



PREDICT 6G

D3.4

Implementation of selected release 2 AI-driven inter-domain network control, management, and orchestration innovations

UPC



Funded by
the European Union

©PREDICT-6G 2023-2025

Revision v4.1

Work package	WP3
Task	T3.1, T3.2, T3.3, T3.4
Due date	31-03-2025
Submission date	31-03-2025
Deliverable lead	UPC
Version	v4.1
Authors	<p>Salvatore Spadaro, Fernando Agraz, Albert Pagès, Luis Velasco, Marc Ruiz, Davide Careglio (UPC)</p> <p>Matteo Ravalli, Pietro Giardina, Juan Brenes (NXW)</p> <p>Sebastian Robitzsch, Muhammad Awais Jadoon, Abinaya Babu, Laksh Bhatia, Filipe Conceicao (IDE)</p> <p>Péter Szilágyi, Tamás Kárász, Szabolcs Nováczki, Zoltán Vincze, Csaba Vulkán (NOK)</p> <p>Claudio Casetti, Carla Fabiana Chiasserini, Riccardo Rusca (POLITO)</p> <p>Claudio Zunino, Manuel Cheminod, Stefano Vitturi (CNR)</p> <p>José Luis Cárcel, Joaquín Cáceres (ATOS)</p> <p>Deram, Sai Pavan, Pablo Picazo, Carlos J. Bernardos (UC3M)</p> <p>Iacob Crucianu (SIM)</p> <p>Luis M. Contreras, Marta Blanco Caamaño (TID)</p>
Reviewers	<p>Otilia Bularca (SIM)</p> <p>Arturo Azcorra, Antonio de la Oliva (UC3M)</p> <p>Pietro Giardina (NXW)</p>

Abstract

This document reports the final release of the software modules of the AICP. A set of external components that can be connected to the AICP to enhance its functionalities is presented as well. The integration and validation of the different software components is provided. Finally, the implementation view and validation of the operational workflows that were defined in previous deliverables is reported.

Keywords

Control Plane, Cross-domain, AI/ML, Digital Twinning, Time-sensitiveness, Predictability, Reliability, TSN, DetNet, WiFi, 3GPP

Document revision history

Version	Date	Description of change	Contributor(s)
v0.1	16-10-2024	Initial ToC	S. Spadaro
v0.2	21-01-2025	Redefined ToC	S. Spadaro
v1.0	03-03-2025	First round of contributions	WP3
v1.5	11-03-2025	Second round of contributions	WP3
v2.0	15-03-2025	First complete draft	S. Spadaro, F. Agraz
v3.0	24-03-2025	Document ready for internal review	S. Spadaro, F. Agraz
v4.0	27-03-2025	Reviewed version (internal)	SIM, UC3M, NXW
v4.1	31-03-2025	Final version	S. Spadaro, F. Agraz

Disclaimer

The information, documentation and figures available in this deliverable are provided by the PREDICT-6G project's consortium under EC grant agreement **101095890** and do not necessarily reflect the views of the European Commission. The European Commission is not liable for any use that may be made of the information contained herein.

Copyright notice

©PREDICT-6G 2023-2025



Funded by
the European Union

Document information

Nature of the deliverable [OTHER]

Dissemination level

PU Public, fully open. E.g., website



CL Classified information as referred to in Commission Decision 2001/844/EC

SEN Confidential to PREDICT-6G project and Commission Services

* Deliverable types:

R: document, report (excluding periodic and final reports).

DEM: demonstrator, pilot, prototype, plan designs.

DEC: websites, patent filings, press and media actions, videos, etc.

OTHER: software, technical diagrams, etc.

Table of contents

Executive summary	16
1 Introduction	17
1.1 Summary of content from previous deliverables	17
1.2 Software components associated to PREDICT-6G AICP MSs	18
2 AICP Management Services final release	22
2.1 Time Synchronization	22
2.2 Service Ingestion	23
2.3 Exposure services: topology, capabilities and resources	25
2.4 Service Automation	28
2.5 Path Computation	33
2.6 Resource Configurator	35
2.7 Data Collection and management	41
3 External entities	43
3.1 Network Digital Twin for the Ethernet TSN domain	43
3.2 WiFi Digital Twin	48
3.3 E2E Digital Twin	51
3.4 AI/ML-assisted Control Plane	56
4 Operational workflows validation	70
4.1 E2E Exposure and Deterministic Service Provisioning	70
4.2 E2E Service Decommissioning	81
4.3 MLOps – WiFi DT workflow for Federated Learning	86
4.4 E2E Service Maintenance	89
5 Conclusions	97
6 References	98
7 Appendixes	100

7.1	Appendix A – AICP MSs integration tests.....	100
-----	--	-----

List of figures

Figure 2-1. The architecture of the implemented Time Sync components.....	23
Figure 2-2. REST API exposed by the Service Ingestion to handle E2E deterministic service requests.....	24
Figure 2-3. Technological domain exposure mechanism and information models	26
Figure 2-4. Information models exchange in support of E2E Topology Exposure	27
Figure 2-5. E2E Topology Exposure procedure	28
Figure 2-6. E2E Service Automation submodule - Software Design	29
Figure 2-7. MD Service Automation Submodule - Software Design	31
Figure 2-8. Local MD (intra-domain) path computation process	33
Figure 2-9. E2E multi-domain path computation process	34
Figure 2-10. Gating mechanism in a wireless TSN setup	36
Figure 2-11. Workflow of the resource configuration tool.....	37
Figure 2-12 from the IEEE 802.1Qbv standard (Tan et al, 2022). Eth-TSN Resource Configuration follows this approach.....	38
Figure 2-13. TSN software switch architecture	40
Figure 3-1. Network topology.....	44
Figure 3-2. E2E delay and jitter for requested TS flows.....	45
Figure 3-3. Delay vs number of switches traversed by the flow	47
Figure 3-4. Delay vs number of switches without outliers.....	47
Figure 3-5. DT KPI evaluation accuracy	48
Figure 3-6: Implemented Time Aware Shaper with 8 queues and a gate control schedule.....	49
Figure 3-7: Network scenario implemented in the DT: A TS multi-domain network with the Wi-Fi domain leveraging the TWT mechanism.....	50

Figure 3-8: Example of temporal evolution of STA's power states with TWT, over a period of 32 ms.....	51
Figure 3-9: Example of temporal evolution of STA's power states without TWT, over a period of 32 ms.....	51
Figure 3-10. E2E network scenario.....	52
Figure 3-11. E2E vs Domain KPIs. RTT and Throughput are considered.....	54
Figure 3-12. Final MLOps framework SW architecture - Centralized Learning.....	57
Figure 3-13. Final MLOps framework SW architecture - Federated Learning.....	57
Figure 3-14. Kubeflow pipeline code.....	60
Figure 3-15. Pytorch trainer stage.....	60
Figure 3-16. MinIO datasets view.....	61
Figure 3-17. MinIO view showcasing datasets isolation.....	62
Figure 3-18. Flower IA model aggregation process.....	64
Figure 3-19. Training convergence of the RL policy for 10 different training instances in a periodic traffic scenario.....	67
Figure 3-20. Evaluation of trained AI algorithm on a proprietary ns3-based Wi-Fi 7 simulator.	67
Figure 3-21. PRM in support of the E2E Path computation mechanism.....	69
Figure 4-1. Implementation view of the E2E deterministic service provisioning workflow.....	71
Figure 4-2. Forwarding of E2E service provisioning request from E2E Service Ingestion to E2E Service Automation.....	72
Figure 4-3. E2E-PCE log output for domain sequence computation over the abstract topology.....	73
Figure 4-4. Request and response result of the Path Computation process in the MD Service Automation for a MD Service in TSN domain.....	73
Figure 4-5. MD-PCE log output illustrating local path computation and KPI estimation request to the DT.....	74
Figure 4-6. E2E-PCE log output that illustrates the E2E KPI estimation request and reply from the E2E DT.....	74

Figure 4-7. E2E Path Computation response received at the E2E Service Automation during the E2E Service Provisioning.....	75
Figure 4-8. Forwarding of E2E service provisioning request from E2E Service Automation to MD Service Automation.....	75
Figure 4-9. Request result of the Lifecycle Management in the MD Service Automation for the provisioning of a MD Service MD Service in TSN domain, leading to contact the Resource Configurator.....	75
Figure 4-10. Result of the Lifecycle Management in the MD Service Automation during the provisioning phase after receiving the Resource Configurator response in TSN domain.....	75
Figure 4-11. Request of configuration of the monitoring of a technological domain sent by the Service Automation. The monitored KPIs are all the ones for which the service requestor provided a value.....	76
Figure 4-12. Service Exposure information about MD service 1125 provisioned in TSN domain.....	77
Figure 4-13. MD Service Automation response received at the E2E Service Automation.....	77
Figure 4-14. Request of configuration of the monitoring at the E2E MD level sent by the E2E Service Automation. The E2E service encompasses all the local services previously provisioned.....	78
Figure 4-15. Notification of the accomplishment of the provisioning of the service sent by the E2E Service Automation to the E2E Service Ingestion and forwarded to the final user.....	78
Figure 4-16. E2E Service Exposure response with the MD service over the TSN domain associated to the E2E service 33382.....	79
Figure 4-17. E2E Service Exposure response with the E2E service 33382 provisioned over the TSN domain.....	80
Figure 4-18. Implementation view of the E2E deterministic service decommissioning workflow.....	81
Figure 4-19. Forwarding of E2E service decommissioning request from E2E Service Ingestion to E2E Service Automation.....	81
Figure 4-20. E2E Service decommissioning request received and performed at the E2E Service Automation.....	82
Figure 4-21. Request result of the Lifecycle Management in the MD Service Automation for the decommissioning of a MD Service in TSN domain, leading to contact the Resource Configurator.....	82

Figure 4-22. Result of the Lifecycle Management in the MD Service Automation during the decommissioning phase after receiving the Resource Configurator response in TSN domain	82
Figure 4-23. Request of stopping the monitoring of a technological domain sent by the Service Automation. A GET request is performed before the DELETE to retrieve the data source ID coupled to the service ID	82
Figure 4-24. E2E Service decommissioning response received at the E2E Service Automation from the MD Service Automation including E2E Monitoring stop and E2E Service Decommissioning response sent to the E2E Service Ingestion	83
Figure 4-25. Request of stopping the monitoring at the E2E MD level sent by the E2E Service Automation. A GET request is performed before the DELETE in order to retrieve the data source ID coupled to the E2E service ID	83
Figure 4-26. E2E Service Exposure shows E2E service 33382 as decommissioned in TSN domain	84
Figure 4-27. Notification of the accomplishment of the decommissioning of the service sent by the E2E Service Automation to the E2E Service Ingestion and forwarded to the final user.	85
Figure 4-28. Isolated Dataset Storage in MinIO	86
Figure 4-29. Model Aggregation in Kubeflow Pipeline for Client 3	87
Figure 4-30. Federated Learning model aggregation result	88
Figure 4-31. Federated Learning aggregated model in Model Storage (MinIO)	88
Figure 4-32. Federated Learning model inference performed	89
Figure 4-33: Holistic system architecture for Closed-Loop Automation of localisation and sensing use case	90
Figure 4-34. Experimental setup	91
Figure 4-35. Experiment 1, interfering traffic	93
Figure 4-36. Experiment 2, dynamic re-configuration	94
Figure 4-37. Experiment 3 Time Aware Shaper on the TSW 1 and zoom for a comparison with other experiments	95
Figure 4-38. Experiment 4, interfering traffic mapped to high priority and TAPRIO	96
Figure 7-1. E2E and MD Service Automation Architecture - Tests and External Modules	110

Figure 7-2. E2E and MD Service Automation Architecture - Service decommissioning tests.120

List of tables

Table 1-1. PREDICT-6G AICP - List of software components	21
Table 2-1.REST interface schema for the Resource Configurator.....	35
Table 3-1. TSN flows characteristics for the considered applications.....	44
Table 4-1. Setup latencies	92

Acronyms and definitions

AI	Artificial Intelligence
AICP	AI-driven Multi-stakeholder Inter-domain Control-Plane
AMF	Access and Mobility Management Function
AP	Access Point
API	Application Programming Interface
CCA	Clear Channel Assessment
CLA	Closed Loop Automation
CRUD	Create Read Update Delete
CSMA/CA	Carrier Sense Multiple Access/Collision Avoidance
DetNet	Deterministic Networking
DoA	Description of Action
DT	Digital Twin / Digital Twinning
DUG	Data Unit Group
E2E	End-to-End
IIoT	Industrial Internet of Things
IETF	The Internet Engineering Task Force
KPI	Key Performance Indicator
MDP	Multi-domain Data-Plane

MIMO	Multiple-Input and Multiple-Output
ML	Machine Learning
MF	Management Function
MS	Management Service
MLOps	Machine Learning Model Operationalization
NBI	North-Bound Interface
NDT	Network Digital Twin
OFDMA	Orthogonal Frequency Division Multiple Access
PCE	Path Computation Element
POP	Pipeline Orchestration Platform
PREOF	Packet Replication, Elimination, and Ordering Functions
PRM	Path Redundancy Mechanism
QoS	Quality of Service
REST	Representational State Transfer
RTT	Round-Trip Time
RU	Resource Unit
SA	Service Automation
SBI	South-Bound Interface
SE	Service Exposure
SP	Service Period

STA	Station
TWT	Target Wake Time
TSN	Time-Sensitive Networking
TAS	Time-Aware Shaper
TDD	Time Division Duplex
UDP	User Datagram Protocol
WCTT	Worst-Case Transversal Time
WP	Work Package

Table of partners

Short Name	Partner
UC3M	<u>Universidad Carlos III de Madrid</u>
NOK	<u>Nokia Solutions and Networks KFT</u>
ERC	<u>Ericsson Espana SA</u>
INT	<u>Intel Deutschland GMBH</u>
TID	<u>Telefónica Innovación Digital SA</u>
ATOS	<u>ATOS IT Solutions and Services Iberia SL</u>
GES	<u>Gestamp Servicios SA</u>
NXW	<u>Nextworks</u>
COG	<u>Cognitive Innovations Private Company</u>
SIM	<u>Software Imagination & Vision SRL</u>
POLITO	<u>Politecnico di Torino</u>
UPC	<u>Universitat Politecnica de Catalunya</u>
CNR	<u>Consiglio Nazionale delle Ricerche</u>
UNIPD	<u>Universita degli Studi di Padova</u>
IDE	<u>Interdigital Europe LTD</u>

Executive summary

This document reports the final version of the AI-driven multi-stakeholder Inter-domain Control Plane (AICP) of the PREDICT-6G project. To this end, the document begins with a brief recap on the AICP architecture and functionalities. Afterwards, the updates, final description and implementation notes of the software components, which implement the Management Services (MSs) that have been defined for the AICP are provided. The report describes next the implementation and validation of a set of software components that, once connected to the AICP through its MLOps framework, provide AI-based enhancements in support of deterministic service provisioning and maintenance. Finally, the document revisits the operational workflows that have been defined throughout WP3 to provide their implementation view and experimental validation.

For the sake of completeness, a set of integration tests that have been conducted for the experimental validation of the described MSs have been placed in the appendixes section.

The key innovations presented in this document are the following:

- Implementation and experimental integration of the MSs defined for the AICP in support of deterministic service provisioning and maintenance.
- Implementation and functional validation of digital twinning solutions to estimate the Key Performance Indicators (KPIs) associated to deterministic services.
- Implementation and functional validation of an MLOps framework in support of AI-based deterministic service provisioning and maintenance.

1 Introduction

This deliverable consolidates the implementation of the first functional release of the AI-driven multi-stakeholder Inter-domain Control Plane (AICP) of the PREDICT-6G project, whose architecture was defined in (PREDICT-6G/D3.1, 2023). In this regard, the document reports the updates and final design, implementation and functional integration of the software modules that support the AICP MSs operation. Such modules' preliminary version was detailed in (PREDICT-6G/D3.2, 2023). The deliverable is structured as follows: the current section (section 1) contextualizes the deliverable in the framework of the project. To do this, a summary of the previous deliverables that have some impact on (or are impacted by) this one is provided first. Next, a list and summary of the software components that implement the MSs associated to the AICP is given. Section 2 elaborates on the updates and final design of the software components previously listed. Section 3 describes a set of software components and applications that aim at extending or enhancing the AICP capabilities. Section 4 provides the experimental integration of the AICP MSs by exercising the operational workflows, which were defined in (PREDICT-6G/D3.2, 2023) and refined in (PREDICT-6G/D3.3, 2025). Section 5 concludes the document and, finally, extensive integration testing of the AICP MSs is provided in Appendix 7.1 for further reference.

1.1 Summary of content from previous deliverables

In order to provide some context that facilitates the understanding of the current document without over-extending it, this section summarizes the work that has been previously reported regarding the AICP.

Deliverable D3.1 (PREDICT-6G/D3.1, 2023) defined the AICP architecture. The AICP follows a service-oriented approach where a set of MSs were defined to support the deterministic service provisioning, maintenance and decommissioning. Aiming to implement cross-domain multi-technology support, the AICP was designed in two levels, namely local management domain (MD) and end-to-end (E2E). Hence, the local-level AICP would be instantiated over different technological management domains (MDs), and the E2E-level AICP (E2E) would be instantiated over the MDs AICP instances to provide E2E support. (PREDICT-6G/D3.1, 2023) also defined the MSs required to implement the AICP functionalities at both levels.

Deliverable D3.2 (PREDICT-6G/D3.2, 2023) provided the functional design and preliminary implementation of the software modules that would implement the MSs defined in (PREDICT-6G/D3.1, 2023). More specifically, the software components aimed to implement the time synchronization, and the deterministic service ingestion, computation, configuration and management were listed and described. In addition, the report defined the preliminary version

of the operational workflows that described the interactions between the different MSs to provide the deterministic service provisioning, maintenance and decommissioning functionalities. The development methodology and roadmap were also defined in the last sections of (PREDICT-6G/D3.2, 2023).

Deliverable D3.3 (PREDICT-6G/D3.3, 2025) provided an update of the AICP architecture where the MSs were classified into core capabilities of the AICP and specific ones. A revised version of the operational workflow in accordance with the new architecture was reported as well.

In this context, this deliverable extends (PREDICT-6G/D3.2, 2023) by providing a description of the final release of the software components and applications that implement the AICP functionalities, and an implementation perspective of the operational workflows. Both elements (i.e., the software components and the operational workflows) are experimentally validated.

1.2 Software components associated to PREDICT-6G AICP MSs

Table 1-1 summarizes the software components that have been developed and integrated to compose the MSs of the AICP.

AICP module	Short description	License	Link in the repository
Time Synch	Time Synchronization module has the responsibility to setup and maintain time synchronization consistently in the PREDICT-6G system possibly spanning over multiple technology domains.		The code of the implementation is not public but available to reviewers on a request (NOK contact: Péter Szilágyi, peter.1.szilagyi@nokia-bell-labs.com).
Service Ingestion	AICP module responsible for validating requests for E2E	APACHE2.0	https://gitlab.netcom.it.uc3m.es/predict-6g/aicp/service-ingestion

	deterministic services		
Service Automation	AICP module designed to ensure the correct deployment and maintenance of deterministic E2E and MD services.		The code of the implementation is not public but available to reviewers on a request (EVIDEN contact: José Luis Cárcel, jose.carcel@eviden.com).
Path Computation (and Exposure)	AICP module responsible for computing the paths for both E2E and local domain case. It also implements abstract topology exposure functionalities.	APACHE 2.0	https://gitlab.netcom.it.uc3m.es/predict-6g/aicp/path-computation
Resource Configurator (and Exposure)	AICP module responsible for configuring the devices of a given domain. It also implements topology, capabilities and resource exposure functionalities.		<p>Ethernet-TSN: https://gitlab.netcom.it.uc3m.es/predict-6g/aicp/resource-configurator</p> <p>Wi-Fi: The code of the implementation is not public but available to reviewers on a request (INTEL contact: hamza.chahed@intel.com)</p> <p>3GPP: The Code of the implementation is not public but available to reviewers on a request (ERICSSON contact: miguel.angel.lopez.serrano@ericsson.com)</p>
Data Collection and Management	AICP module responsible for gathering monitoring data from MDP to	APACHE 2.0	https://gitlab.netcom.it.uc3m.es/predict-6g/aicp/monitoring-and-data-collection

	provide to analytic processes based on AI/ML and DT		
TSN Network Digital Twin and E2E Digital Twin	API exposing the DT capabilities for the TSN and E2E domains	APACHE 2.0	https://gitlab.netcom.it.uc3m.es/predict-6g/aicp/upc-dt
WiFi DT	DT of a TSN network for realistic simulations of IEEE 802.11ax with TWT feature		The code of the implementation is not public but available to reviewers on a request (POLITO contact: Carla Fabiana Chiasserini, carla.chiasserini@polito.it).
TSN Federated Learning Scheduling	A Federated Learning framework for training a neural network model to predict the optimal scheduling policy for TSN traffic in an IEEE 802.11ax network with TWT enabled	APACHE 2.0	https://gitlab.netcom.it.uc3m.es/predict-6g/polito-tsn-fl-scheduling
Path Redundancy Mechanism	PRM service for dynamically computing redundant the paths for E2E based on the current network load	APACHE 2.0	The code of the implementation is not public but available to reviewers on a request (SIMAVI contact: Iacob Crucianu at iacob.Crucianu@simavi.ro)
AI/ML algorithmic frameworks	AI-based Slicing algorithm	APACHE 2.0	https://gitlab.netcom.it.uc3m.es/predict-6g/AI-based_Wi-Fi_Slicing

MLOps Framework	Module to offer AI-as-a-Service capabilities to the AICP, supporting the design, training and serving of AI/ML models.		The code of the implementation is not public but available to reviewers on a request (EVIDEN contact: José Luis Cárcel, jose.carcel@eviden.com).
-----------------	--	--	--

Table 1-1. PREDICT-6G AICP - List of software components

2 AICP Management Services final release

This chapter reports on the final release and updates of the software modules that implement the Managements Services (MSs) of the AICP. It is worth noting here that all the software modules and the functionalities implemented to support the AICP operation have been tested and validated. For the sake of completeness, the reporting of the conducted extensive integration tests can be found in the Appendix (section 7).

2.1 Time Synchronization

Time Synchronization (TS) module in the AICP has the responsibility to setup and maintain time synchronization consistently in the PREDICT-6G system possibly spanning over multiple technology domains. Sections 7.2, 8.2 and 9.4 of (PREDICT-6G/D1.2, 2023) and section 7.2 of (PREDICT-6G/D3.1, 2023) defined the desired architecture and operation of the TS module whereas section 4.1 of (PREDICT-6G/D3.2, 2023) reported on the software design of the TS components. Moreover, section 6 of (PREDICT-6G/D2.3, 2025) specified OpenAPIs related to the time synch operation. The implementation of the time synch module followed the principles and specifications set forth in the aforementioned documents. In line with the AICP design, the TS AICP module was implemented in a two-level hierarchical architecture where the lower-level TS MS module is in charge of the technology domain specific time synchronization management and the upper level E2E TS Management MS module is responsible for the cross-domain time synchronization operation consistency. The implemented time synchronization software module architecture is shown in Figure 2-1.

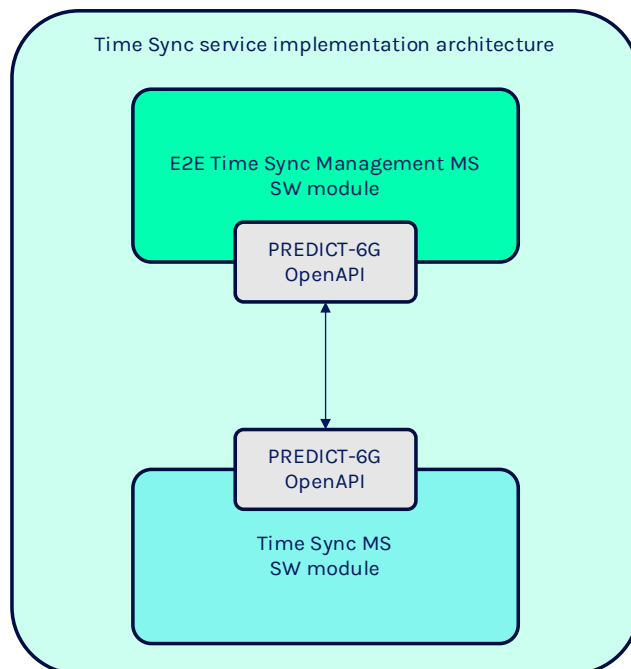


Figure 2-1. The architecture of the implemented Time Sync components

The TS MS and E2E TS Management MS modules were implemented in Python language using the OpenAPI specifications defined in section 6 of (PREDICT-6G/D2.3, 2025) and tested in the Nokia Open Lab (see section 2.1 of (PREDICT-6G/D4.2, 2024)).

The validation of this MS consisted of evaluating whether the E2E Time Sync Management MS was able to collect and evaluate the time sync configuration in the PREDICT-6G system. In the proposed tests, the E2E TS Management MS was querying each TS MS of the technology domains for their respective configuration using the OpenAPI specified in (PREDICT-6G/D2.3, 2025). Upon the query from the E2E TS Management MS, the TS MS collected the required information and sent the collected information back to the E2E TS Management MS. At the end of the test, the E2E TS MS received all the requested information and successfully completed the configuration collection task. The specification of the tests can be found in Appendix 7.1.

2.2 Service Ingestion

The Service Ingestion (SI) module is the entry point to the AICP for external users. As described in section 4.2 of (PREDICT-6G/D3.2, 2023), the component is designed to validate the E2E deterministic service provisioning and decommissioning requests by authenticating the requestor through an Identity and Access Management (IAM) Platform and by checking if the

request is compliant with the AICP information model defined in section 8 of (PREDICT-6G/D3.2, 2023).

The SI resides in the E2E level of the AICP and interacts with the E2E Service Automation to whom forwards the service request for the service lifecycle management.

The E2E SI module has been implemented according to the software design provided in (PREDICT-6G/D3.2, 2023). The following paragraph briefly recalls the component functionalities and interfaces described in section 4.2.1 of (PREDICT-6G/D3.2, 2023).

- The **Pre-Validation logic** pre-processes the received service request by extracting the two sub-requests to be managed by authentication logic and request parsing logic, respectively. It also stores the request status in the Request Status Register and notifies the requestor about the outcome of the request.
- The **Authentication Logic** validates the token provided by the requestor by leveraging on an external and centralized IAM. The token is obtained from the IAM by exchanging a set of credentials and contains user info, so the validation is performed by checking user role and scope.
- The **Request Parsing Logic** checks the correctness of the data structure representing the E2E service model.
- The **Request Status Register** keeps track of the status of each service request in the following format:
<Request_ID>, <Request_type>, <Requestor_ID>, <Status>, <Information>
- The **Clients** component represents the South-Bound Interface (SBI) towards the IAM and the E2E Service Automation.

Figure 2-2 shows the implemented APIs exposed by the Service Ingestion NBI.

ingestion Handle lifecycle management requests for E2E deterministic services (Provisioning, Decommissioning)		^
GET	/service Request the status of a given request	▼
POST	/service Request the provision of an E2E service with certain characteristics	▼
DELETE	/service/{id} Request the decommissioning of an E2E service	▼
PUT	/service/{id} Request a modification of an existing E2E service	▼

Figure 2-2. REST API exposed by the Service Ingestion to handle E2E deterministic service requests

Regarding the code, tools and framework adopted for the implementation, the Service Ingestion NBI consists in a **Flask**¹ server with the logic developed in Python. The IAM platform

¹ <https://flask.palletsprojects.com/en/stable/>

has been implemented with **Keycloak**² and the Request Status Register with **InfluxDB**³ as a timeseries DB. All the sub-modules (server, IAM platform and timeseries DB) are containerized with Docker. The SI MS interacts with the E2E Service Automation MS. The detail of the validation of the SI MS integration is provided in Appendix 7.1.

2.3 Exposure services: topology, capabilities and resources

The AICP architecture considers two levels of management, namely the technological domain and the end-to-end (E2E) (PREDICT-6G/D3.1, 2023). On the one hand, the exposure of the topology, capabilities and resources is tackled, assuming such management levels, as well. Hence, at the technological domain level, the exposure services aim at collecting the specific domain's data plane information that is needed for intra-domain service computation, provisioning and maintenance. On the other hand, the E2E exposure services aim at collecting abstracted information of the different technological domains to compute, configure and manage multi-domain E2E services.

This section presents the design and implementation of the exposure services at both technological domain and E2E levels. The interactions reported in this section have been tested within the integration work that is reported in Appendix 7.1.

MD Exposure Services

The role of the MD exposure is two-fold. First, it is responsible to collect the data plane information to be used by other MD services such as the Path Computation (MD-PCE). Second, the MD exposure is responsible for exposing abstracted topological information about the underlying technological domain to the E2E level for multi-domain operation. In particular, the E2E Topology Exposure collects abstracted information from the MD services under its responsibility to compose the topology that will be used at E2E level to compute, provision and maintain multi-domain services.

As explained in (PREDICT-6G/D3.2, 2023), the MD exposure services, which were initially split into different modules, have been grouped into one that embraces the exposure of the topology, capabilities and resources of the specific technological domain in a unified way. Additionally, to homogenize the exposure across the different domains, a single data model that exploits the commonalities of the heterogeneous data plane technologies composing the PREDICT-6G's MDP (PREDICT-6G/D2.1, 2023) was defined in (PREDICT-6G/D2.3, 2025) and its implementation is reported in (PREDICT-6G/D2.4, 2025).

² <https://www.keycloak.org>

³ <https://www.influxdata.com>

Hence, the MD exposure service module residing on top of each technological domain is responsible of collecting the data plane information related to the network elements and their capabilities (specifically the ones related to the time sensitiveness and predictability, such as FRER, PREOF, etc.), and the interconnection (i.e., links) between them, again enriched with specific information such as latency that enables the AICP TSN operation. This information, which is formatted following the common information model, is used by other MD services, such as the MD Path Computation. As said at the beginning of the section, the MD exposure is also in charge of abstracting the data plane information to compose the topology that will be used at the E2E level for multi-domain/multi-technology service computation. To do this, a common model has been defined and implemented based on IETF standardization to expose the abstracted topology of the different technological domains.

MD Exposure implementation considerations

Although the PREDICT-6G AICP architecture depicts the Exposure MS as a single separated module, in the current implementation of the AICP, the functionalities provided by this MS have been split across different MSs.

In this way, the responsibility of collecting the data plane topological, resource and capabilities information has been assumed by the Resource Configurator MS of each technological domain, which expose it in the format provided by the MDP Open API to the other MSs of the same MD, as shown in Figure 2-3.

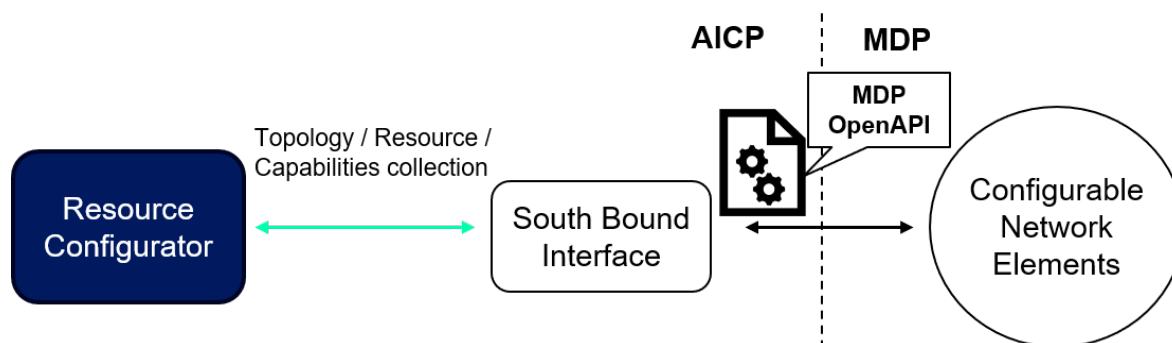


Figure 2-3. Technological domain exposure mechanism and information models

One of the consumers of this information model is the MD Path Computation, which uses this information to compute intra-domain paths. In addition, the MD Path Computation implements the second functionality of the Exposure MS, which is related to the abstraction of the data plane topology to be exposed to the E2E components of the AICP. Hence, the MD Path Computation computes a set of routes between the border nodes of the domain and abstracts them as network slices (RFC9543, 2024) that will be exposed to the E2E MS (e.g., the

E2E Path Computation) for multi-domain service computation. This operation is illustrated in Figure 2-4.



Figure 2-4. Information models exchange in support of E2E Topology Exposure

E2E Topology Exposure Service

The E2E Topology Exposure MS is responsible for collecting the abstracted topologies exposed by the MD Exposure service of each technological domain to compose a topology that provides enough detail from the different domains to enable an informed E2E service computation, which maximizes the guarantees that the KPIs posed by the request will be satisfied. Hence, the E2E Topology Exposure contacts the MD Exposure service module of the different underlying domains to collect the corresponding abstract topology model and composes the complete abstract view of the E2E. The complete E2E model can be used by the E2E AICP components (such as the E2E Path Computation) for their operation.

E2E Topology Exposure implementation considerations

Following a similar approach as the MD Exposure MS, the E2E exposure functionality has been implemented in the E2E Path Computation, since, in the current implementation, it is the only entity that consumes the abstracted topology from the different MDs under its control. In this way, as depicted in Figure 2-5, the E2E Path Computation collects the abstract topologies from its controlled MDs by contacting the corresponding MS Path Computation Elements. With this information, the E2E Path Computation composes the complete abstract multi-domain topology. Such topology is used to compute the domain sequence to be followed by the E2E service. It is worth noting here that the network slicing approach that supports the abstraction of the multi-domain, multi-technology data plane allows to maximize the chances that the computed paths satisfy the requirements posed by the service request, thus offloading the different DTs of the architecture, reducing the control overhead, and speeding up the service configuration process.

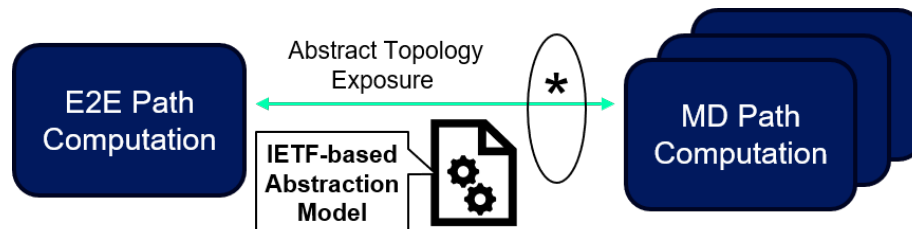


Figure 2-5. E2E Topology Exposure procedure

IETF-based Abstraction Model

As said in previous sub-sections, the key elements of the exposure services are the models used to homogenize and abstract the heterogeneous data plane technologies and components that compose the PREDICT-6G MDP. While the model used to expose the specific data plane technologies into the MD level of the AICP and its associated Open API definition are described in (PREDICT-6G/D2.3, 2025) and (PREDICT-6G/D2.4, 2025), this sub-section details the model that has been defined in the project to abstract the topologies of the different MDs to compose the E2E one. Aiming at standards compliance, the E2E abstraction model has been defined based on the IETF proposals for topological and service information exchange between different management entities. In this line, the abstraction strategy follows the network slicing concept (RFC9543, 2024). More specifically, the technological domains are abstracted as a set of border nodes interconnected by different slices that fulfil a set of performance requirements (i.e., KPIs). The border nodes are presented to the E2E layer as Provisioning Edges (PEs) and the exposed ports are translated into Service Demarcation Points (SDPs, (RFC9543, 2024)). The intra-domain connectivity (i.e., the network slices provisioned between borders) is abstracted in the form of Connectivity Constructs (CCs, (RFC9543, 2024)). The KPIs offered by each intra-domain connectivity element (i.e., CC) are modelled in the form of Service Level Objectives/Expectations (SLOs/SLEs, (RFC9543, 2024)). Finally, the inter-domain connectivity existing between the PEs of different domains is modelled following the Attachment Circuit (AC, (Boucadair, 2025)) concept.

2.4 Service Automation

As described in section 4.4 of (PREDICT-6G/D3.2, 2023), the Service Automation (SA) module is the AICP component designed to ensure the correct deployment and maintenance of deterministic E2E and MD services in multiple heterogeneous infrastructures. The module manages all stages of the service lifecycle through a closed-loop implementation that controls and automates the service provisioning, assurance and termination.

The SA module implementation encompasses four specific Management Services: (i) the E2E Service Automation, which oversees the implementation and management of an E2E service

across all technology domains; (ii) the MD Service Automation, which interacts with each technology domain to ensure the closed-loop automation of a MD service; (iii) the E2E Service Exposure (SE), which exposes E2E service information to other MSs and/or users; and (iv) the MD Service Exposure, which exposes MD service information in a similar manner.

For the implementation, as stated in (PREDICT-6G/D3.2, 2023), the SA module has been coherently developed following a two-level architecture with two submodules: E2E SA submodule, which includes E2E SA and E2E SE capabilities, and the MD SA submodule, which implements MD SA and MD SE features.

Based on the proposed design, next subsections update the description of the software implementation provided in (PREDICT-6G/D3.2, 2023), considering E2E and MD SA submodules and their interactions with the rest of AICP.

E2E Service Automation

As described in section 4.4.1.1 of (PREDICT-6G/D3.2, 2023), the E2E SA interacts with other MSs to manage the lifecycle of E2E services during the provisioning and decommissioning phases.

To interact with the rest of the modules, the E2E SA has been designed following a modular architecture, where several functions have been defined to cover specific functionalities. Figure 2-6 shows the final E2E SA submodule design, which remains unchanged with respect to the one described in (PREDICT-6G/D3.2, 2023).

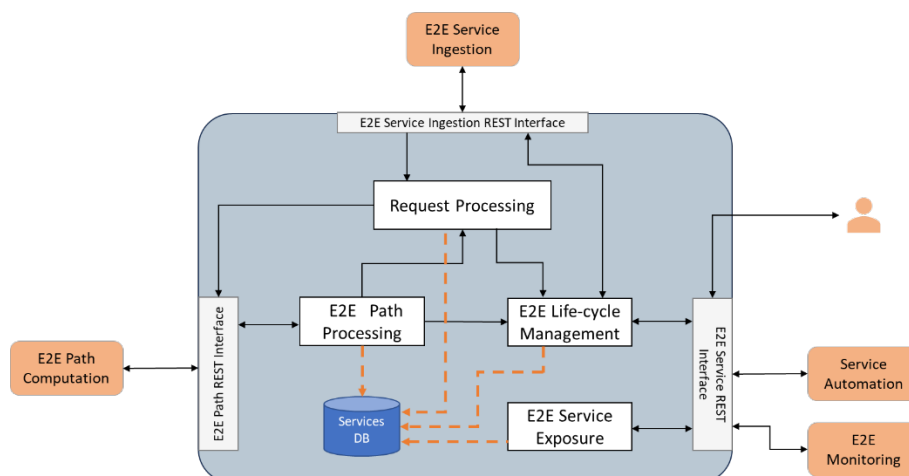


Figure 2-6. E2E Service Automation submodule - Software Design

As there has not been any modification performed with respect to the initial design presented in section 4.4.1.1 of (PREDICT-6G/D3.2, 2023), the mission of each function is briefly described below:

- The **E2E Request Processing SW Function** processes the E2E Service Provisioning or Decommissioning requests coming from the E2E Service Ingestion and forwards them to the E2E Path Processing or to the E2E Lifecycle Management functions in provisioning and decommissioning phases, respectively. Additionally, this function is also able to store or retrieve E2E service information from the Services DB when requests are received.
- The **E2E Path Processing SW Function** manages and processes all information related to the path allocation of E2E services. Forwards and processes the E2E path computation data via the E2E path interface, forwards the E2E paths and domain information for its storage in the Services DB, and triggers the E2E Service lifecycle management during the provisioning.
- The **E2E Lifecycle Management SW Function** manages the lifecycle of the E2E services. As part of this process, initializes and enables the service provisioning and decommissioning in MDs, notifies E2E AI and DT about the service status, requests the monitoring of the E2E Service, and forwards, requests and updates E2E service information by interacting with Services DB.
- The **E2E Service Exposure SW Function** exposes information of the E2E services after receiving requests through the E2E service interface and querying the Services DB.
- The **Services DB** stores all the information related to E2E services including all characteristics shown in the E2E Service Data Model (see section 8.3 of (PREDICT-6G/D3.2, 2023)) and the link between E2E services, paths and selected domains. This database is also used by the MD SA submodule to store information related to local services.
- The **E2E Service Ingestion REST Interface** is used to interact with the E2E SI module when receiving Service Provisioning or Decommissioning requests from the E2E SI or when forwarding the outcome, i.e., confirmation or rejection, for those requests.
- The **E2E Path Computation REST Interface** is designed to interact with the E2E Path Computation module for forwarding E2E path computation requests and to receive path and domain selection information.
- The **E2E Service REST Interface** is designed to interact with the MD SA, E2E AI, E2E DT and E2E Monitoring modules. It is used to start the service provisioning and decommissioning in local domains, receive the outcome of those processes, notify about the status of E2E services to E2E AI and DT MS, request the monitoring of a specific E2E service, and expose information related to E2E services to users, operators or other external MSs.

MD Service Automation

As described in section 4.4.1.2 of (PREDICT-6G/D3.2, 2023), the MD SA interacts with other MSs to manage the lifecycle of local MD services during the service provisioning and decommissioning phases.

To enable such interaction, the MD SA has been designed following the modular architecture depicted in Figure 2-7. Note that the final design has slightly changed with respect to the one described in (PREDICT-6G/D3.2, 2023), since the Resource Exposure interaction is now modelled as part of the Resource Configuration MS.

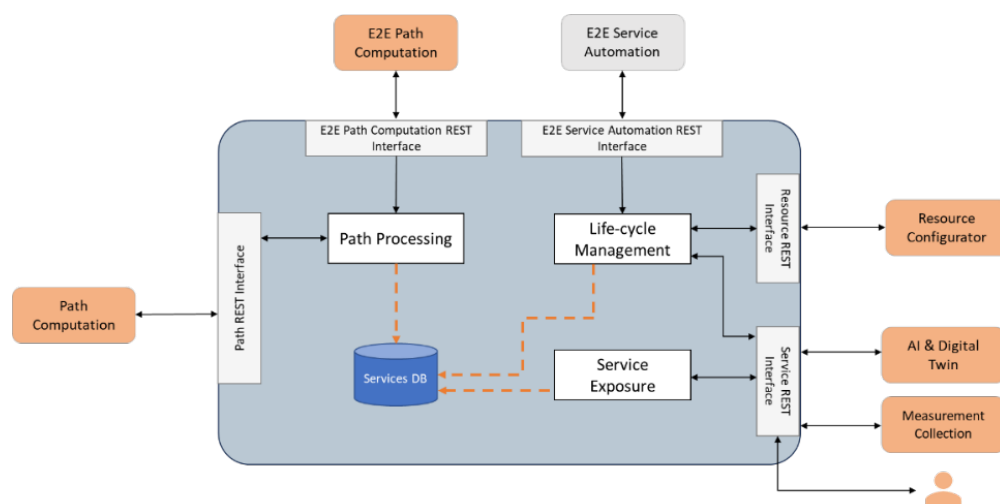


Figure 2-7. MD Service Automation Submodule - Software Design

The role of each of the proposed components is the following:

- The **Path Processing SW Function** is designed to manage and process all information related to path allocation in the MD. This function receives the path computation information received from the E2E Path Computation module through the E2E path interface. With this data, the component forwards a path request towards the MD Path Computation to compute the path in the MD. After the path is computed, it receives a path computation response, which uses to update the MD service stored in the Services DB. After that, it triggers the service lifecycle management process. This function is only used during the service provisioning phase.
- The **Lifecycle Management SW Function** is designed to manage the service lifecycle at MD level, after the E2E domain is split into several MDs. This function is responsible for establishing both service provisioning and decommissioning in the local domains when a MD service provisioning or decommissioning request arrives from the E2E SA.

