Net-Zero self-adaptive activation of distributed self-resilient augmented services

D6.2 System Integration on the testbeds, Pilot installations and implementations.r1

Lead beneficiary	NOVA	**Lead author**	Ioannis Markopoulos, Angelos Lampropoulos
Reviewers	Jorge Pose, Julio Suárez, Joaquín Escudero (GRAD), Kostas Pournaras, Kostas Lampropoulos (PNET)		
Type	R	**Dissemination**	PU
Document version	V1.0	**Due date**	31/10/2025

Co-funded by the European Union

6GSNS

Project funded by

Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Swiss Confederation

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

UK Research and Innovation

Project information

Project title	Net-Zero self-adaptive activation of distributed self-resilient augmented services
Project acronym	NATWORK
Grant Agreement No	101139285
Type of action	HORIZON JU Research and Innovation Actions
Call	HORIZON-JU-SNS-2023
Topic	HORIZON-JU-SNS-2023-STREAM-B-01-04 Reliable Services and Smart Security
Start date	01/01/2024
Duration	36months

Document information

Associated WP	WP6
Associated task(s)	T6.2, T6.3
Main Author(s)	Ioannis Markopoulos (NOVA), Angelos Lampropoulos (NOVA)
Author(s)	Mohammed Alshawki, Péter Vörös (ELTE), Vincent Lefebvre, Mark Angoustures (TSS), Nasim Nezhadsistani (UZH), Antonios Lalas, Virgilios Passas, Asterios Mpatziakas, Alexandros Papadopoulos, Athanasios Papadakis, Nikolaos Vakakis, Aristeidis Papadopoulos, Eleni Chamou, Anna Maria Pistela, Evangelos V. Kopsacheilis, Nikos Makris, Donatos Stavropoulos, Thanasis Korakis, Anastasios Drosou (CERTH), Maria Safianowska (ISRD), Shankha Gupta, Sumeyya Birtane, Mays AL-Naday (UESSEX), Kostas Pournaras, Kostas Lampropoulos (PNET), Edgardo Montes de Oca, Maxime Vinh Hoa La (MONT), Joachim Schmidt, Leonardo Padial (HES-SO), Tom Goethals (IMEC), Jorge Pose, Julio Suárez (GRAD), Wissem Soussi, Gökcan Cantali, Gürkan Gür (ZHAW), Rana Abu Bakar, Layal Ismail, Marco Lucio Mangiacapre, Francesco Paolucci (CNIT)
Reviewers	Jorge Pose, Julio Suárez, Joaquín Escudero (GRAD), Kostas Pournaras, Kostas Lampropoulos (PNET)
Type	R – Document, Report
Dissemination level	PU – Public
Due date	M22 (31/10/2025)
Submission date	31/10/2025

Document version history

Version	Date	Changes	Contributor (s)
v0.1	18/06/2025	Draft initial document for ToC validation	Ioannis Markopoulos , Angelos Lampropoulos (NOVA), Antonios Lalas (CERTH), all authors
v0.2	17/07/2025	Finalise testbed setup	Testbed Owners
v0.3	04/09/2025	Filalise which component will be installed in which testbed	Use Case Owners
v0.4	24/09/2025	Finalize test scenarios / test cases	Use Case Owners
v0.5	13/10/2025	Deliver final draft	Ioannis Markopoulos, Angelos Lampropoulos (NOVA), all authors
v0.6	17/10/2025	Review of document completed	Jorge Pose, Julio Suárez, Joaquín Escudero (GRAD), Kostas Pournaras, Kostas Lampropoulos (PNET)
v0.7	22/10/2025	Review comments implemented	All authors
v0.8	27/10/2025	Quality review completed	Eryk Schiller, Leonardo Padial (HES-SO)
v0.9	30/10/2025	Final review and refinements	Antonios Lalas, Evangelos V. Kopsacheilis (CERTH)
v1.0	31/10/2025	Document ready for submission	Antonios Lalas (CERTH)

