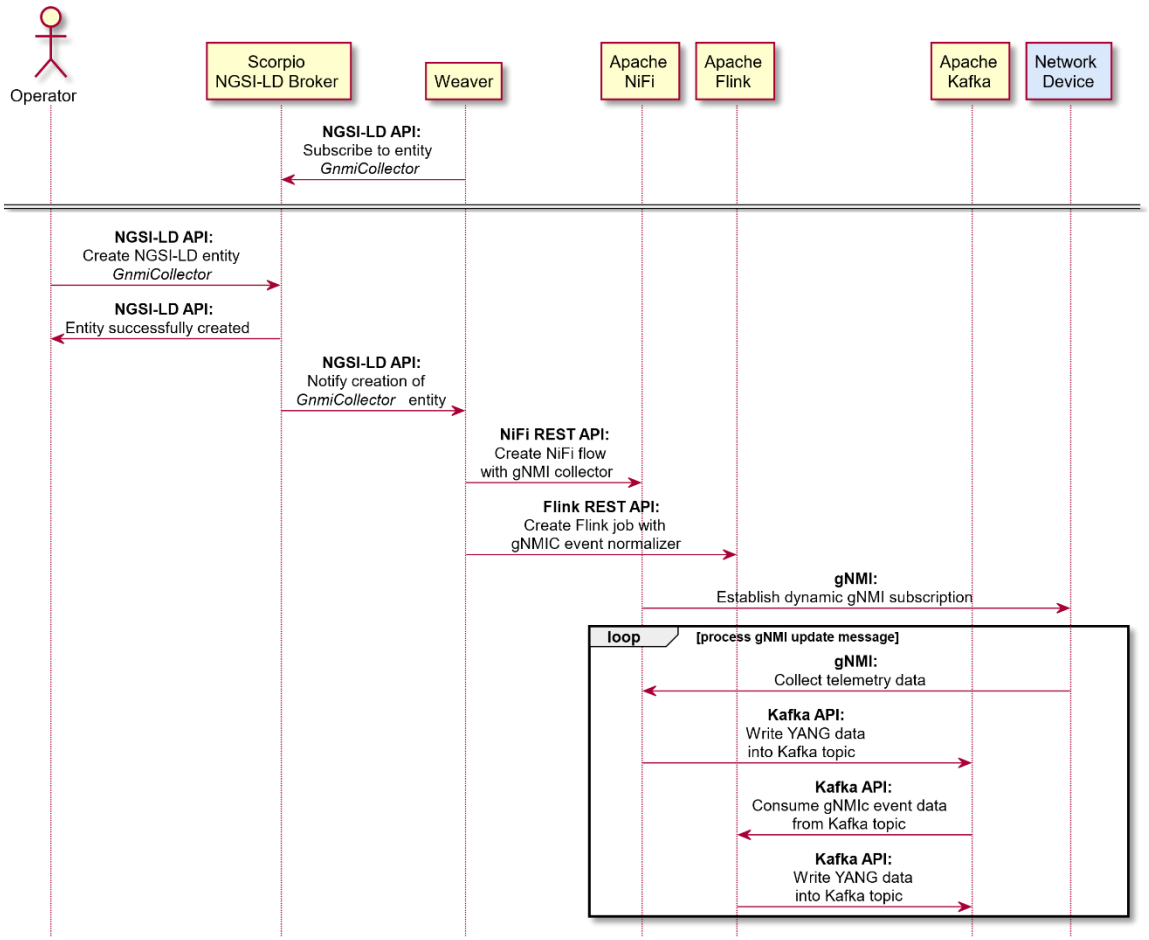



**Figure 3-44. Creation of GnmiCollector step of a data pipeline within the DSF**

With this information, the Weaver orders NiFi to instantiate a new flow that runs the gNMIc client to subscribe and collect telemetry data from the target device. This client allows to parse the received gNMI update messages, which can be stored as JSON data in a specific a Kafka topic. In parallel, the Weaver also sends a request to Flink to run a job that consumes the JSON data produced by gNMIc and writes the

normalized data into a separate Kafka topic for further processing. The process of collecting telemetry data in NiFi and the data normalization done with Flink, has been previously detailed in Section 3.2.3.

#### 3.4.1.1.3 InterfaceKPIAggregator

Similarly, Figure 3-45 captures the workflow that takes place for the InterfaceKPIAggregator task. During the start-up of the DSF, the Weaver creates an NGSI-LD subscription to receive notifications regarding entities of the InterfaceKPIAggregator type. From this point on, DSF operators can configure new instances of this task by creating a new InterfaceKPIAggregator entities in the Context Broker. Once an entity is created, an NGSI-LD notification is sent to the Weaver, which in turn parses the contents of the InterfaceKPIAggregator entity that will be used to schedule the respective job in Apache Flink. In particular, the Weaver inspects the value of the *kpi* property to determine whether it must schedule a Flink job that runs the throughput or the packet-loss aggregating application as described in Section 3.2.4.

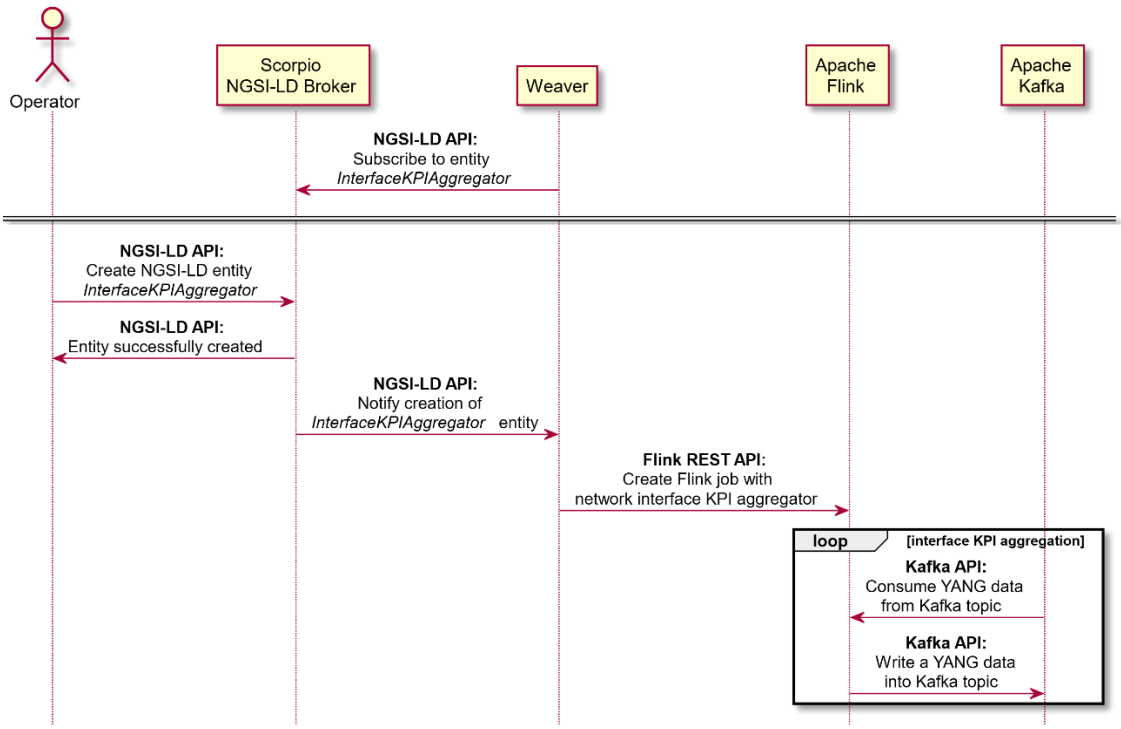



**Figure 3-45. Creation of InterfaceKPIAggregator step of a data pipeline within the DSF**

#### 3.4.1.1.4 DataLakeDispatcher

Lastly, Figure 3-46 details the last step of the data pipeline. In this case, the Weaver subscribes to updates on the DataLakeDispatcher entity type. When a new entity of this type is created in the Context Broker, a notification is sent to Weaver, which orders Flink to run a job that wraps network interface KPIs as YANG instance data, and then, orders NiFi to collect these data and write them in an S3 bucket in the Data Lake platform. Details on this process are provided in the next section.

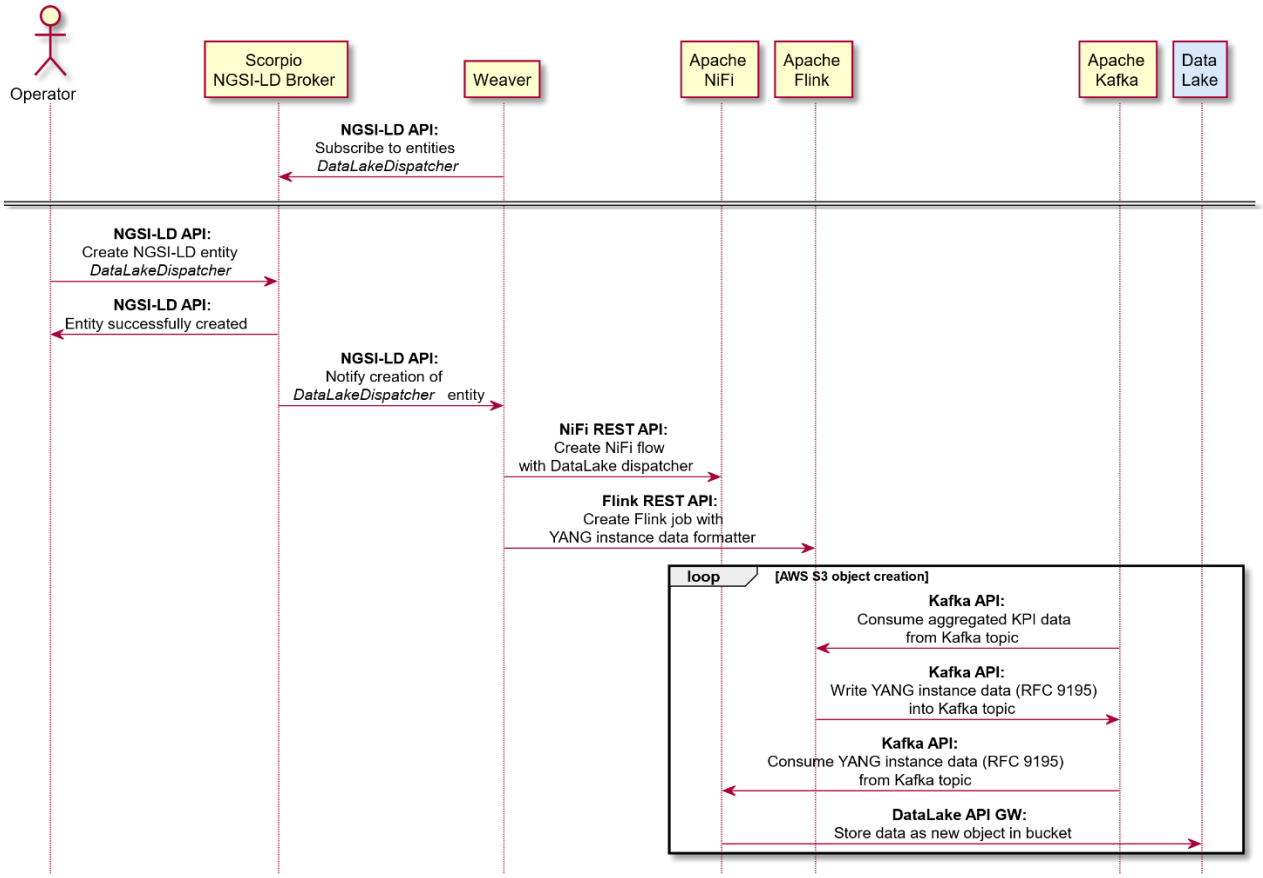



**Figure 3-46. Creation of DataLakeDispatcher step of a data pipeline within the DSF**

### 3.4.1.2 Writing Data into the Data Lake

This subsection addresses how data aggregated within the DSF can be written into the Data Lake (i.e., when the Data Lake is registered as data consumer). As commented before, the DSF leverages Apache NiFi and Apache Flink big data tools to collect, transform, and deliver data. In this sense, to write aggregated data into the Data Lake, two data pipeline steps are chained as depicted in Figure 3-47.

As described before, one of the principles of the DSF is that all data handled internally must be structured as per YANG data models. Therefore, the DSF first executes an Apache Flink application that reads data from an internal Kafka topic and wraps the data into the YANG instance data model as defined in RFC 9195.

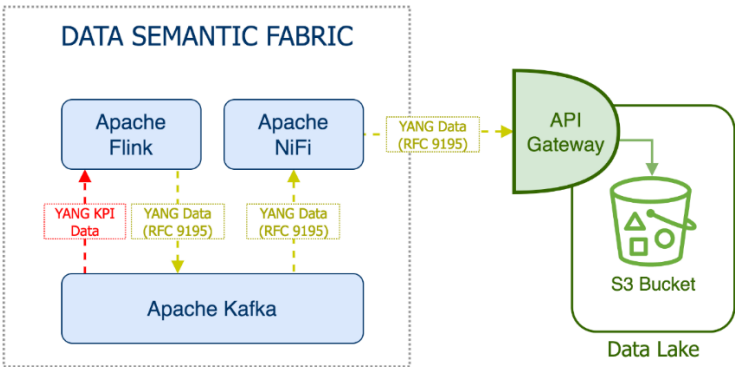



**Figure 3-47. Data pipeline that writes data into Data Lake**

```
module: ietf-yang-instance-data
  structure instance-data-set:
    +--name?                string
    +--format-version?      string
    +--includes-defaults?   enumeration
    +--content-schema
    |  +--(content-schema-spec)?
    |     +--:(simplified-inline)
    |     |  +--module*                 module-with-revision-date
    |     +--:(inline)
    |     |  +--inline-yang-library     <anydata>
    |     +--:(uri)
    |        +--same-schema-as-file?    inet:uri
    +--description*         string
    +--contact?             string
    +--organization?        string
    +--datastore?           ds:datastore-ref
    +--revision* [date]
    |  +--date           string
    |  +--description?   string
    +--timestamp?           yang:date-and-time
    +--content-data?        <anydata>
```

**Figure 3-48. YANG tree representation of YANG instance data model**

This RFC specifies a standard data model for storing YANG instance data files in an external storage system such as a data lake. This data model, as depicted in Figure 3-48, comprises two sections: content schema and content data. The content schema section is particularly interesting as it contains a reference to the YANG data model that structures the content data, which allows for consumer of the data to understand the structure and semantics of the data (i.e., the schema of the data).

Once data have been packaged into the YANG instance data model, they are sent back to another Kafka topic, so the second step of the chain comes into play. An Apache NiFi flow, which subscribes to this Kafka topic, collects new data events, and writes them into a specified bucket in the Data Lake by making HTTP PATCH requests through the API Gateway. As a result of this process, a new object (i.e., YANG instance data file) is created in the specified bucket for every event processed by NiFi.

### 3.4.2 Data Lake as the Data Source to DSF

This is the second type of configuration when integrating the Data Lake and the DSF. In this integration, the Data Lake platform is registered as another data source in the DSF. As a result, the DSF enables collecting AWS S3 files from the Data Lake so that their contents can be later aggregated by using Flink applications. Eventually, the aggregated data can be adapted to a particular format and delivered to other data consumers. Note that the Data Lake itself can also be a consumer of these data, when users of the Data Lake want to leverage the aggregation mechanisms offered by the DSF but still rely on the Data Lake platform as the storage system for data analysis.

Nevertheless, as of this deliverable, we have not identified any use case to validate this second type of integration. Yet, this release of the DSF implements the first building blocks that would enable the registration of the Data Lake platform and the discovery of its capabilities. The following subsections provide details on these implementations, namely, an NGSI-LD information model related to the Data Lake, and the new Data Lake Explorer service to auto-discover the capabilities of the platform.

#### 3.4.2.1 Context information of the Data Lake

As part of the integration, context information (i.e., capabilities) related to the Data Lake must be captured by the DSF. This context information must be made available to operators that intend to build data pipelines within the DSF that collect data from the Data Lake. By registering this information DSF users can discover what data are available in the Data Lake.

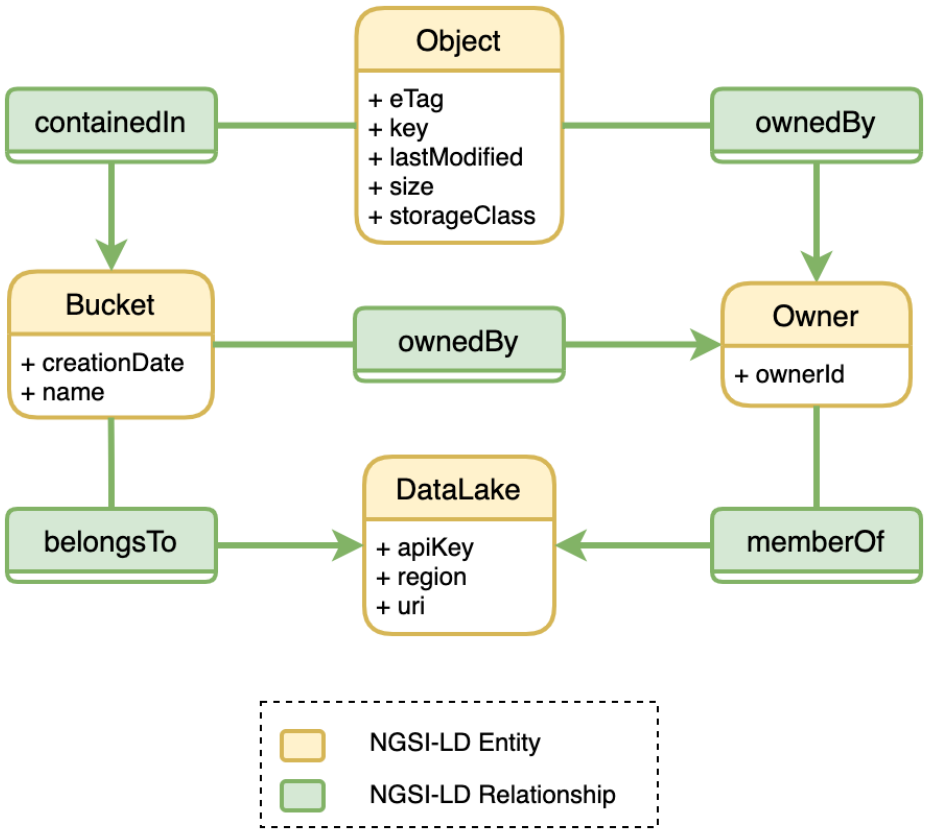



**Figure 3-49. NGSI-LD information model for Data Lake**

To represent the context information associated with the Data Lake, an NGSI-LD information model is envisioned as depicted in Figure 3-49. The Data Lake builds on AWS S3 object storage service. Thus, data stored in the Data Lake are arranged into buckets, which in our case are modelled as *Bucket* entities. Each *Bucket* may contain one or more *Object* entities that represent actual files of data in the Data Lake. The *Object* entity includes several properties containing relevant metadata such as the last modification date or the actual size the file. Lastly, the concept of *Owner* is modelled as a separate entity since multiple *Bucket* and *Object* entities can be owned by the same person.

Lastly, it must be noted that, ideally, the proposed NGSI-LD information model should be further enriched by introducing a *Schema* entity that would be linked to the *Object* entity. The purpose of the *Schema* entity is to describe the structure – and meaning – of the data pertaining to an object in AWS S3. This kind of information is crucial for data analysts and data scientists to understand data consumed from the Data Lake, and therefore, to achieve efficient data exploitation.

### 3.4.2.2 Data Lake registration and discovery of capabilities

The DSF introduces a component named the Data Lake Explorer. This component is a microservice that enables, first, the registration of the Data Lake as a data source, and second, exposing the capabilities available in the Data Lake.

To register the Data Lake in the DSF, the Data Lake Explorer initially subscribes to updates on context information related to the DataLake entity type. Then, DSF operators can request the registration of the DataLake platform by creating a DataLake entity in the Context Broker by means of the NGSI-LD API. Once this entity is successfully created in the Context Broker, an NGSI-LD notification, which contains all the entity's context information, is sent to the Data Lake Explorer.

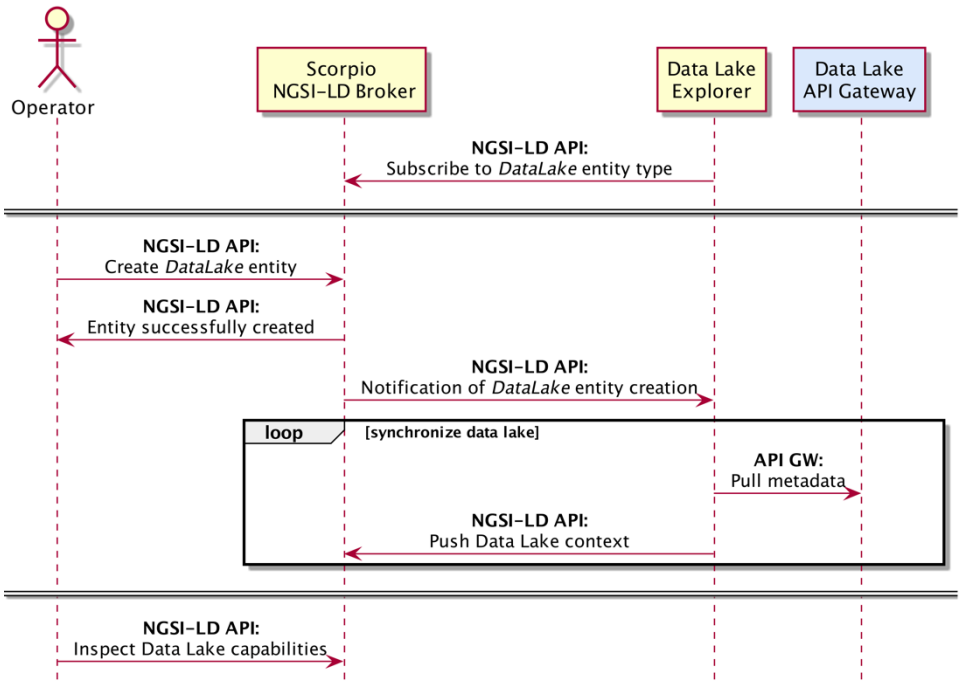



**Figure 3-50. Registration of Data Lake and discovery of capabilities through the NGSI-LD API**

Based on this information, the Data Lake Explorer establishes a connection with the Data Lake's API Gateway, which allows for pulling metadata from the Data Lake as depicted in Figure 3-50.

These metadata, which ranges from available buckets to objects stored in buckets, are processed by the Data Lake Explorer to produce context information as defined in the NGSI-LD information model for the Data Lake. This synchronization process is periodically triggered every hour by default, albeit this value can be modified in the Data Lake Explorer configuration. As a result, by pulling metadata through the Data Lake's API Gateway, the Data Lake Explorer can retrieve the capabilities available in the Data Lake. Consequently, DSF operators can later discover the capabilities of the Data Lake by sending queries through the NGSI-LD API to navigate the property graph captured in the information model.

# 4 Machine Learning Algorithms

In this section, the initial implementation of the ML algorithms described in 5G-CLARITY D4.2 [1] are extended and employed in real-world scenarios. Such algorithms not only facilitate automatic network management, but also provide a wider spectrum of network functionalities, the details of which can be found in D4.2 [1]. In what follows, we briefly mention each algorithm, its initial purpose, and the extension provided in this section. The ML algorithms are as follows:

**eAT3S evaluation (Section 4.1):** This section discusses our implementation of a hybrid model-free and model-based deep reinforcement learning (DRL) that is explained in D4.2 [1]. We refer to the algorithm to as model-augmented soft actor-critic DRL. The objective of the algorithm is to find an optimal policy to dynamically steer MPTCP subflows over multiple WATs, e.g., Wi-Fi and LiFi. A performance gain is shown later, and a validation is given by using an emulated residential scenario.

**RAN slicing in multi-tenant networks (Section 4.2)**: The algorithm expands on the previously proposed Deep Q-Network (DQN) to address the problem of capacity allocation to RAN slices while fulfilling each tenant's SLA requirements and efficient resource utilization. In particular, it is enhanced to learn generalizable policies that allow incorporating new tenants without having to retrain the DQN model. Moreover, new simulation results are provided to assess the performance of the proposed solution when adding new tenants in the scenario and when considering heterogeneous traffic distributions.

**Optimal Access Networks (Section 4.3):** Reinforcement Learning (RL) was employed in D4.2 [1] in the context of multi-connectivity architecture to steer, switch, and split the traffic intelligently. In this section, in order to validate the feasibility of the DQN-based solution, Open AI Gym and NS3 are deployed to build a simulation setup. The proposed approach will predict access network states to recommend a set of optimal multi-WAT access network policy that maximize the QoS and mobility.

**Optimal Compute Offloading (Section 4.4):** This section aims for minimizing delay and power consumption in multi-access edge computing network. To this end, it builds upon the solution proposed in the previous deliverable and leverages both machine learning and traditional optimization methods to solve the mixed-integer non-linear programming problem. It then draws on NS3 and AI Gym to assess the performance.

**RRP in multi-tech RAN sim extension (Section 4.5):** This section expands on the algorithm proposed in D4.2 [1] to allocate 5G spectrum to URLLC and eMBB services at every gNB. The solution architecture has been extended by including a master algorithm in charge of coordinating the different DRL agents and handling Wi-Fi offloading. Besides, the analytical model is extended, thereby representing a more accurate URLLC agent. The agents' performance is then assessed with the aid of detailed simulations using a RAN system-level simulator.

**Long-term transport network setup (Section 4.6):** This section focuses on the extension of the Deep Reinforcement Learning-based solution for the purpose of the long-term configuration of TSN-based transport networks (TNs) for accommodating 5G-CLARITY slices while guaranteeing their deterministic delay requirements. In particular, the approach proposed in 5G-CLARITY D4.2 [1] is generalized to make it applicable to a wider range of scenarios. Also, the agents design has been refined for generality, i.e., to incorporate a wider range of configurations. Such generalization is then evaluated by testing the algorithm in the complex scenarios of unseen environments.

In Table 4-1, a summary of the ML model types and their progress with respect to background (pre-5G-CLARITY) and to previous deliverables, as well as links to work carried out in WP3, can be found. Furthermore, Figure 4-1, indicates the relation of the algorithms presented in this deliverable with 5G-CLARITY system level

architecture.

**Table 4-1. ML Model Progress vs Previous Deliverables**

| Use Case | ML Model Type | Background | Progress w.r.t D4.2 | Relation to D3.3 |
|---|---|---|---|---|
| **eAT3S evaluation** | RL | DRL agents to intelligently steer traffic from systems consisting multiple WATs from MPTCP-enabled user devices. The proposed algorithm is evaluated against various MPTCP congestion controls to show it can improve the existing system. | New ML model, extension of simulation scenario | LiFi's link level simulation |
| **RAN slicing in multi-tenant networks** | RL | Existing capacity sharing solutions before 5GCLARITY and their limitations:<br>- Heuristic approaches for single and multi-cell scenarios<br>- Single agent DRL-based solutions working mostly at single cell basis or aggregated for multiple cells, but not considering multiple cells jointly.<br>- Scalability when adding/removing number of tenants not addressed.<br>- SLAs mostly specified with QoS parameters at user level, but not on aggregate terms per tenant.<br>- Training/inference of DRL models hardly discussed in the literature | Upgraded version of the algorithm to make it more generalizable and allow inclusion of new tenants. Additional results under heterogeneous traffic distributions. | - |
| **Optimal network access problem** | RL | Existing frameworks for ATSSS are limited to single objective optimization and simple set of s variables. To optimize the ATSSS in a multi-connectivity and multi-WAT the usage of Machine Learning is essential to deal with the complexity and enable automation. | Problem modelling as Integer Linear Program, setup of AI Gym- and NS3- based simulation environment and initial validation of the reward function compared exact solutions of the optimization model. | LL-ATSSS interaction |
| **Optimal compute offloading** | RL | The three processing methods for tasks are local processing, processing in Multi-access Edge Computing (MEC) server, and processing in the data centre. Each of them was optimized separately by previous proposed solutions. As a results, none of the considered. | The problem modelling and early test where ongoing during the D4.2 submission, as a result it was reported on D4.3 submission. | |
| **Resource provisioning in a multi-technology RAN** | RL | Independent DRL agents to perform radio resource provisioning in 5G NR for network slices of type eMBB and URLLC. For the agents training, simplified models of the 5G NR were employed. Solution and environment implemented in Matlab. | Refinement of DRL-based agents' design, analytical modelling extension, solution design refinements, Wi-Fi-offloading capability, and master algorithm for coordinating the agents. Solution and environment implemented in Python using Stable-baselines3 and AI Gym. | - |

| | | | | |
|---|---|---|---|---|
| **Transport network setup** | RL | DRL multi-agent solution to find valid configurations of an asynchronous TSN network. The solution is dependent of the scenario (e.g., number of 5G-CLARITY slices to be accommodated and network topology). Solution and environment implemented in Matlab. | Solution architecture and implementation enhancements and refinements for improving the capacity of generalization. Solution and environment implemented in Python using Stable-baselines3 and AI Gym. | - |

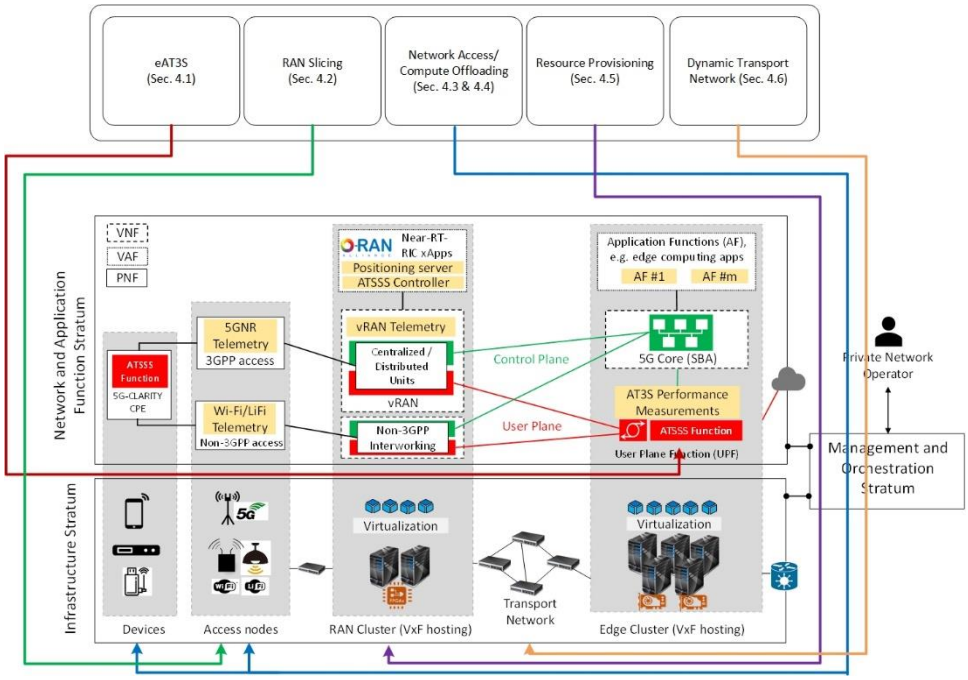



**Figure 4-1. Algorithms presented with 5G-CLARITY system level architecture**

## 4.1 eAT3S evaluation

In this section, details of our study on the real-time (RT) RAN intelligent controller (RIC) for AT3S traffic routing/handover are discussed. We start by discussing our scenarios and problem statement. Then, our system model covering necessary components to realize the scenarios are explained. Next, we will focus on detailing our deep reinforcement learning (DRL) to solve the problem. The computer simulation results and discussions are then given at the end of this section.

### 4.1.1 Scenarios under observation and problem statement

To clarify the framework that we will investigate in our simulations, we will start by illustrating our scenario first. Then, our problem statement will be formulated.

#### 4.1.1.1 Scenario description

The considered scenarios will be based on the descriptions specified in TGax [40] . The main reason for this is that they can be easily adopted with additional LiFi APs, which are based on the IEEE Task Group for the IEEE 802.11bb (TGbb). Only the residential scenario from the IEEE Task Group for the IEEE 802.11ax (TGax) described in [40] is adopted. In the residential scenario, a five-story building with 2 x 10 apartments on each floor, and each apartment's dimensions are 10 m x 10 m x 3 m is assumed. Figure 4-2 shows the illustrations of this scenario.

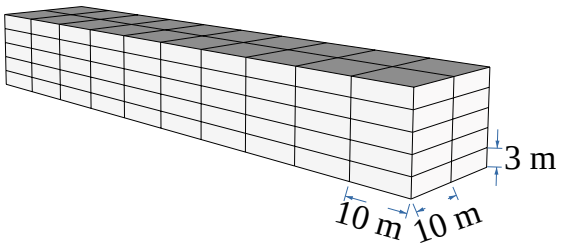



**Figure 4-2. Description of the residential scenario: a five-story building with 2 x 10 apartments in each floor, and the dimensions of each apartment are 10 m x 10 m x 3 m [41] [42] [43]**

Based on the TGax document [40], the number of APs is fixed, i.e., one AP in an apartment in the residential scenario and four APs in an office in the enterprise scenario. In addition, the APs are randomly located within a room located in the residential scenario, while they are fixed in an office in the enterprise scenario. All STAs are assumed to employ both the IEEE 802.11ax and the IEEE 802.11bb. In this chapter, the number of STAs and the locations of STAs are varying in order to incorporate random orientations, mobility, and blockage. Objects in the rooms are randomly generated. For example, realizations of an apartment is shown in Figure 4-3 . The realization illustrates a uniform placement of APs and random generation of human models with different activities while using their mobile devices.

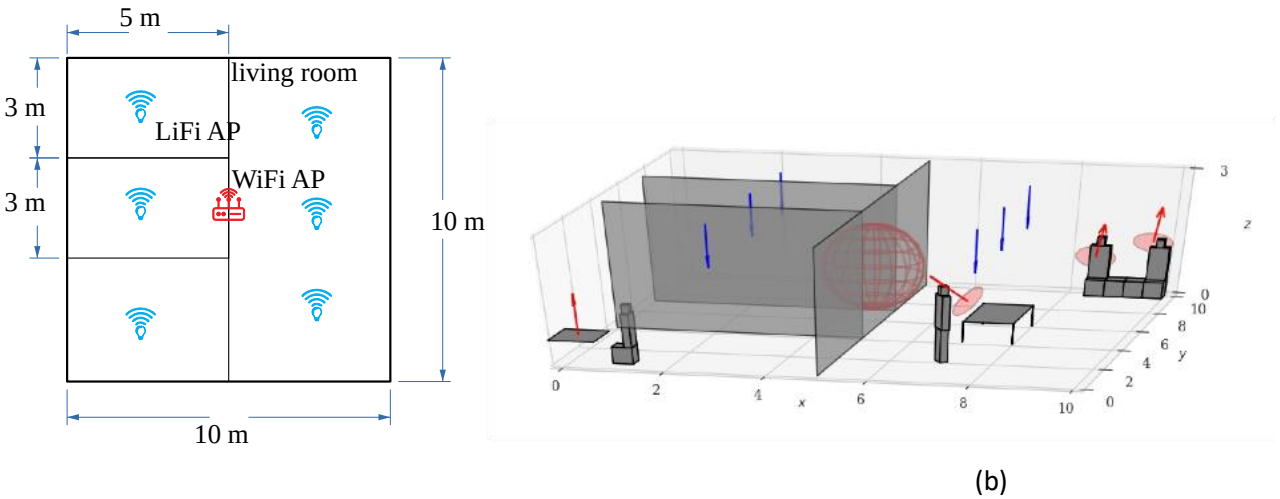



**Figure 4-3. Description of the scenario; (a) A floor plan of a realization of the interior of an apartment in the residential scenario, and (b) the 3D realization of the apartment using owcsimpy**

The number of people in the small room in the residential scenario is modelled as a Bernoulli distribution. In the living room, the number of people follows the Poisson distribution with a mean of 3. The activity of each person is modelled as a uniform distribution over a feasible set of options. If a realization is not feasible, then a rejection sampling is used. The location of each person also follows a uniform distribution over a set of feasible locations. The uniform distribution is chosen due to no prior information of the location of persons for our scenario.

### 4.1.1.2 System model

Here, we will explain in detail the necessary components to build an emulator as illustrated in Figure 4-4. Our emulator will use mininet as the emulator platform as shown in Figure 4-4. As there is already mininet-Wi-Fi[9], where the vanilla mininet is equipped with additional tools to emulate a Wi-Fi system, we first extend mininet-Wi-Fi such that we can also emulate a LiFi system. In order to have such system, we must perform a

[9] https://mininet-Wi-Fi.github.io/

link-to-system mapping, where we can estimate the packet error ratio given a channel quality metric, such as RSSI. Figure 4-5 shows the performance of the link-to-system mapping abstraction compared to the simulated one, where we compare the abstracted and simulated packet error ratios (PERs) vs. the signal-to-interference-noise ratio (SINR) over various modulation coding schemes (MCSs). It can be seen that the abstracted one can approach the simulated one. More details information about this mapping can be found in [41].

Unlike many other DRL-based MPTCP implementations, e.g., [42] and references therein, that modify an MPTCP implementation, which is in the kernel space, we add a Netfilter implementation as depicted in Figure 4-6. By using this approach, we can directly use any existing MPTCP implementation, e.g., the Linux kernel implementation from [43]. Later our DRL agent can intelligently steer the subflow by dynamically adjusting the value $\alpha$ in all devices, which shows the ratio of traffic that passes an interface compared to the other.