The provisioning or decommissioning will be performed by contacting the Resource Configurator which will allocate/release the resources in the MD, notifying AI and DT about the service status, and trigger or stop the measurement collection for the MD service. To this aim the function will use two interfaces: one for resource related requests (Resource REST interface) and one for the remaining interactions (Service REST interface).

- The **Service Exposure SW Function** is designed to expose information of the MD services. This function is designed to receive exposure requests through the Service Interface, request the information of the MD services to the Services DB, and send it back via the Service Interface.
- The **Services DB** is used to store all information related to MD Services. It can be accessed by the path processing, service lifecycle and service exposure functions.
- The **E2E Path Computation REST Interface** is designed to interact with the E2E Path Computation module to retrieve path computation requests and forwarding path allocation notifications.
- The **E2E Service Automation REST Interface** is designed to interact with the E2E SA module to receive Service Provisioning or Decommissioning requests and to forward service provisioning/decommissioning notifications.
- The **Service REST Interface** is designed to notify the status of the MD service to AI and DT MSs, start/stop the Measurement Collection of a MD Service, and to expose information related to MD services.
- The **Resource REST Interface** is designed to interact with the Resource Configurator MS to allocate service resources in the corresponding technology domain.
- The **Path REST Interface** is designed to interact with the MD Path Computation module to retrieve path computation requests, as well as forwarding path allocation notifications.

From an integration perspective, the E2E SA interacts with the SI, Path Computation and Monitoring MSs at the E2E AICP level, and with the MD SA of each underlying technological domain. In turn, the MD SA interacts with the E2E SA and the E2E Path Computation of the E2E-level AICP, and with the Path Computation, Resource Configuration and Measurement Collection (i.e., Monitoring) MSs at the technological MD level. All these interactions have been tested and the specific integration tests are reported in Appendix 7.1.

2.5 Path Computation

As reported in previous deliverables, the PREDICT-6G AICP architecture defines two levels of path computation and, thus, two path computation components. First, the local MD Path Computation component (MD-PCE) is responsible for the intra-domain path computation, that is, the computation of paths inside a technological domain (or Managed Domain, MD). Second, the architecture defines an E2E Path Computation component, the E2E-PCE, which is responsible for computing multi-domain paths over an abstracted view of the multi-domain multi-technology scenario. This section provides the implementation details and validation of such modules as well as a description of the updates realized over such components from the previous deliverables.

MD Path Computation

The implementation of the MD-PCE MS has followed the design reported in (PREDICT-6G/D3.2, 2023). Nonetheless, as described in section 2.3, some functionalities have been added to the module with regard to the exposure of the topology of the technological domain and its further abstraction. In addition, as reported in (PREDICT-6G/D3.3, 2025), the service computation at the MD level was modified and, for some implementations of the MD such as the TSN domain, the MD-PCE might take the responsibility to request the KPI estimation to the DT at the MD (if available) and to validate it prior to request the provisioning to the Service Automation. For the sake of generality, the MD-PCE currently implements the interfacing with the DT of the MD. Such an interface can be activated or deactivated by configuration in the MD-PCE depending on the availability of a DT in the domain. Figure 2-8 illustrates the current implementation of the intra-domain path computation process as part of the E2E service provisioning. The integration tests detailed in Appendix 7.1 include the interactions depicted in the figure.

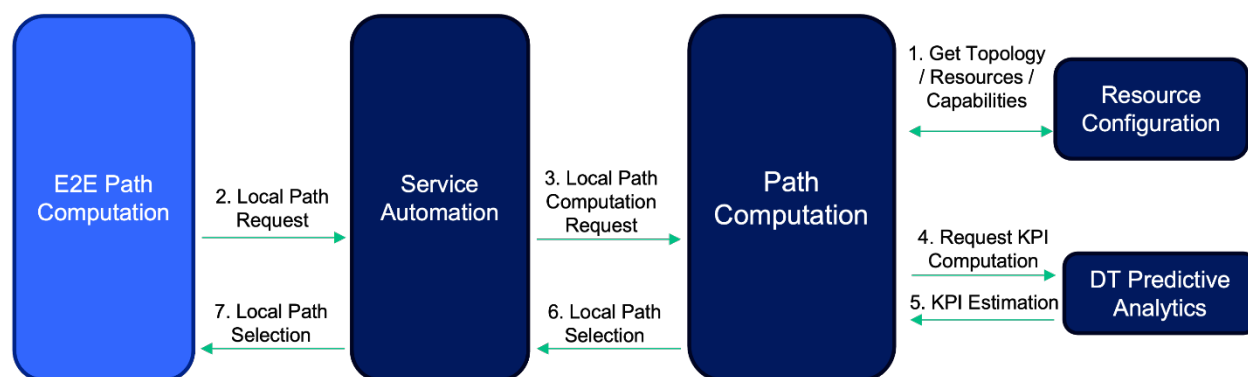


Figure 2-8. Local MD (intra-domain) path computation process

Figure 2-8 also depicts the interaction between the MD-PCE and the Resource Configuration service to collect the technological domain topological information, including resources and capabilities (step 1) and the interaction with the DT of the MD (if available) in steps 4 and 5.

E2E Path Computation

The implementation of the E2E-PCE has also followed the design reported in (PREDICT-6G/D3.2, 2023). In line with the modifications implemented to the MD-PCE regarding the exposure services, the E2E-PCE contacts the MD-PCEs of the underlying domains to obtain the abstracted local topologies and composes the E2E abstract view. Hence, it can be stated that the E2E-PCE implements the E2E Topology Exposure functionalities. Figure 2-9 shows the E2E path computation process considering the current implementation of the different MSs.

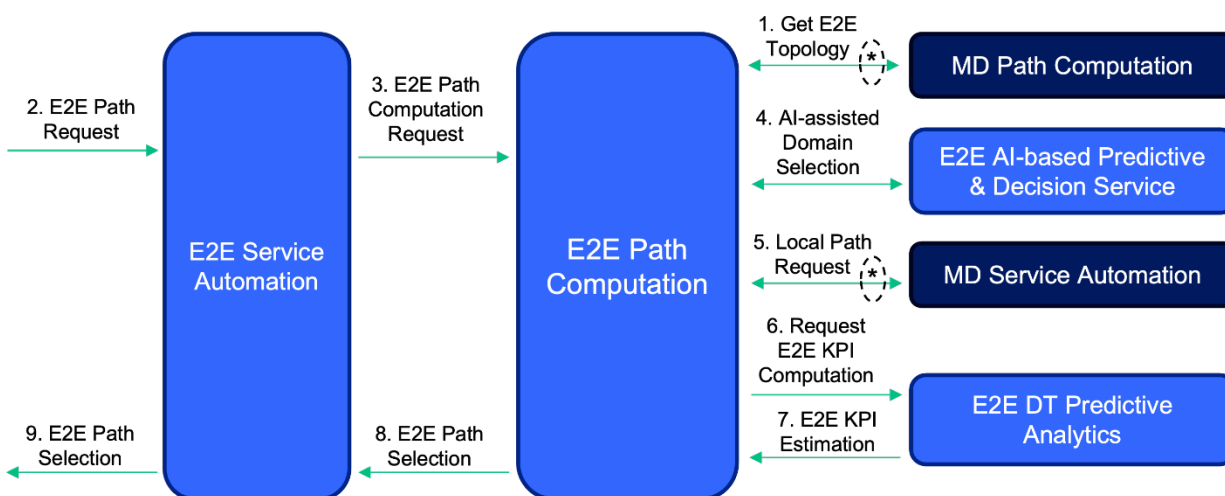


Figure 2-9. E2E multi-domain path computation process

As depicted in Figure 2-9, the E2E-PCE contacts the MD-PCE of each underlying technological domain to compose the E2E abstract topology. The rest of the process is the same as reported in (PREDICT-6G/D3.2, 2023). As in the case of the MD-PCE, the integration tests associated to the interactions depicted in the figure are detailed in Appendix 7.1.

2.6 Resource Configurator

This section reports the implementation of the Resource Configurator (RC). Three implementations corresponding to the three domains considered in the project are presented. All three follow the REST interface schema defined in Table 2-1.

Interface	Name	HTTP Method	Endpoint	Input/Output
Resource Configurator REST Interface	Resource Configurator Provisioning	POST	/save-request	I: Service Provisioning Request O: HTTP Code 200 OK with the resources configured. 500 Error if there is a mismatch with the flow model.
	Resource Configurator Decommissioning	DELETE	/delete-request	I: Service Decommissioning Request O: HTTP Code 200 OK with the resources field void. 404 Error if the service ID is not found.

Table 2-1. REST interface schema for the Resource Configurator

The integration of the three instances of the RC with their respective MD SA MS and data plane, can be found in Appendix 7.1.

Wi-Fi Resource Configurator

The wireless communication channel is inherently shared, allowing multiple stations to access the medium simultaneously. To avoid data transmission collisions, the classical approach employs the CSMA/CA protocol, which senses the channel before transmitting, sends data only when the channel is free, and retransmits after a random back-off period in case of a collision. However, this strategy is unsuitable for traffic with hard deadlines, as random back-off times can violate maximum latency bounds.

The TSN approach schedules medium access deterministically. Each station or access point (AP) transmitting critical traffic is allocated a specific time slot, during which only it can send critical traffic, while all other stations/APs are prohibited from transmitting any traffic. The transmitting station/AP must also cease non-critical transmissions during its slot. This behaviour is enforced using the gating mechanism proposed in IEEE 802.1Qbv (Serna, 2018), as shown in Figure 2-10. This mechanism places a gate at each queue of a Wi-Fi controller, allowing transmission only when the respective gate is open. By mapping traffic classes to different queues and controlling gate states via a schedule, the transmission slots for each traffic class were managed.

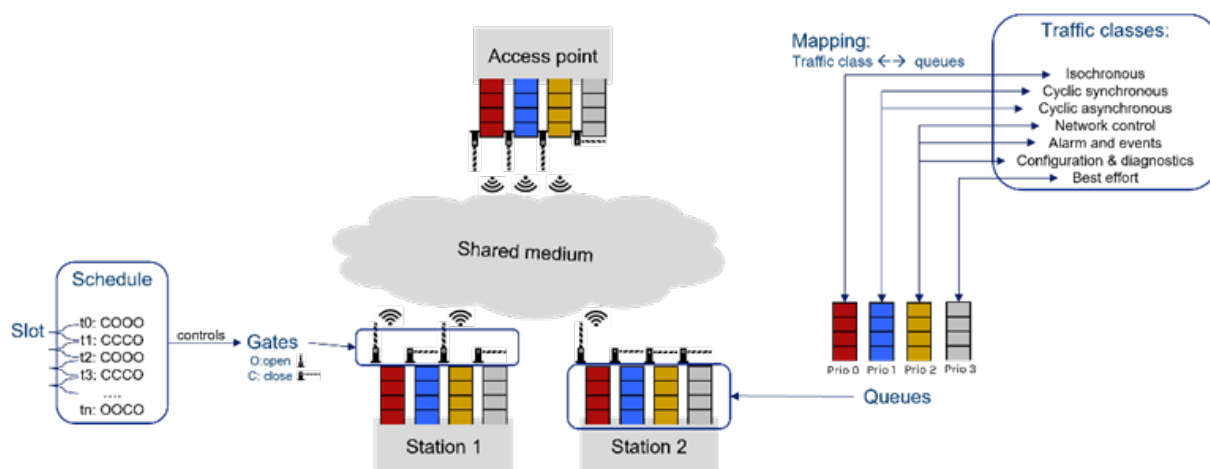


Figure 2-10. Gating mechanism in a wireless TSN setup

This component introduces a tool to configure transmission schedules at the Wi-Fi controller in each station/AP, enabling timely management of the communication channel within a Wi-Fi domain, down to the millisecond range.

The software requires a Linux machine with an AX210 Wi-Fi controller. The machine needs to be attached to a TSN network implementing the 802.1AS (for synchronization) and the 802.1Qbv (for transmission scheduling).

The tool is a Python script that accepts an input file specifying a list of streams and the name of the Wi-Fi controller to configure or to remove previous configuration. It parses the streams, computes a schedule, transforms it into a *qdisc* configuration and applies it to the interface. Transforming the schedule into a *qdisc* configuration involves mapping the traffic classes to the priority queues as well as defining the base-time of the schedule. The time at which the schedule is activated. Figure 2-11 summarises the workflow of the resource configuration tool.

The resource configuration tool is a command line tool that offers the following parameters:

- -i <name>: the Wi-Fi controller to configure
- -f <file path>: the path to the input file defining the streams to configure
- -u: when provided the tool ignores the -f option and deletes the existing configuration from the specified interface
- --verbose: when specified, the tool shows some logs of the operations.

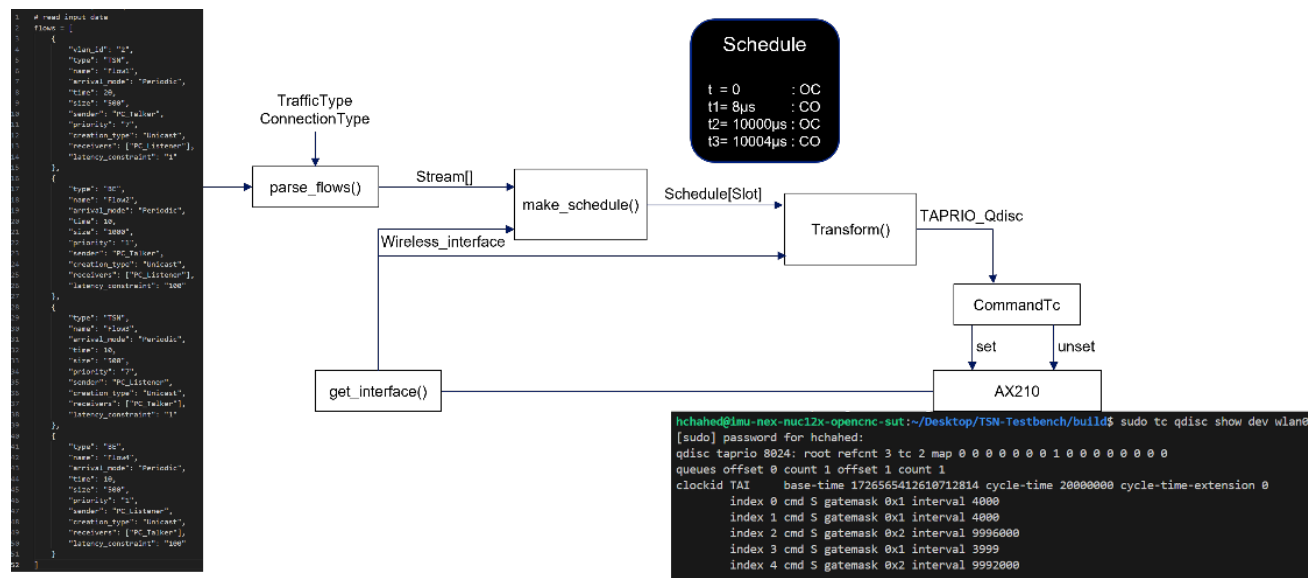


Figure 2-11. Workflow of the resource configuration tool

Ethernet-TSN Resource Configurator

The Ethernet TSN approach for resource configuration aligns with the deterministic need of critical traffic, offering predictable medium access through scheduled transmission. This contribution introduces a TSN Ethernet Resource Configurator tool designed to manage transmission schedules at the Ethernet TSN controller, employing standards like IEEE 802.1Qbv (Serna, 2018) and 802.1AS (802.1AS, 2020) for precise synchronization and traffic scheduling. The gating mechanism restricts transmission to specific time slots, ensuring bounded latency by segregating critical and non-critical traffic. The tool enables the configuration of Ethernet interfaces within a TSN domain, facilitating low-latency and jitter-free communications.

To guarantee performance under peak network loads, the configurator incorporates a Worst-Case Transversal Time (WCTT) analysis. The WCTT analysis evaluates the maximum possible delay that packets might experience in the network, considering factors like network congestion, frame pre-emption, and traffic shaping. The analysis process includes:

- **Network Simulation:** The configurator runs simulations using real-time network parameters to predict delays under worst-case conditions.
- **KPI Measurements:** It measures critical metrics like delay, jitter, and throughput, capturing worst-case performance scenarios.
- **Performance Evaluation:** The tool compares measured KPI against predefined thresholds and adjust Configurations if any constraints are violated.
- **Configuration Adjustments:** Iterative Updates are applied to the schedule and traffic policies until the WCTT constraints are satisfied.

The TSN Ethernet Resource Configurator is implemented as a Java-based software suite utilizing the *Pegase* library (Pegase, 2025) for network configuration. The tool is designed to generate and configure Ethernet network topologies, define communication needs, and manage flow scheduling according to IEEE 802.1Qbv standards.

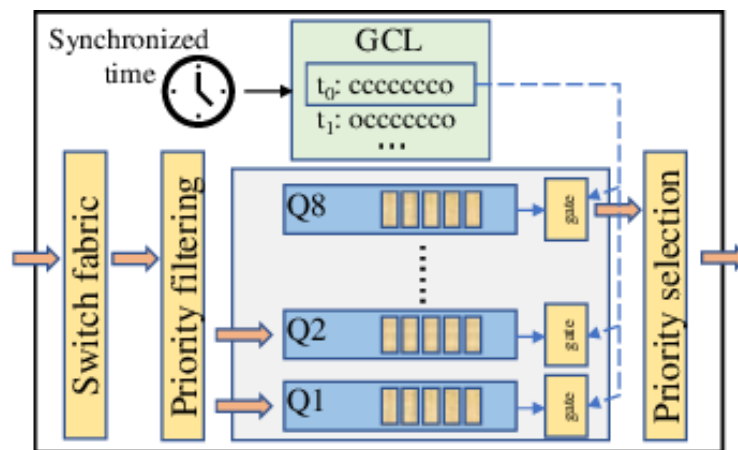


Figure 2-12 from the IEEE 802.1Qbv standard (Tan et al, 2022). Eth-TSN Resource Configuration follows this approach

The software suite consists of three main components:

- **NodeTopology:** Handles the creation of nodes (end-devices) and switches within the Ethernet TSN domain. This class initializes nodes, adds switches (optionally using pre-defined profiles like *Relyum* switches (Relyum, 2025)), and establishes links between nodes and switches. The links are configured based on data transmission rates, and the topology exposure is automated via file reading.
- **FlowConfig:** Manages the flow Configuration by defining communication needs (*EthernetFrameComNeed*) and routing paths. The class parses communication requirements, can compute routes, and applies gating schedules for traffic flows. The

flow configuration runs as a separate thread to allow concurrent execution, improving responsiveness.

- **RCTSN Main Class:** Acts as the entry point of the application. It initializes the dataset, creates the network topology using *NodeTopology*. The main workflow includes:
 1. Creating the Ethernet topology and adding nodes, switches, and links.
 2. Defining communication needs and routing using the *Pegase* API.
 3. Launching the flow configuration in a dedicated thread to apply traffic schedules based on the IEEE 802.1Qbv gating mechanism.
 4. Optionally displaying the results through a graphical interface.

3GPP Resource Configurator

The wireless nature of 3GPP networks introduces several issues, compared with the fixed networks, such as the latency and reliability. For 3GPP networks to work into a TSN ecosystem must be addressed enhancements in reducing the variability of the latency and increase the reliability. To improve the time-sensitiveness and reliability of the 3GPP networks it has been developed innovations, beyond the state of the art, around the software switch concept introduced in (TR21918, 2024). The TSN switch is designed to help in the implementation of new protocols, schedulers and algorithms on top of 3GPP networks.

Regarding the variability of the latency, synchronous schedulers, as defined in IEEE 802.1Qbv, and de-jittering procedures, based on the standard hold and forward mechanism, are featured. On the other hand, to improve the reliability, the work is focused on a packet replication solution according to FRER defined in IEEE 802.1CB, that it is implemented using separate elements of a 5G System.

The TSN switch implements a mechanism to schedule packets without the intervention of the Linux Kernel, to do so, takes advantage of the XDP, an eBPF-based high-performance data path implemented in the Linux kernel that provides a mechanism to forward packets between the NIC drivers and the Kernel user plane.

There are two ports exposed to the UE side, one dedicated to the TSN traffic, and another to the background traffic. Toward the 5GS, the TSN switch exposes two VxLAN ports: one for all traffic and another secondary one for FRER replication.

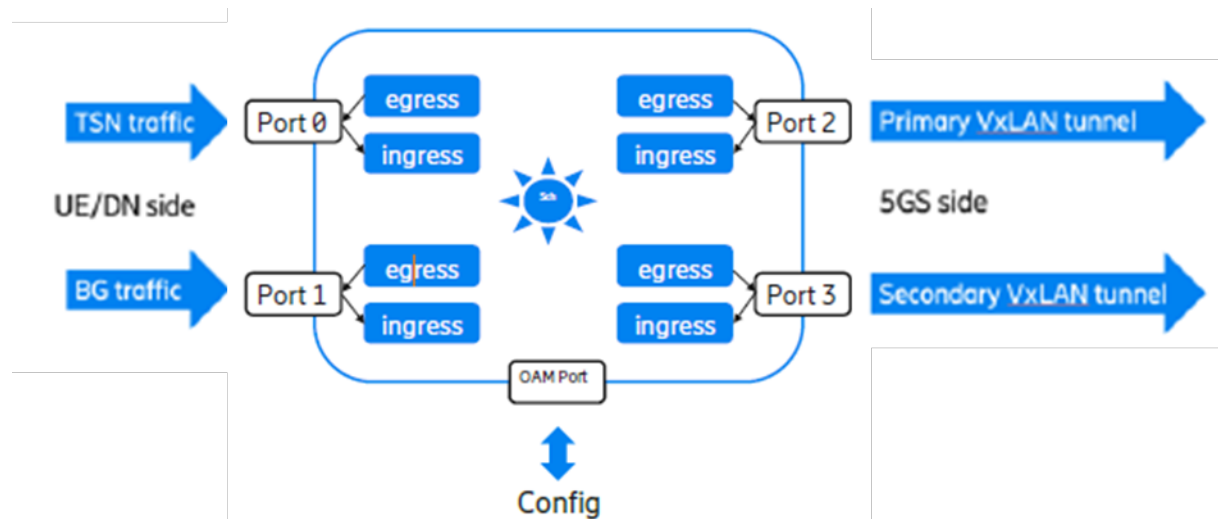


Figure 2-13. TSN software switch architecture

The TSN software switch has two sub-routines associated to each layer 2 port, one for transmitting packets (frames) and the other for receiving. Every routine, ingress and egress, has its own configurable logic.

Initially, 3GPP network, without TSN enhancements, is configured through a REST based API (POST `/nnp/{nnpId}/configuration`, being the `nnpId` the non-public network identifier), the JSON included in the request body provides the parameters that model the 3GPP network radio characteristics (e.g., TDD patterns, spectrum assignment, bandwidth, MIMO, power, spectrum usage technique...). Based on the actual 3GPP configuration and the incoming information such as the expected QoS by the 3GPP path (e.g., RTT, jitter, Packet loss percentage, reliability...) and the traffic characteristics (e.g., bit rate, periodicity, burst size...) the 3GPP Resource Configurator is requested to compute a path in the 3GPP domain, including KPI estimations. Following scenarios may take place:

- The current 3GPP configuration can meet the QoS restrictions at least in one path: the 3GPP resource configurator compute the path including the predicted KPI estimated by the 3GPP Network Digital Twin.
- The current 3GPP configuration cannot meet the QoS restrictions, in terms of bit rate or latency, in any of the available paths: the actual 3GPP configuration may be changed (e.g., radio bandwidth may be increased) to meet the requested QoS characteristics.
- The current 3GPP configuration cannot meet the QoS restrictions in terms of jitter and/or reliability: in that case the resource configurator may compute a path including TSN translators in UE and network sides.

The 3GPP Resource Configurator, to configure the TSN switches, makes use of a tool called *gopsav2*, implemented in Go language, that allows to configure the UE and 5G network side TSN switches, including how each flow is classified, based on configured conditions, and prioritized according to the synchronous scheduling configuration.

2.7 Data Collection and management

As described in section 4.8 of (PREDICT-6G/D3.2, 2023), the Data Collection and Management (DC&M) module is the AICP component responsible for the gathering of monitoring data from the MDP, to feed the logic of analytic processes based on AI/ML and DT for deterministic service monitoring and data-driven network automation decisions.

The DC&M encompasses two MSs: one at the technological domain (MD) level (Measurement Collection MS) and one at the E2E level (E2E Monitoring). The former collects metrics from the coupled technological domain, uniforms the data in a common format and provide the results both in a near real-time fashion and in form of timeseries. The latter assembles metrics from multiple MD Measurement Collection MSs, thus composing a hierarchical structure among all the monitoring platforms and performs the same operations of adaptation and data exposure (to the final user) of the local MD counterpart (PREDICT-6G/D3.2, 2023).

In the context of the AICP operational workflow, that is the provisioning and decommissioning of an E2E deterministic service, the data collection and management is triggered by the SA (both MD and E2E) to provide actual values of the KPIs to DTs.

Although the implementation progressed, the software design of the DC&M component has not undergone any changes with respect to (PREDICT-6G/D3.2, 2023), so this paragraph reports a brief description of the submodules and their functionalities, which are shared between the MD Measurement Collection and the E2E Monitoring.

- **Adaptation Layer.** Translates data gathered from heterogeneous data sources, by Programmable Data Collectors configured on demand, in a common format.
- **Event Streaming Bus.** Exposes uniformed data in a near real-time manner.
- **Time-series DB.** Stores the uniformed data in form of timeseries.
- **Data Manipulation Logic (optional).** Computes aggregated values, such as mean and standard deviation, from raw metrics.
- **Configuration Logic.** Performs the operations behind the REST-based APIs exposed by the Monitoring Platform NBI, which in summary allows at runtime to both create/terminate Programmable Data Collectors and authenticate/authorize data consumers.

- **Multi-technology access interface.** Represents the set of exposed interfaces: The **Configuration I/F**, the **Near-RT** and the **Historic Data Retrieval I/Fs** to interact with the **Event Streaming Bus** and the **Time-series DB**.

As reported in (PREDICT-6G/D3.2, 2023), the Authentication and Authorization feature has been implemented for the Config Manager, too, thus allowing only authorized to change the configuration. The implementation leverages on *Keycloak* and the mechanism is the same performed for the Service Ingestion and described in section 2.2, which is based on token exchange with the IAM platform and validation of that token with respect to user role and scope. As done with the previous MSs of the AICP, the integration tests associated to the DC&M at both MD and E2E level are reported in Appendix 7.1

3 External entities

This chapter presents external entities that provide enhanced capabilities to the AICP, namely Network Digital Twin (NDT) solutions and AI/ML applications that can be injected by means of the deployed MLOps framework.

Regarding NDT solutions, they model TSN technologies in the Ethernet and the WiFi domains. TSN enhances deterministic communication in modern networks by providing precise scheduling and low-latency data transmission. The key component is the Time-aware Shaper (TAS). TAS introduces a gate control mechanism, a time-triggered scheduling approach that regulates network traffic according to a predefined sequence. This mechanism ensures organized and deterministic communication by defining when specific traffic queues are allowed to forward packets. Each network switch egress port supports multiple traffic queues, where packets are classified and assigned based on predefined rules. The gate control schedule determines whether a queue's gate is open or closed at any given time. Only packets from queues with open gates can proceed to transmission. This chapter describes the updates of such mechanisms with respect to (PREDICT-6G/D3.2, 2023) and (PREDICT-6G/D3.3, 2025) as well as a set of functional validation tests of the NDT solutions for the technological domains. An E2E NDT is presented to provide E2E KPI estimations (delay and jitter) based on the estimations from the domain NDTs. The E2E NDT performance is assessed as well.

Finally, the chapter includes the final release of the MLOps framework. As explained in (PREDICT-6G/D3.2, 2023) and (PREDICT-6G/D3.3, 2025), the MLOps framework is a module designed to offer AI-as-a-Service capabilities to the rest of AICP MSs, supporting the design, training and serving of AI/ML models in the management and orchestration domain control loops of centralized or distributed production environments. As for the NDTs, a set of operational tests have been carried out for validating the MLOps framework.

3.1 Network Digital Twin for the Ethernet TSN domain

From a design and implementation perspective, TSN domain NDT has not suffered changes from the one reported in (PREDICT-6G/D3.2, 2023) and (PREDICT-6G/D3.3, 2025). Nonetheless, the models used by the component have been updated and refined to improve accuracy. In this regard, this section presents the functional validation that has been conducted to assess the performance of the component in terms of KPI estimation accuracy.

Test 1 - Accuracy of KPI estimation

Description

This test focuses on the evaluation of the accuracy of KPI estimation for TS flow provisioning. For the evaluation, the network topology in Figure 3-1, where robots exchange packets with users through a TSN network was considered.

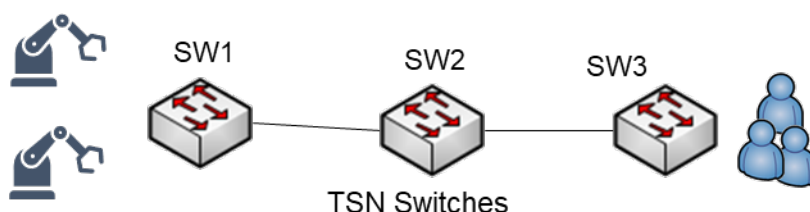


Figure 3-1. Network topology

Two applications are considered that generate traffic flows with different characteristics. Table 3-1 summarizes the characteristics of TSN flows. For these tests, the duration of the super-frame was set to 10ms.

<i>Param</i>	<i>TSN Application-1</i>	<i>TSN Application-2</i>
<i>Period</i>	1 ms	10 ms
<i>Max delay</i>	1 ms	10 ms
<i>Max Jitter</i>	100 μ s	1 ms
<i>Traffic</i>	random [90, 120] bytes	random [900, 1200] bytes

Table 3-1. TSN flows characteristics for the considered applications

The assumption is that links connecting the switches accumulate a transmission delay of 300 μ s.

Execution steps

A simulator that follows the provisioning algorithm described in (PREDICT-6G/D3.1, 2023) was developed. The simulator generates single-flow service requests for a randomly selected TSN application and involving devices/servers from the customers. The requests arrive sequentially, and if a request is accepted, the selected resources are reserved, and the

simulator continues with the next request, which is processed on the resources remaining available on the network.

Because of the characteristics of the network topology and the applications, the simulation is stopped when a request cannot be served, as no more requests will be accepted.

Results

Figure 3-2 presents the obtained E2E delay (upper row) and jitter (lower row) for the requested TS flows; the evolution of the maximum and mean values is presented separately for each application.

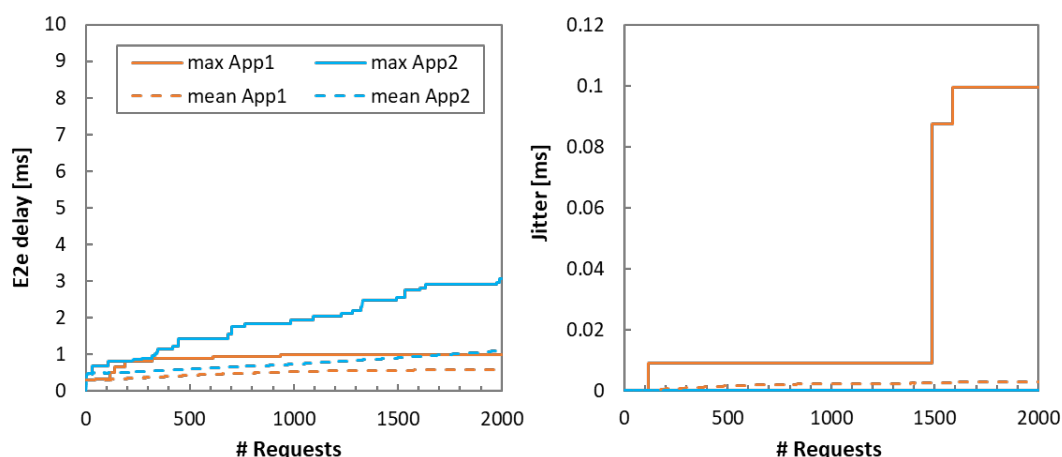


Figure 3-2. E2E delay and jitter for requested TS flows

It is observable that delay, and jitter increase with the number of flows in the network (load), being the maximum delay and jitter under the required ones. It is worth noting that the delay of flows for application 2 has zero jitter since the period of those flows and the duration of the SF coincide.

Test 2 – KPI estimation accuracy assessment in front of experimental measurements

Description

Once the KPI estimation is validated by simulation for a large number of requests, in this test, the estimation with regard to experimental measurements is validated. To that end, a number of experiments to collect such real measurements on the following experimental setup were carried out:

- Path Topologies: 1.4 TSN switches

- TS flows:
 - Robot Control Loop (User->Robot)
 - Robot Odometry (Robot->User)
- Captures (csv files)
 - Duration: 60 seconds
 - For each flow: packet id; timestamp tin; timestamp tout; latency=tout-tin

Considered Traffic Models

Robot Control Loop (User->Robot)

```
"traffic_characteristics":{
  "direction": "One-Way",
  "periodicity": true,
  "period": 10ms,
  "burst_size": 100 byte,
  "maximum_flow_bitrate": - }
"qos_characteristics": {
  "priority": 1,
  "td_reliability": 100,
  "td_packet_loss": 0,
  "td_delay": 20,
  "td_rtt": -,
  "td_jitter": 10,
  "burst_arrival_time_window": {
    "t1": -,
    "t2": -
  },
  "burst_completion_time_window": {
    "t1": -,
    "t2": -
  }
}
```

Robot Odometry (Robot->User)

```
"traffic_characteristics":{
  "direction": "One-Way",
  "periodicity": true,
  "period": 20ms,
  "burst_size": 1000 byte,
  "maximum_flow_bitrate": - }
"qos_characteristics": {
  "priority": 1,
  "td_reliability": 100,
  "td_packet_loss": 0,
  "td_delay": 10,
  "td_rtt": -,
  "td_jitter": 5,
  "burst_arrival_time_window": {
    "t1": -,
    "t2": -
  },
  "burst_completion_time_window": {
    "t1": -,
    "t2": -
  }
}
```

Experimental measurements and data cleaning

Figure 3-3 shows the delay vs the number of switches of traversed by the flows for all the packets conveyed. It is observable that a large majority of packets experienced a delay that concentrated very closely to the median. However, around 5% of packets experienced delays that differs significantly from the median and can be considered as outliers.

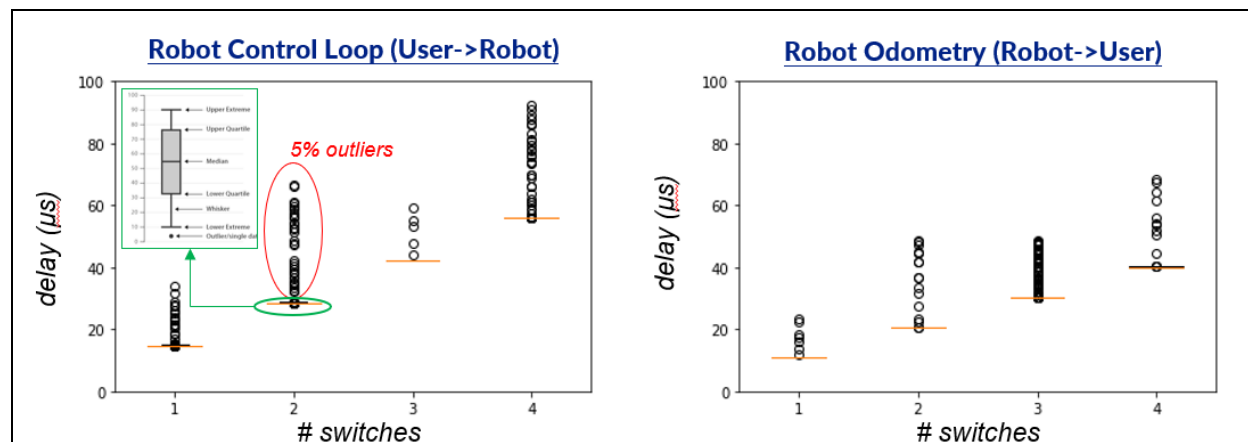


Figure 3-3. Delay vs number of switches traversed by the flow

Then, the outliers and obtained the following delays for the flows were removed (Figure 3-4).

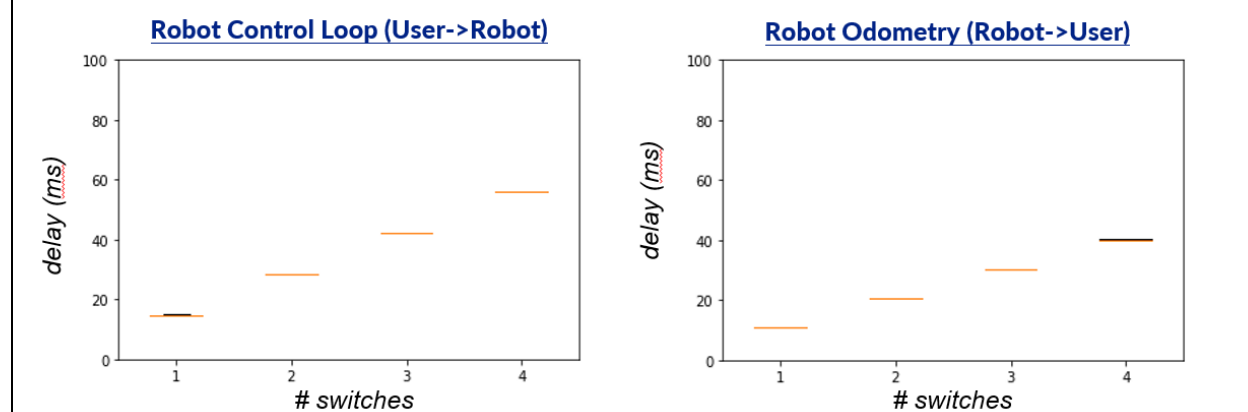


Figure 3-4. Delay vs number of switches without outliers

Experimental validation of DT KPI estimation

The experimental measurements were used for evaluating the accuracy of the estimation of delay and jitter from the DT. To this end, we consider the maximum delay from the experiments as the value for the percentile 95, whereas the jitter value was computed as the difference between the maximum (percentile 95) and the minimum (percentile 5) delay experiences by the packets of the flows.

Figure 3-5 shows the results for the two flows. It is observable that the accuracy of KPI estimation is noticeably higher for both delay and jitter, which experimentally validates the DT models.

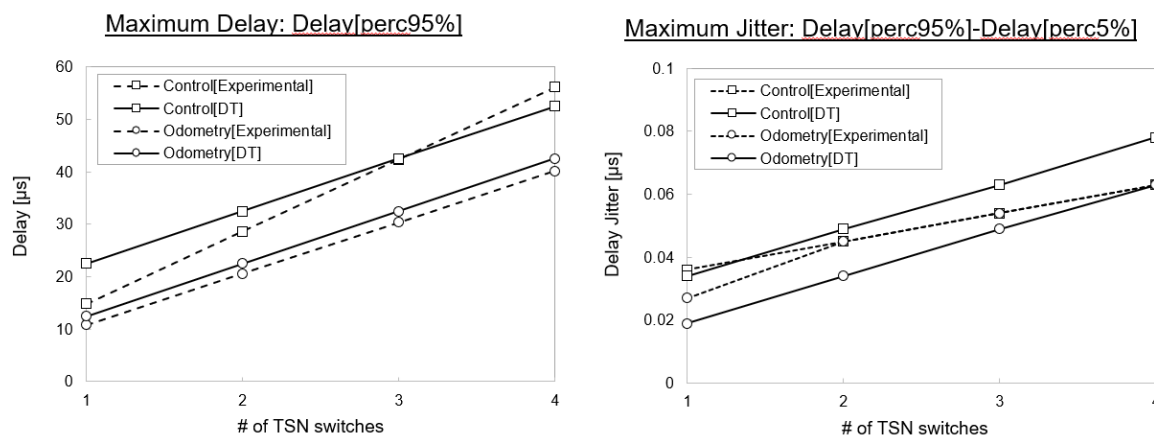


Figure 3-5. DT KPI evaluation accuracy

3.2 WiFi Digital Twin

This section presents the tests conducted to validate the performance of the DT for the Wi-Fi domain, where the Target Wake Time (TWT) mechanism is used for deterministic channel access as well as for energy savings.

The network scenario implemented in the DT has been developed on top of the advanced ns3 Wi-Fi TWT simulator, as detailed in (Venkateswaran, 2024). To achieve low-latency and low-jitter communication, the simulator was extended with the TAS, as specified in (802.1Qbv, 2015).

As illustrated in Figure 3-6, every switch egress port can support up to eight traffic queues, each controlled by an individual gate positioned between the queue and the L3 switch. The entire scheduling operation follows a cyclically repeating gate control schedule.

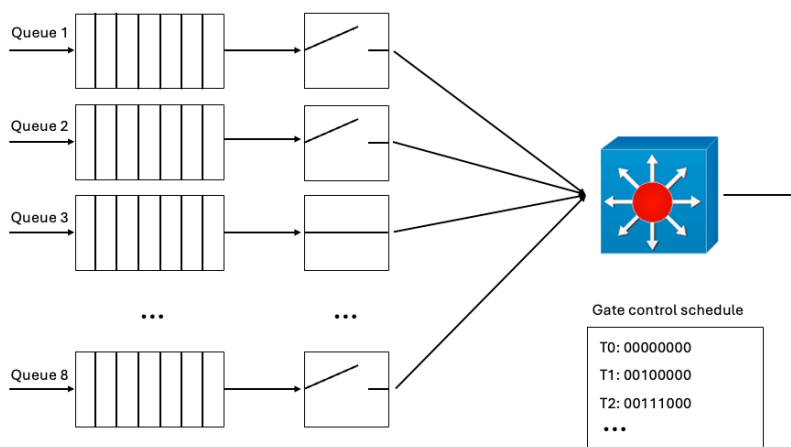


Figure 3-6: Implemented Time Aware Shaper with 8 queues and a gate control schedule.

The implemented solution builds upon the approach introduced in (Krummacker, 2020) and enables the configuration of two key parameters:

1. **Packet Filtering Mechanism**, which classifies incoming packets based on the network of the sending STA. This classification determines the priority class of the packet;
2. **Gate control schedule**, which defines the time slots during which each queue's gate remains open. This cyclic schedule dictates which priority classes are allowed to transmit packets at given points in time.

One of the key benefits of TWT is its ability to improve energy efficiency. By allowing STAs to enter a low-power sleep mode when they are not actively transmitting or receiving data, they can significantly reduce energy consumption. Without this mechanism, STAs would have to keep their radios in receive mode, consuming power during overhearing, i.e., when receiving data packets that are not intended for them or during idle time, i.e., while performing operations like Clear Channel Assessment (CCA).

Test 1: TWT energy efficiency

Description

The considered scenario is depicted in Figure 3-7.

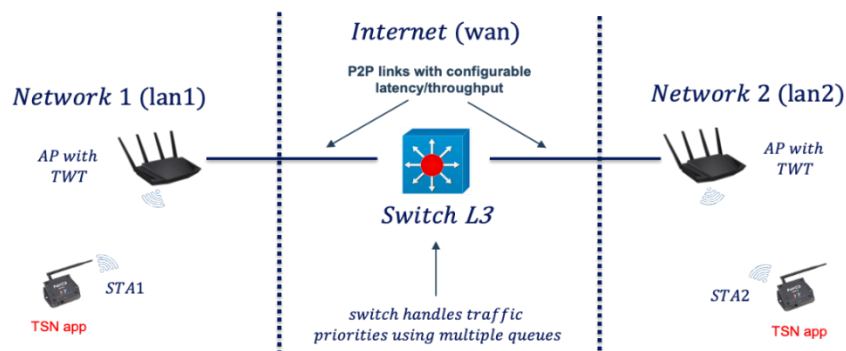


Figure 3-7: Network scenario implemented in the DT: A TS multi-domain network with the Wi-Fi domain leveraging the TWT mechanism.

includes eight STAs, representing different types of IIoTs with varying energy capabilities and consumption characteristics, transmitting data to an AP. The generated traffic consists of 1,500-byte UDP packets carrying sensor data to be transmitted towards the AP. Each STA s ($s = 1, \dots, 8$) receives a UDP packet from its higher layers at time intervals of $s \cdot 5$ ms, consistently with the fact that IIoTs typically send data at regular intervals. Taking advantage of this predictability of the generated traffic, the network can leverage a TWT mechanism where each TWT Service Period (SP) begins as soon as a UDP packet arrives at the STA from its higher layers. For the sake of clarity, in the considered scenario, each TWT SP is configured to last slightly longer than 5 ms.