Disclaimer

Copyright message

Contents

List of acronyms and abbreviations

Abbreviation	Description
16-QAM	16-Quadrature Amplitude Modulation
5GC	5G Core
64-QAM	64-Quadrature Amplitude Modulation
AI	Artificial Intelligence
AMF	Access and Mobility Management Function
ARNO	Advanced Research on Networking testbed
AUSF	Authentication Server Function
AV	Autonomous Vehicle
BPSK	Binary Phase Shift Keying
CNF	Container Network Function
CNI	Container Network Interface
CNN	Convolutional Neural Network
CPE	Customer Premises Equipment
CRD	Custom Resource Definition
CSI	Channel State Information
CTI	Cyber Threat Intelligence
DB	Data Base
DFE	Decentralized Feature Extraction
DL	Downlink
DOST	Denial of Sustainability
DPDK	Data Plane Development Kit
E2SM	E2 Service Model
E2SM-CCC	E2SM - Cell Configuration and Control
E2SM-RC	E2SM - Radio Control
eNB	evolved Node B
FDD	Frequency Division Duplex
FR1	Frequency Range 1
gNB	Next-Generation Node B
HTTP	Hyper Text Transport Protocol
KDR	Key Disagreement Rate
KGR	Key Generation Rate
KPI	Key Performance Indicator
MAE	Mean Absolute Error
MEC	Multi access Edge Computing

Abbreviation	Description
MIMO	Multiple Input Multiple Output
ML	Machine Learning
MTD	Moving Target Defense
NB-IoT	Narrow Band Internet of Things
Near-RT RIC	Near- Real Time RAN Intelligent Controller
NFV	Network Function Virtualization
NFVI	NFV Infrastructure
NFVO	NFV Orchestrator
NIST	National Institute of Standards and Technology
NRF	Network Repository Function
NSA	Non-Stand Alone
NSaaS	Network Slice as a Service
NSSF	Network Slice Management Function
OAI	Open Air Interface
OFDM	Orthogonal Frequency Division Multiplexing
OSM	Open-Source Management and Orchestration
Orch	Orchestration
PCF	Policy Control Function
PKG	Physical Key Generation
PRB	Physical Resource Block
QoE	Quality of Experience
QoS	Quality of Service
QPSK	Quadrature Phase Shift Keying
QSFP-DD	Quad Small Form-factor Pluggable Double Density
RAN	Radio Access Network
RB	Resource Block
RBAC	Role-Based Access Control
REST API	Representational State Transfer Application Programming Interface
RF	Radio Frequency
RIS	Reconfigurable Intelligent Surfaces
RNN	Recurrent Neural Network
RU	Radio Unit
SA	Stand Alone
SDK	Software Development Kit
SDN	Software Defined Network

Abbreviation	Description
SDR	Software Defined Radio
SIM	Subscriber Identity Module
SISO	Single Input Single Output
SMF	Session Management Function
SSH	Secure Shell
STFT	Short Time Fourier Transform
SVM	Support Vector Machine
TDD	Time Division Duplex
TEE	Trusted Execution Environment
UDM	Unified Data Management
UE	User Equipment
UHD	USRP Hardware Driver
UL	Uplink
UPF	User Plane Function
USRP	Universal Software Radio Peripheral
V2X	Vehicle to Everything
VM	Virtual Machine
VNF	Virtual Network Function
VPN	Virtual Private Network
WAI	Wirespeed Artificial Intelligence
WASM	WebAssembly

List of figures

List of tables

Executive summary

Deliverable D6.2 “System Integration on the testbeds, Pilot installations and implementations.r1”, outlines the first steps in the validation of the NATWORK system, prior to the pilot trials. Within this deliverable, the infrastructure of the testbeds for NATWORK project is fully described. These testbeds were constructed to host NATWORK components that are derived from the sixteen (16) Use Cases of the project. Several test scenarios have been identified to ensure the technical readiness of the NATWORK system for each individual component.

Furthermore, within this technical report the deployment of the Attack Generation System is thoroughly presented. This Attack Generation System emulates potential attacks on the network and allows verification of the performance of NATWORK modules. Training datasets for the AI models are also described.

Fourteen (14) testbeds are currently employed by the NATWORK project belonging to thirteen (13) partners. These testbeds have been set-up throughout Europe. More specifically, the testbeds for the NATWORK project reside in the following areas (countries): Greece, Spain, Italy, France, Belgium, United Kingdom, Poland, Hungary, Switzerland.

These testbeds, used by the NATWORK project, are controlled environments that evaluate the sixteen (16) Use Cases (UCs) and the related components that have been identified in previous stages of the project. Currently, this test framework evaluates the technical elements of NATWORK components.

The components of the NATWORK project have been defined in D2.3 “Architecture, Interfaces & Specifications”. Additional information for the components was introduced in D6.1 “Definition of the evaluation framework & Pilot specifications”. Currently, forty-three (43) components have been identified from the 16 Use-Cases of the project. Each component determines the related NATWORK service or services and illustrates an element(s) of the system. For mature components, specific test scenarios have been identified and demonstrated within the report. Each test scenario covers a specific functionality of the component in question.