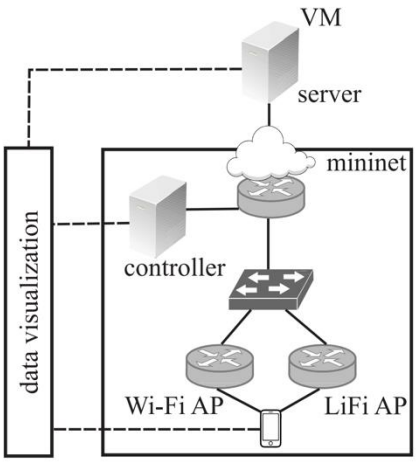



**Figure 4-4. Emulator diagram**

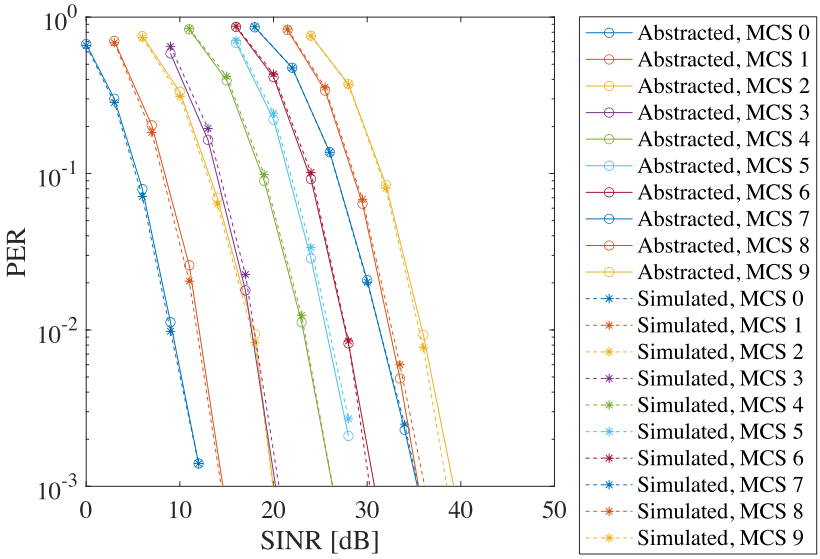



**Figure 4-5. PER vs. SINR (dB) computer simulation results for the residential scenario**

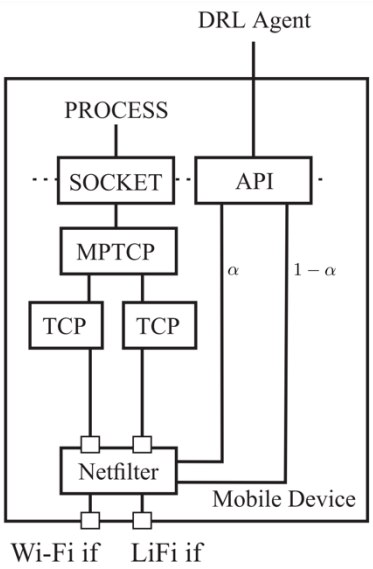



**Figure 4-6. Subflow steering using a Netfilter**

#### 4.1.1.3 Problem formulation

Our problem will be formulated by using typical RL notations. We aim to implement a hybrid model-based and model-free RL, which will be referred to as model-augmented soft actor-critic algorithm. By following [44], this algorithm tries to optimize the following objective function:

$$\pi^* = \arg\max_{\pi_\theta} \sum_t \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t)} \left[ r(t) + \beta \mathbb{E}_{\mathbf{a}_t} \left[ -\log \pi_\theta(\mathbf{a}_t | \mathbf{s}_t) \right] \right],$$

where $\pi_\theta$ is a parameterized actor network, $\mathbf{s}_t$is a collection of state information at time *t,* $\mathbf{a}_t$ is a collection of actions taken by the DRL agent at time *t*, $r(t)$ is a reward at time *t*, and ß is a temperature variable. The reward function $r(t)$ is defined as:

$$r(t) = \sum_{i=1}^{K} \log g_{t,i}$$

where *K* is the total number of subflows and $g_{t,i}$ is the goodput at the subflow $i$ at the time $t$.

### 4.1.2 Proposed deep reinforcement learning (DRL) algorithm

We propose a model-augmented soft actor-critic (SAC) algorithm [44], where a model network (parameterized by $\boldsymbol{\nu}$) is added to provide an estimated future state information denoted by $\hat{\mathbf{s}}_{t+1}$. The state $\mathbf{s}$ is defined as a collection of congestion windows and round-trip times from all MPTCP subflows. This estimated future state information is then passed to a critic network, which is parameterized by $\boldsymbol{\varphi}$. Features and results in a soft *Q* value denoted by $Q(\mathbf{s}, \mathbf{a})$. Then, the actor network (parameterized by $\boldsymbol{\theta}$) uses the *Q* value and the state to output actions $\mathbf{a}$, which is a collection of means and standard deviations of *K* Gaussian distributions to generate values $\alpha$ for all devices.

To be more specific, we employ an LSTM model for the model network which aims to minimize the error of the actual future state and the estimated one by taking advantage of the reply buffer, which stores the history of states, actions, and rewards. By following [44], the critic network is trained to minimize the following objective function:

$$J_Q(\boldsymbol{\phi}) = \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})} \left[ \left( Q(\mathbf{s}_t, \mathbf{a}_t) - (r(t) - \gamma V(\mathbf{s}_{t+1})) \right)^2 \right]$$

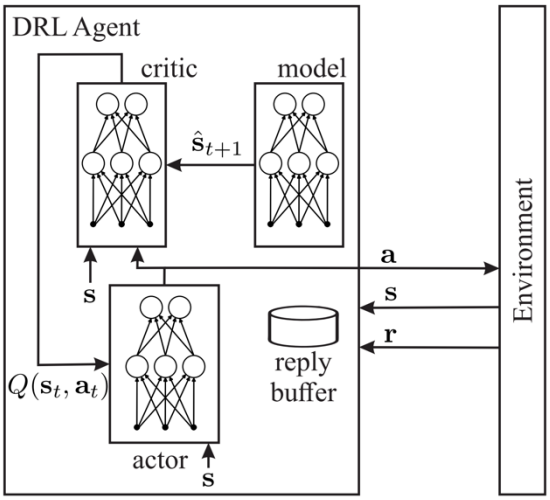



**Figure 4-7. Model-augmented SAC**

where:

$$V(\mathbf{s}_{t+1}) = \mathbb{E}_{\mathbf{a}}\left[Q(\mathbf{s}_{t+1}, \mathbf{a}_t) - \beta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\right]$$

Note that $\hat{\mathbf{s}}_{t+1}$ is supplied by the model network. Meanwhile, the actor network is trained to minimize the following objective function:

$$J_\pi(\theta) = \mathbb{E}_{\mathbf{s}}\left[\mathbb{E}_{\mathbf{a}}\left[\beta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) - Q(\mathbf{s}_t, \mathbf{a}_t)\right]\right].$$

The output of the actor network is a collection of $\mu_{t,i}$and $\sigma_{t,i}$, which are the mean and the standard deviation of a Gaussian distribution for the subflow *i*. Then, we use a re-parameterization trick defined by [45], where the value $\alpha$ is obtained as follows:

$$\alpha_{t,i} = \tanh(\mu_{t,i} + \epsilon\sigma_{t,i}), \text{ where } \epsilon \sim \mathcal{N}(0,1).$$

Algorithm 4-1 summarizes the training phase of our proposed model, where $\lambda$ is a learning rate.

**Algorithm 4-1: Pseudocode of model-augmented SAC**

**Input:** $\nu$, $\theta$, and $\phi$ ▷ Initial parameters
Initialize $\nu$, $\theta$, and $\phi$ with random weights
Initialize the replay buffer
**for** each iteration **do**
  **for** each environment step **do**
    Sample future states $\hat{\mathbf{s}}_{t+1}$ from the model network
    Sample actions $\mathbf{a}_t$ from the actor network
    Sample $\mathbf{s}_t$ and $r(t)$ from the environment
    Store $\{\hat{\mathbf{s}}_{t+1}, \mathbf{a}_t, \mathbf{s}_t, r(t)\}$ to the replay buffer
  **end for**
**end for**
Train the model network w.r.t. $\nu$
**for** each gradient step **do**
  $\phi = \phi - \lambda_Q \nabla_\phi J_Q(\phi)$
  $\theta = \theta - \lambda_\pi \nabla_\theta J_\pi(\theta)$
**end for**
**Output:** $\nu$, $\theta$, and $\phi$ ▷ Optimized parameters

### 4.1.3 Results and discussions

First, we recorded a simple demonstration showing a comparison of our proposed approach vs. the vanilla

MPTCP implementation where we show the fact that our algorithm can intelligently adjust the Netfilter coefficient depending on the signal quality of the links as can be found in the 5G-CLARITY YouTube channel[10]. Figure 4-8 shows the average performance the total throughput of different implementations obtained by running them multiple times

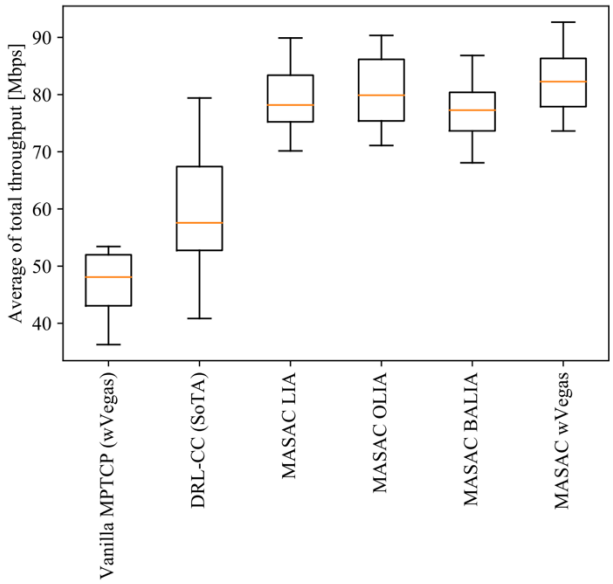



**Figure 4-8. Performance comparison between the vanilla MPTCP implementation based on [43], DRL-CC based on [42], and the proposed DRL approach (referred to as 'MASAC' for short)**

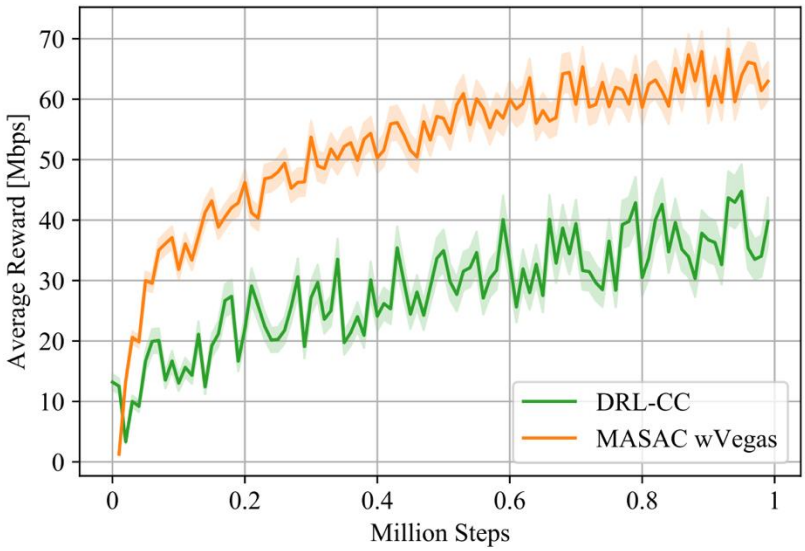



**Figure 4-9. Training curve comparison**

In summary, this section presented our approach in intelligently steer MPTCP subflows of all devices in a hybrid Wi-Fi and LiFi network. There are two main contributions made in this section, which are the use of a Netfilter instead of modifying an MPTCP implementation, and a model-augmented SAC. We showed that our

[10] https://www.youtube.com/watch?v=-o6nZiXeXEs

approach can achieve the average total throughput of 82 Mbps compared to 57 Mbps from DRL-CC and 48 Mbps from the vanilla MPTCP implementation.

## 4.2 RAN slicing in multi-tenant networks

This section considers a private venue owner of a Radio Access Network (RAN) infrastructure composed of *N* cells, each one with a different amount of physical resources that provide a certain cell capacity. The RAN is shared among *K* tenants, each of them provided with a RAN Slice Instance (RSI). Then, the problem we consider is to determine how the available capacity in each cell should be distributed among the different RAN slices while fulfilling the SLA requirements of each tenant and at the same time achieving an efficient utilization of the available resources. For addressing this problem, Section 4.4.3 of deliverable D4.1 [46] presented the initial design of a Multi Agent Reinforcement Learning (MARL) algorithmic solution based on Deep Q-Network (DQN). The solution, here referred to as DQN-MARL, considers one DQN agent per tenant that adjusts the resource quota (i.e. the proportion of physical resources in a cell) allocated to the RAN slice of the tenant jointly for each of the cells. It takes into account that the SLA requirements of the *k*- tenant are defined in terms of: (a) the Scenario Aggregated Guaranteed Bit Rate($SAGBR_k$) which is the aggregated capacity to be provided across all cells to tenant *k* if requested, and (b) the Maximum Cell Bit Rate( $MCBR_{k,n}$) which is the maximum bit rate that can be provided to tenant *k* in cell *n,* and is defined to avoid that a single tenant uses all the capacity in a cell under highly heterogeneous spatial load distributions with tenants demanding excessive capacity in certain cells.

The initial evaluation results of the DQN-MARL algorithmic solution were presented in section 3.3 of deliverable D4.2 [1]. The evaluation intended to assess the capability of the algorithm to adapt the assigned capacity to the traffic requirements of each tenant and to conduct a sensitivity analysis of two algorithm parameters, namely the action step, which determines the increase/decrease in the resource quota allocation, and the periodicity at which the resource quota is modified by algorithm. Starting from these previous studies, which reflected the promising behavior of the proposed solution, this deliverable presents the following extensions:

- An upgraded version of the algorithmic solution is provided. This new version targets a more scalable solution that allows adding new tenants in the scenario without having to re-train the previously learnt policies. This is mainly achieved through a modification of the agent's state to include the SLA requirements. This facilitates the capability of generalizing a policy learnt by the agent of one tenant so that it can be used by other tenants with different requirements.
- New simulation results are provided to assess the performance of the upgraded version of the solution, including an analysis of the capability of generalizing the learnt policies for tenants with different requirements, the behavior under the addition of a new tenant in the scenario, an optimality analysis, and the behavior of the solution under heterogeneous traffic distributions.

### 4.2.1 Final design of the DQN-MARL solution

The DQN-MARL algorithmic solution includes one DQN agent for each tenant that applies a policy to dynamically adjust the resource quota assigned to the RAN slice of the tenant in time steps of duration $\Delta t$. The resource quota of tenant *k* at time step *t* is formally defined as $\boldsymbol{\alpha_t(k)}=[\alpha_t(k,1),\ldots,\alpha_t(k,n),\ldots,\alpha_t(k,N)]$, where each component $\alpha_t(k,n)$ is the resource quota assigned to tenant *k* in cell *n*, given by the ratio between the physical resources assigned to the tenant and the total number of physical resources in the cell. It ranges $0 \le \alpha_t(k,n) \le MCBR(k,n)/C_T(n)$, where $C_T(n)$ (b/s) is the total capacity in cell *n*. Formally, the policy $\pi_k$ applied by the DQN agent of the tenant *k* is defined as $\pi_k = \underset{\boldsymbol{a(k)}}{argmax}\, Q_k(\boldsymbol{s(k)},\boldsymbol{a(k)},\theta(k))$ where $Q_k(\boldsymbol{s(k)},\boldsymbol{a(k)},\theta(k))$ is the expected cumulative reward when starting at state $\boldsymbol{s(k)}$ and taking action $\boldsymbol{a(k)}$ and is provided by a Deep

Neural Network (DNN) with weights $\theta(k)$.

To learn the policies, which in practice means to set the appropriate weights $\theta(k)$ of the DNN, a DQN agent interacts with a training environment that simulates the behavior of the RAN. At time step *t,* the DQN agent associated to tenant *k* obtains the state $\boldsymbol{s_t}(k)$ from the environment and, accordingly, it selects an action $\boldsymbol{a_t}(k)$ that updates the resource quota $\boldsymbol{\alpha_t(k)}$. This action selection follows an ε-Greedy strategy that chooses an action based on the currently learnt policy $\pi_k$ with probability 1-ε and explores a random action with probability ε. At the next time step *t*+1, a reward $r_{t+1}(k)$ assessing the suitability of action $\boldsymbol{a_t}(k)$ for the state $\boldsymbol{s_t}(k)$ is obtained as well as the new state $\boldsymbol{s_{t+1}}(k)$. Then, the agent stores the experience tuple <$\boldsymbol{s_t}(k)$, $\boldsymbol{a_t}(k)$, $r_{t+1}(k)$, $\boldsymbol{s_{t+1}}(k)$> in an experience dataset that will be used to update the policy $\pi_k$ following the training procedure that was detailed in Section 4.4.3 of deliverable D4.1 [46]. This training process stops after a sufficient number of time steps that ensures the convergence of the process. At this point, the resulting learnt policy $\pi_k$ defined by the weights $\theta(k)$ can be applied on the real network during the inference stage. Further details on the training and inference stages were presented in section 3.3 of deliverable D4.2 [1].

The definition of the states and rewards has been upgraded with respect to the initial solution of D4.1 [46]. The state enhancement has been done with the objective that the DQN agent of one tenant is able to learn a general policy that can be applied also by the DQN agents of other tenants with different SLA requirements. This has been achieved mainly through the inclusion of the SLA parameters in the state. In turn, the reward upgrades intend to better capture the SLA fulfilment and resource utilization targets. These aspects are described in the following.

**State:** It is denoted as $\boldsymbol{s_t(k)} = [s_t(k,1),\ldots, s_t(k,n), \ldots, s_t(k,N), SAGBR_k/C, \sum_{k'=1,k'\neq k}^{K} SAGBR_{k'}/C]$, where *C* is the aggregate system capacity that results from adding the cell capacities $C_T(n)$ of all cells. Each component $s_t(k,n)$ corresponds to the state of the tenant *k* in cell *n* given by $< \rho_t(k,n), \rho_t^A(n), \alpha_{t-1}(k), \alpha_{t-1}^A(n), MCBR_{k,n}/C_T(n) >$. The value of $\rho_t(k,n)$ is the resource usage, computed as the fraction of resources used by the tenant *k* in the cell *n* during the last time step ($t$-$\Delta t$, $t$), $\rho_t^A(n)$ are the available resources not used by any tenant in the cell and $\alpha_{t-1}^A(n)$ is the available resource quota in the cell *n* not assigned to any tenant.

**Action:** It is given by $\boldsymbol{a_t(k)}= [a_t(k,1), \ldots, a_t(k,n), \ldots, a_t(k,N)]$, where $a_t(k,n)$ is the specific action for each cell *n* and can take three different values $a_t(k,n)\in\{\Delta,0,-\Delta\}$, which correspond to increasing, maintaining and decreasing the resource quota as $\alpha_t(k,n)= \alpha_{t-1}(k,n)+a_t(k,n)$. This update is performed as long as the resulting resource quota $\alpha_t(k,n)$ is in the range $0\leq \alpha_t(k,n)\leq MCBR_{k,n}/C_T(n)$. Otherwise, no update is performed. Moreover, it must be ensured that the resource quotas of all tenants satisfy the condition $\sum_{k=1}^{K} \alpha(k,n) \leq 1$. Therefore, when this condition is not satisfied, the available resource quota $\alpha_t^A(n)$ is computed before applying the actions of the tenants willing to increase (i.e. with $a_t(k,n)=\Delta$). Then, in case that $\alpha_t^A(n)>0$, the resource quotas of these tenants are obtained by distributing $\alpha_t^A(n)$ among them proportionally to their $SAGBR_k$ values. Otherwise, the actions of these tenants are not applied.

**Reward:** The reward experienced by tenant *k* at time *t* is given by:

$$r_t(k)=\delta_t^{(1)}(k)^{\varphi_1}\cdot\delta_t^{(2)}(k)^{\varphi_2} \quad (1)$$

This considers two main factors, $\delta_t^{(1)}(k)$ and $\delta_t^{(2)}(k)$, with their corresponding weights, $\varphi_1$ and $\varphi_2$. The first factor, $\delta_t^{(1)}(k)$, promotes the satisfaction of the SLA of tenant *k* and is given by the ratio between the aggregated throughput of the tenant among all cells $T_k(t)$ and the aggregated offered load of the tenant among all cells $O_k(t)$, as long as the aggregate offered load of all tenants in the system, $O(t)$, is lower than the total capacity in the system *C*. Instead, if $O(t)$ is greater than *C*, $\delta_t^{(1)}(k)$ is computed as the ratio between $T_k(t)$ and $\min(SAGBR_k+\beta_t(k), O_k(t))$, where $\beta_t(k)$ is the amount of assigned

capacity that is left unused by the other tenants. The second factor, $\delta_t^{(2)}(k)$, measures the capacity overprovisioning and is defined by the ratio between $T_k(t)$ and the assigned capacity to the tenant among all cells (i.e. the summation of $C_T(n)\cdot\alpha_{t-1}(k,n)$ for all $n$=1…$N$).

### 4.2.2 Performance evaluation under homogeneous traffic conditions

The assumed scenario comprises a RAN infrastructure with $N$=5 cells using 5G NR technology that serve the users of two different tenants, denoted as Tenant 1 and Tenant 2. The configuration of the scenario is presented in Table 4-2, including the cells configuration and the SLA parameters established for each tenant.

The model has been developed in Python by using the library *TF-Agents* [47], which provides tools for the development of DRL models, including DQN. The developed model has been trained according to the parameters of the right side of Table 4-2. The dataset considered for training is composed of 1400 synthetically generated offered load patterns of Tenant 1 and Tenant 2 in the different cells during one day, considering different combinations of $SAGBR_k$ values for both tenants.

After the model has been trained, the resulting policies $\pi_k$ are evaluated using the offered load patterns shown in Figure 4-10. The figure plots the aggregated offered loads among all the cells of Tenant 1, $O_1(t)$, and Tenant 2, $O_2(t)$, during one day. The figure also includes the values of $SAGBR_1$ and $SAGBR_2$, the total system capacity $C$ and the aggregated offered loads of both tenants $O(t)$. Note that the offered loads of both tenants exceed their $SAGBR_k$ at some point during the day and the system offered load $O(t)$ is higher than $C$ during the time period from 900 min to 1300 min. Moreover, a uniform distribution of the load among the different cells has been considered.

**Table 4-2. Parameters of the Scenario and the DQN-MARL Model**

| Scenario parameters | | | DQN-MARL model parameters | |
|---|---|---|---|---|
| **Parameter** | | **Value** | **Parameter** | **Value** |
| Number of tenants ($K$) | | 2 | Initial collect steps | 5000 |
| Number of cells ($N$) | | 5 | Maximum number of time steps for training | $2\cdot10^6$ |
| Physical Resource Block (PRB) Bandwidth | | 360 kHz | Experience Replay buffer maximum length ($l$) | $10^7$ |
| Number of PRBs per cell | | 65 PRBs | Mini-batch size ($J$) | 256 |
| Average spectral efficiency | | 5 b/s/Hz | Learning rate ($\tau$) | 0.0001 |
| Total cell capacity ($C_T(n)$) | | 117 Mb/s | Discount factor(γ) | 0.9 |
| Total system capacity ($C$) | | 585 Mb/s | ε value (ε-Greedy) | 0.1 |
| $SAGBR_k$ | Tenant 1 | 351 Mb/s (60% of $C$) | DNN configuration | 100 nodes x 1 layer |
| | Tenant 2 | 234 Mb/s (40% of $C$) | Reward weights ($\varphi_1$, $\varphi_2$) | (0.5,0.4) |
| $MCBR_{k,n}$ | Tenant 1 | 93.6 Mb/s (80% of $C_T(n)$) | Time step duration (Dt) | 3 min |
| | Tenant 2 | | Action step (D) | 0.03 |


**Figure 4-10. Offered loads of Tenants 1 and 2 during a day**

The evaluation is conducted in terms of the Key Performance Indicators (KPIs) that were detailed in section 3.3.2 of deliverable D4.2 [1]. They are the assigned capacity per tenant and time step, the reward per tenant, the SLA satisfaction per tenant and the system utilization.

#### 4.2.2.1 Generalization of the learnt policies

In the considered approach the DQN agent of each tenant learns its own policy during the training and then this policy is applied during the evaluation. However, considering that the training of the different tenants has been done under very different situations of their own load and the load of the others and for different SLA parameters, the following results intend to analyze to what extent there are significant differences between the policies learnt by the different tenants. In this way, the main goal is to assess whether it is possible or not to generalize a policy leant by one tenant so that it can be also used by another tenant.

To conduct the analysis, the assigned capacity for the offered loads of Figure 4-10 is obtained under two different policy application modes. In *Mode A*, the DQN agent of each tenant applies its trained policy, i.e., the DQN agent of Tenant 1 applies policy $\pi_1$, and the DQN agent of Tenant 2 applies policy $\pi_2$. In turn, *Mode B* considers that the DQN agents of both Tenant 1 and Tenant 2 apply the same policy $\pi_1$ learnt for Tenant 1.

Figure 4-11 presents the temporal evolution of the offered load of Tenant 2, $O_2(t)$, against its assigned capacity $A_2(t)$ for policy application *Mode A* and *Mode B*. The assigned capacity for both policy application modes generally adapt to the offered load for all the situations where the total offered load $O(t)$ (seen in Figure 4-10) does not exceed the system capacity $C$. In turn, when $O(t)$ exceeds the system capacity, the assigned capacity to Tenant 2 is kept in the $SAGBR_2$ value. The figure shows that very little differences are observed in the assigned capacity $A_2(t)$ when applying the policies according to *Mode A* and *Mode B*. Moreover, to quantitatively assess the differences between both modes, Table 4-3 provides the average reward and the SLA satisfaction for both tenants in addition to the average system utilization. The obtained values show that the achieved performance for both policy application modes is very similar, with differences lower than 1% for all the analyzed KPIs. As a result, it can be concluded that, thanks to the training process using a dataset composed of several offered load situations and diverse combinations of SLA requirements, the agents of the two tenants have learnt equivalent policies that can be generalised to many offered load situations and SLA requirements. This has important positive implications on the practicality of the DQN-MARL approach, because it means that a single training process carried out by one DQN agent using a dataset that covers a wide range of offered load situations and SLA requirements can be sufficient to obtain a policy that is valid for multiple tenants. As a consequence, a reduction of the complexity of the training process will be achieved in a multi-agent scenario. Moreover, this also facilitates the scalability of the model to add new

tenants in the scenario, because the addition of a tenant can be done without retraining the previous learnt policies, as it will be studied in the next sub-section.

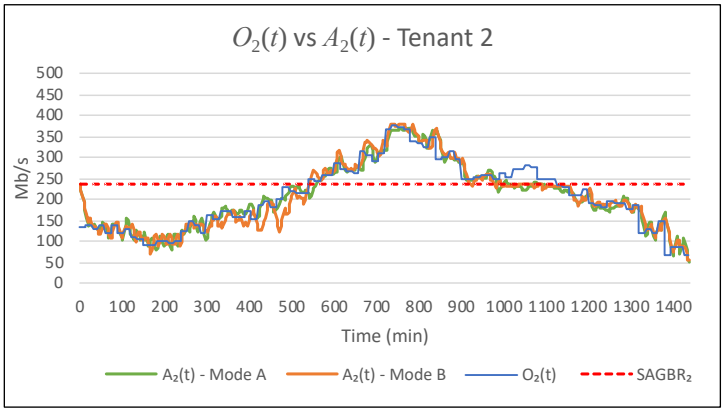



**Figure 4-11. Offered load vs assigned capacity for Tenant 2 for Modes A and B**

**Table 4-3. KPIs for Both Policy Application Modes**

| Policy application mode | | Mode A | Mode B |
|---|---|---|---|
| Average reward | Tenant 1 | 0.9673 | 0.9674 |
| | Tenant 2 | 0.9541 | 0.9483 |
| SLA Satisfaction | Tenant 1 | 0.9725 | 0.9742 |
| | Tenant 2 | 0.9705 | 0.9577 |
| Average system utilization | | 0.8885 | 0.8861 |

#### 4.2.2.2 Addition of a new tenant

Following the observed generalization capability of the trained policies next results aim at assessing the association of already trained policies to new tenants that are added in the scenario, without neither training new policies for the new tenants nor retraining (i.e., training again) the policies from the existing tenants. To this end, a new tenant, denoted as Tenant 3, is introduced to the previous scenario of Table 4-1. Instead of performing a separate training for the new Tenant 3, the previously trained policy for Tenant 1, $\pi_1$, is used for this new tenant as well as for Tenant 1 and 2. Since the $SAGBR_k$ of Tenants 1 and 2 use the total system capacity of Table 4-1, in order to support the new tenant, the capacity in the system is extended by increasing the number of PRBs in each cell to 78 PRBs, providing a total cell capacity $C_T(n)$ =140 Mb/s and, thus, a total system capacity of $C$ =700 Mb/s. The SLA established for Tenant 3 considers $SAGBR_3$=93.6 Mb/s and $MCBR_{3,n}$=114.56 Mb/s, corresponding to 80% of the cell capacity. The $SAGBR_k$ of Tenant 1 and 2 remain the same as in Table 4-1, whereas the $MCBR_{k,n}$ of those tenants is updated to $MCBR_{1,n}$=$MCBR_{2,n}$= 114.56 Mb/s given that the cell capacity has increased.

Figure 4-12 shows the offered loads $O_k(t)$ against the assigned capacity $A_k(t)$ of Tenant 1, 2 and 3, in addition to their $SAGBR_k$ values. The offered loads of Tenant 1 and Tenant 2, $O_1(t)$ and $O_2(t)$, are the same as in previous study, and the offered load of Tenant 3, $O_3(t)$, presents lower values than the other tenants, reaching its higher values at $t$=570 min and $t$=880 min when its $SAGBR_3$ is exceeded. Despite introducing Tenant 3, the total offered load of the three tenants only slightly exceeds the system capacity from $t$=1000 min to $t$=1200 min. Then, since most of time there is enough capacity to fulfil the offered load of the three tenants, the offered loads are satisfied nearly all day. When the overall offered load $O(t)$ exceeds the system capacity, the tenants that required more capacity than their $SAGBR_k$ are assigned with lower capacity than their offered load, such as Tenant 2 from $t$=1035 min to $t$=1115 min. In the case of Tenant 3, the offered load $O_3(t)$ is generally satisfied. These results show qualitatively that the policy learnt by one tenant is general enough to properly assign the capacity to the other tenants according to their offered loads and SLA

requirements and, additionally, performs satisfactorily in front of changes in the system capacity, since the internal parameters of the DQN agent (i.e., state, reward factors, actions, etc.) are defined in relative values. Moreover, to perform a quantitative assessment, Table 4-3 compares the obtained KPIs for the case when the policy $\pi_1$ is applied for all tenants against the case of applying separate policies $\pi_1, \pi_2$ and $\pi_3$ specifically trained for each tenant. Once again, the comparison reveals very small differences, lower than 1.5% for all KPIs. These results highlight the capability of scaling of the DQN-MARL solution, as the already trained policies can be used by new tenants in the scenario without retraining the whole solution again.

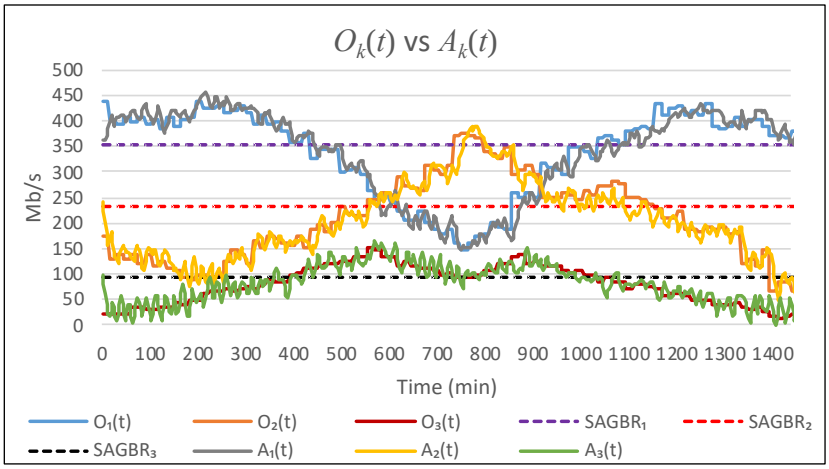



**Figure 4-12. Offered load vs assigned capacity for each tenant**

**Table 4-4 KPI Values**

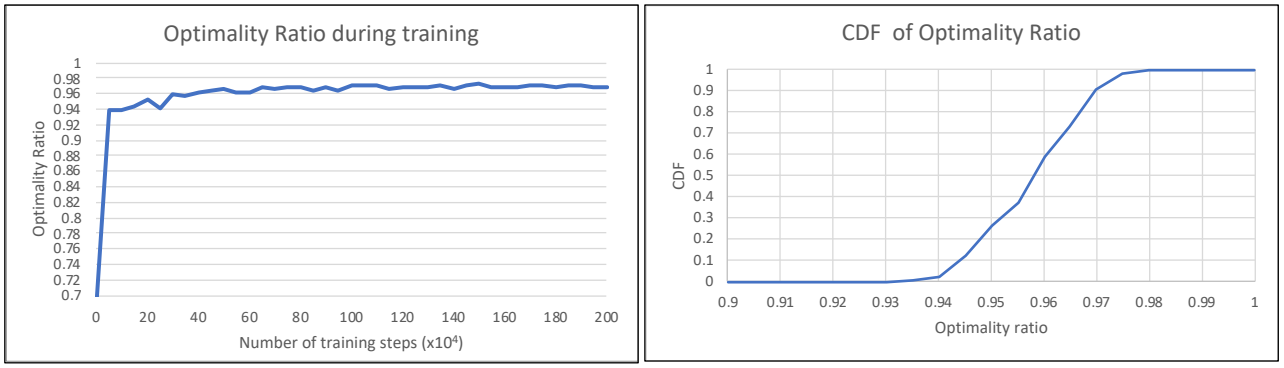
| Applied Policy | | Tenant-Specific Policies | Tenant 1 Policy |
|---|---|---|---|
| **Average reward** | **Tenant 1** | 0.964 | 0.967 |
| | **Tenant 2** | 0.939 | 0.949 |
| | **Tenant 3** | 0.873 | 0.859 |
| **SLA Satisfaction** | **Tenant 1** | 0.986 | 0.979 |
| | **Tenant 2** | 0.957 | 0.961 |
| | **Tenant 3** | 0.901 | 0.893 |
| **Average system utilization** | | 0.843 | 0.845 |

### 4.2.2.3 Optimality analysis

In the following, the optimality of the DQN-MARL approach is analyzed by comparing its performance to the optimum in the scenario with two tenants of Table 4-1. The optimum has been obtained by an exhaustive search algorithm that evaluates in each time step all the possible values of resource quota $\boldsymbol{\alpha_t(k)}$ of Tenant 1 and Tenant 2, discretized in steps of Δ, and selects the one that achieves the maximum aggregate reward of both tenants. To assess the optimality in a wide range of offered load situations, results have been obtained for a set of 240 offered load temporal patterns of one day duration, which include diverse offered load behaviors with diverse complementarities between the offered loads of Tenant 1 and Tenant 2. For each pattern, results have been obtained by applying the trained policy $\pi_1$ of Tenant 1 to both tenants. Results are given in terms of the optimality ratio, defined as the average of the aggregate reward of Tenant 1 and Tenant 2 obtained with the DQN-MARL approach divided by the average optimum reward over all the time steps of an offered load pattern.

Figure 4-13 (a) presents the evolution of the optimality ratio during the training process for the offered load pattern of Figure 4-10. This has been obtained by evaluating the policy $\pi_1$ every $5{\cdot}10^4$ training steps and computing the optimality ratio. It is observed that, initially, the optimality ratio increases abruptly with the

number of training steps and, after approximately $5 \cdot 10^4$ training steps, it achieves values higher than 0.94. Then, it increases slowly with the number of training steps and stabilises to a value of around 0.97, corresponding to the situation when the algorithm has converged. The figure also reflects that no significant improvements are obtained by increasing the number of training steps beyond 50·104. To analyze the optimality ratio under a broader range of situations, Figure 4-13 (b) shows the Cumulative Density Function (CDF) of the optimality ratios obtained for the different offered load patterns with the policy learnt after $200 \cdot 10^4$ time steps.




(a) (b)

**Figure 4-13. (a) Optimality ratio during training, (b) CDF of the optimality ratio**

The results reveal that the optimality ratios for all the analyzed offered load patterns range between 0.94 and 0.98. Moreover, it has been obtained that the average optimality ratio is 0.96. Overall, the results reveal that the DQN-MARL approach achieves a behavior very close to the optimum and they highlight the capability of the trained policy $\pi_1$ to adapt to diverse offered loads.

It is also worth noting that these near optimal results are obtained with very small execution times of the trained policy (i.e. the execution of one time step during the evaluation stage lasts 3.8ms on average using a machine with 2 CPU AMD Opteron 4386 operating with Ubuntu 18.03, configured to use 2 cores and 8G RAM), while the exhaustive search method requires to assess all the combinations for each time step, which is highly time consuming and requires execution times higher in several orders of magnitude than the DQN-MARL approach.

### 4.2.3 Performance evaluation under heterogeneous traffic conditions

The following study intends to assess the behavior of the DQN-MARL solution in a scenario where the offered load of the different tenants is heterogeneously distributed in the different cells. For this purpose, the scenario is a 3 km x 3 km area with $N$=5 cells and $K$=2 tenants, denoted as Tenant 1 and Tenant 2. The scenario and the DQN-MARL model have been configured with the parameters of Table 4-4. To generate heterogeneous spatial and temporal distributions of the offered load of the two tenants in the different cells, it is assumed that at time step $t$ the offered load density (Mb/s/km$^2$) of tenant $k$ is spatially distributed according to the sum of a constant offered load density $\mu_k$ and a bivariate Gaussian distribution centered at the position $(x_k(t),y_k(t))$ with standard deviation $d_k$ and offered load density in the center $m_k$. The center of the Gaussian distribution $(x_k(t),y_k(t))$ moves horizontally along the scenario with speed $v_k$ Then, the offered load of tenant $k$ in cell $n$ at time step $t$, $o_{k,n}(t)$, is obtained by aggregating the offered load density over the cell service area determined by the Voronoi tessellation. Based on this methodology, the DQN-MARL model has been evaluated under four different offered load situations that reflect different levels of heterogeneity, denoted as *Situations* 1-4, whose configuration parameters are given in Table 4-5. For each situation, the offered load of each tenant in each cell has been obtained during a day. Situation 1 corresponds to a nearly homogeneous spatial distribution of the offered load of one tenant among the different cells. Then, the level

of heterogeneity is increased in Situations 2-4, being Situation 4 the one with the most unbalanced load among cells. To illustrate this, Figure 4-14 plots the maps with the offered load densities of both tenants in Situation 4 at some illustrative times. The black triangles indicate the positions of the 5 cells.

