

Rail to Digital automated up to autonomous train operation

D24.2 – Network management specification and realized selected proof-of-concepts for T24.2

Due date of deliverable: 30/04/2026

Actual submission date: 04/09/2026

Leader/Responsible of this Deliverable: Roelle, H.; SMO, Mattes, R.

Reviewed: Y

Document status		
Revision	Date	Description
01	17/10/2025	D24.2 document skeleton
02	20/01/2026	D24.2 working draft version 1
03	13/03/2026	D24.2 working draft version 2: Integration of review comments and missing sections and sub-sections
04	26/03/2026	D24.2 working draft version 3: Integration of review comments of Version 2
05	27/03/2026	D24.2 submission for TMT review
06	24/04/2026	D24.2 TMT comments incorporated and ready for SteCo review
07	04/09/2026	D24.2 improvements based on external JU feedback

Project funded from the European Union's Horizon Europe research and innovation programme		
Dissemination Level		
PU	Public	X
CO	Confidential, restricted under conditions set out in Model Grant Agreement	

Start date: 01/12/2022

Duration: 42 months

ACKNOWLEDGEMENTS



This project has received funding from the Europe's Rail Joint Undertaking (ERJU) under the Grant Agreement no. 101102001. The JU receives support from the European Union's Horizon Europe research and innovation programme and the Europe's Rail JU members other than the Union.

REPORT CONTRIBUTORS

Name	Company	Details of Contribution
Moritz, Matthias	DB	Elaboration, second opinion, review
De Paola, Enrico	DB	Elaboration, second opinion, review
Abbati, Lorenzo	FSI	Review
Durante, Raffaele	FSI	Review
Ferri, Giulio	FSI	Review
Tarmati, Benedetta	FSI	Review
Andrea, Bellitto	FT	Review
Dan, György	KTH	Review
Schuerch, Stefan	SBB	Elaboration, second opinion, review
Arndt, Igor	SMO	Elaboration, second opinion, review
Kirsche, Michael	SMO	Elaboration, second opinion, review
Knollmann, Volker	SMO	Elaboration, second opinion, review
Mattes, Rainer	SMO	Elaboration, second opinion, review
Rahn, Karsten	SMO	Elaboration, second opinion, review
Roelle, Harald	SMO	Elaboration, second opinion, review
Daniel, Christian	SNCF	Elaboration, second opinion, review
Roullier, Sylviane	SNCF	Elaboration, second opinion, review
Eriksson, Joachim	TRV	Review
Hallner, Pipsa	TRV	Review
Larsson, Henrik	TRV	Review

Disclaimer

Funded by the European Union. Views and opinion expressed are however those of the author(s) only and do not necessarily reflect those of the European union. Neither the European Union nor the granting authority can be held responsible for them. The project FP2-R2DATO is supported by the Europe's Rail Joint Undertaking and its members.

EXECUTIVE SUMMARY

The present document constitutes the Deliverable D24.2 “Network management specification and realized selected proof-of-concepts for T24.2”. It has been produced within the framework of the Innovation Pillar Flagship Project R2DATO under Work Package 24 (WP24), Tasks T24.2 and T24.3. This deliverable is part of a series contributing to the proposal for the upcoming TSI 202x, which will define the future common onboard communication network.

User stories, initially developed in R2DATO WP23, form the basis for the subsequent work in WP24. This approach ensures shared scoping and consistent specifications across communication functionality (WP23) and management functionality (WP24). These user stories serve as the foundational input, ensuring the inclusion of all necessary communication and management functionalities for their successful realization.

This deliverable contains the specification for the network management functionality of the Onboard Communication Network (OCN) (also known as the “One Common Bus”) and the documentation of the realized proof-of-concepts for network management. Drawing primarily from the solutions outlined in Chapter 4 of [1], this document refines and elaborates upon the methods for the selected user stories, providing a deeper level of technical specification.

To validate these approaches, this deliverable includes a detailed implementation description for a chosen number of user stories, serving as a proof of concept that confirms their practical feasibility within a specialized Laboratory Demonstrator. A key achievement of this work is the introduction of a dual-layered specification framework, which successfully separates human-centric process descriptions from machine-centric technical interface specifications. This ensures that the OCN management is not only technically robust but also operationally integrated into the lifecycles of the train manufacturer (OEM), Train Operating Company (TOC), and the maintainer.

The results presented here demonstrate that the OCN is the crucial communication backbone for next generation rolling stock. By leveraging standardized IT technologies—such as containerized microservices (Podman), secure network access control (IEEE 802.1X), and automated identity management—the project shows that a unified physical network can be logically partitioned to meet the rigorous safety and security demands of railway Control-Command and Signalling (CCS), in accordance with [2].

Furthermore, this deliverable establishes a technical and procedural blueprint that extends beyond signalling. The management functionalities defined herein are designed to be scalable, supporting the future convergence of TCMS (Train Control and Monitoring System) and OMTS (On-board Multimedia and Telematic Services) into a single, interoperable platform. Ultimately, D24.2 provides the European railway sector with a clear path towards a standardized, secure, evolvable and cost-effective network architecture, ensuring long-term technological leadership.

ABBREVIATIONS AND ACRONYMS

ALG	Application Layer Gateway
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
ARP	Address Resolution Protocol
ATO	Autonomous Train Operations
CA	Certificate Authority
CCS	Control Command and Signalling
CCTV	Closed-Circuit Television
CoA	Change of Authorization
COS	Customer Oriented Services
CPU	Central Processing Unit
CRL	Certificate Revocation List
DB	Database
DCP	Discovery and Configuration Protocol
DDOS	Distributed Denial of Service
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DOS	Denial of Service
EAP-TLS	Extensible Authentication Protocol - Transport Layer Security
ECN	Ethernet Consist Network
EMI	Electromagnetic Interference
ERJU	Europe's Rail Joint Undertaking
ETBN	Ethernet Train Backbone Node
ETCS	European Train Control System
EVC	European Vital Computer
FFFIS	Form Fit Function Interface Specification
FIS	Functional Interface Specification
FMEA	Failure Mode and Effects Analysis
FTA	Fault Tree Analysis
HIL	Hardware-in-the-Loop
HLD	High-Level Design
HSRP	Hot Standby Router Protocol
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure

HW	Hardware
ICMP	Internet Control Message Protocol
ID	Identification number/token
IDS/IPS	Intrusion Detection/Prevention System
IEEE	Institute of Electrical and Electronics Engineers
IPFIX	IP Flow Information Export
JSON	JavaScript Object Notation
JRU	Juridical Recording Unit
KPI	Key Performance Indicator
LAN	Local Area Network
LLDP	Link Layer Discovery Protocol
LLD	Low-Level Design
MAC	Medium Access Control
MSTP	Multiple Spanning Tree Protocol
NAC	Network Access Control
NetFlow	Network Flow
NMS	Network Management System
NTP	Network Time Protocol
NTS	Network Time Security
NVC	Non-Vital Computer
OB	Onboard
OCN	Onboard Communication Network
OEM	Original Equipment Manufacturer
OMTS	Onboard Multimedia and Telematic Subsystem
ONS	Onboard Network Switch
OSI	Open Systems Interconnection
OT	Operational Technology
PCP	Priority Code Point
PIS	Passenger Information System
PoC	Proof of Concept
QoS	Quality of Service
R2DATO	Rail to Digital automated up to Autonomous Train Operation
RAMS	Reliability, Availability, Maintainability and Safety
REST	REpresentational State Transfer
RoT	Root of Trust

RSTP	Rapid Spanning Tree Protocol
SCADA	Supervisory Control and Data Acquisition
SDTv2/SDTv4	Safe Data Transmission version 2/version 4
SIEM	Security Information and Event Management
SIL	Safety Integrity Level
SMS	Short Message Service
SNMP	Simple Network Management Protocol
SOC	Security Operations Center
SQL	Structured Query Language
TAS	Train Authentication Server
TCN	Train Communication Network
TIS	Train Inventory System
TRDP	Train Real-time Data Protocol
SSH	Secure Shell
STP	Spanning Tree Protocol
SW	Software
TCMS	Train Control and Management System
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TOC	Train Operating Company ¹
TRL	Technical Readiness Level
TSI	Technical Specification for Interoperability
UI	User Interface
UIC	Union Internationale des Chemins de fer (International Union of Railways)
URL	Uniform Resource Locator
UUID	Universally Unique Identifier
US	User Story
VLAN	Virtual Local Area Network
VRRP	Virtual Router Redundancy Protocol
WP	Work Package
YANG	Yet Another Next Generation

¹ The term “TOC” was chosen intentionally. It is vital to clarify, that, in commercial setups where responsibility for the track and responsibility for the trains is held by different companies, the interests of the train-running company are addressed.

TABLE OF CONTENTS

Acknowledgements.....	2
Report Contributors.....	2
Executive Summary	3
Abbreviations and Acronyms	4
Table of Contents.....	7
List of Figures	8
List of Tables	8
1 Introduction	9
2 Network Management Specification	10
2.1 Process Descriptions	13
2.1.1 Network Management Functionality: “Add new or differing end devices to the network”	13
2.1.2 Network Management Functionality: “Allow/forbid data exchanges”.....	17
2.1.3 Network Management Functionality: “Adding a new safe functionality to the network”	19
2.2 Interface Descriptions	24
2.2.1 Functional Interface Specifications	24
2.2.2 Technical Interface Specifications.....	35
3 Lab Demonstrator	44
3.1 Setup and Scope	44
3.1.1 Working with a Device:	44
3.1.2 Working with the Inventory.....	45
3.1.3 Working with the Network: Orchestrating Network-Wide Changes	45
3.2 Overall Architecture	46
3.2.1 Architectural Overview: Containerized Service Landscape	46
3.2.2 Software Component Descriptions.....	48
3.2.3 Operational Interface: Demonstrator Control Panel.....	49
3.3 Description of implemented proof-of-concepts.....	51
3.3.1 Process for demo case: “Working with a device”.....	51
3.3.2 Process for demo case: “Working with the Inventory”	60
3.3.3 Process for demo case: “Working with the Network: Orchestrating Network-Wide Changes”	64
4 Conclusions	65

4.1	Summary of Achievements	65
4.2	Fulfillment of Objectives.....	65
4.3	Reflections, Lessons Learnt, and Challenges	66
4.4	Final Synthesis	66
5	References.....	68

LIST OF FIGURES

Figure 1: System Architecture	10
Figure 2: Guide for Stakeholders	11
Figure 3: Roles and Responsibilities for “Add new or differing end device to the network”	17
Figure 4: Roles and Responsibilities for “Allow/Forbid Data Exchange”	18
Figure 5: Roles and Responsibilities for “Adding a New Safe Functionality to the Network”	23
Figure 6: Logical distribution of services	46
Figure 7: Components and the chosen software implementation	48
Figure 8: Demonstrator Control Panel.....	50
Figure 9: Demo case health status in FastApi UI.....	53
Figure 10: Demo case health status via getdeviceinfo button.....	54
Figure 11: Demo case UpdateDeviceConfig panel update mask before update	56
Figure 12: Demo case UpdateDeviceConfig panel update mask after update complete	58
Figure 13: Demo case UpdateDeviceConfig modal mask before update.....	59
Figure 14: Demo case UpdateDeviceConfig modal mask after update.....	59
Figure 15: Demonstrator Control Panel: Initial State (UIC-Device-002 Not in Inventory)	61
Figure 16: Demonstrator Control Panel: Device Disabled with Inventory Data Expanded	62
Figure 17: Demonstrator Control Panel: Device Added to Inventory (with AMQP Notification)	63

LIST OF TABLES

Table 1: User story overview	13
Table 2: Inventory response as a structured data object	43

1 INTRODUCTION

This deliverable outlines advancements in the definition of the Onboard Communication Network (OCN), a foundational element for future Control Command and Signalling (CCS) applications within train systems with a focus on defining the network management functionalities. As part of the R2DATO project, work packages WP23 and WP24 are dedicated to establishing a robust and standardized OCN that will serve as the primary communication backbone for all CCS applications, adopting a layered communication architecture approach. Unlike specific applications, the OCN's purpose is to provide a common technological framework adaptable to a wide array of CCS functionalities.

This document is a subsequent continuation in a series contributing to the proposed standardization of the future common onboard communication network, which is intended for incorporation into the future Technical Specification for Interoperability (TSI) 202x. The working group's efforts aim to ensure a unified and interoperable communication infrastructure across the railway sector.

The foundation of this work is built upon the user stories initially proposed in R2DATO D23.1 [3]. These user stories have been instrumental in guiding the tasks within WP24 (T24.1 and T24.2), ensuring a shared scope and consistent specifications for both communication functionality (WP23) and management functionality (WP24). By incorporating these essential inputs, we can confirm that all necessary communication and management functionalities are addressed for successful implementation.

Specifically, this deliverable focuses in detail on network management aspects.

Chapter 2 describes the network management specification based on the selected user stories generally categorized into the process descriptions and interface descriptions. The management functionalities based on interface descriptions are subsequently further elaborated specifically considering the functional interface specifications and the technical interface specifications.

The laboratory demonstrator with its setup and scope, as well as its overall architecture and the corresponding demo case are presented in chapter 3.

2 NETWORK MANAGEMENT SPECIFICATION

The reliability, security, and seamless integration of the Onboard Communication Network (OCN) are fundamental to the safety and efficiency of modern railway operations. This chapter provides a comprehensive framework for managing this complex ecosystem, specifically within the Control-Command and Signalling (CCS) environment. By focusing on both the human-centric processes and the machine-centric technical requirements, this section ensures that end devices—such as the ETCS-OB, ATO-OB and the interface to the rolling stock (TCMS) as well as the onboard recording device can be exchanged, configured, maintained and operated within a robust and well-governed structure through interaction between the stakeholders: Train Manufacturer, TOC and Maintainer.

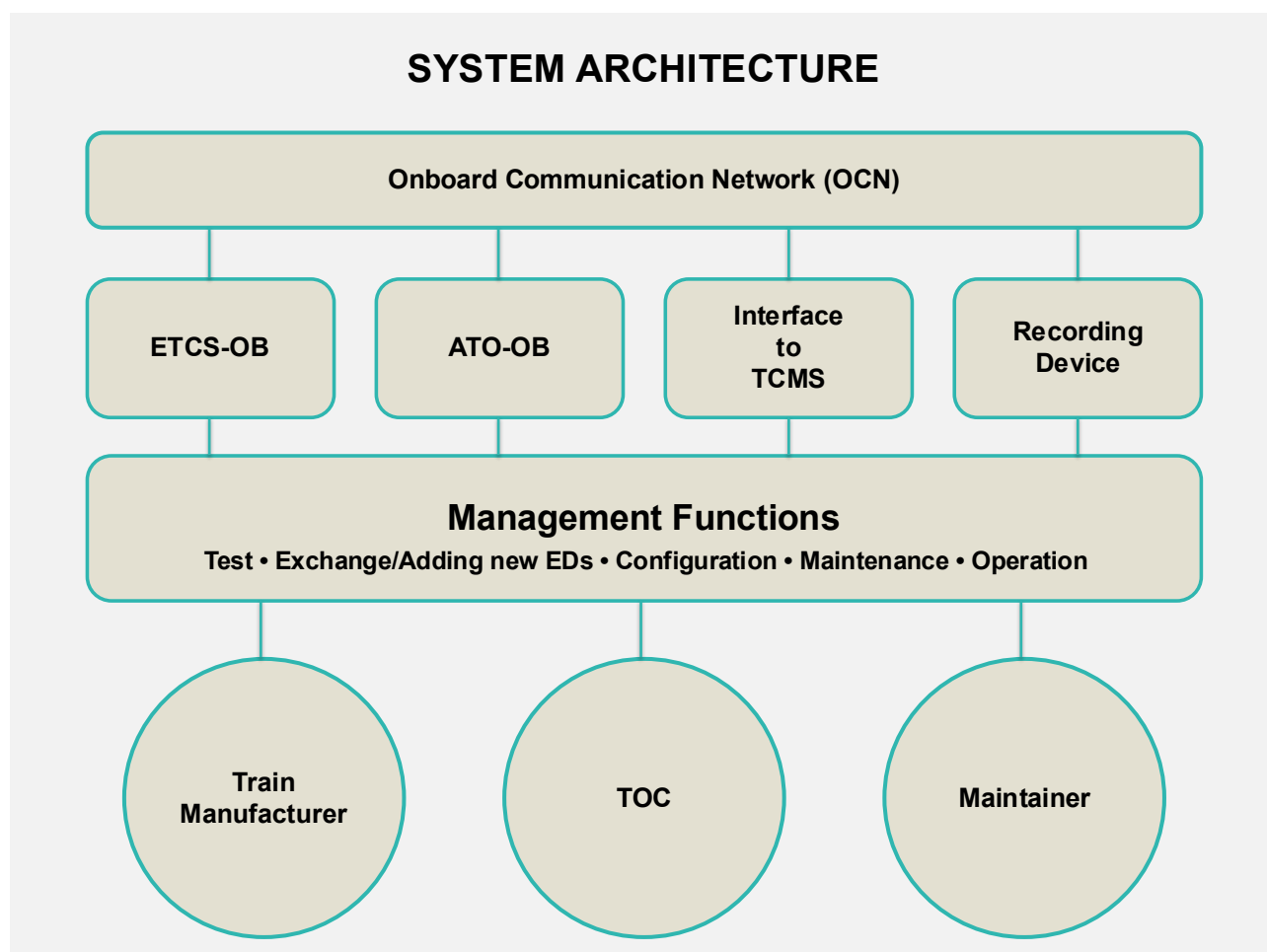


Figure 1: System Architecture

The chapter is divided into two primary areas:

- **Process Descriptions (Section 2.1):** This section outlines the systematic methodologies and lifecycle stages of the network. It defines the critical workflows for device integration and maintenance, while clearly delineating the roles and handovers between the train manufacturer, the train operating company (TOC), and/or the maintainer.

- **Technical Specifications and Interactions (Section 2.2):** Here, the focus shifts to the functional and technical interfaces that power the network and the corresponding end devices. This includes the:
 - **Functional Interface Specifications (Section 2.2.1),** which detail the high-level services, health monitoring, and event handling protocols that ensure all components communicate effectively and predictably.
 - **Technical Interface Specifications (Section 2.2.2):** This section provides detailed technical blueprints, including protocols, data formats, and interaction sequences. This subsection is specifically designed for engineers and developers to ensure the precise, secure, and interoperable integration of all components.

Together, these sections serve as a definitive guide for stakeholders to ensure the continuous functionality, regulatory compliance, and operational excellence of the onboard communication infrastructure and the operation of the respective end devices.