Results

The energy-saving potential of TWT, which is particularly valuable for battery-operated IIoTs with, is illustrated in Figure 3-8 and Figure 3-9.

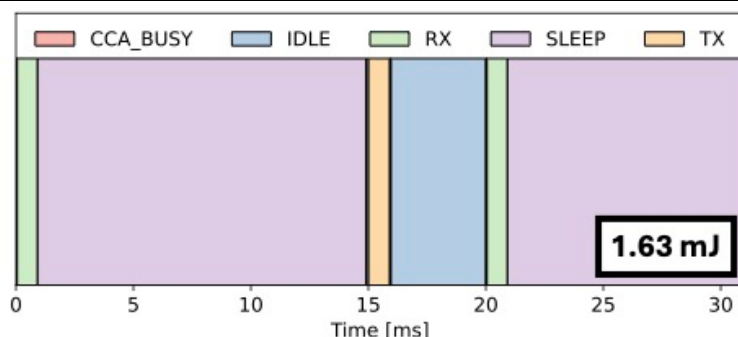


Figure 3-8: Example of temporal evolution of STA's power states with TWT, over a period of 32 ms.

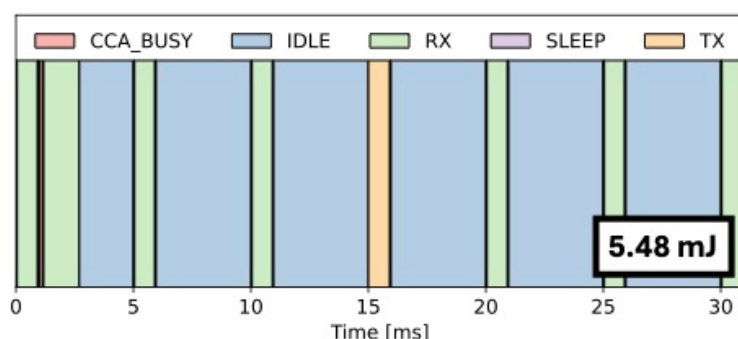


Figure 3-9: Example of temporal evolution of STA's power states without TWT, over a period of 32 ms.

Figure 3-8 and Figure 3-9 compare the physical layer state evolution of STA 3 over time, respectively, with TWT and without TWT. The figures also report the total energy consumption measured over a reference period of 32 ms. One can observe that when TWT is enabled, the STA remains in the low-power sleep state for most of the time, except for two key instances: when it receives the beacon from the AP and during its scheduled TWT SP, i.e., when it transmits data. In contrast, without TWT, the STA is unable to enter the sleep state at all. It continuously switches between idle and receive states, entering the latter when it overhears other STAs transmitting towards the AP. Since STAs consume significantly more energy in idle state than in sleep state, leveraging TWT leads to substantial energy savings.

Specifically, the results show a remarkable 3.4x reduction in energy consumption over just a 32-ms reference period.

3.3 E2E Digital Twin

The E2E DT is the component in charge of performing E2E KPI estimations during the E2E service provisioning procedure. This building block receives estimations (outputs) from

domain DTs, i.e., Ethernet TSN DT, Wi-Fi DT, and 3GPP DT and it produces as output the estimation of relevant KPI (throughput, delay) obtained by combining domain DT estimations.

The functionalities that the E2E DT must support are as follows:

- It must expose an interface with the E2E PCE to receive KPI estimation requests, as detailed in the workflow in section 4.1.
- It needs to perform E2E estimations based on the partial heterogeneous estimations of domain DTs. Results need to be produced in a way that can be easily processed and interpreted by the E2E PCE module, to perform unequivocal KPI validation. Communication of E2E KPI estimations are returned as responses to KPI estimation requests through the same interface.
- Additionally, the E2E DT needs to be able to retrieve monitoring data from E2E monitoring database to check whether KPI estimations of existing established flows were accurate. Moreover, retrieved monitoring data might be used for internal model validation purposes.

Next, the tests that have been performed to validate the functionalities of the E2E DT described above are presented:

Test 1: E2E KPI estimation
Description
<p>This test focuses on the validation of the process that estimates E2E KPIs based on the KPIs provided by each domain DT. The list of E2E KPIs is limited to throughput and RTT, which can be accurately composed from independent domain estimations.</p> <p>The considered E2E scenario is depicted in Figure 3-10 and consists in the evaluation of the E2E KPIs of a TSN traffic flow across multiple domains.</p>

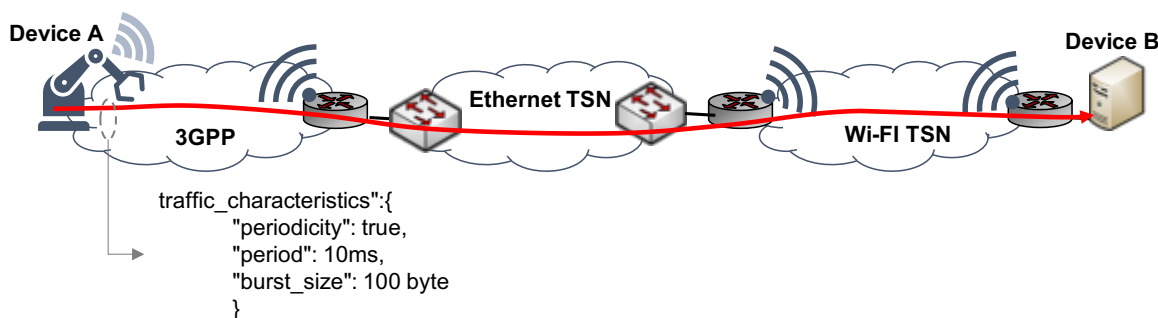


Figure 3-10. E2E network scenario

Execution steps

1. The request received from the PCE and validated in Test 1 is emulated based on actual domain DT measurements. Then, it is parsed to retrieve the values of the predicted KPIs for each domain. The resultant data is structured as follows:

```
{
  [
    {
      "index": 0,
      "rtt": 144.375,
      "throughput": 0.0234
    },
    {
      "index": 1,
      "rtt": 5.092,
      "throughput": 0.0217
    },
    {
      "index": 2,
      "rtt": 251.453,
      "throughput": 0.0251
    }
  ]
}
```

This data object is the input of E2E KPI estimation process

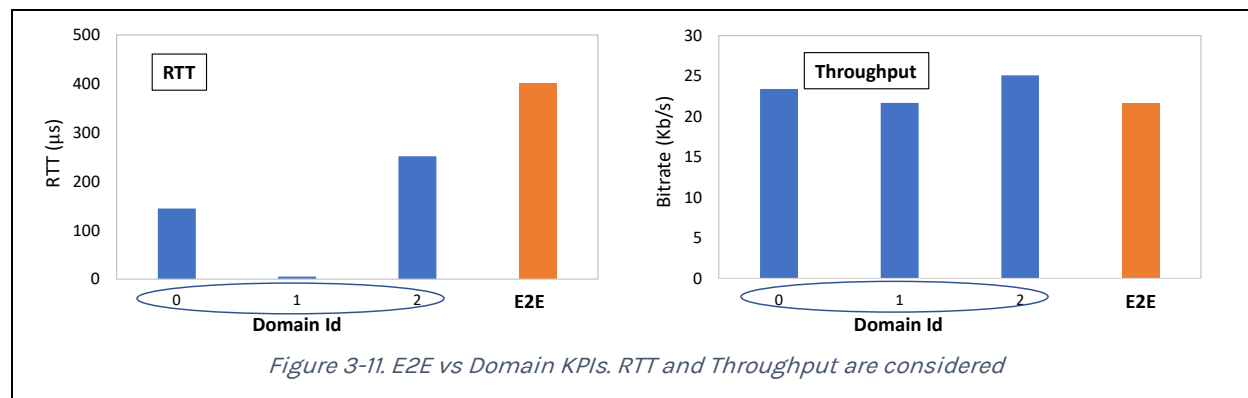
2. E2E KPI estimation process is executed for the obtained input data. It performs the aggregation of RTT and throughput values from domains to E2E estimations. The resultant output result is a simple structured data object with E2E KPI estimations:

```
{
  "index": "e2e",
  "rtt": 400.921,
  "throughput": 0.0217
}
```

This output data is embedded into the response message detailed in Test 1.

Results

The E2E KPI estimation process has been evaluated and validated. The following figure compares the domain KPIs (input) and E2E KPIs (output) of this process for the example detailed in the description section:



Test 2: E2E KPI monitoring

Description

This test focuses on the evaluation of the KPI monitoring of an E2E flow. Once a flow is established a monitoring process controls the behaviour of the flow by comparing telemetry measurements with the expected QoS parameters assigned to the flow.

Execution steps

1. A flow has been established and its QoS parameters are monitored
2. A sample is collected periodically from the Telemetry. The data has the format below:

```
,result,table,start,stop,time,value,field,measurement,host,pathLabel,serviceLabel
,result,0,2025-03-10T10:31:12.214324138Z,2025-03-10T10:32:12.214324138Z,2025-03-10T10:31:15Z,1,value,latency,telegraf_mdp,Path-1,7
,result,0,2025-03-10T10:31:12.214324138Z,2025-03-10T10:32:12.214324138Z,2025-03-10T10:31:20Z,1,value,latency,telegraf_mdp,Path-1,7
,result,0,2025-03-10T10:31:12.214324138Z,2025-03-10T10:32:12.214324138Z,2025-03-10T10:31:25Z,1,value,latency,telegraf_mdp,Path-1,7
,result,0,2025-03-10T10:31:12.214324138Z,2025-03-10T10:32:12.214324138Z,2025-03-10T10:31:30Z,1,value,latency,telegraf_mdp,Path-1,7
,result,0,2025-03-10T10:31:12.214324138Z,2025-03-10T10:32:12.214324138Z,2025-03-10T10:31:35Z,1,value,latency,telegraf_mdp,Path-1,7
,result,0,2025-03-10T10:31:12.214324138Z,2025-03-10T10:32:12.214324138Z,2025-03-10T10:31:40Z,1,value,latency,telegraf_mdp,Path-1,7
,result,0,2025-03-10T10:31:12.214324138Z,2025-03-10T10:32:12.214324138Z,2025-03-10T10:31:45Z,1,value,latency,telegraf_mdp,Path-1,7
,result,0,2025-03-10T10:31:12.214324138Z,2025-03-10T10:32:12.214324138Z,2025-03-10T10:31:50Z,1,value,latency,telegraf_mdp,Path-1,7
,result,0,2025-03-10T10:31:12.214324138Z,2025-03-10T10:32:12.214324138Z,2025-03-10T10:31:55Z,1,value,latency,telegraf_mdp,Path-1,7
,result,0,2025-03-10T10:31:12.214324138Z,2025-03-10T10:32:12.214324138Z,2025-03-10T10:32:00Z,1,value,latency,telegraf_mdp,Path-1,7
,result,0,2025-03-10T10:31:12.214324138Z,2025-03-10T10:32:12.214324138Z,2025-03-10T10:32:05Z,1,value,latency,telegraf_mdp,Path-1,7
,result,0,2025-03-10T10:31:12.214324138Z,2025-03-10T10:32:12.214324138Z,2025-03-10T10:32:10Z,1,value,latency,telegraf_mdp,Path-1,7
```

3. The telemetry measurements are compared with the QoS.
4. A notification is generated and sent to the SA module if the QoS is violated:

```
{
  "service_id": 7,
  "status": "QoS violated"
}
```

Results

The E2E KPI monitoring process has been evaluated.

Test 3: E2E DT interface validation

Description

This test focuses on the evaluation of the DT interface that exposes the E2E KPI estimation capability. A request containing the description of the E2E service and the KPIs of each domain is received at the interface and the data is processed by the DT.

Execution steps

1. A POST request from the PCE is sent to the E2E DT endpoint with the following format:

```
{
  "id": "33333",
  "endpoints": {
    "source": {"id": "DeviceA"},
    "destination": {"id": "DeviceB"}
  },
  "e2e_path_selection": {
    "e2e_path_id": "path1",
    "e2e_path_status": "provisioned",
    "domain_list": [
      {"index": 0, "path_domains": {
        "name": "Domain_1", "domain_class": "3GPP"
      }},
      {"index": 1, "path_domains": {
        "name": "Domain_2", "domain_class": "WiFi"
      }},
      {"index": 2, "path_domains": {
        "name": "Domain_3", "domain_class": "TSN"
      }}
    ]
  },
  "qos_characteristics": {
    "priority": 1, "reliability": 100,
    "packet_loss": 10, "delay": 400,
    "rtt": 300, "jitter": 42,
    "burst_arrival_time_window": {
      "t1": 5, "t2": 10
    },
    "burst_completion_time_window": {
      "t1": 6, "t2": 12
    }
  },
  "traffic_characteristics": {
    "direction": "bidirectional",
    "periodicity": false,
    "period": 0,
    "burst_size": 20,
    "maximum_flow_bitrate": 5000
  },
  "service_lifetime": {
    "service_duration": {
      "t1": 20000,
      "t2": 60000
    },
    "recurrent_service_interval": {
      "t1": 30000,
      "t2": 80000,
      "recurrence_interval": 7
    }
  },
  "md_services": {
    "local_service": [
      {
        "index": 0,
        "service": {
          "id": "1111",
          "service_status": "pending",
          "target_domain": {
            "name": "Domain_1",
            "domain_class": "3GPP"
          },
          "predicted_kpis": {
            "rtt": 167.956,
            "jitter": 41.527,
            "throughput": 0.0234
          }
        }
      },
      {
        "index": 1,
        "service": {
          "id": "1112",
          "service_status": "pending",
          "target_domain": {
            "name": "Domain_2",
            "domain_class": "WiFi"
          },
          "predicted_kpis": {
            "rtt": 144.373,
            "jitter": 51.177,
            "throughput": 0.0234
          }
        }
      },
      {
        "index": 2,
        "service": {
          "id": "1113",
          "service_status": "pending",
          "target_domain": {
            "name": "Domain_3",
            "domain_class": "TSN"
          },
          "predicted_kpis": {
            "rtt": 28.874,
            "jitter": 10.156,
            "throughput": 0.0234
          }
        }
      }
    ]
  }
}
```

2. The estimation of the E2E KPIs is calculated (see Test 2). The data is formatted and returned to the PCE with the requested data. The response has the following format:

<pre> { "service_id": "33333", "kpi": { "throughput": 0.0234, "rtt": 341.203 } } </pre>
Results
<p>The E2E KPI estimation interface has been tested. The request is processed by the E2E DT and the result is formatted and returned to the PCE that will evaluate the feasibility of the flow's establishment.</p>

3.4 AI/ML-assisted Control Plane

MLOps Algorithmic Framework

As explained in (PREDICT-6G/D3.2, 2023) and (PREDICT-6G/D3.3, 2025), the MLOps framework is a module designed to offer AI-as-a-Service capabilities to the rest of AICP Management Services, supporting the design, training and serving of AI/ML models in the management and orchestration domain control loops of centralized or distributed production environments.

The final implementation of the MLOps framework covers a set of the AI/ML MSs described in (PREDICT-6G/D3.1, 2023) and then redistributed in (PREDICT-6G/D3.3, 2025) as part of the MLOps framework system architecture.

The list of AI/ML MS functionalities defined in (PREDICT-6G/D3.1, 2023) and covered by the MLOps framework are: (i) AI/ML Model Registry to store and serve information of the models already onboarded in the platform; (ii) Dataset Registry to store and serve information about the datasets available for model training or pre-training; (iii) Model Repository to store and serve the AI/ML models already onboarded in the platform; (iv) Dataset Repository to store and serve the datasets already onboarded in the platform; and (v) Resource Orchestration/Learning Manager to allow the creation of new AI/ML models, allow the monitoring and update and redeployment of AI/ML models according to specific metrics or external requests.

Additionally, as identified in (PREDICT-6G/D3.2, 2023), specific extensions were also required for the MLOps framework to support the generation of AI/ML pipelines based on **Pytorch**⁴ framework, the extension of the framework to support Federated Learning, and the integration with the rest of AICP MSs. Considering these additional requirements, the MLOps framework software architecture design has been extended with respect to the previous version defined in (PREDICT-6G/D3.2, 2023), leading to define two different architectures for centralized and distributed scenarios.

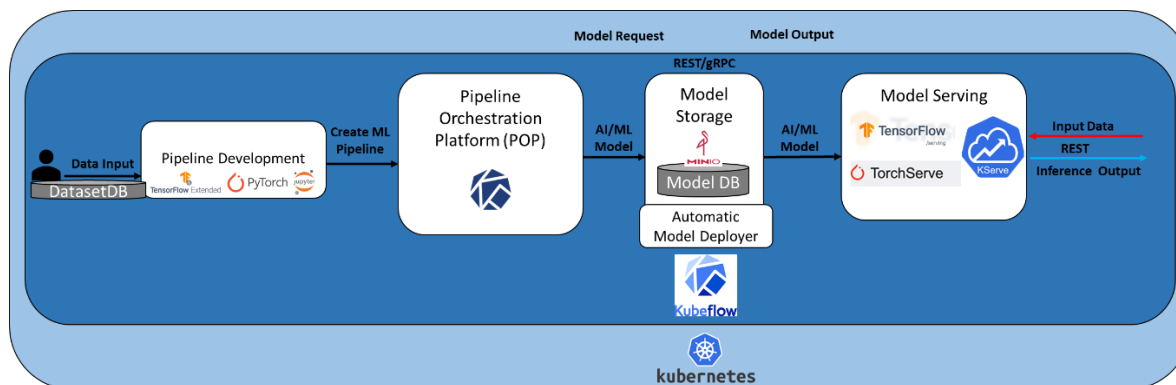


Figure 3-12. Final MLOps framework SW architecture - Centralized Learning

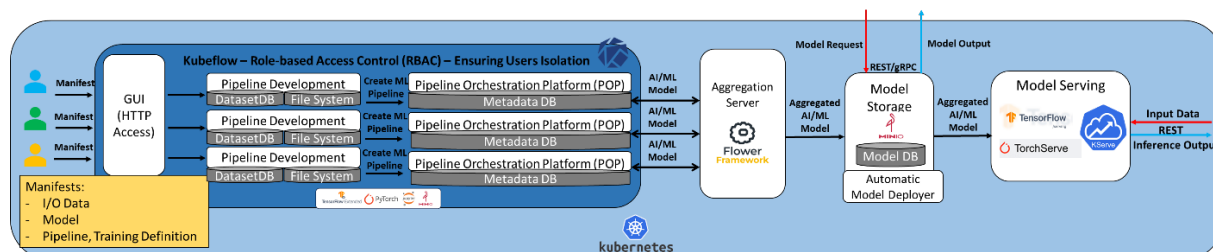


Figure 3-13. Final MLOps framework SW architecture - Federated Learning

As described in (PREDICT-6G/D3.2, 2023), to ensure the principles of portability and scalability, the MLOps framework design relies on the concepts of containerization and its orchestration using Kubernetes (K8s). Based on this approach, the MLOps framework has been built on top of Kubeflow to allow the deployment and orchestration of the AI/ML framework in any infrastructure where K8s is available.

⁴ <https://pytorch.org>

Leveraging **Kubeflow**⁵ as the foundations of the solution, the MLOps framework integrates other libraries and tools like **MinIO**⁶, **KServe**⁷ or **Flower**⁸ as part of a set of modules to meet the required functionalities in centralized or distributed scenarios. Based on this philosophy, the set of modules included in both architectures to support Centralized and Federated Learning strategies are the following:

DatasetDB: Database used to allow the storage of different types of data, including dataset registry and repository functionalities that allow to retrieve complete datasets as well as their most relevant features. The final implementation of this module is based on MinIO and is used as the data source for feeding the pipeline development module.

Pipeline Development: Module that runs over Kubeflow and that enables the development of E2E production AI/ML pipelines through the combination of different data science and AI/ML stages such as data ingestion, data preparation, data validation, model design, model training, or model testing, among others. This module supports Jupyter Notebooks, the generation of pipelines with Tensorflow Extended and has been extended in this final version of the framework to support the development of Pytorch E2E production pipelines based on TorchX.

Pipeline Orchestration Platform (POP): Module designed to orchestrate and automatize the generation of the E2E AI/ML pipelines and workflows. The POP relies on Kubeflow Pipelines and has been extended to support the generation of pipelines in distributed scenarios, e.g. Federated Learning, through the creation of a Role-Based Access control mechanism that allows to isolate data from different users and replicate specific pipeline stages related to data validation, preparation, etc.

Aggregation Server: Module designed to enable the aggregation of model weights from multiple clients in a Federated Learning environment. The module allows to create a global model by learning and improving after the execution of several rounds where each client distributes their own trained weights. The strategy of aggregation and the number of rounds is a parameter that can be configured. This module is based on Flower library.

Model Storage and Automatic Model Deployer: Module envisaged to store the ML model generated as the outcome of the different orchestrated pipelines. The model storage module has available a REST/gRPC interface to serve models under request and relies on MinIO. This module also integrates the Automatic Model Deployer (AMD), which acts as an add-on to

⁵ <https://www.kubeflow.org>

⁶ <https://min.io>

⁷ <https://www.kubeflow.org/docs/external-add-ons/kserve/introduction/>

⁸ <https://flower.ai>

monitor the creation of new models. If new models are created, the AMD forwards the model to the Model Serving module.

Model Serving: Module designed to offer AI/ML model and inference serving capabilities. The module can be deployed by integrating TensorFlow Serving or TorchServe serving modules, which only work in case of TensorFlow or Pytorch-based models, or through the deployment of a specific KServe serving module, which acts as a server agnostic framework and supports the serving of inferences for different types of models through APIs.

Note that the model and metrics monitoring module, which was present in the first version of the MLOps framework, has not been included in this final version of the architecture since it has not been finally required for any use case.

As done with the NDTs described previously, the MLOps framework functional validation is reported in the tables below.

Test 1: Pipeline development and orchestration for Centralized Learning - Pytorch	
Description	
<p>This test focuses on validating the extension performed over the MLOps framework to support the development, deployment and orchestration of AI/ML pipelines based on Pytorch framework in a centralized learning architecture.</p> <p>The test has been performed over a Kubeflow environment by deploying a Kubeflow pipeline composed of several Pytorch stages including data preparation, model training and storage stages.</p> <p>The expected output of the test is the successful generation and deployment of a Kubeflow Pipeline including 3 stages devoted to data generation and ingestion, model training and model storage. The correct deployment of the pipeline will result in the storage of a specific AI/ML model in the Data Storage module.</p>	
Execution steps	
<ol style="list-style-type: none"> 1. Data ingestion: Reproduce the ingestion of data obtained from the DatasetDB. The output of this step is the generation of a specific Python file (datagen.py). 2. Data Pre-processing: Pre-process and structure the ingested data in the shape of Torch tensors to prepare the inputs and output for the model training stage. 3. Model Training: Define the model architecture, in this example a Deep Neural Network (DNN), together with the parameters required to train the model (epochs, batch size) and the optimizer. The output of steps 2 and 3 is the generation of a specific Python file (trainer.py). 4. Model Storage: Store the AI/ML model previously trained in Step 3 in the Model Storage module. The output of this step is the generation of a specific Python file (archiver.py). 	

5. **Pipeline Development:** Create the pipeline by defining the number and type of stages. In this example, three stages will be created to: i) generate the data, ii) pre-process the data and train the model, and iii) store the model. The inputs to the pipeline development will be the Python files generated in Steps 1, 2, 3 and 4. The output of this step will be the generation of a Python file and YAML file to develop and deploy the pipeline (ile_forecasting_pipeline.py).
6. **Pipeline Orchestration (POP):** Once created, the pipeline will be run in a Kubeflow Pipelines environment, performing the training and storage of the AI/ML model.

Results

Figure 3-14 shows the code used to develop a pipeline on Kubeflow by defining three different stages: datagen, trainer and archiver:

```
@kfp.dsl.container_component
def data_gen(filesystem_host: str,
            filesystem_port: str,
            filesystem_user: str,
            filesystem_passwd: str,
            samples: int):
    return kfp.dsl.ContainerSpec(
        image='registry.atosresearch.eu:18467/mlops/kubeflow/pipelines/ile_forecasting:v0.0.1',
        command=['python', '/app/stages/datagen.py'],
        args=[
            '--pipeline_name', pipeline_name,
            '--filesystem_host', filesystem_host,
            '--filesystem_port', filesystem_port,
            '--filesystem_user', filesystem_user,
            '--filesystem_passwd', filesystem_passwd,
            '--samples', samples
        ]
    )

trainer_stage = trainer(filesystem_host=filesystem_host,
                        filesystem_port=filesystem_port,
                        filesystem_user=kbflw_and_filesystem_user,
                        filesystem_passwd=filesystem_passwd,
                        timestamp=timestamp)
trainer_stage.set_caching_options(False)
trainer_stage.after(generate_data_stage)
kubernetes.set_image_pull_policy(trainer_stage, "Always")

archiver_stage = archiver(filesystem_host=filesystem_host,
                          filesystem_port=filesystem_port,
                          filesystem_user=kbflw_and_filesystem_user,
                          filesystem_passwd=filesystem_passwd,
                          timestamp=timestamp,
                          version=model_version)
archiver_stage.set_caching_options(False)
archiver_stage.after(trainer_stage)
kubernetes.set_image_pull_policy(archiver_stage, "Always")
```

Figure 3-14. Kubeflow pipeline code

Figure 3-15 shows a snapshot of the trainer stage where a Pytorch DNN model is initialized, together with the visualization of the pipeline generated over Kubeflow graphical interface:

```
from model import initialDNN

model = initialDNN(25,25)

X_train = torch.load(os.path.join(local_training_directory,"X_train.pt"))
y_train = torch.load(os.path.join(local_training_directory,"y_train.pt"))

# Define the loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training the model
num_epochs = 10
batch_size = 32
```

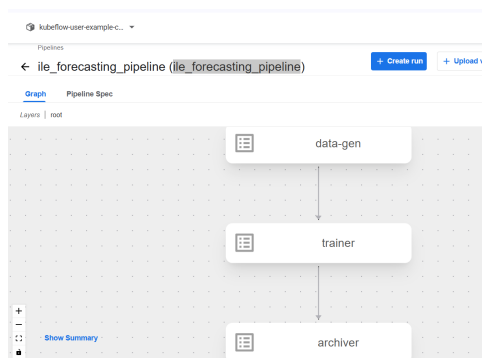


Figure 3-15. Pytorch trainer stage

Test 2: User isolation (Federated Learning)

Description

This test focuses on validating the extension performed over the MLOps framework to support the logic isolation of users in Kubeflow with the objective of emulating a Federated Learning environment where different clients are isolated in their specific workspaces.

The test has been performed over a Kubeflow environment by generating two isolated user profiles and their corresponding two isolated workspaces the DatasetDB environment, which are created in MinIO. The access to each profile is managed by Dex as Authenticator in Kubeflow, and with MinIO authentication for the DatasetDB.

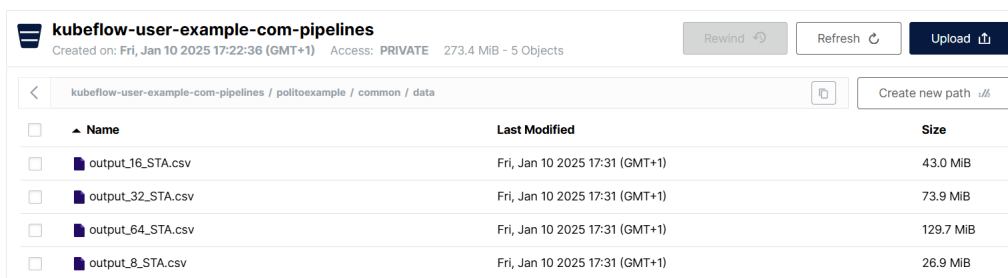
The expected output of the test is the successful generation of two isolated user profiles in Kubeflow and two workspaces in the DatasetDB environment, where each user is only able to see the data stored in his DatasetDB environment and the pipelines generated in their own Kubeflow workspace.

Execution steps

1. **Kubeflow User Isolation:** Create different user profiles in Kubeflow restricting the access and the visibility of each user to their own workspace by leveraging Dex Authentication.
2. **MinIO User Isolation:** Link the generation of the different user profiles to specific workspaces in MinIO, where each user will only be able to access the data stored in their own DatasetDB workspace.
3. **Isolation Validation:** Validate the implemented user isolation by comparing the information that can be seen by a user profile with full access permission and the information that each isolated user profile can see.

Results

Figure 3-16 shows the MinIO view of a user with full access permissions, proving that this user is able to access all the datasets stored across all MinIO workspaces.



The screenshot shows the MinIO web interface for a workspace named 'kubeflow-user-example-com-pipelines'. The interface includes a breadcrumb trail: 'kubeflow-user-example-com-pipelines / politoexample / common / data'. Below the breadcrumb is a table listing four CSV files. Each row has a checkbox, a file name, a 'Last Modified' timestamp, and a 'Size' in MiB. The files are 'output_16_STA.csv', 'output_32_STA.csv', 'output_64_STA.csv', and 'output_8_STA.csv', all modified on 'Fri, Jan 10 2025 17:31 (GMT+1)'.

<input type="checkbox"/>	Name	Last Modified	Size
<input type="checkbox"/>	output_16_STA.csv	Fri, Jan 10 2025 17:31 (GMT+1)	43.0 MiB
<input type="checkbox"/>	output_32_STA.csv	Fri, Jan 10 2025 17:31 (GMT+1)	73.9 MiB
<input type="checkbox"/>	output_64_STA.csv	Fri, Jan 10 2025 17:31 (GMT+1)	129.7 MiB
<input type="checkbox"/>	output_8_STA.csv	Fri, Jan 10 2025 17:31 (GMT+1)	26.9 MiB

Figure 3-16. MinIO datasets view

Figure 3-17 illustrates the MinIO view of two users (client1 and client2) created under the logic isolation principle. This figure shows that these users can only access the datasets stored in their own MinIO workspaces, respecting the data isolated principle needed in Federated Learning:

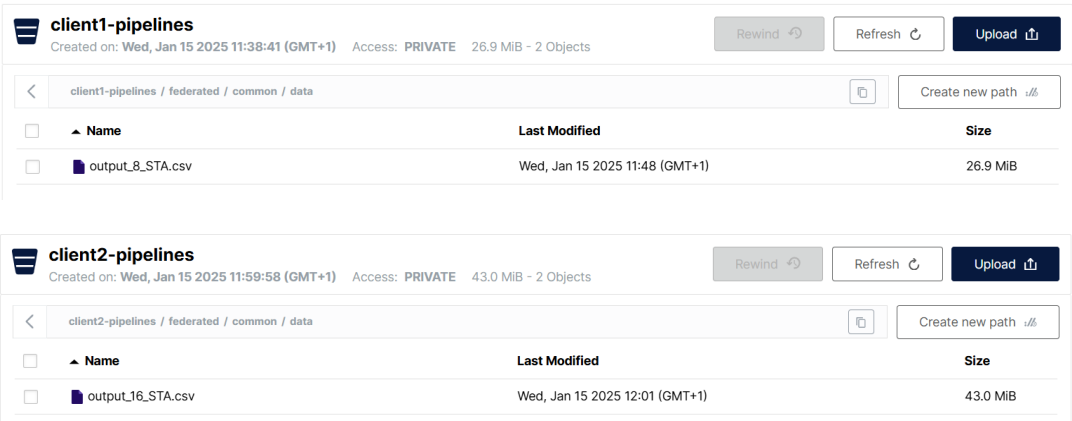


Figure 3-17. MinIO view showcasing datasets isolation

Test 3: Model Aggregation (Federated Learning)

Description

This test focuses on validating the extension performed over the MLOps framework to support the model aggregation operation performed in a Federated Learning environment. This test reuses the isolated user environment generated in Test 2, where two different clients leverage the data isolation principle to train a Keras Neural Network model with different data inputs. Once the models are trained locally, the weights of each model will be shared through a specific number of interactions between the clients and an aggregation server as part of the model aggregation process.

The test has been performed over a Kubeflow environment leveraging Flower framework for performing the model aggregation.

The expected output of the test is the generation of an aggregated Federated Learning model obtained because of the interaction between 2 clients and one aggregation server in a distributed environment based on Flower.

Execution steps

1. **Data ingestion, preprocessing and model initialization:** Each user will load, preprocess and structure their own dataset for using it as input for training the model. In each user, the same DNN model will be initialized, setting the number of

layers, learning rate, metrics and optimizer parameters. The data structured and the model initialization will be used to create a Flower Client for each user.

2. **Flower Server initialization:** To perform the model aggregation, an aggregation server needs to be implemented in the context of Flower framework. The Flower Server interacts with the different clients for aggregating the weights of the local models towards the generation of an aggregated model. The server specifies the number of clients, the number of rounds during which the weights are exchanged with the clients, as well as the strategy used to aggregate these metrics. Once initialized, the server needs to be started.
3. **Flower Clients initialization:** Once the Flower server is up and running, and each client has structured the data and initialized their models, each of the Flower Clients will be initialized and executed.
4. **Model Aggregation and Storage:** To perform the model aggregation a set of iterations (rounds) is performed between the clients and the server to aggregate the weights of the different models based on the aggregation strategy defined in the Flower server. Once the number of rounds is reached, the final aggregated model will be ready to be stored in the Model Storage module and served (Test 4).

Results

Figure 3-18 shows the model aggregation process performed with 2 Flower clients and 10 rounds of iterations between the clients and the Flower server. The metrics shown during the aggregation are loss and accuracy:

```
16/16 ----- 0s 3ms/step - accuracy: 0.7810 - loss: 4.5990
INFO flwr 2025-02-26 16:21:02,535 | server.py:125 | fit progress: (5, 8.602290153503418, {'accuracy': 0.6179999789129333}, 42.38734230400587)
DEBUG flwr 2025-02-26 16:21:02,535 | server.py:173 | evaluate_round 5: strategy sampled 2 clients (out of 2)
DEBUG flwr 2025-02-26 16:21:02,653 | server.py:187 | evaluate_round 5 received 2 results and 0 failures
DEBUG flwr 2025-02-26 16:21:02,654 | server.py:222 | fit_round 5: strategy sampled 2 clients (out of 2)
DEBUG flwr 2025-02-26 16:21:03,931 | server.py:236 | fit_round 6 received 2 results and 0 failures
16/16 ----- 0s 3ms/step - accuracy: 0.7810 - loss: 5.3705
INFO flwr 2025-02-26 16:21:04,072 | server.py:125 | fit progress: (6, 10.164799690246582, {'accuracy': 0.6179999789129333}, 43.92498385000481)
DEBUG flwr 2025-02-26 16:21:04,073 | server.py:173 | evaluate_round 6: strategy sampled 2 clients (out of 2)
DEBUG flwr 2025-02-26 16:21:04,184 | server.py:187 | evaluate_round 6 received 2 results and 0 failures
DEBUG flwr 2025-02-26 16:21:04,184 | server.py:222 | fit_round 7: strategy sampled 2 clients (out of 2)
DEBUG flwr 2025-02-26 16:21:05,015 | server.py:236 | fit_round 7 received 2 results and 0 failures
16/16 ----- 0s 3ms/step - accuracy: 0.7810 - loss: 5.8823
INFO flwr 2025-02-26 16:21:05,761 | server.py:125 | fit progress: (7, 11.291449546813965, {'accuracy': 0.6179999789129333}, 45.614865404006396)
DEBUG flwr 2025-02-26 16:21:05,762 | server.py:173 | evaluate_round 7: strategy sampled 2 clients (out of 2)
DEBUG flwr 2025-02-26 16:21:05,910 | server.py:187 | evaluate_round 7 received 2 results and 0 failures
DEBUG flwr 2025-02-26 16:21:05,910 | server.py:222 | fit_round 8: strategy sampled 2 clients (out of 2)
DEBUG flwr 2025-02-26 16:21:07,259 | server.py:236 | fit_round 8 received 2 results and 0 failures
16/16 ----- 0s 3ms/step - accuracy: 0.7810 - loss: 6.9240
INFO flwr 2025-02-26 16:21:07,393 | server.py:125 | fit progress: (8, 13.212478637695312, {'accuracy': 0.6179999789129333}, 47.24577488900104)
DEBUG flwr 2025-02-26 16:21:07,394 | server.py:173 | evaluate_round 8: strategy sampled 2 clients (out of 2)
DEBUG flwr 2025-02-26 16:21:07,523 | server.py:187 | evaluate_round 8 received 2 results and 0 failures
DEBUG flwr 2025-02-26 16:21:07,523 | server.py:222 | fit_round 9: strategy sampled 2 clients (out of 2)
DEBUG flwr 2025-02-26 16:21:08,760 | server.py:236 | fit_round 9 received 2 results and 0 failures
16/16 ----- 0s 3ms/step - accuracy: 0.7810 - loss: 6.9648
INFO flwr 2025-02-26 16:21:08,915 | server.py:125 | fit progress: (9, 13.354443556109063, {'accuracy': 0.6179999789129333}, 48.760839522001345)
DEBUG flwr 2025-02-26 16:21:08,916 | server.py:173 | evaluate_round 9: strategy sampled 2 clients (out of 2)
DEBUG flwr 2025-02-26 16:21:09,025 | server.py:187 | evaluate_round 9 received 2 results and 0 failures
DEBUG flwr 2025-02-26 16:21:09,026 | server.py:222 | fit_round 10: strategy sampled 2 clients (out of 2)
DEBUG flwr 2025-02-26 16:21:10,287 | server.py:236 | fit_round 10 received 2 results and 0 failures
Saving global model after round 10
```



```

FL_Test INFO flwr 2025-02-18 12:41:01,354 app.py:206 | Starting Flower server, config: num_rounds=10, no round_timeout
FL_Test INFO flwr 2025-02-18 12:41:01,377 app.py:219 | Flower ECE: grpc server running (10 rounds), SSL is disabled
FL_Test INFO flwr 2025-02-18 12:41:01,378 server.py:92 | [INIT]
FL_Test INFO flwr 2025-02-18 12:41:01,378 server.py:281 | Requesting initial parameters from one random client
FL_Test INFO flwr 2025-02-18 12:41:11,448 server.py:288 | Received initial parameters from one random client
FL_Test INFO flwr 2025-02-18 12:41:11,449 server.py:94 | Starting evaluation of initial global parameters
FL_Test INFO flwr 2025-02-18 12:41:12,089 server.py:97 | Initial parameters (loss, other metrics): 21.285385131835938, {'accuracy': 0.0820000022649765}
FL_Test INFO flwr 2025-02-18 12:41:12,089 server.py:113 | [ROUND 1]
FL_Test INFO flwr 2025-02-18 12:41:14,485 server.py:226 | configure_fit: strategy sampled 2 clients (out of 2)
FL_Test INFO flwr 2025-02-18 12:41:18,101 server.py:240 | aggregate_fit: received 2 results and 0 failures
FL_Test WARNING flwr 2025-02-18 12:41:18,104 fedavg.py:252 | No fit_metrics_aggregation_fn provided
FL_Test INFO flwr 2025-02-18 12:41:18,244 server.py:131 | fit progress: (1, 5.193828582763672, {'accuracy': 0.6179999709129333}, 6.154819502999089)
FL_Test INFO flwr 2025-02-18 12:41:18,244 server.py:178 | configure_evaluate: strategy sampled 2 clients (out of 2)
FL_Test INFO flwr 2025-02-18 12:41:18,546 server.py:192 | aggregate_evaluate: received 2 results and 0 failures
FL_Test WARNING flwr 2025-02-18 12:41:18,546 fedavg.py:283 | No evaluate_metrics_aggregation_fn provided
FL_Test INFO flwr 2025-02-18 12:41:18,546 server.py:112 | [ROUND 2]
FL_Test INFO flwr 2025-02-18 12:41:18,547 server.py:113 | configure_fit: strategy sampled 2 clients (out of 2)
FL_Test INFO flwr 2025-02-18 12:41:19,767 server.py:240 | aggregate_fit: received 2 results and 0 failures
FL_Test INFO flwr 2025-02-18 12:41:19,931 server.py:131 | fit progress: (2, 4.419455051422119, {'accuracy': 0.6179999709129333}, 7.841761724999742)
FL_Test INFO flwr 2025-02-18 12:41:19,931 server.py:178 | configure_evaluate: strategy sampled 2 clients (out of 2)
FL_Test INFO flwr 2025-02-18 12:41:20,063 server.py:192 | aggregate_evaluate: received 2 results and 0 failures
FL_Test INFO flwr 2025-02-18 12:41:20,063 server.py:112 | [ROUND 3]
FL_Test INFO flwr 2025-02-18 12:41:20,064 server.py:113 | configure_fit: strategy sampled 2 clients (out of 2)
FL_Test INFO flwr 2025-02-18 12:41:21,672 server.py:240 | aggregate_fit: received 2 results and 0 failures
FL_Test INFO flwr 2025-02-18 12:41:21,864 server.py:131 | fit progress: (3, 4.096471691131592, {'accuracy': 0.6179999709129333}, 9.774912048998885)
FL_Test INFO flwr 2025-02-18 12:41:21,865 server.py:178 | configure_evaluate: strategy sampled 2 clients (out of 2)
FL_Test INFO flwr 2025-02-18 12:41:22,015 server.py:192 | aggregate_evaluate: received 2 results and 0 failures
FL_Test INFO flwr 2025-02-18 12:41:22,016 server.py:112 | [ROUND 4]
FL_Test INFO flwr 2025-02-18 12:41:22,016 server.py:113 | configure_fit: strategy sampled 2 clients (out of 2)
FL_Test INFO flwr 2025-02-18 12:41:22,016 server.py:226 | configure_fit: strategy sampled 2 clients (out of 2)
FL_Test INFO flwr 2025-02-18 12:41:23,134 server.py:240 | aggregate_fit: received 2 results and 0 failures
FL_Test INFO flwr 2025-02-18 12:41:23,264 server.py:131 | fit progress: (4, 5.452873229988469, {'accuracy': 0.6179999709129333}, 11.174563358999876)
FL_Test INFO flwr 2025-02-18 12:41:23,264 server.py:178 | configure_evaluate: strategy sampled 2 clients (out of 2)
FL_Test INFO flwr 2025-02-18 12:41:23,381 server.py:192 | aggregate_evaluate: received 2 results and 0 failures

FL_Test INFO flwr 2025-02-18 12:41:30,797 server.py:113 | [ROUND 10]
FL_Test INFO flwr 2025-02-18 12:41:30,797 server.py:226 | configure_fit: strategy sampled 2 clients (out of 2)
FL_Test INFO flwr 2025-02-18 12:41:32,188 server.py:240 | aggregate_fit: received 2 results and 0 failures
FL_Test INFO flwr 2025-02-18 12:41:32,426 server.py:131 | fit progress: (10, 12.3774528509341797, {'accuracy': 0.6179999709129333}, 20.33649949699975)
FL_Test INFO flwr 2025-02-18 12:41:32,426 server.py:178 | configure_evaluate: strategy sampled 2 clients (out of 2)
FL_Test INFO flwr 2025-02-18 12:41:32,620 server.py:192 | aggregate_evaluate: received 2 results and 0 failures
FL_Test INFO flwr 2025-02-18 12:41:32,621 server.py:496 | [SUMMARY]
FL_Test INFO flwr 2025-02-18 12:41:32,621 server.py:497 | Run finished 10 round(s) in 20.53s
FL_Test INFO flwr 2025-02-18 12:41:32,622 server.py:500 | History (loss, distributed):
FL_Test INFO flwr 2025-02-18 12:41:32,622 server.py:500 |   round 1: 1.6784402440293137
FL_Test INFO flwr 2025-02-18 12:41:32,622 server.py:500 |   round 2: 1.6475240826988112
FL_Test INFO flwr 2025-02-18 12:41:32,622 server.py:500 |   round 3: 1.9335377216339111
FL_Test INFO flwr 2025-02-18 12:41:32,622 server.py:500 |   round 4: 2.2827556282281876
FL_Test INFO flwr 2025-02-18 12:41:32,622 server.py:500 |   round 5: 2.505874276161194
FL_Test INFO flwr 2025-02-18 12:41:32,623 server.py:500 |   round 6: 2.749951049685478
FL_Test INFO flwr 2025-02-18 12:41:32,623 server.py:500 |   round 7: 2.9120757430791855
FL_Test INFO flwr 2025-02-18 12:41:32,624 server.py:500 |   round 8: 3.4622561931610107
FL_Test INFO flwr 2025-02-18 12:41:32,624 server.py:500 |   round 9: 3.675622910261154
FL_Test INFO flwr 2025-02-18 12:41:32,624 server.py:500 |   round 10: 3.6943067014217377
FL_Test INFO flwr 2025-02-18 12:41:32,625 server.py:500 | History (loss, centralized):
FL_Test INFO flwr 2025-02-18 12:41:32,625 server.py:500 |   round 0: 21.285385131835938
FL_Test INFO flwr 2025-02-18 12:41:32,625 server.py:500 |   round 1: 5.193828582763672
FL_Test INFO flwr 2025-02-18 12:41:32,627 server.py:500 |   round 2: 4.419455051422119
FL_Test INFO flwr 2025-02-18 12:41:32,628 server.py:500 |   round 3: 4.096471691131592
FL_Test INFO flwr 2025-02-18 12:41:32,628 server.py:500 |   round 4: 5.452873229988469
FL_Test INFO flwr 2025-02-18 12:41:32,628 server.py:500 |   round 5: 6.32213838047876
FL_Test INFO flwr 2025-02-18 12:41:32,629 server.py:500 |   round 6: 6.66124153137207
FL_Test INFO flwr 2025-02-18 12:41:32,629 server.py:500 |   round 7: 8.835464477539062
FL_Test INFO flwr 2025-02-18 12:41:32,629 server.py:500 |   round 8: 10.888349609375
FL_Test INFO flwr 2025-02-18 12:41:32,629 server.py:500 |   round 9: 11.554642677387129
FL_Test INFO flwr 2025-02-18 12:41:32,630 server.py:500 |   round 10: 12.3774528509341797
FL_Test INFO flwr 2025-02-18 12:41:32,630 server.py:500 | History (metrics, centralized):
FL_Test INFO flwr 2025-02-18 12:41:32,630 server.py:500 |   {'accuracy': [(0, 0.0820000022649765),
FL_Test INFO flwr 2025-02-18 12:41:32,630 server.py:500 |   (1, 0.6179999709129333),
FL_Test INFO flwr 2025-02-18 12:41:32,631 server.py:500 |   (2, 0.6179999709129333),
FL_Test INFO flwr 2025-02-18 12:41:32,631 server.py:500 |   (3, 0.6179999709129333),
FL_Test INFO flwr 2025-02-18 12:41:32,631 server.py:500 |   (4, 0.6179999709129333),
FL_Test INFO flwr 2025-02-18 12:41:32,631 server.py:500 |   (5, 0.620000047683716),
FL_Test INFO flwr 2025-02-18 12:41:32,631 server.py:500 |   (6, 0.620000047683716),
FL_Test INFO flwr 2025-02-18 12:41:32,632 server.py:500 |   (7, 0.6179999709129333),
FL_Test INFO flwr 2025-02-18 12:41:32,632 server.py:500 |   (8, 0.6179999709129333),
FL_Test INFO flwr 2025-02-18 12:41:32,632 server.py:500 |   (9, 0.6179999709129333),
FL_Test INFO flwr 2025-02-18 12:41:32,632 server.py:500 |   (10, 0.6179999709129333)]}]

```

Figure 3-18. Flower IA model aggregation process

Test 4: Model Serving (Centralized and Federated)

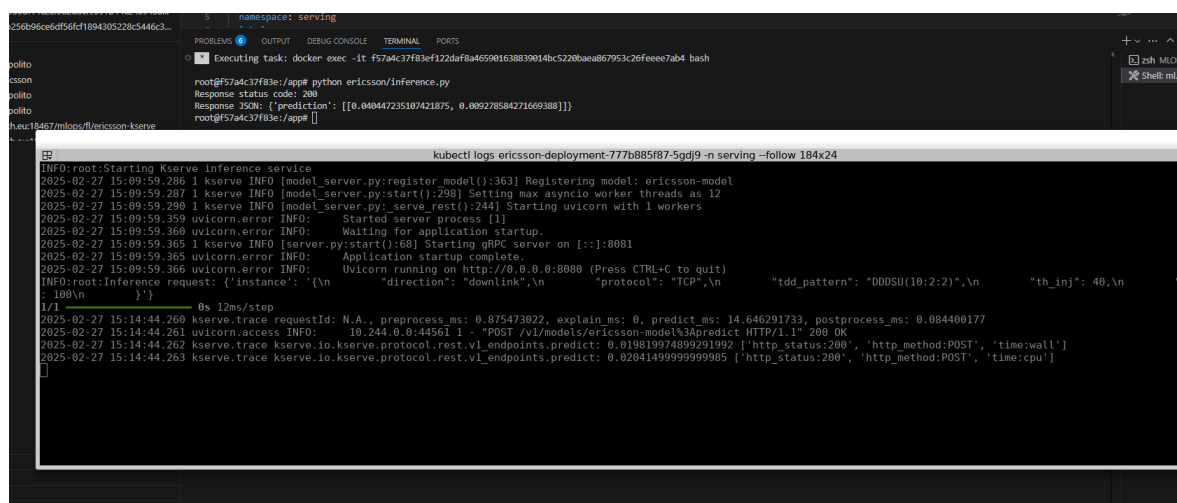
Description

This test validates the implementation of the model serving component in both centralized and federated learning architectures as part of the MLOps framework. The objective of this component is exposing AI/ML model as a service capability so external users can perform inferences over the models generated in Tests 1 and 3. The test has been performed over a Kubeflow environment leveraging Kserve for serving the models and exposing services with endpoints where users can request model inferences.