**Table 4-5. Parameters of the Scenario and the DQN-MARL Model**

| Scenario Parameters | | | DQN-MARL Model Parameters | |
|---|---|---|---|---|
| **Parameter** | | **Value** | **Parameter** | **Value** |
| Number of tenants ($K$) | | 2 | Initial collect steps | 5000 |
| Number of cells ($N$) | | 5 | Maximum number of time steps for training | $2 \cdot 10^6$ |
| Physical Resource Block (PRB) Bandwidth | | 360 kHz | Experience Replay buffer maximum length ($l$) | $10^7$ |
| Number of PRBs per cell | | 78 PRBs | Mini-batch size ($J$) | 256 |
| Average spectral efficiency | | 5 b/s/Hz | Learning rate ($\tau$) | 0.0001 |
| Total cell capacity ($C_T(n)$) | | 140 Mb/s | Discount factor(γ) | 0.9 |
| Total system capacity ($C$) | | 700 Mb/s | ε value (ε-Greedy) | 0.1 |
| $SAGBR_k$ | Tenant 1 | 420 Mb/s (60% of $C$) | DNN configuration | 100 nodes x 1 layer |
| | Tenant 2 | 280 Mb/s (40% of $C$) | Reward weights ($\varphi_1$, $\varphi_2$) | (0.5,0.4) |
| $MCBR_{k,n}$ | Tenant 1 | 112 Mb/s (80% of $C_T(n)$) | Time step duration (Dt) | 5 min |
| | Tenant 2 | | Action step (D) | 0.03 |

**Table 4-6. Configuration of Offered Load Situations**

| Parameter | | Tenant 1 | Tenant 2 |
|---|---|---|---|
| Initial position($x_k(0)$,$y_k(0)$) (km) | | (0, 0.5) | (1.5, 2.5) |
| Speed ($v_k$) (km/h) | | 0.125 | -0.29 |
| Offered load density configuration ($m_k$(Mb/s/km$^2$), $d_k$(km) ) | Situation 1 | (24, 5) | (16, 5) |
| | Situation 2 | (28, 3) | (24, 3) |
| | Situation 3 | (36, 1) | (36, 1) |
| | Situation 4 | (72, 1) | (96, 0.5) |
| Constant offered load density ($\mu_k$) (Mb/s/km$^2$) | | 20 | 16 |

Figure 4-15 compares the resulting average offered load and the average assigned capacity (both expressed as a percentage of the cell capacity) in each cell for both tenants in Situations 1-4. The aggregated offered load and the aggregated assigned capacity of each tenant at system level (i.e., among all cells) is also included as a percentage of the total capacity. Results reveal that the assigned capacity takes close values to the offered load requirements both at cell and system levels for the different situations, regardless of the level of heterogeneity. In fact, the obtained differences between the offered loads and the assigned capacities are lower than 8% for all cases, which are mainly due to the incremental action design, which makes that the assigned capacity fluctuates around the offered load within a margin between Δ and -Δ. The highest differences are observed for cell 4 in Situations 2 and 3 and for cells 2 and 5 in Situation 4, since their total offered load in these cells exceeds the cell capacity during some periods, so the offered load of both tenants in those cells cannot be satisfied all the time. Moreover, results show that in certain cases when the traffic among cells is unbalanced and, in some cells, the offered load is higher than the relative $SAGBR_k$. The policy is able to support this load by smartly distributing the assigned capacity in accordance with the spatial traffic distribution. For example, the average offered load of Tenant 1 in Situation 4 in cells 4 and 5 exceeds the relative $SAGBR_k$ of 60% but the policy is able to support it since the offered load in the rest of cells is much lower than 60%. These results highlight the capability of the proposed solution to satisfactorily adapt to diverse levels of offered load heterogeneity among cells.

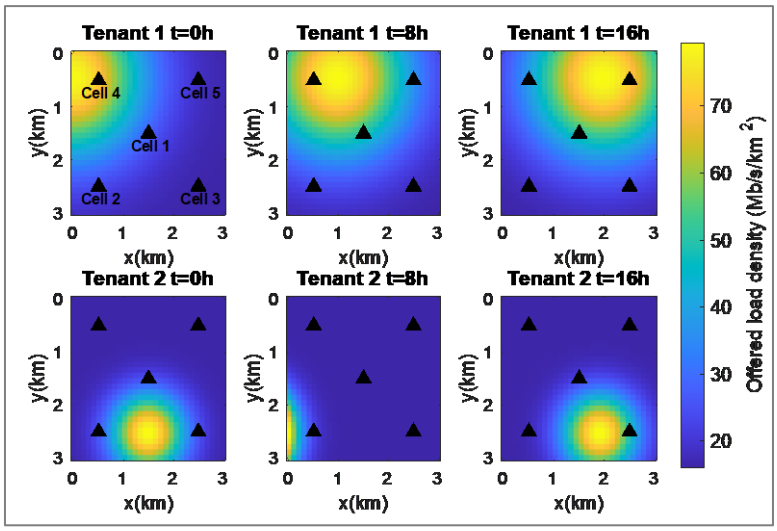



**Figure 4-14. Offered load density maps of Tenant 1 and 2 during a day**

Table 4-6 includes the reward, the SLA satisfaction and the assigned capacity ratio (i.e. the ratio between the assigned cell capacity and the cell offered load over all the cells in the scenario) per tenant and averaged over one day. The results show that the learnt policies achieve high average reward for both tenants in all situations. The good performance is also reflected in the values of SLA satisfaction, which are higher than 0.96 for Tenant 1 and 0.93 for Tenant 2. In relation to the assigned capacity ratio, the obtained values are close to 1, with maximum deviations of 8%. This indicates that the assigned capacity properly matches the offered load with little overprovisioning.

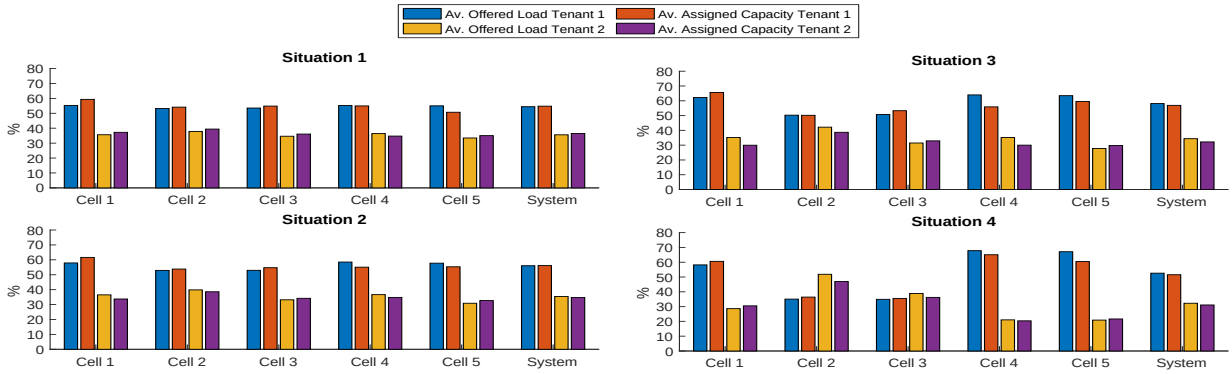



**Figure 4-15. Average offered load and assigned capacity per cell and at system level for each situation**

**Table 4-4-7. KPI Values**

| Applied policy | | Situation 1 | Situation 2 | Situation 3 | Situation 4 |
|---|---|---|---|---|---|
| **Average reward** | **Tenant 1** | 0.96 | 0.97 | 0.96 | 0.96 |
| | **Tenant 2** | 0.95 | 0.94 | 0.94 | 0.94 |
| **SLA Satisfaction** | **Tenant 1** | 0.97 | 0.98 | 0.97 | 0.96 |
| | **Tenant 2** | 0.94 | 0.97 | 0.95 | 0.93 |
| **Assigned capacity ratio** | **Tenant 1** | 1.04 | 1.01 | 1.01 | 1.08 |
| | **Tenant 2** | 1.04 | 1.02 | 1.08 | 1.05 |

### 4.2.4 Conclusions

This section has presented the performance assessment of DQN-MARL solution for RAN slicing in multi-tenant and multi-cell scenarios. Results have shown that: (i) The DQN-MARL solution satisfactorily adapts the capacity assigned to each tenant to their traffic and SLA requirements. (ii) The policies learnt by the agents associated to each tenant are generalizable to any tenant, given that the dataset used for training is

composed of a wide range of traffic requirement situations and SLA requirements. (iii) The proposed approach is easily scalable to deal with the addition of new tenants simply by associating to the new tenant a new DQN agent with a previously learnt policy. (iv) The trained policies are able to provide results very close to the optimum when they are applied to diverse offered load patterns, with observed optimality ratios ranging between 0.94 and 0.98. (v) The evaluation in a scenario with spatial traffic heterogeneity among cells has shown that the solution adapts the assigned capacity to each tenant in each cell to the traffic requirements while satisfying the SLA with average satisfaction ratios above 0.93, and efficiently using the available resources. Overall, the results presented here reflect the potential and adequacy of the proposed DQN-MARL solution for RAN slicing.

## 4.3 Optimal network access

5G and beyond (5GB) networks combine various 3GPP and non-3GPP radio access technologies (RATs), such as 4G LTE, 5G NR, Wi-Fi and Li-Fi. This convergence allows the integration between several wireless networks with flexible access to share resources as well as provides a pervasive multi-connectivity through different technologies. 3GPP and non-3GPP RATs convergence are not new in the telecom market, solutions such as LWIP, MuLTEfire, LWA/eLWA, LAA/eLAA, LTE-U have been proposed and commercialized for 4G LTE networks allowing multi-connectivity, 5GB networks will enhance their integration by enabling multipath transport protocols for efficient multi-connectivity between convergent WATs environment, such as Multipath TCP (MPTCP), Multipath QUIC (MPQUIC), the traffic can be scheduled and managed more flexibly. In 3GPP Rel-16, the multi-connectivity architecture is standardized as Access Technology Steering, Switching, Splitting (ATSSS) function and kinds of solutions are provided. However, owing to the complexity and the dynamism of 5GB wireless network, it lacks an efficient and intelligent way to manage the ATSSS function. Hence, our research is to propose an ATSSS manager, which can steer, switch, and split the traffic intelligently in the heterogeneous 5GB wireless network. To validate our solution, we set up our simulation platform and extend the experiments reported on D4.2 [1].

In this chapter, a simple ATSSS management scenario is emulated and a Deep Q network-based solution is evaluated.

### 4.3.1 System description

As Figure 4-16 shown, one user equipment keeps moving within a given square area. In the centre of the square area, there are a 5G gNodeB base station and a Wi-Fi access point. The user equipment is supported by both 5G and Wi-Fi connections and keeps transmitting through 5G or Wi-Fi connection to test the network performance, e.g., throughput. The propagation loss is considered for both the Wi-Fi and 5G transmission. While 5G with higher power can cover the whole square area, the signal strength of Wi-Fi gets weaker with the distance extending and eventually gets too weak to support the stable transmission in the edge of the square area.

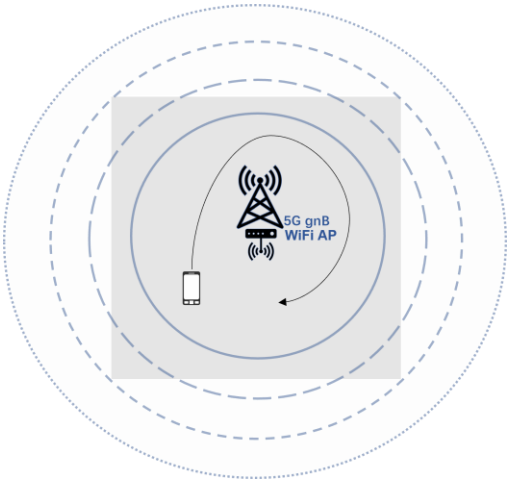



**Figure 4-16. Network scenario**

The interaction logic between the network side and the management side for intelligent ATSSS function is illustrated as Figure 4-17. This architecture is enhanced by the 5G CLARITY elements, such as AI Engine.

According to the design in Figure 4-17, our simulation platform introduced in Figure 4-18, integrates Network Simulator 3 (NS3) and OpenAI-Gym with our Reinforcement Learning Agent (RL Agent) based on Deep Q Network to enable Optimal Network Access.

The Network Model defining the simulation environment, components, and network protocols is deployed by NS3. The Open AI Gym enables the Environment Proxy which run the data interchange between the RL Agent and the Environment Gateway connected to the Network Model.

The implemented Network Model consists of a multi-WAT mobile access network combining 5G-NR and Wi-Fi. The setup includes a Remote Host representing multiple edge servers connecting multiple Wi-Fi access points and one EPC connecting multiple based stations (5G-NR BSs or RRUs). Based on the received signal and cost, the UE switches between Wi-Fi access point and 5GNR base station.

The experiment runs in the Network Model interchange data with the RL Agent through the Open AI Gym. There are the five steps in the simulation interaction process.

**Step 1**: the initial network state is provided to the Open AI Gym and RL Agent.

**Step 2:** the action space is defined based on the first network state considering the network environment.

**Step 3:** main iterations begin with the Open AI Gym and RL Agent.

**Step 4**: interaction ends, and reward is calculated by the RL Agent.

**Step 5**: final reward and convergence of the DQN algorithm to achieve the optimal network access policy.

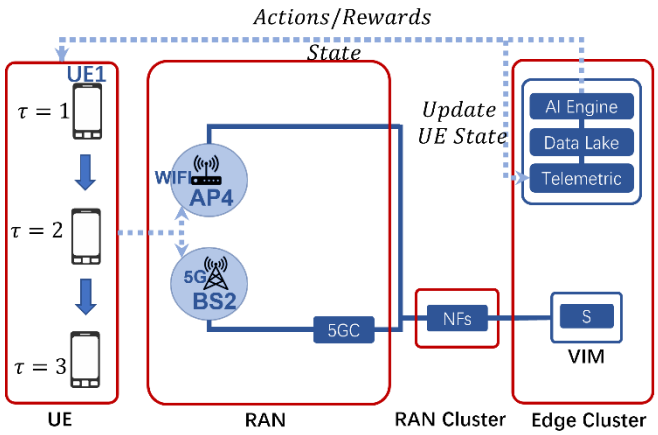



**Figure 4-17. System design for multi-WAT access**

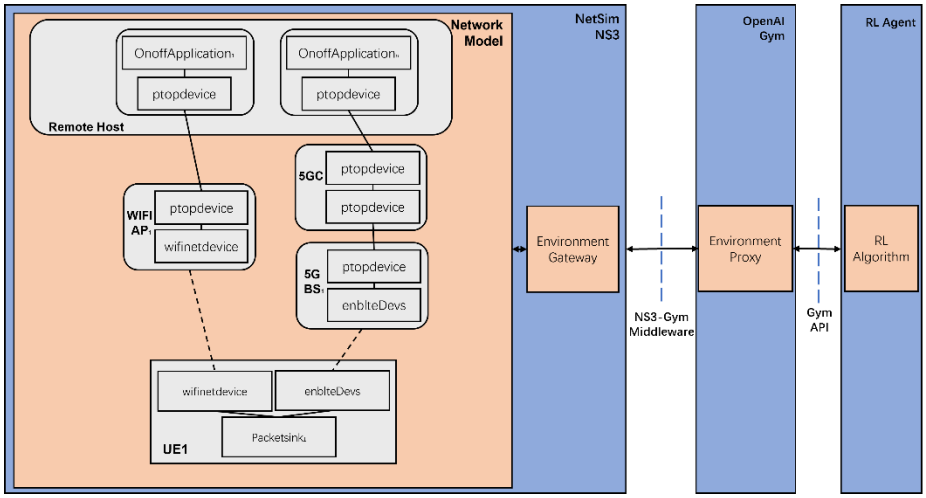



**Figure 4-18. Network model simulating multi-WAT access**

### 4.3.2 Proposed solution

The proposed solution is based on Deep Q Network (DQN). The algorithm is shown as Figure 4-19. In the DQN-based algorithm, the key elements are defined as follow:

State Space: consisting of RSRP of 5G and the SNR of Wi-Fi, represented as $S_\tau = [RSRP_{5G}, SNR_{WiFi}]$

Action Space: selection to connecting with 5G (0) or Wi-Fi (1), represented as $A_\tau = [0,1]$

Reward: jointly considering the received/transmitted rate, and the cost of 5G radio (with a 0.5 discount). And give a punishment (-1) is the signal quality of the selected radio is bad, represented as $R$ .

$$R = \begin{cases} \frac{r_\tau}{t_\tau} & (a = 1) \cap (\frac{r_\tau}{t_\tau} \geq \frac{1}{2}) \\ \frac{r_\tau}{2t_\tau} & a = 0 \\ -1 & \frac{r_\tau}{t_\tau} < \frac{1}{2} \end{cases}$$

Where $r_\tau$ is the number of received bytes, and $t_\tau$ is the number of transmitted bytes during a given period.

**Algorithm 1** Deep Q Network Algorithm [1]

Initialize replay memory $D$ to capacity $N$;
Initialize action-value function $Q$ with random weights $\theta$;
Initialize target action-value function $\hat{Q}$ with weights $\hat{\theta} = \theta$;
Initialize sequence $s_1 = x_1$ and pre-processed sequence $\phi_1 = \phi(s_1)$;
**for** $t = 1, 2, 3, ..., T$ **do**
    With probability $\epsilon$ select a random action $a_t$, otherwise select $a_t = argmax_a Q(\phi(s_t), a; \theta)$
    Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and pre-process $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random mini-batch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
    Set

$$y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma max_{a'} \hat{Q}(\phi_{j+1}, a'; \hat{\theta}) & \text{Otherwise} \end{cases}$$

    Perform a gradient decent step on $(y_i - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameter $\theta$
    Every $C$ step reset $\hat{Q} = Q$
**end for**

**Figure 4-19. Algorithm for model training**

### 4.3.3 Performance evaluation

During the training process, the reward is increased after several Episodes, and the loss is reduced greatly and tends to be stable as the Figure 4-20 and Figure 4-21 shown. It indicates that the model is effectively trained.


**Figure 4-20. Reward of each episode**

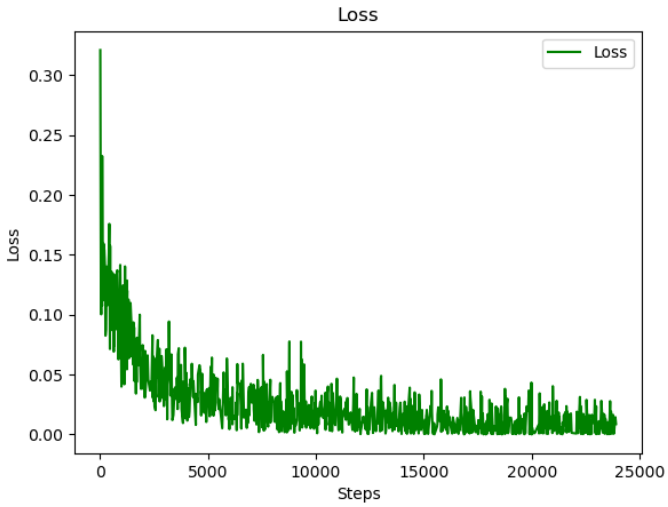



**Figure 4-21. Loss of each step**

Then the trained model is saved and applied back to the network environment to test the performance. The comparison of the random selection and the trained model is made over a same mobility trace of UE. The performance of the trained model is shown in Figure 4-22. The trained model increased the throughput and avoids the drop-off caused by the weak Wi-Fi signal.

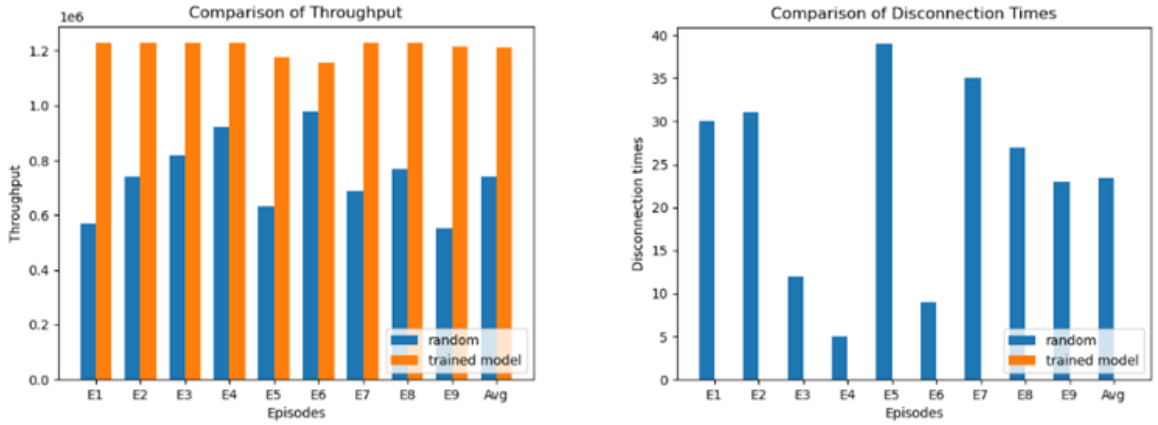



**Figure 4-22. Comparison of throughput and disconnection times over random selection and trained model**

### 4.3.4 Conclusion and future direction

For the intelligent management of the ATSSS function, the DQN-based algorithm shows great potential for optimal access decisions. In this experiment, the 5G cost, throughput, and connection persistence are proved to be enhanced by the trained model. In the next step, a stronger heuristic method-based selection will be applied for comparison. Besides, the network environment with more user equipment and access points will be considered. In that case, both the handover within the same radio access technology and between different radio access technologies require to be solved, which is further complicated by the different access control mechanisms of 5G and Wi-Fi. Moreover, the QoS requirements of different applications are another important involved factor.

## 4.4 Optimal compute offloading

In this section, we introduce our proposed solution of optimal compute offloading among multi-access edge

computing networks. In addition, with the objective of minimizing long-term overall latency and energy consumption, we provide an edge network management policy including actions on transmission resources, computing resources, and offloading decisions.

### 4.4.1 Background

Currently, there are three different processing methods for tasks namely local processing, processing in Multi-access Edge Computing (MEC) server, and processing in the data centre. As mentioned in Section 4.6 of D4.1 [46], local processing is constrained by the limited computing capability and also costs high execution delay and energy consumption. MEC could provide more sufficient resources than local devices, whereas it leads to longer transmission latency and its computing resources are also under a certain limit. In comparison, processing in the data centre could provide the lowest execution delay without any resources and power limits. However, it is at the expense of the most expensive transmission power and the farthest propagation distance.

In previous works, some of the offloading algorithms assumed there were unlimited computing resources on the MEC [48] [49], whereas in fact, with the emerging of computing-intensive and stringent latency tasks, MEC resources can easily be used up, leading to issues related to resource competition among tasks. With the raising of resource limitation issues on the MEC, the shortcomings of a single MEC server system that ignores the essential property of unbalanced network traffic have also been discussed in the past few years. Nowadays, in the context of multi MEC server systems with limited execution capability, technologies in solving the task offloading can be mainly divided into machine learning-based and optimization-based, respectively. However, the optimization-based method mainly used the first fit algorithm that cannot take the unbalance distribution property of the requests into consideration. Besides, in terms of machine learning-based methods, their action space explosion problem constraints their applicability in the real network. In addition, although the current work has greatly advanced edge computing in 5G and beyond, most of them failed to resolve the long-term resource scheduling that over distribution may cause the lack of resources for upcoming requests.

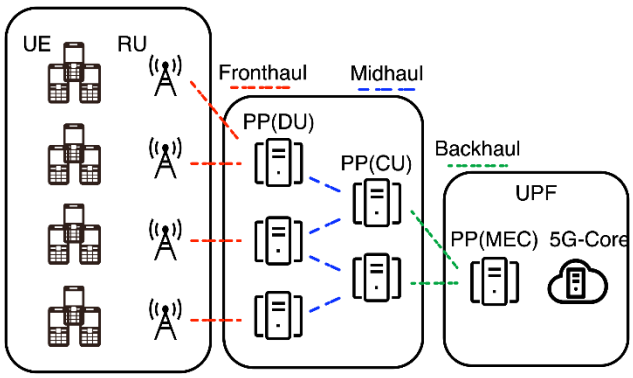



**Figure 4-23. Task offloading architecture in 5G and beyond networks**

### 4.4.2 Problem statement

There are three main problems need to be solved in designing offloading management policies. In prior joint offloading algorithms, the researchers failed to consider the geographic distance between mobile devices and servers. The FF-based solutions they used did not fundamentally solved the unbalanced traffic distribution problem [50]. In addition, the action space explosion in DRL-based strategies remained unclear. More importantly there is no research has been published on long-term reward and it hindered further improvement in network development. In order to solve these problems, with the objective of minimizing latency and power consumption in MEC network, we designed a mixed-integer non-linear programming problem. We then solved it through the combination of deep reinforcement learning and optimization-based

method. The following are the objective function and its constraints.

$$\max \sum_{n=1}^{F_{md}} \sum_{m=1}^{G} \sum_{d=1}^{H} d_{nm} \left( \beta_n^t \frac{T_n^l - T_{nm}^r}{T_n^l} + \beta_n^e \frac{E_n^l - E_{nm}^r}{E_n^l} \right) / F_{md} (1)$$

Subject to:

$$\text{C1: } F_{md}^d \leq F_{md}, \forall m \in G, \forall d \in H (2)$$

$$\text{C2: } \sum_{n}^{F_{md}^d} 0 \leq D_{nm} \leq G_{md}, \forall m \in G, \forall d \in H (3)$$

$$\text{C3: } G_{md} \leq R_d, \forall m \in G, \forall d \in H (4)$$

The objective function is designed to evaluate the profit of task offloading compared to local processing in a multi-user, multi-server and long-term edge computing system, where $F$ is the set of users, $G$ is the set of servers and $H$ is the set of time slots. $T$ and $E$ are latency and power consumption. $d$ represents the offloading decision. $\beta^t$ and $\beta^e$ are the preference of task on latency and energy, where they sum to 1. The constraints are mostly resource-related in terms of transmission, computation, etc. Among them, $D$ and $R$ stands for the distributed computing resources and the remaining resource on the server.

It is a NP-hard problem. To solve this problem, we divided it into two parts namely long-term offloading profit maximization subproblem P1 and short-term profit maximization subproblem P2. The long-term subproblem P1 is solved by a deep reinforcement learning algorithm by converting the offloading scenario into a Markov decision process. The short-term subproblem P2 is solved by an optimization-based method instead of DRL as to avoid the action space explosion because there may be thousands of offloading requests in the edge network. These two subproblems can be written as:

$$\text{P1: } M = \max \sum_{m=1}^{G} \sum_{d=1}^{H} L \quad \text{subject to: C1, C3(5)}$$

$$\text{P2: } L = \max \sum_{n=1}^{Q_{md}} d_{nm} \left( \beta_n^t \frac{T_n^l - T_{nm}^r}{T_n^l} + \beta_n^e \frac{E_n^l - E_{nm}^r}{E_n^l} \right) / F_{md} \mid m = m', d = d' \text{ subject to C2 (6)}$$

It is worth to mention that $D$ in constraint C2 of subproblem P2 is determined by the results of subproblem P1. Therefore, they are inter-related instead of independent. The relationship of P1 and P2 is summarized in Figure 4-24. Subproblem P2 will resolve the offloading profit and occupation time of the distributed resource and then sent them to subproblem P1 as the reward and interaction information of DRL environment. Other details will be discussed in the following sections. As we set some restrictions on the action space, the solution we can reach is sub-optimal.

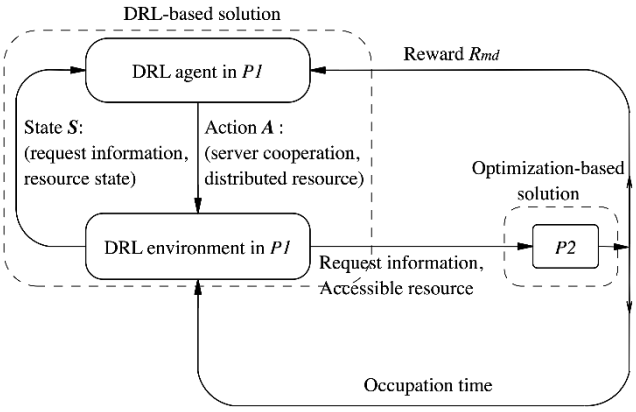



**Figure 4-24. Solution of our proposed minx integer non-linear programming problem; The relationship of P1 and P2 in the problem; The relationship of machine learning based solution and optimization-based solution**

### 4.4.3 DRL-based long-term resource planning

The long-term schedule is realized by building the continuous MEC network offloading into a Markov Decision Process and then solving it with deep reinforcement learning algorithms. The discounted reward used in Markov Decision Process (MDP) can be represented as:

$$\mathcal{R}_d = \sum_{i=0}^{\infty} \gamma^i r_{d+i} \quad (7)$$

Based on that, the action value function is defined to evaluate the return by selecting action $a_d$ at state $s_d$. In this section, we solved the subproblem P1 by deep Q learning method. Its basic principle is a time differential algorithm which composes real observation reward and estimation action value can be written as:

$$Q(s_d, a_d) \approx r_d + \gamma Q(s_{d+1}, a_{d+1}) \quad (8)$$

The state of DRL includes request information over the network (the number of requests, the number of requests with higher latency requirement, the computing resource requirement of all the requests) and the computing resource remainder of all the MEC servers. The action of DRL includes the cooperation status of all servers and the resource reservation status of all servers. The reward is calculated by subproblem P2.

In addition to theoretical simulation, we also provide an artificial intelligent agent placement solution as shown in Figure 4-25. The AI agent will be placed on the control layer. It collects the MEC computing resource and mobile device request information as training data from the lower layers. After training, AI agents can apply neural networks to react in real-time to new network information to maximize the resource utilization and minimize the service latency and power consumption.

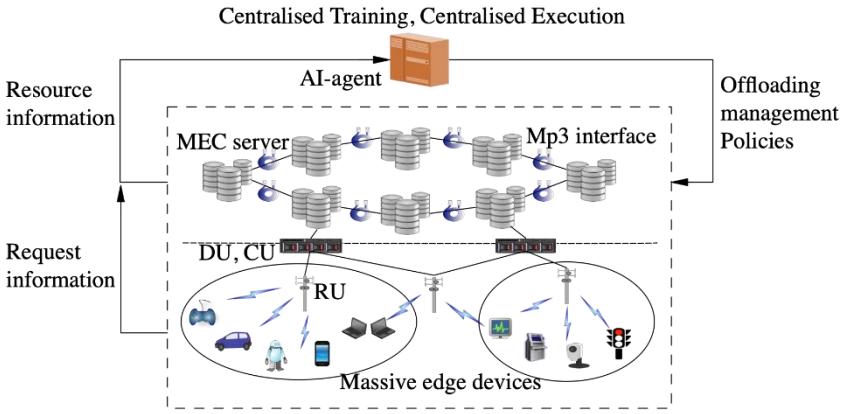



**Figure 4-25. AI agent deployment method and the centralized training and centralized execution architecture**

### 4.4.4 Optimization-based short-term resource management

The subproblem P2 is solved by optimization-based method. Specifically, equation 6 can be then divided into two new subproblems. The first is to optimize the transmission power allocation and to distribute the reserved resource from the subproblem P1. The second is to find out the optimal offloading decision algorithm. Here we provide two corresponding algorithms.

**Algorithm 1** Transmission Power Allocation Policy

Set a small number $\delta$
Calculate $\phi(p_0) = \gamma_n \log_2(1 + a_{nm'}p_0) - a_{nm'}/\ln 2 \cdot (\eta_n + \gamma_n p_0)(1 + a_{nm'}p_0)$
**for** $n$ in $Q^d_{m'd'}$ **do**
  **if** $\phi(p_0) = 0$ **then**
    $p_{nm'} = p_0$
  **else**
    Initialize $p_s = 0$ and $p_t = p_0$
    **while** $p_t - p_s \leq \delta$ **do**
      $p_l = (p_t + p_s)/2$
      **if** $\phi(p_l) \leq 0$ **then**
        $p_s = p_l$
      **else**
        $p_t = p_l$
      **end if**
      $p_{nm'} = (p_t + p_s)/2$
    **end while**
  **end if**
**end for**

**Algorithm 2** Offloading Decision Policy

Based on delay requirement, select $Q_{md}$ from $Q_d$
Offload $Q_d - Q_{md}$ to data center
$time = 0$
Sort $Q_{md}$ in descending order of $\beta^T_n/(data\ size)$
**for** $n$ in $Q_{md}$ **do**
  $time = time + C_n/G_{md}$
  **if** $\beta^t_n > \beta^e_n$ & $time \geq T^l_n$ & $V_{nm'd'} < 0$ **then**
    Reject
  **else if** $\beta^e_n > \beta^t_n$ & $E^r_{nm'} \geq E^l_n$ & $V_{nm'd'} < 0$ **then**
    Reject
  **else if** $V_{nm'd'} < 0$ **then**
    Reject
  **else**
    Offload to $m'$
  **end if**
**end for**

### 4.4.5 **Illustrative example**

The topology of MEC network used for the initial results are included in Figure 4-26. The offloading requests are sent to MEC or Data centre through the order of Radio Unit (RU), Distributed Unit (DU), and Core Unit (CU). In addition, in terms of the supporting technologies of server cooperation, the Mp3 interface is used to realize the communication between MECs [51]. There are overall eight MEC servers in the simulation network.

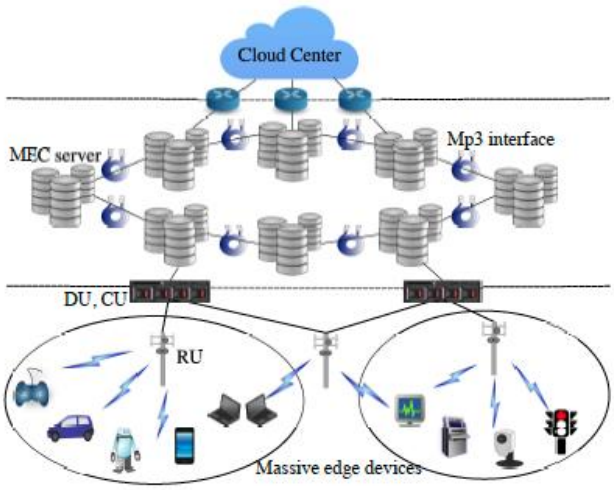



**Figure 4-26. Network environment setup**

Within this network, to resolve the mixed integer nonlinear programming problem, we performed two experiments including the optimization in short term resource offloading and in long-term resource reservation. For short term solution for subproblem P2, Figure 4-27 compares the reward obtained by three algorithms including a State-of-the-art algorithm (Turquois) and all-locally offloading **(**Green) and all uploaded *offloading* (Red) compared to our proposed solution. It can be seen that by reusing the released resources, our proposed solution can greatly improve the offloading reward. Moreover, As the resource become sufficient, our algorithm will offload all the requests on the server and therefore realizing same reward as all uploaded strategy. This is consistent with the trend of convergence of black and red lined as shown in Figure 4-27.

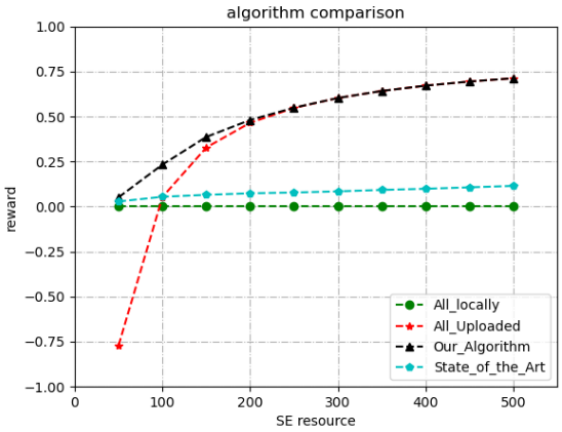



**Figure 4-27. Reward vs resource allocation**

Based on these short-term outcomes as a reward, we trained the DQN model and the training process is shown in Figure 4-28. In addition, in Figure 4-29, we compared three algorithms to schedule resources over 500 time slots including random allocation, over distribution and DQN algorithm. Random allocation is to randomly distribute the resources whenever the server receives requests. Over distribution uses all the resources on the server. DQN learns how to schedule the resources for the upcoming requests. As can be seen from Figure 4-29, the reward of random allocation is distributed between 0.366 and 0.061 and on average of 0.214. In comparison, over allocation can achieve higher reward but with high variance. This proves that the benefits of over allocation on short term rewards are at the expense of future rewards. DQN is a better solution. Although it cannot achieve much higher average rewards than over allocation, it has much lower variance and therefore realizes better reliability for the offloading requests.

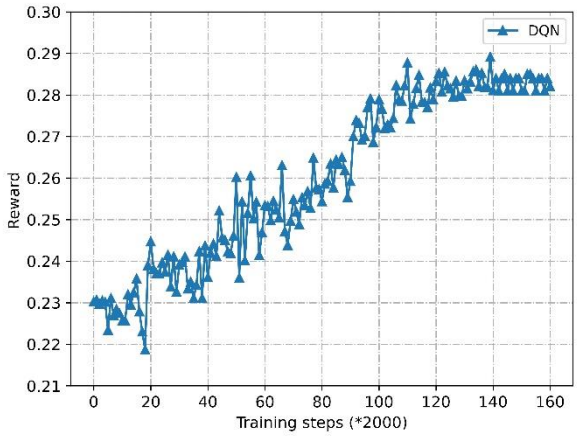



**Figure 4-28. Convergence property of DQN in resource scheduling of subproblem P1**

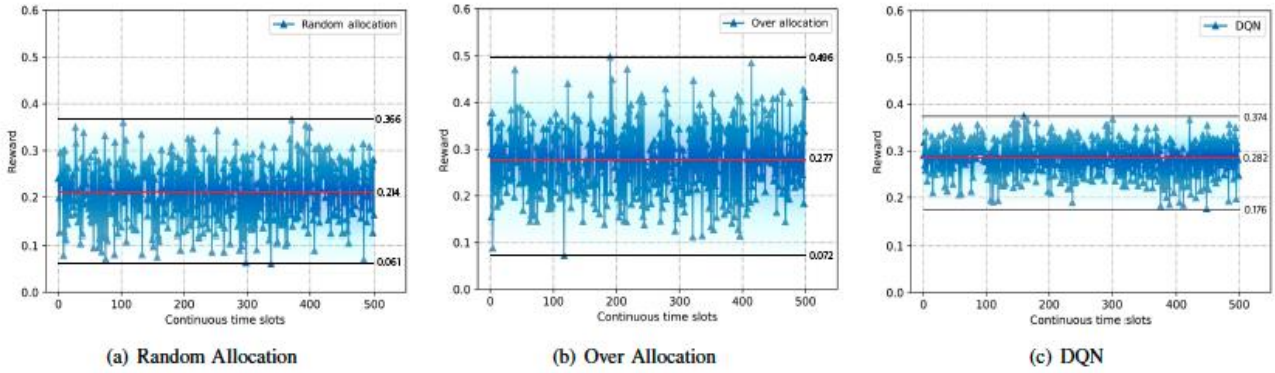



**Figure 4-29. Resource scheduling reward over 500 time slots**

### 4.4.6 Conclusion and future works

Optimal Resource Offloading in B5G networks might support the zero-touch management and improve the resource allocation and quality of services. While allowing the cooperation between edge servers with the inclusion of intelligent capabilities, it is possible to improve the efficiency of long-term and short-term resources and capacity scheduling of tasks and resources associated to them. DRL is able to increase the average reward and the reliability of offloading policies. For increasing the precise of consideration on server cooperation and taking the integrations between servers, future studies could focus on the exploration multi agent deep reinforcement learning algorithms based on actor and critic architecture.