This relationship is illustrated in Figure 2:

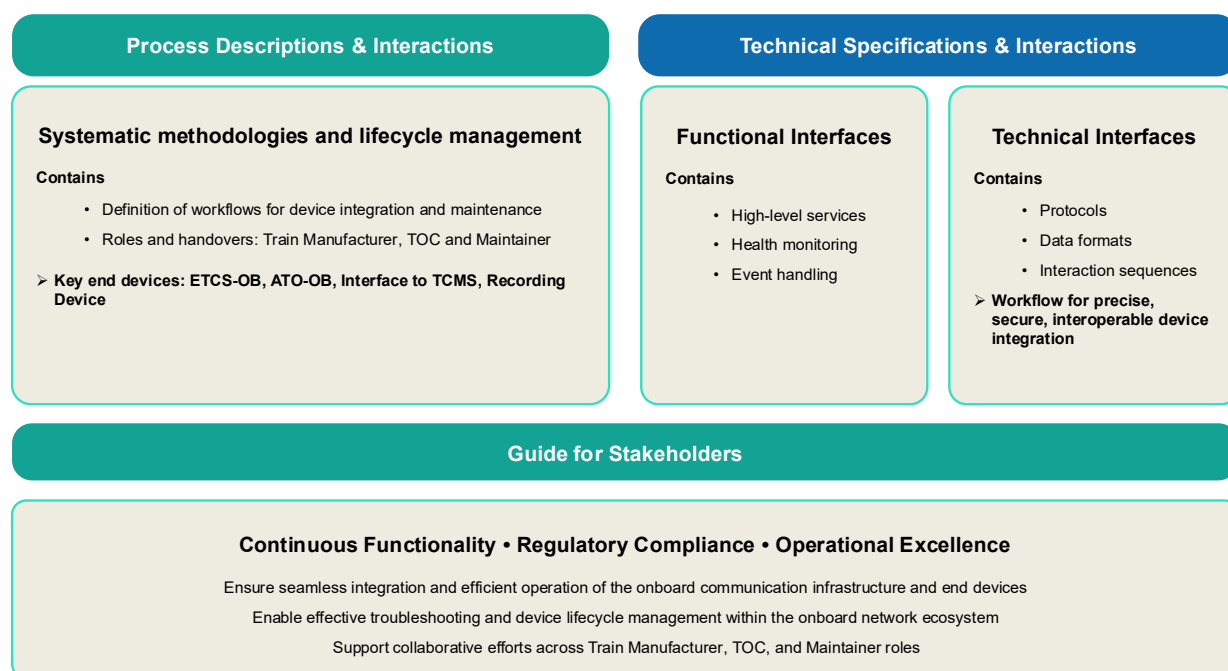


Figure 2: Guide for Stakeholders

The inherently given network functionalities are out of scope of this document due to the fact that the management functionalities listed in section 4.4 of [1] do not require new management specifications.

The user stories referenced throughout this document are presented in deliverable D24.1 [1] in full detail. To give the reader a quick reference, a user story overview is presented in Table 1. The selection of the presented user stories is based on structural equivalency of user stories on a technical level. Every user story and its typical cases are covered by the demonstrator presented in chapter 3.

ID	Actor	User Story
10002	TOC	As a TOC, I want to assess the health status of the network so that can decide whether corrective actions on network devices are necessary.
10004	Train manufacturer / TOC	As a Train Manufacturer/TOC, I want to be able to add new end devices in the network without heavy/complex configuration tasks and without assistance (esp. from the manufacturer) so that PoC/new services deployment are made easy.
10005	TOC	As a TOC, I want to be able to add new end devices on the network without breaking the homologation status so that PoC/new services deployment are made easy.
10010	TOC	As a TOC, I want to be able to allow/forbid data exchanges between different services/end devices so that there are no irrelevant communications on the network.
10017	TOC	As a TOC, I want to be informed when any unusual traffic occurs so that corrective actions can be taken.
10025	Train manufacturer / maintainer / TOC	As a train manufacturer/maintainer/TOC, I need to replace defective/obsolete network devices or end nodes without affecting the overall network functionality during the replacement process and without assistance from the manufacturer. This user story implies that the new device uses a different hardware or different hardware version compared to the original part.
10026	Train manufacturer / maintainer	As a train manufacturer or maintainer, I need to replace defective network devices or end nodes. These replacement parts must be able to communicate over the network immediately and without the need for significant configuration.
10029	Maintainer / TOC	As a maintainer or TOC, I want to receive alarms if failed IEEE 802.1X device authorization has been detected so that I can take appropriate security measures.
10030	Train manufacturer / maintainer	As a train manufacturer or maintainer, I want to be able to permanently remove an end device from the network. Note: this user story is different from the "failed device" story because a "failed device" should be reported in the health monitoring while a "removed" device does not.
10033	TOC / train manufacturer / maintainer	As a TOC or train manufacturer or maintainer, I need to be able to retrieve key performance indicators (e.g., bandwidth) of the network and its devices so that I can identify bottlenecks or other critical issues.
10038	Maintainer / TOC	As a maintainer, I want to have a procedure to add or update a safe functionality, ensuring that all affected devices are updated, and that the functionality is still safe after the update. That shall be proven and visible remotely.

ID	Actor	User Story
		User story remark: Please refer to deliverable D23.6 [2] for further information on how this user story is satisfied.
10039	Maintainer / TOC / 3rd party	As TOC, train manufacturer or 3rd party, I want to be able to integrate a new network-device or new end device solely based on independently acquired certificates. These certificates are based on lab tests and the vehicle specifications.

Table 1: User story overview

2.1 PROCESS DESCRIPTIONS

This section delves into the systematic approaches and methodologies required for managing various aspects of the onboard communication network, specifically within a CCS network environment utilizing end devices such as ETCS-OB, ATO-OB and the interface to the rolling stock (TCMS) as well as the onboard recording device.

These process descriptions outline the essential steps and considerations for maintaining a robust and reliable onboard network, highlighting the critical interactions and responsibilities among the key stakeholders: the train manufacturer, the Train Operating Company (TOC), and the Maintainer.

From integrating new devices (a process often initiated by the Train Manufacturer during vehicle design or by the Train Operating Company for fleet upgrades) to ensuring secure data exchanges and implementing critical safety functionalities, these processes detail how each entity contributes. The train manufacturer typically defines initial specifications and integration guidelines, while the train operating company is responsible for operational deployment and adherence to regulatory standards. The maintainer, in turn, performs the routine and corrective actions, ensuring the continuous functionality and safety of the network and its end devices.

Each subsection provides a detailed breakdown of the procedures, prerequisites, and verification steps necessary for successful execution, explicitly defining the handovers and collaborative efforts required between the train manufacturer's design and integration expertise, the train operating company's operational oversight, and the maintainer's field execution and troubleshooting capabilities.

2.1.1 Network Management Functionality: “Add new or differing end devices to the network”

To add new or differing end devices to the network (**REF US: 10005, 10025, 10026 and 10039**), it is foreseen that the train manufacturer will provide an abstract process description. An abstract process description focuses on the high-level steps, goals, and outcomes of a process, without a deep dive into the technical details. It is about understanding the procedure, flow and purpose and should contain the relevant documents necessary for the CCS end device integration on the network.

The technical specifications of the process description will be provided by the train manufacturer, in accordance with the abstract process description.

This specification will contain:

1. **Process:** Adding a new end device to an Ethernet network / replacing faulty/defective devices with different hardware or the same hardware but different software/firmware versions.

This process ensures that any new device can be securely and effectively integrated into the existing network infrastructure of the OCN, to successfully connect and configure a new end device or to replace faulty/defective devices within an Ethernet network, allowing it to communicate with other end devices.

2. Prerequisites:

The specification contains as prerequisites a description of all requirements for installing and operating the end device within the railway environment:

- Switch Port: Unique Ethernet port on the network switch explicitly designated for the integration of this end device.
- Ethernet Cable: A suitable Ethernet cable with the specified category (e.g. Cat5e, Cat6a, Cat7 or higher depending on the link speed) of appropriate length.
- Coding of the M12 plug: (e.g. D-coded for 100Mbit or X-Coded for Gigabit).
- Maximum expected power consumption and heat dissipation.
- Input-voltage and input-current range.
- Design and installation dimensions
- Network Configuration Details (if static IP is required): IP address, subnet mask, default gateway, and DNS server addresses.
- Administrator Access: Necessary permissions on the device and potentially on network equipment (e.g., switch, DHCP server) if manual configuration is needed.
- Physical Location: The device should be placed in its intended operational location in the train.

3. Step-by-Step Guide:

Prepare the end device and workspace:

Ensure all necessary accessories (power supply, adapters, etc.) are available. Place the device in its final desired location, ensuring it's near an available power outlet and network port.

Connect Power:

- Plug the device's power adapter into the device and then into a power outlet.
- Do not power on the device yet, unless specifically instructed by the device's setup guide.
- Connect to the Ethernet Network:

- Locate the Ethernet port on the new device (often labelled with a network icon or "LAN").
- Take one end of the Ethernet cable and firmly plug and screw it into the device's Ethernet port.
- Locate the available and designated Ethernet port of the network switch.
- Take the other end of the Ethernet cable and firmly plug and screw it into the network port.
- Power On and Initial Boot of the new end device:
 - o Power on the new end device.
 - o Self-check: A user might see a link light (usually green or amber) illuminate on both the device's Ethernet port and the switch port, indicating a physical connection.
 - o Allow the device to boot up completely. This may involve an initial setup wizard or operating system loading.
- Network Configuration (Automatic – DHCP, or static – e.g. DCP):
 - o Most modern devices are configured to obtain network settings automatically via DHCP (Dynamic Host Configuration Protocol). To avoid address conflicts when static and dynamic addressing should be used simultaneously on the same subnet, exclusively predefined IP addresses should be set via DHCP, for example by using option 82.
 - o Alternatively, a static IP address assignment provides complete control and predictability. This could be required and essential for end devices that need to be consistently reachable at a specific address, like servers, network infrastructure, or critical control systems. If devices support DCP this assignment could be done via DCP (Discovery and Configuration Protocol)
 - o Once the device boots up, it should automatically request an IP address, subnet mask, default gateway, and DNS servers from the network's DHCP server, or in the case of a static IP address assignment, besides the IP address, the subnet mask, default gateway, and DNS server addresses are directly set to the device's network interface e.g. via DCP or command line interface.

4. Verification:

- Check the device's network configuration to confirm it has received valid IP address information (dynamic or static). If the network policy requires static IP addresses for certain devices, it should be described that it needs to be configured manually as a prerequisite and it shall be ensured that there will be no IP-address conflict with other static or dynamically configured EDs. To make the necessary static changes to the IP address and network configuration, perform the following steps:

- Access the device's network settings (as described above or via its web interface for network appliances).
- Select the option to "Use the following IP address" or similar.
- Enter the pre-assigned IP address, subnet mask, default gateway, and DNS server addresses.
- Save the changes. (Caution: Ensure the static assigned IP address is unique and not already in use on the network to avoid conflicts! Also take care of correct subnet mask, default gateway and DNS.)
- Test Network Connectivity (e.g. via ping on a command prompt or terminal)

5. Additional Steps — Authentication and Authorization (if supported by ED and Switch):

If the end device and the switch support authentication and authorization via **IEEE 802.1X-2004** or **MAC-based port security**, the following additional steps shall be performed:

- **IEEE 802.1X-2004 Authentication (Port-Based Network Access Control):**
 - Verify that both the end device (supplicant) and the network switch (authenticator) support IEEE 802.1X-2004.
 - Ensure that an Authentication Server (e.g., a RADIUS server) is available and properly configured within the network infrastructure.
 - Configure the switch port to operate in 802.1X mode, enabling port-based access control for the designated port.
 - Configure the end device with the appropriate 802.1X credentials (e.g., EAP method, certificates, or username/password) as required by the network policy and in accordance with SS-147.
 - Upon powering on the end device, the authentication exchange between the supplicant (end device), authenticator (switch), and authentication server shall take place automatically.
 - Verify that the port transitions to an authorized state, allowing network traffic to flow. If authentication fails, the port shall remain in an unauthorized state and access shall be denied.
 - Document the authentication method and credentials used, in accordance with the applicable security policies.
- **MAC-Based Port Security (according to SS-147):**
 - If IEEE 802.1X-2004 is not supported by the end device, MAC-based port security may be applied as an alternative access control mechanism, in accordance with SS-147.
 - Retrieve and register the MAC address of the new end device prior to connection.
 - Configure the network switch port to allow only the specific MAC address(es) of the authorized end device(s).

- Define the violation policy on the switch port (e.g., restrict, shutdown, or protect) in case an unauthorized MAC address attempts to access the port.
- Connect the end device and verify that the switch port recognizes and permits the device's MAC address, allowing network access.
- Confirm that any unauthorized device attempting to connect to the same port is correctly blocked according to the configured violation policy.
- Record the registered MAC address and port security configuration in the network documentation, in compliance with SS-147 requirements.

• **Final Verification:**

- After successful authentication or MAC-based authorization, repeat the network connectivity test (e.g., ping) to confirm that the end device is fully operational and communicating within the network.

Ensure all authentication events and port security configurations are logged and traceable for audit purposes.

The roles and responsibilities of the different steps for the management functionality: “Add new or differing end device to the network” is shown in Figure 3:

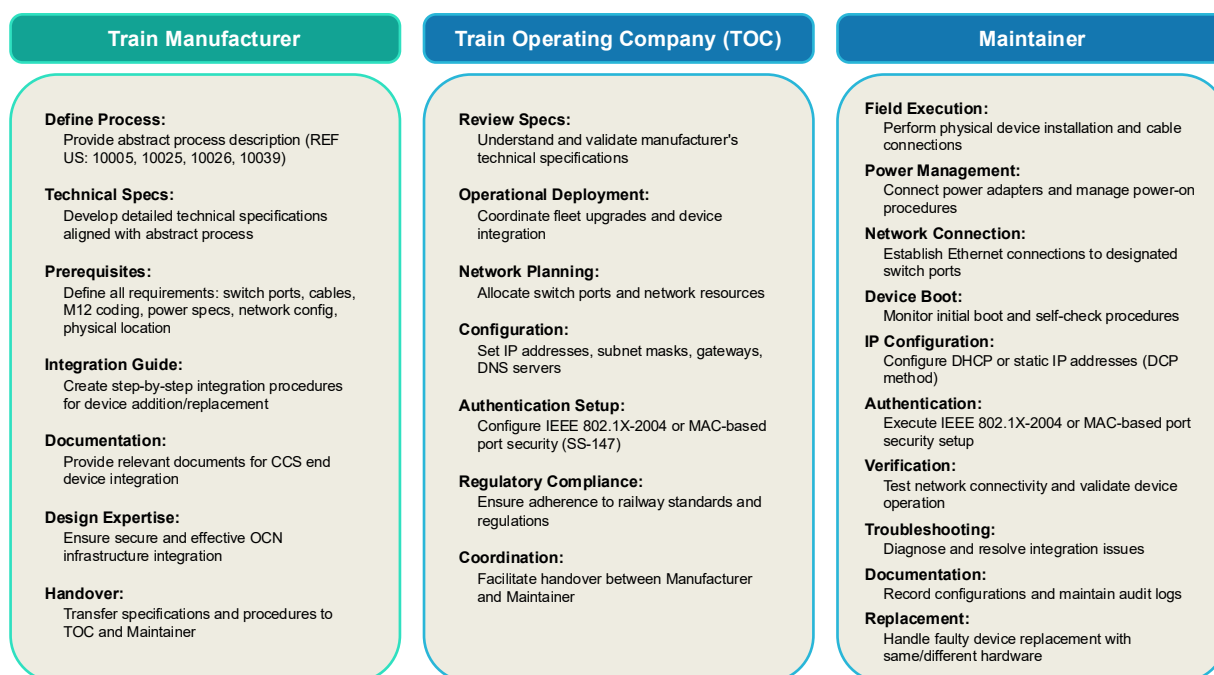


Figure 3: Roles and Responsibilities for “Add new or differing end device to the network”

2.1.2 Network Management Functionality: “Allow/forbid data exchanges”

The management functionality: “Allow / forbid data exchanges” (REF US: 10010) is also based on a process description and on the idea that the TOC can control the network traffic between different services/end devices so that there are no irrelevant and unintended communications on the network outside the specified zones or in general. Besides that, this management functionality implemented using VLANs, allows one physical network to separate in logical zones/domains. In a network, a

logical zone is a group of devices that share common characteristics, access requirements, or security policies (e.g. CCS, TCMS, OMTS, COS). A logical separation by using VLANs for this purpose offers further advantages:

1. **Enhanced Security:** If one zone is compromised, the others can remain isolated and protected.
2. **Improved Performance:** By reducing the size of broadcast domains (where all devices "hear" each other's network chatter), network traffic can be localized under some conditions, (e.g. VLANs spanning exclusively over a vehicle) leading to better performance.
3. **Simplified Management:** It's easier to apply specific security policies, quality of service (QoS) settings, or troubleshooting steps to a smaller, well-defined zone.
4. **Flexibility and Scalability:** TOC/maintainer can add or move devices to different zones without physically rewiring the network.

The management functionality is based on the configuration of the active network components (e.g. L2-Switch) and provided by the network itself. Therefore, the train manufacturer must specify and provide a set of VLAN-IDs for the different zones, and the end device manufacturer must preconfigure its end device with this dedicated VLAN-ID and PCP or to allow to configure the VLAN-ID and PCP via configuration interface for the respective zone. If the corresponding switch-port where the end device is connected to, is configured as a member of this aligned VLAN, then exclusively the traffic this VLAN is being forwarded. With these specifications, this user story can be fully mapped.