Execution steps

1. **Model Storage:** Ensure that the AI/ML models to be served and exposed are stored in the Model Storage module. If not, a Kubeflow pipeline should be generated including the archiver stage for pushing the model to the Model Storage based on MinIO.
2. **Model Serving:** With the models available in MinIO, a Python application will be created to load the model and serve it through a Kserve service in a specific endpoint. The application can be exposed as a Kserve InferenceService in Kubeflow or as a simple Kubernetes service to allow external users to perform model inferences. It includes loading, preprocessing and predict functions to load the model, preprocess the inference data and perform the model prediction based on the data received.
3. **Inference Validation:** To test the model serving, an inference request is sent to the endpoint `/v1/models/example:predict`. The inference data is sent in the body's request. The response contains the result of the prediction in the predictions field.

Results



```

namespace: serving
25b096cedd756f1c94305228c5446c3...
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Executing task: docker exec -it f57b4c37f83ef122daf8a465981638839014bc5228bae0b7953c26feee7ab4 bash
root@f57b4c37f83e:/app# python ericsson/inference.py
Response status code: 200
Response 350k: {'prediction': [[0.040447235107421875, 0.009278584271669388]]}
root@f57b4c37f83e:/app#

kubectll logs ericsson-deployment-777b885f87-5gjd9 -n serving --follow 184x24
INFO:root:Starting Kserve inference service
2025-02-27 15:09:59.286 1 kserve INFO [model_server.py:register_model():363] Registering model: ericsson-model
2025-02-27 15:09:59.287 1 kserve INFO [model_server.py:start():298] Setting max asyncio worker threads as 12
2025-02-27 15:09:59.298 1 kserve INFO [model_server.py:serve_rest():244] Starting uvicorn with 1 workers
2025-02-27 15:09:59.359 uvicorn.error INFO: Started server process [1]
2025-02-27 15:09:59.368 uvicorn.error INFO: Waiting for application startup.
2025-02-27 15:09:59.365 1 kserve INFO [server.py:start():68] Starting gRPC server on [::]:8081
2025-02-27 15:09:59.365 uvicorn.error INFO: Application startup complete.
2025-02-27 15:09:59.366 uvicorn.error INFO: Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C to quit)
INFO:root:Inference request: {'instance': '{\n  "direction": "downlink",\n  "protocol": "TCP",\n  "tdd_pattern": "D00SU(10:2:2)",\n  "th_inj": 40,\n  "bw": 100\n}',\n  'data': '1/1'}
0s 12ms/step
2025-02-27 15:14:44.260 kserve.trace requestId: N.A., preprocess_ms: 0.875473022, explain_ms: 0, predict_ms: 14.646291733, postprocess_ms: 0.084480177
2025-02-27 15:14:44.261 uvicorn.access INFO: 10.244.0.0:44561 1 - "POST /v1/models/ericsson-models3predict HTTP/1.1" 200 OK
2025-02-27 15:14:44.262 kserve.trace kserve.io.kserve.protocol.rest.v1_endpoints.predict: 0.019019974899291992 ['http_status:200', 'http_method:POST', 'time:wall']
2025-02-27 15:14:44.263 kserve.trace kserve.io.kserve.protocol.rest.v1_endpoints.predict: 0.020414999999999985 ['http_status:200', 'http_method:POST', 'time:cpu']

```

AI/ML-assisted control applications

This section illustrates different examples of the application of AI/ML techniques and algorithms in support of deterministic service provisioning. Such mechanisms are aimed to be implemented by means of the above-described MLOps framework.

AI-based resource allocation: Wi-Fi OFDMA slicing

In this subsection, the first proposed application of the AI-assisted control plane is illustrated. In particular, a reference reinforcement learning environment that simulates the Wi-Fi network to deliver a testable open-source AI slicing algorithm as the internal development of the algorithm was done with a proprietary simulator. This reference environment enables any user of the AI algorithm to train a slicing model out-of-the-box and have a working example of how to integrate any simulator or real system with the algorithm.

Both the AI algorithm and the reference environment are released as open source in the repository of the project (section 1.2). The AI algorithm approach, which was described formally in (PREDICT-6G/D3.3, 2025), is elaborated in the next paragraphs.

The goal of the AI-assisted control application consists in finding a model that performs a slicing decision to partition the available Wi-Fi radio resources into N different slices, such that traffic with different QoS requirements can be assigned to a slice that will allocate the appropriate number of resources as to fulfil the QoS requirements. The model will make its decisions dynamically and based on the current and historical system state.

For Wi-Fi particularly, a centralized schedule of a Wi-Fi network using OFDMA enables the access point (AP) to decide at each transmission opportunity how many frequencies resource units (RUs) to use for a peer-to-peer download or upload transmission. The slicing problem to solve then consists in aggregating RU's for a particular class of traffic belonging to a specified slice.

The proposed AI algorithm applies a variant of constrained reinforcement learning that extends the monitored system state with the Lagrangian multipliers used to combine the QoS constraints in the main objective function. This enables to create a system that can support more than one optimal solution based on the current violation of constraints.

The proposed algorithm consistently learns to maximize the objective function (total received bytes) as seen in Figure 3-19, where the X axis corresponds to the training step, which has been averaged by $\times 1000$ and the Y axis corresponds to the total received bytes. The model has been trained from scratch in 10 different instances using the presented simulation environment with a periodic traffic scenario.

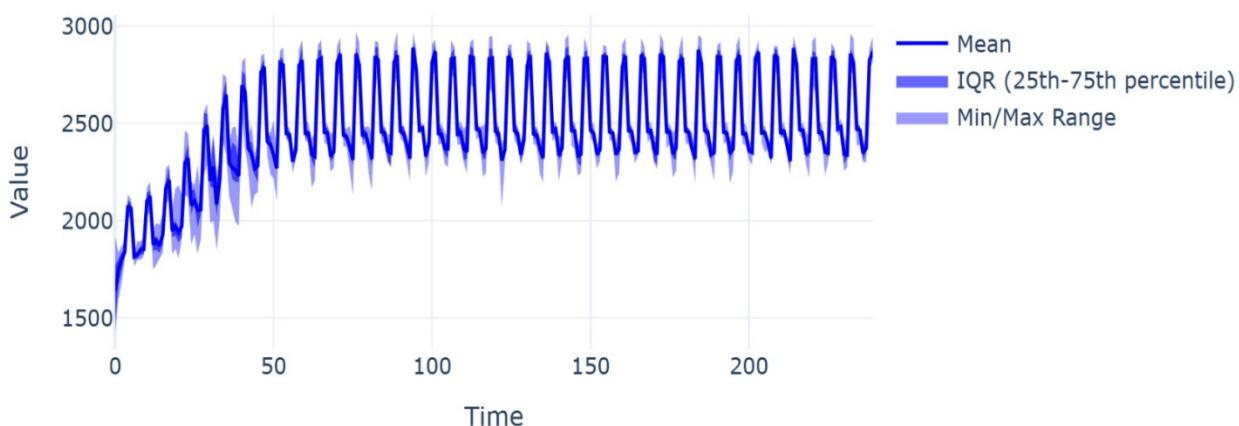


Figure 3-19. Training convergence of the RL policy for 10 different training instances in a periodic traffic scenario.

Figure 3-20 illustrates the evaluation of the proposed mechanism. The first graph (left) shows the traffic demand (in packets) simulated for 3 slices, periodically alternating. The second one shows the slicing decisions (in percentage) made by the algorithm. The last two graphs show the resulting reward and constraint signals which correspond to total received bytes and average latency penalty, respectively. The latency penalty is computed as the average packet latency plus a penalty of 100 assigned to every non-delivered packet. For all the graphs the X axis corresponds to the RL environment time step.

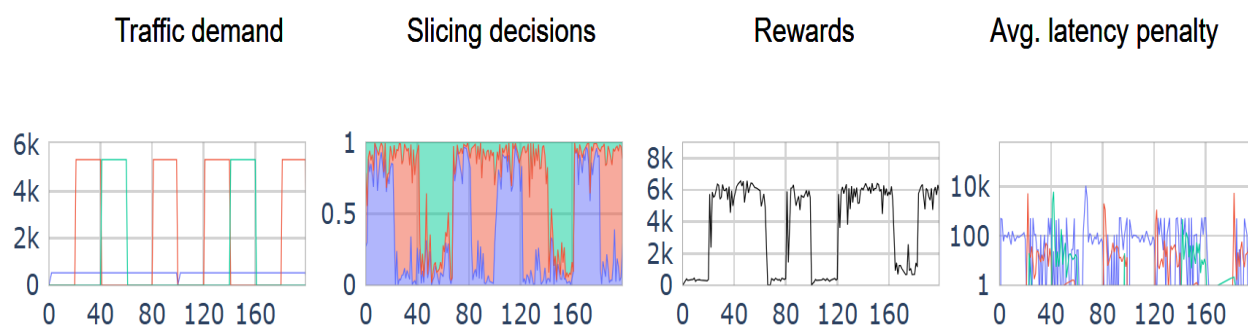


Figure 3-20. Evaluation of trained AI algorithm on a proprietary ns3-based Wi-Fi 7 simulator.

Path redundancy mechanism in support of reliability

A second application proposed as part of AI-Based Predictive & DSS Support service is based on the topology and consists in a Path Redundancy Mechanism (PRM) that dynamically adjusts to network conditions using real-time monitoring.

Aiming at the integration of the mechanism with the Access and Mobility Management Function (AMF), changes in AMF are anticipated to access the new services. The following key functionalities are available:

- Extended AMF to exchange services with the Path Redundancy Service.
- Continuously monitoring of network performance and computes alternative paths.
- Caching the best redundant paths to improve response time.

Step 1: Create network topology

- Nodes: Routers, Switches, AMFs, etc.
- Edges: Network Links
- Weights: Latency, Bandwidth, Reliability

Step 2: AMF Uploads Monitoring Data

- The AMF collects real-time network monitoring data (e.g., congestion, link failures, jitter).
- It uploads this data to the Path Redundancy Mechanism.
- AI-assisted Graph Pruning removes cycles and congested/faulty links before path computation

Step 3: Path Redundancy Mechanism Computes Optimal Paths

- The mechanism analyzes monitored data and uses Graph Theory to get the best paths. The Dijkstra's Algorithm is used for path computation considering lowest-latency path.
- It precomputes and caches alternative paths based on:
 - Network resilience (e.g., avoiding failure-prone links).
 - Latency and bandwidth constraints.
 - Traffic load balancing.
- Alternative paths will be stored in a cache mechanism (Redis) for quick retrieval by the AMF. This minimizes path computation latency by caching optimal routes.

- Periodic Re-Evaluation updates paths based on real-time data.

Step 4: AMF Requests the Best Path

- When a path is needed, AMF queries the Path Redundancy Mechanism.
- The mechanism retrieves and returns from the cache the best available paths. The mechanism is able to return up to three alternative paths (primary, secondary, and failover).
- If no cached path meets the requirements, a new path is computed on-demand.

The technologies used are as follows:

- PRM services are exposed as REST APIs.
- Paths Computations are developed in Python using Network library
- Cache mechanism uses Redis

Figure 3-21 contextualizes the PRM as a support mechanism for the E2E Path computation MS.

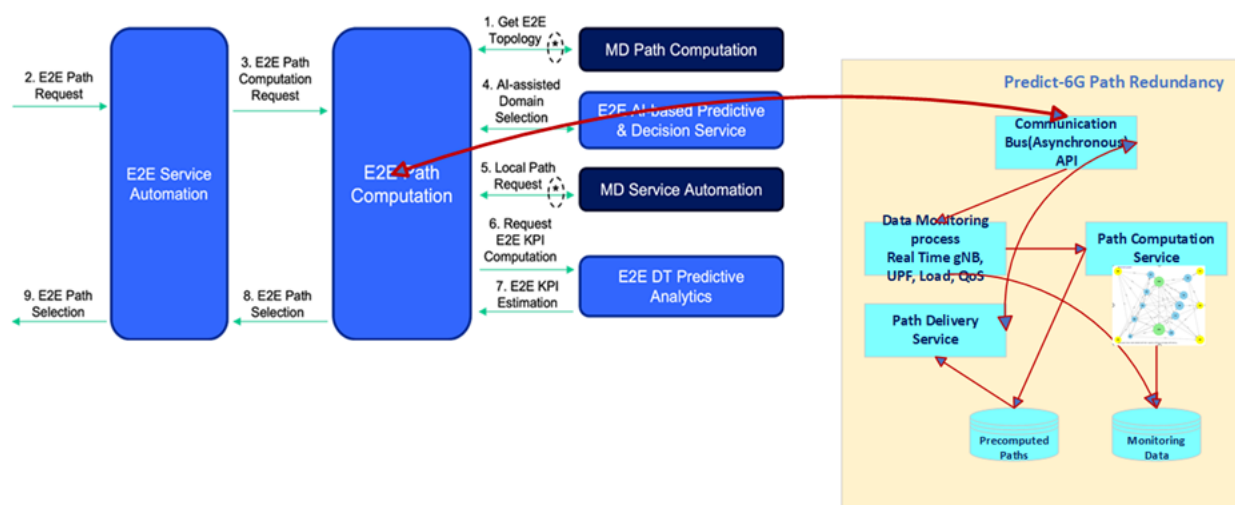


Figure 3-21. PRM in support of the E2E Path computation mechanism

4 Operational workflows validation

This chapter reports on the integration of the different AICP components to implement the operational workflows, which were defined in (PREDICT-6G/D3.2, 2023) and updated in (PREDICT-6G/D3.3, 2025).

It is worth noting here that the operational workflows are used as a driver to showcase control plane integration, hence, the data plane configuration is not reported in this deliverable.

4.1 E2E Exposure and Deterministic Service Provisioning

Figure 4-1 revisits the E2E service provisioning workflow described in (PREDICT-6G/D3.3, 2025). to provide its implementation view. This means that the functionalities and the process remain the same but the interactions between the actual implemented modules have been depicted. Note that the workflow also illustrates the exposure process that has been explained and validated in section 2.3, since it is necessary to understand the operation of the provisioning phase.

As depicted in the figure, just after its boot-up, the MD Path Computation (MD-PCE) of each local technological domain contacts the Resource Configurator (RC) to request the data plane topology, resources and capabilities (step 1). Afterwards, each MD-PCE composes the abstracted view of the data plane topology, which will be exposed (under request) to the E2E MSs. In this regard, during its boot-up process, the E2E Path Computation (E2E-PCE) contacts the MD-PCEs of the MD it oversees to request the mentioned abstract views of the technological domains (step 2). Once the E2E-PCE has collected all the single abstract topologies, it composes the E2E multi-domain abstract topology that it will use for the domain selection process (step 3), which is needed for the multi-domain E2E path computation process that takes part of the E2E deterministic service provisioning workflow.

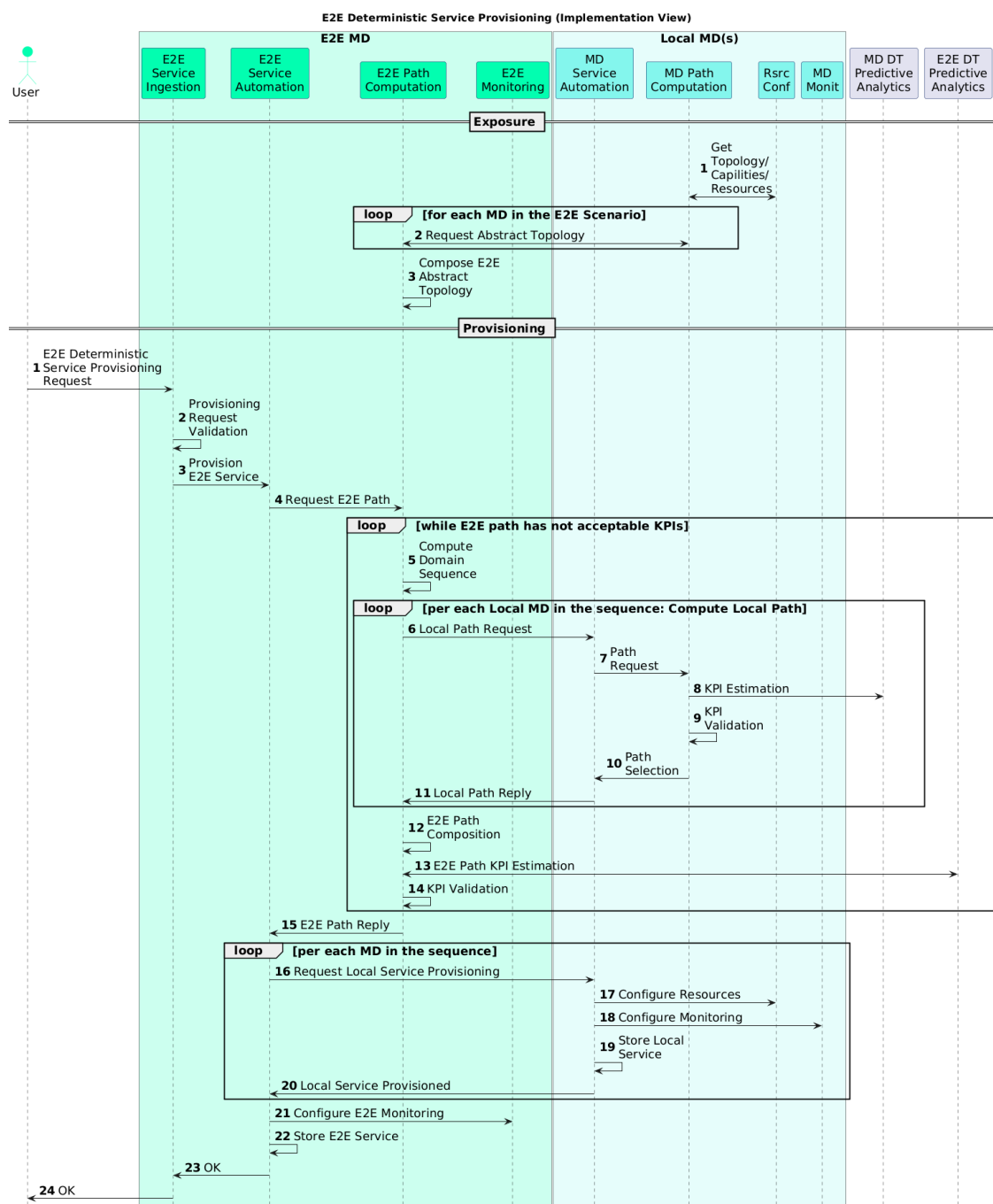


Figure 4-1. Implementation view of the E2E deterministic service provisioning workflow

The provisioning phase starts with the interaction between the client and the service ingestion to request the E2E deterministic service (step 1 in Figure 4-1). This process' details and validation are reported in section 2.2. Once the request has been validated by the E2E Service Ingestion MS, it requests the provisioning of the service to the E2E Service Automation (Figure 4-2).

```
DEBUG:root:Sending the E2E service provisioning request to E2E Service Automation for processing:
{
  "endpoints": {
    "source": {
      "id": "PC_Talker"
    },
    "destination": {
      "id": "PC_Listener"
    }
  },
  "qos_characteristics": {
    "priority": 7,
    "reliability": 100,
    "packet_loss": 10,
    "delay": 20,
    "rtt": 30,
    "jitter": 10,
    "burst_arrival_time_window": {
      "t1": 0,
      "t2": 5
    },
    "burst_completion_time_window": {
      "t1": 0,
      "t2": 5
    }
  },
  "traffic_characteristics": {
    "direction": "directional",
    "periodicity": true,
    "period": 20,
    "burst_size": 500,
    "maximum_flow_bitrate": 5000
  },
  "service_lifetime": {
    "service_duration": {
      "t1": 0,
      "t2": 2
    },
    "recurrent_service_interval": {
      "recurrence_interval": 1,
      "t1": 0,
      "t2": 1
    }
  }
}
DEBUG:urllib3.connectionpool:Starting new HTTP connection (1): 10.5.15.60:80
DEBUG:urllib3.connectionpool:http://10.5.15.60:80 "POST /e2erequestprocessing/e2eprovisioningrequest HTTP/1.1" 200 7
DEBUG:root:E2E SERVICE ID: '33382'
```

Figure 4-2. Forwarding of E2E service provisioning request from E2E Service Ingestion to E2E Service Automation

Upon the reception of the provisioning request, the E2E Service Automation contacts the E2E Path Computation (step 4), which computes the E2E path over the abstract topology and, in turn, contacts the involved local MD Service Automation components to request the local paths for each local domain (steps 5 and 6, the process is showcased in Figure 4-3).


```
e.u.g.p.p.core.pce.PathComputation : Computed E2E Path
e.u.g.p.p.core.pce.PathComputation : PC_Talker - PC_Listener
e.u.g.p.p.core.pce.PathComputation : -----
e.u.g.p.p.core.pce.PathComputation : currentLink: PC_Talker - PC_Listener
e.u.g.p.p.core.pce.PathComputation : Positioning pathIt2
e.u.g.p.p.core.pce.PathComputation : E2E Path for a Single Domain path request?
e.u.g.p.p.core.pce.PathComputation : Domain Sequence:
e.u.g.p.p.core.pce.PathComputation : Treating Domain: mdTsn
e.u.g.p.p.core.pce.PathComputation : TdNodes PC_Talker --> PC_Listener
e.u.g.p.p.core.sa.SARestClient : Sendign Local Path Computation Request to MD Service Automation
e.u.g.p.p.core.sa.SARestClient : Reply = <200 OK OK,"Path Computation (1125) in progress",[Date:"Mon, 27 Jan 2025 16:05:43 GMT", Cor

e.u.g.p.p.core.pce.PathComputation : -----
e.u.g.p.p.nbi.server.NBIPceService : Processing Local Path for MD Service 1125 (10.5.15.61)
e.u.g.p.p.nbi.server.NBIPceService : Processing Local Path TempServiceId0
e.u.g.p.p.nbi.server.NBIPceService : Sending E2E Path Reply to E2E-SA: E2EPathReply [id=33382, status=pending, endPoints=EndPoints [sou
trafficCharacteristics=TrafficCharacteristics [direction=directional, periodicity=true, period=20, burstSize=500, maxFlowBitrate=5000], serv
t2=1, recurrenceInterval=1]], mdServices=MdServices [localService=LocalServiceEntry [index=0, service=Service [id=1125, serviceStatus=pending
e.u.g.p.p.core.sa.SARestClient : Sending Path Computation Result to E2E Service Automation
e.u.g.p.p.core.sa.SARestClient : Reply = <200 OK OK,{ "message": "Requested provisioning of E2E service with ID 33382 successfully."}
```

Figure 4-3. E2E-PCE log output for domain sequence computation over the abstract topology

Each MD Service Automation coordinates the local domain path computation by contacting the corresponding MD Path Computation element (step 7, showcased in Figure 4-4).

```
2025-01-27T16:05:43.116 [INFO]rest_app|postpathrequest()|rest_app.py:46|Received request to forward to Path Computation service.
2025-01-27T16:05:43.203 [INFO]rest_manager|addMDServiceID()|rest_manager.py:95|1124
2025-01-27T16:05:43.203 [INFO]rest_manager|addMDServiceID()|rest_manager.py:99|1125
2025-01-27T16:05:43.204 [INFO]rest_app|postpathrequest()|rest_app.py:54|MD ID Service Created: 1125
2025-01-27T16:05:43.279 [INFO]rest_app|postpathrequest()|rest_app.py:58|Response from Path Computation:Path Computation (1125) in progress
INFO: 10.1.194.4:44088 - "POST /path/mdservice/pathrequest HTTP/1.1" 200 OK
2025-01-27T16:05:43.531 [INFO]rest_app|postpathresponse()|rest_app.py:84|Response from E2E Path Computation service:Local Path Processing (1125) in progress
INFO: 10.1.194.4:42184 - "POST /path/mdservice/pathresponse HTTP/1.1" 200 OK
```

Figure 4-4. Request and response result of the Path Computation process in the MD Service Automation for a MD Service in TSN domain

The MD Path Computation calculates the route over the complete data plane topology of the technological domain and, if available, request the KPI estimation to the domain DT (steps 8 and 9). Figure 4-5 showcases the TSN domain with the DT reported in section 3.1.

Figure 4-5. MD-PCE log output illustrating local path computation and KPI estimation request to the DT

Figure 4-6. E2E-PCE log output that illustrates the E2E KPI estimation request and reply from the E2E DT



```
2025-01-27T16:05:43.145 INFO|rest_app|postpathresponse()|rest_app.py:32|Processing E2E Path Computation response for Service ID: 33382
2025-02-27T16:05:43.206 INFO|rest_app|postpathresponse()|rest_app.py:46|Successfully forwarded Service ID 33382 to E2E Lifecycle Management.
INFO: 10.1.118.198:39122 - "POST /e2epathprocessing/e2eservice/33382/e2epathresponse HTTP/1.1" 200 OK
```

Figure 4-7. E2E Path Computation response received at the E2E Service Automation during the E2E Service Provisioning

The service configuration starts when the E2E Service Automation contacts each local domain involved in the E2E service to request the local service configuration (step 16, showcased in Figure 4-8).

```
2025-01-27T16:05:43.413 INFO|rest_app|provisionService()|rest_app.py:38|Received request to provision E2E service with ID 33382.
2025-01-27T16:05:43.516 INFO|rest_app|provisionService()|rest_app.py:49|Requested provisioning of E2E service with ID 33382 successfully.
INFO: 10.1.118.254:34000 - "POST /provisione2eservice HTTP/1.1" 200 OK
```

Figure 4-8. Forwarding of E2E service provisioning request from E2E Service Automation to MD Service Automation

Then, the MD Service Automation of the local technological domain coordinates the data plane configuration for the service (step 17, Figure 4-9 and Figure 4-10), as well as the monitoring and data collection (step 18, Figure 4-11).

```
2025-01-27T16:05:43.453 INFO|rest_app|provisionService()|rest_app.py:43|Received request to provision MD service with ID 1125.
2025-01-27T16:05:43.476 INFO|rest_app|provisionService()|rest_app.py:54|MD service with ID 1125 requested provisioning successfully.
INFO: 10.1.194.4:38302 - "POST /lifecycle/provisionmdservicerequest HTTP/1.1" 200 OK
```

Figure 4-9. Request result of the Lifecycle Management in the MD Service Automation for the provisioning of a MD Service MD Service in TSN domain, leading to contact the Resource Configurator

```
2025-01-27T16:05:48.825 INFO|rest_app|assignmgmt()|rest_app.py:85|Assigning management for MD service with ID: 1125
2025-01-27T16:05:49.263 INFO|rest_manager|triggerMonitoring()|rest_manager.py:206|Monitoring triggered successfully.
2025-01-27T16:05:49.263 INFO|rest_manager|notifyDT()|rest_manager.py:273|Notifying MD DT for service ID 1125.
2025-01-27T16:05:49.326 INFO|rest_manager|notifyDT()|rest_manager.py:276|Notification sent successfully to DT. Response: 200
2025-01-27T16:05:49.919 INFO|rest_app|assignmgmt()|rest_app.py:99|Service 1125 successfully provisioned, monitoring triggered and DT notified.
INFO: 10.1.194.4:38306 - "PUT /lifecycle/provisionmdserviceresponse/1125 HTTP/1.1" 200 OK
```

Figure 4-10. Result of the Lifecycle Management in the MD Service Automation during the provisioning phase after receiving the Resource Configurator response in TSN domain

```
DEBUG:connexion.apis.flask_api:Getting data and status code
DEBUG:connexion.decorators.validation:http://10.5.15.64:8888/datasources/MDP validating schema...
DEBUG:connexion.decorators.validation:http://10.5.15.64:8888/datasources/MDP validating parameters...
DEBUG:connexion.decorators.parameter:Function Arguments: ['body', 'type_']
INFO:root:Received request to create datasource of type MDP with the following configuration:
{
  "service_id": "1125",
  "qos_characteristics": {
    "priority": 7,
    "td_reliability": 100,
    "td_packet_loss": 10,
    "td_delay": 20,
    "td_rtt": 30,
    "td_jitter": 10,
    "burst_arrival_time_window": {
      "t1": 0,
      "t2": 5
    },
    "burst_completion_time_window": {
      "t1": 0,
      "t2": 5
    }
  },
  "traffic_characteristics": {
    "direction": "directional",
    "periodicity": true,
    "period": 20,
    "burst_size": 500,
    "maximum_flow_bitrate": 5000
  }
}
```

Figure 4-11. Request of configuration of the monitoring of a technological domain sent by the Service Automation. The monitored KPIs are all the ones for which the service requestor provided a value

Finally, the local service is stored in the Service Exposure module (step 19), which, as detailed in section 2.4, resides inside the Service Automation component. Figure 4-12 depicts the provisioned local service information as returned by the Service Exposure.

```

{
  "qos_characteristics": {
    "td_rtt": 80,
    "priority": 0,
    "td_delay": 20,
    "td_jitter": 10,
    "td_packet_loss": 100,
    "td_reliability": 100,
    "burst_arrival_time_window": {
      "t1": 0,
      "t2": 5
    },
    "burst_completion_time_window": {
      "t1": 0,
      "t2": 5
    },
    "td_nodes": {
      "gateway": {
        "gateway_list": [
          {
            "id": "PC_listener"
          }
        ]
      },
      "endpoint": {
        "id": "PC_talker"
      }
    },
    "id": "1125",
    "service_lifetime": {
      "service_duration": {
        "t1": 0,
        "t2": 2
      },
      "recurrent_service_interval": {
        "t1": 0,
        "t2": 1,
        "recurrence_interval": 1
      }
    },
    "traffic_characteristics": {
      "period": 80,
      "direction": "directional",
      "burst_size": 100,
      "periodicity": true,
      "maximum_flow_bitrate": 5000
    },
    "e2e_service_id": "33382",
    "td_path_selection": {
      "path_id": "1",
      "path_status": "computed",
      "involved_nodes": {
        "node_list": [
          {
            "id": "PC_talker",
            "node_resources": {
              "resource_list": {
                "p1": {
                  "status": "express",
                  "priority": "0"
                }
              }
            }
          }
        ]
      }
    }
  },
  "id": "Rely_Switch1",
  "node_resources": {
    "resource_list": {
      "PORT_0": {
        "status": "express",
        "priority": "0"
      },
      "PORT_1": {
        "status": "express",
        "priority": "0"
      },
      "PORT_2": {
        "status": "express",
        "priority": "0"
      },
      "PORT_3": {
        "status": "express",
        "priority": "0"
      }
    }
  },
  "id": "Rely_Switch2",
  "node_resources": {
    "resource_list": {
      "PORT_0": {
        "status": "express",
        "priority": "0"
      },
      "PORT_1": {
        "status": "express",
        "priority": "0"
      },
      "PORT_2": {
        "status": "express",
        "priority": "0"
      },
      "PORT_3": {
        "status": "express",
        "priority": "0"
      }
    }
  },
  "id": "Rely_Switch3",
  "node_resources": {
    "resource_list": {
      "PORT_0": {
        "status": "express",
        "priority": "0"
      },
      "PORT_1": {
        "status": "express",
        "priority": "0"
      }
    }
  }
}

```

Figure 4-12. Service Exposure information about MD service 1125 provisioned in TSN domain

Once the local service has been provisioned, each MD Service Automation notifies the E2E Service Automation (step 20, Figure 4-13), which, in turn, stores the E2E service in its E2E Service Exposure component and configures the E2E monitoring and data collection in steps 21 and 22 (Figure 4-14).

```

2025-01-27T16:05:49.362|INFO|rest_app|provisionedService()|rest_app.py:88|Received response for provisioned MD service with ID 1125.
2025-01-27T16:05:49.658|INFO|rest_manager|triggerMonitoring()|rest_manager.py:222|E2E Monitoring triggered successfully.
2025-01-27T16:05:49.659|INFO|rest_manager|notifyStatusE2EDT()|rest_manager.py:193|Sending status of the provision to: http://e2epathcomputation-service:9500
/e2e-service
2025-01-27T16:05:49.796|INFO|rest_manager|notifyStatusE2EServiceIngestion()|rest_manager.py:170|Sending status of the provision to: http://10.5.15.65:8300/s
ervice-provisioning-notification
2025-01-27T16:05:49.872|INFO|rest_app|provisionedService()|rest_app.py:99|MD service with ID 1125 provisioned successfully.
INFO: 10.1.118.198:52408 - "POST /e2elifecycle/provisionedmdservice HTTP/1.1" 200 OK

```

Figure 4-13. MD Service Automation response received at the E2E Service Automation

```
DEBUG:connexion.apis.flask_api:Getting data and status code
DEBUG:connexion.decorators.validation:http://10.5.15.65:8888/datasources/E2E validating schema...
DEBUG:connexion.decorators.validation:http://10.5.15.65:8888/datasources/E2E validating parameters...
DEBUG:connexion.decorators.parameter:Function Arguments: ['body', 'type']
INFO:root:Received request to create datasource of type E2E with the following configuration:
{
  "e2e_service_id": "33382",
  "mdservice_ids": [
    "1125"
  ]
}
INFO:root:Neither local services ID list nor datasources ID list is provided. Retrieving all of them from DBs
DEBUG:root:DB DATASOURCES TABLES: ['mdp_datasources', 'las_datasources']
DEBUG:root:ROW CONTENT:[(UUID('c959cc74-efd8-43c7-bde7-e37228834261'),), (UUID('a9436673-17fe-4093-bedb-3799df65344d'),), (UUID('48b269b6-fdb4-47db-97d0-108a875d6a72'),), (UUID('7a5765e0-b25d-485c-9cf2-blad315c230b'),)]
DEBUG:root:ROW CONTENT:[('7',), ('8',), ('22',), ('1125',)]
DEBUG:root:ROW CONTENT:[]
DEBUG:root:ROW CONTENT:[]
DEBUG:root:DATASOURCES ID: [UUID('c959cc74-efd8-43c7-bde7-e37228834261'), UUID('a9436673-17fe-4093-bedb-3799df65344d'), UUID('48b269b6-fdb4-47db-97d0-108a875d6a72'), UUID('7a5765e0-b25d-485c-9cf2-blad315c230b')]
DEBUG:root:SERVICES ID: ['7', '8', '22', '1125']
INFO:root:Final configuration of E2E datasource:
{
  "e2e_service_id": "33382",
  "mdservice_ids": [
    "1125"
  ],
  "local_services_id": [
    "7",
    "8",
    "22",
    "1125"
  ],
  "datasources_id": [
    "c959cc74-efd8-43c7-bde7-e37228834261",
    "a9436673-17fe-4093-bedb-3799df65344d",
    "48b269b6-fdb4-47db-97d0-108a875d6a72",
    "7a5765e0-b25d-485c-9cf2-blad315c230b"
  ]
}
```

Figure 4-14. Request of configuration of the monitoring at the E2E MD level sent by the E2E Service Automation. The E2E service encompasses all the local services previously provisioned

Finally, the E2E Service Automation notifies the E2E Service Ingestion about the successful provisioning (step 23). The E2E Service Ingestion, in turn, informs the client (step 24, showcased in Figure 4-15).

```
DEBUG:connexion.apis.flask_api:Getting data and status code
DEBUG:connexion.decorators.validation:http://10.5.15.65:8380/service-provisioning-notification validating schema...
INFO:root:Received request to notify the user about the outcome of the provisioning request for service with ID 33382
INFO:root:Retrieve the service requests status with the given parameters from the Request Status Register
INFO:root:Query InfluxDB bucket 'request_status_register' for last '1h' of data
INFO:root:Filtering by measurement 'SERVICE_PROVISION'
INFO:root:Filtering by tag information '33382'
DEBUG:root:Full query:
from(bucket:"request_status_register") |> range(start: -1h) |> filter(fn:(r) => r._measurement == "SERVICE_PROVISION") |> filter(fn:(r) => r.information == "33382")
DEBUG:root:Table:
FluxTable() columns: 11, records: 1
DEBUG:root:Record:
FluxRecord() table: 0, {result: 'result', 'table': 0, 'start': datetime.datetime(2025, 1, 27, 15, 5, 49, 818195, tzinfo=tzlocal()), 'stop': datetime.datetime(2025, 1, 27, 16, 5, 49, 818195, tzinfo=tzlocal()), 'time': datetime.datetime(2025, 1, 27, 16, 5, 43, 125769, tzinfo=tzlocal()), '_value': 'In_Progress', '_field': 'status', '_measurement': 'SERVICE_PROVISION', 'information': '33382', 'request_id': 'ef2b2535-1d76-4787-af45-6b67bf08ab60', 'requestor_id': '28ec6a83-52ad-4c39-ae64-f9a6dff228d3'}
DEBUG:root:Query result:
[
  {
    "request_id": "ef2b2535-1d76-4787-af45-6b67bf08ab60",
    "requestor_id": "28ec6a83-52ad-4c39-ae64-f9a6dff228d3",
    "request_type": "SERVICE_PROVISION",
    "status": "In_Progress",
    "information": "33382"
  }
]
INFO:root:Storing the request 'ef2b2535-1d76-4787-af45-6b67bf08ab60' status in the Request Status Register
INFO:root:<ef2b2535-1d76-4787-af45-6b67bf08ab60>, <PROVISION>, <28ec6a83-52ad-4c39-ae64-f9a6dff228d3>, <Accepted>, <33382>
DEBUG:connexion.apis.abstract:Getting data and status code
DEBUG:connexion.apis.abstract:Got framework response
INFO:werkzeug:10.9.0.2 - - [27/Jan/2025 16:05:49] "POST /service-provisioning-notification HTTP/1.1" 200 -
```

Figure 4-15. Notification of the accomplishment of the provisioning of the service sent by the E2E Service Automation to the E2E Service Ingestion and forwarded to the final user

Figure 4-16 and Figure 4-17 show the provisioned E2E service information upon a request to the E2E Service Exposure.

```
{
  "traffic_characteristics": {
    "period": 20,
    "direction": "directional",
    "burst_size": 500,
    "periodicity": true,
    "maximum_flow_bitrate": 5000
  },
  "md_services": {
    "local_service": [
      {
        "index": 0,
        "service": {
          "id": "1125",
          "target_domain": {
            "name": "mdTsn",
            "class": "DetNet"
          },
          "service_status": "provisioned"
        }
      }
    ]
  },
  "id": "33382",
  "e2e_path_selection": {
    "id": "7",
    "domain_list": [
      {
        "path_domains": {
          "name": "mdTsn",
          "class": "DetNet"
        }
      }
    ],
    "path_status": "computed"
  },
}
```

Figure 4-16. E2E Service Exposure response with the MD service over the TSN domain associated to the E2E service 33382

```
"service_status": "provisioned",
"qos_characteristics": {
  "rtt": 30,
  "delay": 20,
  "jitter": 10,
  "priority": 7,
  "packet_loss": 10,
  "reliability": 100,
  "burst_arrival_time_window": {
    "t1": 0,
    "t2": 5
  },
  "burst_completion_time_window": {
    "t1": 0,
    "t2": 5
  }
},
"service_lifetime": {
  "service_duration": {
    "t1": 0,
    "t2": 2
  },
  "recurrent_service_interval": {
    "t1": 0,
    "t2": 1,
    "recurrence_interval": 1
  }
},
"endpoints": {
  "source": {
    "id": "PC_Talker"
  },
  "destination": {
    "id": "PC_Listener"
  }
}
```

Figure 4-17. E2E Service Exposure response with the E2E service 33382 provisioned over the TSN domain

4.2 E2E Service Decommissioning

Like the provisioning, this section reports the E2E service decommissioning process. Figure 4-18 depicts the implementation version of the decommissioning workflow defined in (PREDICT-6G/D3.2, 2023).