## 4.5 RRP in multi-tech RAN sim extension

Let us assume an industrial private 5G network that includes a multi-Wireless Access Technology (WAT) Radio Access Network (RAN) integrating 5G New Radio (5GNR) and Wi-Fi (Wireless Fidelity). The industrial private 5G network is deployed as a Standalone Non-Public Network (SNPN), i.e., a segregated private 5G network which is not supported by any Public Land Mobile Network (PLMN). The SNPN is managed by unique private network operator. There are two coexisting types of services in the considered scenario: i) non-critical human-centric based services, hereinafter referred to as enhanced Mobile Broadband (eMBB) services, and ii) delay-sensitive services for process automation, from now on referred to as Ultra-Reliable Low Latency Communication (URLLC) services. eMBB and URLLC traffics are served by slices of type eMBB and URLLC, respectively. The RAN consists of $B_G$ gNBs and $B_W$ Wi-Fi access points (being $B_G$ and $B_W$ the total number of 5G gNBs and Wi-Fi access points, respectively). Due to *listen-before-talk* and backoff mechanisms

implemented in the MAC layer of Wi-Fi technology, there are no guarantess that Wi-Fi can meet the strict latency requirements of URLLC traffic. Then, here we assume that URLLC services are only served by 5GNR. In contrast, eMBB traffic can be served by either technology (e.g., 5G or Wi-Fi). The main objectives are to devise ML-based solutions to allocate 5G spectrum to URLLC and eMBB services at every gNB, prioritizing URLLC traffic, and decide the additional Wi-Fi bandwidth required by eMBB slices to meet their throughput requirements.

An initial DRL-based solution together with some preliminary results for the 5G radio resources allocation to URLLC and eMBB traffics were presented in Section 3.6 of the deliverable D4.2 [1]. Here, we will include the following extensions for that solution and its evaluation:

The inclusion of a more realistic and accurate analytical model for training the URLLC agent. This will make the URLLC agent more generic and will provide it with better initial performance in real scenarios.

A complementary solution to offload eMBB users from 5GNR to Wi-Fi. This complementary solution is built upon a heuristic method. It coordinates the operation of the ML-based agents and perform eMBB users offloading to Wi-Fi when 5G NR resources are not enough to meet the services requisites.

The experimentation, testing and validation of the involved agents is going to be extended. The different agents will be integrated into an industrial RAN system-level simulator. The simulator includes accurate simulation models of the different parts of the industrial RAN, which allow us to make an idea of the agents' performance in a real scenario.

### 4.5.1 Solution enhancements

In deliverable D4.2 [1], we have described DQN-based dynamic radio resource provisioning solutions for both eMBB and URLLC slices. In a nutshell, the resulting DQN agents output the minimum amount of PRBs to be allocated to each of the slices in order to fulfil the service level agreements (SLAs). Specifically, we assumed the respective SLAs for URLLC slices include a minimum aggregated throughput, maximum delay and maximum packet loss ratio at the radio interface as performance requisites, whereas the eMBB ones include only a minimum aggregated throughput. Next, we describe the RL model of these two agents, which includes changes with respect to the previous version:

**a. State**

At each time step $t$, the state $s_t$ is obtained from the environment. It can be expressed as $s_t = \{s_t(c,b)\}$, where each element $s_t(c,b)$ represents the state of the slice $c$ in cell $b$. The state provides the indispensable information from the environment required for the agent operation. Table 4-8 gathers the inputs (features of the state) for each kind of agent (URLLC and eMBB). In particular, the state of each of the agents is defined by the following metrics:

URLLC agent: slice throughput to be met according to the SLA $R_{c,b}$, the resource quota or number of PRBs allocated to the slice $c$ in cell $b$ $\xi_{c,b}$, the slice delay requirement $D_{c,b}$, and the Packet Loss Ratio $PLR_{c,b}$, understood as the fraction of lost packets, considering those packets discarded if they are not delivered in an interval time less than $\tau_{max}$.

eMBB agent: slice throughput to be met according to the SLA $R_{c,b}$, the resource quota or number of PRBs allocated to the slice deployed in the cell $\xi_{c,b}$, and summay of users SINR.

**Table 4-8. Design of Agents' State**

| URLLC Agent | eMBB Agent |
|---|---|
| Throughput of slice $c$ in cell $b$ ($R_{c,b}$)<br>Resource quota of slice $c$ in cell $b$ ($\xi_{c,b}$)<br>Slice delay requirement ($D_{c,b}$) | Throughput of slice $c$ in cell $b$ ($R_{c,b}$)<br>Resource quota of slice $c$ in cell $b$ ($\xi_{c,b}$)<br>Summary of users SINR |

| | |
|---|---|
| Slice PLR requisite ($PLR_{c,b}$) | |

**b. Action**

Once the agent has observed the state, it triggers an action in order to adapt the allocated radio resources to the new environment conditions. The action taken at time step *t* for the slice *c* in cell *b* is denoted as $a_t(c,b)$ and it will update the associated resource quota $\xi_{c,b}$ by increasing, decreasing or maintaining the value determined in the previous step *t-1*. The action will modify the quota progressively in steps of a given size Δ. Thus, three different values are possible to be taken by the action, $a_t(s,b) \in \{\Delta_q , -\Delta_q , 0\}$, resulting in the modification of the slice quota, as indicated in the following expression: $\xi_{c,b}(t) = \xi_{c,b}(t-1) + a_t(c,b)$.

**c. Reward**

The reward is the metric that evaluates the goodness of the action $a_t$ that is taken by the agent in step *t* given the state $s_t$. The definition of the reward for both agents (i.e., URLLC and eMBB agents) follows the same philosophy. Algorithm 4-1 shows the reward design:

**Algorithm 4-1: Reward Design of DRL Agents**

| Reward Design |
|---|
| 1: **if** action contributes to meet the slice requisites: |
| 2: Reward = +1 |
| 3: **else if** action worsen the slice requisites: |
| 4: Reward = -1 |
| 5: **end if** |
| 6: **if** action is an invalid action: |
| 7: Reward = -10 |
| 8: **end if** |

One of the main drawbacks of the URLLC agent implementation in D4.2 is the computational complexity exhibited by the analytical model employed for the initial training of the agent. The aforementioned model is proposed and described in [52]. This analytical model is based on a Markov chain. Its main source of computational complexity is a step in which the vector of stationay probabilities has to be found. This is the most time-consuming step as a system of $L_{max}$ linear equations needs to be solved, being $L_{max}$ directly proportional to the number of PRBs used by the gNB to serve the URLLC slice.To overcome this issue, we have sampled several configurations of the scenario (e.g., URLLC UE packet delay budget, packet loss ratio, URLLC slice traffic load, bandwidth allocated to the URLLC slice, and UE spatial distribution). For instance, Figure 4-30 shows the packet loss ratio versus the aggregate traffic load of a URLLC slice for several bandwidth values allocated to it. The curves shown in Figure 4-30 correspond to a URLLC delay constraint of 1 ms and a realization of the UE spatial distribution. Then, we use linear interpolation of those samples to estimate the URLLC performance given a bandwidth allocation during the URLLC agent's training phase, thus expediting this process.

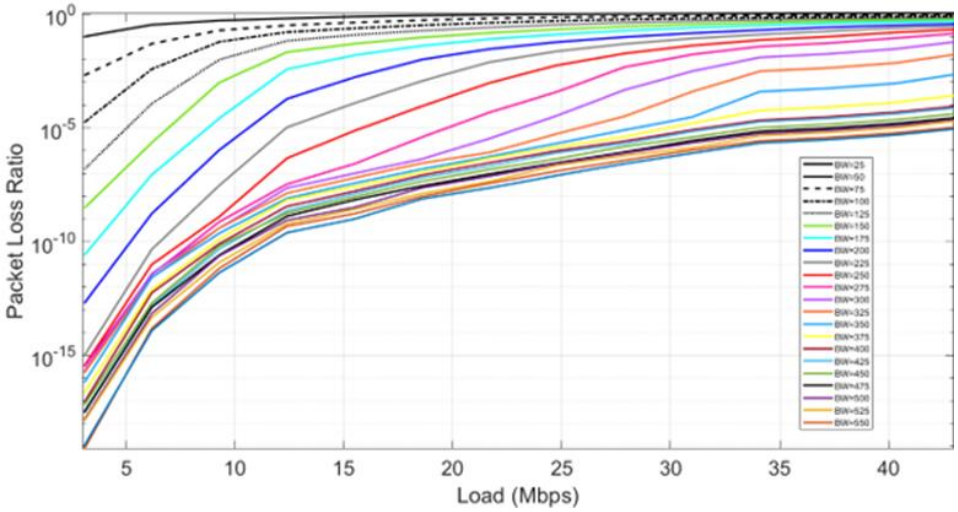



**Figure 4-30. Packet loss ratio as a function of the URLLC traffic load and bandwidth value to ensure a delay of 1 ms and a given realization of the UE spatial distribution**

Figure 4-31 illustrates a graphic representation of the dynamic radio resource provisioning solution for the allocation of radio resources in an industrial private 5G network. The radio resource provisioning solution is based on a proactive mechanism which is executed every Δt units of time in order to estimate the amount of radio resources required to meet the performance agreed on the SLAs for a given network workload.

Deepening on the specific operation of the radio resource provisioning solution, every gNB consists of an URLLC and an eMBB agent, both responsible for allocating 5G radio resources to the URLLC and eMBB slices, respectively. In other words, there is an agent per slice and per gNB. It is worth highlighting that 5G radio resources are prioritized for URLLC slices due to their stringent latency constraints. Taking this assumption into consideration, the URLLC and the eMBB agents will estimate the amount of radio resources required to ensure an agreed performance given the observed network workload. Then, the offloading agent will decide whether to offload eMBB users in order to be served by Wi-Fi technology depending on the availability of 5G radio resources. In other words, given the amount of 5G radio resources estimated to be allocated to the URLLC and eMBB slices by their respective agents, the offloading agent will check if there are enough 5G radio resources in the gNB to serve both slices. In the case in which the slices demand exceeds the available 5G bandwidth, the offloading agent will offload to Wi-Fi those eMBB users with weakest 5G SINR level perceived.

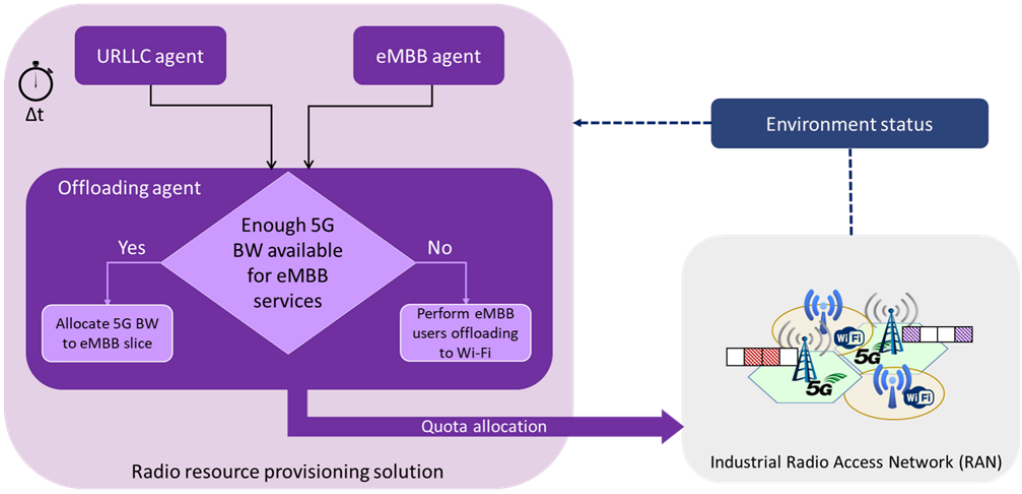



**Figure 4-31. ML-based radio resource provisioning solution for an industrial RAN**

In this way, the output of the "Radio resource provisioning solution" module will be the 5G and Wi-Fi radio resource quota allocated to each slice.

The design of the offloading algorithm is specified in the form of pseudocode below:

**Algorithm 4-2: Offloading Algorithm Executed by the Offloading Agent**

| **Offloading algorithm** |
|---|
| A list of eMBB users ($U_{eMBB}$) ordered in an ascendent way according to the SINR level they perceive from 5G is obtained |
| Select the first N users of the $U_{eMBB}$ list in order to be offloaded to Wi-Fi APs, obtaining the users list $N\_U_{eMBB}$ |
| $sk(u) = 0 \; \forall \; u \in N\_U_{eMBB}$ \\Selected Wi-Fi APs per user |
| **for** each user $u \in N\_U_{eMBB}$: |
| Obtain a list of Wi-Fi APs per user if the received SINR is greater than a defined threshold $A$ = APs $(SINR_{UE} \geq SINR_{thr})$ |
| **while** $(A \neq \emptyset)$ **do** |
| $k = \underset{\forall\, k \in A}{argmax} \{SINR_{k,u}\}$ |
| **if** !isAPoverloaded($k$) **then** |
| $sk(u) = k$ |
| **break** |
| **else** |
| $A = A \setminus k$ \\AP $k$ is removed from set $A$ |
| **end if** |
| **end while** |
| Go to the next user in the list $N\_U_{eMBB}$ |
| **end for** |

## 4.5.2 Evaluation results

This subsection includes the evaluation results of the DRL-based radio resource provisioning solution described previously. Specifically, it describes the system model considered, the scenario setup used for testing and validating the agents operation, and the main results obtained.

### 4.5.2.1 System model and testing scenario setup

For the performance evaluation of our proposed solution, we consider the same industrial scenario as the one described in Section 3.6.2.1 of deliverable 4.2 [1], which tries to resemble the BOSCH factory under study in 5G-CLARITY UC2.1 [53]. Figure 4-32 represents the abstract model of the testing scenario under consideration. As our proposed solution is based on an agent per slice and gNB, the testing scenario consists of a 5G gNB, one URLLC slice (which comprises the industrial devices in charge of controlling the industrial processes), one eMBB slice (which comprises the data-hungry users and factory workers), and several Wi-Fi access points to which eMBB users are offloaded.

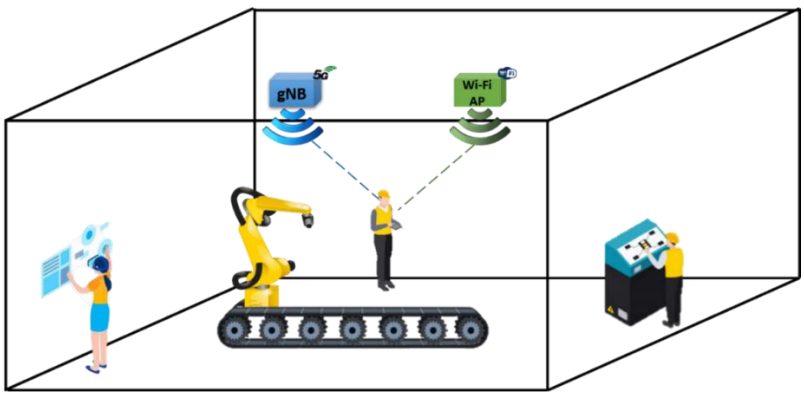



**Figure 4-32. System model of the testing scenario**

#### 4.5.2.2 Setup and results

In this section the setup of the DQN agents hyperparameters, the obtained results together with their description are shown.

As described in D4.2, the design of the DQN agents is based on a critic representation. The development of the agents has been carried out using Stable-Baselines3, which is a set of reinforcement learning algorithms in PyTorch. Since the design of the agents has been modified, the configuration of some of the DQN agents hyperparameters has suffered some changes with respect to D4.2 (e.g., discount factor and epsilon). The architecture of the neural network is the same as the one described in D4.2 (see Section 3.6.3.1 of [1]). Table 4-9 shows the settings of the main parameters related to the configuration of the neural network and the DQN agent hyperparameters:

**Table 4-9. Design of the DQN Agent Hyperparameters**

| DQN Agent Hyperparameters | Configuration |
|---|---|
| Reinforcement learning method | DQN with critic network (value based) |
| Learning rate | 0.001 |
| Mini-batch size | 32 |
| Discount factor (γ) | 0.95 |
| Target update frequency | 4 |
| Target update method | Periodic |
| **ε-greedy exploration** | |
| Epsilon | 1 |
| EpsilonMin | 0.05 |
| EpsilonFraction | 0.3 |

The obtained results of the training process of the URLLC agent are shown in Figure 4-33. This figure includes several parameters of the training process against the number of steps taken for training the agent. Particularly, the mean reward, the mean episode length in number of steps, the exploration rate, the loss function and the number of Frames Per Second (FPS). Something remarkable regarding the mean reward graphic is that compared to the agent of deliverable 4.2, the agent described in this deliverable needs more training steps to get the convergence due to the enhancement of the agent design with. Specifically, the URLLC agent in the D4.2 [1] needed 200000 steps to reach the convergence, whereas now the agent needs around two million steps to start reaching the convergence.

Therefore, in Figure 4-33 we can also see that the convergence of the agent begins to get reached when the exploration rate is lower. In other words, the agent convergence begins to get reached when the probability of the environment exploration is lower, thus increasing the probability of the exploitation phase since the agent has learnt more about the environment.

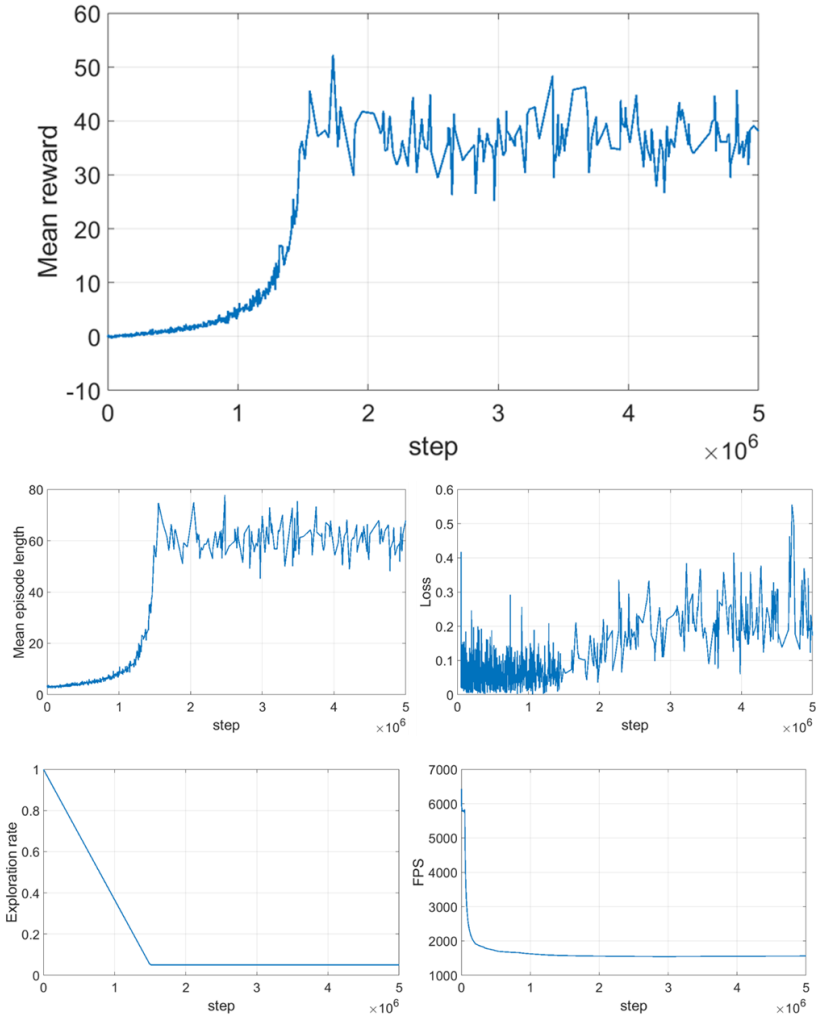



**Figure 4-33. Graphics related to the training process of the URLLC agent**

Figure 4-34 shows a validation of the URLLC agent for a slice whose requirements is a delay of 1 ms and a PLR of $10^{-4}$.

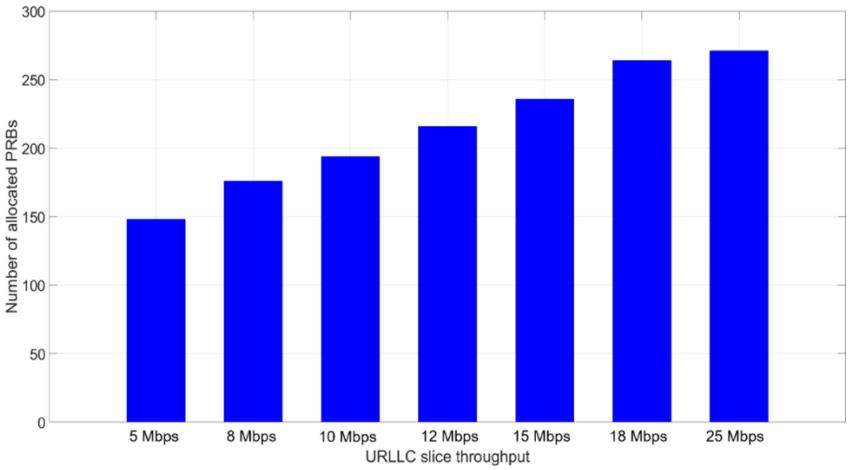



**Figure 4-34. Validation of URLLC agent operation**

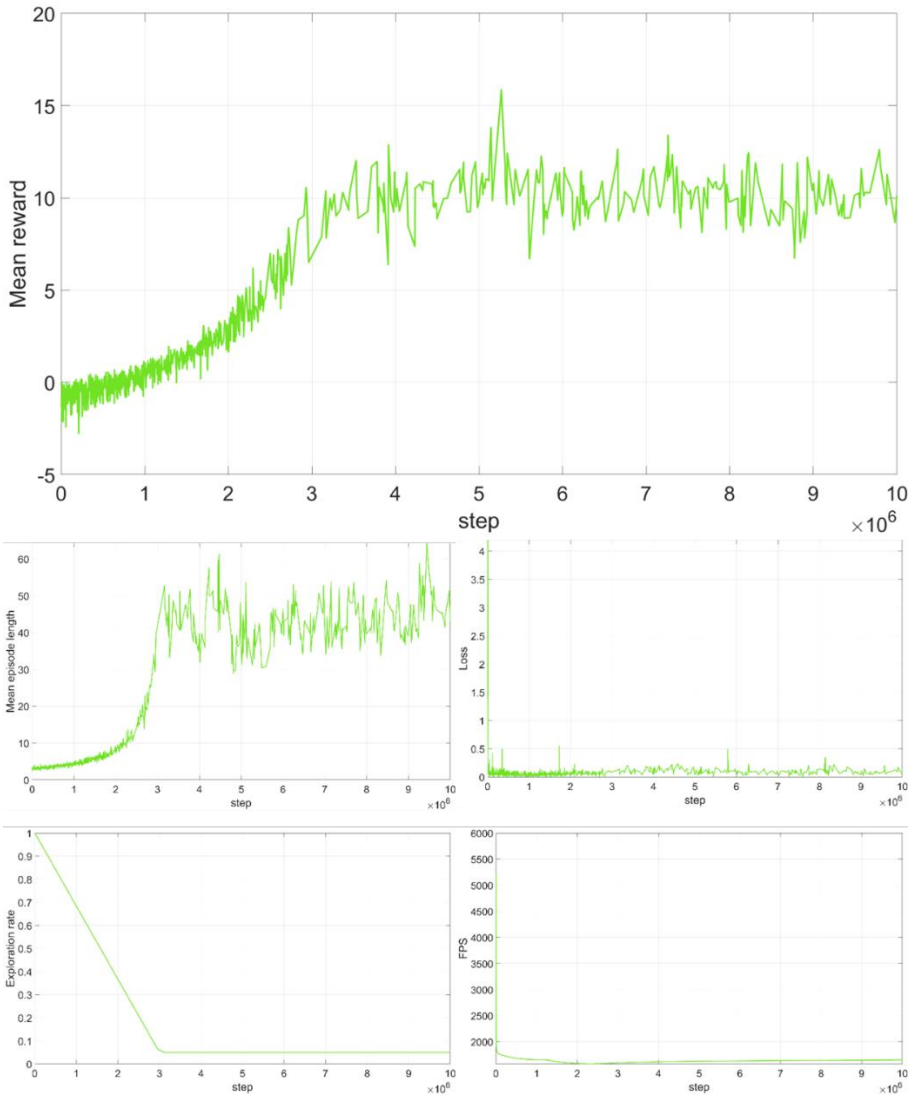



**Figure 4-35. Graphics related to the training process of the eMBB agent**

In the figure we can observe how the URLLC agent adapts the amount of radio resources (PRBs) allocated to the URLLC slice depending on its throughput requirements, while the delay and packet loss ratio requirements are met.

In the same way as with the URLLC agent, Figure 4-35 shows the same parameters exhibited in Figure 4-33 for the training process of the eMBB agent. Please note that, in this case, the agent needs around 4 million steps to reach the convergence for the same configuration of the hyperparameters (see Table 4-9). If we compare the mean episode length of the eMBB agent with the mean episode length of the URLLC agent (see Figure 4-33) we can see that the values reached by the URLLC agent are higher. This is due to the number of PRBs that have to be allocated to the respective slices. So, from these figures we can deduce that, in average, the numer of PRBs required to serve the URLLC slice is higher than the number of PRBs allocated to the eMBB slice.

For the validation of the eMBB agent we have performed a number of tests for several configurations of the testing scenario. Specifically, we have checked the adaptation of the resource allocation agent operation to different throughput requirements of the eMBB slice and for different spatial distributions of the eMBB users that conform to the eMBB slice. Figure 4-36 and Figure 4-37 depicts the validation of the operation of the eMBB agent, each for a different configuration of the spatial distribution of eMBB users in the scenario.

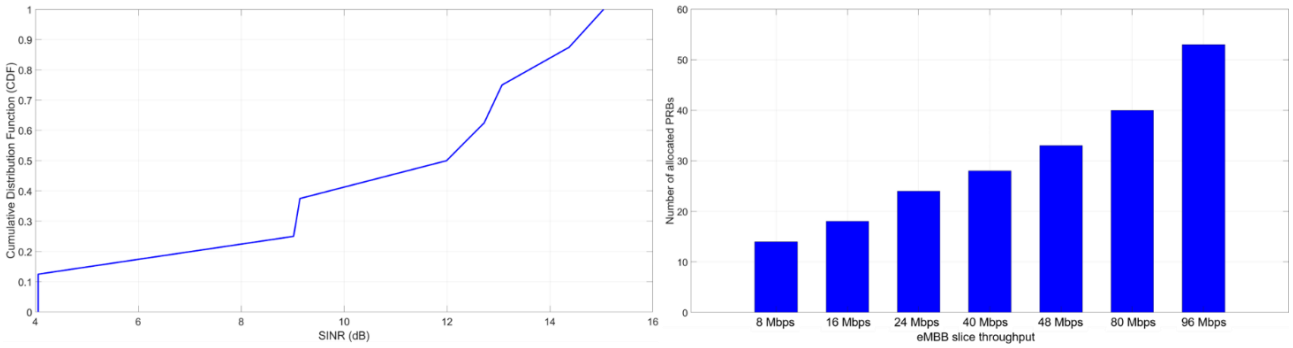



**Figure 4-36. CDF of SINR of eMBB users and operation of the eMBB agent for the scenario configuration 1**

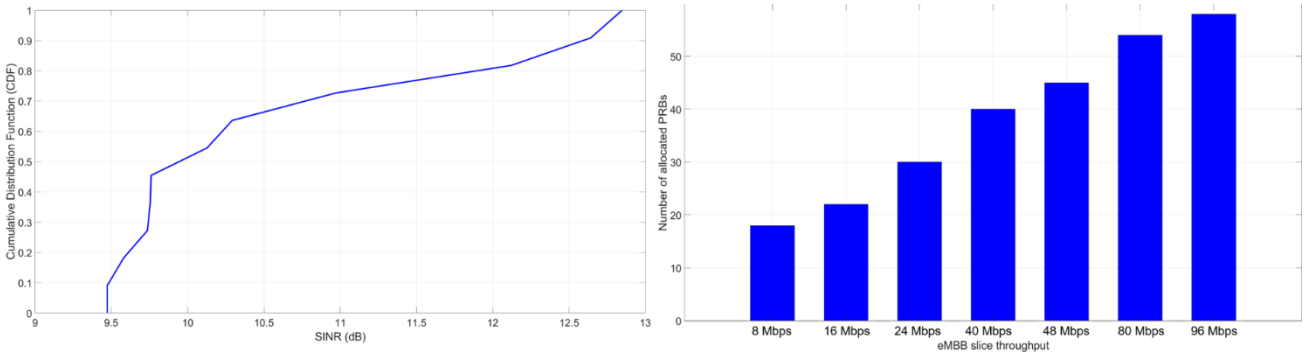



**Figure 4-37. CDF of SINR of eMBB users and operation of the eMBB agent for scenario configuration 2**

The left graphics of Figure 4-36 and Figure 4-37 represent the Cumulative Distribution Function (CDF) of the SINR of the users belonging to the eMBB slice and served by the gNB in which the agent is being tested. The right graphics of Figure 4-36 and Figure 4-37 depict the number of PRBs computed by the agent for several values of the eMBB slice throughput requirement.

From Figure 4-36 and Figure 4-37 we can conclude that the eMBB agent works properly. We can observe that in the second scenario more radio resources are required to meet the same throughput requisites due to lower levels of the users perceived SINR, as can be deduced from the graphics depicting the CDF of the SINR.

Also, observing Figure 4-34, Figure 4-36 and Figure 4-37, and as it was mentioned before, we can notice that the number of allocated PRBs to the URLLC slice is significantly higher than the number of PRBs allocated to the eMBB slice. This makes evident that meeting the more stringent requites (in this case in terms of latency and PLR) entails spending more resources. From this result, we can deduce that serving URLLCs is more expensive, at least in terms of radio resources and, in turn, of 5G spectrum.

Lastly, some results related to the offloading agent operation are included. Table 4-10 includes the requirements of both URLLC and eMBB slices set to be met. The DQN agents will compute the radio resources needed in the gNB to meet the requisites.

**Table 4-10. Requirements of URLLC and eMBB slices**

| Requirement | Configuration |
|---|---|
| **URLLC slice** | |
| Throughput (Mbps) | 30 |
| Packet Loss Ratio | $10^{-4}$ |
| Delay (ms) | 1 |
| **eMBB slice** | |
| Throughput (Mbps) | 100 |
| Number of users | 16 |

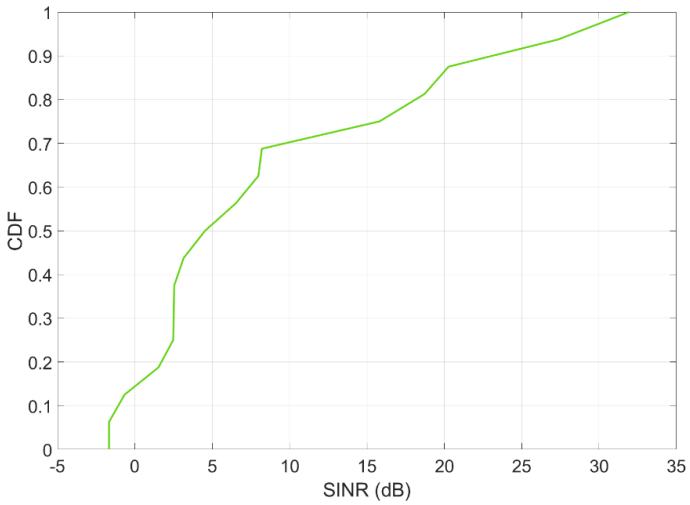



**Figure 4-38. CDF of the SINR of eMBB users**

Figure 4-38 depicts the Cumulative Distribution Function (CDF) of the SINR of the eMBB users under the coverage area of the gNB of the testing scenario considered. This figure provides information about the SINR level of the users that are being served by 5GNR before performing the offloading algorithm. As can be observed in the figure, around the 15% of the users have poor level of SINR. This implies that these users need larger amounts of radio resources to satisfy their throughput requirements.

Under the mentioned service requirements (see Table 4-10) and the testing scenario specificities, the trained DQN agents compute the amount of PRBs needed to provide the requisites of the services (i.e., URLLC and eMBB). Table 4-11 shows the amount of PRBs computed by each agent.

It can be noticed that both URLLC and eMBB agents are operating in the same gNB (see the system model of the testing scenario in Figure 4-32). Consequently, the total number of PRBs must not exceed the total bandwidth available in the gNB, which here is assumed to be 100 MHz (555 PRBs considering a PRB bandwidth of 180 KHz). In this example, as can be calculated from Table 4-11, a total of 602 PRBs would be required to serve both slices. Since the total number of PRBs required to serve the slices exceeds the total available bandwidth of the gNB and the URLLC slice is prioritized to be served by 5G NR, the offloading agent performs eMBB users offloading to Wi-Fi. Table 4-12 shows the number of PRBs computed by the eMBB agent once the offloading procedure has been carried out. Moreover, Figure 4-39 the CDF of the SINR of eMBB users served by 5G NR and Wi-Fi after performing eMBB users offloading. It can be observed that the minimum values of the user's perceived SINR have increased compared to the ones measured before the offloading procedure was carried out (see Figure 4-38).

**Table 4-11. Number of PRBs Computed by the DQN Agents**

| Type of Agent | Number of PRBs |
|---|---|
| URLLC | 328 |
| eMBB | 274 |

**Table 4-12. Offloading Performance Results**

| | Before Offloading Performance | After Offloading Performance |
|---|---|---|
| Number of eMBB users served by 5G NR | 16 | 12 |
| Number of eMBB users served by Wi-Fi | 0 | 4 |
| Number of PRBs computed by the eMBB agent | 274 | 136 |

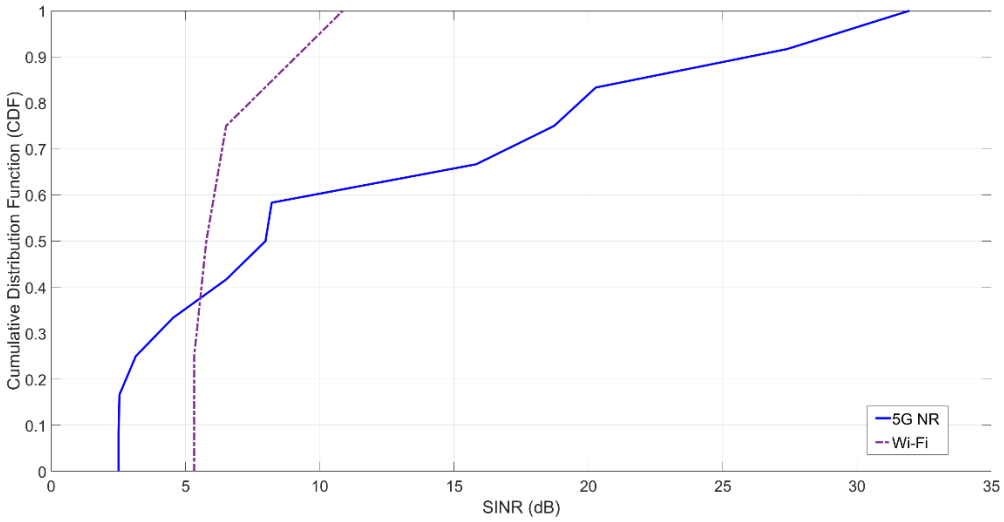



**Figure 4-39. CDF of the SINR of eMBB users served by 5G NR and Wi-Fi after offloading procedure**

## 4.6 Long-term transport network setup

Let us assume a multi-tenant private 5G network consisting of a 5G System (5GS) whose components are interconnected through a layer 2 Time-Sensitive Networking (TSN) network. We consider the Asynchronous Traffic Scheduler (ATS) to arbitrate the transmission of the packets through the different shared Data Plane (DP) resources whose queuing delay might be significant. There are $M$ tenants, each with $n_m$ slices. Then, $N = \sum_{m=1}^{M} n_m$ is the total number of slices to be accommodated. The goal is to find a prioritization of the $N$ slices at each DP resource so that the end-to-end (e2e) delay and jitter constraints of every slice are met. Considering the ATS to arbitrate the access to the different resources, the worst-case packet delay $D_\tau^e$ and jitter $J_\tau^e$ for the slice $\tau$ at its priority level $p_\tau^e$ in the resource $e$ are given by [54] [55]:

$$D_\tau^e = \frac{\sum_{k=1}^{p_\tau^e} \hat{b}_k^e + \hat{l}_{p_\tau}^e}{C_e - \sum_{k=1}^{p_\tau^e - 1} \hat{r}_k^e} + \frac{l_\tau}{C_e},$$

$$J_\tau^e = \frac{\sum_{k=1}^{p_\tau^e} \hat{b}_k^e + \hat{l}_{p_\tau}^e}{C_e - \sum_{k=1}^{p_\tau^e - 1} \hat{r}_k^e},$$

where $\hat{r}_k^e$ and $\hat{b}_k^e$ are the aggregated rate and aggregated burstiness or burst size at the priority level $k$ of the resource $e$, respectively. $\hat{l}_{p_\tau}^e$ is the maximum frame size allowed in the priority levels lower than the priority level $p_\tau^e$ assigned to the 5G-CLARITY slice $\tau$. $C_e$ denotes the minimum nominal packet transmission at resource $e$. And $l_\tau$ stands for the maximum packet size generated by the 5G-CLARITY slice $\tau$. The e2e worst-case packet delay and jitter for the slice $\tau$ can be computed as:

$$D_\tau = \sum_{e \in \mathcal{E}} D_\tau^e$$

$$J_\tau = \sum_{e \in \mathcal{E}} J_\tau^e$$

An initial DRL-based solution for the URLLC prioritization at the Transport Network (TN) together with some

preliminary results validating the proposed solution were presented in Section 3.7 of the 5G-CLARITY deliverable D4.2 [1]. Here, we will include the following extensions for that solution and its evaluation:

- The solution will be generalized to make it valid for a wider spectrum of configurations. In D4.2 [1], the agents were trained and tested to perform the slice prioritization for a specific configuration. Here, we refine the design of the solution in order to make it more generic. Specifically, the solution comprises an agent instance in charge of the IEEE 802.1Q Traffic Classes (TCs) prioritization for each bridge output port, the clustering of the 5G-CLARITY slices to map them onto the eight TCs, and the delay/jitter budget distribution along the different forwarding plane devices.
- Regarding the solution testing, in this deliverable, we put the emphasis on assessing and showing the solution generalization capacity, i.e., the ability of the solution to find valid configurations in unseen environments that were not used during the training phase.
- We extend the evaluation of the solution to assess its proper operation in a more complex scenario. Particularly, the scenario will include a more complex topology for a private TN.
- Last, we include a study of the RL agent hyperparameters tunning (e.g., number of neurons in the neural network, exploration-exploitation balance, and learning rate).