The roles and responsibilities of the different steps for the management functionality: "Allow/Forbid Data Exchange" is shown in Figure 4:

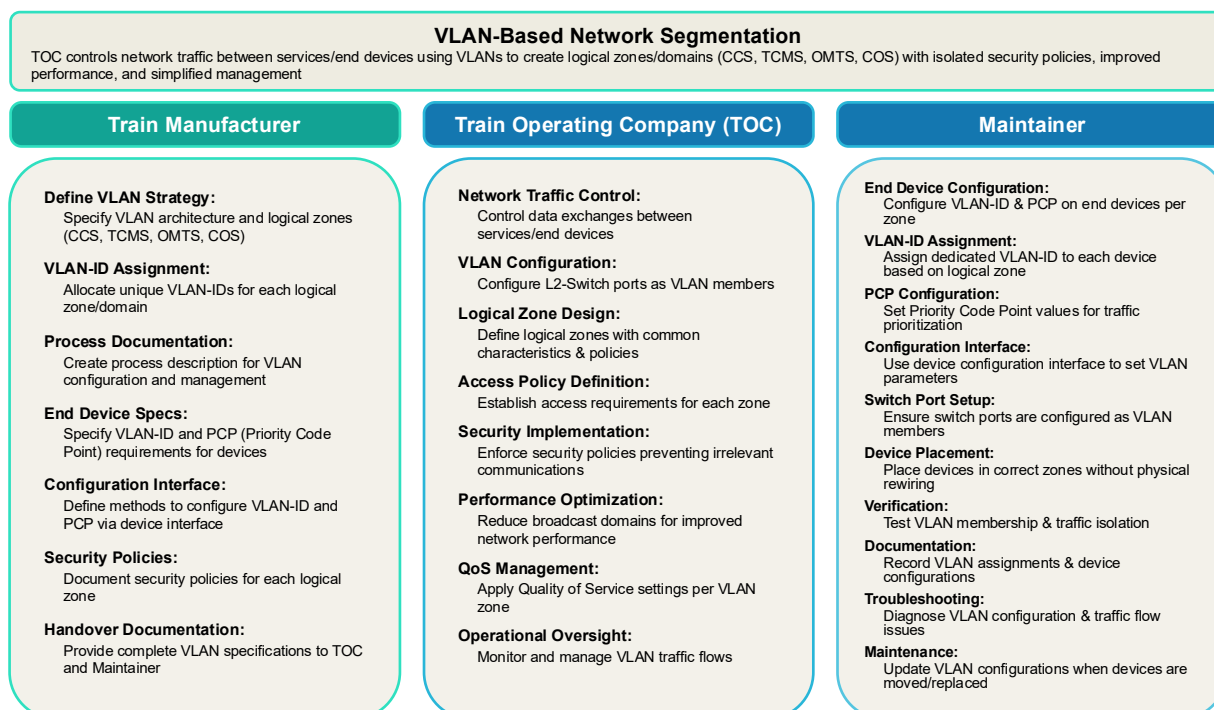


Figure 4: Roles and Responsibilities for "Allow/Forbid Data Exchange"

2.1.3 Network Management Functionality: “Adding a new safe functionality to the network”

The management functionality: “Adding a new safe functionality to the network” (REF US: 10038) is also based on a process description. This process description emphasizes safety, regulatory compliance, and the unique operational environment of a train. "Safe functionality" here refers to features that enhance mainly safety integrity of a function or a service or directly contribute to the safety of the train and its passengers.

The train manufacturer will provide a process description containing the initial requirements and the steps needed to add or update a safe functionality. The steps and processes can be performed by TOC and/or maintainer depending on the functionality and granularity (e.g. pure software update could be initiated by TOC or maintainer). These steps are described in the following example, which can serve as a template adapted to the specific safety function:

1. Step 1: Define and understand the new functionality

This initial phase is significant for safety-critical systems.

• 1.1. Requirement Gathering:

- **What specific operational, safety, or security enhancement does this functionality provide?** (e.g., "Enhanced real-time feature for braking system," "Improved automatic obstacle detection" "New passenger information system with emergency communication features").
- **What are the precise functional and non-functional requirements** (e.g., latency, throughput, availability, fault tolerance, data integrity, response times)?
- **Crucially, what are the safety requirements?** This includes defining the Safety Integrity Level (SIL) for any safety-critical functions, redundancy needs, and failure modes.

• 1.2. Scope Definition & Impact Analysis:

- Which train systems or subsystems will be affected? (e.g., Train Control and Management System (TCMS), Passenger Information System (PIS), CCTV, Passenger Wi-Fi, Ethernet Train Backbone Network (ETBN), specific sensors).
- **Conduct a formal safety impact assessment:** How will this new functionality interact with and potentially affect existing safety-critical systems (e.g., braking, traction, door control, signalling interfaces)?

• 1.3. Regulatory & Standards Compliance:

- Identify *all* applicable railway standards (e.g., **EN 50155** for electronic equipment, **EN 50129** for the life cycle management of a safety-related electronic system, **EN 50716** for software, **EN 50126** for RAMS (Reliability, Availability, Maintainability, Safety) and **IEC 61375** for Train Communication Network/Ethernet Train Backbone Network (TCN/ETBN)).
- What specific certifications will be required for the new functionality or its components by national or European railway authorities?

- **1.4. Cybersecurity Assessment:**

- How does this functionality impact the train's overall cybersecurity posture? Does it introduce new vulnerabilities or attack vectors? How will it be protected? (Referencing standards like e.g. **EN 50126**, **EN 50716**, **EN 50129** or **TS 50701**).

2. Step 2: Design and Planning (The "How" – with Railway Engineering Rigor)

This phase translates requirements into a detailed, robust, and certifiable design.

- **2.1. Architectural Design:**

- **High-Level Design (HLD):** Detail the system architecture, interfaces, data flows, and integration points with the existing TCN (ECN + ETB). Emphasize redundancy, fail-safe mechanisms, and recovery.
- **Low-Level Design (LLD):** Specify hardware (must be **EN 50155** and **EN 50129 compliant**), software architecture, communication protocols (e.g., TRDP + SDTv2/SDTv4), IP addressing schema, network topology changes, and Quality of Service (QoS) configurations for critical traffic.

- **2.2. Environmental & Physical Considerations:**

- Detail how the new components will withstand train-specific environmental conditions (vibration, shock, temperature extremes, humidity, electromagnetic interference (EMI)).
- Develop a precise physical installation plan: mounting, cabling specifications (e.g., fire-retardant, robust for railway use), power supply integration, and cooling solutions.

- **2.3. Safety & Reliability Analysis:**

- Perform comprehensive analyses such as FMEA (Failure Mode and Effects Analysis), FTA (Fault Tree Analysis), and RAMS (Reliability, Availability, Maintainability, Safety) analysis.
- Define and document the Safety Integrity Levels (SIL) for all relevant software and hardware components.

- **2.4. Integration Strategy:**

- Plan for seamless and non-disruptive integration with existing train management systems (CCS/TCMS/OMTS) and other subsystems.
- Define precise communication interfaces, data exchange formats, and API specifications.

- **2.5. Rollback Plan:**

- Develop a meticulously detailed and *tested* plan to revert to the previous operational state if any issue arises during implementation or testing. This includes procedures for restoring previous configurations, physically removing hardware, and ensuring the train can safely return to service without the new functionality.

3. Step 3: Preparation and Approval:

This phase focuses on documentation, formal reviews, and securing all necessary approvals.

- **3.1. Extensive Documentation & Review:**

- Generate comprehensive design documents, detailed test plans, a robust safety case, installation guides, and operational manuals.
- Conduct formal reviews by internal safety experts, engineering teams, and *crucially*, external certification bodies.
- **3.2. Change Management & Certification Approval:**
 - Submit formal change requests through the railway operator's and train manufacturers internal change management processes.
 - Obtain all necessary regulatory approvals and certifications from national or European railway authorities before any physical implementation on an operational train. This is a fundamental step for railway systems.
- **3.3. Procurement & Staging:**
 - Source all hardware, ensuring it is fully **EN 50155, EN 50121 and EN 45545 compliant** and certified.
 - Pre-configure and extensively test all components in a lab environment that *simulates* train conditions (e.g., vibration tables, temperature chambers).
- **3.4. Configuration Backups:**
 - Perform full and verified backups of all affected train system configurations *before* any changes are made.

4. Step 4: Implementation (Controlled & Monitored)

Execution must be precise and conducted under strict control.

- **4.1. Scheduled Maintenance Window:**
 - Implementation *must* occur during a pre-approved, planned, and extended maintenance window when the train is out of service.
- **4.2. Qualified Personnel:**
 - Only certified, highly trained, and authorized technicians with specific railway experience should perform the installation and configuration.
- **4.3. Step-by-Step Execution:**
 - Follow the detailed installation and configuration plan precisely.
 - Document every step, including timestamps, personnel involved, and observations.
- **4.4. Physical Installation:**
 - Install hardware according to the design specifications, ensuring proper mounting, robust cabling, and secure power connections, adhering to all railway installation best practices.

5. Step 5: Testing and Validation

This is arguably the most critical phase for a train network, involving multiple layers of testing.

- **5.1. Lab Testing:**

- Thorough functional, performance, and stress testing in a lab environment, ideally utilizing Hardware-in-the-Loop (HIL) simulation to mimic real-world train behaviour.
- Additional environmental testing (vibration, shock, temperature cycling) of the actual hardware to proof the EN 50155 conformance. (Note: These Tests are optional and depending of the availability of specific test-equipment)
- **5.2. Static On-Train Testing:**
 - After physical installation, perform comprehensive tests with the train stationary in the depot.
 - Verify all interfaces, communication paths, and basic functionality.
 - **Safety function validation:** Rigorously test any safety-related aspects, including failure injection tests in accordance with EN50129
- **5.3. Dynamic On-Train Testing (Trial Runs):**
 - Conduct extensive trial runs under various operational scenarios (different speeds, routes, environmental conditions, passenger loads).
 - Continuously monitor performance, stability, and interaction with *all* other train systems.
 - Collect extensive logging and diagnostic data for post-analysis.
 - **Emergency scenario testing:** Validate behaviour under fault conditions, emergency stops, and degraded modes.
- **5.4. Cybersecurity Testing:**
 - Perform penetration testing and vulnerability assessments specifically tailored to the onboard network and its interfaces.
- **5.5. Stakeholder Verification:**
 - Train operators, maintenance personnel, safety officers, and relevant regulatory bodies must formally verify and approve the functionality.

6. Step 6: Monitoring and Optimization (Continuous Safety Assurance)

Once deployed, continuous vigilance is key.

- **6.1. Establish Robust Monitoring:**
 - Integrate the new functionality into the train's existing diagnostic, remote monitoring, and fault reporting systems.
 - Set up granular alerts for performance deviations, faults, and security incidents.
 - Utilize specialized railway diagnostic tools and data analytics.
- **6.2. Post-Deployment Review:**
 - After an initial operational period (e.g., several weeks or months), conduct a formal review of performance, reliability, and any incidents or anomalies.
- **6.3. Ongoing Maintenance & Updates:**

- Plan for regular software updates, security patches, and hardware maintenance, meticulously aligning these with scheduled train maintenance cycles.

7. Step 7: Documentation and Training (Essential for Long-Term Safety and Operations)

Knowledge transfer and comprehensive documentation are vital for the long-term safety and maintainability of the train.

• 7.1. Update Comprehensive Documentation:

- Update all relevant train documentation: detailed wiring diagrams, system schematics, software versions, IP plans, maintenance manuals, and the formal safety case.
- Ensure all documentation is version-controlled, easily accessible, and compliant with railway standards.

• 7.2. Create Detailed Operational & Maintenance Procedures:

- Develop clear, step-by-step procedures for operators and maintenance staff, including comprehensive troubleshooting guides specific to the new functionality.

• 7.3. Certified Training:

- Provide mandatory, certified training for all personnel involved in operating, maintaining, or troubleshooting the new functionality. This includes train drivers, maintenance technicians, control center staff, and emergency responders.

The roles and responsibilities of the different steps for the management functionality: “Adding a New Safe Functionality to the Network” is shown in Figure 5:

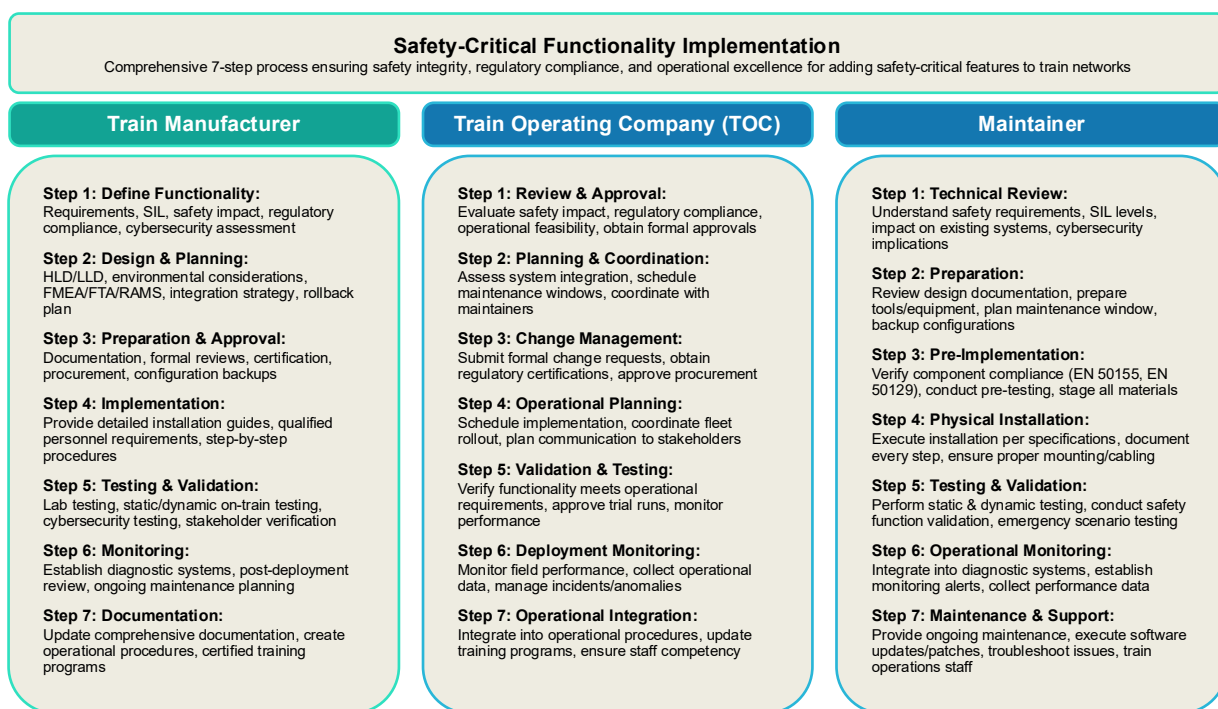


Figure 5: Roles and Responsibilities for “Adding a New Safe Functionality to the Network”

2.2 INTERFACE DESCRIPTIONS

This chapter focuses on the technical specifications and interactions that define how different components of the onboard communication network communicate and operate. It details the functional and technical interfaces that enable essential management functionalities, such as monitoring device health, handling event notifications, polling key performance indicators, and managing device lifecycles. Understanding these interfaces is crucial for ensuring seamless integration, efficient operation, and effective troubleshooting within the complex railway network ecosystem.

2.2.1 Functional Interface Specifications

This subsection outlines the high-level capabilities and services that govern the interactions between various network elements. It describes the "what" of the communication, focusing on the intended purpose and expected outcomes of each interface. By detailing how different functionalities, such as health status reporting and event notifications, are achieved, this section provides a clear understanding of the operational requirements and expected behaviours of the onboard communication network and the connected end devices.

2.2.1.1 Management functionality to get network device health status

In order to be able to form an intersection and thus determine a basic health status that indicates OK/NOK as a Boolean value, which was requested as a minimal viable example of the user stories (**Ref US: 10002, 10025 and 10026**) of D24.1 [1] the methods/protocols used to obtain the desired information for end devices should be implemented via "HTTP-REST" polling. For network devices and also for end-devices the methods and protocols are largely manufacturer specific. This is the reason why the description of the interface regarding the health status for network devices is on the level of a FIS.

To get an overview of a network device (e.g. an Ethernet switch) health status, there are different categories introduced below, depending on the information and the granularity which could be requested by a central Network Management System (NMS), to form an intersection of this information or to use the individual health status information for logging, messaging, alarming, diagnosis and other purposes. Listed below are five different categories, which, however, cannot claim to be complete, as both the status information itself and the methods are largely manufacturer specific.

1. **Basic Connectivity & Reachability:** Is the network device online and responding?
2. **Interface (Port) Health:** Are the individual connections working correctly?
3. **Resource Utilization:** Is the network device's internal hardware stressed?
4. **Environmental Conditions:** Is the physical environment affecting the network device?
5. **Redundancy Status:** If applicable, is its redundancy functioning?

Capabilities and Services to Get a network device Health Status

1. Basic Connectivity & Reachability

This is the foundational check to ensure the network device is present and communicating on the network.

- **Capability:** Verify the network device's network presence and responsiveness.
- **Services / Protocols:**
 - **ICMP (Ping):** Send an ICMP echo request to the network device's management IP address. A successful reply indicates basic network layer reachability.
 - **TCP Port Scan (e.g., , HTTP/HTTPS, SSH):** Attempt to connect to the specific ports used by the network device's management interfaces. If these ports are open and respond, it indicates that the management plane is active.
 - **ARP Table Monitoring:** Check if the network device's MAC address is present in the ARP tables of connected devices or routers, confirming its presence on the local segment.

2. Interface (Port) Health

Individual port health is vital, as these are the connections to the end devices.

- **Capability:** Monitor the operational state and performance of each physical and logical port.
- **Services / Protocols:**
 - **Link Up/Down Status:** It provides the information, if a port is physically connected and active. An "up" status means the link light is on; a "down" status means there's no physical connection or a problem.
 - **Administrative Status:** Shows if a port is enabled or disabled by configuration. It is expected to be "up" for active ports.
 - **Error Counters:** Track packets that were malformed or corrupted. Rising error counts can indicate bad cabling, faulty transceivers, or duplex mismatches.
 - **Discard Counters:** Track packets that the network device dropped due to buffer overflow or policy. High discards suggest congestion or an overloaded port.
 - **Bandwidth Utilization:** Monitor the amount of data flowing through the port. High utilization can indicate a bottleneck or excessive traffic.
 - **Duplex and Speed Settings:** Verify that ports are operating at the expected speed (e.g., 100 Mbps, 1 Gbps, 10 Gbps) and duplex (full-duplex is almost always desired). Mismatches can cause significant performance issues.

3. Resource Utilization

The network device itself has internal components that can become overloaded.

- **Capability:** Assess the load on the network device's internal processing and memory.
- **Services / Protocols:**

- **CPU Utilization:** Monitor the percentage of CPU being used by the network device. Consistently high CPU can indicate heavy traffic processing, misconfiguration, or even a software bug.
- **Memory Utilization:** Track the amount of RAM being used. High memory usage could point to memory leaks or large routing tables.
- **Buffer Utilization:** Monitor the network device's internal packet buffers. Overflows here contribute to discards and can indicate congestion.

4. Environmental Conditions

Network redundancy is essential in an OCN.

- **Capability:** Monitor the physical environment of the network device.
- **Services / Protocols:**
 - **Temperature Sensors:** Many industrial switches, esp. EN50155 capable network devices have built-in temperature sensors. Supervising the temperature can inform the network management system about overheating issues in cabinets.
 - **Power Supply Status:** For network devices with redundant power supplies, monitor the status of each supply (e.g., "active," "standby," "failed").

5. Redundancy Status (only if applicable)

If the network design includes redundancy, its health is also very important.