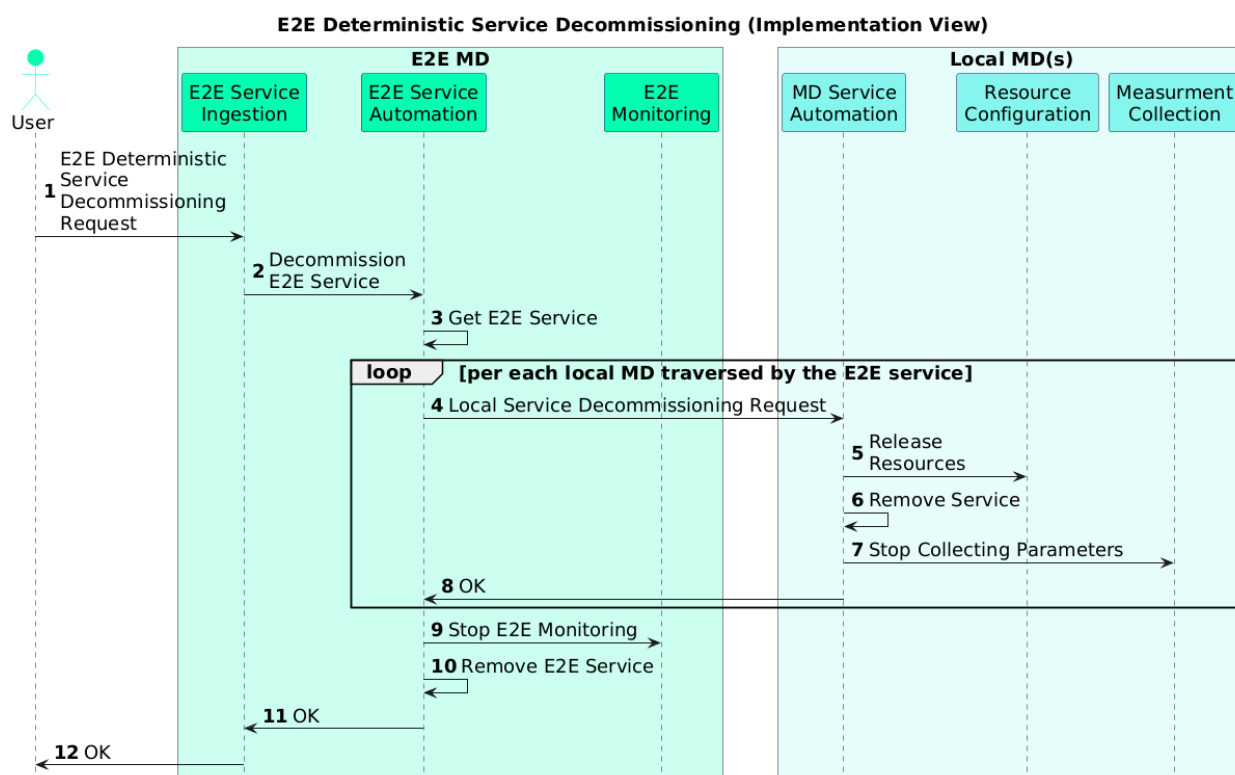


Figure 4-18. Implementation view of the E2E deterministic service decommissioning workflow

The decommissioning request is sent by the client (step 1 in Figure 4-18) by means of the E2E service ingestion and forwarded to the E2E Service Automation component (step 2, showcased in Figure 4-19).

```

DEBUG:connexion.decorators.validation:http://10.5.15.65:8300/service/33382 validating parameters...
DEBUG:connexion.decorators.parameter:Function Arguments: ['id_']
INFO:root:Received request to decommission the E2E service with ID 33382
INFO:root:Retrieve the service requests status with the given parameters from the Request Status Register
INFO:root:Query InfluxDB bucket 'request_status_register' for last '1h' of data
INFO:root:Filtering by tag information '33382'
DEBUG:root:Full query:
from(bucket:"request_status_register") |> range(start: -1h) |> filter(fn:(r) => r.information == "33382")
    
```

Figure 4-19. Forwarding of E2E service decommissioning request from E2E Service Ingestion to E2E Service Automation

Upon the reception of the request, the E2E Service Automation coordinates the decommissioning process by requesting the decommissioning of the local services that compose the E2E one to the corresponding MD Service Automation components in step 4 (Figure 4-20).

```
2025-01-27T16:35:09.238|INFO|rest_app|decommissionService()|rest_app.py:63|Received request to decommission E2E service with ID 33382.
2025-01-27T16:35:09.331|INFO|rest_manager|decommissionService()|rest_manager.py:94|Decommissioning response for local service: True
2025-01-27T16:35:09.331|INFO|rest_app|decommissionService()|rest_app.py:74|Requested decommissioning of E2E service with ID 33382 successfully.
INFO: 10.1.194.4:46970 - "DELETE /decommissione2eservice/33382 HTTP/1.1" 200 OK
```

Figure 4-20. E2E Service decommissioning request received and performed at the E2E Service Automation

Each MD Service Automation de-configures the local service by means of the Resource Configurator (step 5, Figure 4-21 and Figure 4-22).

```
2025-01-27T16:35:09.291|INFO|rest_app|decommissionService()|rest_app.py:64|Received request to decommission MD service with ID 1125.
2025-01-27T16:35:09.324|INFO|rest_app|decommissionService()|rest_app.py:75|MD service with ID 1125 requested decommissioning successfully.
INFO: 10.1.194.4:46970 - "DELETE /Lifecycle/decommissionmdservicerequest/1125 HTTP/1.1" 200 OK
```

Figure 4-21. Request result of the Lifecycle Management in the MD Service Automation for the decommissioning of a MD Service in TSN domain, leading to contact the Resource Configurator

```
2025-01-27T16:35:09.947|INFO|rest_app|releasemgmt()|rest_app.py:123|Service 1125 successfully decommissioned, monitoring triggered and DT notified.
INFO: 10.1.194.4:46974 - "PUT /Lifecycle/decommissionmdserviceresponse/1125 HTTP/1.1" 200 OK
```

Figure 4-22. Result of the Lifecycle Management in the MD Service Automation during the decommissioning phase after receiving the Resource Configurator response in TSN domain

Once the service has been successfully decommissioned, the monitoring is stopped (step 7, showcased in Figure 4-23).

```
DEBUG:connexion.apis.flask_api:Getting data and status code
DEBUG:connexion.decorators.validation:http://10.5.15.64:8888/datasources/MDP?service_id=1125 validating parameters...
DEBUG:connexion.decorators.parameter:Function Arguments: ['type', 'service_id']
DEBUG:connexion.operations.abstract:Query Parameter 'service_id' (sanitized: 'service_id') in function arguments
DEBUG:connexion.operations.abstract:service_id is a {'name': 'service_id', 'in': 'query', 'description': 'Identifier of the service to deal with.', 'required': False, 'style': 'form', 'explode': True, 'schema': {'type': 'string'}}
INFO:root:Received request to retrieve the configuration of all the active datasources of type MDP with service_id 1125
DEBUG:connexion.apis.abstract:Getting data and status code
DEBUG:connexion.apis.abstract:Got framework response
INFO:werkzeug:10.0.0.2 - - [27/Jan/2025 16:35:09] "GET /datasources/MDP?service_id=1125 HTTP/1.1" 200 -
DEBUG:connexion.apis.flask_api:Getting data and status code
DEBUG:connexion.decorators.validation:http://10.5.15.64:8888/datasources/MDP/id/7a5765e8-b25d-485c-9cf2-blad315c238b validating parameters...
DEBUG:connexion.decorators.parameter:Function Arguments: ['type', 'id_']
INFO:root:Received request to delete the active datasource of type MDP with id 7a5765e8-b25d-485c-9cf2-blad315c238b
DEBUG:filelock:Attempting to acquire lock 129968650666912 on telegraf_mdp_conf.lock
DEBUG:filelock:lock 129968650666912 acquired on telegraf_mdp_conf.lock
DEBUG:root:Critical section accessed
INFO:root:Removing Exec inputs for MDP OpenAPI for service 1125
INFO:root:Removing Kafka and InfluxDB inputs for service 1125
INFO:root:Produced new telegraf configuration after deletion, writing it on the file
INFO:root:Configuration wrote
INFO:root:Running update_telegraf_mdp_service()
DEBUG:filelock:Attempting to release lock 129968650666912 on telegraf_mdp_conf.lock
DEBUG:filelock:lock 129968650666912 released on telegraf_mdp_conf.lock
DEBUG:connexion.apis.abstract:Getting data and status code
DEBUG:connexion.apis.abstract:Got framework response
INFO:werkzeug:10.0.0.2 - - [27/Jan/2025 16:35:09] "DELETE /datasources/MDP/id/7a5765e8-b25d-485c-9cf2-blad315c238b HTTP/1.1" 200 -
```

Figure 4-23. Request of stopping the monitoring of a technological domain sent by the Service Automation. A GET request is performed before the DELETE to retrieve the data source ID coupled to the service ID

The Service Automation of each local domain notifies the result of the local service decommissioning to the E2E (step 8, Figure 4-24).

```
2025-01-27T16:35:09.710 [INFO]rest_app[decommissionedService()]rest_app.py:113|Received response for decommissioned MD service with ID 1125.
2025-01-27T16:35:09.834 [INFO]rest_manager[triggerMonitoring()]rest_manager.py:235|E2E Monitoring stopped successfully.
2025-01-27T16:35:09.835 [INFO]rest_manager[notifyStatusE2EDT()]rest_manager.py:193|Sending status of the provision to: http://e2epathcomputation-service:9500
/e2eservice
2025-01-27T16:35:09.851 [INFO]rest_manager[notifyStatusE2EServiceIngestion()]rest_manager.py:170|Sending status of the provision to: http://10.5.15.65:8300/s
ervice-decommissioning-notification
2025-01-27T16:35:09.982 [INFO]rest_app[decommissionedService()]rest_app.py:124|MD service with ID 1125 decommissioned successfully.
INFO: 10.1.118.198:42684 - "POST /e2elifecycle/decommissionedmdservice HTTP/1.1" 200 OK
```

Figure 4-24. E2E Service decommissioning response received at the E2E Service Automation from the MD Service Automation including E2E Monitoring stop and E2E Service Decommissioning response sent to the E2E Service Ingestion

Once all the local services that compose the E2E one have been de-configured, the E2E service monitoring is stopped (step 9, Figure 4-25).

```
DEBUG:connexion.apis.flask_api:Getting data and status code
DEBUG:connexion.decorators.validation:http://10.5.15.65:8888/datasources/E2E?service_id=33382 validating parameters...
DEBUG:connexion.decorators.parameter:Function Arguments: ['type_', 'service_id']
DEBUG:connexion.operations.abstract:Query Parameter 'service_id' (sanitized: 'service_id') in function arguments
DEBUG:connexion.operations.abstract:service_id is a {'name': 'service_id', 'in': 'query', 'description': 'Identifier of the service to deal with.', 'required': False, 'style': 'form', 'explode': True, 'schema': {'type': 'string'}}
INFO:root:Received request to retrieve the configuration of all the active datasources of type E2E with service_id 33382
DEBUG:connexion.apis.abstract:Getting data and status code
DEBUG:connexion.apis.abstract:Got framework response
INFO:werkzeug:10.0.0.2 - - [27/Jan/2025 16:35:09] "GET /datasources/E2E?service_id=33382 HTTP/1.1" 200 -
DEBUG:connexion.apis.flask_api:Getting data and status code
DEBUG:connexion.decorators.validation:http://10.5.15.65:8888/datasources/E2E/id/a30787cf-4bc9-4340-8fd1-37c5cea57097 validating parameters...
DEBUG:connexion.decorators.parameter:Function Arguments: ['type_', 'id_']
INFO:root:Received request to delete the active datasource of type E2E with id a30787cf-4bc9-4340-8fd1-37c5cea57097
DEBUG:filelock:Attempting to acquire lock 124165995829664 on telegraf_e2e_conf.lock
DEBUG:filelock:Lock 124165995829664 acquired on telegraf_e2e_conf.lock
DEBUG:root:Critical section accessed
INFO:root:Removing all inputs and outputs for E2E datasource
INFO:root:Produced new telegraf configuration after deletion, writing it on the file
INFO:root:Configuration wrote
INFO:root:Running update_telegraf_e2e_service()
DEBUG:filelock:Attempting to release lock 124165995829664 on telegraf_e2e_conf.lock
DEBUG:filelock:Lock 124165995829664 released on telegraf_e2e_conf.lock
DEBUG:connexion.apis.abstract:Getting data and status code
DEBUG:connexion.apis.abstract:Got framework response
INFO:werkzeug:10.0.0.2 - - [27/Jan/2025 16:35:09] "DELETE /datasources/E2E/id/a30787cf-4bc9-4340-8fd1-37c5cea57097 HTTP/1.1" 200 -
```

Figure 4-25. Request of stopping the monitoring at the E2E MD level sent by the E2E Service Automation. A GET request is performed before the DELETE in order to retrieve the data source ID coupled to the E2E service ID

Next, the service is tagged as decommissioned in the E2E Service Automation database (step 10, Figure 4-26).

```
{
  "traffic_characteristics": {
    "period": 20,
    "direction": "directional",
    "burst_size": 500,
    "periodicity": true,
    "maximum_flow_bitrate": 5000
  },
  "md_services": {
    "local_service": [
      {
        "index": 0,
        "service": {
          "id": "1125",
          "target_domain": {
            "name": "mdTsn",
            "class": "DetNet"
          },
          "service_status": "decommissioned"
        }
      }
    ]
  },
  "id": "33382",
  "e2e_path_selection": {
    "id": "7",
    "domain_list": [
      {
        "path_domains": {
          "name": "mdTsn",
          "class": "DetNet"
        }
      }
    ],
    "path_status": "computed"
  },
  "service_status": "decommissioned",
  "qos_characteristics": {
    "rtt": 30,
    "delay": 20,
    "jitter": 10,
    "priority": 7,
    "packet_loss": 10,
    "reliability": 100,
    "burst_arrival_time_window": {
      "t1": 0,
      "t2": 5
    },
    "burst_completion_time_window": {
      "t1": 0,
      "t2": 5
    }
  },
  "service_lifetime": {
    "service_duration": {
      "t1": 0,
      "t2": 2
    },
    "recurrent_service_interval": {
      "t1": 0,
      "t2": 1,
      "recurrence_interval": 1
    }
  },
  "endpoints": {
    "source": {
      "id": "PC_Talker"
    },
    "destination": {
      "id": "PC_Listener"
    }
  }
}
```

Figure 4-26. E2E Service Exposure shows E2E service 33382 as decommissioned in TSN domain

Finally, the E2E Service Automation notifies the decommissioning result to the E2E Service Ingestion (Figure 4-27).

```
INFO:root:Sending E2E service decommissioning request to E2E Service Automation for processing
DEBUG:root:E2E SERVICE RESPONSE: '{"message": "Requested decommissioning of E2E service with ID 33382 successfully."}'
INFO:root:Storing the request '65f63a68-382a-48ad-b2de-f86420894597' status in the Request Status Register
INFO:root:<65f63a68-382a-48ad-b2de-f86420894597>, <DECOMMISSION>, <28ec6a83-52ad-4c39-ae64-f9a6dffa228d3>, <In_Progress>, <33382>
DEBUG:connexion.apis.abstract:Getting data and status code
DEBUG:connexion.apis.abstract:Got framework response
INFO:werkzeug:10.0.0.2 - - [27/Jun/2025 16:35:09] "DELETE /service/33382 HTTP/1.1" 200 -
DEBUG:connexion.apis.flask_api:Getting data and status code
DEBUG:connexion.decorators.validation:http://10.5.15.65:8380/service-decommissioning-notification validating schema...
DEBUG:connexion.decorators.parameter:Function Arguments: ['body']
INFO:root:Received request to notify the user about the outcome of the provisioning request for service with ID 33382
INFO:root:Retrieve the service requests status with the given parameters from the Request Status Register
INFO:root:Query InfluxDB bucket 'request_status_register' for last '1h' of data
INFO:root:Filtering by measurement 'SERVICE_DECOMMISSION'
INFO:root:Filtering by tag information '33382'
DEBUG:root:Full query:
from(bucket:"request_status_register") |> range(start: -1h) |> filter(fn: (r) => r._measurement == "SERVICE_DECOMMISSION") |> filter(fn: (r) => r.information == "33382")
DEBUG:root:Table:
FluxTable() columns: 11, records: 1
DEBUG:root:Record:
FluxRecord() table: 0, {'result': '_result', 'table': 0, '_start': datetime.datetime(2025, 1, 27, 15, 35, 9, 869187, tzinfo=tzlocal()), '_stop': datetime.datetime(2025, 1, 27, 16, 35, 9, 869187, tzinfo=tzlocal()), '_time': datetime.datetime(2025, 1, 27, 16, 35, 9, 352312, tzinfo=tzlocal()), '_value': 'In_Progress', '_field': 'status', '_measurement': 'SERVICE_DECOMMISSION', 'information': '33382', 'request_id': '65f63a68-382a-48ad-b2de-f86420894597', 'requestor_id': '28ec6a83-52ad-4c39-ae64-f9a6dffa228d3'}
DEBUG:root:Query result:
[
  {
    "request_id": "65f63a68-382a-48ad-b2de-f86420894597",
    "requestor_id": "28ec6a83-52ad-4c39-ae64-f9a6dffa228d3",
    "request_type": "SERVICE_DECOMMISSION",
    "status": "In_Progress",
    "information": "33382"
  }
]
INFO:root:Storing the request '65f63a68-382a-48ad-b2de-f86420894597' status in the Request Status Register
INFO:root:<65f63a68-382a-48ad-b2de-f86420894597>, <DECOMMISSION>, <28ec6a83-52ad-4c39-ae64-f9a6dffa228d3>, <Accepted>, <33382>
DEBUG:connexion.apis.abstract:Getting data and status code
DEBUG:connexion.apis.abstract:Got framework response
INFO:werkzeug:10.0.0.2 - - [27/Jun/2025 16:35:09] "POST /service-decommissioning-notification HTTP/1.1" 200 -
```

Figure 4-27. Notification of the accomplishment of the decommissioning of the service sent by the E2E Service Automation to the E2E Service Ingestion and forwarded to the final user

4.3 MLOps – WiFi DT workflow for Federated Learning

This section summarizes the validation of the operational workflow designed to describe the interaction between the MLOps framework and the Digital Twin used to model the WiFi domain. In particular, the objective of the workflow is to develop a Federated Learning model able to select the best traffic scheduling policy in different WiFi domains with a variable number of Access Points.

The validation of the workflow has been performed by leveraging the data obtained from the WiFi Digital Twin in four domains to develop a Federated Learning model. The domains have a variable number of WiFi access points (APs): Domain 1 has 8 APs, Domain 2 has 16 APs, Domain 3 has 32 APs and Domain 4 has 64 APs. To leverage the data of each domain, four different datasets have been generated by the Digital Twin: *output_8_STA.csv*, *output_16_STA.csv*, *output_32_STA.csv* and *output_64_STA.csv*.

client1-pipelines <small>Created on: Wed, Jan 15 2025 11:38:41 (GMT+1) Access: PRIVATE 26.9 MiB - 2 Objects</small>			Rewind ↺	Refresh ↻	Upload ↗
client1-pipelines / federated / common / data			Create new path 📁		
<input type="checkbox"/>	▲ Name	Last Modified	Size		
<input type="checkbox"/>	📄 output_8_STA.csv	Wed, Jan 15 2025 11:48 (GMT+1)	26.9 MiB		

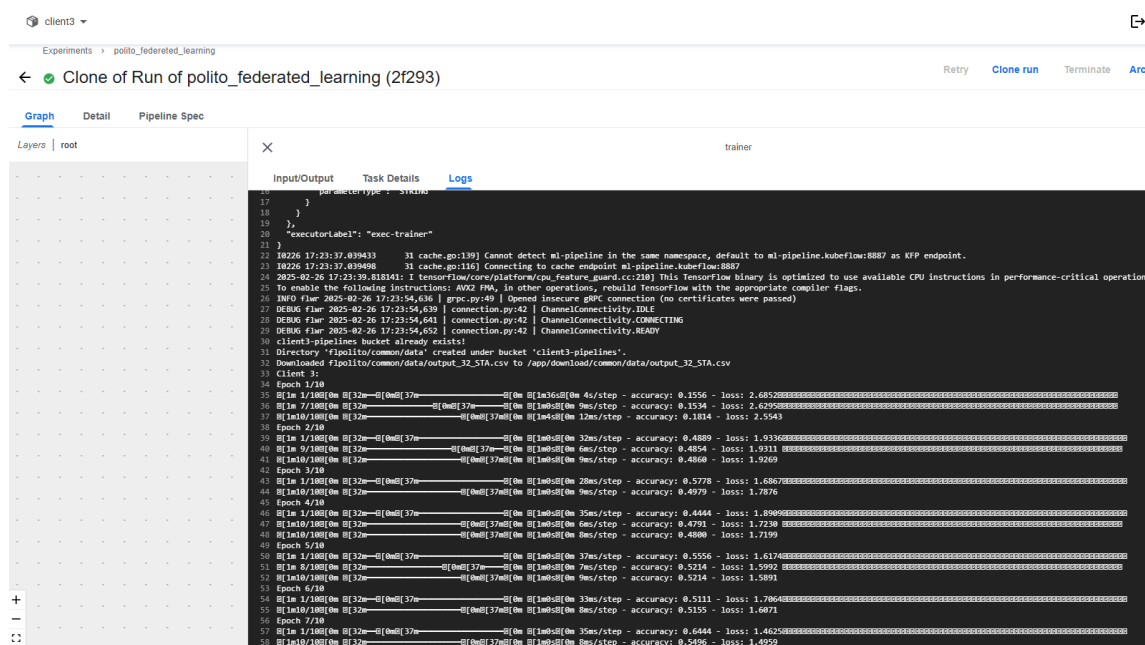
client2-pipelines <small>Created on: Wed, Jan 15 2025 11:59:58 (GMT+1) Access: PRIVATE 43.0 MiB - 2 Objects</small>			Rewind ↺	Refresh ↻	Upload ↗
client2-pipelines / federated / common / data			Create new path 📁		
<input type="checkbox"/>	▲ Name	Last Modified	Size		
<input type="checkbox"/>	📄 output_16_STA.csv	Wed, Jan 15 2025 12:01 (GMT+1)	43.0 MiB		

client3-pipelines <small>Created on: Wed, Jan 15 2025 12:38:01 (GMT+1) Access: PRIVATE 73.9 MiB - 2 Objects</small>			Rewind ↺	Refresh ↻	Upload ↗
client3-pipelines / flpolito / common / data			Create new path 📁		
<input type="checkbox"/>	▲ Name	Last Modified	Size		
<input type="checkbox"/>	📄 output_32_STA.csv	Wed, Feb 26 2025 18:22 (GMT+1)	73.9 MiB		

client4-pipelines <small>Created on: Wed, Jan 15 2025 12:49:37 (GMT+1) Access: PRIVATE 129.7 MiB - 2 Objects</small>			Rewind ↺	Refresh ↻	Upload ↗
client4-pipelines / flpolito / common / data			Create new path 📁		
<input type="checkbox"/>	▲ Name	Last Modified	Size		
<input type="checkbox"/>	📄 output_64_STA.csv	Wed, Feb 26 2025 18:26 (GMT+1)	129.7 MiB		

Figure 4-28. Isolated Dataset Storage in MinIO

Once the four clients have been created, a specific Kubeflow pipeline has been created for each user. In each pipeline, a DNN model has been defined for each client, leveraging the corresponding dataset for triggering a Federated Learning model training based on model aggregation principles. To this end, the Flower Aggregation Server described in section 3.4 (Test 3) has been used to aggregate the weights of each DNN through an iterative process where model weights have been exchanged during 50 rounds.



After obtaining the aggregated Federated Learning model, the Kubeflow pipeline has pushed the model to the Model Storage module, i.e. MinIO repository, so the model serving module is able to access the aggregated model for serving the inference through a Kserve endpoint.

```
16/16 ----- 0% 4ms/step - accuracy: 0.5817 - loss: 2.6290
INFO flwr 2025-02-26 17:28:18,257 | server.py:125 | fit progress: (6, 4.7630899071350698, {'accuracy': 0.527999997138977}, 1011.1389063590015)
DEBUG flwr 2025-02-26 17:28:18,258 | server.py:173 | evaluate_round 6: strategy sampled 4 clients (out of 4)
DEBUG flwr 2025-02-26 17:28:18,441 | server.py:187 | evaluate_round 6 received 4 results and 0 failures
DEBUG flwr 2025-02-26 17:28:18,441 | server.py:222 | fit_round 7: strategy sampled 4 clients (out of 4)
DEBUG flwr 2025-02-26 17:28:19,631 | server.py:236 | fit_round 7 received 4 results and 0 failures
16/16 ----- 0% 4ms/step - accuracy: 0.6338 - loss: 2.8445
INFO flwr 2025-02-26 17:28:19,806 | server.py:125 | fit progress: (7, 5.432831287384033, {'accuracy': 0.5519999861717224}, 1012.679730556003)
DEBUG flwr 2025-02-26 17:28:19,807 | server.py:173 | evaluate_round 7: strategy sampled 4 clients (out of 4)
DEBUG flwr 2025-02-26 17:28:19,994 | server.py:187 | evaluate_round 7 received 4 results and 0 failures
DEBUG flwr 2025-02-26 17:28:19,995 | server.py:222 | fit_round 8: strategy sampled 4 clients (out of 4)
DEBUG flwr 2025-02-26 17:28:21,251 | server.py:236 | fit_round 8 received 4 results and 0 failures
16/16 ----- 0% 4ms/step - accuracy: 0.6102 - loss: 3.1987
INFO flwr 2025-02-26 17:28:21,438 | server.py:125 | fit progress: (8, 5.9978861808776855, {'accuracy': 0.5379999876022339}, 1014.3119881740131)
DEBUG flwr 2025-02-26 17:28:21,439 | server.py:173 | evaluate_round 8: strategy sampled 4 clients (out of 4)
DEBUG flwr 2025-02-26 17:28:21,629 | server.py:187 | evaluate_round 8 received 4 results and 0 failures
DEBUG flwr 2025-02-26 17:28:21,629 | server.py:222 | fit_round 9: strategy sampled 4 clients (out of 4)
DEBUG flwr 2025-02-26 17:28:22,900 | server.py:236 | fit_round 9 received 4 results and 0 failures
16/16 ----- 0% 3ms/step - accuracy: 0.6869 - loss: 3.0701
INFO flwr 2025-02-26 17:28:23,054 | server.py:125 | fit progress: (9, 5.9883290367126465, {'accuracy': 0.5759999752044678}, 1015.9271426630148)
DEBUG flwr 2025-02-26 17:28:23,054 | server.py:173 | evaluate_round 9: strategy sampled 4 clients (out of 4)
DEBUG flwr 2025-02-26 17:28:23,243 | server.py:187 | evaluate_round 9 received 4 results and 0 failures
DEBUG flwr 2025-02-26 17:28:23,244 | server.py:222 | fit_round 10: strategy sampled 4 clients (out of 4)
DEBUG flwr 2025-02-26 17:28:24,472 | server.py:236 | fit_round 10 received 4 results and 0 failures
Saving global model after round 10
model bucket already exists!
Directory 'trained_model' created under bucket 'model'.
Uploaded kserve/model_output/global_model.keras to trained_model/global_model.keras in MinIO
16/16 ----- 0% 4ms/step - accuracy: 0.7413 - loss: 3.3985
INFO flwr 2025-02-26 17:28:24,751 | server.py:125 | fit progress: (10, 6.075433921813965, {'accuracy': 0.6019999880926514}, 1017.6249193749973)
DEBUG flwr 2025-02-26 17:28:24,752 | server.py:173 | evaluate_round 10: strategy sampled 4 clients (out of 4)
DEBUG flwr 2025-02-26 17:28:24,933 | server.py:187 | evaluate_round 10 received 4 results and 0 failures
INFO flwr 2025-02-26 17:28:24,934 | server.py:153 | FL finished in 1017.807390210015
INFO flwr 2025-02-26 17:28:24,935 | app.py:225 | app_fit: losses distributed [(1, 2.4415033901284033), (2, 1.8665865063667297), (3, 2.0459578186273575), (4, 2.419506706677185), (5, 2.6213411390781403), (6, 6.5370594551697), (7, 3.31025112055023), (8, 3.66285938076973), (9, 3.7281513661146164), (10, 3.816574126482011)]
INFO flwr 2025-02-26 17:28:24,935 | app.py:226 | app_fit: metrics distributed fit {}
INFO flwr 2025-02-26 17:28:24,935 | app.py:227 | app_fit: metrics distributed {}
INFO flwr 2025-02-26 17:28:24,935 | app.py:228 | app_fit: losses centralized [(0, 33.05892562866211), (1, 3.0695197582244873), (2, 2.9462854862213135), (3, 3.3226869106292725), (4, 4.160017490386963), (5, 4.204315185547), (6, 4.7630899071350088), (7, 5.432831287384033), (8, 5.9978861808776855), (9, 5.9883290367126465), (10, 6.075433921813965)]
INFO flwr 2025-02-26 17:28:24,936 | app.py:229 | app_fit: metrics centralized {'accuracy': [(0, 0.026000000000364418093), (1, 0.01789999709129333), (2, 0.550000011920929), (3, 0.492000013589859), (4, 0.4839999919563), (5, 0.4779999852180481), (6, 0.527999997138977), (7, 0.5519999861717224), (8, 0.5379999876022339), (9, 0.5759999752044678), (10, 0.6019999880926514)]}
```

Figure 4-30. Federated Learning model aggregation result

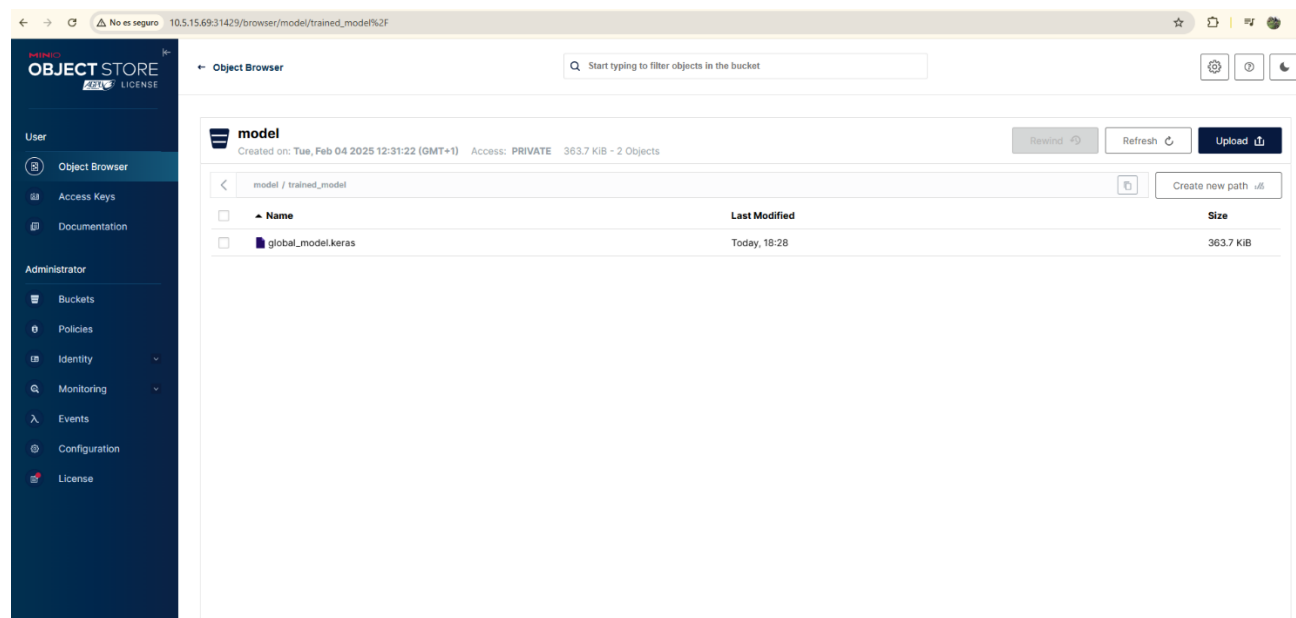


Figure 4-31. Federated Learning aggregated model in Model Storage (MinIO)

The serving of the aggregated Federated Learning model is made available through the endpoint: <http://10.5.15.69:31234/v1/models/global-model:predict>

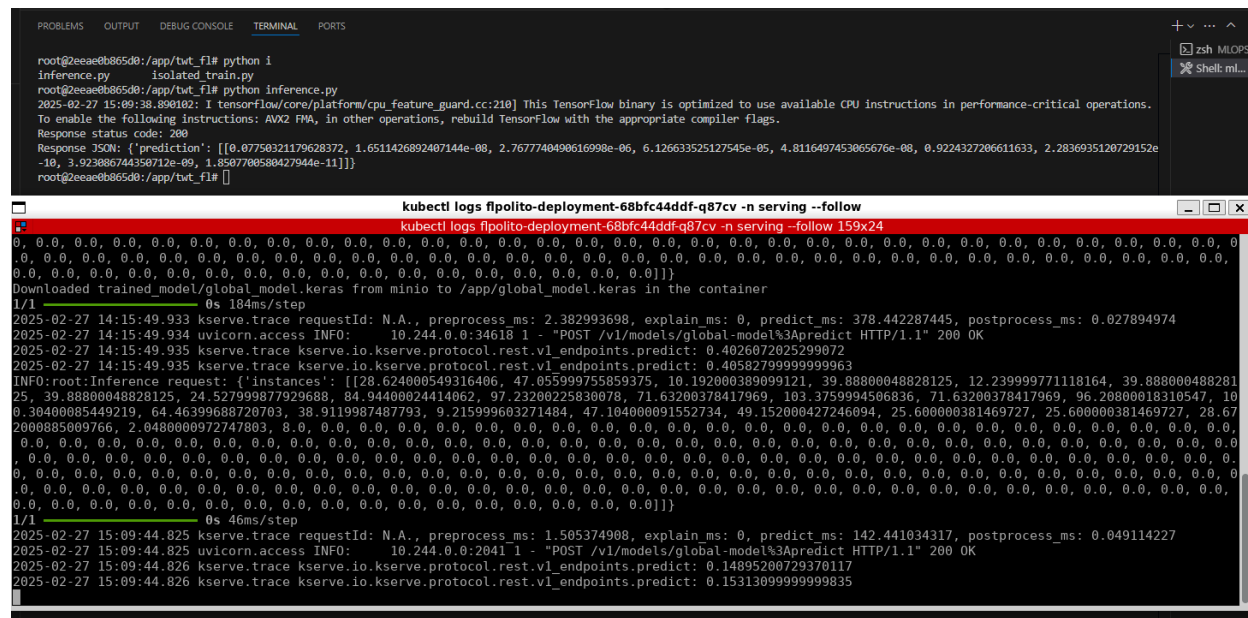


Figure 4-32. Federated Learning model inference performed

4.4 E2E Service Maintenance

This section provides some preliminary work on the service maintenance and quality assurance that has been done throughout the project and that will be finalized in the scope of the framework integration and validation work package (WP4).

Closed-loop automation for the Localisation and Sensing Use Case

In the localisation and sensing use case Closed-loop Automation (CLA) procedures have been designed, implemented and validated. The CLA procedures surround the proposition to utilise topology and link monitoring information from two domains to adjust the PREOF settings of all DetNet routers for specific service flows. Furthermore, as the DetNet router of this demo incorporates the concept of Data Unit Groups (DUGs) (Robitzsch, 2024) which allows to apply PREOF configurations to a set of packets that carry a single application data unit. The PREOF capabilities of the DUG-enabled DetNet router is packet replication and elimination which can be configured at run-time via Create Read Update Delete (CRUD) API calls.

The system architecture for the CLA methodology is illustrated in Figure 4-33 and illustrates the AICP as a set of the three components Data Repository, Model Management and Service Flow Management and the domain-specific components DetNet Controller and DetNet Router. The interfaces denoted as E are the end-to-end interfaces for the domain to interact with the E2E AICP and the interfaces denoted as N as domain-specific.

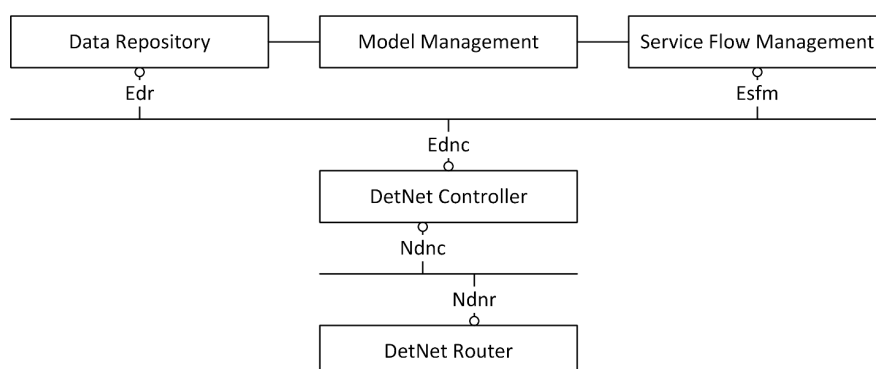


Figure 4-33: Holistic system architecture for Closed-Loop Automation of localisation and sensing use case

All static (topology and capabilities) and dynamic (port and link monitoring) information from the domain is sent via the Edr interface into the Data Repository. Requesting a new, update or delete an existing service flow is conducted via the Esfm interface. The Ednc interface allows the Service Flow Management to create, update or delete a service flow inside the domain.

The CLA continuously uses the live data monitoring information about a domain to assess whether the configured PREOF conditions are sufficient to meet the KPIs such as acceptable packet loss. If the packet loss exceeds the configured KPI threshold (which the AICP learns from the Data Repository), the Service Flow Management component models which PREOF conditions can overcome the packet loss and updates the confirmation of any existing service flow for all domains the service flow traverses.

Study on SDN-based closed loop performance for the wired TSN domain

Aiming to collect some numbers about the performance that could be expected in the implementation of the closed loop, a testing environment has been developed in the framework of the project. This environment implements and validates the performance of a technique for dynamically reconfiguring network paths to mitigate traffic delays. The mechanism uses a latency measurement tool based on TSN time synchronization between sender and receiver nodes. The method leverages the ETF mechanism for precise frame transmission and the Linux kernel's *SO_TIMESTAMPING* API for accurate latency assessment, isolating network communication delays.

The system continuously monitors latency. If a predefined threshold is exceeded, it automatically adjusts the network path via an SDN controller. The receiver evaluates the latency and, in the case the threshold is exceeded, sends a message to the controller that

Four experiments have been conducted to evaluate the system performance under stress, injecting interfering traffic:

1. no reconfiguration.
2. reconfiguration using and Open vSwitch.
3. reconfiguration within a TSN switch.
4. reconfiguration with interfering traffic with high priority.

The diagram illustrates a PTP network topology for a 5G NR system. It shows a central OVS (Open vSwitch) connected to a CONTROLLER and an OSW (Open Switch). The OSW is connected to a sender (SLAVE) and two TSWs (TSW 1 and TSW 2). TSW 1 is connected to two interfaces (int 1 and int 2) and a receiver (SLAVE). TSW 2 is connected to a sink. The network is configured for PTP (Precision Time Protocol) synchronization, with the sender and receiver acting as slaves and the TSWs acting as masters.

Figure 4-34. Experimental setup

TSW 1 and TSW 2 are NXP LS1028ARDB switches (dual-core ARM Cortex-A72 1.3 GHz, 4GB RAM) with 1Gbps Ethernet and four 1Gbps QSGMII ports, supporting TSN and IEEE 1588. An additional NUC (sink) draws interfering traffic, connected via a Netgear GS105 switch (SW) to converge paths to the receiver. The controller on the OSW node awaits REST commands from the receiver to trigger path changes when a latency threshold (calculated from a 1000-sample moving average of 1ms periodic traffic, then assessed every second) is exceeded. Sender and receiver are PTP-synchronized with TSW 1 as master; interfering nodes are not synchronized.

Interfering nodes are controlled via network commands at the start of each experiment, scheduling a wait time before launching *iperf3* with specific traffic characteristics. Initially, the traffic under analysis flows from sender to receiver via TSW 1, configured in the Open vSwitch (OVS) with the following static rules:

1. Sender to receiver traffic is directed to TSW 1;
2. Receiver to sender traffic is directed to TSW 1;
3. REST commands from the receiver are directed to the controller.

Preliminary tests characterized network component latency (TSW 1, TSW 2, SW) using 1,000,000 samples at 1ms intervals (Table 4-1). As expected, hardware switches showed consistent latency. Software switch (OSW) latency was evaluated by measuring end-to-end latency, attributing the variability to the OSW. DPDK-optimized OVS significantly outperformed a standard OVS installation (which averaged ~80μs latency, instead of ~25μs).

Device	Latency (ns)				
	Avg	Min	P99	Max	Std
TSW 1 & TSW 2	2711.2	2600	2788	3278	38.6
SW	4951.3	4827	5052	5086	56.29
E2E trough OSW	25530.3	14327	26358	33646	679.4

Table 4-1. Setup latencies

Experiment 1 tested the impact of interfering traffic on TSW 1. Two nodes (int 1 and int 2) used *iperf3* to generate ~800 Mbps of combined UDP traffic. After a 15-second delay following a start command, each *iperf3* instance transmitted 400 Mbps for 30 seconds. Figure 4-35 shows a latency spike >75μs around sample 16000 (interference start), followed by fluctuations between 50-75μs. Latency returned to normal after the interference stopped.