### 4.6.1 Solution architecture description

In 5G-CLARITY deliverable D4.2 [1], we have presented a preliminary design of an autonomous RL-based solution for configuring the transport network in the 5G-CLARITY system. Here, we enhanced that solution to make it more general. For instance, the solution described in 5G-CLARITY D4.2 depends on the number of 5G-CLARITY slices and, therefore, specific RL agents must be developed and trained for the specific scenario. The problem addressed by the solution presented below is described in the previous subsection. Although the problem statement considers asynchronous TSN as layer 2 technology, the solution can be easily adapted to work with bare IEEE 802.1Q Ethernet. To that end, it is only required to change the underlying worst-case delay/jitter models to train the RL agents. Deriving tight worst-case delay for standard Ethernet has a high computational complexity [56]. However, a compositional analysis, i.e., the e2e worst-case delay/jitter is computed as the sum of the per-hop worst-case delay/jitter models, can be used to overcome the problem complexity at the expense of reducing the network resources utilization.

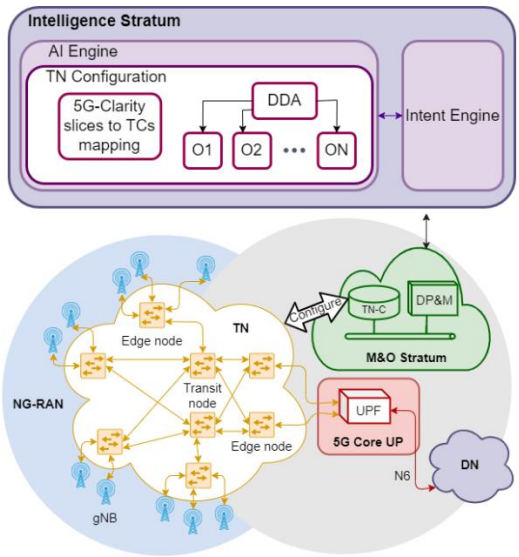



**Figure 4-40. High-level RL-assisted 5G-CLARITY TN configuration solution**

Figure 4-40 depicts a sketch of the primary components of the proposed solution and the primary 5G-

CLARITY system entities interacting with the solution. The solution gets the required telemetry and data analytics from the Data Processing and Management (DP&M) entity of the 5G-CLARITY system. On the other hand, the TN Controller (TN-C) is responsible for applying the configuration computed by the solution. The solution comprises three major components:

- **5G-CLARITY slice clustering**: When the number of 5G-CLARITY slices is greater than eight, this block, labelled as "*5G-CLARITY slices to TCs mapping*" in Figure 4-40, is invoked to cluster them into eight groups. In this way, the solution becomes independent of the number of 5G-CLARITY slices to be accommodated in the TN. For simplicity, in this deliverable, we use k-means to realize this block. The goal of the proof-of-concept carried out in this deliverable aims just to show this block functionality and k-means serves this purpose. Nonetheless, more sophisticated ML-assisted solutions could be used to improve the degree of optimality of this component. These solutions could rely on the output of components below for their training.
- **Delay/jitter distribution agent (DDA)**: This component, labelled as "*DDA*" in Figure 4-40, is responsible for distributing the e2e delay/jitter budget among the TN hops. This agent is invoked for every IEEE 802.1Q traffic class (TC) and source-destination pair. The paths interconnecting the different source-destination pairs are predefined and their computation is out of the scope of the solution described here. This block uses a summary of the aggregated traffic (e.g., aggregated data rate, maximum frame size, and aggregated burstiness) of each TC at each hop in the respective path, and the nominal capacities of each link in the path.
- **TC prioritization at each TN device output port**: This agent, whose instances are labelled as "*O1, O2…, ON*" in Figure 4-40*,* is in charge of prioritizing the TCs at every TSN device output port. Therefore, either there is an agent instance, or the agent must be invoked one time per TN device output port.

The solution's components instances are run sequentially and in the same order as in the list above. First, the 5G-CLARITY slice clustering instance is run to map the 5G-CLARITY slices onto the IEEE 802.1Q TCs. This information is encoded in the Priority Code Point (PCP) field of the IEEE 802.1Q header. Then, the DDA is run as many times as required to distribute the delay/jitter budget for every TC and every source-destination pair. The delay/jitter budget for a given TC is set to the most stringent delay/jitter requirement of all the 5G-CLARITY slices mapped onto that TC during the 5G-CLARITY slices clustering process. Last, the TC prioritization at each TN device output port is computed and the TN-C configures the TSN-devices accordingly. The resulting KPIs of interest from the TN configuration found by the solution can be monitored and the different agents can be properly rewarded. Next, we provide details on the design and modelling of each component of the solution.

### 4.6.2 Solution design and components modelling

Below we specify the key aspects of the design and modelling of the three components making up the proposed AI-assisted TN configuration solution:

5G-CLARITY slice clustering (5CSC): For this component, we use k-means algorithm to cluster the 5G-CLARITY slices into eight groups, each standing for an IEEE 802.1Q TC. We choose k-means for its simplicity, interpretability and explainability. Besides, we enhance the method by including adjustable weights for the different features used to characterize each TC. These weights set the importance of each feature considering the optimization objective. They are adjusted using a trial-and-error approach. More specifically, the weights are randomly sampled, and the goodness of each sample is measured according to a reward function based on the global optimization goal of the solution. The weights configuration that results in the highest goodness is selected. The reward function is proportional to the number of flows allocated in the

network and the number of priority levels used at each TSN bridge's output port. Observe that the computation of the reward requires coordination with the rest of the solution's block, which ultimately results in a higher degree of optimality of the solution. The main features considered to characterize each 5G-CLARITY slice and perform the clustering are: i) the TN delay and jitter requirements, ii) the aggregated data rate, iii) the aggregated burstiness, and iv) the maximum frame length of the slice. We choose these features because the worst-case delay model of the ATS-based networks included in TSN standard is a function of them [54].

- Delay/jitter distribution agent (DDA): We opt for an RL agent to make this decision. More precisely, this agent is in charge to assign a percentage of the delay/jitter budget to a hop for each source and destination pair. Observe that DDA could also distribute the delay per TC, though for simplicity here we carry out the delay distribution per path. If there is a conflict between the delay budget assigned by the DDA for different source-destination pairs and the same TC, then the most stringent delay/jitter constraint is considered for that TC. The key aspects of the associated RL model are summarized below:
    - **Environment**: L2 TN comprising an arbitrary number of networking devices with traffic prioritization support at every output port. The sources and destinations at the network edge are interconnected through predefined paths. The network diameter considered here is seven hops, i.e., the DDA developed here can perform the TN delay distribution for paths with a number of hops less than or equal to seven.
    - **Actions**: The set of DDA's actions considered here is $A_{DDA} = \{0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$, where each action stands for the percentage of the e2e TN delay budget assigned to a hop given the current step within an episode. For instance, in the first step of any episode, DDA assigns a percentage of the delay budget to the first of hop of the path interconnecting a source with a destination. Please note that a larger set of actions may be considered to increase the granularity of the delay budget distribution.
    - **Observations**: IEEE 802.1Q TC characteristics per hop $h$: TC utilization $U_\tau^h$ (aggregated data rate of the TC divided by the link capacity), TC burstiness or maximum burst size $B_\tau^h$, and TC maximum frame size $l_\tau^h$, and TC Delay/Jitter requisite $D_{h,\tau}^{QoS}$.
    - **Reward**:
        - For each step, the agent is rewarded with +10 if the prioritization problem at the respective hop is solved given the percentage of delay budget assigned by DDA. Otherwise, DDA is penalized with -10. Observe that we rely on the TCPA described below to solve the prioritization problem at each hop.
        - At the end of the episode, the agent is rewarded with +100 if the prioritization problem is solved for all the nodes in the path. Otherwise, the agent is penalized with -100.
        - At the end of the episode, the agent is rewarded with +50 if the sum of the percentage of delay budgets assigned to each hop of the path equals 100. That is to encourage the agent to consume the whole delay budget. If the sum of the percentage of delay budgets assigned to each hop is greater than 100, the agent is penalized with -50.
    - **Terminal states**: Each episode has a maximum number of seven steps. However, if at a given point the sum of the per hop delay budget is greater than the e2e TN delay budget, the episode is interrupted, and the agent penalized with -50 as mentioned above.

- TC prioritization at each TN device output port agent (TCPA): Again, we use an RL agent to perform the TC to priority assignment at each bridge output port's scheduler handling the frames transmission of a given link. We have included some changes to the original design of this agent described in 5G-CLARITY D4.2 [1]. For instance, the observations, action space and reward have been refined to make the agent design more efficient and effective. The summaries of the primary parts of the respective RL model are the following:
    - **Environment**: Strict priority scheduler at any L2 networking device egress port. Eight priority levels are considered as it is the default value considered in IEEE 802.1Q standards. Although the setup in this deliverable refers to TSN ATS-based TNs, the proposed TCPA design might be also valid for bare IEEE 802.1Q networks with support for prioritization.
    - **Actions**: The set of TCPA's actions considered here is $A_{TCPA} = \{\downarrow_\tau \ \forall \tau \in TC\} = \{\downarrow_1, \downarrow_2, \downarrow_3, \downarrow_4, \downarrow_5, \downarrow_6, \downarrow_7, \downarrow_8\}$, where $\downarrow_\tau$ stands for the agent decreases the priority level of the TC $\tau \in [1, TC]$. At the beginning of each episode, the highest priority level (priority 1) is assigned to all the eight TCs, each identified by a priority code point. At each step, the priority level of the respective TC is lowered according to the action issued by the agent.
    - **Observations**: IEEE 802.1Q TC characteristics and setup, namely, TC utilization $U_\tau$ (aggregated data rate of the TC divided by the link capacity), TC burstiness or maximum burst size $B_\tau$, and TC maximum frame size $l_\tau$, and TC Delay/Jitter requisite divided by the time required to transmit the maximum frame size of the TC $\frac{D_\tau^{QoS}}{D_F}$, and the priority assigned to the TC $P_\tau$.
    - **Reward**: The rationale behind the reward proposed for this agent is that decreasing the priority of a given TC $\tau$ increases or does not affect its worst-case delay, but it decreases or does not affect the worst-case delay of the rest of the TCs. In other words, lowering the priority of a TC only might have a negative impact on itself, but it benefits all the other TCs. Below is a summary of the reward function:
        - Each action is rewarded with +N, where N is the number of delay/jitter requirements met after the action because of the action (before the action they were not met)
        - Each step has a default reward of -0.5 in order to minimize the required number of steps.
        - If the problem is solved at any time (the delay requirement is fulfilled for all the traffic classes), the episode is finished (terminal state) and the agent is rewarded with +100.
    - **Terminal states**: Each episode has a maximum number of 28 steps. As we are considering eight TCs and eight priority levels, this number of steps is enough to enable the agent setup any TC prioritization in an episode. However, if at a given point the prioritization is solved (the delay/jitter constraints are met for all the TCs), the episode is interrupted, and the agent rewarded with -100 as mentioned above.

**Algorithm 4-3. Master Algorithm to Coordinate the RL Agents of the Transport Network Setup Solution**

| Transport Network Setup Master Algorithm |
|---|
| slices_to_pcp = 5G_CLARITY_slices_clustering(Slices_features, N_TCs=8); // (5GSC) |
| **For** each source s: |
| **For** each destination of the source s: |
| path[s][ $d_i^s$]= Select_path( s, $d_i^s$, network ); |

```
5:          Endfor
6:    Endfor
7:    For each path r:
8:          Distribute_delay_budget_among_hops(r, network); // (DDA)
9:          Update_delay_budget_per_TC
10:   Endfor
11:   For each link l:
12:         Compute_TCs_Prioritization(l, network); // (TCPA)
13:         If the delay/jitter constrai nt is unfulfilled for any TC then:
14:               break;
15:         Endif
16:   Endfor
```

Algorithm 4-3 shows the operation of the master algorithm considered to coordinate the different components described above. First, the 5G-CLARITY slices clustering is carried out (line 1). Then, once the paths interconnecting the different sources and destinations are established, the delay distribution among the links involved in the different paths is performed using the DDA (lines 8 and 9). If a given link is shared among several paths and each impose different per TC delay constraints on it, the most stringent delay budgets are considered for that link. Last, the TCPA performs the TCs prioritization at every link (line 12).

### 4.6.3 Evaluation results

This subsection includes the evaluation results of the RL-based TN configuration solution. Asynchronous TSN or ATS-based TN networks are considered for all the experiments reported below. However, it shall be noted that the proposed design is also compliant bare IEEE 802.1Q-based TNs (without TSN support) with traffic prioritization support. To that end, only the appropriate environment must be used for the agents training. The different agents were developed in Python using Stable Baselines3 in PyTorch. The trainings and evaluations are based on simulation. OpenAI Gym were used to develop the different training environments. The trainings were carried out in a server with two processors Intel(R) Xeon(R) CPU E5-2603 v4 @ 1.70GHz and 32 GB of RAM.

As mentioned, the solution has been extended and refined to improve the agents' learning efficiency and to make it more general compared to its initial design described in 5G-CLARITY deliverable D4.2 [1]. The evaluation study of the solution is also extended in the following ways:

- An initial study of the learning rate and discount factor hyperparameters tuning is carried out for the TCPA using a grid search approach.
- The capacity of the RL TCPA generalization is assessed. In [1], the agents were trained specifically for the target scenario. The TCPA used here has been trained using a wider range of scenarios in order to improve its generalization. Then, a first idea of the number of experiences required by the agent to offer a fair capacity of generalization can be provided.
- Last, a more complex scenario than the one in [1] is considered to validate the operation of the TN configuration solution. In contrast to [1], the solution described in this deliverable includes refinements for the TCPA and two additional components (i.e., 5G-CLARITY slices clustering into traffic classes identified by a PCP and delay distribution agent).

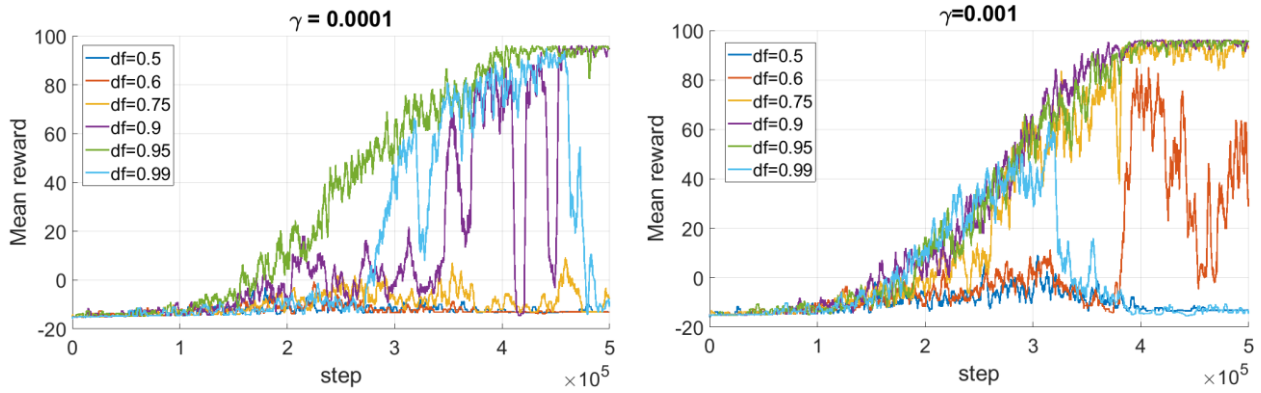
Let us start with the hyperparameters tuning study. Hyperparameters are those parameters that serve to control the learning process and therefore must be set in before it starts. In the context of DRL, examples of hyperparameters are the learning rate, mini-batch size, experience buffer length, exploration-exploitation balance, discount factor, and topology and number of neurons of the critic network, among others. Hyperparameters tuning is crucial as the agent performance and the training process efficiency heavily

depend on them. In this study, we use grid-search technique to gauge the impact of the learning rate and the discount factor on the agent performance and efficiency of the training process. Grid search is one of the simplest algorithms used for hyperparameters tuning and suitable only for tuning a low number of hyperparameters due to its computational complexity. Alternative techniques to ameliorate the complexity exhibited by grid search include random search and Bayes search. Specifically, in grid search, a discrete set of values for each hyperparameter under consideration is defined and explored. For instance, given hyperparameters $h_1$ and $h_2$ and their respective sets if values $H_1 = \{u_1, u_2\}$ and $H_2 = \{v_1, v_2\}$, grid search technique exhaustively assesses the training process and the resulting agent performance for the four possible combinations, i.e., $(h_1 = u_1, h_2 = v_1)$, $(h_1 = u_1, h_2 = v_2)$, $(h_1 = u_2, h_2 = v_1)$, and $(h_1 = u_2, h_2 = v_2)$.

As mentioned, in our hyperparameters tuning study, we consider the discount factor $df$ and the learning rate $\gamma$. On the one hand, the discount factor takes values from the real interval $[0, 1]$ and establishes the importance of the reward to be obtained after several steps $N$ given the current state. On the other hand, the learning rate that also takes values from the real interval $[0, 1]$ determines the step size towards minimizing the loss function at each weight update iteration. Specifically, in our setup, the learning rate is a parameter of the stochastic gradient descend algorithm used to optimize the weights of the critic network. The set of values considered for $df$ and $\gamma$ were respectively $DF = \{0.5, 0.6, 0.75, 0.9, 0.95, 0.99\}$ and $\Gamma = \{0.1, 0.01, 0.001, 0.0001\}$. In the training, a link with capacity 100 Gbps and utilization of 27.45% were considered. The delay requisites of the different traffic classes, the prioritization found by the TPCA, and the resulting packet delay is included in Table 4-13. Figure 4-41 shows the results obtained from the grid search of these two hyperparameters. As observed, the hyperparameters configuration of $(df = 0.9, \gamma = 0.001)$ results in the fastest, most stable, and highest TCPA's mean reward.

**Table 4-13. Delay Requirements and Prioritization for ATS Link Used in the Hyperparameters Study. The Capacity of the Link is 100 Gbps and the Utilization is 27.45%**

| TC | Delay Budget | Delay | Prio |
|---|---|---|---|
| #1 | 1.265 µs | 1.094 µs | 4 |
| #2 | 1.2282 µs | 1.094 µs | 4 |
| #3 | 1.261 µs | 0.764 µs | 3 |
| #4 | 0.80671 µs | 0.764 µs | 3 |
| #5 | 0.58371 µs | 0.493 µs | 2 |
| #6 | 0.43462 µs | 0.36 µs | 1 |
| #7 | 1.4688 µs | 1.329 µs | 5 |
| #8 | 2.2174 µs | 1.329 µs | 5 |



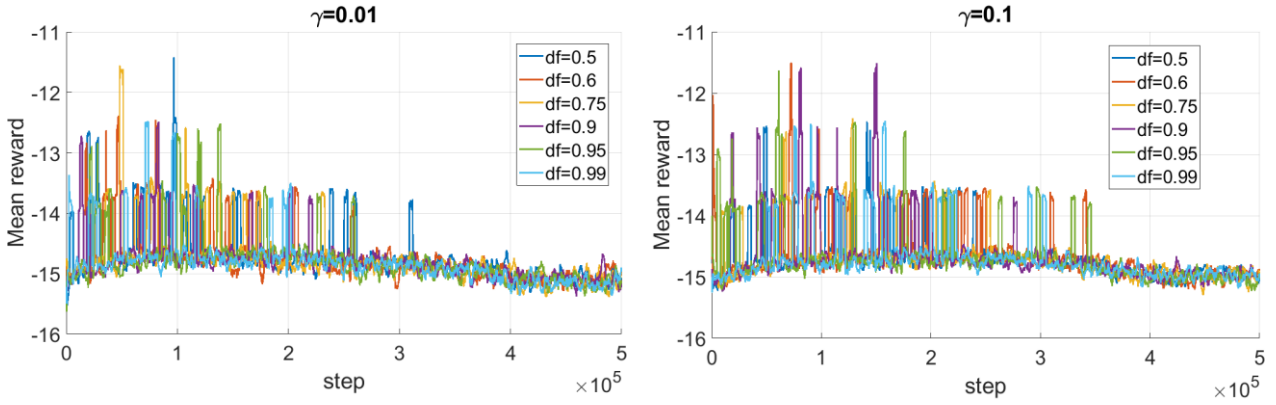



**Figure 4-41. Hyperparameters study results for the 5G-CLARITY RL-based transport network setup solution**

Next, the training process and results regarding the TCPA generalization are presented and discussed. Besides the design improvements for the generalization, a scenario generator was developed to create suitable and scenarios with a diversity of features for the TPCA training and testing. From the TPCA viewpoint, a scenario is a link where the frame transmissions are scheduled according to a strict priority. Every scenario outputted by the scenario generator is solvable. To that end, the generator first designs the scenario, then verifies if the scenario is solvable using a brute force algorithm. If not, the scenario is discarded. In this way, the training of the agent is facilitated. Using the scenario generator, a database of 100 solvable scenarios were generated for the TCPA training. Figure 4-42 depicts a characterization of that database. More precisely, the histograms of the minimum number of required priority levels, utilizations and link capacities are shown. Among the valid set of feasible solutions, that one needing the minimum number priority levels is considered to generate the histogram on the left in Figure 4-42.

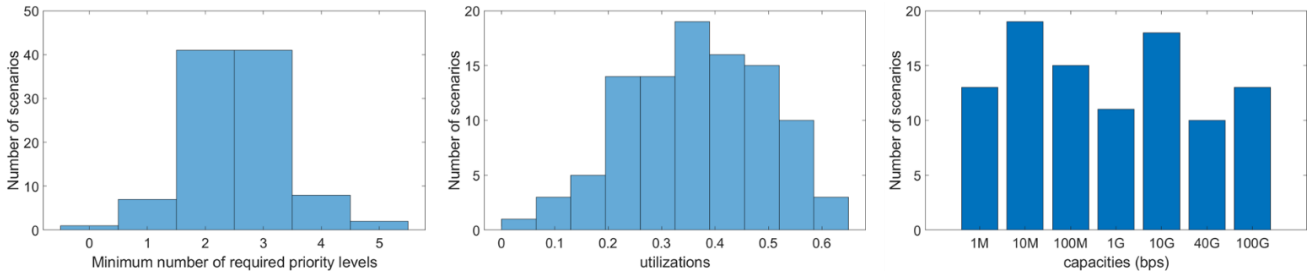



**Figure 4-42. Characterization of the 100 scenarios database employed for the training of the TCPA**

The histogram in the middle of the Figure 4-42 shows the scenarios distribution according to the link utilization, i.e., sum of the aggregated data rate of all traffic classes divided by the link capacity. Last, the distribution of scenarios according to the link capacity is on the left of Figure 4-42.

The TCPA was trained using the 100 scenarios database and the configuration of the main hyperparameters included in Table 4-14. The obtained results in terms of the mean episode length (in number of steps), mean reward, loss function, exploration rate, and frames per second (fps) of the initial training are shown in Figure 4-43. The maximum number of steps per episode was set to 28 because they are enough for the agent to find any possible prioritization given there are eight TCs and eight priority levels. To test the generalization capacity of the TCPA, a new database with 10000 unseen solvable scenarios (none of them was used during the training process) was generated. The TCPA found a valid configuration for 59% of them. After, the TCPA was retrained several times with number of steps between 30 and 50 million, but the success rate was not significantly improved. The unsolved scenarios were characterized, but no correlation between the TCPA failure rate and any of the studied features (e.g., minimum number of required priority levels, utilizations or capacity) was found. The reasons why we did not get a success rate higher than 59% can be the following:

- The values considered for some hyperparameters (e.g., critic network topology) are not suitable. In this regard, the hyperparameters tuning study should be extended and enhanced by using a more efficient search method.
- The training database used does not include a representative number of scenarios and, therefore, a larger set of scenarios including a wider variety of characteristics is required.
- Longer trainings are required.

**Table 4-14. Primary Hyperparameters Configuration for the DRL Agent Used to Configure the ATS-Based Transport Network**

| DQN agent hyperparameters | Configuration |
|---|---|
| Reinforcement learning method | DQN with critic network (value based) |
| Learning rate | 0.001 |
| Maximum number of steps per episode | 28 |
| Mini-batch size | 32 |
| Discount factor | 0.9 |
| Experience buffer length | 10000 |
| Target update frequency | 4 |
| Target update method | Periodic |
| Critic Network | 2 hidden layers with 256 neurons each |
| **ε-greedy exploration** | |
| Epsilon | 1 |
| EpsilonMin | 0.05 |
| EpsilonFraction | 0.5 |

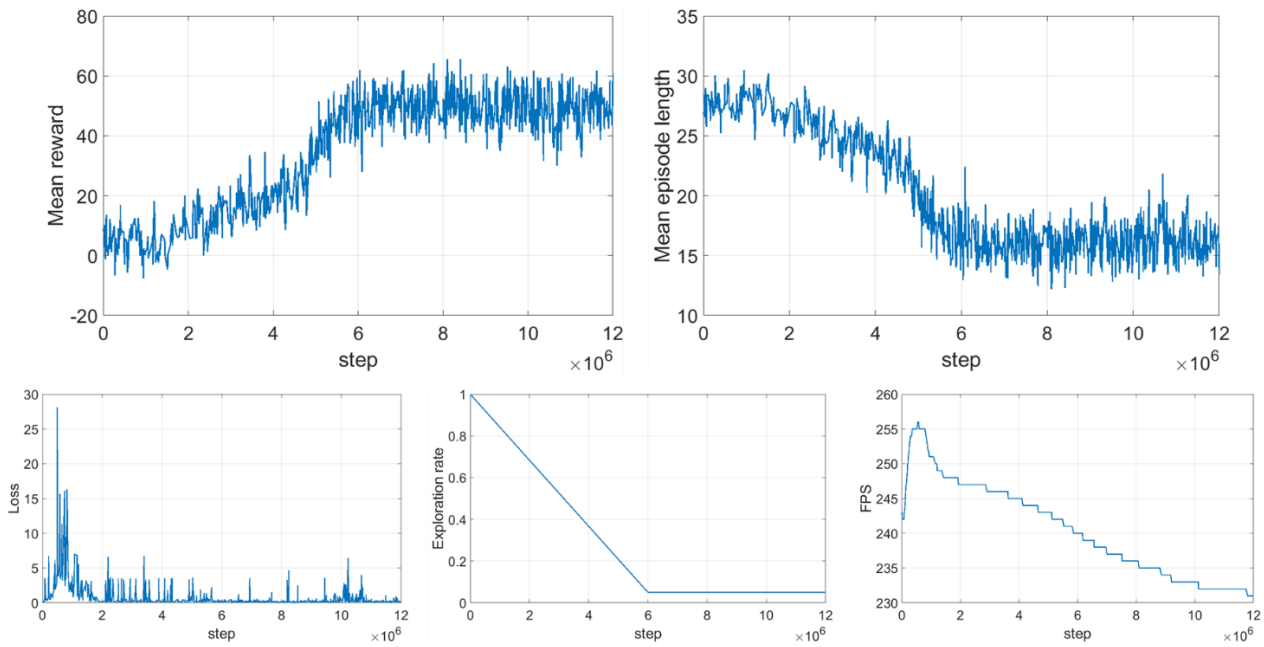



**Figure 4-43. First TCPA training for generalization**

Finally, the TN shown in Figure 4-44 was considered to test the validity of the whole RL-based solution for configuring 5G-CLARITY TNs and illustrate the coordination of its different components. The TN interconnects three server racks, each hosting eight UPF instances, with two gNBs. A dedicated UPF instance is considered

for each 5G-CLARITY slice. Only north-south (downlink) traffic was considered whose target destination is equally distributed between gNB 1 and gNB 2. In our setup, we consider that each 5G-CLARITY slice is dedicated to a typical industrial service (see Table 4-14). The TN delay requirements for these services were set to 10% of their E2E delay requirements (see Table 2-2 in 5G-CLARITY D2.1 [53]). The paths interconnecting the server racks with the gNBs are included in Table 4-16. Please observe they are expressed as a set of links IDs and the link-to-IDs assignments is included in Figure 4-44. The paths were computing using an offline algorithm that tries to balance the utilization of all the links.

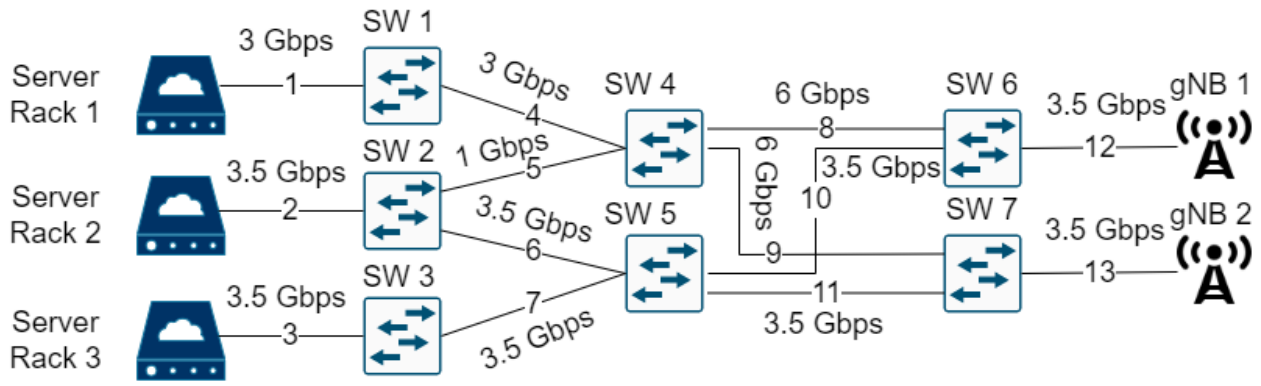



**Figure 4-44. Infrastructure stratum considered for testing the solution**

The setup computed by the RL-based TN configuration solution for the TN depicted in Figure 4-44 is included in Table 4-14 and Table 4-17. Concretely, the 5G-CLARITY slices clustering into TCs, each identified by a PCP, is included in the fifth column of Table 4-14. On the other side, the per TC TN delay distribution among the links performed by the DDA (labelled as "PDB") and the traffic prioritization carried out by the TCPA at every link (labelled as "Prio") are included in Table 4-17. Table 4-17 does not include entries for link 5 as no traffic passes through it given the predefined paths. Table 4-17 also includes the per TC utilization at every link (labelled as "Link Util."), i.e., the aggregated traffic of the respective TC at the link divided by the link capacity, and the per TC worst-case packet delay at every link (labelled as "Delay") given the prioritization issued by the TCPA. Figure 4-45 shows the obtained E2E TN worst-case delay per TC and per predefined paths together with the E2E TN delay budget (labelled as "E2E PDB"). As observed, all the delay requirements are met, thus validating the proper operation of the solution.

**Table 4-15. Features of the 5G-CLARITY Slices Considered in the Setup to Validate the Proper Operation of the RL-Based Transport Network Setup Solution**

| 5G-CLARITY Slice ID | Service | TN delay Budget | Server Rack (Dedicated UPF) | Traffic Class (PCP) |
|---|---|---|---|---|
| #1 | Motion control | 50 µs | #1 | 5 |
| #2 | Motion control | 70 µs | #2 | 5 |
| #3 | Motion control | 100 µs | #3 | 5 |
| #4 | Control-to-control | 500 µs | #1 | 5 |
| #5 | Control-to-control | 700 µs | #2 | 2 |
| #6 | Control-to-control | 1 ms | #3 | 2 |
| #7 | Mobile control panels | 200 µs | #1 | 5 |
| #8 | Mobile control panels | 300 µs | #2 | 5 |
| #9 | Mobile control panels | 400 µs | #3 | 5 |
| #10 | Mobile robots | 50 ms | #1 | 3 |
| #11 | Mobile robots | 25 ms | #2 | 0 |
| #12 | Mobile robots | 5 ms | #3 | 6 |
| #13 | Massive wireless networks | 1 ms | #1 | 2 |
| #14 | Massive wireless networks | 500 µs | #2 | 5 |

| | | | | |
|---|---|---|---|---|
| #15 | Massive wireless networks | 750 µs | #3 | 2 |
| #16 | Closed-loop process control | 900 µs | #1 | 2 |
| #17 | Closed-loop process control | 600 µs | #2 | 2 |
| #18 | Closed-loop process control | 300 µs | #3 | 5 |
| #19 | Process monitoring | 9 ms | #1 | 1 |
| #20 | Process monitoring | 6 ms | #2 | 7 |
| #21 | Process monitoring | 3 ms | #3 | 4 |
| #22 | Plant asset management | 7.5 ms | #1 | 1 |
| #23 | Plant asset management | 2.5 ms | #2 | 4 |
| #24 | Plant asset management | 5.5 ms | #3 | 6 |

**Table 4-16. Predefined paths in the TN shown in Figure 4-44**

| Path ID | Source | Destination | Links |
|---|---|---|---|
| #1 | Server Rack 1 | gNB1 | 1, 4, 8, 12 |
| #2 | Server Rack 1 | gNB2 | 1, 4, 9, 13 |
| #3 | Server Rack 2 | gNB1 | 2, 6, 10, 12 |
| #4 | Server Rack 2 | gNB2 | 2, 6, 11, 13 |
| #5 | Server Rack 3 | gNB1 | 3, 7, 10, 12 |
| #6 | Server Rack 3 | gNB2 | 3, 7, 11, 13 |

**Table 4-17. Per Link and TC Traffic Demands, Delay Budgets, Latency and Prioritization**

| TC | Links 1 and 4 | | | | Links 2 and 6 | | | | Links 3 and 7 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Link Util. | LDB | Delay | Prio | Link Util. | LDB | Delay | Prio | Link Util. | LDB | Delay | Prio |
| #1 | 0 | - | - | - | 0,0179 | 5,55 ms | 21,54 µs | 2 | 0 | - | - | - |
| #2 | 0,0417 | 1,87 ms | 21,07 µs | 2 | 0 | - | - | - | 0 | - | - | - |
| #3 | 0,0417 | 149,99 µs | 21,07 µs | 2 | 0,0357 | 133,35 µs | 21,54 µs | 2 | 0,0357 | 133,32 µs | 17,92 µs | 2 |
| #4 | 0,0209 | 12,49 ms | 21,07 µs | 2 | 0 | - | - | - | 0 | - | - | - |
| #5 | 0 | - | - | - | 0,0179 | 555,62 µs | 21,54 µs | 2 | 0,0179 | 555,50 µs | 17,92 µs | 2 |
| #6 | 0,0625 | 12,49 µs | 12 µs | 1 | 0,0536 | 11,11 µs | 10,28 µs | 1 | 0,0536 | 11,11 µs | 10,29 µs | 1 |
| #7 | 0 | - | - | - | 0 | - | - | - | 0,0357 | 1,11 ms | 17,92 µs | 2 |
| #8 | 0 | - | - | - | 0,0179 | 1,33 ms | 21,54 µs | 2 | 0 | - | - | - |
| TC | Links 8 and 9 | | | | Links 10 and 11 | | | | Links 12 and 13 | | | |
| | Link Util. | LDB | Delay | Prio | Link Util. | LDB | Delay | Prio | Link Util. | LDB | Delay | Prio |
| #1 | 0 | - | - | - | 0,00894 | 5,55 ms | 25,17 µs | 2 | 0,00894 | 8,33 ms | 35,10 µs | 2 |
| #2 | 0,0104 | 937,45 µs | 10,13 µs | 2 | 0 | - | - | - | 0,0179 | 2,49 ms | 35,10 µs | 2 |
| #3 | 0,0104 | 74,99 µs | 10,13 µs | 2 | 0,0357 | 133,32 µs | 25,17 µs | 2 | 0,0536 | 199,97 µs | 13,71 µs | 1 |
| #4 | 0,00521 | 6,24 ms | 10,13 µs | 2 | 0 | - | - | - | 0,00894 | 16,66 ms | 35,10 µs | 2 |
| #5 | 0 | - | - | - | 0,0179 | 555,51 µs | 25,17 µs | 2 | 0,0179 | 833,22 µs | 35,10 µs | 2 |
| #6 | 0,0156 | 6,24 µs | 6,00 µs | 1 | 0,0536 | 11,11 µs | 10,29 µs | 1 | 0,0804 | 16,66 µs | 13,71 µs | 1 |
| #7 | 0 | - | - | - | 0,0179 | 1,11 ms | 25,17 µs | 2 | 0,0179 | 1,66 ms | 35,10 µs | 2 |
| #8 | 0 | - | - | - | 0,00894 | 1,33 ms | 25,17 µs | 2 | 0,00894 | 1,99 ms | 35,10 µs | 2 |

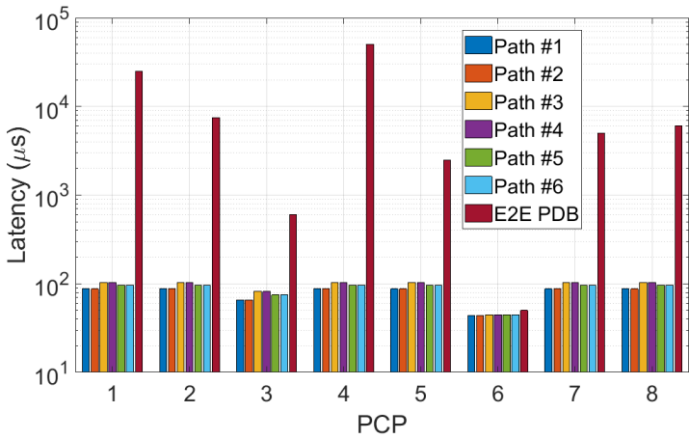



**Figure 4-45. E2E TN packet delay per path and per PCP for the TN depicted in Figure 4-44 given the TN configuration found by the RL-based solution**

## 4.7 Learnings and conclusions from 5G-CLARITY AI algorithms

Due to the ever-increasing complexity of the mobile 5G networks, ML is envisioned as a cornerstone for achieving their full automation through assisting the different decision engines at the management planes. In this vein, 5G-CLARITY system provides an intelligence stratum to host, handle, and configure all the required ML-based algorithms assisting the control and management in private 5G networks. The interface between the ML-based algorithms making up the AI engine and the rest of the 5G-CLARITY system is the Intent Engine. Besides, the Intent Engine enables to specify high-level policies that drives the decision process of the different algorithms to configure the network. In this deliverable and 5G-CLARITY D4.2 [1], several ML-based solutions have been designed, developed and tested in the context of the 5G-CLARITY system. The resulting pool of ML-based algorithms addresses the key decisions to be made in the 5G-CLARITY system involving all its domains, namely, the multi-WAT RAN, the transport network, the computing domain, and the data network. Several learnings and conclusions have been extracted from the development of all these ML-based solutions, below are the primary ones:

Given the heterogeneity in the context and nature of the different decisions to be made in current mobile networks, defining a monolithic agent configuring holistically a whole domain or even a subsystem is a challenging task. Besides, huge amounts of data would be required to train such an ML model. Many of the ML solutions developed in 5G-CLARITY project cope with this problem by defining several different ML models, each designed for a very specific task. By way of illustration, two DL models are needed to work hand-in-hand in order to optimally steer the MPTCP subflows for the algorithm titled eAT3S evaluation (see Subsection 4.1). Sometimes a master algorithm can be required for the coordination of the different ML models conforming the respective solution. For example, the 5G-CLARITY RL-based solution for RRP in multi-WAT RAN (see Subsection 4.5) relies on a master algorithm that coordinates two RL agents, each for the radio resource allocation to a given type of service, and integrates them with a Wi-Fi offloading algorithm. Further, many of the 5G-CLARITY solutions require to be coordinated through a master algorithm to ensure the cohesion and satisfiability of the configurations applied to the distinct domains. For instance, the E2E delay budgets imposed by the services need to be distributed among the different network domains (e.g., the solution of Subsection 4.6).