- **Capability:** Verify that redundancy protocols are active and functioning correctly.
- **Services / Protocols:**
 - **RSTP/MSTP (Spanning Tree Protocol):** Monitor port roles (Root, Designated, Blocked), bridge priorities, and topology changes. Unexpected changes can indicate a loop or a failed link.
 - **VRRP/HSRP (Virtual Router Redundancy Protocol/Hot Standby Router Protocol):** If the network device acts as a router, monitor the master/backup status of the virtual IP address.

Tools and Systems to Collect and Present Health Status

To effectively gather and interpret all this data, a Network Management System (NMS) is essential, which is also able to monitor the network and end devices.

- **Network Management Systems (NMS):** These are specialized software platforms designed for managing and monitoring of network and end-devices.
- **SCADA/HMI Systems:** While primarily for process control, some SCADA systems can integrate basic network device monitoring.
- **Centralized Logging Systems:** To collect and analyze syslog messages from switches.

2.2.1.2 Management functionality covered by event notifications

In the environment of modern trains, maintaining a robust, secure, and efficient onboard network is critical for everything from passenger services to operational control systems. Understanding the status of every connected device is key to ensuring continuous service and addressing issues fast. This functional interface is designed to provide clear, actionable notifications regarding the status and behaviour of Ethernet network and end devices across the overall onboard communication network.

This interface acts as a monitoring unit on the network, translating complex technical events into understandable alerts that empower service and maintenance teams to manage and respond effectively.

The core purpose of this interface is to deliver timely information about significant changes or potential concerns related to onboard Ethernet end devices. The monitoring system will receive immediate notifications for crucial events, including:

1. “Device Added”: Get an alert the moment a new Ethernet device successfully joins the network, helping to track new additions or expected reconnections.
2. “Device Removed”: Be informed instantly when an Ethernet device leaves the network, allowing for quick investigation of unexpected removals or planned disconnections.
3. “Device Enabled”: Understand when a device that was previously inactive or offline becomes operational, confirming successful re-enablement or power-up.
4. „Device Disabled”: Receive a notification when a device goes offline or is intentionally disabled, aiding in troubleshooting or verifying planned maintenance.
5. “Unusual Traffic Detected”: Be alerted to unusual or anomalous network traffic originating from a device, which could indicate a security threat, a misconfiguration, or a malfunctioning component. (to be discussed)
6. “Failed IEEE 802.1X Authorization”: Get immediate notice if a device attempts to connect but fails to meet security authentication requirements (e.g., IEEE 802.1X), highlighting potential unauthorized access attempts or configuration errors.

For the management functionality event notification, the solution for the listed UCs: **(Ref.: UC 10004, 10017, 10025, 10029 and 10030)** of D24.1 [1] is based on AMQP in accordance with D23.4.

Proposals for the methods and the content of the descriptions of the interfaces are described below.

There are basically the following three roles involved:

- **AMQP Broker:** A robust broker like RabbitMQ should be used.
- **Publishers:** Various network management systems, security tools, or custom scripts that detect these events.
- **Consumers:** Different applications (alerting systems, inventory databases, security dashboards, ticketing systems) that need to react to these events.

Event-Specific Implementation Details

For each event, different roles/tasks have to be considered:

1. **The Publisher:** What system detects this event?
2. **The Message Structure (Payload):** What data should be included in the message?
3. **Severity Level (Header):** How critical is the event? (e.g. define and assign a value like critical, error, warning, info, or debug.)
4. **The Routing Key:** How will the message be categorized for routing?
5. **Example Consumers:** Who might be interested in this event?

1. “Device Added”:

Publisher:

- **NMS (Network Management System):** When a new device is discovered (e.g., via HTTP Rest, Syslog, LLDP).
- **Inventory Management System:** When an administrator manually adds a device to the inventory.
- **Orchestration System:** When a new virtual device or network function is provisioned.

Message Structure (Payload):

Example for realization by a JSON file:

```
{
  "eventType": "device.added",
  "timestamp": "2026-01-09T14:00:00Z",
  "deviceId": "EVC-SMO-001",
  "deviceName": "Siemens-European Vital Computer-1",
  "ipAddress": "10.0.10.1",
  "macAddress": "00:2A:2B:3C:4D:5E",
  "vendor": "Siemens",
  "model": "Trainguard OBU",
  "location": "Berlin",
  "discoveredBy": "NMS-Discovery-Service"
}
```

Routing Key: network.device.added

Example Consumers:

- **Inventory Database:** To update its records.
- **Configuration Management System:** To initiate baseline configuration or compliance checks.
- **Monitoring System:** To start polling the new device for KPIs.

- **Security System:** To register the new device for security monitoring.

2. “Device Removed”:

Publisher:

- **NMS:** When a device becomes unreachable for an extended period or is explicitly removed.
- **Inventory Management System:** When a device is de-commissioned.
- **Orchestration System:** When a virtual device is de-provisioned.

Message Structure (Payload):

Example for realization by a JSON file:

```
{
  "eventType": "device.removed",
  "timestamp": "2026-01-09T15:30:00Z",
  "deviceId": "JRU-ALS-005",
  "deviceName": "Alstom-Juridical-Recorder-5",
  "ipAddress": "10.0.20.5",
  "reason": "Decommissioned",
  "removedBy": "Admin-User"
}
```

Routing Key: network.device.removed

Example Consumers:

- **Inventory Database:** To mark the device as retired or removed.
- **Monitoring System:** To stop polling the device.
- **Security System:** To remove the device from active security policies.

3. “Device Enabled”:

Publisher:

- **NMS:** Detects an interface or device status change (e.g., ifOperStatus goes from down to up, or sysUpTime restarts).
- **Configuration Management System:** After a device reboot or configuration change that enables it.

Message Structure (Payload):

Example for realization by a JSON file:

```
{
  "eventType": "device.enabled",
  "timestamp": "2026-01-07T16:15:00Z",
  "deviceId": "FW-DMZ-001",
  "deviceName": "DMZ Firewall 1",
  "ipAddress": "10.0.0.1",
  "statusChange": "up",
  "details": "Device rebooted and came online"
}
```

Routing Key: network.device.status.enabled (or network.device.up)

Example Consumers:

- **Alerting System:** To send a "device back online" notification.
- **Monitoring System:** To resume full monitoring or clear any "down" alarms.
- **Dashboard:** To update the device status visually.

4. "Device Disabled":

Publisher:

- **NMS:** Detects an interface or device status change
- **Configuration Management System:** After an administrator disables an interface or shuts down a device.

Message Structure (Payload):

Example for realization by a JSON file:

```
{
  "eventType": "device.disabled",
  "timestamp": "2026-01-10T16:30:00Z",
  "deviceId": "AP-Co3-V007",
  "deviceName": "Access Point Container 3 Vehicle 7",
  "ipAddress": "10.0.3.7",
  "statusChange": "down",
  "reason": "Interface Gi0/1 administratively shut down",
  "impact": "Loss of wireless connectivity in Vehicle 7"
}
```

}

Routing Key: network.device.status.disabled (or network.device.down)

Example Consumers:

- **Alerting System:** To send a critical "device down" alert (e.g. email, SMS, PagerDuty).
- **Ticketing System:** To automatically create an incident ticket.
- **Dashboard:** To highlight the device as offline.
- **Troubleshooting System:** To initiate diagnostic routines.

5. "Unusual Traffic Detected":

Publisher:

- **NetFlow/IPFIX Analyzer:** Detects anomalies in traffic patterns (e.g., sudden spike in outbound traffic, connections to unusual ports).
- **IDS/IPS (Intrusion Detection/Prevention System):** Flags suspicious traffic signatures.
- **Security Analytics Platform:** Correlates various data points to identify unusual behaviour.
- **ALG between two security zones:** unintended traffic detected

Message Structure (Payload):

Example for realization by a JSON file:

```
{
  "eventType": "security.traffic.unusual",
  "timestamp": "2026-01-12T17:00:00Z",
  "sourceIp": "10.0.1.100",
  "destinationIp": "203.0.113.50",
  "sourcePort": 54321,
  "destinationPort": 80,
  "protocol": "TCP",
  "bytesTransferred": 102400000,
  "anomalyType": "High outbound traffic to unknown IP",
  "detectionTool": "NetFlow-Analyzer-X",
  "severity": "High"
}
```

Routing Key: security.traffic.unusual

Example Consumers:

- **SIEM (Security Information and Event Management) System:** For correlation with other security events.
- **Security Operations Center (SOC) Dashboard:** To display real-time threats.
- **Automated Response System:** To potentially block the source IP or quarantine the affected device.
- **Alerting System:** To notify security analysts.

6. “Failed IEEE 802.1X Authorization”:

Publisher:

- **RADIUS Server (Authentication Server):** Logs failed authentication attempts.
- **Network Access Control (NAC) System:** Monitors 802.1X authentications and detects failures.
- **Switch/AP:** Logs local authentication failures (if not using RADIUS).

Message Structure (Payload):

Example for realization by a JSON file:

```
{
  "eventType": "security.auth.8021x.failed",
  "timestamp": "2026-01-08T17:45:00Z",
  "macAddress": "AA:BB:CC:DD:EE:FF",
  "username": "johndoe",
  "authenticatorPort": "Gi1/0/1",
  "authenticatorDevice": "SW-BLDG-A-003",
  "reason": "Invalid credentials",
  "authServer": "RADIUS-SERVER-1",
  "attemptCount": 3
}
```

Routing Key: security.auth.8021x.failed

Example Consumers:

- **SIEM System:** To detect brute-force attempts or unauthorized access.
- **Security Operations Center (SOC) Dashboard:** For real-time monitoring of access attempts.
- **User Management System:** To potentially lock user accounts after multiple failures.
- **Alerting System:** To notify administrators of suspicious authentication activity.

2.2.1.3 Management functionality for polling network KPIs

The management functionality for polling network KPIs by using HTTP-REST provides a solution for the **US 10033** of D24.1 [1]. While SNMP is traditionally dominant, many “newer” devices expose rich REST APIs for this purpose.

A “minimal specification” here implies defining the endpoints, methods, parameters, and expected response structure for some of the listed network KPIs:

- Traffic and bandwidth KPIs (e.g. throughput in bits per second, queue lengths).
- Error and quality KPIs (e.g. dropped packets and CRC errors).
- Performance usage KPIs (e.g. RAM and CPU utilization).
- Availability KPIs (e.g. link status, up- or downtime).
- Security related KPIs (e.g. unauthorized access attempts).

A base URL is assumed for a network device/controller, for example `https://[device-ip-or-hostname]/api/v1/`.

Subsequently, a minimal HTTP REST polling specification should be broken down for the KPI categories ‘Traffic and Bandwidth KPIs’ and ‘Error and Quality KPIs’:

1. Traffic and Bandwidth KPIs

KPIs: Throughput (bits per second), Queue Lengths

Endpoint: Get Interface Statistics

Purpose: Retrieve real-time traffic statistics for a specific network interface.

HTTP Method: GET

Path: /interfaces/{interfaceId}/stats

Path Parameters:

- interfaceId (string, required): Unique identifier for the network interface (e.g., Gi0/1, Ethernet1/1, port-channel1).

Query Parameters:

- interval (integer, optional): The duration in seconds over which to calculate averages (e.g., 300 for 5 minutes). If not provided, the device might return instantaneous values or its own default averaged values.

Expected Response (HTTP 200 OK):

Example for realization by a JSON file:

```
{
  "timestamp": "2024-01-18T18:00:00Z",
  "interfaceId": "Gi0/1",
```

```

    "inOctetsPerSecond": 123456789, // Throughput in bits/sec (or
    bytes/sec, clarify in API doc)
    "outOctetsPerSecond": 987654321, // Throughput in bits/sec
    "inPacketsPerSecond": 15000,
    "outPacketsPerSecond": 20000,
    "inputQueueLength": 5, // Number of packets in input queue
    "outputQueueLength": 2 // Number of packets in output queue
}

```

Notes:

- inOctetsPerSecond and outOctetsPerSecond should ideally be in bits per second, but the API documentation should clearly state the unit (bits vs. bytes).
- The device is responsible for calculating the "per Second" metrics based on its internal counters and the requested interval.

2. Error and Quality KPIs

KPIs: Dropped Packets, CRC Errors

Endpoint: Get Interface Errors

Purpose: Retrieve error statistics for a specific network interface.

HTTP Method: GET

Path: /interfaces/{interfaceId}/errors

Path Parameters:

- interfaceId (string, required): Unique identifier for the network interface.

Query Parameters:

- resetCounters (boolean, optional): If true, the device resets the error counters after returning the current values. Use with caution, typically false or omitted for polling.

Expected Response (HTTP 200 OK):

Example for realization by a JSON file:

```

{
  "timestamp": "2024-01-18T18:01:00Z",
  "interfaceId": "Gi0/1",
  "inErrors": 15,           // input errors since last reset/boot
  "outErrors": 2,          // output errors
  "inDiscards": 10,        // input packets discarded
  "outDiscards": 0,        // output packets discarded
  "inCrcErrors": 3,        // Total CRC errors on input

```

```
"inFrameErrors": 0,      // Total frame errors on input
  "lastClearTime": "2024-01-17T08:00:00Z" // When counters were
last reset
}
```

- **Notes:**

- The monitoring system will need to calculate the *rate* of errors (e.g., errors per second) by polling at intervals and comparing current values to previous ones.
- lastClearTime is crucial for understanding the context of the absolute error counts.

2.2.2 Technical Interface Specifications

This section dives into the "how" of network communication, providing detailed technical blueprints for the interfaces that facilitate essential management functionalities. It specifies the protocols, data formats, and interaction sequences required for tasks like adding, removing, and updating devices on the network. These specifications are crucial for the train manufacturer, the TOCs and the maintainer to ensure precise implementation, interoperability, and secure integration of all components within the onboard communication system.

2.2.2.1 Management functionality to add a new device

The management functionality to “add a new device” details the technical interfaces and interactions required for onboarding a new CCS end device onto a train's onboard network, focusing on integration and operational readiness., ensuring secure authentication and proper access management in accordance with the listed user stories (**Ref. US 10004, 10025 and 10026**) of D24.1 [1].

The onboarding process involves three main steps:

1. **Device Identity Registration:** The device's unique identity and initial network parameters are securely recorded in the central **Train Inventory System (TIS)**.
2. **Access Provisioning:** The TIS (alternatively a specific service or a manual provision of access data) configures the **Train Authentication Server (TAS)** and the relevant **Onboard Network Switch (ONS)** to grant network access to the new CCS device.
3. **Device Authentication & Activation:** The CCS device authenticates itself, and the TIS is notified of its successful authentication, marking it as fully operational.

System Components (Train-Specific)

- **Train Inventory System (TIS):** Central management system for all onboard network devices.
- **Train Authentication Server (TAS):** RADIUS server (e.g., FreeRADIUS) responsible for authenticating CCS devices using 802.1X.

- **Onboard Network Switch (ONS):** Network access device within the train, connecting CCS end devices.
- **CCS End Device (CCS-Dev):** The specific Communication Control System device being added (e.g., a new control unit, diagnostic terminal, or communication module).

Interface Specifications

Step 1: CCS Device Identity Registration (TIS API)

An authorized entity (e.g., maintenance technician, automated provisioning tool) registers the new CCS device with the TIS.

- **Interface:** RESTful API (HTTPS)
- **Action:** POST /ccs-devices/register
- **Key Information Sent to TIS:**
 - deviceId: Unique identifier for the CCS device (e.g., UUID, serial number).
 - deviceName: Human-readable name (e.g., "BrakeControlUnit-Car3").
 - publicCertificate: The public key certificate for 802.1X EAP-TLS authentication.
 - allowedPort (optional): Specific ONS port if known/required.
 - deviceFunction: Critical function of the CCS device (e.g., "Brake Control", "Passenger Information", "Diagnostics").

Step 2: Access Provisioning (TIS -> TAS & TIS -> ONS)

The TIS configures the necessary network components for the CCS device.

2.1. TIS -> Train Authentication Server (TAS API)

- **Interface:** RESTful API (HTTPS)
- **Action:** POST /radius/ccs-devices
- **Key Information Sent to TAS:**
 - deviceId: Unique identifier of the CCS device.
 - publicCertificate: The public key certificate for TAS to use during 802.1X authentication.
 - activationStatus: Set to "active" for immediate use.

2.2. TIS -> Onboard Network Switch (ONS API - If Necessary)

- **Interface:** RESTful API (HTTPS) or SNMP
- **Action:** PUT /ports/{portId}/config
- **Key Information Sent to ONS (for the specific port):**
 - status: "up" (unblock).
 - mode: "802.1x-controlled" (enable 802.1X).
 - vlan: Specific VLAN for CCS traffic (e.g., a safety-critical VLAN).

Step 3: CCS Device Authentication & Activation Notification (TAS -> TIS API)

Once the CCS device powers on and successfully authenticates via 802.1X, the TAS notifies the TIS.

- **Interface:** RESTful API (HTTPS)
- **Action:** POST /ccs-devices/authenticate
- **Key Information Sent to TIS (from TAS, e.g., via rlm_rest module):**
 - deviceId: The unique identifier of the authenticated CCS device.
 - authenticationStatus: "success".
 - authenticatedAt: Timestamp of successful authentication.
 - sourceIp: IP address assigned to the CCS device.
 - sourcePort: The ONS port where the device authenticated.
 - authMethod: "802.1X EAP-TLS".

2.2.2.2 Management functionality to remove a device

The Management functionality to remove a device details the technical interfaces and interactions required to securely and completely decommission a CCS end device from a train's onboard network. The process ensures that the device no longer has network access and its identity is removed from all relevant systems. (Ref.: **US 10004, 10025, 10026, 10030**) of D24.1 [1].

The decommissioning process involves three main steps, essentially reversing the onboarding:

1. **Device De-registration Request:** An authorized entity requests the removal of a specific CCS device from the central **Train Inventory System (TIS)**.
2. **Access Revocation:** The TIS communicates with the **Train Authentication Server (TAS)** and the relevant **Onboard Network Switch (ONS)** to revoke network access for the device.
3. **Device Identity Removal Confirmation:** The TIS confirms the complete removal of the device's identity across all systems.

System Components

- **Train Inventory System (TIS):** Central management system for all onboard network devices. Orchestrates the decommissioning process.
- **Train Authentication Server (TAS):** RADIUS server (e.g., FreeRADIUS) responsible for authenticating CCS devices. Will be instructed to remove the device's authentication credentials.
- **Onboard Network Switch (ONS):** Network access device within the train. Will be instructed to block or disable the port previously used by the device.
- **CCS End Device (CCS-Dev):** The specific Control Command and Signalling end device being removed.

Interface Specifications

To provide a better overview, exclusively step 1 has been provided with JSON examples.

Step 1: CCS Device De-registration Request (TIS API)

This step involves an authorized entity initiating the decommissioning process by requesting the removal of a CCS device from the TIS.