Dry run tests have been performed for specific (mature) components of NATWORK. These tests are preliminary and indicate the initial behavior of the system. The dry run test results are presented in this report. Most components have been installed on a dedicated testbed. In case that a component is set up in more than one testbed, the test results are reported in one single report per component. Through D6.3, the second version of “System Integration on the testbeds, Pilot installations and implementations” report, the second phase of dry run tests of the components will be documented. In that second report, all components, including components that currently are in their early stage, will be validated. Moreover, use case trials and demos will be performed as part of T6.3 “Use Cases Trials and Demonstration” and the relevant outcomes will also be reported in D6.3 in M32.

As a final step, several potential attack scenarios have been formulated for NATWORK. These attacks have identified vulnerabilities of the services and components of the NATWORK system. By identifying these vulnerabilities, the security and overall performance of NATWORK components can be improved. The attacks have been triggered through the Attack Generation System, which is a tool that processes potential attacks to a given system. These attacks and the corresponding response of NATWORK are also reported within the current deliverable.

1. Introduction

The main goal of T6.2 “Testbed integration & attack generation system.r1” is to prepare NATWORK for the forthcoming use case trials of the system and the validation and evaluation of the NATWORK framework while T6.3 “Use Cases Trials and Demonstration” focuses on the implementation/execution and reporting of these use case trials and demonstrations, taking into account end-to-end use case requirements and architecture implementation (WP2) for the beyond 5G/6G security framework envisioned in NATWORK. To accomplish these goals, several activities have been performed in this period and reported in D6.2 “System Integration on the testbeds, Pilot installations and implementations.r1”. More specifically, the set-up of the NATWORK components into the testbeds is thoroughly described. For the components installed, dry run tests have been performed to indicate that the components are ready for use for the next stages of the project. The actual test scenarios and the results of these tests are also presented within the present report. Finally, several attacks were identified and triggered through the Attack Generation System. At the current stage, the response of each individual Use Case to these attacks has been also reported in D6.2.

This specific deliverable covers the initial assessment of the validation of NATWORK components. In the first version, D6.2 focuses on the successful installation of the 43 components of the NATWORK project. Moreover, this deliverable identifies the results of the dry run tests of mature components. At M32, the second version of this report, D6.3, will be submitted having the dry run test results for all components. In this second report, the interworking of the developed architectural elements will be assessed. The final definition and set-up of the use-case environments, the initial set of generated results and findings from T6.3 “Use Cases Trials and Demonstration” will be also presented in D6.3.

1.1. Purpose and structure of the document

D6.2 “System Integration on the testbeds, Pilot installations and implementations.r1” focuses on the interconnection of the NATWORK components– currently 43, and the validation of the technical readiness of the system. More precisely, this deliverable includes a detailed report for each individual testbed. In addition, this deliverable highlights the successful installation of the NATWORK components onto the testbeds and verifies that the components are up and running. These verifications are performed through specific Test Scenarios and related Test Cases that are also presented in this document. Moreover, relevant security attacks have been triggered against the NATWORK components. The response to these attacks is demonstrated in the report.

The sections of this document can be summarized as follows:

- **Section 2 – Testbed Environments**: Section 2 describes the 14 testbeds within NATWORK and the related components that have been installed in the testbeds.
- **Section 3 – Dry Run Tests for NATWORK Components**: This part of the deliverable identifies the process of the dry run tests for each NATWORK component. Moreover, the dry run test results or the period in which the dry run tests will be performed are presented (see Appendix).
- **Section 4 – Attacks**: Throughout this section, multiple types of attacks towards NATWORK ecosystem are presented and the related response to these attacks are depicted.
- **Section 5 – Conclusions**: This section discusses the key elements of the document, i.e. the set-ups of the Testbeds, the individual tests (Dry Run Testing) of the components or group of components, and the related attacks that the NATWORK ecosystem could, in certain conditions, face.
- **Appendix – Test Scenarios of Components**: This appendix demonstrates the Test Scenarios for the matured components of NATWORK project. Within each scenario, the period in which the dry run test has been or will be conducted is indicated. Test scenarios that have been run also contain their corresponding results.