ML is regarded as an alternative approach to deal with the high computational complexity exhibited by exact optimization methods or even to solve intractable problems. Nonetheless, large amount of high-quality data can be required for the effective training processes of the ML models and producing ready-to-use ML-based solutions. Simulation-based data generation can be cumbersome, time-consuming, and computational

resources demanding. In this regard, the use of analytical performance models can ameliorate this problem. For instance, the solution proposed in Subsection 4.5 makes use of an analytical model to shape the behaviour of URLLCs.

Many ML-based solutions to provide network intelligence target long-term decisions at high-times scales ranging from a few seconds to several days. For these cases, collecting a significant amount of data directly from the network for an effective training might need unacceptable periods of time. Moreover, the learning efficiency of some ML techniques, such as Reinforcement Learning, accentuates this problem. For instance, the agent for prioritizing IEEE 802.1Q traffic classes in the RL-based 5G-CLARITY TN configuration (see Subsection 4.6) required more than four million of training episodes (agent's tries to configure the network). However, as these configurations in a real environment are infrequent, several years might be required to get a proper performance for the agent. In this context, offline training through the use of simulation and analytical performance models becomes crucial to produce ML-based models with a fair performance as to make appropriate decisions at the early stages of their deployments in the real environment. This offline training approach is used for example in the ML model of section 4.2 through a simulated training environment.

The hyperparameters of the ML models (i.e., those parameters that control the training process) highly impact on the training effectiveness and resulting performance of the ML models. What is more, a wrong configuration of the hyperparameters might hinder the convergence of the ML model, though it is well-designed. In this regard, the tuning of the hyperparameters using any approach (e.g., grid-search, random-search, Bayesian optimization) is crucial before the ML model training process.

# 5 Experimental Evaluation of Intelligence Stratum, Data Lake, and Indoor non-LoS Identification

In this section we provide an integrated demonstration of the 5G-CLARITY intelligence stratum. To this end we have selected one of the ML algorithms that was developed in D4.2 [1], namely the NLoS identification algorithm, and we have integrated this algorithm with the Data Lake, the AI Engine and the Intent Engine.

Section 5.1 provides an overview of the implementation and integrations that were required to validate the intelligence stratum, whereas the rest of the section provides a detailed description of all the required steps.

## 5.1 Overview of required implementation and integrations

The implementation and integration carried out here is mainly based on four components developed and employed in the context of 5G-CLARITY project. In particular, NLoS identification function, AI Engine, Intent Engine, Data Lake are the elements brought together to conduct this demonstration. The details of each component is described in Table 5-1. In what follows, we describe the procedure by which we integrate all above-mentioned components.

**Table 5-1. Overview of modules involved in the demonstration of the 5G-CLARITY intelligence stratum**

| Module | Background | Extensions in 5G-CLARITY | Responsible partner | Module integrations validated in this section |
|---|---|---|---|---|
| NLoS identification ML function | N/A | Developed from scratch | IHP | DNN trained offline on CIRs. |
| AI Engine | The AI Engine was built upon the Open-source Function-as-a-Service (FaaS) platform. OpenFaaS is a flexible and lightweight toolkit that advertises to be able to run anywhere, with any code and at any scale. A custom language configuration was created for AI Engine models which exposed additional monitoring capabilities to the model authors. | The OpenFaaS toolkit was extended with additional monitoring capabilities for model authors allowing for external monitoring through tools like Grafana. An interface was also designed between the AI Engine and Intent Engine for the intent driven execution of models. | LMI | Direct querying of the Data Lake through the AI Engine / Data Lake interface. |
| Intent Engine | The Intent Engine was built on Adaptive Policy EXecution (APEX). APEX is a fully featured policy engine that executes anything from simple to adaptive policies that can modify its behaviour based on the current conditions of the network and systems. The internal execution of policies behave similar to state machine | A collection of intent policies were designed, coordinated and executed within the APEX policy engine. A dynamic interface was also provided allowing the Intent Engine to communicate with service providers through a common execution. | LMI | Identification and triggering of appropriate ML models within the AI Engine in response to received intent request. |

| | | | | |
|---|---|---|---|---|
| | allowing for high levels of flexibility and adaptability during the decision making process. | | | |
| Data Lake | The Data Lake is a cloud-based approach where the cloud computing platform AWS is provided by Amazon. It comprises a data storage service along with various other services. | An interface is developed to enable mobile devices to provide their channel impulse response telemetry to the Data Lake. This telemetry data is stored in a specific database which is then fetched by an ML algorithm residing in the AI engine to predict whether the channel has a LoS or NLoS link. | IDCC | API to enable AI engine to fetch telemetry data from the Data Lake.<br>Note: Data storage and schema details of the CIR telemetry data object is validated in Section 3. |
| CIR Telemetry | N/A | Developed from scratch | IHP | Integrated with the Data Lake component. Details provided in Section 3. |

The integration of the previous software was demonstrated at EuCNC 2022. A video demonstrating the integration of the previous software modules to is available in [57].

## 5.2 NLoS Identification

The goal of this section is the evaluation of the intelligence stratum by integration of the NLoS identification algorithm, data lake and intent engine, all described in D4.2 [1]. To this end, this section shows how the integrated NLoS identification algorithm into the AI Engine fetches required telemetry data from Data Lake. The details of CIR telemetry data within the data lake are provided in Section 3.1.4. Requests from other parts of the network, e.g., localization server, can be submitted to the NLoS identifier algorithm through the intent engine (steps (1) and (2) in Figure 5- 1). To respond, the algorithm retrieves the CIR corresponding to the request from the data lake, steps (3) and (4), feeds it to its core kernel, i.e., the pre-trained DNN, and returns its final decision on the channel condition, i.e., LoS or NLoS (steps (5) and (6)). The CIRs are sent to the data lake from the Access Points (APs).

To integrate the NLoS identifier as a function into the AI engine, we draw on the OpenFaas, the details of which is not in the scope of this section. Nevertheless, the commands needed for the purpose of this section are thoroughly described. Furthermore, the output of each step is shown in form of a figure for further clarification. In the sequel, we describe the integration steps in detail.

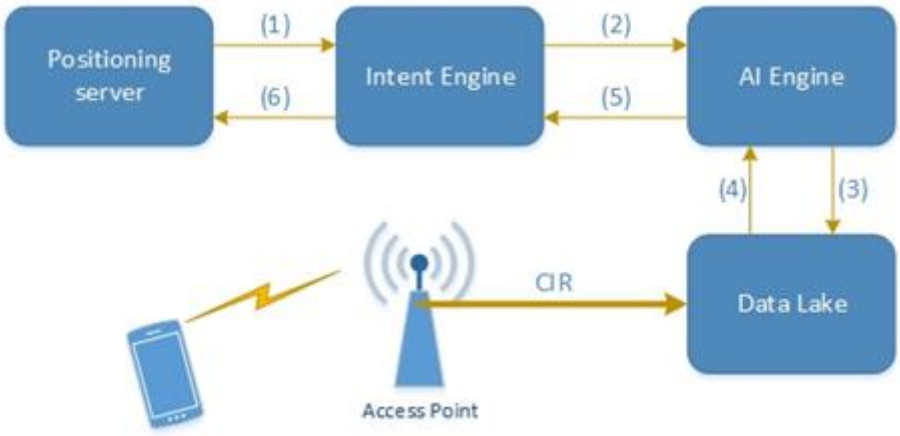



**Figure 55-1. An exemplifying scenario where localization server requests a decision on the link condition, e.g., NLoS or LoS**

## 5.3 NLoS identification as an AI engine function

In this section, we delve into the details of the function creation, function integration into the AI Engine, i.e., build, push and deploy steps, and CIR retrieving from the Datalake.

### 5.3.1 Function Creation

The first step towards integration of the algorithm into the AI engine is creating an OpenFaas template. In general, OpenFaas can create function templates for different programming languages. In this deliverable, we utilize the Python template suitable for AI engine and developed by LMI. In particular, the command

```
faas-cli new --lang python3-aiengine nlos --prefix=username
```

results in Figure 5-2, which is an indication that a folder with the name “nlos” has been created.


**Figure 5-2. Creating the nLoS function template using OpenFaas**

This folder contains four different files, namely “__init__”, “handler”, “metric_reporter”, and “requirements”. For the purpose of this section, the files “__init__” and “metric_reporter” are not modified and remain as they are. The former is an automatically generated file as part of the Python 3 AI Engine template and it is used to identify directories as python packages so that they can be easily imported. The latter allows for pushing metrics recorded inside the model to the AI Engine Prometheus client and it is a way for the designer of the ML model to expose monitoring information. For the NLoS identification algorithm to function within the AI engine entity, the Python code corresponding to the algorithm needs to be integrated into the “handler” Python file. In particular, the function “handler” is responsible for executing

the Python code, interacting with other modules within and outside of the AI engine, and finally returning the outcome of the execution. We manually copy the code corresponding to the algorithm into the "handler". Furthermore, the Python packages required for the algorithm are listed in the "requirements" file. This is particularly necessary for handler to function as intended. Another subtle point to take into account here is that all the necessary files used in the handler need to be put into the function folder. Moreover, all the addresses in the handler referring to those files must be in the form of "function/<filename>".

### 5.3.2 Build, push, and deploy

The next step, after creating the function template and integrating the Python code corresponding to the NLoS identifier, is to build the function image. The command

```
faas-cli build -f ./nlos.yml
```

builds the function image into the local docker library. The output of the command has been shown in Figure 5- 3. Note that, the log has been shortened to depict only the first and last steps after the execution of the command.

Once the image is built, we can push it into the remote container registry, which is the repository to store the container images. The corresponding command is

```
faas-cli push -f ./nlos.yml
```

The final step will be then to deploy the function. This can be done by running

```
faas-cli deploy -f ./nlos.yml
```

which deploys the function into a cluster of images. The output of the above-mentioned commands can be found in Figure 5-4.

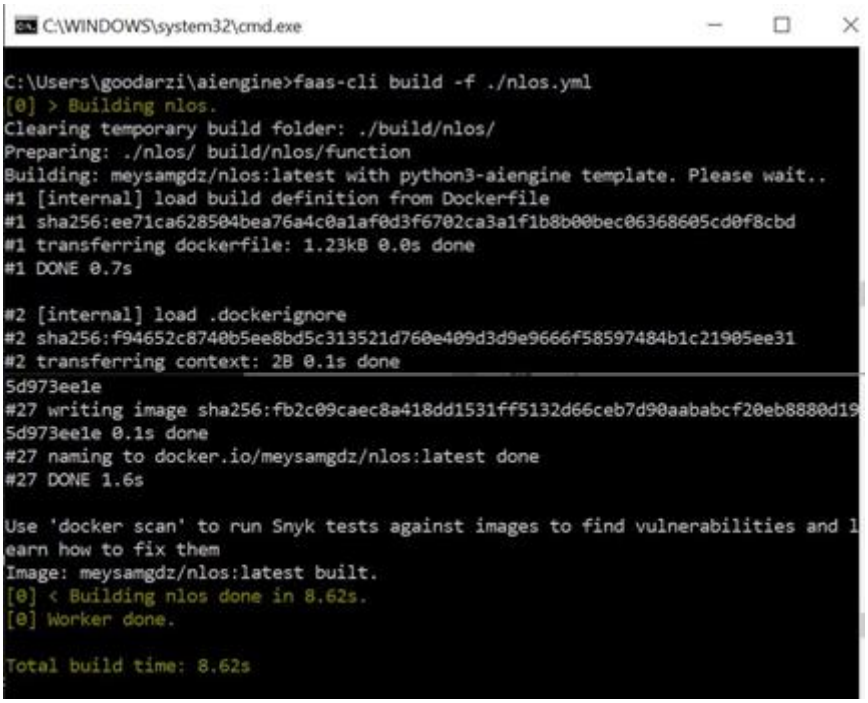



**Figure 5-3. Building the nLoS function image using OpenFaas**

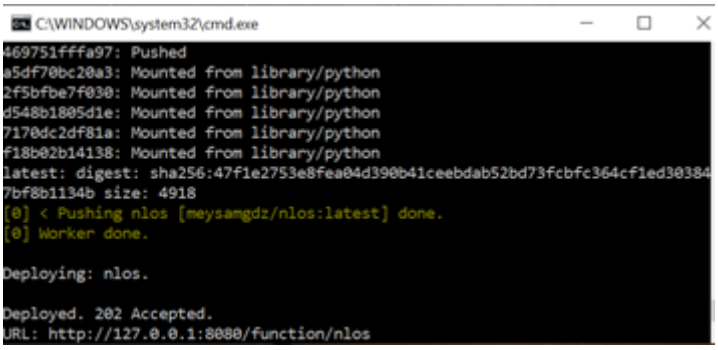



**Figure 5-4. Output of the OpenFaas deploy command**

Another simple way to build, push and deploy a function image is to use the command

```
faas-cli up -f ./nlos.yml
```

which automatically executes all the three commands explained above. Having the function ready, we can execute it to evaluate the outcome. In the sequel, we evaluate the outcome of the build function image.

### 5.3.3 Retrieving the CIRs

In order to return a decision on the communication link condition, the NLoS identifier function needs to retrieve the CIRs stored in the data lake. The data lake is created using Amazon S3 Bucket and allows for storing and retrieving the data by means of an API with a URL and authentication key. As described in Section 3.1, the latest measured CIR is uploaded to the data lake using "*requests.put(url, json_file, headers)*" where *url* is the directory where the data is to be stored, *json_file* is the measured CIR in form a JSON file, and *headers* is defined as

*headers = {"Accept": "application/json", "x-api-key": Authentication Key}.*

Upon receiving a decision request (step 2), the NLoS identifier function retrieves the latest uploaded CIR using the "*requests.get(url, headers)*" (step 3 and 4) and pass it into the pre-trained neural network.

## 5.4 Experimental evaluation

In this section, we firstly present the outcome of function integration into the AI Engine. Next, we provide and analysis on the outcome of the interaction between the AI Engine and the Intent Engine. Lastly, the experimental results are visualized explained.

### 5.4.1 Integration in AI Engine

To evaluate the performance of the algorithm, we use the OpenFaas GUI placed on the local host with the following IP address: http://127.0.0.1:8080/ui/, where the function images are stored and can be executed upon request. Figure 5-5 shows the execution of the NLoS identifier function using the 'Invoke' button. As soon a request is initiated from the intent engine through "invoking", the data is retrieved from the data lake, fed into the pre-trained DNN model in the AI engine, and a decision on the link condition as well as the belief in the decision is returned to the Intent Engine.

### 5.4.2 Interaction with Intent Engine

The NLoS identifier function integrated in the AI engine can be invoked upon requests from other network entities, e.g., localization server. Such requests are submitted to the AI engine through the intent engine. Figure 5-6 exemplifies the manner in which a request can be submitted to NLoS identifier function as well as the response received by the intent engine. In particular, the intent engine submits a request with the content "*check the line of sight*". The "matching" module integrated into the AI engine finds the best

matching function, the NLoS identifier in this case, and invokes the function. The output is returned to the intent engine as a JSON file and contains three elements, i.e., the "response", which can be "los" or "nlos", the belief in the response denoted by "probability", and the "decision_flag", which indicates whether the function has identified the link condition correctly or not. The latter is only added for the purpose of testing. We note that the intent request depicted in the Figure 5-6 can be extended to incorporate parameters as well, e.g., one can pass the mobile user index in order to receive the NLoS identification for that specific user.

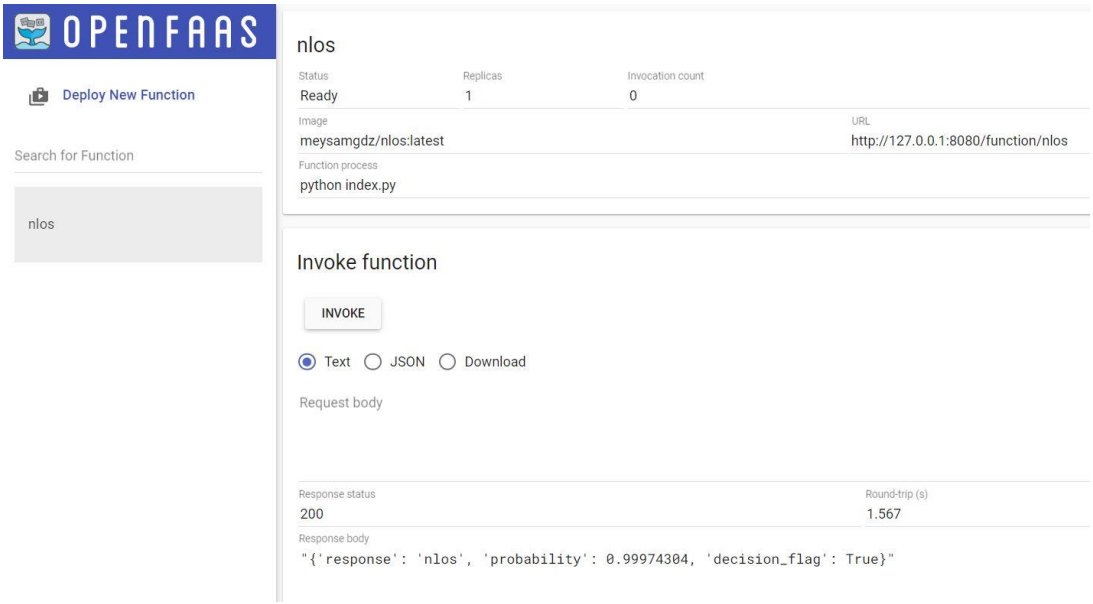



**Figure 5-5. NLoS function image created and executed using OpenFaas**

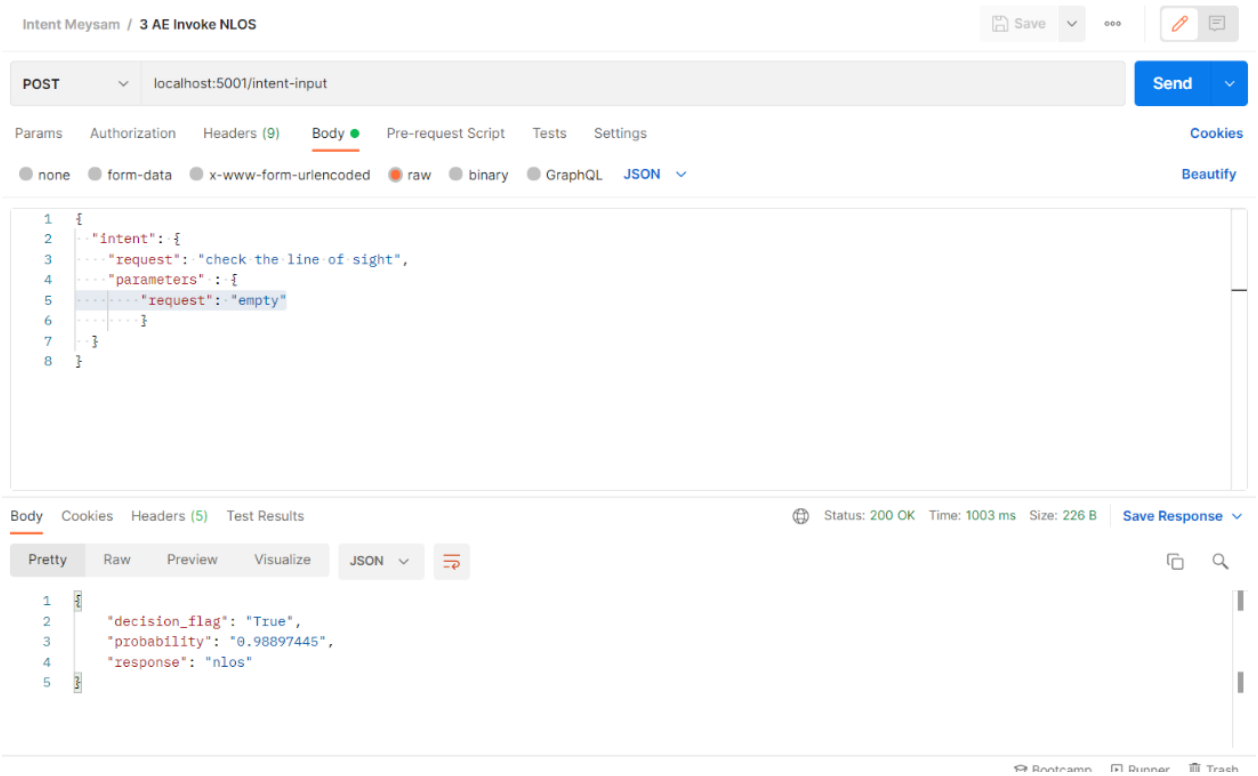



**Figure 5-6. Intent engine submitting a NLoS identification request to the AI Engine and receiving back the response**

The following figures indicate a real-time experiment conducted in an office environment to verify the functionality of intent-based NLoS identification algorithm. An SDR is periodically sending a *maximum length*

*sequence* (mls), which is received on the other side by another SDR. The corresponding CIR is then extracted from the autocorrelation of the mls and is directly uploaded to the datalake. The uploaded CIR is then retrieved by the AI Engine upon requests from the intent engine and fed into the algorithm. The resulting decision on the communication link condition is returned to the intent engine (shown on the screen of the laptop in the figures).

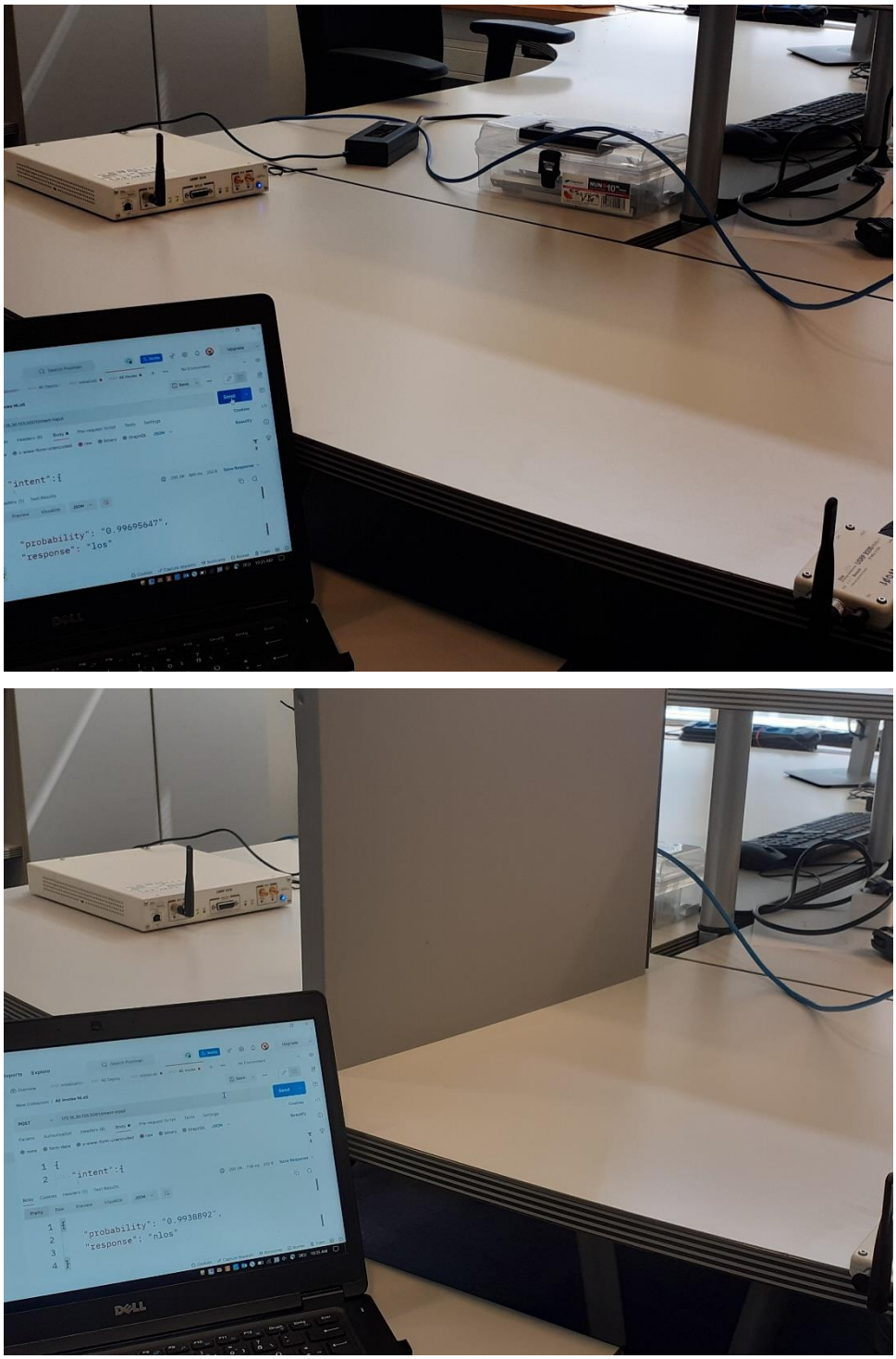



**Figure 5-7. Real-time intent-based NLoS identification**

# 6 Private-Public Network Integration

Private-public network integration is one of the main distinguished features of the 5G-CLARITY system. This feature represents the ability to make 5G-CLARITY capabilities interwork with MNO's managed capabilities seamlessly, (i) allowing for consistent operation of PNI-NPN, from an E2E service viewpoint, and (ii) ensuring trustworthiness between private and public administrative domains. As captured in D4.2 [1], public-private network integration builds upon three main enablers.

Mediation Function, a MF in the M&O stratum which allows 5G-CLARITY provider to expose capabilities to MNOs in a secure, controllable and auditable way. **D4.2** reported on the first solution design for the mediation function. In **D4.3**, we will elaborate on the final solution design, and illustrates the usage and applicability of this mediation function using a use case-based approach (see Section 6.1)

Service delivery models, which specify how to cluster capabilities in such a way they can be delivered to the MNOs in a consistent way. **D4.2** reported on the main merits and limitations on 5G-CLARITY service delivery models (originally defined in Deliverable D2.2 [2]), with focus on NFVI as a service (NFVIaaS) and Slice as a Service (SlaaS). In **D4.3**, we will report on their usage on relevant application scenarios (see Section 6.2).

Location of the AI engine. In **D4.2,** we discussed the pros and cons of moving AI engine between public and private administrative domains in terms of data management (regulation, data pipelines) and performance. Further progress on this enabler will be done in context of in-project pilots, and thus it will be reported in Deliverable **D5.2** [20]**.**

## 6.1 Mediation function

The solution design of 5G-CLARITY Mediation Function is depicted in Figure 6-1. As seen, it is composed of the following modules:

**API Gateway (mandatory)**, which is the front-end service for 5G-CLARITY management and orchestration stratum, enforcing policies and access control between MFs and external consumers. As the entrance of the mediation function, all requests shall go through the API gateway to the specific MF service.

**API Portal (mandatory),** which has an informative role for external consumers. The portal describes what APIs are available for usage, listing them all and providing a description for their consumption: API endpoint (e.g., IP address, Fully Qualified Domain Name [FQDN]), API lifecycle information, eligibility to be the consumer of the API, API health insights (e.g., real-time monitoring), etc. The documentation on the portal should also provide the authentication and authorization mechanism, use cases that describe the business context and live real implementations.

**API orchestration (optional)**, which is a MF service responsible for consuming service bus exposed APIs (i.e., APIs offered by the different 5G-CLARITY management and orchestrated MFs) and applying transformation operations on them, when needed. Once transformed, these APIs can be securely exposed through the API gateway.

**Supporting services,** which are MF services that support the operation of API gateways and API portal. On the one hand, there is a database, in charge of keeping a registry with available and published APIs together with their endpoints. On the other hand, there is the messaging system, which allows the exchange of internal messages on the 5G-CLARITY mediation function.

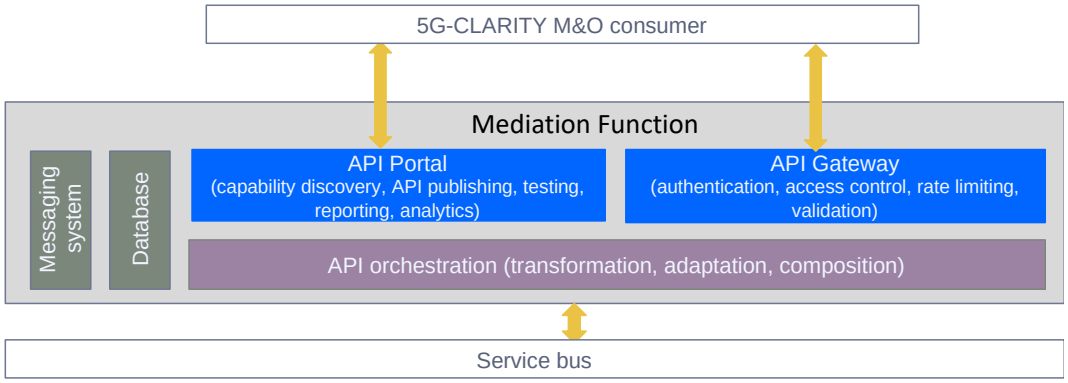



**Figure 6-1. 5G-CLARITY mediation function solution design**

In 5G-CLARITY D4.2 [1], the discussion was focused on multi-tenancy support, with the ability of the mediation function to define customized yet separate management spaces for different tenants. Each space allows the 5G-CLARITY operator to provide a controllable and auditable exposure of capabilities to 5G-CLARITY tenants, based on their specific needs. On the one hand, *controllable* means that the provider can regulate the particular set of resources each tenant is allowed to access and under which conditions, leveraging Role Based Access Control (RBAC). RBAC is defined around predefined roles. Roles are a collection of permissions that you can bind to a resource; this binding allows the privileges associated with that role (e.g., read-only, write and read, etc.) to be performed on those resources, using access tokens. 5G-CLARITY mediation function grants different roles to different tenants, according to their specific needs. On the other hand, *auditable* means that every interaction between 5G-CLARITY system and the tenant need to be logged with accurate timestamps (for traceability) and support non-repudiation (for SLA verification).

In this new deliverable, we focus on plausible solutions for this mediation function. One important aspect to bear in mind is that 5G-CLARITY system aspires to become a reference solution for 5G (and beyond) private networks, with a number of 3rd parties wanting to consume offered capabilities, including MNOs, hyperscalers and application developers, among others. To ensure wide market adoption and an attractive economy of scale for all these tenants, it is essential for 5G-CLARITY Mediation Function to offer APIs adhered to these three main principles:

*Open*. 5G-CLARITY shall avoid offering proprietary APIs; it needs to leverage as much as possible on standard-based or de-facto APIs, following industry recommendations.

*Global*. 5G-CLARITY offered APIs shall allow every tenant to have a uniform and consistent service experience across a global footprint, with the effortless portability of applications across different private network platforms (design once run everywhere approach) and easy service replicability. The lower the integration efforts, the more likely to have 3rd parties onboard.

*User-friendly*. 5G-CLARITY offered APIs need to be abstracted out of internal APIs, to hide 5G-CLARITY internal complexity and make them easy to use (consume) to 3rd parties, especially those with no telco expertise/background experience.

Based on the above rationale, we provide plausible solutions for the Mediation Function components, in particular for two of the core components: **API orchestration** and **API gateway**.

### 6.1.1 API orchestration

API orchestration allows transforming 5G-CLARITY internal APIs into open, global and user-friendly APIs. Though optional, this transformation is advisable, as it helps facilitating adoption by 5G-CLARITY tenants (the more, the better), consolidating 5G-CLARITY system as a reference system solution. The API orchestration may be deployed as a microservice with a twofold purpose: i) keeping the information on correspondences

between tenant-facing APIs (external APIs) and 5G-CLARITY-facing APIs (internal APIs), being this info captured in a mapping table; ii) coordinating the workflow execution to enforce these correspondences.

In the following, we provide an example on three tenant-facing APIs that can be offered through the 5G-CLARITY mediation function: resource capabilities discovery, edge application lifecycle management and slice provisioning.

The resource capabilities discovery APIs allow the 5G-CLARITY tenant to browse the different flavors available for use.

In Table 6-1, the resource URI has not been included, since these APIs are just mere examples, for illustration purposes (specification of information model is out of scope of 5G-CLARITY), and in Table 6-2 the resource URI has not been included, since these APIs are just mere examples, for illustration purposes (specification of information model is out of scope of 5G-CLARITY).

The edge application onboarding management APIs allow the 5G-CLARITY tenant to onboard the application server into a 5G-CLARITY edge node, and manage its lifecycle afterwards. It represents an on-prem IaaS offering.

**Table 6-1. Resource Capabilities Discovery**

| Operation | HTTP Method | Qualifier | Input params | Output params |
|---|---|---|---|---|
| Query compute resource capabilities | GET | M | Tenant ID | supportedComputeCapabilities[1]<br>availableComputeFlavors[2]<br>status[3] |
| Query wireless resource capabilities | GET | M | Tenant ID | supportedWirelessCapabilities[4]<br>status[3] |
| Query transport resource capabilities | GET | M | Tenant ID | supportedTransportCapabilities[5]<br>status[3] |

NOTE1. supportedComputeCapabilities parameter informs about the capabilities available in the compute infra, including supported CPU architecture types (x86-64 Intel and/or ARM), supported operating systems (RHEL Linux, Ubuntu, Windows, macOS, etc.) and supported acceleration capabilities (GPU, FPGA, etc.).

NOTE2. availableComputeFlavors parameter allows the tenant to discover the set of flavors available for selection. This parameter is an array of flavors, each profiled with the following attributes: flavorId, cpuArchType, OS, numCPU (number of virtual CPUs in the selected OS), phyMemory (RAM size), rootDiskSize (amount of disk space to use for the root [/] partition), egressBandwidth (max bandwidth attainable; if not specified, best-effort traffic policy will be applied) and computeAccel.

NOTE3. status can be associated to different response codes: 200 (successful), 401 (authorization information is missing or invalid), 404 (content not found), 500 (internal server error) or 503 (server unavailable)

NOTE4. supportedWirelessCapabilities parameter informs about the technologies available in the access infrastructure. Examples: 5GNR (with information on 5G Release), Wi-Fi (with reference to IEEE standards version) and LiFi.

NOTE5. supportedTransportCapabilities parameter informs about the technologies available in the transport infrastructure. Examples: IPSec available or not, TSN available or not.

**Table 6-2. Edge Application Lifecycle Management**

| Operation | HTTP Method | Qualifier | Input params | Output params |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| Onboard application | POST | M | Tenant ID<br>appSpec[1]<br>appArtifacts[2] | appInstanceId<br>appInstanceInfo[3]<br>status[4] |
| Update application | PUT | O | Tenant ID<br>appInstanceId<br>appInstanceInfo<br>appComponentSpecs | appInstanceId<br>appInstanceInfo<br>status[4] |
| Remove application | DELETE | M | Tenant ID<br>appInstanceId | status[4] |
| Query application | GET | M | Tenant ID<br>appInstanceId<br>appAttributelistIn[5] | appAttributeListOut[6]<br>status[4] |

NOTE1. appSpec parameter includes the following attributes: appProviderId, appName, appMetaData (application version, if the application supports mobility or not, if the application supports single user or multiple user clients, etc.), and appQoSProfile (latency constraints, guaranteed data transfer bandwidth, etc.).

NOTE2. appArtifacts parameter specifies the docker containers image files(s) and associated application component descriptors, including VNFDs/NSDs and config files/Helm charts/Terraform scripts.

NOTE3. appInstanceInfo parameter includes key-value pairs for the following parameters: appInstanceState (pending, running, failed, ..), endPointsInfo (details of IP address/FQDN, port, socket, etc.), appSpec and appArtifacts.

NOTE4. status can be associated to different response codes: 200 (successful), 401 (authorization information is missing or invalid), 404 (content not found), 500 (internal server error) or 503 (server unavailable)

NOTE5. appAttributeListIn identifies the appInstanceInfo attributes to be returned by this operation. If this parameter is absent or empty, all attributes will be returned.

NOTE6. appAttributeListOut returns the key-value pairs for every attribute requested in appAttributeListIn.

Finally, the slice provisioning APIs allow the 5G-CLARITY tenant to request the provisioning of an infrastructure slice. This slice provides a B5G connectivity pipe to communicate one or more UEs (handheld terminals, CPEs) with a service (application server). As captured in 5G-CLARITY D2.2 [2] and detailed in both 5G-CLARITY D4.1 [46] and 5G-CLARITY D4.2 [1], a 5G-CLARITY slice includes one wireless resource quota, one transport resource quota, and one compute resource quota. The compute resource quota will be used to host virtualized workloads, including virtualized RAN, 5GC SA and AT3S enabled UPF. Nevertheless, all these details are transparent to the tenant, which is only focused on the connectivity endpoints, ruling out the network level components in between.

**Table 6-3. Slice Provisioning: the resource URI has not been included, since these APIs are just mere examples, for illustration purposes (specification of information model is out of scope of 5G-CLARITY)**

| Operation | HTTP Method | Qualifier | Input params | Output params |
|---|---|---|---|---|
| Create slice | POST | M | Tenant ID<br>computeFlavorId[1]<br>selectedAccessCapabilities[2]<br>selectedTransportCapabilities[3]<br>NEST[4]<br>ueIpAddrList[5]<br>appInstanceList[6] | sliceId<br>sliceInstanceInfo[7]<br>notificationUrl[8]<br>status |
| Update slice | PUT | O | Tenant ID | sliceId |

| | | | | |
|---|---|---|---|---|
| | | | sliceId<br>sliceInstanceInfo | sliceInstanceInfo<br>notificationUrl<br>status |
| Remove slice | DELETE | M | Tenant ID<br>sliceId | status |
| Query slice | GET | M | Tenant ID<br>sliceId<br>sliceAttributelistIn[9] | sliceAttributeListOut[10]<br>notification<br>status |

NOTE1. computeFlavorId specifies the flavor that build out the compute resource quota in the 5G-CLARITY slice. The available flavors can be retrieved using the "query compute resource capabilities" APIs from Table 6-1.

NOTE2. selectedAccessTechnologies parameter is an array that specifies the technologies that will be used to build out the wireless resource quota in the 5G-CLARITY slice. The available access technologies can be retrieved using the "query wireless resource capabilities" APIs from Table 6-1.

NOTE3. selectedAccessTechnologies parameter is an array that specifies the technologies that will be used to build out the wireless resource quota in the 5G-CLARITY slice. The available access technologies can be retrieved using the "query transport resource capabilities" APIs from Table 6-1.