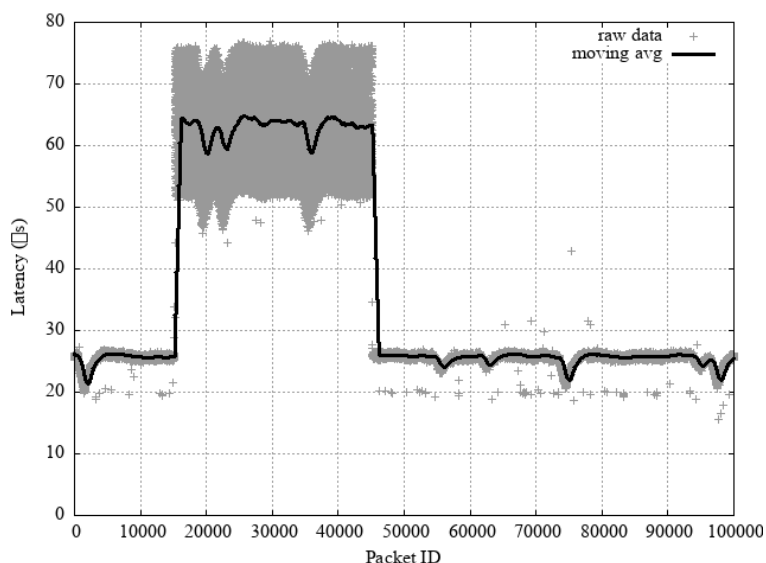


Figure 4-35. Experiment 1, interfering traffic

Experiment 2 implemented dynamic network reconfiguration. The receiver calculated latency using a 1000-sample simple moving average (SMA) (equivalent to 1 second of traffic). A 40 μ s threshold triggered a REST command to the controller on OSW. The controller, initially configured to route traffic via TSW 1, switched to TSW 2 upon receiving the command. New IP traffic rules with higher priority were added while preserving existing rules (including PTP traffic) to prevent packet loss during the switch. Figure 4-36 shows normal operation until interference starts (~sample 16000). When the SMA exceeded 40 μ s, the receiver triggered the path change, restoring lower latency, demonstrating the effectiveness of the dynamic reconfiguration.

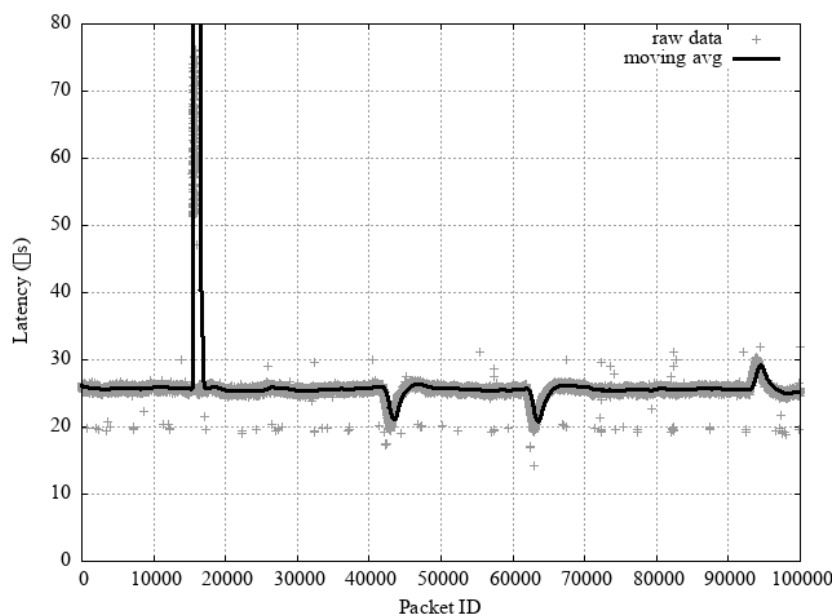


Figure 4-36. Experiment 2, dynamic re-configuration

Experiment 3 explored TSN features. A Time Aware Shaper (IEEE 802.1Qbv) was implemented on TSW 1 to isolate traffic. Sender traffic was assigned priority 3 (critical), interfering traffic priority 0 (Best Effort). The sender used *setsockopt/SO_PRIORITY* and VLAN sub-interfaces to set the Ethernet frame PCP field. TSW 1 was configured with three traffic windows: high priority (300µs), intermediate (300µs), and best effort (400µs) using Linux TAPRIO and hardware offloading. In this experiment, instead of changing paths upon exceeding the latency threshold, the Time Aware Shaper was applied to TSW 1. Figure 4-37 shows a latency increase around sample 16000, followed by a cluster of high-latency packets (up to 30ms) before returning to normal. While initially behaving like Experiment 2, activating Qbv effectively isolated high-priority traffic. 28 packets experienced delays of tens of milliseconds (demarcated by green lines), showing a regular latency decrease as queued packets were dequeued after the configuration change. This experiment demonstrates dynamically adjustable transmission windows based on priority but also highlights challenges: increased delay from reconfiguration and the need for a standardized switch reconfiguration protocol.

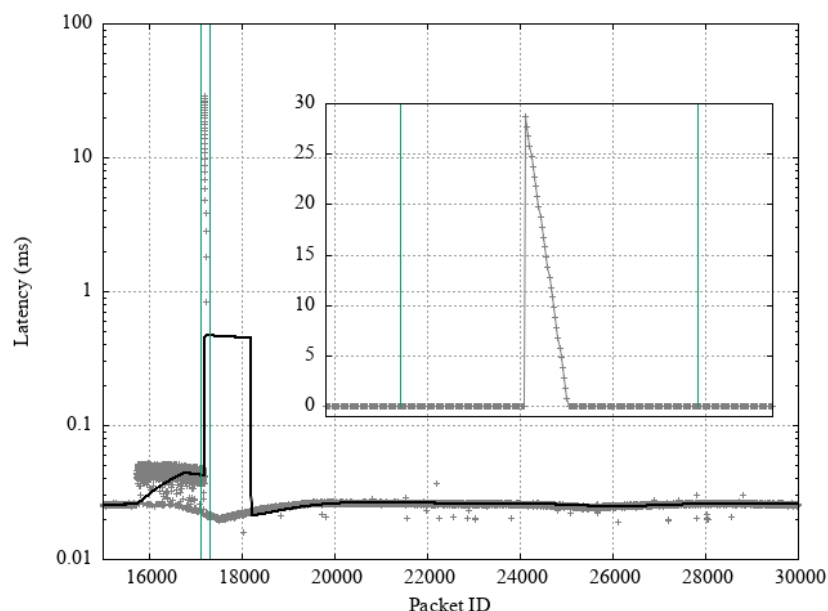


Figure 4-37. Experiment 3 Time Aware Shaper on the TSW 1 and zoom for a comparison with other experiments

Experiment 4 began with TSW 1's Time Aware Shaper active. Interfering traffic from int 1 via *iperf3* was remapped from Best Effort (priority 0) to the same priority as the analyzed traffic (priority 3) using VLAN settings. This again caused interference and delays (Figure 4-38). With a smaller high-priority window (300 μ s instead of 1ms), a single interfering node generating 260 Mbps was sufficient to exceed the latency threshold and trigger a path change.

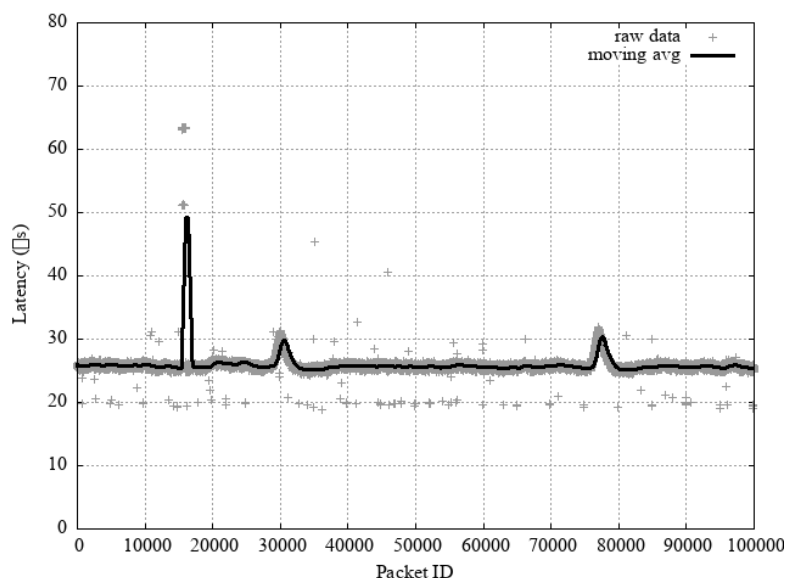


Figure 4-38. Experiment 4, interfering traffic mapped to high priority and TAPRIO

To conclude, this activity evaluated integrating a TSN network with an SDN controller for dynamic reconfiguration to maintain QoS for time-critical data. Experiments demonstrated the effectiveness of the approach effectiveness, highlighting the pros and cons of different functionalities. Open vSwitch introduced some non-determinism but enabled reliable, fast, lossless path switching. Another experiment explored dynamically configuring Time-Aware Shapers on TSN switches, showing effectiveness but also costs. Overall, the experiments demonstrated the need for a flexible topology leveraging both SDN and TSN to dynamically adapt to network load changes. More details can be found in (Zunino, 2024).

5 Conclusions

This deliverable provided the final updates, implementation notes and integration of the AI-driven multi-stakeholder Inter-domain Control Plane (AICP) that has been designed and developed in the framework of WP3. The complete list of the implemented AICP software components is also reported, including the link to the project repository, where most of the components are available.

The AICP design followed a service-based approach, where a set of Management Services (MSs) were defined and designed to implement the control and management functionalities necessary to fulfil the requirements for deterministic service provisioning, maintenance and decommissioning. In addition, two levels of management, namely local (MD) and E2E, were defined in the architecture to provide cross-domain and multi-technology operation. In this regard, the deliverable reported the final software release of the mentioned MSs. Additionally, several experimental tests to showcase the integration among the different components are also reported.

The implementation of the AI-based capabilities that have been added to the AICP by means of its MLOps algorithmic framework are also reported, and their performance validated.

Finally, the implementation of the operational workflows that were defined in the previous deliverables of WP3 was exercised and reported here.

All in all, the different software components of the PREDICT-6G have been implemented and validated. It can be concluded that the results of the overall integrations have been successful and very promising in terms of performance, paving thus the way to the integration with the PREDICT-6G data plane, to finally validate the use cases.

6 References

(PREDICT-6G/D1.2, 2023) PREDICT-6G D1.2 “PREDICT-6G framework architecture and initial specification”, Dec. 2023

(PREDICT-6G/D2.1, 2023) PREDICT-6G D2.1 “Release 1 of PREDICT-6G MDP innovations”, Sep. 2023

(PREDICT-6G/D2.3, 2025) PREDICT-6G D2.3 “Release 2 of PREDICT-6G MDP innovation”, Feb. 2025

(PREDICT-6G/D2.4, 2025) PREDICT-6G D2.4 “Implementation of selected release 2 PREDICT-6G MDP innovations”, due in March 2025

(PREDICT-6G/D3.1, 2023) PREDICT-6G D3.1 “Release 1 of AI-driven inter-domain network control, management, and orchestration innovations”, Oct. 2023

(PREDICT-6G/D3.2, 2023) PREDICT-6G D3.2 “Implementation of selected release 1 AI-driven inter-domain network control, management and orchestration innovations”, Dec. 2023, reviewed version released in Jun. 2024

(PREDICT-6G/D3.3, 2025) PREDICT-6G D3.3 “Release 2 of AI-driven inter-domain network control, management, and orchestration innovations”, Feb. 2025

(PREDICT-6G/D4.2, 2024) PREDICT-6G D4.2 “First report on PREDICT-6G system validation”, Oct. 2024

(RFC9543, 2024) Farrel et al., “A Framework for Network Slices in Networks Built from IETF Technologies”, RFC 9543, March 2024.

(Boucadair, 2025) M. Boucadair, et al., “YANG Data Models for Bearers and 'Attachment Circuits'-as-a-Service (ACaaS)”, draft-ietf-opsawg-teas-attachment-circuit-20 (work in progress), January 2025.

(Serna, 2018) R. Serna Oliver, et al., “IEEE 802.1Qbv Gate Control List Synthesis Using Array Theory Encoding,” in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, Porto, Portugal,, 2018

(802.1AS, 2020) IEEE Std 802.1AS-2020, “IEEE Standard for Local and Metropolitan Area Networks--Timing and Synchronization for Time-Sensitive Applications,” 2020.

(Pegase, 2025) RealTime at Work. "RTaW-Pegase: Designing Safe and Optimized Critical Embedded Networks." [Online]. Available: <https://www.realtimeatwork.com/rtaw-pegase/>, accessed 2025

(Tan et al, 2022) Tan, Wei-Peng et al. "Large-scale Deterministic Transmission among IEEE 802.1Qbv Time-Sensitive Networks." ICC 2022 - IEEE International Conference on Communications (2022): 2315-2320.

(Relyum, 2025) Relyum. "RELY-TSN-BRIDGE: Time-Sensitive Networking Switch 10G." [Online]. Available: <https://www.relyum.com/web/rely-tsn-bridge/>, accessed 2025

(TR21.918, 2024) 3GPP Release 18, 2024

(Venkateswaran, 2024) Venkateswaran, S. et al. (2024) 'IEEE 802.11ax Target Wake Time: Design and Performance Analysis in ns-3', In 2024 Workshop on ns-3 (WNS3 2024).

(802.1Qbv, 2015) 802.1Qbv - Enhancements for Scheduled Traffic, IEEE, <https://www.ieee802.org/1/pages/802.1bv.html>

(Krummacker, 2020) Krummacker, D. et al. (2020) 'TSN Simulation: Time-Aware Shaper implemented in ns-3', In 2020 Workshop on Next Generation Networks and Applications.

(Robitzsch, 2024) Robitzsch, S. et al. (2024) "Formation and Assertion of Data Unit Groups in 3GPP Networks with TSN and PDU Set Support," In *2024 IEEE Wireless Communications and Networking Conference (WCNC)*, doi: 10.1109/WCNC57260.2024.10570842.

(Zunino, 2024) C. Zunino et al., "Time-Sensitive Networking and Software-Defined Networking: An Experimental Setup for Realistic Performances," 2024 IEEE 29th International Conference on Emerging Technologies and Factory Automation (ETFA), Padova, Italy, 2024, pp. 1-7, doi: 10.1109/ETFA61755.2024.10710787.

7 Appendixes

7.1 Appendix A – AICP MSs integration tests

Time Synchronization

Test: Collection and evaluation of E2E Time Synch configuration
Description
<p>The E2E Time Synch Management MS module collects the time sync configuration in the PREDICT-6G system and evaluates the consistency of the end-to-end time sync configuration.</p>
Execution steps
<ul style="list-style-type: none"> • Step 1: E2E Time Synch Management MS sends query to each Time Sync MS present in the PREDICT-6G system using the PREDICT-6G Time Sync OpenAPI. • Step 2: Each Time Sync MS in the PREDICT-6G system replies with the actual configuration of the corresponding technology domain using the PREDICT-6G Time Sync OpenAPI. <pre data-bbox="646 1037 976 1247"> { "PTPVersion": "gPTP", "ProtocolType": "ETH", "DomainNr": 0, "GMCapable": true, "IsGM": true } </pre> <ul style="list-style-type: none"> • Step 3: E2E Time Synch Management MS collects the time sync configuration of the domains in the PREDICT-6g system and evaluates the consistency of the time sync configuration for the end-to-end domain. <pre data-bbox="302 1367 1281 1486"> INFO: PREDICT-6G E2E Time Sync Management MS: sending TS config query to Time Sync MS(s): OK INFO: PREDICT-6G E2E Time Sync Management MS: receiving TS config from Time Sync MS(s): OK INFO: PREDICT-6G E2E Time Sync Management MS: evaluation of time sync status in system: status is OK </pre>
Results
<p>The E2E Time Synch Management MS was able to collect the time sync configuration from the Time Sync MSs present in the PREDICT-6G system. Based on the collected information,</p>

it was able to evaluate the end-to-end time sync configuration consistency of the PREDICT-6G system.

Service Ingestion

Test 1: E2E Service Provisioning request

Description

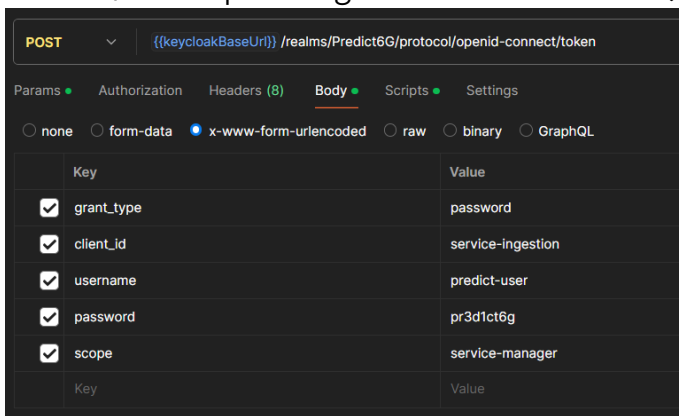
This test aims to validate the E2E SI REST NBI, the authentication logic supported by the IAM Platform, the parsing logic and the use of the E2E Service Automation REST interface.

The expected results of the test are:

1. To correctly authenticate and authorize the requestor.
2. To have a 200 OK response from the E2E Service Automation confirming that the request has been correctly validated and forwarded for processing.
3. To have a new item in the Request Status Register of type *PROVISION* and status *In_Progress*.
4. To receive a notification from the E2E Service Automation confirming the successful provisioning of the service.
5. To have a new item in the Request Status Register of type *PROVISION* and status *Accepted*.

Execution steps

- Send a POST request to *Keycloak_baseURL/realms/Predict6G/protocol/openid-connect/token* providing a set of credentials, the client ID and the scope.



The screenshot shows a REST client interface with the following configuration:

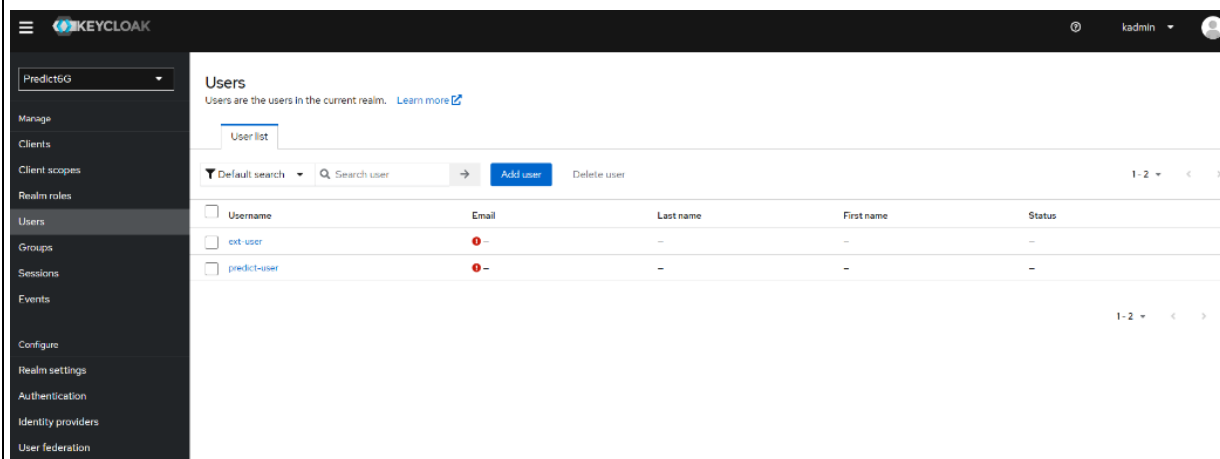
- Method:** POST
- URL:** `{{keycloakBaseUrl}}/realms/Predict6G/protocol/openid-connect/token`
- Params:** (empty)
- Authorization:** (empty)
- Headers (8):** (empty)
- Body:**
 - Content Type:** ☒ x-www-form-urlencoded
 - Form Data:**

Key	Value
<input checked="" type="checkbox"/> grant_type	password
<input checked="" type="checkbox"/> client_id	service-ingestion
<input checked="" type="checkbox"/> username	predict-user
<input checked="" type="checkbox"/> password	pr3d1ct6g
<input checked="" type="checkbox"/> scope	service-manager
- Scripts:** (empty)
- Settings:** (empty)

- Send a POST request to *E2E_Service_Ingestion_baseURL/service* providing a token in the authorization header and a data structure for the service compliant to the AICP information model in the request body.

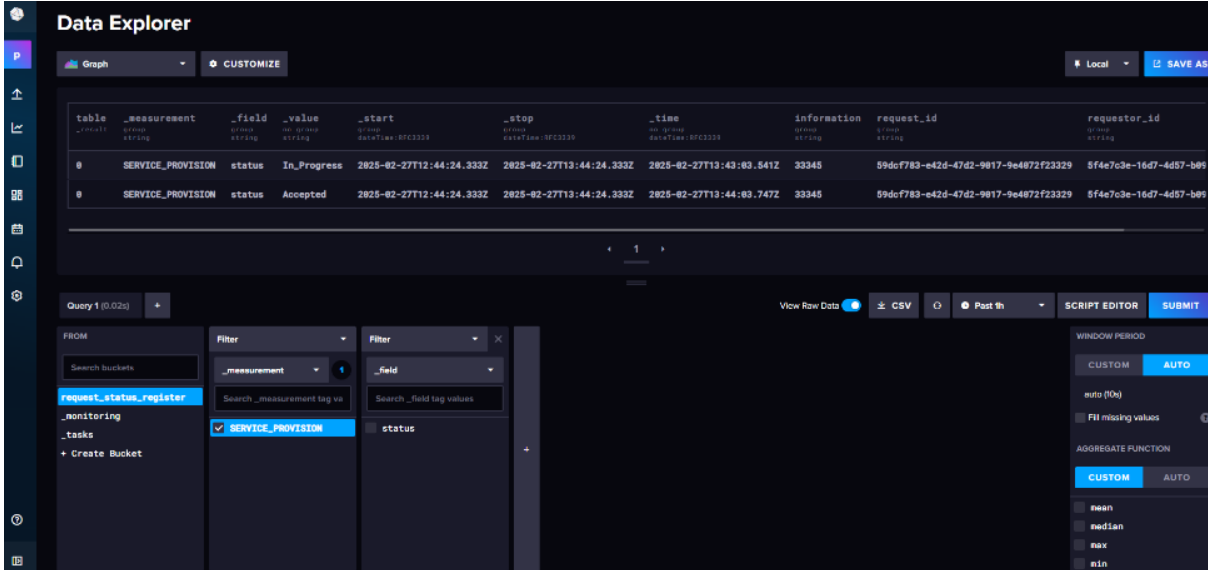
```
{
  "endpoints": {
    "source": {
      "id": "PC Talker"
    },
    "destination": {
      "id": "PC Listener"
    }
  },
  "qos-characteristics": {
    "priority": 7,
    "reliability": 100,
    "packet-loss": 10,
    "e2e-delay": 20,
    "e2e-rtt": 30,
    "jitter": 10,
    "burst-arrival-time-window": {
      "t1": 0,
      "t2": 5
    },
    "burst-completion-time-window": {
      "t1": 0,
      "t2": 5
    }
  },
  "traffic-characteristics": {
    "direction": "directional",
    "periodicity": true,
    "period": 20,
    "burst-size": 500,
    "maximum-flow-bitrate": 5000
  },
  "service-lifetime": {
    "service-duration": {
      "t1": 0,
      "t2": 2
    },
    "recurrent-service-interval": {
      "t1": 0,
      "t2": 1,
      "recurrence-interval": 1
    }
  }
}
```

- The authentication logic checks if the requestor is allowed to send the request by assessing the role through *Keycloak*.



- If the requestor is authorized, the request is parsed and forwarded to the mock-up E2E Service Automation. If the response is 200 OK, an item with the E2E service ID in the *information* field and with status *In_Progress* is stored in the Request Status Register.
- If the service is successfully provisioned, a notification is received from the mock-up E2E Service Automation and an item with status *Accepted* is stored in the Request Status Register.

Results



The screenshot shows the Data Explorer interface with a table of request status register data. The table has columns: table, _measurement, _field, _value, _start, _stop, _time, information, request_id, and requestor_id. The data is filtered to show only 'SERVICE_PROVISION' measurements with 'status' as the field. The status values are 'In_Progress' and 'Accepted'.

table	_measurement	_field	_value	_start	_stop	_time	information	request_id	requestor_id
request_status_register	SERVICE_PROVISION	status	In_Progress	2025-02-27T12:44:24.333Z	2025-02-27T13:44:24.333Z	2025-02-27T13:43:03.541Z	33345	59dcf783-e42d-47d2-9017-9e4072f23329	5f4e7c3e-16d7-4d57-b09
request_status_register	SERVICE_PROVISION	status	Accepted	2025-02-27T12:44:24.333Z	2025-02-27T13:44:24.333Z	2025-02-27T13:44:03.747Z	33345	59dcf783-e42d-47d2-9017-9e4072f23329	5f4e7c3e-16d7-4d57-b09

Test 2: E2E Service Decommissioning request

Description

This test aims to validate the E2E Service Ingestion REST NBI and the use of the E2E Service Automation REST interface.

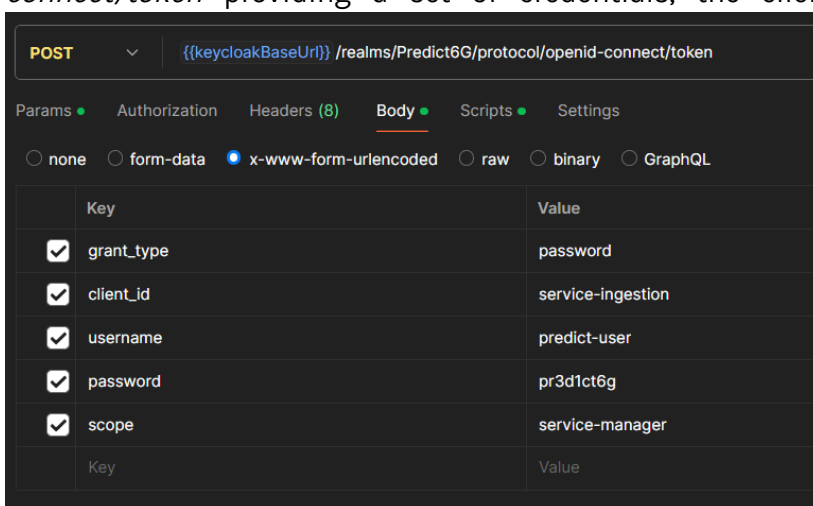
The expected results of the test are:

1. To correctly authenticate and authorize the requestor.
2. To have a 200 OK response from the E2E Service Automation confirming that the request has been correctly validated and forwarded for processing.
3. To have a new item in the Request Status Register of type *DECOMMISSION* and status *In_Progress*.

4. To receive a notification from the E2E Service Automation confirming the successful decommissioning of the service.
5. To have a new item in the Request Status Register of type *DECOMMISSION* and status *Accepted*.

Execution steps

- Preliminary step: to have a provisioned service request with status *Accepted* in the Request Status Register.
- Send a POST request to *Keycloak_baseURL/realms/Predict6G/protocol/openid-connect/token* providing a set of credentials, the client ID and the scope.



Key	Value
grant_type	password
client_id	service-ingestion
username	predict-user
password	pr3d1ct6g
scope	service-manager

- Send a DELETE request to *E2E_Service_Ingestion_baseURL/service/{id}* where *id* is the E2E service ID.
- If the requestor is authorized, the request is forwarded to the mock-up E2E Service Automation. If the response is 200 OK, an item with the E2E service ID in the *information* field and with status *In_Progress* is stored in the Request Status Register.
- If the service is successfully decommissioned, a notification is received from the mock-up E2E Service Automation and an item with status *Accepted* is stored in the Request Status Register.

Results

-

Test 3: E2E Service Update request

Description

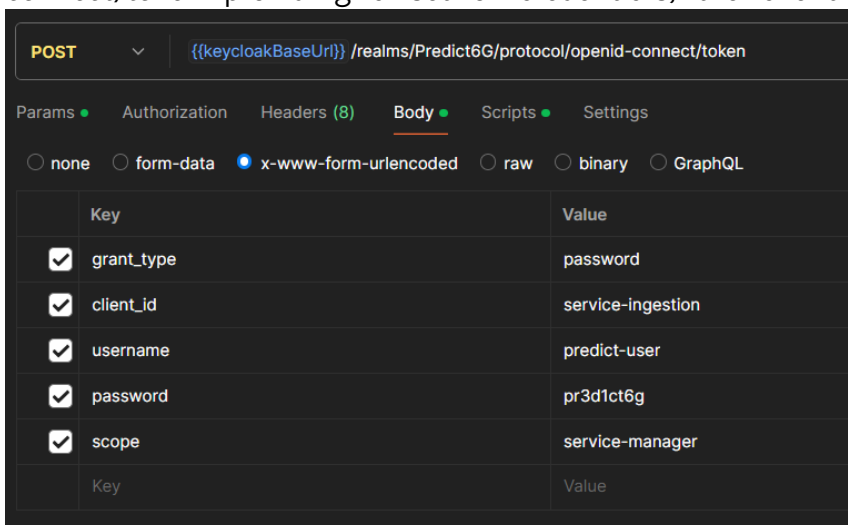
This test aims to validate the E2E SI REST NBI and the use of the E2E Service Automation REST interface.

The expected results of the test are:

1. To correctly authenticate and authorize the requestor.
2. To check if the E2E service with the given ID for which the update is requested is already provisioned.
3. To correctly forward the request to the E2E Service Automation and to receive a 200 OK response.
4. To have a new item in the Request Status Register of type *UPDATE* and status *Accepted*.

Execution steps

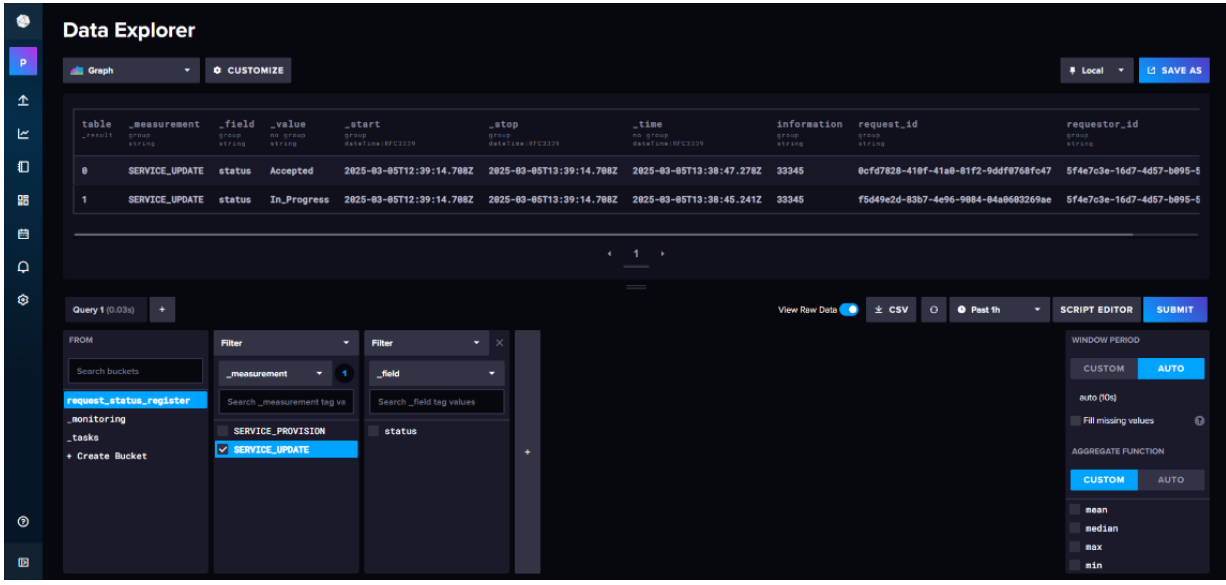
- Preliminary step: to have a provisioned service request with status *Accepted* in the Request Status Register.
- Send a POST request to *Keycloak_baseURL/realms/Predict6G/protocol/openid-connect/token* providing a set of credentials, the client ID and the scope.



Key	Value
<input checked="" type="checkbox"/> grant_type	password
<input checked="" type="checkbox"/> client_id	service-ingestion
<input checked="" type="checkbox"/> username	predict-user
<input checked="" type="checkbox"/> password	pr3d1ct6g
<input checked="" type="checkbox"/> scope	service-manager

- Send a PUT request to *E2E_Service_Ingestion_baseURL/service/{id}* where 'id' is the E2E service ID.
- If the requestor is authorized, the request is forwarded to the mock-up E2E Service Automation. If the response is 200 OK, an item with the E2E service ID in the *information* field and with status *Accepted* is stored in the Request Status Register.

Results



Data Explorer

Graph | CUSTOMIZE | Local | SAVE AS

table	_measurement	_field	_value	_start	_stop	_time	information	request_id	requestor_id
group	group	group	group	group	group	group	group	group	group
id	id	id	id	id	id	id	id	id	id
id	id	id	id	id	id	id	id	id	id
0	SERVICE_UPDATE	status	Accepted	2025-03-05T12:39:14.708Z	2025-03-05T13:39:14.708Z	2025-03-05T13:38:47.276Z	33345	0cfd7828-410f-41a0-81f2-9dd9f9768fc47	5f4e7c3e-16d7-4d57-b095-5
1	SERVICE_UPDATE	status	In_Progress	2025-03-05T12:39:14.708Z	2025-03-05T13:39:14.708Z	2025-03-05T13:38:45.241Z	33345	f5d49e2d-83b7-4e96-9984-04a0603269ae	5f4e7c3e-16d7-4d57-b095-5

Query 1 (0.03s) | View Raw Data | CSV | Post th | SCRIPT EDITOR | SUBMIT

FROM: Search buckets | Filter: _measurement | Filter: _field | WINDOW PERIOD: CUSTOM | AGGREGATE FUNCTION: CUSTOM

Exposure Services

Test 1: TSN MD Topology/Resource/Capabilities Exposure

Description

This test reports the operation of the Exposure MS for the TSN domain when requested by the other modules of the same MD such as the MD PCE.

Execution steps

The Exposure MS collects the topology, resources and capabilities of the technological domain following the information model defined in (PREDICT-6G/D2.3, 2025) and exposes it to the requester.

Results

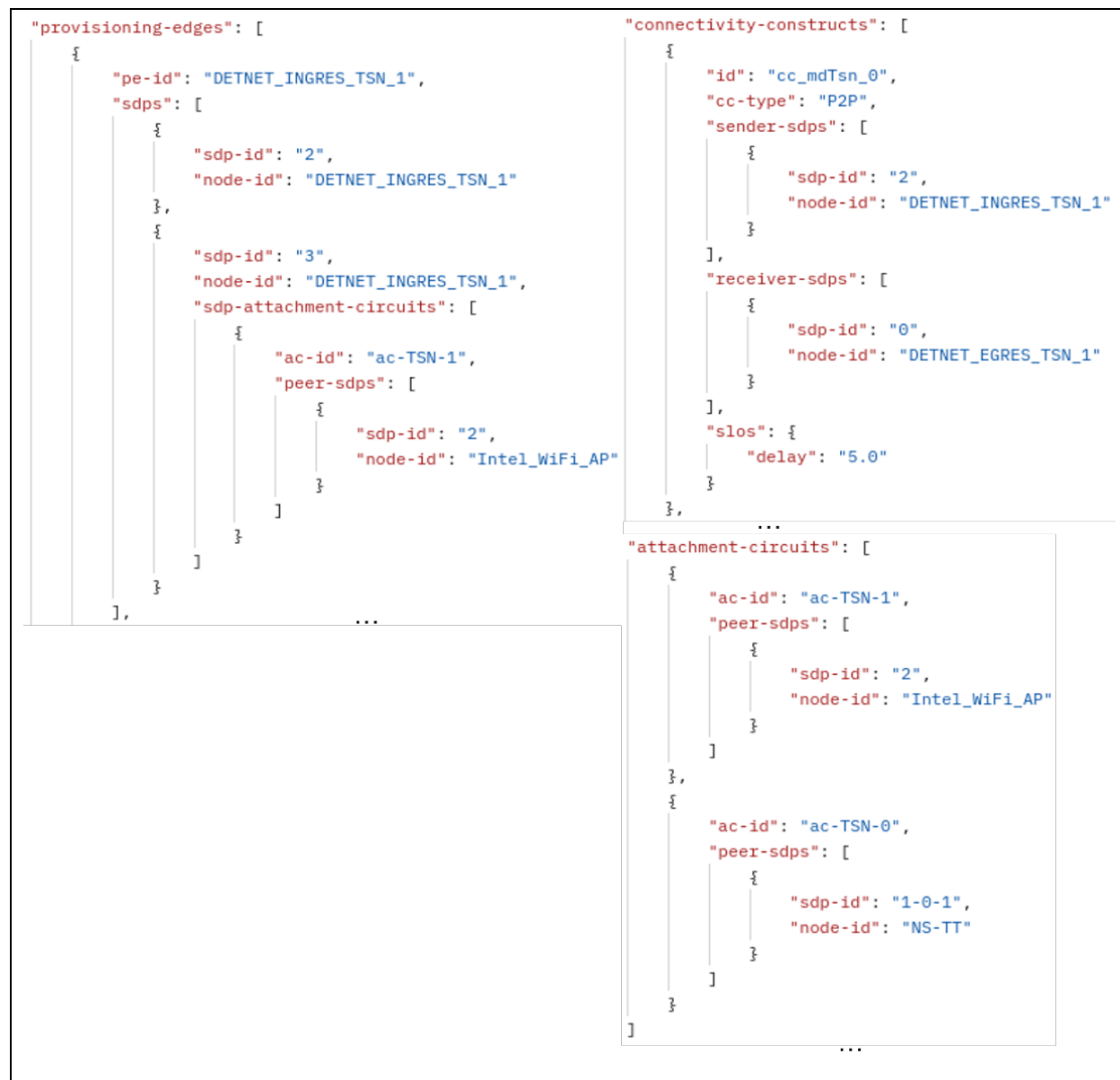
Extract of the topology (enriched with resource and capabilities information) of the TSN domain as obtained upon a request to the local MD Exposure implemented by the Resource Configurator.

<pre> { "links": [{ "bandwidth": "1Gbps", "latency": "0.001ms", "port": "2", "source": "DETNET_INGRES_TSN_1", "target": "Rely_Switch1", "type": "Intradomain" }, { "bandwidth": "1Gbps", "latency": "0.001ms", "port": "2", "source": "Rely_Switch1", "target": "DETNET_INGRES_TSN_1", "type": "Intradomain" }, ...] } </pre>	<pre> { "nodes": [{ "capabilities": "Qbv", "domain": "TSN", "id": "DETNET_INGRES_TSN_1" }, { "capabilities": "Qbv", "domain": "TSN", "id": "Rely_Switch1" }, { "capabilities": "Qbv", "domain": "TSN", "id": "DETNET_EGRES_TSN_1" }] } </pre>
---	---

Test 2: WiFi MD Topology/Resource/Capabilities Exposure
Description
This test reports the operation of the Exposure MS for the WiFi domain when requested by other MD modules such as the MD PCE.
Execution steps
The Exposure MS collects the topology, resources and capabilities of the technological domain following the information model defined in (PREDICT-6G/D2.3, 2025) and sends it to the requester.
Results
Extract of the topology (enriched with resource and capabilities information) of the WiFi domain as obtained upon a request to the local MD Exposure implemented by the Resource Configurator.



Test 3: E2E Topology Collection / MD Abstracted Topology Exposure
Description
This test reports the operation of the Exposure MS for the E2E when requested by another module such as the E2E PCE.
Execution steps
The E2E Topology Exposure MS contact the MD Exposure services to collect the abstracted topology information of the technological domains, following the abstracted topology information model defined in section 2.3.
Results
Extract of the abstract topology of the TSN domain as collected from the local MD Exposure (implemented by the MD Path Computation).



Service Automation

Figure 7-1 highlights in orange the set of modules that have been emulated and the steps taken as part of the testing.

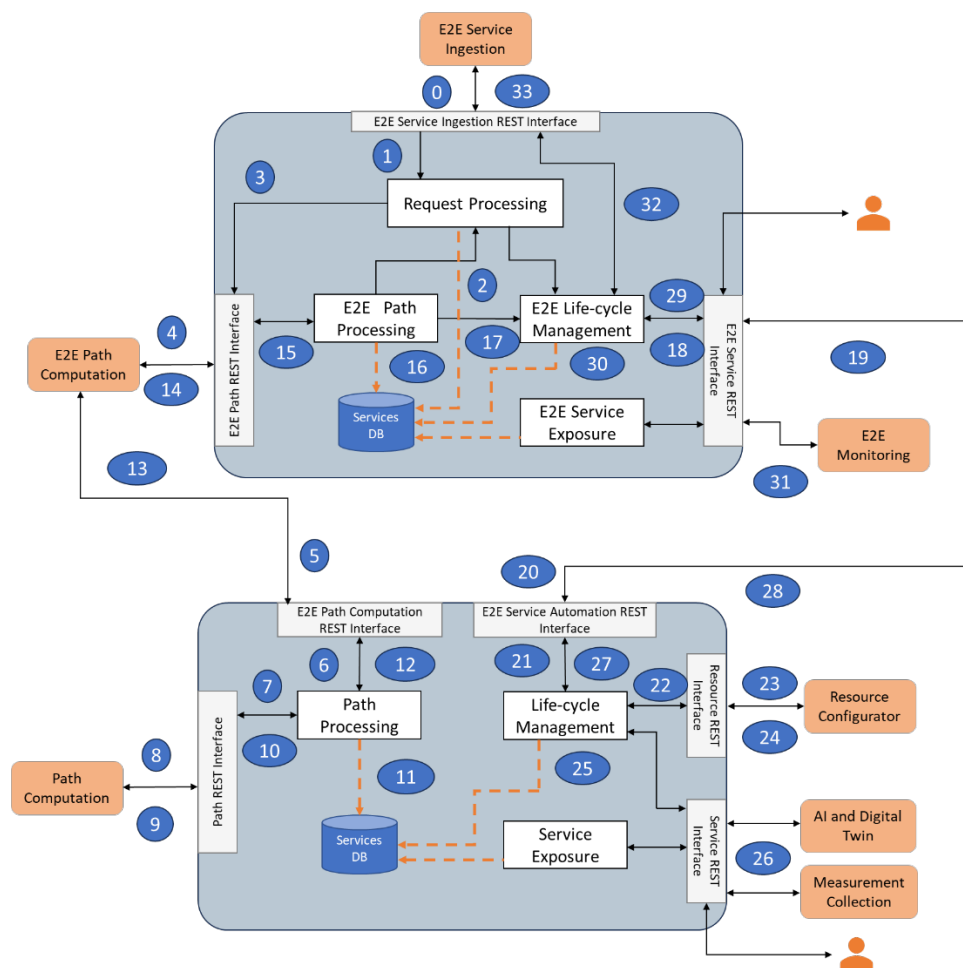


Figure 7-1. E2E and MD Service Automation Architecture - Tests and External Modules

Considering these conditions, the following tables describe the set of unitary tests performed to validate the implementation of E2E and MD SA software modules in the provisioning and decommissioning stages.

Test 1: E2E Service Ingestion → E2E Path Computation (Provisioning)
Description
<p>This test focuses on validating the use of the E2E Service Ingestion REST interface, the E2E Request Processing function and the E2E Path REST Interface for processing E2E Service Requests during the provisioning phase and triggering E2E Path Computation.</p> <p>The expected results of the test are:</p>

- 1) To have a new E2E Service entry in the Services DB with the information processed from the E2E Service Ingestion.
- 2) Receive a 200 OK response from the E2E Path Computation confirming that the request including E2E Service information has been correctly received.

Execution steps

This test has been performed executing the following steps:

Step 0: E2E Service Ingestion mock-up sends a new E2E Service Provisioning Request with a POST method over the endpoint *E2E_SA_URL/e2erequestprocessing/e2eprovisioningrequest*.

Step 1: The E2E Request Processing function of the E2E Service Automation module checks and assesses the format of the request.

Step 2: If the format is correct, the E2E Request Processing creates an E2E Service entry in the Services DB, assigning an E2E Service ID derived from the received request.

Steps 3 and 4: After creating the E2E Service entry, the E2E Request Processing function forwards the E2E Service information to the E2E Path Computation mock-up to start the computing the path in the E2E domain. The information is forwarded calling the endpoint *E2E_PATH_URL/e2eservice/pathrequest* with a POST method.