1.2. Intended Audience

The NATWORK Project's D6.2 "System Integration on the testbeds, Pilot installations and implementations.r1" is devised for public use. This deliverable focuses on the initial assessment of NATWORK through the testbeds that have been set up by the partners. Within the testbeds, the related NATWORK components have been installed for preliminary tests. In addition, several attacks were identified towards the NATWORK system. These attacks are thoroughly described in the current deliverable. The information in this report is an integral part of WP6 activities and activities of other WPs. Furthermore, this report can be beneficial to an audience that is concerned about cybersecurity activities in general.

1.3. Interrelations

The NATWORK consortium integrates a multidisciplinary spectrum of competencies and resources from academia, industry, and research sectors, focusing on user-centric service development, robust economic and business models, cutting-edge cybersecurity, seamless interoperability, and comprehensive on-demand services. The project integrates a collaboration of fifteen partners from ten EU member states and associated countries (UK and CH), ensuring a broad representation for addressing security requirements of emerging 6G Smart Networks and Services in Europe and beyond.

NATWORK is categorized as a "Research Innovation Action - RIA" project and is methodically segmented into 7 WPs, further subdivided into tasks. With partners contributing to multiple activities across various WPs, the structure ensures clarity in responsibilities and optimizes

communication amongst the consortium partners, boards, and committees. The interrelation framework within NATWORK offers smooth operation and collaborative innovation across the consortium, ensuring the interconnection of the diverse expertise from the various entities (i.e., Research Institutes, Universities, SMEs, and Large Industries) enabling scientific, technological, and security advancements in the realm of 6G.

D6.2 “System Integration on the testbeds, Pilot installations and implementations.r1” is directly associated with T6.2 “Testbed integration & attack generation system” and T6.3 “Use Cases Trials and Demonstration”. In D6.3, the second version of “System Integration on the testbeds, Pilot installations and implementations” report, both T6.2 and T6.3 will again report, but with the focus being the final definition, set-up and integration of the use-case environments, the generated trial results and findings. In addition, this deliverable acts as an interconnection between T6.1 that has been concluded at M18 and T6.4 that will be completed after T6.2 and T6.3. Furthermore, D6.2 is related with WP2 as the related Use-Cases and relevant Architecture were established in it, as well as WP3, WP4 and WP5 as the related components were defined through these WPs.

2. Testbed Environments

This section describes the testbeds that NATWORK project uses to verify its components and services in a preliminary stage. These components have been determined previously, through deliverable D2.3 “Architecture, Interfaces & Specifications” in M12. This deliverable depicted the initial version of the architecture of the NATWORK system and the fundamental components that constitute the overall functionalities that NATWORK provides. In addition, from D6.1 “Definition of the evaluation framework & Pilot specifications” that was submitted in M18, it was identified further how the project will demonstrate the performance, security, and sustainability of its proposed solution.

The testbeds examined thoroughly in this section will evaluate the 16 Use Cases and the related components that have been established in previous stages of the project. The main goal is to verify that the NATWORK components are up and running. Additional actions, including integration activities, will be performed at later stages of the project.

In the table below, all of the testbeds that have been used for NATWORK project are depicted. So far, fourteen (14) testbeds have been set-up from thirteen (13) partners. The main aim of using these testbeds is to verify the readiness of NATWORK components. These testbeds are spread throughout EU and more specifically in 9 countries. It should be noted that some components have been installed in more than one testbed. More information can be found in the table below, where each individual testbed including the name of the testbed, the partner responsible for the testbed and the component(s) that have been installed in each testbed are reported:

Table 1: List of testbeds and related components for NATWORK project

#	Testbed	Partner	Component(s)
1	5G Testbed	CERTH	• AI-Based RIS Configuration Component • ML-based MIMO Component • JASMIN & Filter Mitigation Component • Multimodal Fusion Approach for Intrusion Detection System for DoS Attacks Component • Lightweight SDN-based AI-enabled Intrusion Detection System for Cloud-based Services Component • AI-enabled DoS Attack Component

#	Testbed	Partner	Component(s)
			• Multiagent AI based cybersecurity support system • Microservice Behavioral Analysis for Detecting Malicious Actions Component • Security-performance balancer Component
2	NITOS Testbed	CERTH	• Slice Orchestration and Slice Management for beyond 5G Networks Component • Wirespeed traffic analysis in the 5G transport network
3	5GLab	GRAD	• DetAction: Detection and reAction Against Jamming Attacks Component • Characteristics Extractor • Key Generator • Security Evaluator
4	ARNO Testbed	CNIT	• AI-driven Security Monitoring for Anomaly Detection and Root Cause Analysis in IoT Networks Component • Wire-speed AI (WAI) and Decentralized Feature Extraction (DFE) Component • DFE Telemetry Component
5	5G-IoT Testbed	MONT	• AI-driven Security Monitoring for Anomaly Detection and Root Cause Analysis Component • CIA-hardening of x86 payloads Component • CIA-hardening of WASM payloads Component
6	CloudNativeLab	IMEC	• TrustEdge Service • Feather Component • Flocky Component
7	Patras5G-PNET Testbed	PNET	• MTD Controller Component • MTD Strategy Optimizer Component • MTD Explainer Component • MTDFed Component