Interface: RESTful API (HTTPS)

Endpoint: /ccs-devices/deregister/{deviceId} (example)

Method: DELETE

Path Parameter:

- deviceId (string, **required**): The unique identifier of the CCS device to be decommissioned.

Request Body

(as JSON)

```
{  
  "reason": "Device replaced due to upgrade",  
  "requestedBy": "MaintenanceTeam-Lead"  
}
```

- reason (string, optional): Justification for decommissioning.
- requestedBy (string, optional): Identifier of the entity initiating the request.

- **Response (JSON):**

Success (202 Accepted): Indicates the request has been accepted and processing has begun.

```
{  
  "status": "accepted",  
  "message": "Decommissioning request for deviceId 'unique-device-uuid-12345' accepted. Processing initiated.",  
  "deviceId": "unique-device-uuid-12345"  
}
```

Error (400 Bad Request, 404 Not Found, 409 Conflict):

```
{  
  "status": "error",
```

```
"message": "Error details, e.g., 'Device not found' or
'Device is currently active and cannot be immediately
decommissioned'"
}
```

- **TIS Internal Action:** Upon receiving a successful de-registration request, the TIS marks the device as "decommissioning-pending" and initiates Step 2.

Step 2: Access Revocation (TIS -> TAS & TIS -> ONS)

2.1 TIS -> Train Authentication Server

- **Action:** DELETE /radius/ccs-devices/{deviceId}
- **From:** TIS
- **To:** TAS

2.2 TIS -> Onboard Network Switch

- **Action:** PUT /ports/{portId}/status (with status: "down", mode: "disabled")
- **From:** TIS
- **To:** ONS
- **Purpose:** Disable the physical network port previously used by the device.

Step 3: Identity Removal Confirmation

- **Action:** Internal status update in TIS.
- **Purpose:** Mark device as "decommissioned" and archive its record.

2.2.2.3 Management functionality to update firmware, software or configuration

1. Stakeholder Roles and Interaction Model

The update process for safety-critical CCS (Control-Command and Signaling) components like the ETCS-OB or ATO-OB is a coordinated effort between three primary roles. The OCN serves as the secure "data highway" and orchestration platform for these interactions:

- **Train Manufacturer (OEM):**
 - **Responsibility:** Provides the validated software/firmware artifacts and configuration baselines.
 - **Action:** Signs the update packages digitally to ensure authenticity and defines the compatibility matrix (e.g., ensuring the software version matches the hardware revision).
- **TOC (Train Operating Company):**
 - **Responsibility:** Fleet-wide orchestration and "Asset Ownership."

- **Action:** Uses the **Management Web App** to authorize the rollout of a specific update to a train or a sub-fleet. They decide the "maintenance window" to ensure operational schedules are not disrupted.
- **Maintainer:**
 - **Responsibility:** On-site or remote execution and monitoring.
 - **Action:** Triggers the update process via the **Network Management System** (or the Demonstrator Control Panel introduced in section 3 in case of demonstration purpose). They monitor the real-time health status and "Liveness" of the device during the update and perform the final functional validation before the train is released back into service.

2. The Update Package: Artifacts and Release Notes

To ensure full traceability and compliance with railway standards, every update is delivered as a standardized "OCN Update Bundle." This bundle includes:

- **Software/Firmware Artifacts:** The binary images for the target partitions.
- **Cryptographic Signatures:** Ensuring the package has not been tampered with since it left the **Train Manufacturer**.
- **Comprehensive Release Notes:** A mandatory document within the package containing:
 - **Version Identifier:** (e.g., ETCS-OB_Baseline_3.6.0_R1).
 - **Change Log:** Detailed list of fixed bugs, security enhancements, and new functionalities.
 - **Safety & Regulatory Impact:** Information on whether the update requires a new safety assessment or a re-certification of the CCS subsystem.
 - **Rollback Instructions:** Specific procedures for the **Maintainer** in case of a manual intervention.

3. Fault-Tolerant Update Mechanism (A/B Strategy)

To minimize the risk of a train being "immobilized" due to a failed update, a robust **A/B (Old/New) approach** is implemented:

- **Dual-Bank Partitioning:** The end-device (e.g., EVC) maintains two independent memory banks.
 1. **Partition A (Active):** Currently running the production software.
 2. **Partition B (Staging):** The target for the new update.
- **The Workflow:**
 - The **Manufacturer's** package is downloaded to the **Staging Partition** while the train is still operational (background download).
 - Once the **Maintainer** triggers the switch, the device reboots into the new version.
 - **Health Verification:** If the device fails to report a "HEALTHY" status to the **OCN Health Service** within a predefined time and/or after a given number of

attempts, the bootloader automatically reverts to the "Old" partition, ensuring the **TOC** maintains fleet availability.

4. Alignment with Secure Component Specification

This mechanism is designed to be fully forward-compatible with the upcoming **Secure Component Specification**:

- **Secure Boot integration:** The A/B switch is only permitted if the hardware-based Root of Trust (RoT) verifies the **Manufacturer's** signature. In case of a layered structure by using a VM, this could be extended in the following way: The hardware-based Root of Trust (e.g., TPM, Intel Boot Guard) secures the **Hypervisor** (provided by the Hypervisor Vendor), which then extends trust downward to the **Guest VM / OS** (signed by the OS / VM Image Vendor), and further down to the **Application** level (verified through image signing & attestation by the SW / Container Developer). At each layer, the "HW Manufacturer's signature" is replaced by the **signature of the respective layer's vendor** — all chained back to the original hardware RoT at the bottom.
- **Access Control:** Only users with the **TOC** or **Maintainer** role (authenticated via OAuth2/RADIUS or other authenticators) can initiate the update or modify the inventory state.
- **Audit Logging:** Every step—from the Manufacturer providing the package to the Maintainer confirming the successful reboot—is logged in the **Inventory Database** for juridical and regulatory auditing.

2.2.2.4 Management functionality to get the new network inventory

The **Central Network Inventory** serves as the "Single Source of Truth" for the OCN. To ensure that the right information reaches the right stakeholder at the right time, the inventory provides a high-performance REST API (via the inventory-api) that supports granular filtering and role-specific data views.

1. Stakeholder Interactions with the Inventory

The interaction with the inventory varies depending on the operational context and the responsibilities of the three primary roles:

- **Train Manufacturer (OEM):**
 - **Context:** Commissioning and Baseline Management.
 - **Interaction:** The Manufacturer defines the "Digital Twin" of the network. They query the inventory to verify that the physical installation matches the intended design, ensuring all **Device Identities** and **Data Models** (e.g., YANG/JSON-Schema) are correctly registered before the train is delivered to the TOC.
- **TOC (Train Operating Company):**
 - **Context:** Fleet Oversight and Availability.
 - **Interaction:** The TOC performs high-level queries to monitor the **Network Health State** across the entire fleet. They use the inventory to generate compliance reports, ensuring that all devices are running authorized services and that no "ghost devices"

are present in the network. In special cases, such as when updating a digital twin that is to receive an additional ED exclusively, it must be ensured that the vehicle is a sub-fleet of the main fleet, so that all vehicles in the original fleet do not inadvertently receive this update as well.

- **Maintainer:**

- **Context:** Real-time Diagnostics and Troubleshooting.
- **Interaction:** The Maintainer utilizes **Filter Possibilities** to isolate specific issues. For example, they may query for all devices with a `device_health_state = "CRITICAL"` to identify which hardware components require physical inspection or replacement during the next maintenance window.
-

2. The Request: Advanced Filtering Capabilities

To optimize bandwidth and focus on relevant data, the OCN inventory request supports a variety of filters. This is particularly important for the **Maintainer**, who needs to drill down into specific subsystems:

- **Identity Filters:** `uuid`, `mac_address`, `hostname` (e.g., EVC-01).
- **Capability Filters:** `provided_services` (e.g., find all devices offering an `amqp-listener`).
- **Model Filters:** `data_model_version` (to check for outdated configuration schemas).
- **Health Filters:** `device_health` (OK/NOT OK) or `network_state` (REACHABLE / UNREACHABLE).

3. The Response: Comprehensive System Snapshot

The response from the inventory is a structured data object that provides a 360-degree view of the network component. Based on the requirements, the response includes:

Data Category	Description	Role Interest
Device Identities	Unique IDs (UUID), MAC addresses, and hardware serials.	All Roles (Traceability)
Data Models	Definitions of how the device is configured (e.g., YANG, JSON).	Manufacturer (Consistency)
Provided Services	List of active software endpoints (REST, AMQP).	Maintainer (Functionality)
Device Health State	Internal vitals (CPU, Memory, Container status).	Maintainer (Diagnostics)

Network Health State	Connectivity status, port assignment, and latency.	TOC (Availability)

Table 2: Inventory response as a structured data object

4. Technical Implementation Example

When a Maintainer uses the Network Management System to diagnose a connectivity issue, the system executes a filtered request:

The Request (Maintainer Perspective):

```
GET /inventory/devices?device_health=NOT_OK&include_network_state=true
Authorization: Bearer <Maintainer_OAuth2_Token>
```

The Response (JSON Schema):

```
{ "inventory_snapshot": {
  "timestamp": "2026-03-11T21:15:00Z",
  "devices": [
    {
      "identity": {
        "uuid": "72b15ccc-...",
        "name": "JRU-01-Primary"
      },
      "data_model": "OCN-CCS-JRU-v2.1",
      "provided_services": [
        { "service": "log-streamer", "port": 9005 },
        { "service": "rest-api", "port": 8080 }
      ],
      "device_health_state": "CRITICAL (Disk Full)",
      "network_health_state": "REACHABLE (VLAN 200)"
    }
  ]
}
```

3 LAB DEMONSTRATOR

The transition from theoretical architectural requirements to a functional railway communication system requires a controlled environment where modularity, security, and interoperability can be rigorously tested. Chapter 3 details the realization of the OCN (On-board Communication Network) Laboratory Demonstrator, a proof-of-concept environment designed to simulate the "One Common Bus" philosophy within a modern railway context.

The objective of this chapter is to provide a comprehensive view of the demonstrator from three distinct perspectives:

Functional Scope (Section 3.1): Defining the operational boundaries and the specific use cases—ranging from individual device health checks to network-wide orchestration, that the demonstrator is designed to address.

Technical Architecture (Section 3.2): A deep dive into the "engine room" of the system, detailing the containerized microservices landscape (Podman), the specialized software stack (FastAPI, RabbitMQ, RADIUS), and the centralized control interface.

Procedural Validation (Section 3.3): A step-by-step description of the implemented Proof-of-Concepts (PoCs), outlining the exact communication flows and REST/AMQP interactions required to execute the defined use cases.

By combining these perspectives, this chapter serves as a technical blueprint for the OCN implementation, demonstrating how standardized IT technologies can be safely and effectively applied to the rigorous demands of Control-Command and Signalling (CCS) infrastructures.

3.1 SETUP AND SCOPE

The laboratory demonstrator is designed to provide a practical and foundational experience in managing network infrastructures and end devices of the Onboard Communication Network (OCN) (also known as the One Common Bus). This section guides the reader through some of its key functionalities, which cover a selection of use cases from the interaction with individual devices to the management of the network inventory and performing network wide reconfigurations.

Essentially, the following three subsections "1. Working with a Device", "2. Working with the Inventory" and "3. Working with the Network" summarize the selected management functionalities which were chosen for this purpose.

3.1.1 Working with a Device:

This module focuses on the direct interaction with a single network device, giving insight into its health, identity, and lifecycle management status.

a) Querying the Health Status of a Device:

- Health Status (BOOL: OK | NOT OK): to determine the fundamental operational state of a network device. This binary indicator provides an immediate understanding of whether a device is performing within expected parameters or requires attention and maybe a replacement, so that the TOC or maintainer can decide whether corrective actions on network device are necessary.
- Key Performance Indicators (KPIs): These simple yet powerful indicators help to monitor traffic flow, identify potential bottlenecks, or detect unusual activity on specific interfaces.

b) Software/Firmware/Configuration Updates:

Simulated Updates: To understand the critical workflow of updating device software or firmware without the complexities of a full update of a network and/or end device, a "pseudo" mechanism is used for demonstrating the functionality of the specified API. The pseudo process of updating the device's software involves the upload of a simple text file. Following processing by the device's "identification service", this results in the device reporting a new software version (included in the text file). This process serves to demonstrate the update process and its impact.

c) Querying Device Status (Identification):

Reporting Device Identification: Discover how to reliably retrieve essential identification details from a device. This includes crucial information like its firmware version, model, and other unique identifiers, which are indispensable for inventory management, troubleshooting, and ensuring compliance.

3.1.2 Working with the Inventory

Moving beyond individual devices, this module introduces the critical concept of network inventory management – keeping track of the most relevant network assets.

a) Querying the Inventory Status:

The purpose of the inventory is to enable an exploration of the network and its components and to maintain an accurate overview of the network. The backbone of the inventory system in the demonstration is a relational database (e.g., SQLite, PostgreSQL), which is accessible via a Python API that processes the actual REST requests. REST requests are also used to store information in the inventory.

b) Inventory Manipulation / Adding a (new) device to the Inventory: The process of formally integrating a new device into the network inventory. This is not just about physically connecting it; it is about ensuring the network management system is aware of its presence and can thus track the device status and configuration.

c) Inventory Manipulation / Removing a Device from the Network:

Removing a device from the Inventory: Understand the proper procedure for decommissioning a device, ensuring it's correctly removed from an inventory system to maintain data accuracy and avoid managing ghost devices.

3.1.3 Working with the Network: Orchestrating Network-Wide Changes

This final module focuses on executing broader network management tasks that affect multiple devices or the network's overall behaviour.

Blocking or Unblocking a Network Port:

Activating/Deactivating Connectivity: Discover how to control network access at the port level. This functionality is directly linked to activating or deactivating a device's connectivity, allowing the user to isolate or enable specific network segments as needed.

3.2 OVERALL ARCHITECTURE

To validate the architectural concepts of the On-board Communication Network (OCN) and demonstrate its feasibility in a railway environment, a functional laboratory demonstrator was developed. This implementation serves as a modular proof-of-concept, translating high-level requirements into a working software environment. The following subsections detail the technical realization of the demonstrator, beginning with the containerized service landscape, followed by a deep dive into the specific software stack, and concluding with the operational interface used for system orchestration and the demonstration of the selected demonstration cases in chapter 3.3.

3.2.1 Architectural Overview: Containerized Service Landscape

The architecture of the laboratory demonstrator follows a microservices approach, ensuring modularity, scalability, and isolation of railway management functions. Figure 6 illustrates the logical distribution of services across various **Pods** and **Containers** within the Podman host environment.

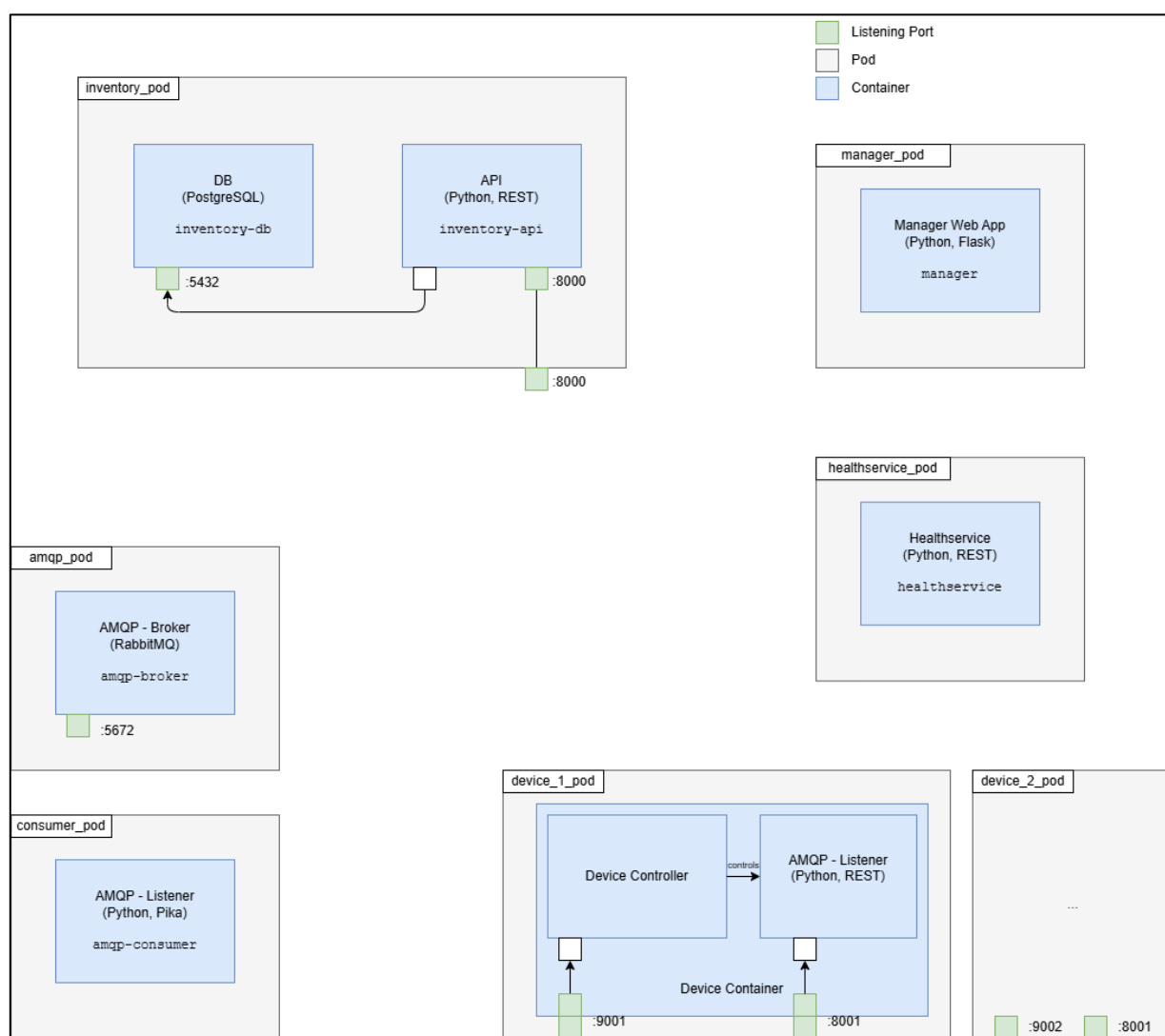


Figure 6: Logical distribution of services

1. Legend and Component Definitions

To ensure clarity across the documentation, the following visual identifiers are used:

- **Pod (White Box):** Represents a group of one or more containers that share the same network namespace and storage.
- **Container (Blue Box):** The individual executable unit containing the application code and dependencies.
- **Listening Port (Green Square):** Indicates an exposed network port through which the service communicates with other pods or external systems.