Results

```
INFO: Will watch for changes in these directories: ['/usr/src/e2e-service-automation']
INFO: Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO: Started reloader process [1] using StatReload
INFO: Started server process [8]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: 10.1.118.198:55084 - "POST /e2erequestprocessing/e2eprovisioningrequest HTTP/1.1" 200 OK
```

```
{
  "id": "33382",
  "e2e_path_selection": null,
  "traffic_characteristics": {
    "period": 20,
    "direction": "directional",
    "burst_size": 1500,
    "periodicity": true,
    "maximum_flow_bitrate": 5000
  },
  "md_services": null,
  "qos_characteristics": {
    "rtt": 30,
    "delay": 3,
    "jitter": 10,
    "priority": 7,
    "packet_loss": 100,
    "reliability": 100,
    "burst_arrival_time_window": {
      "t1": 0,
      "t2": 5
    },
    "burst_completion_time_window": {
      "t1": 0,
      "t2": 5
    }
  },
  "service_status": null,
  "endpoints": {
    "source": {
      "id": "PC_Talker"
    },
    "destination": {
      "id": "PC_Listener"
    }
  }
},
```

Test 2: E2E Path Computation → Path Processing → MD Path Computation → Path Processing → E2E Path Computation (Steps 5 to 13)

Description

This test focuses on validating the E2E Path Computation REST interface, the MD Path Processing function and the Past REST Interface for managing the MD Path Computation process as part of the E2E Path Computation process.

The expected results of the test are:

- 1) Receive a 200 OK response from the MD Path Computation confirming the correct reception of the MD Path Computation request.
- 2) Store a new MD Service in the Services DB with the MD Path Computation information.

- 3) Receive a 200 OK response from the E2E Path Computation confirming the correct reception of the MD Path Computation result.

Execution steps

This test has been performed executing the following steps:

Steps 5 and 6: E2E Path Computation triggers the computation of paths in each MD by calling a POST method over the endpoint *MD_SA_URL/path/mdservice/pathrequest*, at the E2E Path Computation REST Interface.

Steps 7 and 8: The Path Processing function validates the request format, assigns an ID to the MD Service and forwards the MD Service information to the MD Path Computation module via the Path REST Interface, by calling the endpoint *PATH_COMP_URL/mdservice/pathrequest* with a POST method.

Steps 9 and 10: The MD Path Computation computes the path for the MD Service and returns the result via the Path REST Interface, by calling the endpoint *MD_SA_URL/path/mdservice/pathresponse* with a POST method.

Steps 11, 12 and 13: If the path was successfully computed, the Path Processing function stores the MD Service information with the computed path in the Services DB and informs the E2E Path Computation of the result via the E2E Path Computation REST Interface, by calling the endpoint *E2E_PATH_COMP_URL/mdservice/pathcomputationresponse* with a POST.

Results

```

"td_path_selection": {
  "path_id": "7",
  "path_status": "computed",
  "involved_nodes": {
    "node_list": [
      {
        "id": "PC_Talker",
        "node_resources": {
          "resource_list": {
          }
        }
      },
      {
        "id": "Rely_Switch1",
        "node_resources": {
          "resource_list": {
          }
        }
      }
    ]
  }
}

```

```

2025-01-27T16:05:43.116|INFO|rest_app|postpathrequest()|rest_app.py:46|Received request to forward to Path Computation service.
2025-01-27T16:05:43.203|INFO|rest_manager|addMDServiceID()|rest_manager.py:95|1124
2025-01-27T16:05:43.203|INFO|rest_manager|addMDServiceID()|rest_manager.py:99|1125
2025-01-27T16:05:43.204|INFO|rest_app|postpathrequest()|rest_app.py:54|MD ID Service Created: 1125
2025-01-27T16:05:43.279|INFO|rest_app|postpathrequest()|rest_app.py:58|Response from Path Computation:Path Computation (1125) in progress
INFO: 10.1.194.4:44088 - "POST /path/mdservice/pathrequest HTTP/1.1" 200 OK
2025-01-27T16:05:43.531|INFO|rest_app|postpathresponse()|rest_app.py:84|Response from E2E Path Computation service:Local Path Processing (1125) in progress
INFO: 10.1.194.4:42184 - "POST /path/mdservice/pathresponse HTTP/1.1" 200 OK

```

Test 3: E2E Path Computation → E2E Path Processing → E2E Lifecycle Management
Description
<p>This test focuses on validating the E2E Path REST interface and the E2E Path Processing function for processing E2E Path Computation responses during the provisioning phase.</p> <p>The expected results of the test are:</p> <ol style="list-style-type: none"> 1) If E2E Path Computation response confirms that the E2E Path has been computed, update the E2E Service entry with the computed path information 2) Reply with a 200 OK response to the E2E Path Computation confirming the correct reception and processing of the E2E Path Response. 3) Receive a 200 OK response from the E2E Lifecycle Management confirming that the E2E Service Provisioning has been triggered successfully.
Execution steps
<p>This test has been performed executing the following steps:</p> <p>Step 14: E2E Path Computation mock-up sends the outcome of the E2E path computation process to the E2E Path Processing function with a POST method over the endpoint <i>E2E_SA_URL/e2epathprocessing/e2eservice/{e2e_service_id}/e2epathresponse</i></p> <p>Step 15: The E2E Path Processing function of the E2E Service Automation module checks the response format and the outcome of the E2E path computation process.</p> <p>Step 16: If the E2E Path has been computed, the E2E Path Processing updates the E2E Service entry with the E2E path information and updates the path status to “computed”.</p> <p>Step 17: After updating the E2E Service entry, the E2E Path Processing triggers the E2E Service Provisioning by forwarding the E2E Service information to the E2E Lifecycle Management function by calling a POST method on the endpoint <i>E2E_SA_URL/e2elifecycle/provisione2eservice</i></p>
Results
<pre data-bbox="203 1388 516 1732">{ "id": "33382", "e2e_path_selection": { "id": "7", "domain_list": [{ "path_domains": { "name": "mdTsn", "class": "DetNet" } }], "path_status": "computed" } }</pre>

```
INFO: 10.1.118.254:53836 - "POST /provisione2eservice HTTP/1.1" 200 OK
2025-01-27T16:05:43.413|INFO|rest_app|provisionService()|rest_app.py:38|Received request to provision E2E service with ID 33382.
2025-01-27T16:05:43.516|INFO|rest_app|provisionService()|rest_app.py:49|Requested provisioning of E2E service with ID 33382 successfully
```

Test 4: E2E Lifecycle Management → Lifecycle Management → Resource Configurator

Description

This test focuses on validating the E2E Lifecycle Management function for translating E2E Service Requests into one or more MD Service Provisioning Requests for each MD that is part of the E2E. This translation implies one or more calls to the Lifecycle Management function of each MD, which is also tested during the triggering of the MD Service Provisioning process by contacting the Resource Configurator.

The expected results of the test are:

- 1) Generate a call to one or more MD Service Automation modules in each of the domains that are part of the E2E service.
- 2) Receive a 200 OK response from each MD Service Automation after contacting the Resource Configurator, confirming that the MD Service Provisioning has been triggered successfully.

Execution steps

This test has been performed executing the following steps:

Step 17: The E2E Lifecycle Management processes the E2E Service information and extracts the information of each MD Service to be provisioned within the E2E.

Steps 18-20: For each MD Service in the corresponding MD, the E2E Lifecycle Management triggers the MD Service Provisioning with a POST method over the *MD_SA_URL/lifecycle/provisionmdservicerequest* through the E2E Service REST and the E2E Service Automation REST interfaces.

Step 21: Lifecycle Management function receives and validates all the information related to the MD Service to be provisioned.

Steps 22-23: Lifecycle Management function forwards the MD Service information the Resource Configurator to start configuring resources via the Resource REST interface by calling the endpoint *RC_URL/save-request* with a POST method.

Results

```
2025-01-27T16:05:43.453|INFO|rest_app|provisionService()|rest_app.py:43|Received request to provision MD service with ID 1125.
2025-01-27T16:05:43.476|INFO|rest_app|provisionService()|rest_app.py:54|MD service with ID 1125 requested provisioning successfully
INFO: 10.1.194.4:38302 - "POST /lifecycle/provisionmdservicerequest HTTP/1.1" 200 OK
```

Test 5: Resource Configurator → Lifecycle Management → Measurement Collection + AI/DT + E2E Lifecycle Management
Description
<p>This test focuses on validating the Lifecycle Management function when receiving the result of the Resource Configuration. This leads to an update of the MD Service ID in the Services DB, the communication to the MD Measurement Collection and MD AI or Digital Twin modules, and the response to the E2E Lifecycle Management function informing about the result of the MD Service Provisioning process.</p> <p>The expected results of the test are:</p> <ol style="list-style-type: none"> 1) Update the MD Service information in Services DB with the resources assigned to the service during the provisioning phase. 2) Receive a 200 OK response from the MD Measurement Collection confirming that the monitoring has been successfully triggered. 3) Receive a 200 OK response from the AI/DT confirming that the MD Service Provisioning has been successfully communicated. 4) Receive a 200 OK response from the E2E Lifecycle Management confirming that the MD Service Provisioning response has been received successfully.
Execution steps
<p>This test has been performed executing the following steps:</p> <p>Step 24: The MD Lifecycle Management function receives and processes the Resource Configurator response, which is sent via the Resource REST Interface by calling the endpoint <i>MD_SA_URL/lifecycle/provisionmdserviceresponse/{service_id}</i> with a PUT method.</p> <p>Step 25: The MD Lifecycle Management function updates the MD Service information in the Services DB.</p> <p>Step 26: The MD Lifecycle Management triggers the monitoring of the MD service by contacting the MD Measurement Collection via the Service REST interface by calling the endpoint <i>MD_MEAS_URL/datasources/MDP</i> with a POST method. Additionally, it also informs the AI/DT of the service provisioning through the same interface with a POST on the endpoint: <i>MD_DT_URL/Service</i>.</p> <p>Step 27-29: The MD Service Provisioning response is sent back to the E2E Lifecycle Management via the E2E Service Automation REST Interface with a POST call over the endpoint <i>E2E_SA_URL/e2elifecycle/provisionedmdservice</i>.</p>
Results

```
"involved_nodes": {
  "node_list": [
    {
      "id": "PC_Talker",
      "node_resources": {
        "resource_list": {
          "p1": {
            "status": "express",
            "priority": "0"
          }
        }
      }
    },
    {
      "id": "Rely_Switch1",
      "node_resources": {
        "resource_list": {
          "PORT_0": {
            "status": "express",
            "priority": "0"
          },
          "PORT_1": {
            "status": "express",
            "priority": "0"
          },
          "PORT_2": {
            "status": "express",
            "priority": "0"
          }
        }
      }
    }
  ]
}
```

```
2025-01-27T16:05:48.825|INFO|rest_app|assignmgmt()|rest_app.py:85|Assigning management for MD service with ID: 1125
2025-01-27T16:05:49.263|INFO|rest_manager|triggerMonitoring()|rest_manager.py:206|Monitoring triggered successfully.
2025-01-27T16:05:49.263|INFO|rest_manager|notifyDT()|rest_manager.py:272|Notifying MD DT for service ID 1125.
2025-01-27T16:05:49.326|INFO|rest_manager|notifyDT()|rest_manager.py:276|Notification sent successfully to DT. Response: 200
2025-01-27T16:05:49.919|INFO|rest_app|assignmgmt()|rest_app.py:99|Service 1125 successfully provisioned, monitoring triggered and DT notified.
INFO: 10.1.194.4:38306 - "PUT /Lifecycle/provisionmdserviceresponse/1125 HTTP/1.1" 200 OK
```

Test 6: E2E Lifecycle Management – E2E Monitoring – E2E Service Ingestion

Description

This test focuses on validating the E2E Lifecycle Management function when receiving the MD Service Provisioning responses for each domain to compose the E2E Service Provisioning response, trigger the E2E Monitoring and forward the outcome to the E2E Service Ingestion to inform about the result of the initial E2E Service Provisioning request.

The expected results of the test are:

- 1) Update the E2E Service information in Services DB setting the E2E status as provisioned once all responses from MD Service Automations are received.
- 2) Receive a 200 OK response from E2E Measurement Collection confirming that the E2E monitoring has been successfully triggered.
- 3) Receive a 200 OK response from the E2E Service Ingestion confirming that the E2E Service Provisioning response has been received successfully.

Execution steps

This test has been performed executing the following steps:

Step 30: The E2E Lifecycle management receives all MD Service Provisioning responses from the corresponding MDs and updates the E2E Service status to provisioned in the E2E Service stored in Services DB.

Step 31: The E2E Lifecycle Management triggers the E2E Monitoring via the E2E Service REST Interface by calling the endpoint *E2E_MON_URL/datasources/E2E* with a POST method.

Steps 32-33: The E2E Lifecycle Management forwards the result of the E2E Service Provisioning to the E2E Service Ingestion via the E2E Service Ingestion REST Interface by calling the endpoint *E2E_INGESTION_URL/service-provisioning-notification* through a POST method.

Results

```
"service_status": "provisioned",
"endpoints": {
  "source": {
    "id": "PC_Talker"
  },
  "destination": {
    "id": "PC_Listener"
  }
},
}
```

```
"md_services": {
  "local_service": [
    {
      "index": 0,
      "service": {
        "id": "1125",
        "target_domain": {
          "name": "mdTsn",
          "class": "DetNet"
        },
        "service_status": "provisioned"
      }
    }
  ]
},
}
```

```
INFO: 10.1.118.254:53836 - "POST /provisione2eservice HTTP/1.1" 200 OK
2025-01-27T16:05:43.413|INFO|rest_app|provisionService()|rest_app.py:38|Received request to provision E2E service with ID 33382.
2025-01-27T16:05:43.516|INFO|rest_app|provisionService()|rest_app.py:49|Requested provisioning of E2E service with ID 33382 successfully.
INFO: 10.1.118.254:34000 - "POST /provisione2eservice HTTP/1.1" 200 OK
INFO: 10.1.118.198:52408 - "POST /e2elifecycle/provisionedmdservice/ HTTP/1.1" 307 Temporary Redirect
2025-01-27T16:05:49.362|INFO|rest_app|provisionedService()|rest_app.py:88|Received response for provisioned MD service with ID 1125.
2025-01-27T16:05:49.658|INFO|rest_manager|triggerMonitoring()|rest_manager.py:222|E2E Monitoring triggered successfully.
2025-01-27T16:05:49.659|INFO|rest_manager|notifyStatusE2EDT()|rest_manager.py:193|Sending status of the provision to: http://e2epathcomputation-service:9506/e2eservice
2025-01-27T16:05:49.796|INFO|rest_manager|notifyStatusE2EServiceIngestion()|rest_manager.py:170|Sending status of the provision to: http://10.5.15.65:8300/s
service-provisioning-notification
2025-01-27T16:05:49.872|INFO|rest_app|provisionedService()|rest_app.py:99|MD service with ID 1125 provisioned successfully.
INFO: 10.1.118.198:52408 - "POST /e2elifecycle/provisionedmdservice HTTP/1.1" 200 OK
```

Test 7: E2E and MD Service Exposure
<p>Description</p> <p>This test focuses on validating the E2E and MD Service Exposure functions to expose E2E and MD Service information after querying the ServicesDB.</p> <p>The expected results of the test are:</p> <ol style="list-style-type: none"> 1) Receive a 200 OK response from the E2E Service Exposure function with the information of the requested E2E/MD service.
<p>Execution steps</p> <p>This test has been performed executing the following steps:</p> <p>Step 34: A request is sent to the E2E/MD Service Exposure functions with the ID of a specific E2E or MD Service to retrieve. For the E2E Service Exposure case, this query comes via the E2E Service REST Interface with a GET to the endpoint <i>E2E_SA_URL/e2eserviceexposure/e2eservice/{e2e_service_id}</i>. For the MD Service Exposure case, the query goes via the Service REST Interface with a GET to the endpoint <i>MD_SA_URL/exposure/mdservice/{service_id}</i>.</p> <p>Step 35: The E2E/MD Service Exposure functions forward this request to the ServicesDB, checking if the E2E/MD Service exists in the database.</p> <p>Step 36: The E2E/MD Service information is exposed via the E2E Service/Service REST Interface.</p>
<p>Results</p> <pre>INFO: 10.1.118.198:59012 - "GET /e2eserviceexposure/e2eservice/33382 HTTP/1.1" 200 OK 2025-02-17T10:19:14.112 INFO rest_app getService() rest_app.py:39 Received request to fetch TD service with ID: 1125 2025-02-17T10:19:14.156 INFO rest_app getService() rest_app.py:45 Successfully retrieved service with ID: 1125 INFO: 10.1.194.4:40546 - "GET /exposure/mdservice/1125 HTTP/1.1" 200 OK</pre>

Complementing all the tests performed during the E2E and MD Service provisioning phase, one additional test is defined to showcase the E2E and MD Service decommissioning, highlighting the main differences performed with respect to the previous workflow (Figure 7-2). Note that only one test is used to describe the decommissioning since several steps are similar to the ones taken during the provisioning phase:

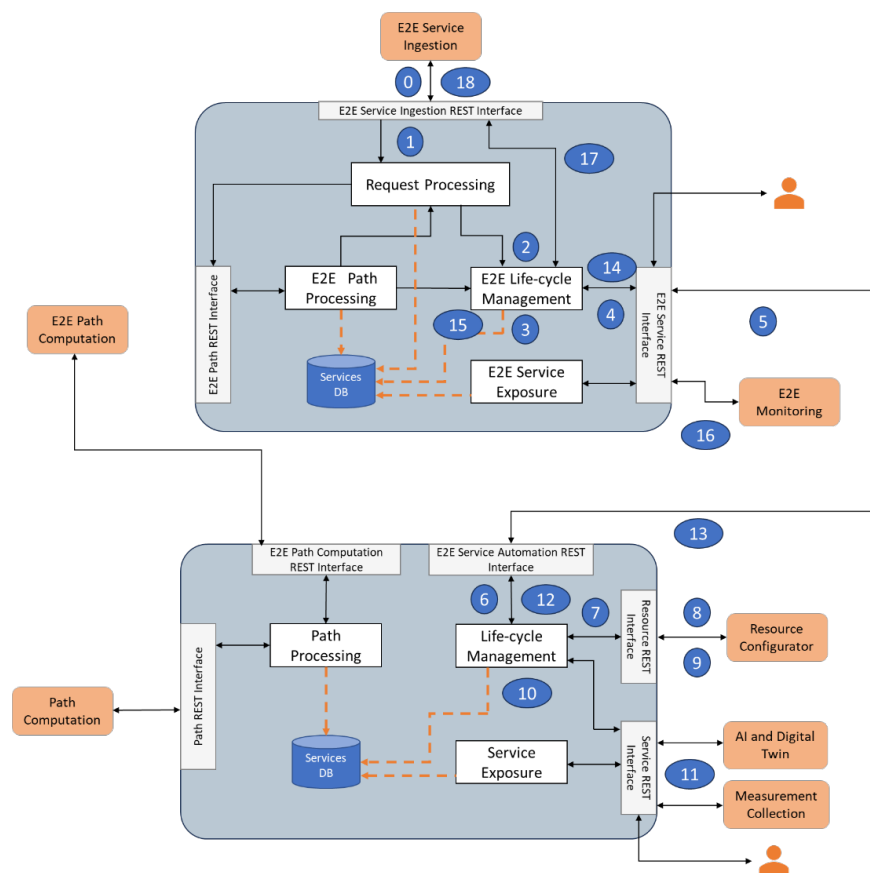


Figure 7-2. E2E and MD Service Automation Architecture - Service decommissioning tests

Test 8: E2E and MD Service Decommissioning

Description

This test focuses on validating the decommissioning of the E2E and MD services generated in the previous tests. The main differences with respect to the provisioning are:

1. The E2E Request Processing function forwards the decommissioning request coming from the E2E Service Ingestion to the E2E Lifecycle Management function since path computation is not required to decommission a service (Steps 0 -3)
2. The E2E Lifecycle Management function follows a similar approach to translate the E2E Service Decommissioning request into several MD Service Decommissioning requests (Steps 4-6).
3. The Resource Configurator releases resources instead of assigning during the MD Service Decommissioning phase (Steps 7-9).

4. After releasing the resources and decommissioning the MD Service, the MD Lifecycle Management stops the measurement collection and informs the AI/DT modules (Steps 10-11).
5. A similar operation is performed in the E2E domain, where the E2E Monitoring is stopped once the E2E Service Decommissioning has been confirmed. After that, the E2E Decommissioning result is sent to the E2E Service Ingestion (Step 12-18).

The expected results of the test are:

- 1) Update the E2E and MD Service entries in the Services DB by setting the status to decommissioned and the resources being released.
- 2) Receive a 200 OK response from the E2E Lifecycle Management function confirming that the E2E Service Decommissioning has been performed successfully.

Execution steps

This test has been performed executing the following steps:

Steps 0-3: E2E Service Ingestion mock-up sends a new E2E Service Decommissioning Request with a DELETE method over the endpoint *E2E_SA_URL/e2erequestprocessing/e2edecommissioningrequest/{e2e_service_id}*. The E2E Request Processing function of the E2E SA module checks the request format, if the E2E service exists and if it has a provisioned status. If so, the E2E Request Processing forwards the request to the E2E Lifecycle Management to trigger the decommissioning of this E2E service. This request is forwarded calling the endpoint *E2E_SA_URL/decommissione2eservice/{e2e_service_id}* with a DELETE method.

Steps 4-6: The E2E Lifecycle Management processes the E2E service information and extracts the information of each MD service to be decommissioned. For each MD service, the E2E Lifecycle Management triggers the MD Service Decommissioning with a DELETE method over the *MD_SA_URL/lifecycle/decommissionmdservicerequest* through the E2E Service REST and the E2E SA REST interfaces.

Steps 7-9: Lifecycle Management function receives and validates all the information related to the MD service to be decommissioned. Then, forwards the MD service information the Resource Configurator to start releasing resources via the Resource REST interface by calling the endpoint *RC_URL/delete-request* with a DELETE method.

Steps 10-11: The Resource Configurator releases the resources and informs the MD Lifecycle Management function via the Resource REST Interface by calling the endpoint *MD_SA_URL/lifecycle/decommissionmdservicerresponse/{service_id}* with a PUT method. Then, the MD Lifecycle Management function updates service information in the Services DB and stops the monitoring of the MD service via the Service REST interface by calling the endpoint *MD_MEAS_URL/datasources/MDP?service_id={service_id}* with a GET method to get the datasource_id and then the endpoint *MD_MEAS_URL/datasources/MDP/id/{datasource_id}* with a DELETE method to stop the

monitoring. Additionally, it also informs the AI/DT of the service decommissioning with a POST on the endpoint *MD_DT_URL/Service*.

Steps 12-18: The MD Service Decommissioning response is sent back to the E2E Lifecycle Management via the E2E SA REST Interface with a DELETE call over the endpoint *E2E_SA_URL/e2elifecycle/decommissionedmdservice*. Once all MD Service Decommissioning responses are received, the E2E Lifecycle Management updates the E2E Service status to decommissioned, stops the E2E Monitoring and forwards the result to the E2E Service Ingestion via the E2E Service Ingestion REST Interface by calling the endpoint *E2E_INGESTION_URL/service-decommissioning-notification* through a POST method. The monitoring is stopped via by calling the endpoint *E2E_MEAS_URL/datasources/E2E?service_id={e2e_service_id}* with a GET method to get the *datasource_id* and then the endpoint *E2E_MEAS_URL/datasources/E2E/id/{datasource_id}* with a DELETE method

Results

```
"service_status": "decommissioned",
"endpoints": {
  "source": {
    "id": "PC_Talker"
  },
  "destination": {
    "id": "PC Listener"
  }
},
```

```
"md_services": {
  "local_service": [
    {
      "index": 0,
      "service": {
        "id": "1125",
        "target_domain": {
          "name": "mdTsn",
          "class": "DetNet"
        },
        "service_status": "decommissioned"
      }
    }
  ]
},
```

```
2025-01-27T16:35:09.238|INFO|rest_app|decommissionService()|rest_app.py:63|Received request to decommission E2E service with ID 33382.
2025-01-27T16:35:09.331|INFO|rest_manager|decommissionService()|rest_manager.py:94|Decommissioning response for local service: True
2025-01-27T16:35:09.331|INFO|rest_app|decommissionService()|rest_app.py:74|Requested decommissioning of E2E service with ID 33382 successfully.
INFO: 10.1.118.252:45618 - "DELETE /decommissionedmdservice/33382 HTTP/1.1" 200 OK
INFO: 10.1.118.198:42684 - "POST /e2elifecycle/decommissionedmdservice/ HTTP/1.1" 307 Temporary Redirect
2025-01-27T16:35:09.710|INFO|rest_app|decommissionedService()|rest_app.py:113|Received response for decommissioned MD service with ID 1125.
2025-01-27T16:35:09.834|INFO|rest_manager|triggerMonitoring()|rest_manager.py:235|E2E Monitoring stopped successfully.
2025-01-27T16:35:09.835|INFO|rest_manager|notifyStatusE2EDT()|rest_manager.py:193|Sending status of the provision to: http://e2epathcomputation-service:9500/e2eservice
2025-01-27T16:35:09.851|INFO|rest_manager|notifyStatusE2EServiceIngestion()|rest_manager.py:170|Sending status of the provision to: http://10.5.15.65:8300/service-decommissioning-notification
2025-01-27T16:35:09.902|INFO|rest_app|decommissionedService()|rest_app.py:124|MD service with ID 1125 decommissioned successfully.
INFO: 10.1.118.198:42684 - "POST /e2elifecycle/decommissionedmdservice HTTP/1.1" 200 OK
```

Path Computation

Test 1: Local MD Path Computation – Technological Domain Exposure (TSN domain)	
Description	
This test reports the collection of the topology, resources and capabilities from the local domain Exposure MS.	
Execution steps	
On boot-up, the Path Computation MS of the local technological domain contacts the corresponding Exposure MS to collect the topology following the model defined for the topology exposure OpenAPI (PREDICT-6G/D2.3, 2025).	
Results	
The figure below depicts the extract of the log of the local MD Path Computation that contains the topology processing.	

```
e.u.g.p.p.core.control.TopologyFunction : Getting Topology from the Exposure MS
e.u.g.p.p.c.t.exposure.TextRestClient : Getting topology to Digital Twin
e.u.g.p.p.c.topology.TopologyTranslator : Treating Link = PC Listener-Rely_Switch4
e.u.g.p.p.c.topology.TopologyTranslator : Looking for reverse link
e.u.g.p.p.c.topology.TopologyTranslator : Adding DstTp to Link PC Listener-Rely_Switch4
e.u.g.p.p.c.topology.TopologyTranslator : Treating Link = PC Talker-Rely_Switch1
e.u.g.p.p.c.topology.TopologyTranslator : Looking for reverse link
e.u.g.p.p.c.topology.TopologyTranslator : Adding DstTp to Link PC Talker-Rely_Switch1
e.u.g.p.p.c.topology.TopologyTranslator : Treating Link = Rely_Switch1-PC Talker
e.u.g.p.p.c.topology.TopologyTranslator : Treating Link = Rely_Switch4-Rely_Switch3
e.u.g.p.p.c.topology.TopologyTranslator : Looking for reverse link
e.u.g.p.p.c.topology.TopologyTranslator : Adding DstTp to Link Rely_Switch4-Rely_Switch3
e.u.g.p.p.c.topology.TopologyTranslator : Treating Link = Rely_Switch4-PC Listener
e.u.g.p.p.c.topology.TopologyTranslator : Treating Link = Rely_Switch1-Rely_Switch2
e.u.g.p.p.c.topology.TopologyTranslator : Looking for reverse link
e.u.g.p.p.c.topology.TopologyTranslator : Adding DstTp to Link Rely_Switch1-Rely_Switch2
e.u.g.p.p.c.topology.TopologyTranslator : Treating Link = Rely_Switch2-Rely_Switch3
e.u.g.p.p.c.topology.TopologyTranslator : Looking for reverse link
e.u.g.p.p.c.topology.TopologyTranslator : Adding DstTp to Link Rely_Switch2-Rely_Switch3
e.u.g.p.p.c.topology.TopologyTranslator : Treating Link = Rely_Switch3-Rely_Switch2
e.u.g.p.p.c.topology.TopologyTranslator : Treating Link = Rely_Switch2-Rely_Switch1
e.u.g.p.p.c.topology.TopologyTranslator : Treating Link = Rely_Switch3-Rely_Switch4
e.u.g.p.p.core.control.TopologyFunction : TextTopology = Topology [nodeList={Rely_Switch2=Node [id=Rely_Switch2, type=SW, tpList={0=TPtsn
[maxDataRate=1000, bufferSize=1000, id=0, type=WIRELESS_TSN], 1=TPtsn [maxDataRate=1000, bufferSize=1000, id=1, type=WIRELESS_TSN]}, domain=Domain [id=mdTsn, type=TSN]], PC_Talker=Node [id=PC_Talker, type=SW,
tpList={2=TPtsn [maxDataRate=1000, bufferSize=1000, id=2, type=WIRELESS_TSN]}, domain=Domain [id=mdTsn, type=TSN]], Rely_Switch1=Node
[id=Rely_Switch1, type=SW, tpList={0=TPtsn [maxDataRate=1000, bufferSize=1000, id=0, type=WIRELESS_TSN], 2=TPtsn [maxDataRate=1000, bufferSize=1000, id=2,
type=WIRELESS_TSN]}, domain=Domain [id=mdTsn, type=TSN]], Rely_Switch4=Node [id=Rely_Switch4, type=SW, tpList={0=TPtsn [maxDataRate=1000, bufferSize=1000, id=0, type=WIRELESS_TSN], 1=TPtsn [maxDataRate=1000, bufferSize=1000, id=1, type=WIRELESS_TSN]}, domain=Domain [id=mdTsn, type=TSN]], Rely_Switch3=Node [id=Rely_Switch3, type=SW, tpList={0=TPtsn [maxDataRate=1000, bufferSize=1000, id=0, type=WIRELESS_TSN], 1=TPtsn [maxDataRate=1000, bufferSize=1000, id=1, type=WIRELESS_TSN]}, domain=Domain [id=mdTsn, type=TSN]], PC_Listener=Node [id=PC_Listener, type=SW, tpList={0=TPtsn [maxDataRate=1000, bufferSize=1000, id=0, type=WIRELESS_TSN]}, domain=Domain [id=mdTsn, type=TSN]], linkList={PC_Listener-Rely_Switch4=Link [id=PC_Listener-Rely_Switch4, srcTp=TPtsn [maxDataRate=1000, bufferSize=1000, id=0, type=WIRELESS_TSN], dstTp=TPtsn [maxDataRate=1000, bufferSize=1000, id=2, type=WIRELESS_TSN], srcTp=TPtsn [maxDataRate=1000, bufferSize=1000, id=2, type=WIRELESS_TSN], dstTp=TPtsn [maxDataRate=1000, bufferSize=1000, id=0, type=WIRELESS_TSN], srcTp=TPtsn [maxDataRate=1000, bufferSize=1000, id=1, type=WIRELESS_TSN], dstTp=TPtsn [maxDataRate=1000, bufferSize=1000, id=0, type=WIRELESS_TSN]}, domainList={mdTsn=Domain [id=mdTsn, type=TSN]]}
```

Test 2: Local MD Path Computation – Path computation (TSN domain)

Description

This test reports the local path computation process as conducted in the Path Computation MS.

Execution steps

Upon the reception of a path computation request from the Service Automation MS, the Path Computation MS calculates a route between the endpoints provided in the request. The KPIs estimation for the obtained path is requested to the local domain DT and, if they are acceptable (that is under the thresholds stated in the request), the path is sent back to the Service Automation MS.

Results

The figure below depicts the extract of the log of the local MD Path Computation that contains the path computation and KPI validation process.

[illegible]

Test 3: E2E Path computation – Abstract topology composition

Description

This test reports the collection of the abstract topology from the different local domains Exposure MSs. As explained in section 2.3, the local domain abstract topology exposure is implemented by the local MD Path Computation.

Execution steps

On boot-up, the E2E Path Computation MS contacts the corresponding Exposure MSs to collect the abstract topology of each domain following the model defined for the abstract topology exposure (section 2.3). Afterwards, the E2E Path Computation builds the complete abstract topology.

Results

The figure below depicts the extract of the log of the E2E Path Computation that contains the topology processing.

```
e.u.g.p.p.core.control.TopologyFunction : Getting Abstract Topology from the MD Exposure MSs to compose E2E
e.u.g.p.p.c.t.exposure.TextRestClient : Requesting Abstract Topology to the MD Service Automation
e.u.g.p.p.c.t.exposure.TextRestClient : Requesting Abstract Topology to the MD Service Automation
e.u.g.p.p.c.t.exposure.TextRestClient : Requesting Abstract Topology to the MD Service Automation
e.u.g.p.p.c.t.i.TopologyTranslator : Treating AC = class AttachmentCircuit {
  id: ac-3GPP-0
  peerSdps: [class Sdp {
    id: 3
    nodeId: PC_Listener
  }]
}
e.u.g.p.p.c.t.i.TopologyTranslator : Creating Direct Link from NS-TT/1-0-1
e.u.g.p.p.c.t.i.TopologyTranslator : Creating Reverse Link to NS-TT/1-0-1
e.u.g.p.p.c.t.i.TopologyTranslator : Treating AC = class AttachmentCircuit {
  id: ac-TSN-1
  peerSdps: [class Sdp {
    id: 2
    nodeId: Intel_WiFi_AP
  }]
}
e.u.g.p.p.c.t.i.TopologyTranslator : Treating Link = Link [id=ac1, srcTp=null, dstTp=TpTsn [maxDataRate=1, bufferSize=1, id=1-0-1,
type=WIRED_TSN]]srcNode = srcNodeTp = dstNode = NS-TTdsrcTp = 1-0-1
e.u.g.p.p.c.t.i.TopologyTranslator : Treating Link = Link [id=ac0, srcTp=TpTsn [maxDataRate=1, bufferSize=1, id=1-0-1,
type=WIRED_TSN], dstTp=null]srcNode = NS-TTsrcNodeTp = 1-0-1dstNode = dstTp =
e.u.g.p.p.c.t.i.TopologyTranslator : Creating Direct Link from PC_Talker/3
e.u.g.p.p.c.t.i.TopologyTranslator : Creating Reverse Link to PC_Talker/3
e.u.g.p.p.c.t.i.TopologyTranslator : Treating AC = class AttachmentCircuit {
  id: ac-TSN-0
  peerSdps: [class Sdp {
    id: 1-0-1
    nodeId: NS-TT
  }]
}
e.u.g.p.p.c.t.i.TopologyTranslator : Treating Link = Link [id=ac3, srcTp=null, dstTp=TpTsn [maxDataRate=1, bufferSize=1, id=3,
type=WIRED_TSN]]srcNode = srcNodeTp = dstNode = PC_TalkerdsrcTp = 3
e.u.g.p.p.c.t.i.TopologyTranslator : Treating Link = Link [id=ac2, srcTp=TpTsn [maxDataRate=1, bufferSize=1, id=3, type=WIRED_TSN],
dstTp=null]srcNode = PC_TalkersrcNodeTp = 3dstNode = dstTp =
e.u.g.p.p.c.t.i.TopologyTranslator : Treating Link = Link [id=ac1, srcTp=null, dstTp=TpTsn [maxDataRate=1, bufferSize=1, id=1-0-1,
type=WIRED_TSN]]srcNode = srcNodeTp = dstNode = NS-TTdsrcTp = 1-0-1
e.u.g.p.p.c.t.i.TopologyTranslator : Treating Link = Link [id=ac0, srcTp=TpTsn [maxDataRate=1, bufferSize=1, id=1-0-1,
type=WIRED_TSN], dstTp=null]srcNode = NS-TTsrcNodeTp = 1-0-1dstNode = dstTp =
e.u.g.p.p.c.t.i.TopologyTranslator : Direct Link found from PC_Listener/3 to NS-TT/1-0-1
e.u.g.p.p.c.t.i.TopologyTranslator : Reverse Link found to PC_Listener/3 from NS-TT/1-0-1
e.u.g.p.p.c.t.i.TopologyTranslator : Treating AC = class AttachmentCircuit {
  id: ac-WiFi-0
  peerSdps: [class Sdp {
    id: 3
    nodeId: PC_Talker
  }]
}
e.u.g.p.p.c.t.i.TopologyTranslator : Treating Link = Link [id=ac3, srcTp=null, dstTp=TpTsn [maxDataRate=1, bufferSize=1, id=3,
type=WIRED_TSN]]srcNode = srcNodeTp = dstNode = PC_TalkerdsrcTp = 3
```

Test 4: E2E Path Computation – Domain sequence computation

Description

This test reports the domain sequence computation process conducted in the E2E Path Computation MS.

Execution steps

Upon the reception of an E2E path computation request from the F2EService Automation MS, the Path Computation MS calculates a route between the endpoints provided in the request over the abstract topology that is likely to fulfil the requested KPIs. From such computation, the E2E Path Computation extracts the domain sequence, and the border nodes involved. Then, it contacts the corresponding local domain Service Automation MSs to request an explicit route between the borders. When all the partial routes have been computed, the E2E Path Computation composes the E2E path and requests the KPI estimation to the E2E DT. If they are acceptable (that is under the thresholds stated in the request), the path is sent back to the E2E Service Automation MS for provisioning.

Results

The figure below depicts the extract of the log of the E2E Path Computation that contains the process.

```
e.u.g.p.p.core.pce.PathComputation      : Compute E2E Path between DS-TT and PC_Talker
e.u.g.p.p.core.pce.PathComputation      : Computed E2E Path
e.u.g.p.p.core.pce.PathComputation      : DS-TT - NS-TT
e.u.g.p.p.core.pce.pce.PathComputation  : NS-TT - PC_Listener
e.u.g.p.p.core.pce.pce.PathComputation  : PC_Listener - PC_Talker
e.u.g.p.p.core.pce.pce.PathComputation  : -----
e.u.g.p.p.core.pce.pce.PathComputation  : currentLink: DS-TT - NS-TT
e.u.g.p.p.core.pce.pce.PathComputation  : Positioning pathIt2
e.u.g.p.p.core.pce.pce.PathComputation  : nextLink: NS-TT - PC_Listener
e.u.g.p.p.core.pce.pce.PathComputation  : md3gpp vs mdTsn
e.u.g.p.p.core.pce.pce.PathComputation  : currentLink: PC_Listener - PC_Talker
e.u.g.p.p.core.pce.pce.PathComputation  : Domain Sequence:
e.u.g.p.p.core.pce.pce.PathComputation  : Treating Domain: md3gpp
e.u.g.p.p.core.pce.pce.PathComputation  : Treating Domain: mdTsn
e.u.g.p.p.core.pce.pce.PathComputation  : TdNodes DS-TT --> NS-TT
e.u.g.p.p.core.pce.pce.PathComputation  : Sending Local Path Computation Request to MD Service Automation
e.u.g.p.p.nbi.server.NBIPceService      : Processing Local Path for MD Service 1131 (10.5.15.62)
e.u.g.p.p.core.sa.SARestClient           : Reply = <200 OK OK,"",[Date:"Mon, 10 Feb 2025 13:54:34 GMT", Content-
Type:"application/json", Content-Length:"2", Connection:"keep-alive">
e.u.g.p.p.core.pce.pce.PathComputation  : TdNodes PC Listener --> PC_Talker
e.u.g.p.p.core.sa.SARestClient           : Sending Local Path Computation Request to MD Service Automation
e.u.g.p.p.core.sa.SARestClient           : Reply = <200 OK OK,"Path Computation (1132) in progress",[Date:"Mon, 10 Feb
2025 13:54:35 GMT", Content-Type:"application/json", Content-Length:"37", Connection:"keep-alive">
e.u.g.p.p.core.pce.pce.PathComputation  : -----
e.u.g.p.p.nbi.server.NBIPceService      : Processing Local Path for MD Service 1132 (10.5.15.61)
e.u.g.p.p.nbi.server.NBIPceService      : Sending E2E Path Reply to E2E-SA: E2EPathReply [id=33389, status=pending,
endPoints=EndPoints [source=Source [id=DS-TT], destination=Destination [id=PC_Talker]],
e2ePathSelection=E2EPathSelection [id=1, pathStatus=computed, domainList=[Domain [pathDomains=PathDomains
[name=md3gpp, domainClass=3GPP]], Domain [pathDomains=PathDomains [name=mdTsn, domainClass=DetNet]]]],
qosCharacteristics=QosCharacteristics [priority=7, reliability=100, packetLoss=10, delay=20, rtt=30, jitter=10,
burstArrivalTimeWindow={t1=0, t2=5}, burstCompletionTimeWindow={t1=0, t2=5}],
trafficCharacteristics=TrafficCharacteristics [direction=directional, periodicity=true, period=20, burstSize=500,
maxFlowBitrate=5000], serviceLifetime=ServiceLifetime [ServiceDuration=ServiceDuration [t1=0, t2=2],
recurrentServiceInterval=RecurrentServiceInterval [t1=0, t2=1, recurrenceInterval=1]], mdServices=MdServices
[localService=[LocalServiceEntry [index=0, service=Service [id=1131, serviceStatus=pending, targetDomain=PathDomains
[name=md3gpp, domainClass=3GPP]], LocalServiceEntry [index=1, service=Service [id=1132, serviceStatus=pending,
targetDomain=PathDomains [name=mdTsn, domainClass=DetNet]]]]]]
e.u.g.p.p.core.sa.SARestClient           : Sending Path Computation Result to E2E Service Automation
```

Resource Configurator

Test 1: Wi-Fi Resource Configurator: Service Provisioning / Decommissioning																	
Description																	
<p>This test has been carried out inside the WiFi-AP, where an incoming Service Provisioning Request from the MD SA was simulated to trigger the script to configure the queues. The characteristics of the flows are the following.</p> <table> <tr> <th><i>Param</i></th><th><i>Flow-1</i></th><th><i>Flow-2</i></th></tr> <tr> <td><i>Size</i></td><td>1000 bytes</td><td>1000 bytes</td></tr> <tr> <td><i>Period</i></td><td>50 ms</td><td>50 ms</td></tr> <tr> <td><i>Latency Constraint</i></td><td>1500 ms</td><td>50 ms</td></tr> <tr> <td><i>Type</i></td><td>Best Effort</td><td>TSN</td></tr> </table> <p>The MAC addresses and the transmission method (Unicast/Multicast) were also given. For the decommission step, the process is the same, but with an incoming Service Decommission from the MD SA.</p>			<i>Param</i>	<i>Flow-1</i>	<i>Flow-2</i>	<i>Size</i>	1000 bytes	1000 bytes	<i>Period</i>	50 ms	50 ms	<i>Latency Constraint</i>	1500 ms	50 ms	<i>Type</i>	Best Effort	TSN
<i>Param</i>	<i>Flow-1</i>	<i>Flow-2</i>															
<i>Size</i>	1000 bytes	1000 bytes															
<i>Period</i>	50 ms	50 ms															
<i>Latency Constraint</i>	1500 ms	50 ms															
<i>Type</i>	Best Effort	TSN															
Execution steps																	
<p>The incoming flow characteristics are stored into a .txt file. Then, the python3 tool is executed upon request. This python3 is able to write at the <i>qdisc</i> level, the queue cycle for each queue.</p> <p>Running commands:</p> <ul style="list-style-type: none"> -Provision: <code>python3 configTimerSolved.py -f flows.txt -i wlp2s0 -verbose</code> -Decommission: <code>python3 configTimerSolved.py -u -i wlp2s0</code> (When a Service Decommission Request is received) 																	
Results																	
<p>Here there is showcase the two possible cases under testing:</p> <p>Service Provisioning:</p>																	


```
netcom@netcom-1:~/IntelWiFiConfigurator$ python3 configTimerSolved.py -f flows.txt -i wlp2s0 --verbose
Device configured !!
```

```
Every 1,0s: sudo tc -s qdisc show dev wlp2s0

qdisc taprio 800a: root refcnt 3 tc 2 map 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
queues offset 0 count 1 offset 1 count 1
clockid TAI    base-time 1738917675862048277 cycle-time 50000000 cycle-time-extension 0
index 0 cmd S gatemark 0x1 interval 8000
index 1 cmd S gatemark 0x1 interval 8000
index 2 cmd S gatemark 0x2 interval 49984000

Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
backlog 0b 0p requeues 0
qdisc pfifo 0: parent 800a:2 limit 1000p
Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
backlog 0b 0p requeues 0
qdisc pfifo 0: parent 800a:1 limit 1000p
Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
backlog 0b 0p requeues 0
```

Service Decommissioning:

```
netcom@netcom-1:~/IntelWiFiConfigurator$ python3 configTimerSolved.py -u -i wlp2s0 --verbose
Configuration deleted !!
```

```
Every 1,0s: sudo tc -s qdisc show dev wlp2s0

qdisc noqueue 0: root refcnt 2
Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
backlog 0b 0p requeues 0
```

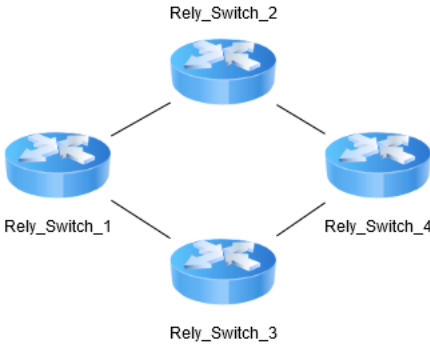
Test 2: Ethernet TSN Resource Configurator: Service Provisioning

Description

This test was carried out inside the ETH-TSN Resource Configurator, where all switches were connected. An incoming Service Provisioning Request from the MD SA was simulated to run the script to configure the queues of each switch. The characteristics of the flows are the following.