#	Testbed	Partner	Component(s)
			• AI-driven Security Monitoring for Anomaly Detection and Root Cause Analysis Component
8	NCL	UESSEX	• Energy efficient over edge-cloud Component • Secure-by-design Orchestration Component • Security-compliant Slice Management Component • Federated Learning for Edge-to-cloud Component
9	TSS Testbed	TSS	• CIA-hardening of x86 payloads Component • CIA-hardening of containerized payloads Component • CIA-hardening of WASM payloads Component
10	ISRD Testbed	ISRD	• JDM-xApp Component • Liquid RAN Component • Liquid Near-RT RIC Component • KPM xApp Component
11	ELTE Testbed	ELTE	• End-to-End Security Management Component • Data plane ML Component • Secure Data Aggregation Component
12	ZHAW Testbed	ZHAW	• MTD Controller Component • MTD Strategy Optimizer Component • MTD Explainer Component • MTDFed Component
13	HES-SO Testbed	HES-SO	• Detection against jamming attacks Component • Setting up of a Mirai botnet • FPGA-based hardware detection of DDoS attacks.
14	UZH Testbed	UZH	• Anomaly Detection Explainer Component

In the following sub-sections, the infrastructure of the individual testbeds, as well as information on how the related components are installed in the testbeds are described.

2.1. 5G Testbed

The 5G Testbed at CERTH comprises two main parts: one dedicated to signal processing for the development and evaluation of services and components at the lower communication layers, and another based on Software Defined Networking (SDN), focusing on upper layers and service-level functionalities. Both parts are described in detail below.

2.1.1. 5G-Signal Processing Testbed Infrastructure

The CERTH 5G Signal Processing testbed integrates all components required to build a network in the FR1 5G frequency bands. It is designed to support experimentation-driven research in both wired and wireless communication networks. The testbed consists of modular components that can be flexibly combined to address diverse scenarios, enabling extensive experimentation and solution evaluation. During the project, the testbed will be expanded with additional hardware components, which are also described in this section.

Software Defined Radios: The testbed includes five USRP B210 SDRs, which enable experimentation across a wide range of scenarios involving multiple base stations, as well as both legitimate and malicious users such as jammers or eavesdroppers. Each USRP B210 operates over a frequency range of 70 MHz to 6 GHz and supports two transmit and two receive chains, allowing 2×2 MIMO operation. The devices provide up to 56 MHz of instantaneous bandwidth through the transceiver front-end and are powered by a Spartan-6 XC6SLX150 FPGA. Connectivity is ensured via USB 3.0 (SuperSpeed), with host integration supported through the UHD driver and GNU Radio.

Processing Units: Testbed includes three NVIDIA Jetson Nano modules as lightweight edge nodes. Each Nano integrates a 128-core Maxwell GPU, a quad-core ARM Cortex-A57 CPU, 4 GB LPDDR4 (25.6 GB/s), making them suitable for on-device DSP, spectrum sensing, and distributed inference close to the radios; camera-centric pipelines and Linux/JetPack support align with our GNU Radio toolchain.

For the heavier workloads, there are three NVIDIA Jetson AGX Orin units provide an Ampere GPU with 2,048 CUDA cores and 64 Tensor Cores alongside a 12-core Arm Cortex-A78AE CPU—for accelerated PHY/MAC processing, neural receivers, and real-time beam/radar inference. This keeps SDR pipelines GPU-offloaded while remaining in a Linux/JetPack environment compatible with GNU Radio.

RIS units: The RIS hardware integrated in the testbed is the TMYTEK XRifle Dynamic RIS designed for operation in the sub-6 GHz 5G band. It consists of a 16x16 PIN-diode array with binary phase control—the most constrained configuration, posing significant challenges for multiplexing schemes. The device covers 4.2–5.2 GHz, supports linear polarization, and provides

incidence/reflection steering capabilities from -60° to +60° along both vertical and horizontal planes.