NOTE4. NEST is a GSMA defined construction that captures the service requirements that a particular tenant wants for a slice.

NOTE5. ueIpAddrList parameter specifies the IPv4 address of individual UEs that will become slice subscribers.

NOTE6. appInstanceList identifies the (list of) service(s) associated to the slice. One slice can be associated to one or more services. appInstanceList is an array of appInstanceId, each identifying one application server that will serve slice subscribed UEs. Note that the appInstanceId is the ID that the "onboard application" API returns in Table 6-2. Note that service-to-slice association can occur at provisioning time (upon slice creation, see "create slice operation") or operation time (see "update slice"). In the latter, existing/new services can be removed/added from the slice.

NOTE7. sliceInstanceInfo parameter includes key-value pairs for the following parameters: sliceInstanceState (pending, running, failed, etc.), sliceEndPointsInfo (pointers to subscribed UEs and attached application server), appInstanceList, allocated NEST values, allocatedPlmnIds and allocatedSSID.

NOTE8. This parameter specifies the URL where the 5G-CLARITY tenant should connect to get information on slice status.

NOTE9. sliceAttributeListIn identifies the sliceInstanceInfo attributes to be returned by this operation. If this parameter is absent or empty, all attributes will be returned.

NOTE10. appAttributeListOut returns the key-value pairs for every attribute requested in appAttributeListIn. These attributes will be notified to the 5G-CLARITY tenant through notificationUrl.

### 6.1.2 API gateway

To make tenant-facing APIs available for 3rd party consumers, there is a need to implement a set of support/common capabilities such as *onboarding (registry), authentication and authorization, discovery, auditing, accounting,* to name a few. These capabilities, to be provided by the API Gateway, allow policing the interaction between the API provider and consumer, when the two entities belong to non-trusted domains. This is what happens precisely in 5G-CLARITY, where the 5G-CLARITY operator (acting as API provider) and the 5G-CLARITY tenant (acting as API consumer) are defined in different administrative domains. For this API Gateway, it is proposed to use of 3GPP Common API Framework (CAPIF) [58] , which provides the abovementioned capabilities. One of the main advantages of CAPIF solution is that though specified by 3GPP, it is not tied to 3GPP APIs; indeed, CAPIF can be used as gateway solution for any API, no matter the API semantics. This means that 3GPP/ETSI/TMF and proprietary APIs can be registered into CAPIF.

This property, together with the fact that CAPIF is a normative solution with wide acceptance at industry, makes CAPIF an ideal implementation solution for the API Gateway in 5G-CLARITY Mediation Function.

The CAPIF architectural framework is illustrated in Figure 6-2. A summary of the functional entities building up the framework is captured in Table 6-4.

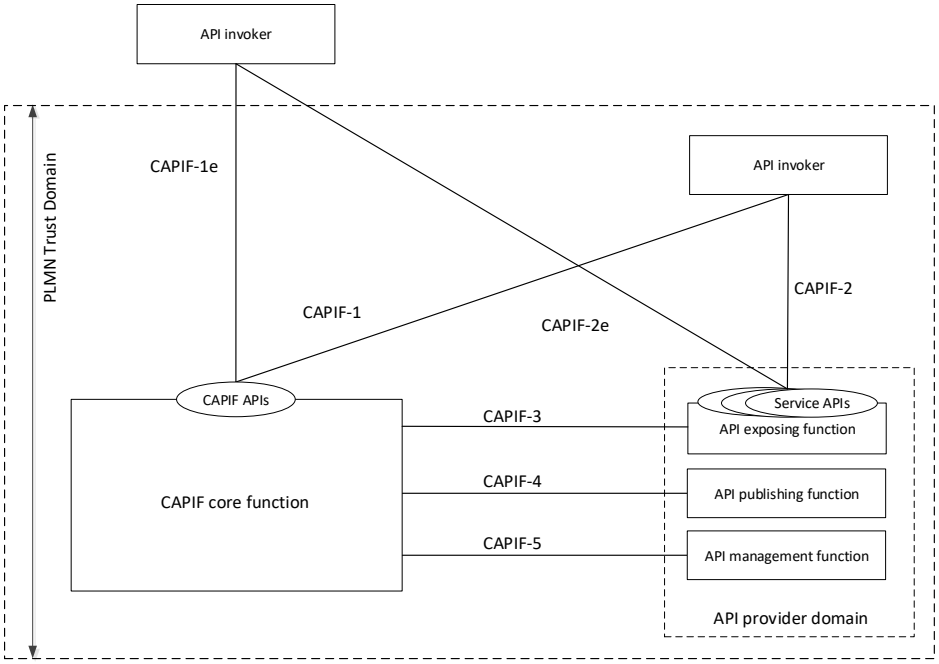



**Figure 6-2. CAPIF architectural framework**

**Table 6-4. CAPIF components and interfaces**

| CAPIF components | |
|---|---|
| **CAPIF Core Function (CCF)** | It acts as an orchestrator that manages the interaction between providers and consumers. The main responsibilities of CCF are authentication of the API invoker, authorization of the API invoker to access the available service APIs, monitoring the service API invocations. |
| **API invoker** | It represents the vertical app which consumes the service APIs utilizing CAPIF. API Invoker provides to the CCF the required information for authentication, discovers and then invokes the available service APIs. |
| **API Exposing Function (AEF)** | It is responsible for the exposure of the service APIs. Assuming that API Invokers are authorized by the CCF, AEF validates the authorization and subsequently provides the direct communication entry points to the service APIs. AEF may also authorize API invokers and record the invocations in log files. |
| **API Publishing Function (APF)** | It is responsible for the publication of the service APIs to CCF in order to enable the discovery capability to the API Invokers. |
| **API Management Function (AMF)** | It supplies the API provider domain with administrative capabilities. Some of these capabilities include, auditing the service API invocation logs received from the CCF, on-boarding/off-boarding new API invokers and monitoring the status of the service APIs. |
| **CAPIF Interfaces** | |
| **CAPIF-1/1e** | API Invoker and CCF interact over CAPIF-1/1e interfaces supporting authentication and authorization of API Invokers, discovery of service APIs, and onboarding / off boarding |

| | |
|---|---|
| | of the API invokers. CAPIF-1 and CAPIF-1e interfaces are used when API invoker is within and outside of PLMN trust domain, respectively. |
| **CAPIF-2/2e** | API Invoker and the AEF interact over CAPIF-2/2e interfaces supporting authentication and authorization of API Invoker and service API invocations by the API Invoker. CAPIF-2 and CAPIF-2e interfaces are used when API invoker is within and outside of PLMN trust domain respectively. |
| **CAPIF-3** | AEF interacts over the CAPIF-3 interface for enforcing access and policy related controls for service API invocations initiated by the API Invoker. |
| **CAPIF-4** | APF interacts over CAPIF-4 interface for publishing and un-publishing of service API information on CCF. |
| **CAPIF-5** | AMF interacts over CAPIF-5 interface or management of service APIs, API invoker and API provider domain function information, onboarding / offboarding of API provider domain functions |

### 6.1.3 Putting it all together

Figure  illustrates a plausible solution for the 5G-CLARITY Mediation Function. On the one hand, API orchestration block allows mapping 5G-CLARITY-facing APIs (e.g. Slice Manager APIs, Multi-WAT non-RT Controller APIs, etc.) into tenant-facing APIs (e.g., the ones captured in Table 6-1, Table 6-2 and Table 6-3). Then, there is an API Gateway, which builds upon CAPIF modules to make tenant-facing APIs available for 3rd party consumption. These modules, which are 'black boxed', include CAPIF Core Function (CCF) and API provider domain functions (AEF, APF, AMF). An existing CCF implementation is available in[11], released from EVOLVED-5G project [59].

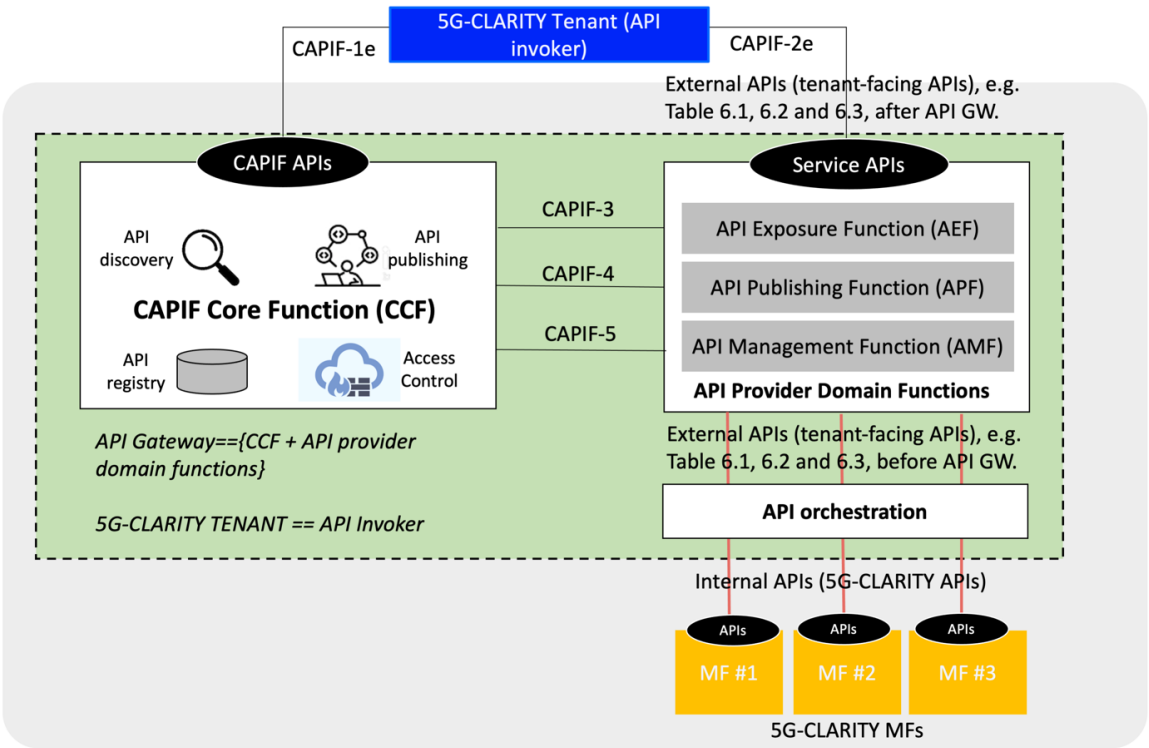



**Figure 6-3. Reference solution for 5G-CLARITY Mediation Function, using CAPIF framework for the API Gateway**

[11] https://github.com/EVOLVED-5G/CAPIF_API_Services

Note that the interaction with the 5G-CLARITY tenant is done through CAPIF-1e and CAPIF-2e interfaces.

- For integrity, replay and confidentiality protection of **CAPIF-1e interface** (see clause 6.4.3 from [54], CCF and API Invoker establish TLS session based on certificate based mutual authentication.
- For authentication, authorization and protection of **CAPIF-2e interface** (see clause 6.4.5 from [54], the API Invoker and the AEF apply the security method selected by the CCF. Different methods could apply, including TLS-PSK, TLS-PKI and TLS with OAuth2.0 (the preferred one).

To better understand how everything works when all the pieces are glued, Figure 6-4 illustrates a generic workflow. This workflow includes all the steps that are needed 1) for the tenant to discover "external APIs" and invoke them, and 2) for the 5G-CLARITY mediation to intercept "external API" calls and translate them into configurable actions into 5G-CLARITY system, by interacting with 5G-CLARITY MFs (e.g. Slice Manager, NFVO, Data Lake, etc.).

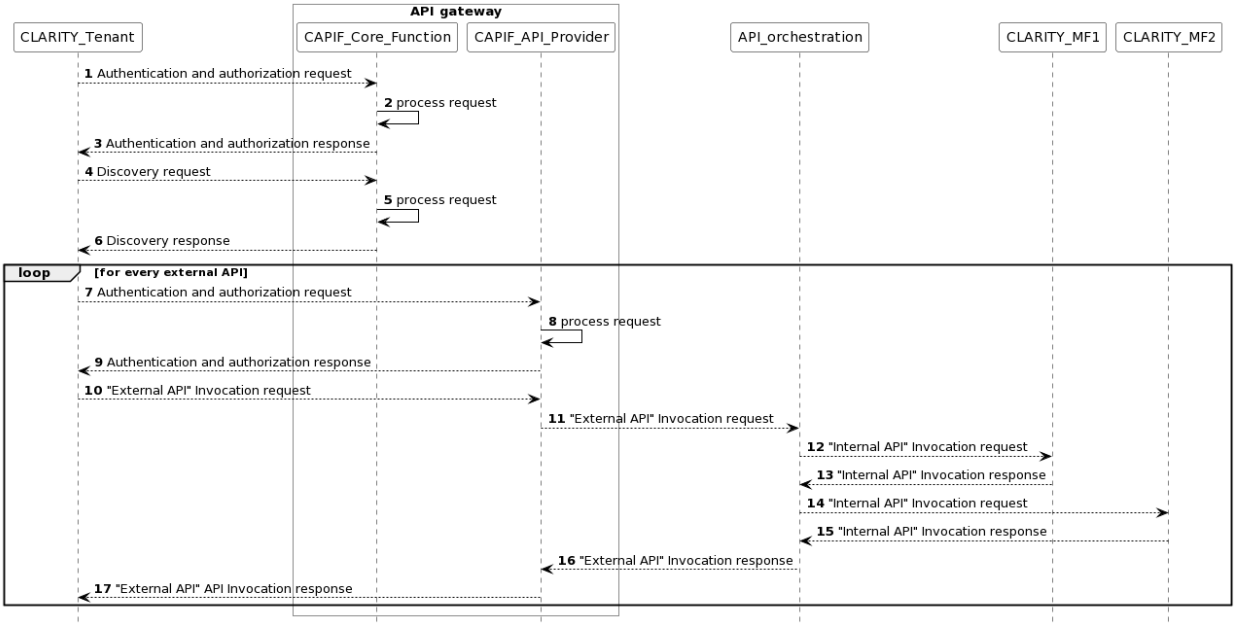



**Figure 6-4. Workflow**

The workflow steps are detailed below, along with some clarification on the usage of CAPIF interfaces.

**Step 1**: The CCF receives an authentication and authorization request from the tenant based on the identity and other information required for AuthN/Z of the tenant.

**Step 2**: The CCF processes the authentication and authorization request.

**Step 3:** The CCF provides the appropriate response to the tenant.

**Step 4:** The CCF receives a request for the discovery of "external APIs" information.

**Step 5:** The CCF processes the discovery request.

**Step 6:** The CCF provides the discovery response to the Tenant.

*NOTE: For steps 1, 3, 4 and 6, request-response message exchanges are sent over the CAPIF-1e interface*.

**Step 7:** The API provider receives an authorization request from the tenant based on the identity and other information required for authorization of the tenant.

**Step 8:** The API provider processes the authorization request.

**Step 9:** The API provider forwards the appropriate response to the tenant.

**Steps 10-11:** Upon receiving a request from the tenant on the invocation of "external API", the API provider

forwards it to the API orchestration block.

**Steps 12-14:** The API orchestration translates the "external API" call into the two "internal API" calls, following the correspondence rules captured in the mapping table.

**Steps 13-15**: The corresponding MFs provides appropriate responses to the API orchestration block.

**Step 16:** The API orchestration block transform the received "internal API" responses into a "external API" response, by applying the correspondence rules captured in the mapping table. This response is sent over to the API provider.

**Step 17**: The API provider forwards the "external API" response to the tenant. This is the counterpart of step 10.

*NOTE: For steps 7,9, 10 and 17, request-response message exchanges are sent over the CAPIF-2e interface*.

As it can be seen, steps 10-17 are repeated each time the tenant invokes an external API (tenant-facing API).

### 6.1.4 Mediation function in motion: use case-driven usage

So far, we have detailed the Mediation Function internals, including API orchestration (Section 6.1.1) and API gateway (Section 6.1.2). We also have pictured the workflow that captures the Mediation Function behaviour when policing request-response messages between the tenant and the 5G-CLARITY management functions (Section 6.1.3). In the following, we present use cases that fairly highlights the usage of Mediation Function capabilities. These use cases are based on the PoC scenario presented in Section 2, where:

- The vertical, which is an enterprise customer from industry 4.0 market, acts as the 5G-CLARITY tenant. This actor also is the 5G-CLARITY infrastructure owner.
- The B2B unit of a Communication Service Provider (CSP) acts a 5G-CLARITY system manager, in charge of operating all the 5G-CLARITY functions (network functions, management functions and application functions) deployed on 5G-CLARITY infrastructure. This actor has the know-how on operating private 5G networks, so it is the one that the vertical designates for this task.

From the above description, one can notice that the vertical is the actor which consumes tenant-facing APIs, while the CSP's B2B unit is the actor which deals with 5G-CLARITY-facing APIs.

Table 6-5 captures an example of the sequence of API calls that the vertical can make towards the CSP's B2B unit. For every API call, the workflow pictured in Figure 6-4 is triggered. As seen, the vertical first query available compute capabilities (API call #1), so it can know which compute flavor selects for application onboarding (API call #2). Once the application server is deployed on a 5G-CLARITY edge node, the customer can ask for the provisioning of a slice. To that end, it queries available wireless and transport capabilities (API calls #3 and #4), and then an issue a slice creation request (API call #5). When monitoring the slice status of through the notification URL, the vertical might observe that the behavior is not as expected (e.g., performance degradation). In this situation, the vertical can update the slice as required (API call #6).

**Table 6-5. Example of API Calls Sequence in an Industry 4.0 PoC**

| Sequence | "External API" call | API Reference | Description |
|---|---|---|---|
| 1 | Query compute resource capabilities | Table 6-1 | The tenant queries about the capabilities of 5G-CLARITY compute infrastructure. |
| 2 | Onboard application | Table 6-2 | Once the tenant discovers the compute capabilities, the tenant is in position to onboard the VAF, namely the object detection function. This workload is now deployed at 5G-CLARITY infrastructure. |
| 3 | Query wireless | Table 6-1 | The tenant queries about the capabilities of 5G-CLARITY |

| | | | |
|---|---|---|---|
| | resource capabilities | | wireless infrastructure. |
| 4 | Query transport resource capabilities | Table 6-1 | The tenant queries about the capabilities of 5G-CLARITY transport infrastructure. |
| 5 | Create slice | Table 6-3 | After discovering the capabilities of 5G-CLARITY infrastructure, the tenant can request for the allocation of a slice with certain QoS. |
| 6 | Update slice | Table 6-3 | At operation time, the tenant can request to modify the QoS profile or capacity associated to the provisioned slice. The reasons for this decision can be diverse, including unexpected traffic loads, failure nodes or SLA modifications. |

## 6.2 Experimental demonstration of 5G-CLARITY service delivery models

In this section we introduce some early demonstration of the 5G-CLARITY Service Delivery models.

### 6.2.1 Intent based NFVIaaS

In 5G-CLARITY D4.2 [1] we describe how 5G-CLARITY Service Delivery Model is enabled in the setup for UCI at the University of Bristol. To perform an early lab testing and experimental demonstration of the capability of the 5G-CLARITY framework to deliver Network-Function-Virtualization as Service we:

(a) Deployed and integrated in the Smart Internet Lab, early releases of software and hardware components of 5G-CLARITY intelligent, M&O, Service, and Infrastructure Strata.

(b) Extended the proposed setup of 5G-CLARITY framework for the UCI narratives reported in the D5.1 [60] for various experimental scenarios and KPI validations.

In this subsection first we introduce briefly the two scenarios extending the UCI narrative with targeted KPIs to be validated, the setup of 5G-CLARITY framework in the Smart Internet Lab and additional components required, followed by documented results, KPIs validations, and lessons learned as well as potential plans for WP5 final demonstration (to be extended in 5G-CLARITY D5.2 [20]).

First, we provide in Section 6.2.1.1 a short summary of the modules and integrations required to perform this demonstration. The next sections present our detailed designed of the intent-based slice provisioning mechanism.

#### 6.2.1.1 Overview of required implementation and integrations

Table 6-6 describes the different modules of the 5G-CLARITY architecture involved in the work reported in this section, highlighting the background and foreground with respect to 5G-CLARITY, as well as the module integrations validated through the experiments reported in this section.

**Table 6-6. Overview of modules involved in the demonstration of the intent-based NFVIaaS delivery model**

| Module | Background | Extensions in 5G-CLARITY | Responsible partner | Module integrations validated in this section |
|---|---|---|---|---|
| WEB SERVER VNF | N/a | Developed a descriptor implementing a Web Interace for Video Content Broadcasting. | UNIVBRIS | NFVO |
| UC1-DASHBOARD | N/a | Login, intent submission, OSM NBI API. | UNIVBRIS | Intent Engine and NFVO |

| | | | | |
|---|---|---|---|---|
| AI Engine | The AI Engine was built upon the Open-source Function-as-a-Service (FaaS) platform. OpenFaaS is a flexible and lightweight toolkit that advertises to be able to run anywhere, with any code and at any scale. A custom language configuration was created for AI Engine models which exposed additional monitoring capabilities to the model authors. | The OpenFaaS toolkit was extended with additional monitoring capabilities for model authors allowing for external monitoring through tools like Grafana. An interface was also designed between the AI Engine and Intent Engine for the intent driven execution of models. | LMI | Intent Engine |
| Intent Engine | The Intent Engine was built on Adaptive Policy EXecution (APEX). APEX is a fully featured policy engine that executes anything from simple to adaptive policies that can modify its behaviour based on the current conditions of the network and systems. The internal execution of policies behave similar to state machine allowing for high levels of flexibility and adaptability during the decision making process. | A collection of intent policies were designed, coordinated and executed within the APEX policy engine. A dynamic interface was also provided allowing the Intent Engine to communicate with service providers through a common execution. | LMI | OSM, AI Engine |
| NFVO | OSM v11 | None | UNIVBRIS | Intent Engine |

#### 6.2.1.2 5G-CLARITY framework setup at Smart Internet Lab of University of Bristol

Early release of 5G-CLARITY Framework software components is deployed in six virtual machines (VMs) hosted in the VIM/NFVI as early deployment of the 5G-CLARITY Edge Cluster setup of the Smart Internet Lab server room. The RAN components are deployed in the office of the HPN group for the multi-WAT demonstration.

The ***5G-CLARITY intelligent stratum*** setup includes two Virtual Machines (VM) deployed into the VIM/NFVI of the 5G-CLARITY Edge Cluster hosted in servers of the Smart Internet Lab. The VM1 host intent engine and VM2 the AI engine. The two VMs are connected to a Data Lake based on Elastic Search [] and to the 5G-CLARITY M&O and Infrastructure strata VMs.

The setup of ***5G-CLARITY M&O stratum*** components are the VM3 Network Function Virtualization Orchestrator (NFVO) and VM4 Software Defined Network Controller. The ***5G-CLARITY infrastructure stratum*** for UC-I early setup includes the 5G-CLARITY CPE and Assistant Robot, VM5 Robot M&C Platform, VM6 Test-

Dashboard to be used for monitoring and the multi-WAT RAM access nodes and network infrastructure.

The UC1 Dashboard application is developed to set up secure connectivity with NFVO (OSM) and d provide a user-friendly interface as well as all functionalities required by the UC1. The main functionalities are Intent registration or update, submission, and monitoring. The intent registration function is presented on Figure 6-5, in which as the first step the UC1 Dashboard application authenticates with NFVO in this case OSM, to generate a Token that will be used by the Intent Engine. After OSM sends the Token the UC1 Dashboard Application proceeds to register all intents related to OSM as well as a catalogue of intents related including third party services.

Ones this process in complete the third-party services can register and submit intents.

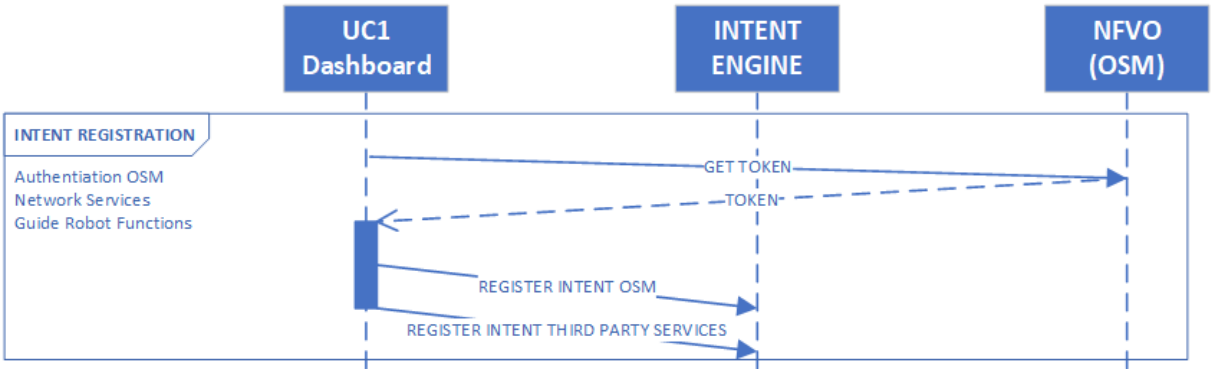



**Figure 6-5. Intent registration using the UC1 dashboard application**

### 6.2.1.3 Scenario 1 - Intent Engine and OSM enables NVFIaaS on UC1 Narrative 1

The Use Case 1 narrative 1 emulates a Standalone Non-Public Network deployment of 5G-CLARITY in a museum or shopping center D5.1. To demonstrate the NFVIaaS and perform some KPI measurement we extend in T4.3 WS2 we integrate the Intent Engine with NFVO-MANO. In this example a third-party advertising company is requesting through the Intent Engine the setup of an advertising system to promote products and services in the premises of the museum or shopping center. In this case the Guide Robot if Use Case I will use its tablets and sensors to deliver the advertising content hosted in VNF onboarded in the NFVI of the. The flow is summarized on Figure 6-6.

The process has two stages, the first focusing on the registration, authentication, submission, and processing of the intent request from the third-party institution. The intent engine might generate a single or multiple requests to components of the M&O stratum to satisfy the user demand. Figure 6-7 presents a simple sequence diagram of stage 1 including the process of registering the third-party institution and its immediate authentication before submitting their intent to deploy an interacting advertising service on the guide robot (Figure 6-6. Steps 1 and 2). Also, describe the process in which UC1-Dashboard submit the Intent and how the Intent engine sends the Rest API to the NFVO to initiate the onboarding of the service.

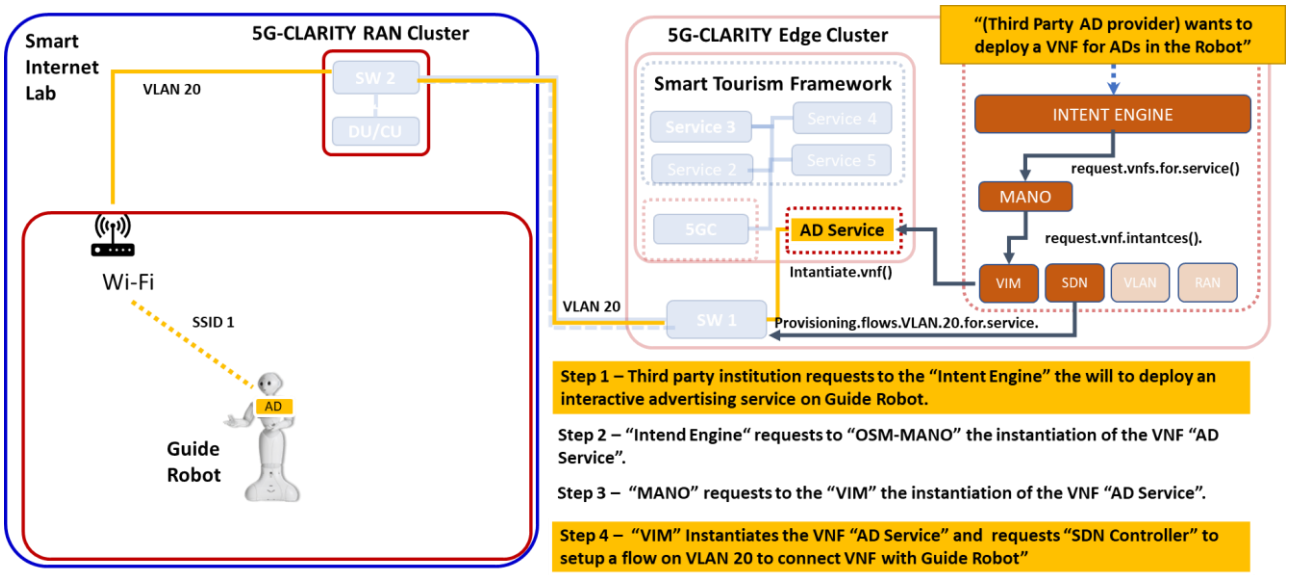



**Figure 6-6. Scenario 1 – setup and main flow**

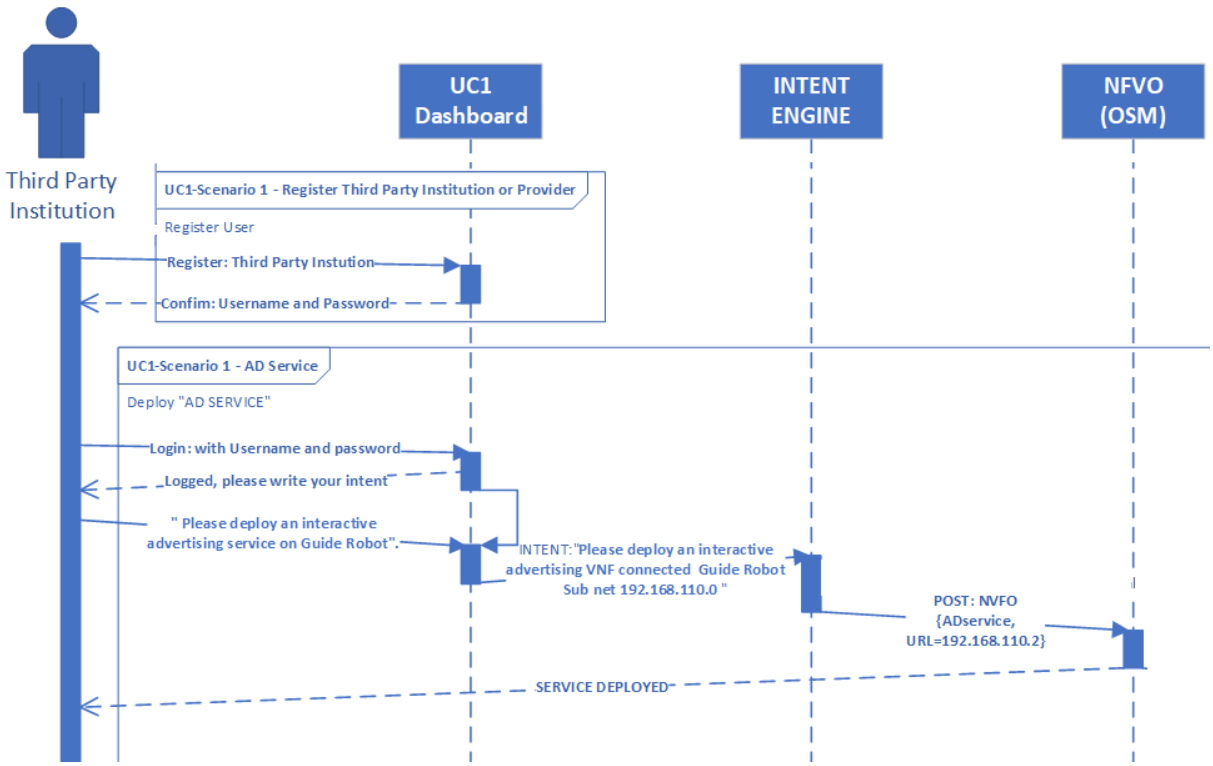



**Figure 6-7. Stage 1 submission and processing the intent to deployment a third-party service**

Next to the confirmation of the NFVO that the third-party service was deployed successfully into the M&O stratum, stage 2 begins with the deployment into the infrastructure stratum.

Figure 6-9 presents the sequence diagram of stage 2 (Steps 3 and 4), in which the third-party service completes its deployment into the NFVI, network, and the Guide Robot tablet. In this stage, the NFVO will deploy the VNF in the NFVI/VIM and start delivering the third-party services.

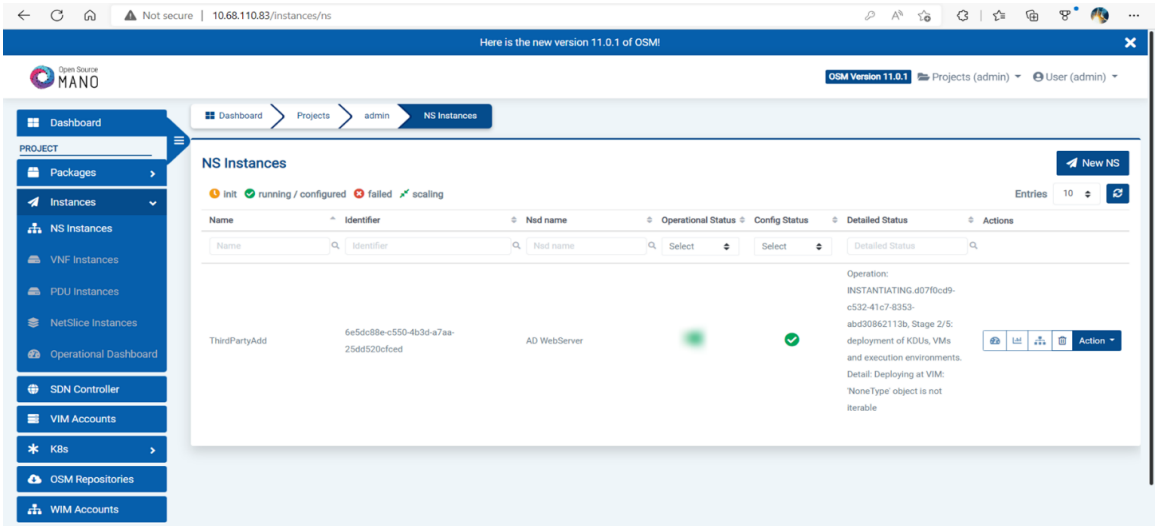



**Figure 6-8. Example of a running / configured NS Instance**

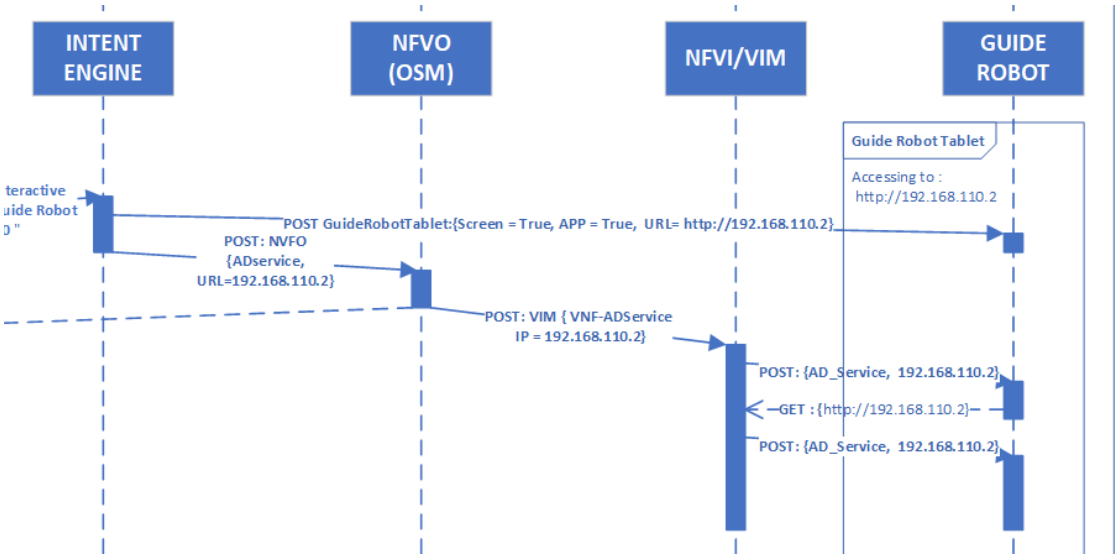



**Figure 6-9. Stage 2 deploying the VNF and starting the advertising in the Guide Robot tablet**

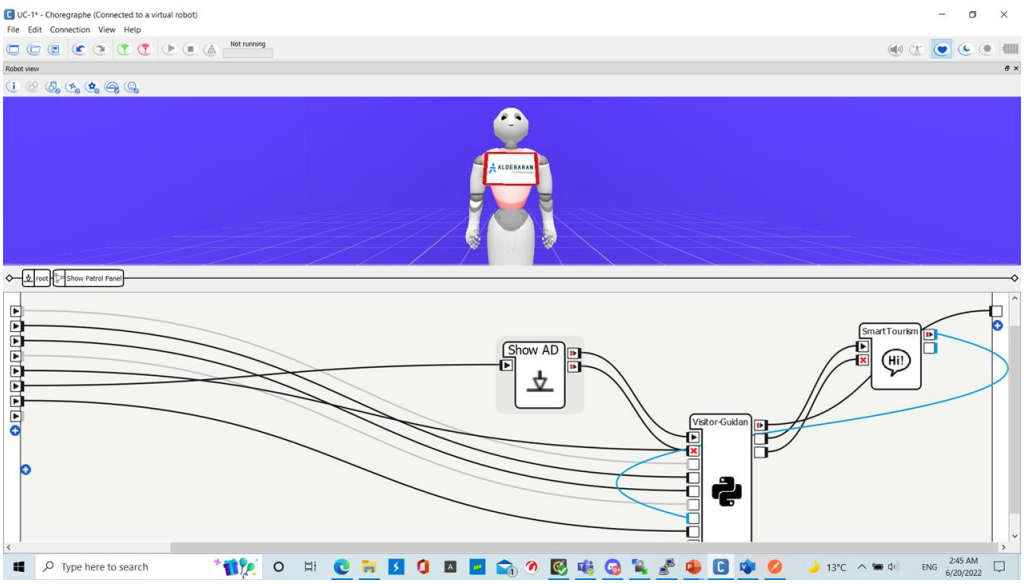



**Figure 6-10. Graphical Interface for virtual robot**

#### 6.2.1.4 Scenario 2 - Intent Engine and OSM enables NVFIaaS for AI services on UC1 Narrative 2

Scenario 2 adds to the already existing Scenario 1 the integration and testing of the AI Engine to provide Computer Vision services for suspicious activity detection. In this scenario the Intent Engine request to the AI Engine the Instantiation of an AI function into the NVFI with the onboarding of a VNF capturing video from the Cameras of the Robot to broadcast it to the UE of a Public Safety (police) officer connected into the wireless network. Figure 6-11 describe the setup and flows of the scenario 2 which also is divided in two stages as scenario 1.