2. Core Service Pods

The central management and data handling are distributed across several specialized pods:

- **Inventory Pod (inventory_pod):**
 - Acts as the central data repository.
 - Contains two containers: **inventory-db** (PostgreSQL) and **inventory-api** (Python REST API).
 - The API serves as the gateway to the database, exposing **Port 8000** for external requests while communicating internally with the DB on **Port 5432**.
- **Manager Pod (manager_pod):**
 - Hosts the **Manager Web App** (Python/Flask).
 - This provides the user interface or orchestration logic for the system.
- **Health Service Pod (healthservice_pod):**
 - Runs a dedicated **Healthservice** (Python/REST) to monitor the status and "liveness" of the OCN components.

3. Messaging and Communication Layer

Asynchronous communication and event-driven logic are handled by the Advanced Message Queuing Protocol (AMQP) infrastructure:

- **AMQP Broker Pod (amqp_pod):** Runs a **RabbitMQ** broker, exposing **Port 5672**. This serves as the central "post office" for messages between services.
- **Consumer Pod (consumer_pod):** Contains an **AMQP Listener** (Python/Pika) designed to process messages from the broker.

4. Edge / Device Integration Pods

The architecture is designed to scale with the number of physical end devices on the train:

- **Device Pods (device_1_pod, device_2_pod, etc.):**
 - Each physical device integration is mirrored by a dedicated pod.
 - **Device Container:** Contains the **Device Controller** and an **AMQP Listener**.

- **Port Mapping:** Each pod uses specific ports for control and monitoring (e.g., **Port 9001/9002** for the controller and **Port 8001** for the listener). This allows the system to distinguish between multiple devices on the same network.

3.2.2 Software Component Descriptions

To translate the logical and networked architecture described in the previous sections into a functional, production-ready system, a robust and interoperable software stack is required. The laboratory demonstrator utilizes a microservices-based approach, where each functional requirement—ranging from secure network access to asynchronous messaging—is handled by a specialized software component.

The selection of these components is guided by three core principles:

1. **Standardization:** Utilizing industry-standard protocols (such as OAuth2, AMQP, and IEEE 802.1X) to ensure compatibility with existing railway infrastructure and future-proof integration.
2. **Modularity:** Leveraging containerization via Podman to isolate services, allowing for independent updates, scaling, and fault tolerance without impacting the entire fleet network.
3. **Performance & Security:** Implementing high-performance frameworks like FastAPI and proven security tools like FreeRadius to meet the rigorous demands of the Control-Command and Signalling (CCS) environment.

The following sections detail the specific implementation of these components, outlining the frameworks and tools that form the "engine" of the OCN service landscape.

Components:	Implemented by means of:
API authentication	OAuth2
AMQP broker	RabbitMQ
AMQP client	Pika
<i>RADIUS server</i>	FreeRadius
<i>Linux 802.1X / RADIUS client</i>	wpa_supplicant
<i>RADIUS CoA Client (optional)</i>	Radclient
Inventory database	PostgreSQL
Inventory service framework	Python FastAPI
Demonstration / use case manager	Python Flask WebApp

Figure 7: Components and the chosen software implementation

Security & Authentication Layer

To ensure secure access to the OCN services, a multi-layered security approach is implemented:

- **OAuth2:** Used for standardized API authentication, ensuring that only authorized services and users can interact with the system's endpoints.
- **RADIUS Stack (FreeRadius, wpa_supplicant, Radclient):** This stack handles Network Access Control (NAC). It ensures that only trusted devices can connect to the physical network ports, utilizing 802.1X protocols and supporting Change of Authorization (CoA) for dynamic policy updates.

Messaging & Communication

Asynchronous communication between the various pods (e.g., from the amqp_pod to the device_pods) is facilitated by:

- **RabbitMQ:** Acts as the central AMQP (Advanced Message Queuing Protocol) broker, managing the distribution of messages and events across the network.
- **Pika:** A pure Python implementation of the AMQP 0-9-1 protocol, used by the individual service containers to interact with the RabbitMQ broker.

Data Management & Backend Services

The "source of truth" for the fleet and device configuration is managed by:

- **PostgreSQL:** A robust, relational database used within the inventory_db container for persistent storage of device metadata and network configurations.
- **FastAPI:** This modern Python framework is used to build the inventory-api. It was selected for its high performance, automatic documentation generation (Swagger), and native support for asynchronous programming.

Management & User Interface

- **Flask WebApp:** The "Manager" or "Use Case Manager" is implemented as a lightweight Flask application. It provides a user-friendly interface for administrators to monitor the system status and trigger specific OCN use cases.

3.2.3 Operational Interface: Demonstrator Control Panel

The **Demonstrator Control Panel** serves as the central orchestration and monitoring interface for a virtualized OCN environment. It provides a high-level abstraction of the underlying microservices, allowing operators to manage the lifecycle of virtualized CCS end devices, monitor system health, and enforce network security policies in real-time.

Figure 8 depicts the Demonstrator Control Panel.

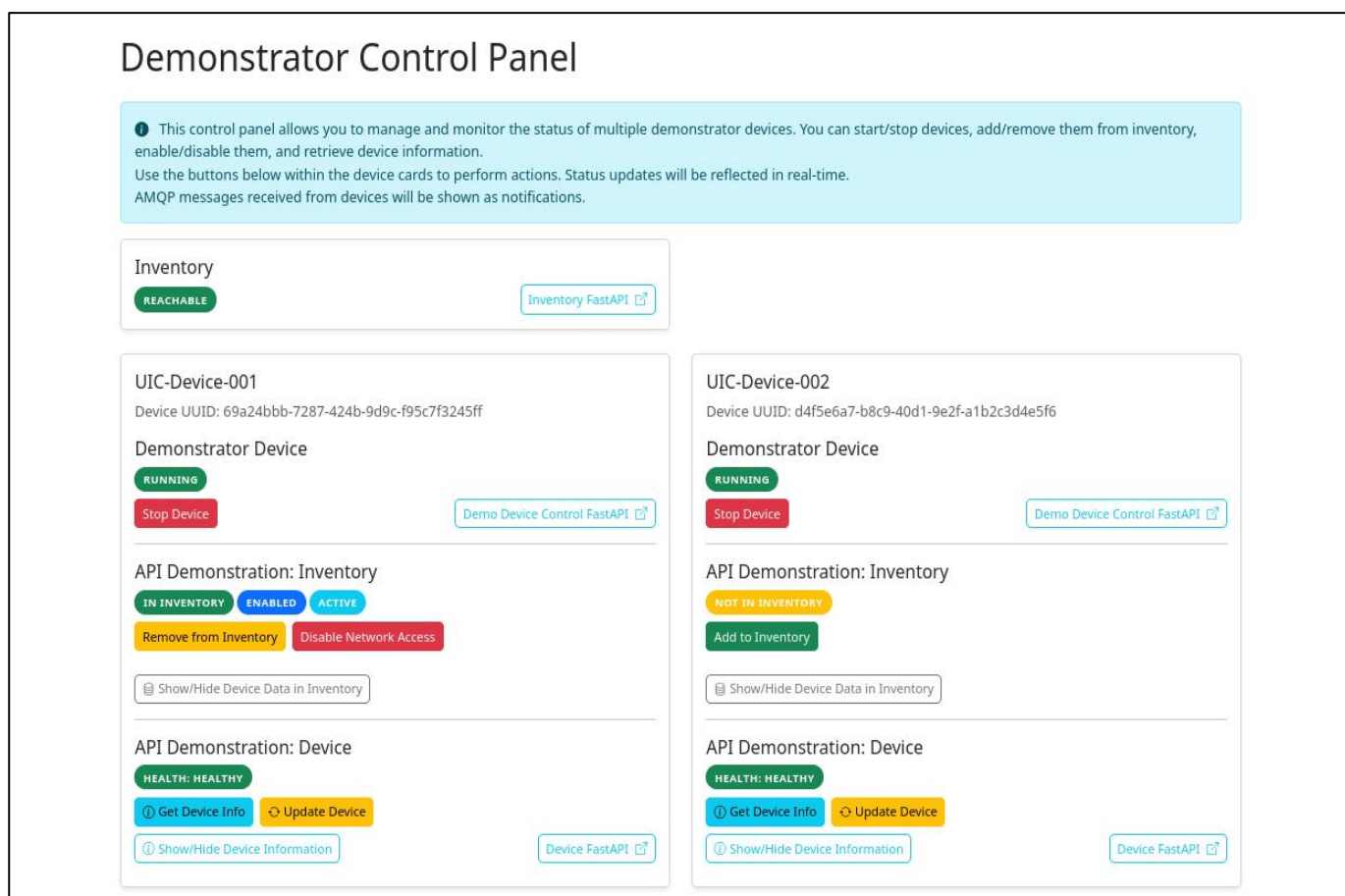


Figure 8: Demonstrator Control Panel

The interface is divided into functional modules that interact directly with the containerized service landscape.

1. Centralized Inventory Status

At the top of the dashboard, the **Inventory** card provides an immediate heartbeat check of the `inventory_pod`.

- **Status Indicator:** Displays "REACHABLE" when the **Inventory FastAPI** and the **PostgreSQL** database are online.
- **Documentation Access:** A direct link to the **Inventory FastAPI** (Swagger UI) allows developers to inspect the raw API endpoints and data schemas.

2. Device Lifecycle and State Management

Each physical or virtual device (e.g., UIC-Device-001) is represented by a dedicated control card. This section interacts with the individual **Device Pods**:

- **Execution Control:** The "**Stop Device**" button allows for the graceful shutdown of the specific device container.
- **Inventory Integration:** Operators can dynamically register or unregister devices using the "**Add to/Remove from Inventory**" buttons. This triggers a POST/DELETE request to the Inventory Service, updating the central database.

- **Real-time Visibility:** The **"Show/Hide Device Data"** toggle fetches and displays the current metadata stored in the PostgreSQL database for that specific UUID.

3. Network Access and Security Enforcement

The Control Panel acts as the management point for the **RADIUS/802.1X** security stack:

- **Network Access Control:** The **"Disable Network Access"** button (seen on UIC-Device-001) allows an operator to instantly revoke a device's network privileges. This interacts with the RADIUS server to trigger a Change of Authorization (CoA), effectively isolating the device at the network layer.

4. Device-Level Operations & Health Monitoring

The bottom section of each device card provides granular control over the device's internal logic:

- **Health Status:** Displays a real-time **"HEALTHY"** status, which is fed by the healthservice_pod monitoring the container's vitals.
- **AMQP-Driven Updates:** The **"Update Device"** and **"Get Device Info"** buttons utilize the **AMQP Broker (RabbitMQ)**. When clicked, a message is published to the broker, picked up by the device's **Pika-based listener**, and the resulting action or data is returned to the UI via asynchronous notifications.

3.3 DESCRIPTION OF IMPLEMENTED PROOF-OF-CONCEPTS

For the demonstration of the laboratory demonstrator implementation, demo cases for the three modules presented in section 3.1 were created and the process of the proof of successful test completion is described stepwise.

To visualize the demo cases in the results, in particular the interactions with the demonstrator control panel, screenshots were also included as illustrations for three demonstration cases.

3.3.1 Process for demo case: "Working with a device"

3.3.1.1 Querying the Health Status of a Device:

Prerequisite: Device is successfully authenticated and registered in the inventory

- Possibility 1: Requesting service sends a REST request (GET request) to the device, addressing directly via the IP address of the target device
- Possibility 2:
 - a. Requesting service does not know the IP address of the target device, but knows its name

- b. Requesting service sends a REST request to the inventory to obtain the IP address of the target service (analogous to DNS)
- c. Same procedure as in possibility 1

Laboratory Demonstrator result of the demo case “Querying the Health Status of a Device”:

Overview

This demo case illustrates how the health status of a virtualized CCS end device can be queried within the OCN Laboratory Demonstrator. The goal is to demonstrate the fundamental capability of determining whether a device is operational, providing a binary **OK / NOT OK** indicator as a minimal viable health check, as required by the relevant user stories (Ref. US: 10002, 10025, and 10026).

Before the health status can be queried, the target device must be:

- **Running**: the device container is active (indicated by the green **"RUNNING"** badge in the Demonstrator Control Panel see Figure 8).
- **Authenticated and registered** in the central inventory (indicated by the **"IN INVENTORY"**, **"ENABLED"**, and **"ACTIVE"** status badges).

Both conditions are visually confirmed in the Demonstrator Control Panel for **UIC-Device-001** (UUID: 69a24bbb-7287-424b-9d9c-f95c7f3245ff). **UIC-Device-002**, while running, is shown as **"NOT IN INVENTORY"**, illustrating the contrast between a fully provisioned and a not-yet-registered device.

Execution

The health status query is performed via a secured **HTTP REST GET** request directed at the device's /status endpoint:

GET /status

As documented in the Device FastAPI (Swagger UI see Figure 9), this endpoint:

- **Requires authentication** via an OAuth2 Bearer Token (obtained via POST /token).
- **Accepts no additional parameters**: the request is intentionally simple to reflect the minimal viable health check concept.

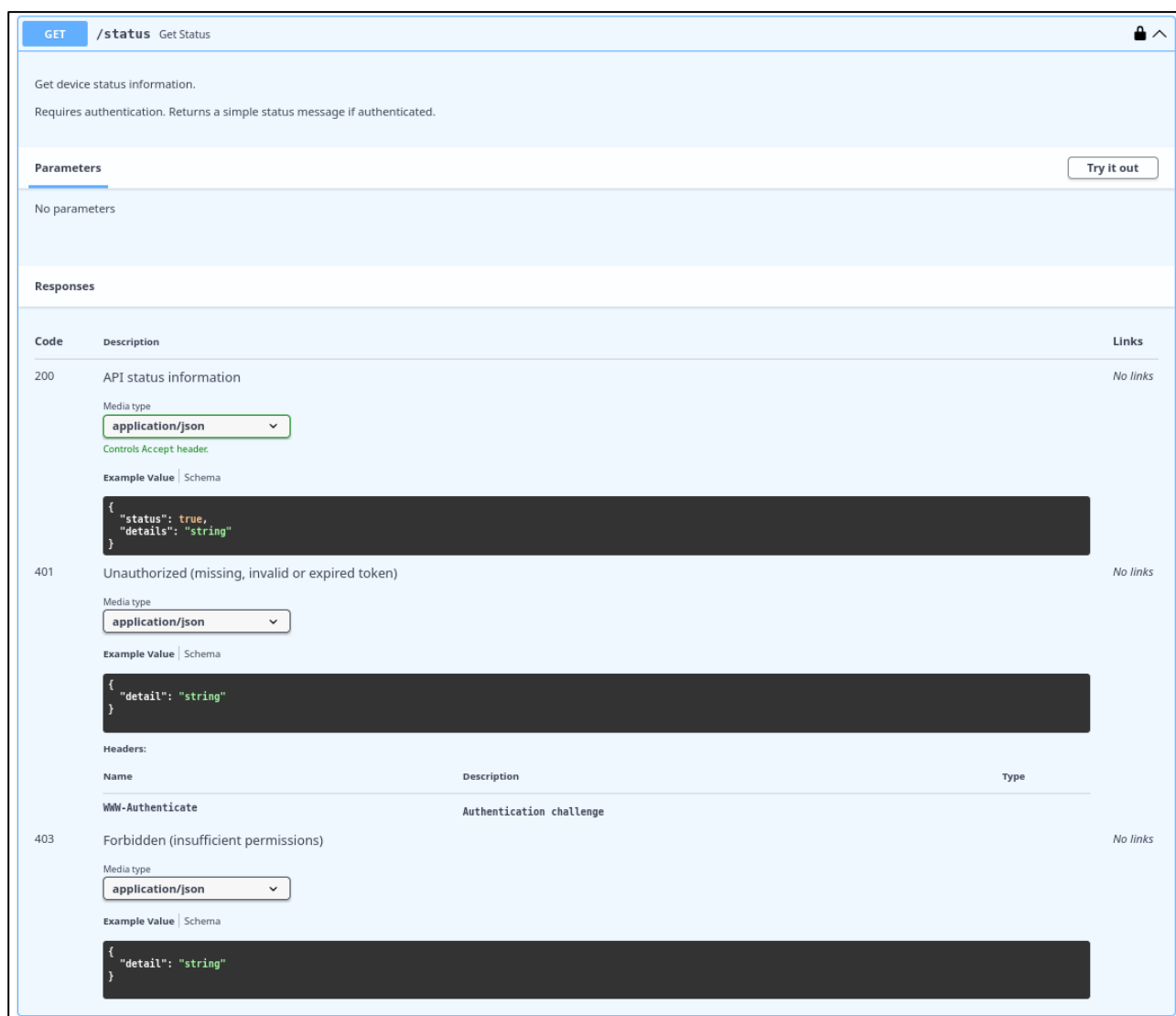


Figure 9: Demo case health status in FastApi UI

Response and Interpretation

Upon successful authentication, the device responds with a 200 OK HTTP status code and a JSON payload conforming to the **HealthState** schema:

```
{
  "status": true,
  "details": "ok"
}
```

Full information of the device including the health state could be displayed in the Demonstrator Control Panel by using the getdeviceinfo-button (see Figure 10):

UIC-Device-001

Device UUID: 69a24bbb-7287-424b-9d9c-f95c7f3245ff

Demonstrator Device

RUNNING

Stop Device

Demo Device Control FastAPI

API Demonstration: Inventory

IN INVENTORY

ENABLED

ACTIVE

Remove from Inventory

Disable Network Access

Show/Hide Device Data in Inventory

API Demonstration: Device

HEALTH: HEALTHY

Get Device Info

Update Device

Show/Hide Device Information

Device FastAPI

Device Information:

```

{
  "base": {
    "certificate": "device-cert-123",
    "device_id": "69a24bbb-7287-424b-9d9c-f95c7f3245ff",
    "device_name": "UIC-Device-001",
    "device_type": "switch",
    "ip": "10.0.10.151",
    "mac": "00:1A:2B:3C:4D:5E",
    "manufacturer": "VendorX",
    "port": 51025,
    "serial_number": "SW-XYZ1234567",
    "vlan": 112
  },
  "installed_packages": {
    "config": {
      "installed_at": "2025-00-18T12:00:00",
      "package_name": "Base Config",
      "package_type": "config",
      "url": "",
      "version": "0.0.1"
    },
    "firmware": {
      "installed_at": "2025-00-18T12:00:00",
      "package_name": "Initial Firmware",
      "package_type": "firmware",
      "url": "",
      "version": "0.0.1"
    },
    "software": {
      "installed_at": "2025-00-18T12:00:00",
      "package_name": "Initial Software",
      "package_type": "software",
      "url": "",
      "version": "0.0.1"
    }
  },
  "started_at": "2026-01-19T13:39:56.936258",
  "status": {
    "details": "ok",
    "status": true
  }
}

```

UIC-Device-002

Device UUID: d4f5e6a7-b8c9-40d1-9e2f-a1b2c3d4e5f6

Demonstrator Device

RUNNING

Stop Device

Demo Device Control FastAPI

API Demonstration: Inventory

NOT IN INVENTORY

Add to Inventory

Show/Hide Device Data in Inventory

API Demonstration: Device

HEALTH: HEALTHY

Get Device Info

Update Device

Show/Hide Device Information

Device FastAPI

Figure 10: Demo case health status via getdeviceinfo button

This demo case successfully demonstrates that a standardized, REST-based health status query can be implemented for CCS end devices within the OCN framework. The binary status field provides the minimal viable health indicator (OK / NOT OK), while the optional details field allows for extensibility. The integration with OAuth2 ensures that only authorized entities—such as the Network Management System or the Maintainer—can access device health information, maintaining the security posture required in a railway environment.