<i>Param</i>	<i>Flow-1</i>	<i>Flow-2</i>	<i>Flow-3</i>	<i>Flow-4</i>
<i>Size</i>	500 bytes	1000 bytes	500 bytes	500 bytes
<i>Period</i>	20 ms	10 ms	10 ms	10 ms
<i>Latency Constraint</i>	1 ms	100 ms	1ms	100 ms
<i>Type</i>	TSN	BE	TSN	BE

The MAC addresses and the transmission method (Unicast/Multicast) were also given. The topology considered was a diamond:


Execution steps
<p>The incoming flow characteristics are stored into a .txt file. Then, the java tool detects if a new flow has to be provisioned. If so, the ETH-TSN Resource Configurator adds the new flow ID to a local database, and generates the YAML configuration files. Once this process is done, the configuration files are sent to the switches using NETCONF standard</p>
Results
<p>A video demo of the full process is available at the Gitlab.</p>

Test 3: 3GPP Resource Configurator: Service Provisioning
Description
<p>The 3GPP Resource Configurator is requested, from the Service Automation, to compute a path in the 3GPP domain including KPI estimations.</p> <p>The 3GPP Resource Configurator will only include TSN switches in the 3GPP path when the QoS restrictions cannot be met without them.</p>
Execution steps
<p>Additionally, when the TSN switches are required, the 3GPP Resource Configurator will run the gopsav2 program:</p> <ul style="list-style-type: none"> • Device side TSN transator (DS-TT): <code>./gopsav2 -dst_ip_vxport 10.3.202.37 -gw_mac_vxport 08:33:ed:6a:a8:23 -vxport enp2s0 -vport enp6s0</code> • Network side TSN translator (NW-TT): <code>./gopsav2 -dst_ip_vxport 10.3.202.67 -gw_mac_vxport e0:47:35:0d:dd:93 -vxport enp59s0f1 -vport enp216s0f3</code>

Results

Path computation request from MD SA:

```
2025-03-05T12:20:36Z | server.go::func1():186 | INFO | Request received: POST /predict-6g-data-collection/v1/path/mdservice/pathrequest Body: {
  "id": "l102",
  "e2e_service_id": "l102",
  "service_status": "pending",
  "td_nodes": {
    "gateway": {
      "gateway_list": [
        {
          "id": "DS-TT"
        }
      ]
    },
    "endpoint": {
      "id": "NS-TT"
    }
  },
  "qos_characteristics": {
    "td_rtt": 20,
    "priority": 7,
    "td_delay": 10,
    "td_jitter": 6,
    "td_packet_loss": 0,
    "td_reliability": 100,
    "burst_arrival_time_window": {
      "t1": 0,
      "t2": 0
    },
    "burst_completion_time_window": {
      "t1": 0,
      "t2": 0
    }
  },
  "traffic_characteristics": {
    "period": 20,
    "direction": "bidirectional",
    "burst_size": 100,
    "periodicity": true,
    "maximum_flow_bitrate": 50
  },
  "service_lifetime": {
    "service_duration": {
      "t1": 0,
      "t2": 0
    },
    "recurrent_service_interval": {
      "t1": 0,
      "t2": 0,
      "recurrence_interval": 0
    }
  }
}
) From: 10.1.54.133:50378
```

Path computation response including KPIs:

```
2025-03-05T12:20:37Z | predict.go::sendSouthboundRequest():126 | INFO | Southbound request: {"id":"l102","e2e_service_id":"l102","service_status":"pending",
"td_nodes":{"gateway":{"gateway_list":[{"id":"DS-TT"}]},"endpoint":{"id":"NS-TT"}}, "td_path_selection":{"path_id":"Path3GPP-1","path_status":"computed","pred
icted_kpis":{"throughput":"99.60222","rtt":"20.23939","jitter":"6.00000"},"involved_nodes":{"node_list":[{"id":"UE_1","node_resources":{"resource_list":{}}},
{"id":"GNB_1","node_resources":{"resource_list":{}}}, {"id":"UPF_1","node_resources":{"resource_list":{}}}, {"id":"AppServer_Edge","node_resources":{"resource
list":{}}}]}},"qos_characteristics":{"td_rtt":20,"priority":7,"td_delay":10,"td_jitter":6,"td_packet_loss":0,"td_reliability":100,"burst_arrival_time_window"
":{"t1":0,"t2":0},"burst_completion_time_window":{"t1":0,"t2":0},"traffic_characteristics":{"period":20,"direction":"bidirectional","burst_size":100,"periodi
city":true,"maximum_flow_bitrate":50},"service_lifetime":{"service_duration":{"t1":0,"t2":0},"recurrent_service_interval":{"t1":0,"t2":0,"recurrence_interval
":0}}}
```

TSN switches configuration:

Device side TSN Translator (DS-TT) provision:



```
time="2025-02-19T11:09:16+01:00" level=info msg="Starting gopsav2 process"
time="2025-02-19T11:09:16+01:00" level=info msg="Port veth (0): enp6s0, Port vxlan (2): enp2s0"
time="2025-02-19T11:09:16+01:00" level=info msg="Creating tasks"
time="2025-02-19T11:09:16+01:00" level=info msg="using all in redirect program"
time="2025-02-19T11:09:16+01:00" level=info msg="using VXLAN redirect program"
time="2025-02-19T11:09:16+01:00" level=info msg="Launching go routines for queue 0"
time="2025-02-19T11:09:16+01:00" level=info msg="starting VxLAN Tx Task" routine=vxlanrx-task-port-2
time="2025-02-19T11:09:16+01:00" level=info msg="starting VxLAN Rx Task" routine=vxlanrx-task-port-2
time="2025-02-19T11:09:16+01:00" level=info msg="Waiting CTRL+C"
time="2025-02-19T11:09:16+01:00" level=info msg="starting Tx Task" routine=tx-task-port-0
time="2025-02-19T11:09:16+01:00" level=info msg="starting metrics thread"
```

Network side TSN Translator (NW-TT) provision:

```
time="2025-02-19T11:09:06+01:00" level=info msg="Starting gopsav2 process"
time="2025-02-19T11:09:06+01:00" level=info msg="Port veth (0): enp216s0f3, Port vxlan (2): enp59s0f1"
time="2025-02-19T11:09:06+01:00" level=info msg="Creating tasks"
time="2025-02-19T11:09:06+01:00" level=info msg="using all in redirect program"
time="2025-02-19T11:09:06+01:00" level=info msg="using VXLAN redirect program"
time="2025-02-19T11:09:06+01:00" level=info msg="Launching go routines for queue 0"
time="2025-02-19T11:09:06+01:00" level=info msg="starting VxLAN Rx Task" routine=vxlanrx-task-port-2
time="2025-02-19T11:09:06+01:00" level=info msg="starting VxLAN Tx Task" routine=vxlanrx-task-port-2
time="2025-02-19T11:09:06+01:00" level=info msg="Waiting CTRL+C"
time="2025-02-19T11:09:06+01:00" level=info msg="starting Tx Task" routine=tx-task-port-0
time="2025-02-19T11:09:06+01:00" level=info msg="starting metrics thread"
```

Data Collection

Test 1: MD Measurement Collection: Configuration of the monitoring for a Local MD

Description

This test aims to validate the Configuration interface of the Data Collection and Management platform, the use of the MDP OpenAPIs and the adaptation of data in both the Event Streaming Bus and the Time-Series DB.

The expected results of the test are:

1. To have a new MDP data source entry in the Monitoring Platform DB with the configuration of the data collector.
2. To have a 200 OK response from the MDP OpenAPIs with the expected JSON for every periodic query performed by the Data Collector.
3. To have the requested monitoring data collected and pushed in Kafka and *InfluxDB*.

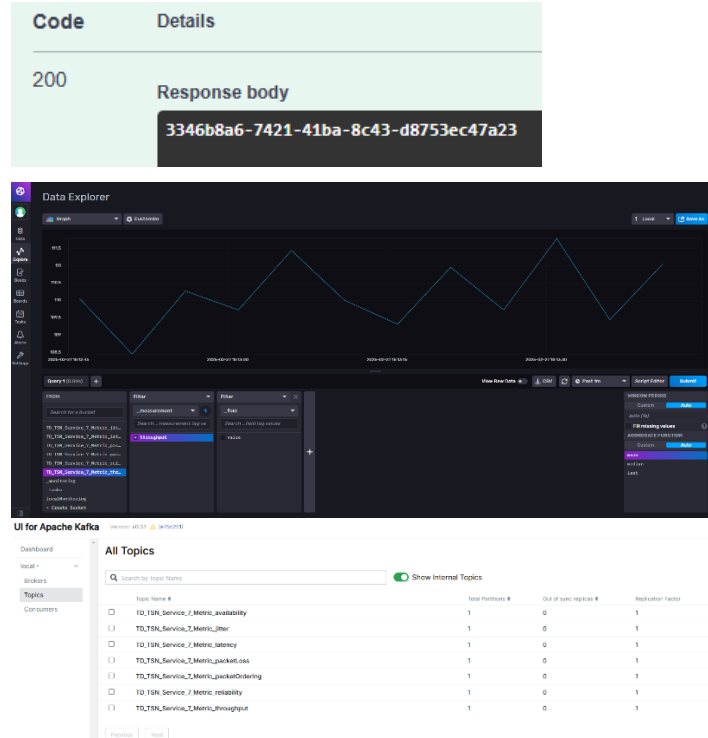
Execution steps

- Send a POST request to *Data_Collection_baseURL/datasources/{type}* with *type* equal to *MDP* and providing information about the metrics to be collected and the belonging service.

```
{
  "qos_characteristics": {
    "priority": 7,
    "td_reliability": 100,
    "td_packet_loss": 10,
    "td_delay": 20,
    "td_rtt": 30,
    "td_jitter": 10,
    "burst_arrival_time_window": {
      "t1": 0,
      "t2": 5
    },
    "burst_completion_time_window": {
      "t1": 0,
      "t2": 5
    }
  },
  "traffic_characteristics": {
    "direction": "directional",
    "periodicity": true,
    "period": 20,
    "burst_size": 500,
    "maximum_flow_bitrate": 5000
  },
  "service_id": "7"
}
```

- If the configuration of the data source is correct, the Measurement Collection creates a MDP data source entry in the Monitoring Platform DB, assigning a data source ID.
- The configured Data Collector (i.e. *Telegraf*) starts sending periodical queries to the mock-up MDP OpenAPI. If the responses are 200 OK, the collected data are adapted in a common format and pushed to Kafka and *InfluxDB*.

Results



Test 2: MD Measurement Collection: Removal of the monitoring for a Local MD

Description

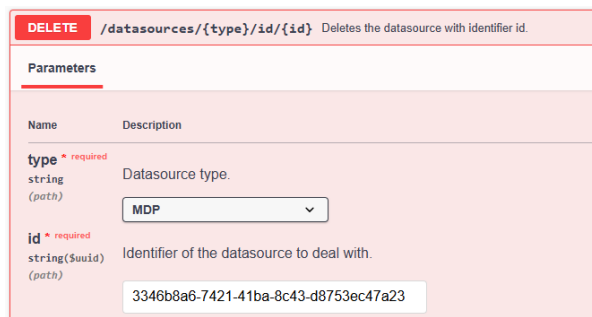
This test aims to validate the Configuration interface of the Data Collection and Management platform.

The expected results of the test are:

1. To no longer have the MDP data source entry in the Monitoring Platform DB with the configuration of the data collector.
2. To no longer have monitoring data collected and pushed in Kafka and *InfluxDB*.

Execution steps

- Preliminary step: to have a configured data source of type *MDP*, i.e. a configured data collector that pushes data to Kafka and *InfluxDB* and an entry in the Monitoring Platform DB with the information of the data source.
- Send a DELETE request to *Data_Collection_baseURL/datasources/{type}/id/{id}* with *type* equal to *MDP* and *id* equal to the UUID returned in the POST request used for creation.



DELETE /datasources/{type}/id/{id} Deletes the datasource with identifier id.

Parameters

Name	Description
type * required string (path)	Datasource type.
id * required string(\$uuid) (path)	Identifier of the datasource to deal with.

type: MDP

id: 3346b8a6-7421-41ba-8c43-d8753ec47a23

- The Data Collector (i.e. *Telegraf*) configuration is cleaned and no more data are pushed to Kafka and *InfluxDB*.
- The MDP data source entry is removed from Monitoring Platform DB as well.

Results

Code	Details
200	<p>Response body</p> <pre>3346b8a6-7421-41ba-8c43-d8753ec47a23</pre>

Test 3: MD Measurement Collection: Update of the monitoring for a Local MD

Description

This test aims to validate the Configuration interface of the Data Collection and Management platform, the use of the MDP OpenAPIs and the adaptation of data in both the Event Streaming Bus and the Time-Series DB.

The expected results of the test are:

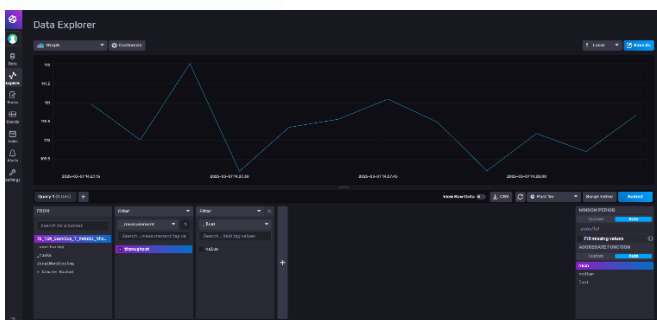
1. To have a MDP data source entry in the Monitoring Platform DB with the same data source ID assigned during the creation but with the configuration modified with the provided information.

2. To have a 200 OK response from the MDP OpenAPIs with the expected JSON for every periodic query performed by the Data Collector.
3. To have the requested monitoring data collected and pushed in Kafka and *InfluxDB*.

Execution steps

- Preliminary step: to have a configured data source of type *MDP*, i.e. a configured data collector that pushes data to Kafka and *InfluxDB* and an entry in the Monitoring Platform DB with the information of the data source.

```
{
  "traffic_characteristics": {
    "direction": "directional",
    "periodicity": true,
    "period": 20,
    "burst_size": 500,
    "maximum_flow_bitrate": 5000
  },
  "service_id": "7"
}
```



UI for Apache Kafka

Version: v0.11.0

Dashboard

Local

Brokers

Topics

Consumers

All Topics

Search by Topic Name

Show Internal Topics

Topic Name	Total Partitions	Out of sync replicas	Replication Factor	Number of messages	Size
TID_TSM_Service_X_Metric_throughput	1	0	1	20	39KB

- Send a PUT request to *Data_Collection_baseURL/datasources/{type}/id/{id}* with *type* equal to *MDP*, *id* equal to the UUID returned in the POST request used for creation and providing the new configuration of the data source in the request body, such as different metrics to be collected.

PUT /datasources/{type}/{id}/{id} Updates configuration of the datasource with identifier specified in the path.

Parameters

Name	Description
type ^{required}	Datasource type.
string (path)	
id ^{required}	Identifier of the datasource to deal with.
string (path)	
string (path)	54a7fca2-98e1-4400-b13f-54c3e3a8e2d1

Request body ^{required} application/json

New Configuration of the datasource:

```

{
  "non_characteristics": {
    "priority": 1,
    "reliability": 100,
    "latency": 10,
    "period": 20,
    "filter": 10,
    "directional_line_id": {
      "id": 0,
      "x1": 0,
      "x2": 5
    }
  },
  "traffic_characteristics": {
    "direction": "directional",
    "periodicity": true,
    "period": 20,
    "max_flow_rate": 5000
  }
}

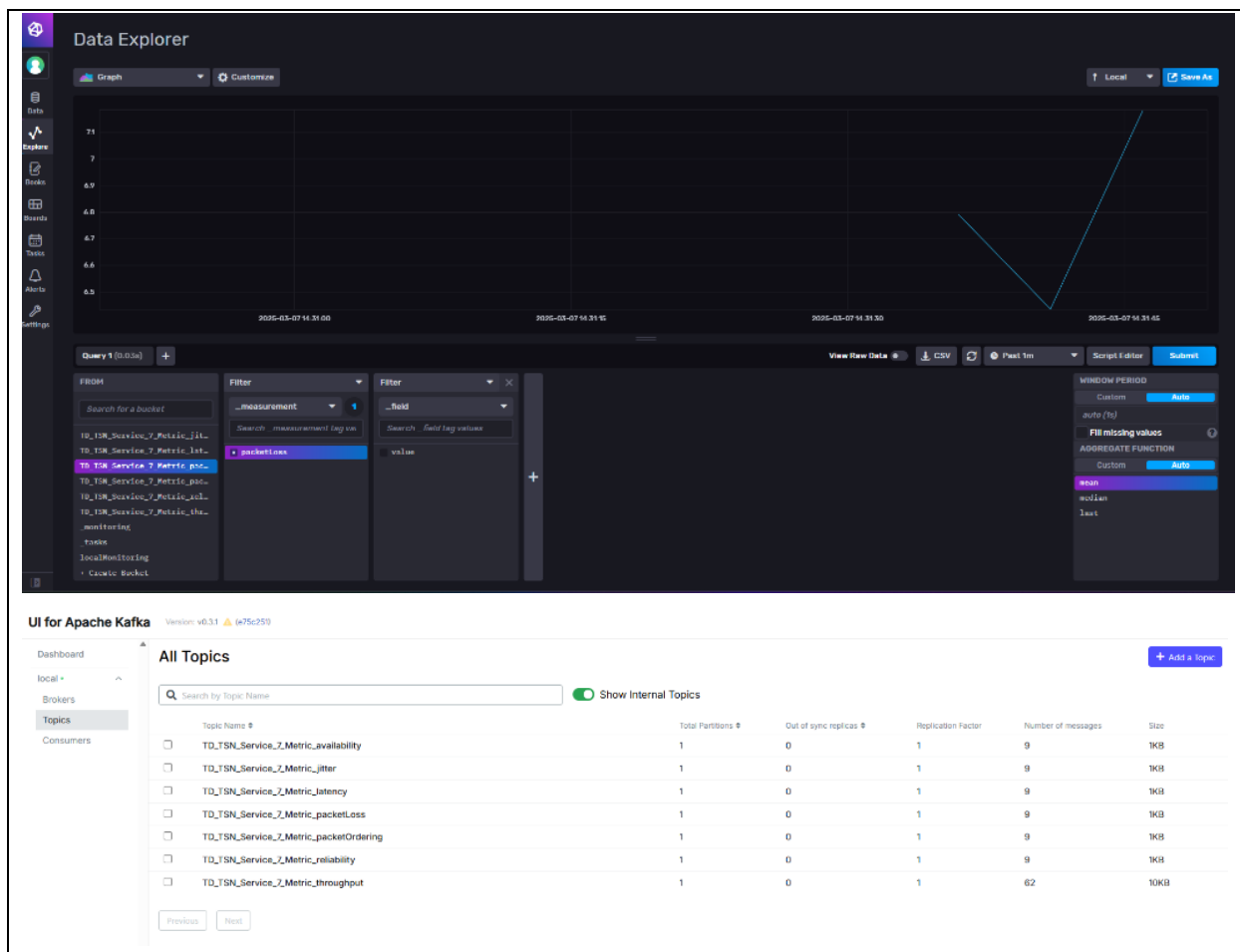
```

Execute

- If the new configuration of the data source is correct, the Measurement Collection updates the MDP data source entry in the Monitoring Platform DB.
- The configured Data Collector (i.e. *Telegraf*) keeps sending periodical queries to the mock-up MDP OpenAPI asking for the new metrics. If the responses are 200 OK, the collected data are adapted in a common format and pushed to Kafka and *InfluxDB*.

Results

In this example, since the *throughput* is requested both in the creation and in the update operations, there are more values for this metric with respect to the new requested metrics.



Test 4: MD Measurement Collection: Data consumer creation with certain permissions

Description

This test aims to validate the Configuration, the Near-RT and the Historic Data Retrieval interfaces of the Data Collection and Management platform.

The expected results of the test are:

1. To have a new user (i.e. data consumer) entry in the Monitoring Platform DB with information about the credentials and the role.
2. To have a new user permissions entry in the Monitoring Platform DB with information about the Kafka and *InfluxDB* authorizations.

3. To correctly consume only the authorized metrics from Kafka and *InfluxDB*.

Execution steps

- Preliminary step: to have a configured data source of type *MDP*, i.e. a configured data collector that pushes data to Kafka and *InfluxDB* and an entry in the Monitoring Platform DB with the information of the data source.
- Send a POST request to *Data_Collection_baseURL/users/add* providing user credentials in the request body.

```
{
  "username": "digital-twin",
  "password": "pr3dict6g",
  "user-role": "normal",
  "organization": "upc"
}
```

- If the user is correctly signed in both Kafka and *InfluxDB*, the Measurement Collection creates a user (i.e. data consumer) entry in the Monitoring Platform DB, assigning a user ID.
- Send a POST request to *Data_Collection_baseURL/permissions/{user_id}/enable* with *user_id* equal to the ID returned in the POST request used for user creation and providing the metrics for which to set the permissions in the request body.

POST `/permissions/{user_id}/enable` Configure permissions on a specific resource for the given user

Parameters

Name	Description
user_id * required string (path)	Username of the client

6u8WDErsRvzfCruJZrac65NUt60wlnT

Request body * required

Kafka/InfluxDB permissions to be configured for a specific user

Examples:

[Modified value]

```
{
  "service_id": "7",
  "metrics": ["throughput"]
}
```

- If the permissions are correctly set in both Kafka and *InfluxDB*, the Measurement Collection creates a user permissions entry in the Monitoring Platform DB, with the authorization information returned by Kafka and *InfluxDB*.
- Query *InfluxDB* by sending request to the *InfluxDB* API v2 with the user token and Kafka by using the script *kafka-utils/kafkaConsumer/kafka_consumer.py* following the instructions reported in the docs of the script.

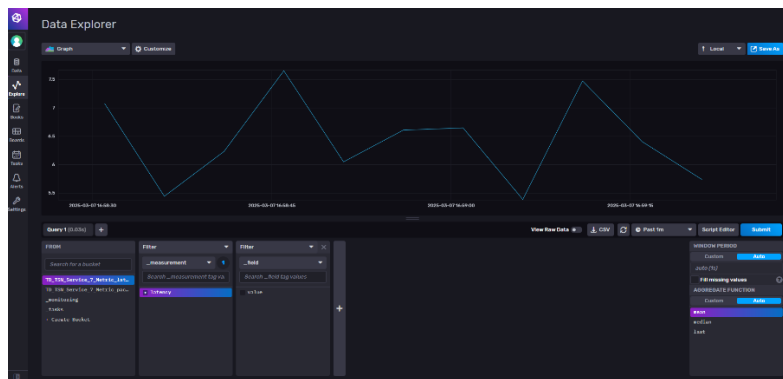
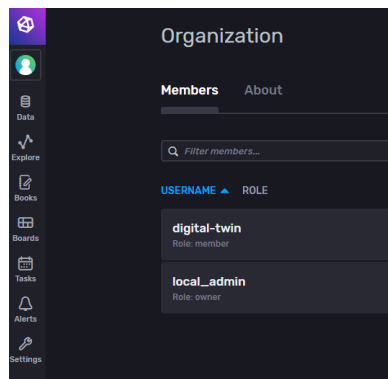
Results

Code	Details
201	Response body
	6u8MDEErsRvzfCruJJZrac65NUt60w1nT

User ID:

User Authorization ID:

Code	Details
201	Response body
	0e8952ace4723000



```
root@kafka_consumer:/usr/src/app# python3 kafka_consumer.py kafka 9092

*** LOGIN ***

Username: digital-twin
Password:

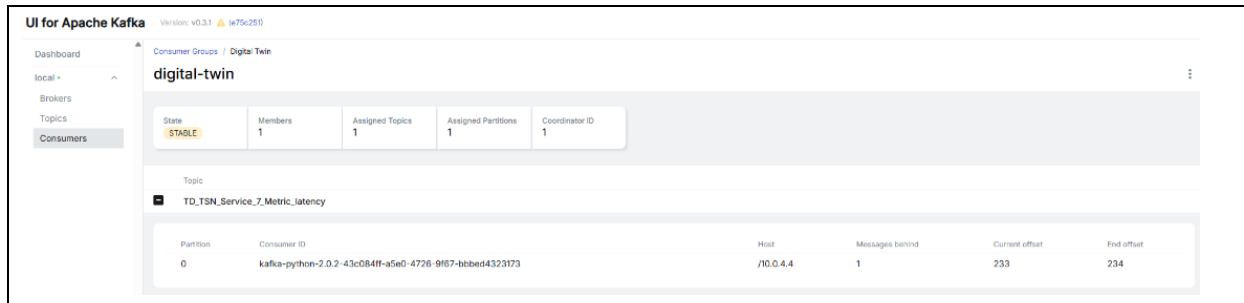
*** AUTHORIZED KAFKA TOPICS ***

TD_TSN_Service_7_Metric_latency
TD_TSN_Service_7_Metric_packetLoss

Enter the topic to subscribe to: TD_TSN_Service_7_Metric_latency

*** KAFKA MESSAGES ***

ConsumerRecord(topic='TD_TSN_Service_7_Metric_latency', partition=0, offset=0, timestamp=1741362488000, timestamp_type=0, key=None, value=b'latency,host=telegraf_mdp,pathLabel=Path_1,serviceLabel=7 value=5.8183883426774
67 17413624880000000000\n', headers=[], checksum=None, serialized_key_size=1, serialized_value_size=102, serialized_header_size=1)
ConsumerRecord(topic='TD_TSN_Service_7_Metric_latency', partition=0, offset=1, timestamp=1741362488500, timestamp_type=0, key=None, value=b'latency,host=telegraf_mdp,pathLabel=Path_1,serviceLabel=7 value=6.5382456270859
714 17413624885000000000\n', headers=[], checksum=None, serialized_key_size=1, serialized_value_size=103, serialized_header_size=1)
ConsumerRecord(topic='TD_TSN_Service_7_Metric_latency', partition=0, offset=2, timestamp=1741362489000, timestamp_type=0, key=None, value=b'latency,host=telegraf_mdp,pathLabel=Path_1,serviceLabel=7 value=6.9532165681831
64 17413624890000000000\n', headers=[], checksum=None, serialized_key_size=1, serialized_value_size=102, serialized_header_size=1)
ConsumerRecord(topic='TD_TSN_Service_7_Metric_latency', partition=0, offset=3, timestamp=1741362490000, timestamp_type=0, key=None, value=b'latency,host=telegraf_mdp,pathLabel=Path_1,serviceLabel=7 value=4.8421710819873
05 17413624900000000000\n', headers=[], checksum=None, serialized_key_size=1, serialized_value_size=102, serialized_header_size=1)
ConsumerRecord(topic='TD_TSN_Service_7_Metric_latency', partition=0, offset=4, timestamp=1741362490500, timestamp_type=0, key=None, value=b'latency,host=telegraf_mdp,pathLabel=Path_1,serviceLabel=7 value=4.597528573145
4 17413624905000000000\n', headers=[], checksum=None, serialized_key_size=1, serialized_value_size=101, serialized_header_size=1)
ConsumerRecord(topic='TD_TSN_Service_7_Metric_latency', partition=0, offset=5, timestamp=1741362500000, timestamp_type=0, key=None, value=b'latency,host=telegraf_mdp,pathLabel=Path_1,serviceLabel=7 value=4.7153395991141
825 17413625000000000000\n', headers=[], checksum=None, serialized_key_size=1, serialized_value_size=103, serialized_header_size=1)
ConsumerRecord(topic='TD_TSN_Service_7_Metric_latency', partition=0, offset=6, timestamp=1741362510000, timestamp_type=0, key=None, value=b'latency,host=telegraf_mdp,pathLabel=Path_1,serviceLabel=7 value=7.1848978634845
27 17413625100000000000\n', headers=[], checksum=None, serialized_key_size=1, serialized_value_size=102, serialized_header_size=1)
ConsumerRecord(topic='TD_TSN_Service_7_Metric_latency', partition=0, offset=7, timestamp=1741362515000, timestamp_type=0, key=None, value=b'latency,host=telegraf_mdp,pathLabel=Path_1,serviceLabel=7 value=6.4353644149382
29 17413625150000000000\n', headers=[], checksum=None, serialized_key_size=1, serialized_value_size=102, serialized_header_size=1)
ConsumerRecord(topic='TD_TSN_Service_7_Metric_latency', partition=0, offset=8, timestamp=1741362520000, timestamp_type=0, key=None, value=b'latency,host=telegraf_mdp,pathLabel=Path_1,serviceLabel=7 value=5.577920723706
68 17413625200000000000\n', headers=[], checksum=None, serialized_key_size=1, serialized_value_size=102, serialized_header_size=1)
ConsumerRecord(topic='TD_TSN_Service_7_Metric_latency', partition=0, offset=9, timestamp=1741362525000, timestamp_type=0, key=None, value=b'latency,host=telegraf_mdp,pathLabel=Path_1,serviceLabel=7 value=5.7967532208004
09 17413625250000000000\n', headers=[], checksum=None, serialized_key_size=1, serialized_value_size=102, serialized_header_size=1)
ConsumerRecord(topic='TD_TSN_Service_7_Metric_latency', partition=0, offset=10, timestamp=1741362530000, timestamp_type=0, key=None, value=b'latency,host=telegraf_mdp,pathLabel=Path_1,serviceLabel=7 value=6.879766956206
14 17413625300000000000\n', headers=[], checksum=None, serialized_key_size=1, serialized_value_size=101, serialized_header_size=1)
ConsumerRecord(topic='TD_TSN_Service_7_Metric_latency', partition=0, offset=11, timestamp=1741362535000, timestamp_type=0, key=None, value=b'latency,host=telegraf_mdp,pathLabel=Path_1,serviceLabel=7 value=6.110773967034
117 17413625350000000000\n', headers=[], checksum=None, serialized_key_size=1, serialized_value_size=102, serialized_header_size=1)
ConsumerRecord(topic='TD_TSN_Service_7_Metric_latency', partition=0, offset=12, timestamp=1741362540000, timestamp_type=0, key=None, value=b'latency,host=telegraf_mdp,pathLabel=Path_1,serviceLabel=7 value=6.186885985669
235 17413625400000000000\n', headers=[], checksum=None, serialized_key_size=1, serialized_value_size=102, serialized_header_size=1)
ConsumerRecord(topic='TD_TSN_Service_7_Metric_latency', partition=0, offset=13, timestamp=1741362545000, timestamp_type=0, key=None, value=b'latency,host=telegraf_mdp,pathLabel=Path_1,serviceLabel=7 value=7.246286692414
886 17413625450000000000\n', headers=[], checksum=None, serialized_key_size=1, serialized_value_size=102, serialized_header_size=1)
ConsumerRecord(topic='TD_TSN_Service_7_Metric_latency', partition=0, offset=14, timestamp=1741362550000, timestamp_type=0, key=None, value=b'latency,host=telegraf_mdp,pathLabel=Path_1,serviceLabel=7 value=5.681218163737
482 17413625500000000000\n', headers=[], checksum=None, serialized_key_size=1, serialized_value_size=102, serialized_header_size=1)
```



Partition	Consumer ID	Host	Messages behind	Current offset	End offset
0	kafka-python-2.0.2-43c084ff-a5e0-4726-9f67-bbbed4323f73	/10.0.4.4	1	233	234

Test 5: MD Measurement Collection: Data consumer deletion

Description

This test aims to validate the Configuration, the Near-RT and the Historic Data Retrieval interfaces of the Data Collection and Management platform.

The expected results of the test are:

1. To no longer have a user permissions entry in the Monitoring Platform DB.
2. To no longer have a user entry in the Monitoring Platform DB.

Execution steps

- Preliminary step: to have a configured user (i.e. a data consumer) with some permissions, i.e. a configured user in both Kafka and *InfluxDB*, that can consume data for which has permissions set.
- Send a DELETE request to *Data_Collection_baseURL/permissions/{user_id}/disable* with *user_id* equal to the ID returned in the POST request used for user creation

DELETE /permissions/{user_id}/disable Delete permissions on a specific resource for the given user

Name	Description
user_id * required string (path)	Username of the client
	6u8WDEErsvzfCruJZrac65NUt60wlnT

- Send a DELETE request to *Data_Collection_baseURL/users/{user_id}/remove* with *user_id* equal to the ID returned in the POST request used for user creation

DELETE
/users/{user_id}/remove
Remove authentication of the given user

Parameters

Name	Description
user_id * required string (path)	User ID <input type="text" value="6u8WDEErSRvzfCruJZrac65NUt60w1nT"/>

Results

Code

Details

201
Undocumented

Response body
0e8952ace4723000

User Authorization ID:

Code

Details

201
Undocumented

Response body
6u8WDEErSRvzfCruJZrac65NUt60w1nT

User ID:

Test 6: E2E Monitoring: Configuration of the monitoring for the E2E MD

Description

This test aims to validate the Configuration interface of the Data Collection and Management platform, the hierarchical architecture of Monitoring Platforms, the use of the MDP OpenAPIs and the adaptation of data in both the Event Streaming Bus and the Time-Series DB.

The expected results of the test are:

1. To have a new E2E data source entry in the Monitoring Platform DB with the configuration of the data collector.
2. To have a 200 OK response from the MDP OpenAPIs of each TD involved in the E2E path with the expected JSON for every periodic query performed by the MD Data Collectors.
3. To have the requested monitoring data collected and pushed in Kafka and *InfluxDB* at the E2E level.

Execution steps

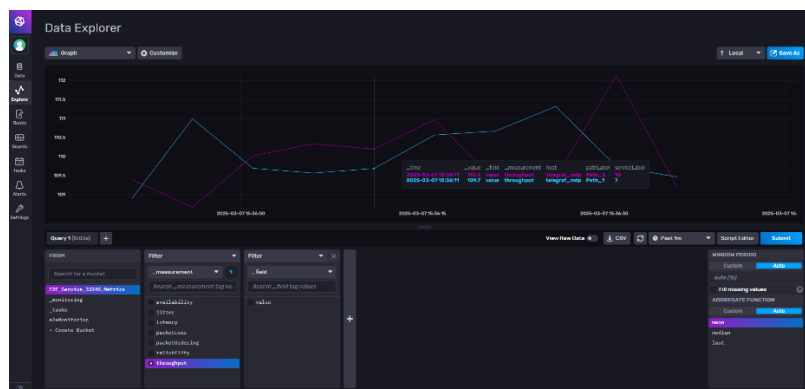
- Preliminary step: to have two configured data sources of type *MDP* and with different service ID, i.e. two configured data collectors that gather service data from mock-up MDP OpenAPI and push them to MD Kafka and *InfluxDB*.

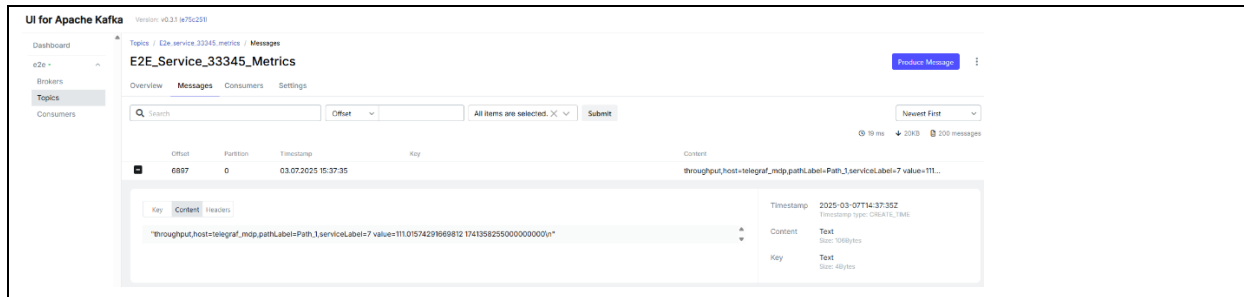
- Send a POST request to *Data_Collection_baseURL/datasources/{type}* with *type* equal to *E2E* and providing information about the metrics to be collected and the belonging service.

```
{
  "e2e_service_id": "33345",
  "mdservice_ids": ["7", "10"],
  "qos characteristics": {
    "priority": 7,
    "reliability": 100,
    "packet_loss": 10,
    "e2e_delay": 20,
    "e2e_rtt": 30,
    "jitter": 10,
    "burst arrival time window": {
      "t1": 0,
      "t2": 5
    },
    "burst completion time window": {
      "t1": 0,
      "t2": 5
    }
  },
  "traffic characteristics": {
    "direction": "directional",
    "periodicity": true,
    "period": 20,
    "burst_size": 500,
    "maximum flow bitrate": 5000
  }
}
```

- If the configuration of the data source is correct, the E2E Monitoring creates an E2E data source entry in the Monitoring Platform DB, assigning a data source ID.
- The configured Data Collector (i.e. *Telegraf*) starts sending periodical queries to Kafka of the MD Monitoring Platforms. If the hierarchy of MD MPs is properly configured, the collected data are merged and pushed to Kafka and *InfluxDB* at the E2E level.

Results





Test 7: E2E Monitoring: Removal of the monitoring for the E2E MD

Description

Execution steps

- Preliminary step: to have a configured data source of type *E2E*, i.e. a configured data collector that pushes to Kafka and *InfluxDB* at the E2E level and an entry in the Monitoring Platform DB with the information of the data source.
- Send a DELETE request to *Data_Collection_baseURL/datasources/{type}/{id}/{id}* with *type* equal to *E2E* and *id* equal to the UUID returned in the POST request used for creation.
- The Data Collector (i.e. *Telegraf*) configuration is cleaned and no more data are pushed to Kafka and *InfluxDB* at the E2E level.
- The E2E data source entry is removed from Monitoring Platform DB as well.

Results

-

Test 8: E2E Monitoring: Update of the monitoring for the E2E MD

Description

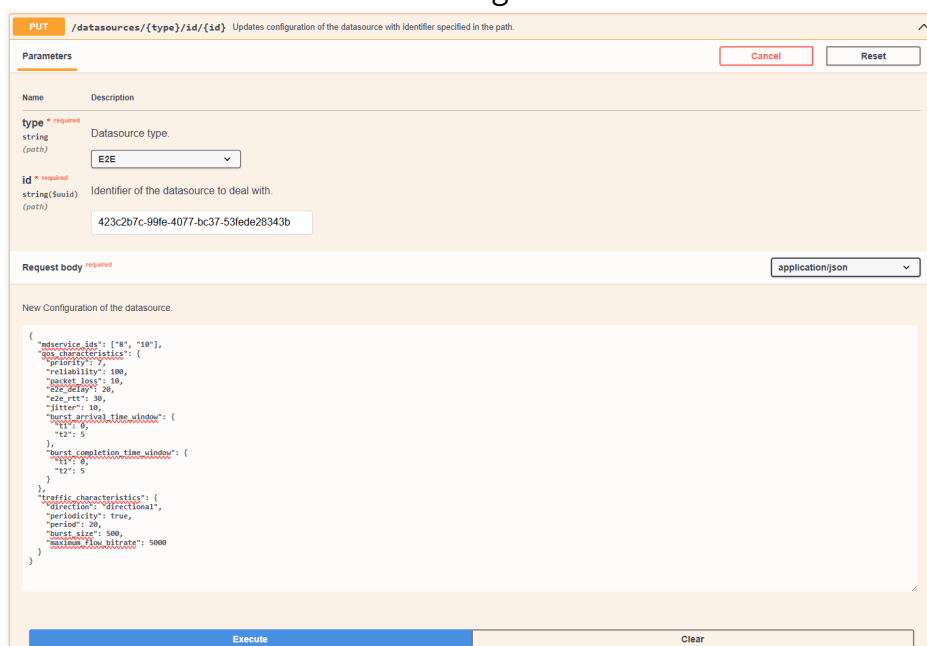
This test aims to validate the Configuration interface of the Data Collection and Management platform, the hierarchical architecture of Monitoring Platforms, the use of the MDP OpenAPIs and the adaptation of data in both the Event Streaming Bus and the Time-Series DB.

The expected results of the test are:

1. To have an E2E data source entry in the Monitoring Platform DB with the same data source ID assigned during the creation but with the configuration modified with the provided information.
2. To have a 200 OK response from the MDP OpenAPIs of each TD involved in the E2E path with the expected JSON for every periodic query performed by the MD Data Collectors.
3. To have the requested monitoring data collected and pushed in Kafka and *InfluxDB* at the E2E level.

Execution steps

- Preliminary step: to have a configured data source of type *E2E*, i.e. a configured data collector that pushes data to Kafka and *InfluxDB* at the E2E level and an entry in the Monitoring Platform DB with the information of the data source.
- Send a PUT request to *Data_Collection_baseURL/datasources/{type}/{id}/{id}* with *type* equal to *E2E*, *id* equal to the UUID returned in the POST request used for creation and providing the new configuration of the data source in the request body, such as different service IDs from which merge metrics.



PUT /datasources/{type}/{id}/{id} Updates configuration of the datasource with identifier specified in the path.

Parameters

Name	Description
type * required	Datasource type.
string (path)	E2E
id * required	Identifier of the datasource to deal with.
string(uuid) (path)	423c2b7c-99fe-4077-bc37-53fede28343b

Request body * required

application/json

New Configuration of the datasource.

```
{
  "service_id": ["8", "10"],
  "qos_characteristics": {
    "priority": 1,
    "reliability": 100,
    "packet_loss": 10,
    "e2e_delay": 20,
    "rtt": 30,
    "jitter": 10,
    "burst_arrival_time_window": {
      "t1": 0,
      "t2": 5
    },
    "burst_completion_time_window": {
      "t1": 0,
      "t2": 5
    }
  },
  "traffic_characteristics": {
    "direction": "directional",
    "periodicity": true,
    "period": 20,
    "burst_size": 500,
    "maximum_flow_rate": 5000
  }
}
```

Execute Clear

- If the new configuration of the data source is correct, the E2E Monitoring updates the E2E data source entry in the Monitoring Platform DB.
- The configured Data Collector (i.e. *Telegraf*) keeps sending periodical queries to Kafka of the MD Monitoring Platforms asking for the new service metrics. If the hierarchy of

MD MPs is properly configured, the collected data are merged and pushed to Kafka and *InfluxDB* at the E2E level.

Results

In this example, the metrics of service with ID “7” are no more collected, since the updated configuration requests metrics of services with ID “8” and “10”.