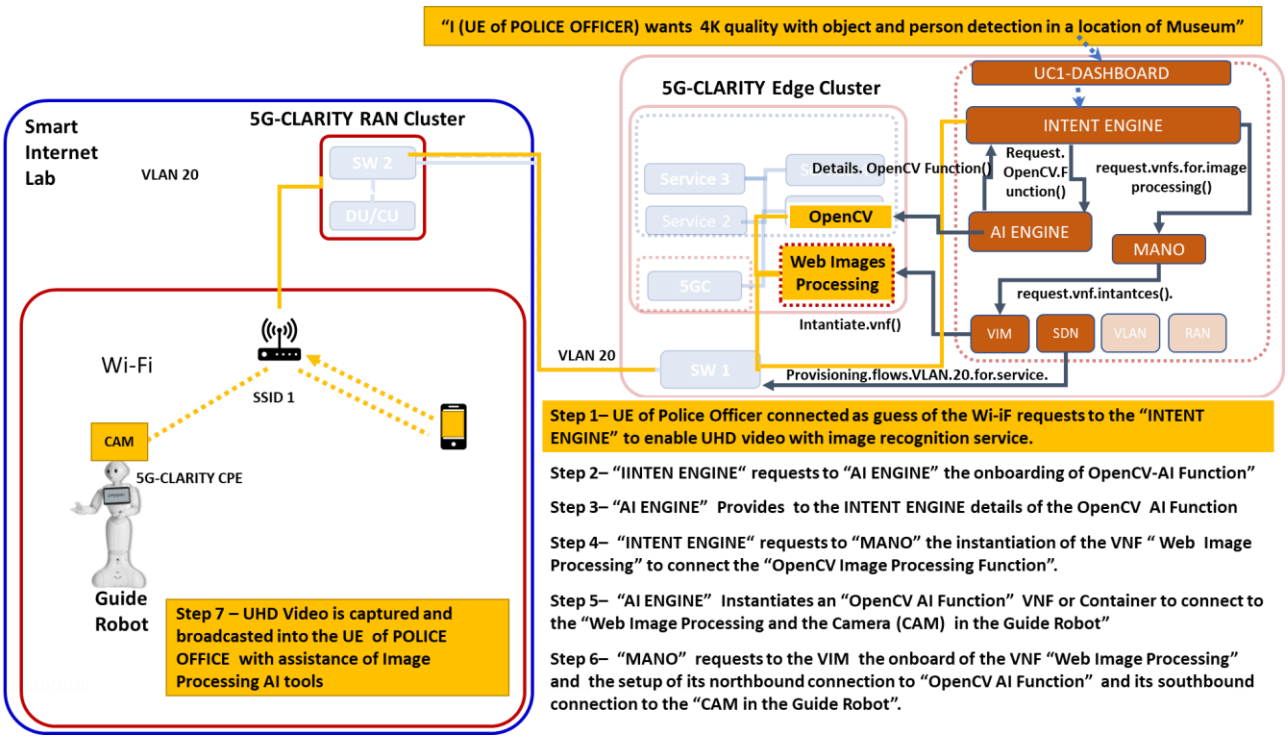



**Figure 6-11. Scenario 2 – Setup and main flow**

The stage one covers from the step 1 to the step 4 following the sequence diagram of Figure 6-11.

```
{
   "intent":{
      "request":"Deploy Open CV function in AI Engine",
      "parameters":{
         "service":"face-detect-OpenCV",
         "image":"alexellis2/facedetect:0.1"
      }
   }
}
```

**Figure 6-12. Example of intent message**

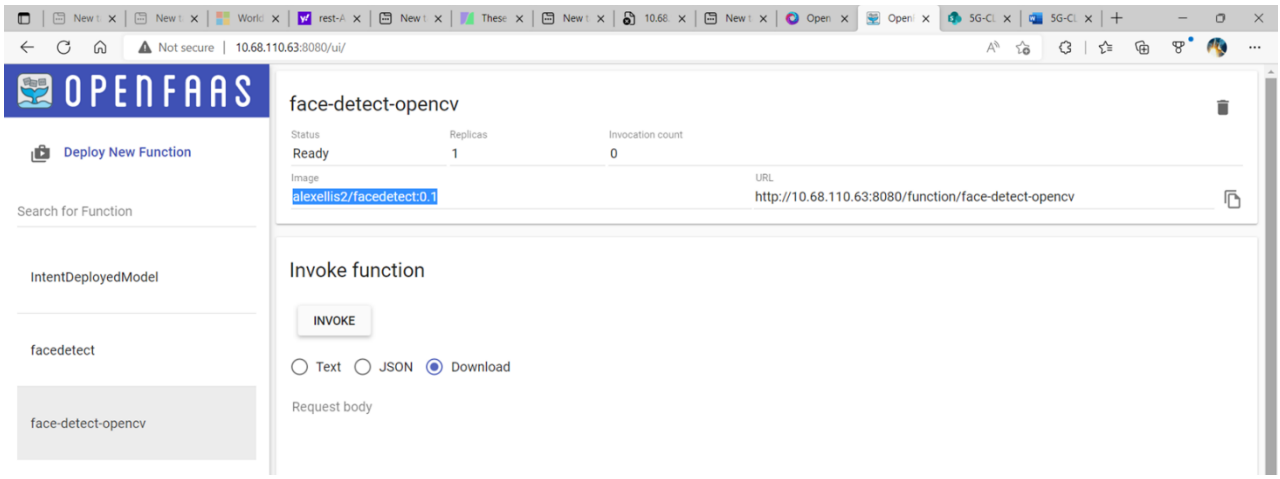



**Figure 6-13. AI Engine with deployed model "face-detect-opencv" in status "Ready"**

### 6.2.2 5G-CLARITY slice as a service (SlaaS)

In this section we describe how to achieve intent based control of the service and slice provisioning subsystem of the 5G-CLARITY M&O stratum. First, we provide in Section 6.2.2.1 a short summary of the modules and integrations required to perform this demonstration. The next sections present our detailed designed of the intent-based slice provisioning mechanism.

#### 6.2.2.1 Overview of required implementation and integrations

Table 6-7 describes the different modules of the 5G-CLARITY architecture involved in the work reported in this section, highlighting the background and foreground with respect to 5G-CLARITY, as well as the module integrations validated through the experiments reported in this section.

**Table 6-7. Overview of modules involved in the demonstration of the intent-based SlaaS delivery model**

| Module | Background | Extensions in 5G-CLARITY | Responsible partner | Module integrations validated in this section |
|---|---|---|---|---|
| ML Modules to manage slices | N/A | Developed from scratch | LMI | Slice Manager |
| AI Engine | The AI Engine was built upon the Open-source Function-as-a-Service (FaaS) platform. OpenFaaS is a flexible and lightweight toolkit that advertises to be able to run anywhere, with any code and at any scale. A custom language configuration was created for AI Engine models which exposed additional monitoring capabilities to the model | The OpenFaaS toolkit was extended with additional monitoring capabilities for model authors allowing for external monitoring through tools like Grafana. An interface was also designed between the AI Engine and Intent Engine for the intent driven execution of models. | LMI | Intent Engine |

| | | | | |
|---|---|---|---|---|
| | authors. | | | |
| Intent Engine | The Intent Engine was built on Adaptive Policy EXecution (APEX). APEX is a fully featured policy engine that executes anything from simple to adaptive policies that can modify its behaviour based on the current conditions of the network and systems. The internal execution of policies behave similar to state machine allowing for high levels of flexibility and adaptability during the decision making process. | A collection of intent policies were designed, coordinated and executed within the APEX policy engine. A dynamic interface was also provided allowing the Intent Engine to communicate with service providers through a common execution. | LMI | Slice Manager |
| Slice Manager | Initial implementation from 5G-CITY project [6] | REST based API integration with Intent Engine | I2CAT | Intent Engine |

A video demonstrating the integration of the previous software modules to provision 5G-CLARITY infrastructure slices through intents is available in [61].

#### 6.2.2.2 Intent based slice provisioning design

The proposed integration consists of the design of four dedicated modules in the AI engine, which are described in Figure 6-14, namely:

Slice Creation Workflow Model (SCW Model): Setting up a 5G-CLARITY slice is a complex process that requires multiple interactions with the Slice Manager function. This process is described in detail in Section 2.1 of D4.2 [1]. The role of the SCW model is to coordinate all the required interactions with the Slice Manager in a single AI model. Thus, when the Intent Engine parses an intent related to a slice provisioning aspect it instantiates an SCW model in the AI engine to serve this intent.

Radio Node Selection Model (RNS Model): A fundamental aspect of the 5G-CLARITY slice provisioning process is to determine what radio access nodes need to be support the provisioned slice, where nodes could be of type 5GNR, Wi-Fi or LiFi. The role of the RNS model is to assist in the selection of the radio nodes required for the slice.

Compute Requirements Model (CR Model): Another key aspect in the 5G-CLARITY slice provisioning process is to determine the compute resources (#vCPUs, RAM, storage) required in the Edge cluster to serve the slice. The role of the CR model is to determine these requirements.

Slice QoS Model (SQS Model): A 5G-CLARITY slice can be configured with a specific level of QoS, e.g. QCI and AMBR (Allowed Maximum Bit Rate) parameters for 5GNR, or Access Category and airtime for Wi-Fi and LiFi. The role of SQS model is to determine the required QoS parameters for the provisioned slice.

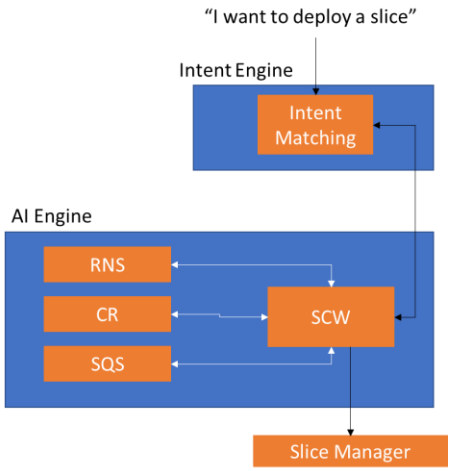



**Figure 6-14. Intent and AI engine design for intent level control of service and slice provisioning subsystem**

### 6.2.2.3 Design of slice provisioning intent

The slice provisioning intent contains two components: i) the intent body and ii) the intent parameters. We described them next:

Intent request: Can be any complex English text indicating the intention to provision a slice that can be mapped to the descriptions of the SCW model provided during the registration phase. For example, the sentences: "*I want to create a slice using the Slice Creation Workflow*" or "*Setup a slice*" are valid intent bodies.

Intent parameters: A set of qualifiers that allow to customize the type of slice being provisioned. The following parameters are supported:

Name: Indicates the name used in the Slice Manager for the created slice, e.g. *Name: my-slice*

User-list: Indicates the set of users identified by an IMSI number that will be provisioned in the core network deployed as part of this slice. Refer to Section 2 for a detailed explanation about how a 5G-CLARITY slice is provisioned. Example: *user-list: {IMSI-list}*

Location: Indicates the geographical coordinates where the provisioned slice needs to be active. This will be used to determine the 5GNR cells or Wi-Fi/LiFi APs that need to be part of the slice. Example of location defined as a rectangular region: *Location: {lat1-long1, lat2-long2, lat3-long3, lat4-long}*

Technology: Indicates the types of access technologies that are considered as part of the slice. This allows to filter across 5GNR cells, and Wi-Fi and LiFi APs. For example, *technology: {Wi-Fi/5g/lifi}*, *technology: {Wi-Fi/lifi}, technology: {Wi-Fi/5g}, technology: {5g}*

Services: Identify the network-services IDs, as hosted in the NFVO, which need to be instantiated along with this slice. Example *services: {ns-id1, ns-id2}*

Quality: Indicates the type of QoS that will be configured in the DNN established to serve the devices connected to this slice. For example, *quality: {gold/silver/bronze}*

The presence of the previous parameters determines the behaviour of the slice related models in the AI engine, as indicated in the following tables where we provide Intent realization examples and described the behaviour of each AI engine model:

Table 6-8 provides an example of intent provisioning a slice that only has a default set of compute resources. Notice that no information is provided to determine the amount of resources required for the

slice.

Table 6-9 provides an example of intent provisioning a slice with both compute and radio resources. Notice how the RNS model admits different implementations, i.e. default and ML-powered, to determine which are the actual radio nodes that need to be part of the provisioned slice.

Table 6-10 provides an example of intent provisioning a slice with compute, radio and applications. Notice how the CR module admits a default and an ML-powered implementation to determine the compute resources required to support the services in the slice.

Table 6-11 provides an example of intent provisioning a slice with compute, radio and a defined level of QoS for the users connecting to the slice. Notice how the SQS admits a default and an ML-powered implementation to determine the actual QoS configuration to be applied to the slice.

**Table 6-8. Intent Provisioning Slice with Only Compute**

| **Intent Example** | **I want to create a slice using the Slice Creation Workflow** |
|---|---|
| **Parameters** | ***Name: my-slice*** |
| **SCW model** | Defines a slice with only a compute chunk<br>Parameters used by SCW to generate Intents towards slice manager<br>*Slice composition = 1 compute chunk [name: my-slice-compute-chunk; username: my-slice-username; pass: my-slice-pass; description: my-slice] + 1 network chunk (data-network [name: my-slice-network-chunk]).*<br>*GET user-id, GET compute-id, GET physical-network-id* |
| **CR model** | Assume minimal CPU/RAM/SRG requirements (vCPU=1, RAM=1GB, SRG=10GB) |
| **RNS model** | Not involved |
| **SQS model** | Not involved |

**Table 6-9. Intent Provisioning Slice with Compute and Radio**

| **Intent Example** | **I want to create a slice using the Slice Creation Workflow** |
|---|---|
| **Parameters** | ***Name: my-slice, user-list: {IMSI-list}, location: {lat1-long1, lat2-long2, lat3-long3, lat4-long}, technology: {Wi-Fi/5g/lifi/5g+Wi-Fi/5g+lifi, …}*** |
| **SCW model** | Defines a slice with compute chunk and radio chunk<br>Parameters used by SCW to generate Intents towards slice manager<br>*Slice composition = 1 compute chunk [name: my-slice-compute-chunk; username: my-slice-username; pass: my-slice-pass; description: my-slice] + 2 network chunks (data-network [name: my-slice-network-chunk-1] and access-network [name: my-slice-network-chunk-2]) + 1 radio chunk [name: my-slice-radio-chunk] + 1 radio service [name: my-slice-radio-service].*<br>*GET user-id, GET compute-id, GET physical-network-id and GET ran-infra-id* |
| **CR model** | Assume the CPU/MEM/SRG defaults for core network + dhcp server (without network service) |
| **RNS model** | Location: Filters the location LET/LONG reported in SM GET configured topology to determine boxes within the location<br>Technology: Selects only nodes of defined type within location<br>**Default behavior**: Selects all nodes that meet the location/technology criteria<br>**ML powered behavior**: It is free to select radio access nodes according to current network state |
| **SQS model** | Do not include QCI or AMBR parameters in slice activation (Slice Manager will fill the |

| | |
|---|---|
| | defaults).<br>*SSID: my-slice; PLMN ID: IMSI-list[0][:5]* |

**Table 6-10. Intent Provisioning Slice with Compute, Radio and Applications**

| **Intent Example** | **I want to create a slice using the Slice Creation Workflow** |
|---|---|
| **Parameters** | ***Name: my-slice, user-list: {IMSI-list}, technology: {Wi-Fi/5g/lifi/5g+Wi-Fi/5g+lifi, …}, services: ns-id1, ns-id2*** |
| **SCW model** | Defines a slice with compute chunk and radio chunk<br>Parameters used by SCW to generate Intents towards slice manager<br>*Slice composition = 1 compute chunk [name: my-slice-compute-chunk; username: my-slice-username; pass: my-slice-pass; description: my-slice] + 2 network chunks (data-network [name: my-slice-network-chunk-1] and access-network [name: my-slice-network-chunk-2] ) + 1 radio chunk [name: my-slice-radio-chunk] + 1 radio service [name: my-slice-radio-service] + NS1, NS2.*<br>*GET user-id, GET compute-id, GET physical-network-id and GET ran-infra-id* |
| **CR model** | GET ns-ids and parse NS compute requirements for each ns-id<br>**Default behavior**: Returns compute requirements equal sum of ns-id-i reqs plus default core network + dhcp server requirements<br>**ML powered behavior**: Based on observation of current utilization in Edge nodes adjusts compute requirements |
| **RNS model** | Technology: Selects only nodes of defined type within location<br>**Default behaviour**: Selects all nodes that meet the location/technology criteria<br>**ML powered behaviour**: It is free to select radio access nodes according to current network state |
| **SQS model** | Do not include QCI or AMBR parameters in slice activation (Slice Manager will fill the defaults).<br>*SSID: my-slice; PLMN ID: IMSI-list[0][:5]* |

**Table 6-11. Intent Provisioning Slice with Compute, Radio and QoS Definition**

| **Intent Example** | **I want to create a slice using the Slice Creation Workflow** |
|---|---|
| **Parameters** | ***Name: my-slice, user-list: {IMSI-list}, technology: {Wi-Fi/5g/lifi/5g+Wi-Fi/5g+lifi, …}, quality: {gold, silver, bronze}*** |
| **SCW model** | Defines a slice with compute chunk and radio chunk<br>Parameters used by SCW to generate Intents towards slice manager<br>*Slice composition = 1 compute chunk [name: my-slice-compute-chunk; username: my-slice-username; pass: my-slice-pass; description: my-slice] + 2 network chunks (data-network [name: my-slice-network-chunk-1] and access-network [name: my-slice-network-chunk-2] ) + 1 radio chunk [name: my-slice-radio-chunk] + 1 radio service [name: my-slice-radio-service].*<br>*GET user-id, GET compute-id, GET physical-network-id and GET ran-infra-id* |
| **CR model** | Assume the CPU/MEM/SRG defaults for core network + dhcp server (without network service) |
| **RNS model** | Technology: Selects only nodes of defined type within location<br>**Default behavior**: Selects all nodes that meet the location/technology criteria<br>**ML powered behavior**: It is free to select radio access nodes according to current network state |

| | |
|---|---|
| **SQS model** | **Default behaviour**:<br>*Gold: QCI = 1 Wi-Fi_ac=AC_VO*<br>*Silver: QCI = 4, Wi-Fi_AC=AC_VI*<br>*Bronze: QCI = 9, Wi-Fi_AC=AC_BE*<br>**ML powered behaviour**: Optimize UL/DL AMBR of each slice based on measured usage to enforce isolation across slices. Optimize airtime weight for Wi-Fi service<br>*SSID: my-slice; PLMN ID: IMSI-list[0][:5]* |

#### 6.2.2.4 Intent triggered slice provisioning workflow

Figure 6-15 describes the sequence workflow across the Intent Engine, the different models in the AI Engine and the Slice Manager that is the final recipient of the slice provisioning configuration.

All intents triggered towards the Intent Engine are indicated in red. The workflow starts by an external slice provisioning intent, defined as indicated in the previous section. The SCW then builds subsequently intents with the various configuration steps required by the Slice Manager to provision a slice. The SCW builds the intermediate intents by gathering the necessary context from the RNS, CR and SQS models. The intermediate intents have an intent body that can be mapped to the corresponding Slice Manager endpoint, and an intent body that maps to the JSON body that needs to be used by the Intent Engine to generate the request towards the Slice Manager. Note how in the proposed implementation the models in the AI engine are completely decoupled from the Slice Manager, since all interactions are mediated through Intents by the Intent Engine, which is who has the Slice Manager registered as a provider.

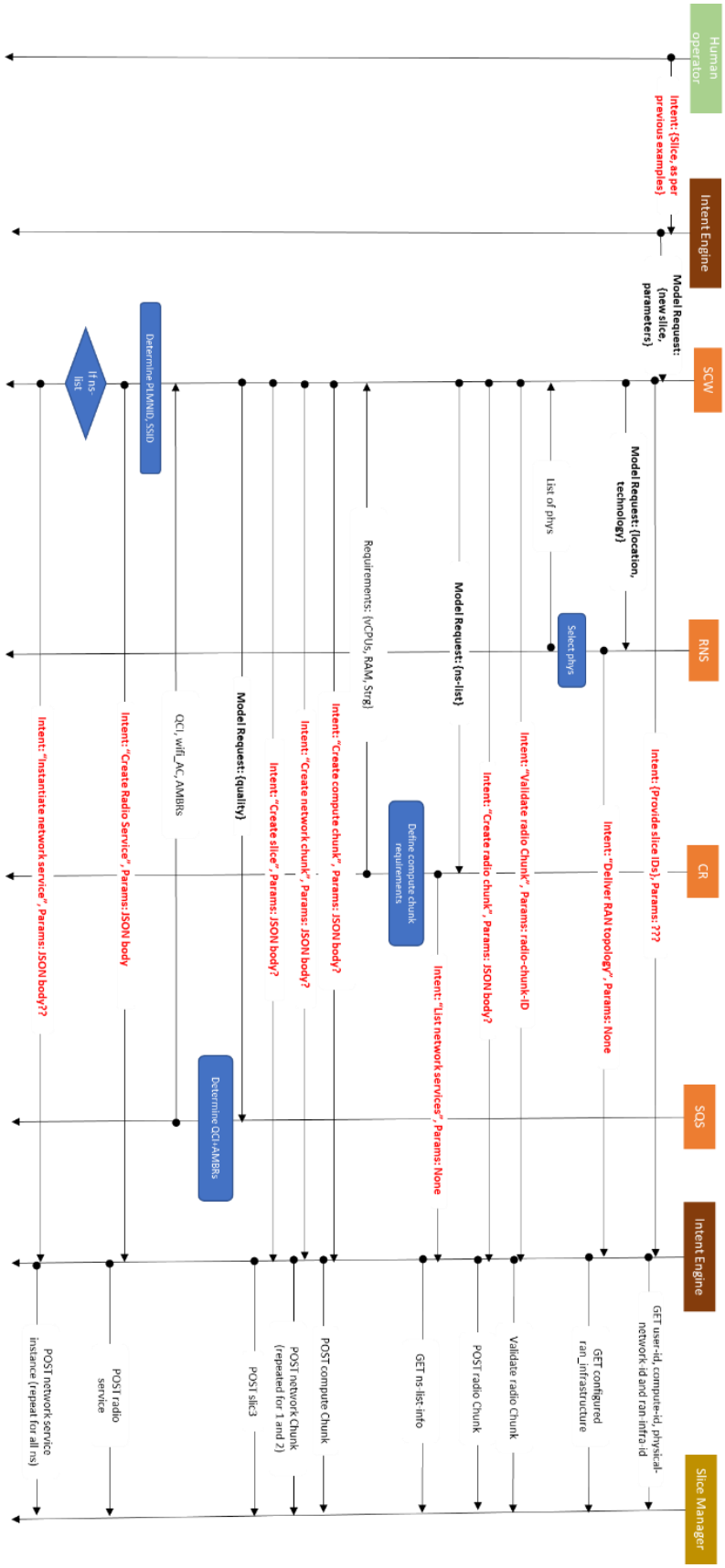



**Figure 6-15. Intent based slice provisioning workflow**

#### 6.2.2.5 Intent based slice provisioning: functional validation

LMI and i2CAT have collaborated to provide a functional demonstration of the intent-based slice provisioning concept presented in this section. A video demonstrating this integration has been uploaded to the 5G-CLARITY YouTube channel [13].

This functional demonstration required the following steps:

- Implementation of the SCW, RNS, CR and SQS models as independent models in the AI engine.
- Registration of the SCW model and Slice Manager as providers in the Intent Engine
- Integration of the SCW calls with the Slice Manager model
- Functional demonstration carried out with a lab-based testbed hosted by i2CAT. The testbed is the same used for the ETSI PoC demonstrator described in section 2 and features an Amarisoft 5GNR base station and a Wi-Fi access point.

Figure 6-16 depicts the high-level intent generated by the user, which uses the format described in Table 6-9. We can see that the intent body expresses "*Create a slice using a Slice Creation Workflow*", whereas parameters are provided to signal the users that need to be part of the slice ("*imsi*") and the cells that need to be involved ("*technology*").

Figure 6-17 depicts the final service activation POST method received by the Slice Manager from the SCW model, after all previous slice creation steps described in Figure 6-15 have been successfully executed. We can see how the SCW module is activating a Wi-Fi network with SSID "*MYSSID*" and a 5G network with PLMNID "*00103*" with an APN called "*clarity*" with one configured IMSI.

Figure 6-18 demonstrates how the 5G-CLARITY CPE can connect to the deployed network slice once this has been set up. First, we see on the left side how the 5G modem scans and find a network with PLMNID "*00103*". Second, on the right side we see how the CPE can connect to the "*clarity*" APN and successfully communicates with a ping.

The interested reader is referred to the 5G-CLARITY YouTube channel [13] where a video of this demonstration has been uploaded.

```
{
    "intent": {
        "request": "Create a slice using Slice Creation Workflow",
        "parameters" : {
            "name" : "nova",
            "user-list" : [
                {
                    "imsi": "001035432100005"
                }
            ],
            "location" : {
                "latitude" : 0.0,
                "longitude" : 0.0
            },
            "technology" : [
                "AMARISOFT_CELL",
                "SUB6_ACCESS"
            ]
        }
    }
}
```

**Figure 6-16. Intent issued for slice provisioning**

```
{
  "name": "REST_Action_Event",
  "version": "0.0.1",
  "nameSpace": "org.onap.policy.apex.auth.clieditor",
  "source": "CLIEditorSource",
  "target": "CLIEditorTarget",
  "body": {
    "name": "nova-radio-service",
    "user_id": "615c01f49a65e9000bceb4c8",
    "slic3_id": "62bafd9951188a003012dbe6",
    "wifi_config": {
      "ssid": "MYSSID",
      "encryption": "NONE",
      "password": ""
    },
    "cellular_config": {
      "plmn_id": "00103",
      "apn": "clarity",
      "ue_data": [
        {
          "imsi": "001035432100005"
        }
      ]
    }
  },
```


**Figure 6-17. Service activation call received by slice manager**

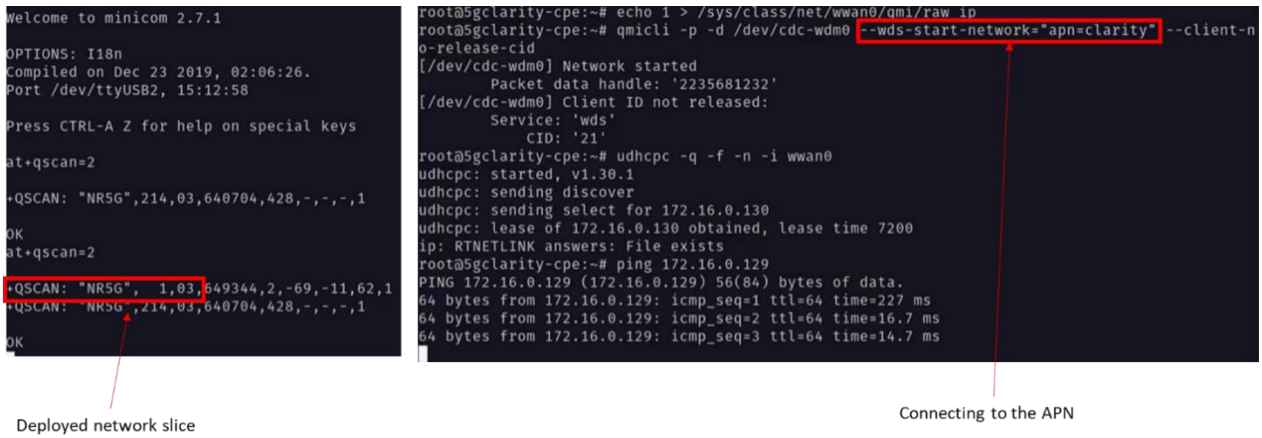



**Figure 6-18. Connection from 5G-CLARITY CPE to deployed network slice**

# 7 Conclusions

This deliverable has presented the evaluation of E2E 5G Infrastructure and Service Slices, and of the Developed Self-Learning ML Algorithms. Expanding on 5G-CLARITY D4.2 [1], we have detailed the final implementation of the 5G-CLARITY Service and Slice Provisioning System and demonstrated it through an intent-driven Slice as a Service use case. The ML models have been expanded and we have produced a wide array of models for a variety of management and control decisions.

In **Section 2** we present the instantiation of network services in each domain and demonstrate the automated deployment of end-to-end network slices comprising private and public domains in less than 10 minutes. In **Section 3** we describe and integrate various data sources with the data lake. We describe data transport within the Data Semantic Fabric and produced an experimental scenario for network telemetry collection. Finally, we detailed the integration of the Data Lake and Data Semantic Fabric. In **Section 4** we provide a pool of ML-based algorithms which address key decisions made in the 5G-CLARITY system. These ML-based algorithms, used in within the Intelligence Stratum, assist in the control and management of private 5G networks. In **Section 5** we present the creation, deployment, and execution of the NLoS identification algorithm in the AI Engine and its communication with other entities of the network. In **Section 6** we present a plausible solution for the Mediation Function and demonstrate the 5G-CLARITY Service Delivery models. Finally, we validate different AI engines using data and task offloading tests, in a variety of scenarios.

The next deliverable, 5G-CLARITY D5.2 [20], will report on the integration of solutions developed in WP3 and WP4. The integrated setup will be evaluated, the results of which will inform the UC1, UC2.1 and UC2.2 demonstrations regarding setup and deployment. Validation tests will be detailed in the deliverable.

# 8 Bibliography


[1] 5G-CLARITY D4.2, "Validation of 5G-CLARITY SDN/NFV Platform, Interface Design with 5G Service Platform, and Initial Evaluation of ML Algorithms," June 2021.

[2] 5G-CLARITY D2.2, "Primary System Architecture," October 2020.

[3] 5G-CLARITY D3.2, "Design Refinements and Initial Evaluation of the Coexistence, Multi-Connectivity, Resource Management and Positioning Frameworks," 2021.

[4] ETSI OSM, [Online]. Available: https://osm.etsi.org/.

[5] OpenStack, [Online]. Available: https://www.openstack.org/.

[6] 5GCity Project, [Online]. Available: https://www.5gcity.eu/.

[7] 5G-PICTURE Project, [Online]. Available: https://www.5g-picture-project.eu/.

[8] 5G CLARITY 5GZorro ETSI ZSM PoC Demo, [Online]. Available: https://www.youtube.com/watch?v=heU_ceO315s&t=225s.

[9] IETF RFC 6421, *Network Configuration Protocol (NETCONF).*

[10] IETF RFC 6020, *YANG – A Data Modeling Language for the Network Configuration Protocol (NETCONF).*

[11] H2020-ICT-2019, "5G ZORRO- Zero-tOuch secuRity and tRust for ubiquitous cOmputing and connectivity in 5G networks.," [Online]. Available: https://5g-ppp.eu/5gzorro/.

[12] ETSI, "Zero touch network & Service Management (ZSM)," [Online]. Available: https://www.etsi.org/technologies/zero-touch-network-service-management.

[13] 5G-CLARITY, "Youtube Channel," [Online]. Available: https://www.youtube.com/channel/UCtAZgpXA-Ud-l8TMfTBPxxw/featured.

[14] A. Fernández-Fernández et al., *Multi-Party Collaboration in 5G Networks via DLT-Enabled Marketplaces: A Pragmatic Approach,* 2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit), 2021, pp. 550-555.

[15] V. Theodorou et al., *Blockchain-based Zero Touch Service Assurance in Cross-domain Network Slicing,* 2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit), 2021, pp. 395-400.

[16] TM Forum, *TM Forum ODF Concepts and Principles; Business Process, Information and Application Frameworks,* TM Forum Reference GB991, 2021.

[17] 5GZORRO Consortium, Deliverable D3.1, *Design of the evolved 5G Service layer solutions,* 2021.

[18] GSM Association, *Generic Network Slice Template,* GSM Association Official Document NG.116, 2021.

[19] 5GZORRO Consortium, Deliverable D4.1, *Design of Zero Touch Service Management with Security & Trust solutions,* 2021.

[20] 5G-CLARITY D5.2, "Integration of solutions and validation," 2022.

[21] 5G-CLARITY D3.3, "Complete Design and Final Evaluation of the Coexistence, Multi-Connectivity, Resource Management, and Positioning Frameworks," 2022.

[22] 5TONIC, *AN OPEN RESEARCH AND INNOVATION LABORATORY FOCUSING ON 5G TECHNOLOGIES.*

[23] *5G CLARITY 5GZorro ETSI ZSM PoC Demo,* 2022. Demo Video Available at: https://www.youtube.com/watch?v=heU_ceO315s .

[24] IETF, *Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks,* 2014.

[25] Vidal, I., Nogales, B., Lopez, D., Rodríguez, J., Valera, F., & Azcorra, A, *A Secure Link-Layer Connectivity Platform for Multi-Site NFV Services,* Electronics, 2021.

[26] 5G-CLARITY D2.4, "Final System Architecture and Its Evaluation," 2022.

[27] *ETSI GS CIM 009 - V1.5.1 - Context Information Management (CIM); NGSI-LD API,* ETSI, November 2021.

[28] "Scorpio ngsi-ld broker," [Online]. Available: https://github.com/ScorpioBroker/ScorpioBroker. [Accessed 18 04 2022].

[29] "Apache NiFi: An easy to use, powerful, and reliable system to process and distribute data," [Online]. Available: https://nifi.apache.org/. [Accessed 06 06 2022].

[30] "Apache Flink: Stateful Computations over Data Streams," [Online]. Available: https://flink.apache.org/. [Accessed 06 06 2022].

[31] "gNMIc," [Online]. Available: https://gnmic.kmrd.dev/. [Accessed 02 06 2022].

[32] S. Chisholm, H. Trevino, *NETCONF Event Notifications,* IETF RFC 5277, 2008.

[33] OpenDaylight Documentation, "YANG Tools Developer Guide," 02 06 2022. [Online]. Available: https://docs.opendaylight.org/en/latest/developer-guides/yang-tools.html.

[34] L. Lhotka, *JSON Encoding of Data Modeled with YANG,* IETF RFC 7951, 2016.

[35] "public/openconfig-interfaces.yang at master openconfig/public," [Online]. Available: https://github.com/openconfig/public/blob/master/release/models/interfaces/openconfig-interfaces.yang. [Accessed 10 06 2022].

[36] "yang/openconfig-interfaces.yang at master aristanetworks/yang," [Online]. Available: https://github.com/aristanetworks/yang/blob/master/EOS-4.28.0F/openconfig/public/release/models/interfaces/openconfig-interfaces.yang. [Accessed 10 06 2022].

[37] "Arista EOS - Cloud Network Operating System," [Online]. Available: https://www.arista.com/en/products/eos. [Accessed 02 06 2022].

[38] "Keysight BreakingPoint," [Online]. Available: https://www.keysight.com/es/en/products/network-security/breakingpoint.html. [Accessed 02 06 2022].

[39] B. Lengyel, B. Claise, "A File Format for YANG Instance Data," IETF RFC 9195, February 2022. [Online].

Available: https://datatracker.ietf.org/doc/html/rfc9195.

[40] Merlin, S., Barriac, G., Sampath, H., Cariou, L., Derham, T., Rouzic, J.-P. L., . . . Wang, X, *TGax Simulation Scenarios,* 2015.

[41] A. Purwita, Studies of Optical Wireless Communications: Random Orientation Model, Modulation, and Hybrid WiFi and LiFi Networks, University of Edinburgh, 2021.

[42] Z. Xu, J. Tang, C. Yin, Y. Wang and G. Xue, "Experience-Driven Congestion Control: When Multi-Path TCP Meets Deep Reinforcement Learning," *IEEE Journal on Selected Areas in Communications,* vol. 37, no. 6, pp. 1325-1336, 2019.

[43] C. Paasch, S. Barre, et al., "Multipath TCP in the Linux Kernel," 2022. [Online]. Available: https://www.multipath-tcp.org.

[44] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Offpolicy maximum entropy deep reinforcement learning with a stochastic actor," *International conference on machine learning.,* p. 1861–1870, 2018.

[45] J. Schulman, N. Heess, T. Weber, and P. Abbeel, *Gradient estimation using stochastic computation graphs,* Advances in Neural Information Processing Systems, 2015.

[46] 5G-CLARITY D4.1, "Initial Design of the SDN/NFV Platform and Identification of Target 5G-CLARITY ML Algorithms," 2020.

[47] S. Guadarrama, et. al, *TF-Agents: A library for Reinforcement learning in TensorFlow,* 2018.

[48] Mao Y, Zhang J, Song S H, et al, *Power-delay tradeoff in multi-user mobile-edge computing systems,* 2016 IEEE global communications conference (GLOBECOM), 2016.

[49] Wang F, Xu J, Wang X, et al, *Joint offloading and computing optimization in wireless powered mobile-edge computing systems,* IEEE Transactions on Wireless Communications, 2017.

[50] B. S. Baker, *A new proof for the first-fit decreasing bin-packing algorithm,* Journal of Algorithms, 1985.

[51] *Mobile edge computing (mec); framework and reference architecture,* ETSI DGS MEC, 2016.

[52] A. Karamyshev, E. Khorov, A. Krasilov and I. Akyildiz, *Fast and accurate analytical tools to estimate network capacity for URLLC in 5G systems,* Comput. Netw., 2020.

[53] 5G-CLARITY D2.1, "Use-Case Specifications and Requirements," 2020.

[54] IEEE 802.1 Working Group, *IEEE Draft Standard for Local and metropolitan area networks–Bridges and Bridged Networks Amendment: Asynchronous Traffic Shaping,* IEEE Std 802.1Qcr-2020, February 2020.

[55] J. Prados-Garzon, T. Taleb and M. Bagaa, *Optimization of Flow Allocation in Asynchronous Deterministic 5G Transport Networks by Leveraging Data Analytics,* IEEE Transactions on Mobile Computing, 2021.

[56] A. Bouillard, L. Jouhet and E. Thierry, *Tight Performance Bounds in the Worst-Case Analysis of Feed-Forward Networks,* IEEE INFOCOM, 2010.

[57] 5G-CLARITY Intelligent Stratum Demo v02, [Online]. Available: https://www.youtube.com/watch?v=-FgdyJBPiJQ.

[58] 3GPP TS 23.222, *Common API Framework for 3GPP Northbound APIs,* 3GPP, September, 2019.

[59] H2020, *EVOLVED-5G.*

[60] 5G-CLARITY 5.1, "Specification of Use Cases and Demonstration Plan," February 2021.

[61] 5G-CLARITY Intent Based Slice Management Demonstration, [Online]. Available: https://www.youtube.com/watch?v=5Jsc2ds-etI.

[62] 3GPP TS 23.222, *LTE; 5G; Common API Framework for 3GPP Northbound APIs,* 3GPP, May 2022.

[63] Paul Borman et al, "gRPC Network Management Interface (gNMI) Specification," [Online]. Available: https://github.com/openconfig/reference/blob/master/rpc/gnmi/gnmi-specification.md#35-subscribing-to-telemetry-updates. [Accessed 02 06 2022].