3.3.1.2 Software/Firmware/Configuration Updates:

- Step 1: Device is successfully authenticated and registered in the inventory
- Step 2: Entity in the network sends a REST request (POST request) to the device, addressing is done directly via the IP address of the target device (if necessary, query from the inventory beforehand – see case 3.3.1.1)
 - The request specifies the update package, version, checksum, etc.
 - Optional: the request does not contain the actual update package, but rather the URL to the resource, and the device then downloads the update
- Step 3: Device evaluates whether the update is permitted with the specified parameters
 - The device responds to the request with the corresponding status (update denied, update started).
- Step 4: The device starts the actual update.
 - o Optional: The device first downloads the update packages.
- Step 5: The status of the update process can be queried on the device via a REST request (GET).
- Step 6: Device completes update / update fails:
 - o AMQP notification with success/failure information, etc.
 - o AMQP reboot notification if device restarts
- Step 7: (at system start) Send device identification (version, etc.)

Laboratory Demonstrator result of the demo case “Software/Firmware/Configuration Updates”:

Figure 11 shows the **Demonstrator Control Panel** with both devices fully provisioned and operational. The control panel is described as supporting real-time status monitoring via **AMQP messaging** — notifications received from devices over the AMQP message broker are displayed directly in the UI, enabling event-driven status updates without requiring manual polling.

Both **UIC-Device-001** and **UIC-Device-002** are shown as:

- **RUNNING:** the device containers are active.
- **IN INVENTORY / ENABLED / ACTIVE:** both devices are registered and enabled in the central inventory, reachable via the Inventory REST API.
- **HEALTH: HEALTHY** both devices report a healthy status via the GET /status REST endpoint.

The expanded **Device Information** panel for **UIC-Device-002** shows the result of a GET /info REST API call, revealing the full DeviceInfo object including base device attributes and the currently installed packages:

```
"installed_packages": {
  "config": {
    "installed_at": "2025-09-18T12:00:00",
    "package_name": "Base Config",
    "package_type": "config",
    "url": "",
    "version": "0.0.1"
  }
}
```

Demonstrator Control Panel

ⓘ This control panel allows you to manage and monitor the status of multiple demonstrator devices. You can start/stop devices, add/remove them from inventory, enable/disable them, and retrieve device information. Use the buttons below within the device cards to perform actions. Status updates will be reflected in real-time. AMQP messages received from devices will be shown as notifications.

Inventory

REACHABLE

Inventory FastAPI

UIC-Device-001

Device UUID: 69a24bbb-7287-424b-9d9c-f95c7f3245ff

Demonstrator Device

RUNNING

Stop Device

Demo Device Control FastAPI

API Demonstration: Inventory

IN INVENTORY ENABLED ACTIVE

Remove from Inventory Disable Network Access

Show/Hide Device Data in Inventory

API Demonstration: Device

HEALTH: HEALTHY

Get Device Info Update Device

Show/Hide Device Information

Device FastAPI

UIC-Device-002

Device UUID: d4f5e6a7-b8c9-40d1-9e2f-a1b2c3d4e5f6

Demonstrator Device

RUNNING

Stop Device

Demo Device Control FastAPI

API Demonstration: Inventory

IN INVENTORY ENABLED ACTIVE

Remove from Inventory Disable Network Access

Show/Hide Device Data in Inventory

API Demonstration: Device

HEALTH: HEALTHY

Get Device Info Update Device

Show/Hide Device Information

Device FastAPI

Device Information:

```
{
  "base": {
    "certificate": "device-cert-456",
    "device_id": "d4f5e6a7-b8c9-40d1-9e2f-a1b2c3d4e5f6",
    "device_name": "UIC-Device-002",
    "device_type": "fantastic-device",
    "ip": "10.0.10.152",
    "mac": "AA:BB:CC:DD:EE:FF",
    "manufacturer": "Codewerk",
    "port": 51026,
    "serial_number": "SN-999999999",
    "vlan": 112
  },
  "installed_packages": {
    "config": {
      "installed_at": "2025-09-18T12:00:00",
      "package_name": "Base Config",
      "package_type": "config",
      "url": "",
      "version": "0.0.1"
    }
  }
}
```

Figure 11: Demo case UpdateDeviceConfig panel update mask before update

Figure 12 shows the Demonstrator Control Panel in the post-update state. Both devices remain RUNNING, IN INVENTORY, and HEALTH: HEALTHY, confirming that the update process did not disrupt device operation or inventory registration.

The expanded Device Information panel for UIC-Device-002 again reflects the result of a GET /info REST API call and now shows the updated installed package:

```
"installed_packages": {
  "config": {
    "installed_at": "2026-01-19T15:19:03.419851",
    "package_name": "TestNewConfig",
    "package_type": "config",
    "url": "https://example.org/update.zip",
    "version": "3.2.1"
  }
}
```

Demonstrator Control Panel

This control panel allows you to manage and monitor the status of multiple demonstrator devices. You can start/stop devices, add/remove them from inventory, enable/disable them, and retrieve device information. Use the buttons below within the device cards to perform actions. Status updates will be reflected in real-time. AMQP messages received from devices will be shown as notifications.

Inventory

REACHABLE

Inventory FastAPI

UIC-Device-001

Device UUID: 69a24bbb-7287-424b-9d9c-f95c7f3245ff

Demonstrator Device

RUNNING

Stop Device

Demo Device Control FastAPI

API Demonstration: Inventory

IN INVENTORY ENABLED ACTIVE

Remove from Inventory Disable Network Access

Show/Hide Device Data in Inventory

API Demonstration: Device

HEALTH: HEALTHY

Get Device Info Update Device

Show/Hide Device Information

Device FastAPI

UIC-Device-002

Device UUID: d4f5e6a7-b8c9-40d1-9e2f-a1b2c3d4e5f6

Demonstrator Device

RUNNING

Stop Device

Demo Device Control FastAPI

API Demonstration: Inventory

IN INVENTORY ENABLED ACTIVE

Remove from Inventory Disable Network Access

Show/Hide Device Data in Inventory

API Demonstration: Device

HEALTH: HEALTHY

Get Device Info Update Device

Show/Hide Device Information

Device FastAPI

Device Information:

```

{
  "base": {
    "certificate": "device-cert-456",
    "device_id": "d4f5e6a7-b8c9-40d1-9e2f-a1b2c3d4e5f6",
    "device_name": "UIC-Device-002",
    "device_type": "fantastic-device",
    "ip": "10.0.10.152",
    "mac": "AA:BB:CC:DD:EE:FF",
    "manufacturer": "Codewerk",
    "port": 51026,
    "serial_number": "SN-9999999999",
    "vlan": 112
  },
  "installed_packages": {
    "config": {
      "installed_at": "2026-01-19T15:19:03.419851",
      "package_name": "TestNewConfig",
      "package_type": "config",
      "url": "https://example.org/update.zip",
      "version": "3.2.1"
    }
  }
}
                
```

Figure 12: Demo case UpdateDeviceConfig panel update mask after update complete

Figure 13 shows the "Update Device: UIC-Device-002" modal on the "Configure Update" tab. A single update package is configured: TestNewConfig, version 3.2.1, type Configuration, with download URL <https://example.org/update.zip>.

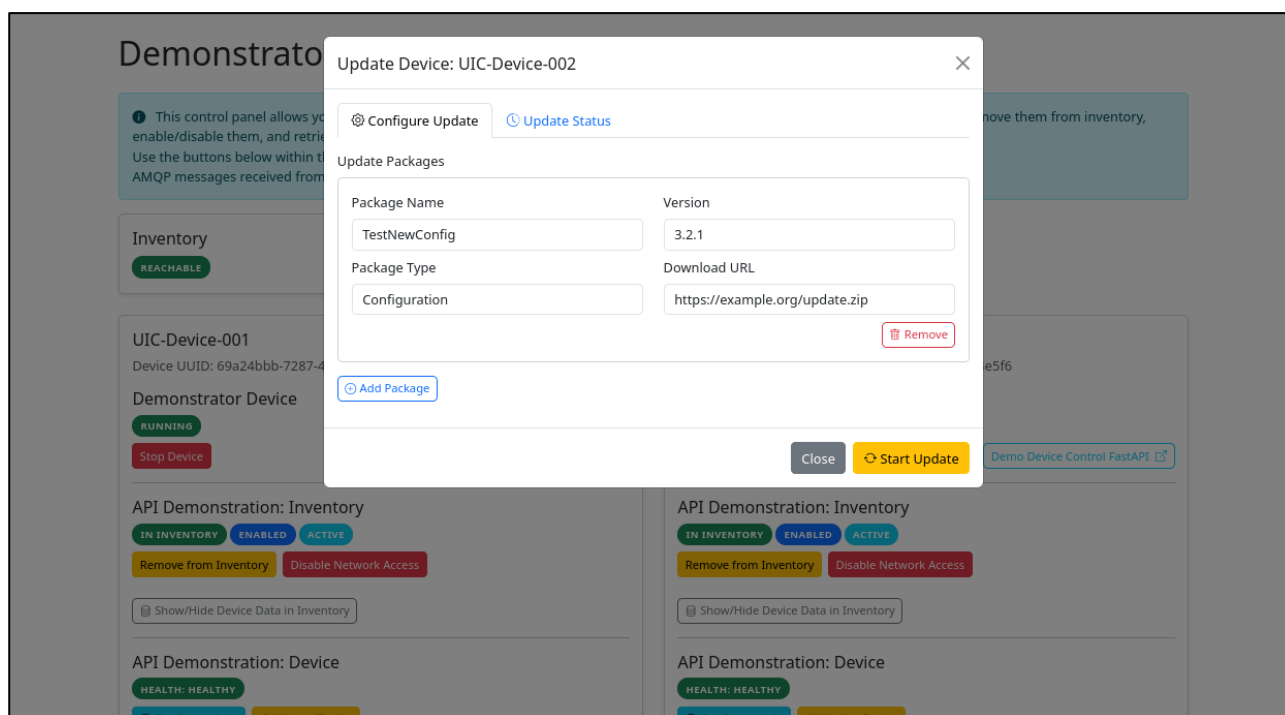


Figure 13: Demo case UpdateDeviceConfig modal mask before update

Figure 14 shows the "Update Device: UIC-Device-002" modal on the "Update Status" tab. The update has completed successfully, with status **COMPLETED**, progress 1/1, and the message "All updates applied successfully."

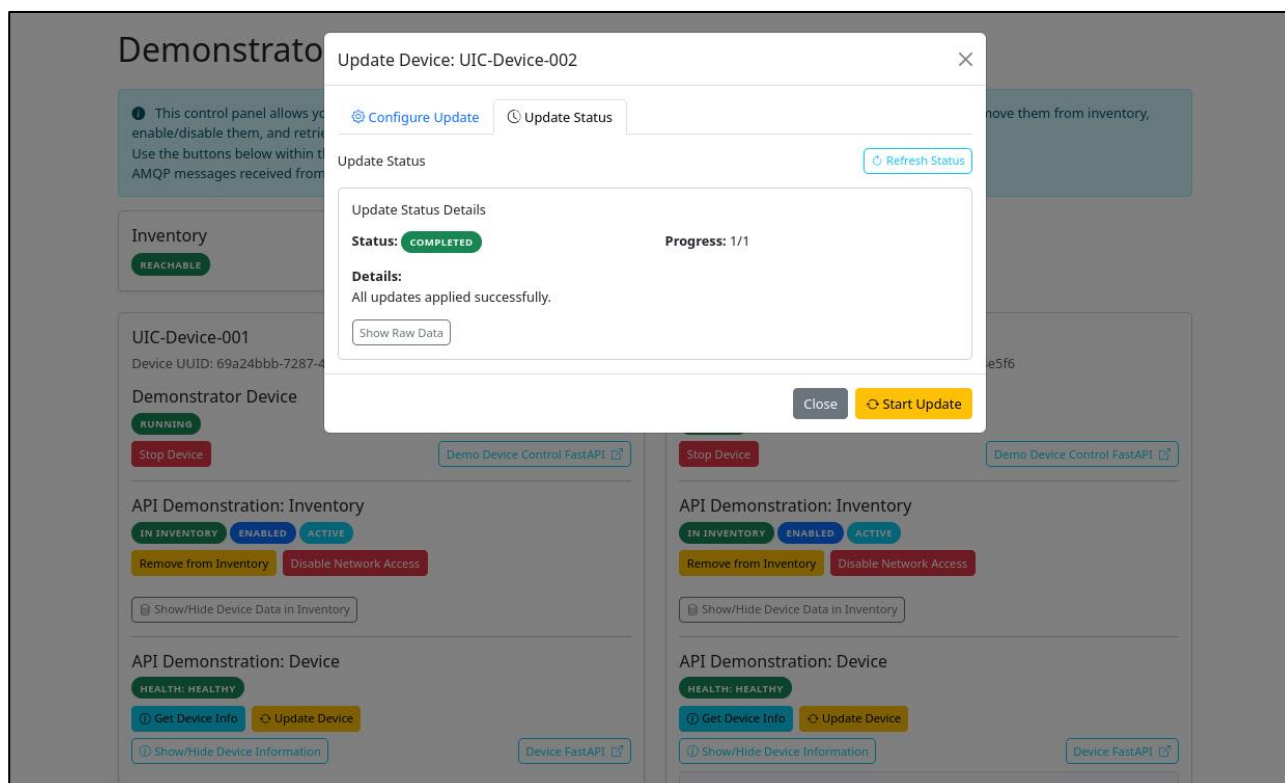


Figure 14: Demo case UpdateDeviceConfig modal mask after update

3.3.1.3 Querying Device Status (Identification):

- Trigger a broadcast via AMQP with a REST request so that all devices in the network can be informed of the status
- The notification would then contain the device name + firmware version ...

3.3.2 Process for demo case: “Working with the Inventory”

3.3.2.1 Querying the Inventory Status:

- Step 1: An entity must store entries for permitted devices in the inventory
 - a. REST request with information such as IP address, MAC address, certificate, port number, etc.
 - b. How can be ensured that not just any service can store information for new devices in the inventory?
- Step 2: REST request to inventory to query all information
- Step 3: REST request with filter strings to inventory to query only devices with specific names/IDs/IPs, etc.

3.3.2.2 Inventory Manipulation: Adding a Device to the Network:

- Step 1: Orchestration entity must communicate with the inventory to store a ‘device identity’.
 - Device identity: the public device certificate for 802.1X authentication.
 - Port: the port at which a device is to be permitted.
- Step 2: The inventory must communicate with other devices involved in order to ‘enable access’ for the new device.
 - Inventory -> Authentication server (RADIUS): store and activate the new device.
 - Inventory -> Switch: unblock a port or configure it if necessary.
- Step 3: The device starts up and logs on to the switch or RADIUS server via 802.1X, authenticating itself successfully.
 - RADIUS server or device itself sends REST request to inventory to mark device as authenticated and active.
- The request only needs to contain a unique device ID (presumably unique device UUID).
 - In addition, inventory can send broadcasts to check which devices are present in the network.

Laboratory Demonstrator result of the demo case “Adding a Device to the Network”:

Figure 15 shows the Demonstrator Control Panel in its **initial state** before UIC-Device-002 has been registered in the inventory. Both device containers are **RUNNING**, and both report **HEALTH: HEALTHY** via the device REST API. However, **UIC-Device-002** displays the status **NOT IN INVENTORY**, with only an **"Add to Inventory"** button available—indicating that no inventory REST API registration has been performed yet for this device. **UIC-Device-001** is already fully provisioned, showing **IN INVENTORY**, **ENABLED**, and **ACTIVE** status. This figure represents the starting point of the demo case, establishing the baseline before the inventory registration workflow is executed.

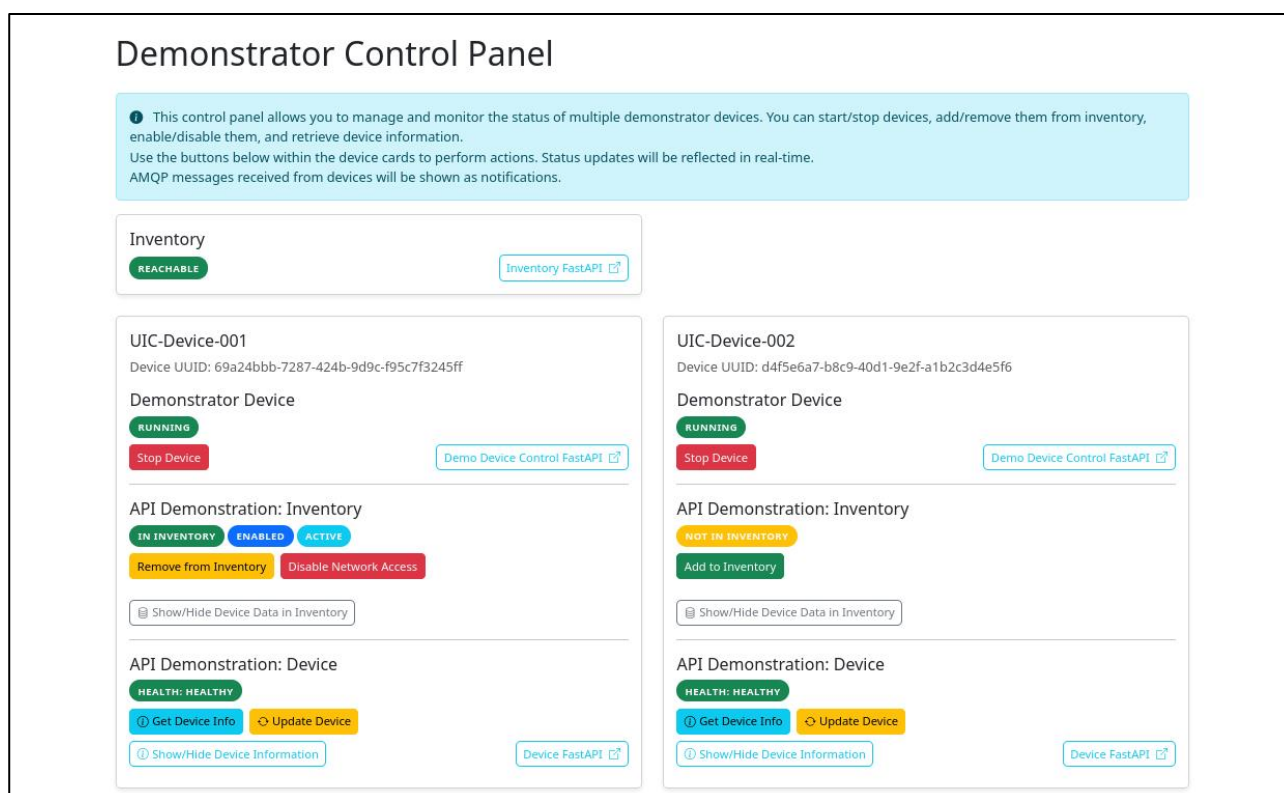


Figure 15: Demonstrator Control Panel: Initial State (UIC-Device-002 Not in Inventory)

Figure 16 shows the Demonstrator Control Panel with **UIC-Device-002** in a **DISABLED** and **INACTIVE** state—network access has been revoked via the inventory REST API. The **"Enable Network Access"** button is available to restore it. The **"Show/Hide Device Data in Inventory"** panel is expanded for UIC-Device-002, displaying the raw JSON response from the Inventory REST API:

```
{
  "active": false,
  "device_id": "d4f5e6a7-b8c9-40d1-9e2f-a1b2c3d4e5f6",
  "enabled": false,
```

```
"ip": "10.0.10.152",
"mac": "AA:BB:CC:DD:EE:FF"
}
```

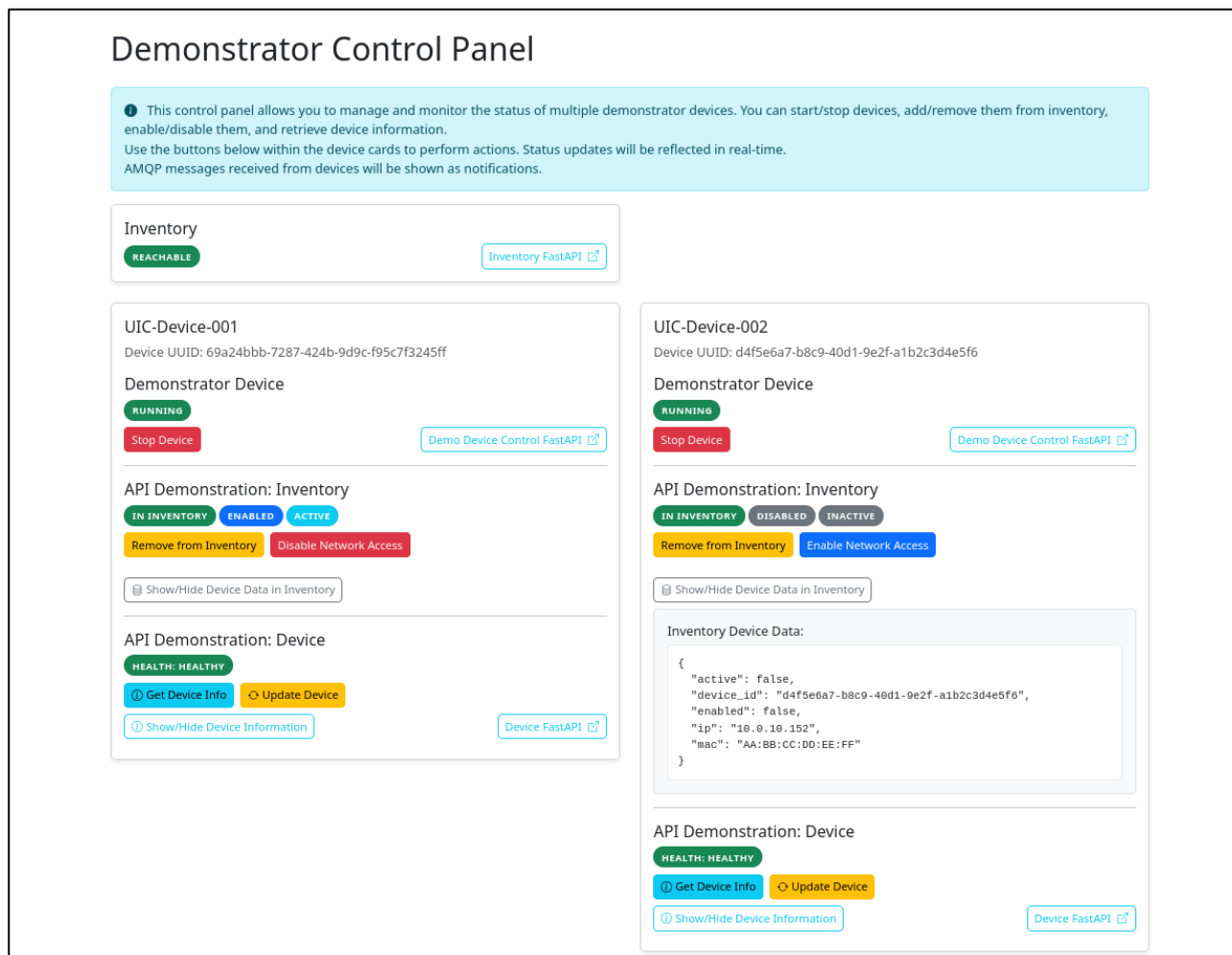


Figure 16: Demonstrator Control Panel: Device Disabled with Inventory Data Expanded

Figure 17 shows the Demonstrator Control Panel immediately after **UIC-Device-002** has been successfully added to the inventory via a REST API call. Two real-time notifications are visible in the top-right corner:

- A **cyan AMQP notification** confirming that an AMQP message was received from the inventory broker with **Topic:** device_added, **Origin:** inventory, **Type:** device_added, and the corresponding **Device ID** (d4f5e6a7-b8c9-40d1-9e2f-a1b2c3d4e5f6), demonstrating the event-driven messaging between the inventory service and the control panel.
- A **green success notification** confirming "Device UIC-Device-002 added to inventory successfully."

UIC-Device-002 now shows the status **IN INVENTORY**, but is currently **DISABLED** and **INACTIVE**, meaning it has been registered but network access has not yet been granted. **UIC-Device-001** remains fully operational with **IN INVENTORY**, **ENABLED**, and **ACTIVE** status. Both devices report **HEALTH:HEALTHY**.

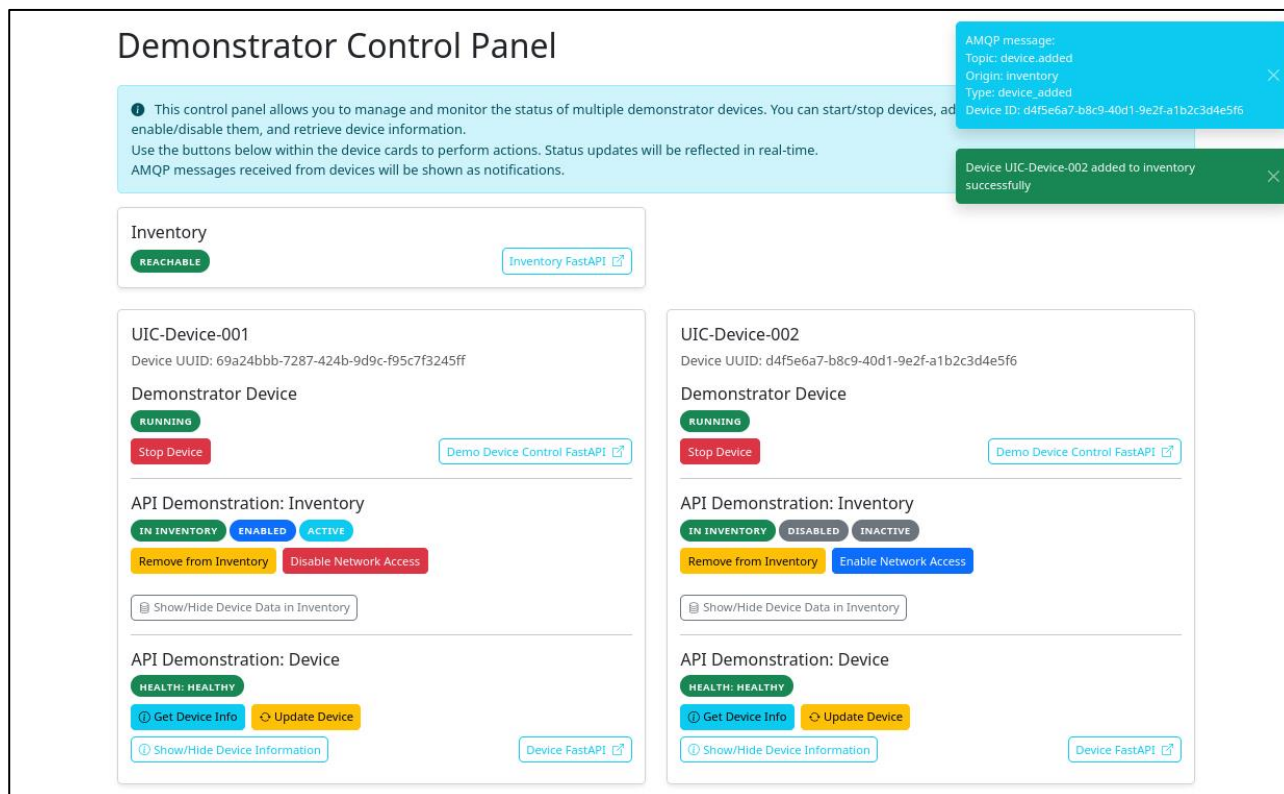


Figure 17: Demonstrator Control Panel: Device Added to Inventory (with AMQP Notification)

3.3.2.3 Inventory Manipulation: Removing a Device from the Network:

- Step 1: Orchestration entity on the network must communicate with the inventory to remove a device via a 'device identity'.
- Step 2: Notification (from inventory) to the device before removal
- Step 3: Inventory must communicate with other devices involved to 'revoke access' to the device.
 - Inventory -> Authentication server (RADIUS): remove the device.
- CoA (Change of Authorisation) and Disconnect Request are RADIUS extensions that allow the RADIUS server or admin tools to force a network device (NAS - Network Access Server) to drop an active session immediately.
 - Inventory -> Switch: 'block' a port if desired (if necessary)

3.3.3 Process for demo case: “Working with the Network: Orchestrating Network-Wide Changes”

3.3.3.1 Working with the network: “Blocking or Unblocking a Network Port”:

- Step 1a: Entity decides to deactivate port and sends REST request with corresponding port to inventory
- Step 2: Inventory links port information to the device for which it is stored
- Step 3: Inventory removes the device from its database
- Step 4: A network management entity sends a REST message to the RADIUS server stating that the specific device is no longer permitted
- Step 5: The RADIUS server removes the device and the associated certificate from its database so that the device can no longer authenticate itself in future. This can be done, for example, via the Certificate Revocation List.
 1. Revoke certificates via a CA.
 2. Update CRL. (For demonstration purpose it is initiated by the Demonstrator Control Panel, in real train applications the update of the CRL should also be initiated by a CA)
- Step 6: Current session must be terminated. IEEE 802.1X sessions are persistent until:
 - The device disconnects.
 - The switch forces a re-authentication.
 - The Switch sends a CoA (Change of Authorisation, RFC 5176) or Disconnect Message. Switch must support RADIUS Dynamic Authorisation.
- Step 7: The network management entity (for demonstration purpose: the Demonstrator Control Panel is the corresponding entity) generates message that is sent to the switch where the port must be deactivated.

Note: To shorten the demonstration case in the description, the case: “unblocking a port” was omitted. However, the underlying API is still able to unblock a port.

4 CONCLUSIONS

This chapter provides a comprehensive synthesis of the technical and procedural advancements detailed throughout this deliverable. It serves to bridge the gap between the initial architectural requirements and the practical results achieved within the R2DATO project framework (WP23 and WP24). By evaluating the transition from conceptual user stories to a validated laboratory environment, the following sub-sections summarize the key achievements, confirm the fulfilment of the project's core objectives, and reflect on the critical lessons learned during the development of the Onboard Communication Network (OCN) and the management interface to the CCS end-devices. Finally, it outlines how this work establishes a future-proof blueprint for the next generation of integrated railway architectures.

4.1 SUMMARY OF ACHIEVEMENTS

This deliverable has successfully advanced the definition and practical realization of the Onboard Communication Network (OCN) management functionalities, which serve as the technological backbone of the R2DATO project (WP23 and WP24). By focusing on the transition from abstract user stories to a concrete, validated laboratory environment, the work has established a robust framework for future OCN-based railway communication architectures.

Key achievements include:

- **Standardized Management Framework:** The establishment of a dual-layered specification approach, separating human-centric process descriptions from machine-centric technical interface specifications.
- **The "Onboard Communication Network" Validation:** The successful implementation of the Laboratory Demonstrator, proving that standardized IT technologies—such as Podman containers, FastAPI, RabbitMQ, and IEEE 802.1X—can meet the rigorous demands of railway Control-Command and Signalling (CCS).
- **Role-Based Orchestration:** The definition of clear interaction models between the Train Manufacturer (OEM), Train Operating Company (TOC), and Maintainer, ensuring the OCN supports the entire vehicle lifecycle.

4.2 FULFILLMENT OF OBJECTIVES

The objectives stated in the introduction have been fully fulfilled. Specifically:

- **Architecture Definition:** A layered communication architecture has been established, providing a modular backbone for future CCS applications like the ETCS-OB, ATO-OB and JRU.
- **Interoperability:** By aligning with the proposed TSI 202x standards and utilizing REST/AMQP protocols, this deliverable ensures a unified communication infrastructure across the European railway sector.
- **Functional Validation:** The three core modules—*Working with the Device*, *Working with the Inventory*, and *Working with the Network*—were successfully demonstrated, confirming that the "OCN" network management philosophy is technically feasible and operationally ready.

4.3 REFLECTIONS, LESSONS LEARNT, AND CHALLENGES

The development of the OCN Laboratory Demonstrator provided several critical insights into the convergence of IT and railway OT (Operational Technology):

- **The Power of Microservices:** Moving to a containerized landscape (Podman) significantly reduced the complexity of managing diverse CCS end-devices. This modularity is essential for future-proofing railway systems against hardware obsolescence.
- **Security vs. Complexity:** Implementing IEEE 802.1X for network access control highlighted the necessity of a robust Central Network Inventory as a "Single Source of Truth." Automated identity management is the only way to ensure high security without increasing the manual workload for maintenance teams.
- **Fault Tolerance:** The A/B (Old/New) update strategy proved to be a vital requirement. In a safety-critical environment, the ability to automatically rollback a failed firmware update is a prerequisite for maintaining high fleet availability.
- **Standardization Gaps:** Aligning with the Secure Component Specification while it is still under development required a flexible architectural approach. This highlights the ongoing need for synchronized standardization efforts across the Europe's Rail Joint Undertaking.

4.4 FINAL SYNTHESIS

The work carried out in this deliverable demonstrates that the Onboard Communication Network (OCN) is the critical communication backbone of next generation rolling stock. By replacing proprietary, siloed communication systems with a standardized, secure, and manageable network, we are enabling the next generation of Autonomous Train Operations (ATO) and ETCS applications.

Furthermore, this deliverable serves as a technical and procedural blueprint that extends beyond signalling; the established network management functionalities are designed to support future Train Control and Monitoring System (TCMS) and On-board Multimedia and Telematic Services (OMTS) applications. By leveraging a unified physical network architecture logically partitioned into distinct domains, the OCN provides a scalable and cost-effective framework for cross-system convergence. This approach ensures high levels of security and performance while maintaining the necessary isolation between functional areas, ultimately ensuring that the European railway sector remains at the forefront of global technological innovation.

Outlook: Contribution to Future Railway Standards

The findings, architectures, and methodologies developed within this deliverable are not only immediately applicable to current rolling stock programmes but are also positioned to make a meaningful and lasting contribution to the evolution of European railway standards esp. the TSI CCS. The following outline the key area where this work can and should feed into future standardization efforts, and how this transition can be realized in practice.

Contribution to TSI CCS (Technical Specification for Interoperability – Control Command and Signalling)

The OCN architecture defined in this deliverable directly addresses the communication infrastructure requirements that underpin ETCS and ATO functionality.

Specifically:

- The **logical domain partitioning model**, which separates safety-critical signalling traffic from operational and multimedia data, provides a concrete and implementable reference for future TSI CCS revisions addressing onboard network requirements.
- The **security framework** developed for the OCN — including network access control, traffic isolation, and integrity monitoring — aligns with the cybersecurity obligations increasingly embedded in TSI CCS and the associated ERA guidelines. These can serve as a technical basis for more prescriptive cybersecurity annexes in future TSI editions.
- The **standardized interfaces** between the OCN and ETCS/ATO subsystems reduce vendor dependency and support the interoperability objectives that are central to the TSI CCS mandate. These interface definitions should be proposed as candidate specifications to the relevant ERA (European Union Agency for Railways) working groups.

Realization pathway: Engagement with ERA's TSI CCS revision process — currently evolving towards greater alignment with ATO GoA2+ requirements — should be pursued by submitting this deliverable's architectural findings as technical input documents. Participation in the relevant CEN/CENELEC working groups (e.g., TC9X) provides a formal channel for this contribution.

5 REFERENCES

- [1] I. Arndt, C. Daniel, E. De Paola, M. Kirsche, V. Knollmann, R. Mattes, M. Moritz, K. Rahn, H. Roelle, S. Roullier and S. Schuerch, "D24.1 – Definitive and aligned requirements set for managing an Onboard Communication Network and concluded selection of suitable management functionality," ERJU Innovation Pillar R2DATO, 2025.
- [2] R. Mattes, H. Roelle and S. Roullier, "D23.6 – Homologation concept for Onboard Communication Network (aka One Common Bus)," ERJU Innovation Pillar R2DATO, 2025.
- [3] Z. Ahmed, C. Daniel, E. De Paola, V. Knollmann, M. Moritz, K. Rahn, H. Roelle, S. Roullier, P. Rozijn and S. Schuerch, "D23.1 – List of user stories for the Onboard Communication Network," ERJU Innovation Pillar R2DATO, 2024.
- [4] Z. Ahmed, C. Daniel, E. De Paola, V. Knollmann, M. Moritz, K. Rahn, H. Roelle, S. Roullier, P. Rozijn and S. Schuerch, "D23.2 – Definitive and aligned requirements set for Onboard Communication Network and Basic Services," ERJU Innovation Pillar R2DATO, 2024.