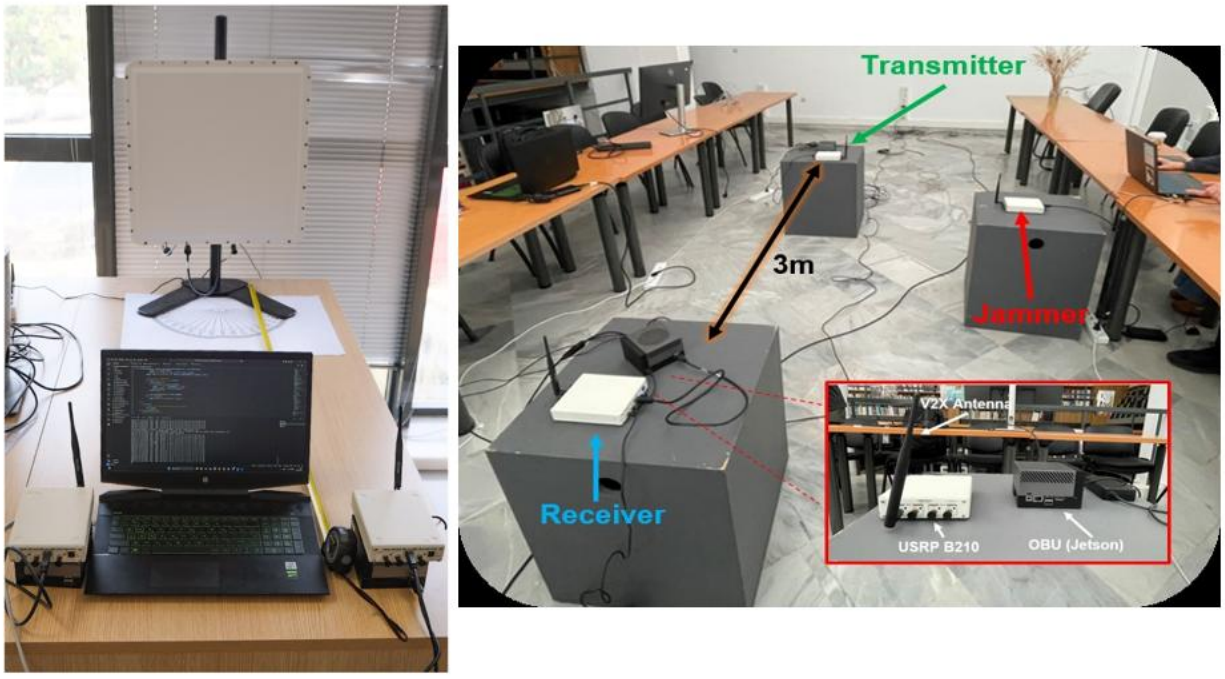

Figure 1: CERTH 5G Signal Processing Testbed

Supplementary equipment: The testbed is further equipped with periodic antennas, WiFi antennas, RF splitters, and amplifiers, which extend its flexibility for diverse wireless experiments. Omnidirectional antennas allow baseline coverage and broadcast scenarios, while WiFi antennas support integration with commodity devices and benchmarking against standard WLAN technologies. RF splitters and amplifiers provide precise power control, distribution, and link-budget adaptation across the different SDR front-ends. In the upcoming period, the setup will be expanded with horn antennas and MIMO-capable arrays. The horn antennas will enable high-gain, highly directional measurements, particularly useful for RIS characterization, angular selectivity studies, and interference control. The addition of MIMO antenna arrays will unlock spatial multiplexing experiments, enabling realistic evaluations of the project outcomes.

2.1.1.1. AI-Based RIS Configuration

For the evaluation of this service, the parts of the testbed that will be used are the following:

- USRPs B210
- RIS unit
- Horn antennas for reception and transmission
- The processing units

2.1.1.2. ML-based MIMO

For the evaluation of this service, the parts of the testbed that will be used are the following:

- USRPs B210
- MIMO antennas
- The processing units

2.1.1.3. JASMIN & Filter Mitigation

- USRPs B210
- MIMO, WiFi, periodic antennas
- The processing units
- Amplifiers

2.1.2. Available cloud and edge resources

The testbed integrates **Cloud and Edge Computing resources** to support experimentation and innovation in distributed infrastructures. It provides a flexible environment with **Kubernetes and Docker** for containerized applications, **OpenStack** for cloud orchestration, **Open Source MANO (OSM)** for network function management, and **virtual machine (VM) capabilities** for legacy and hybrid workloads. This combination enables researchers and developers to design, deploy, and evaluate advanced cloud-native and edge-native services in a realistic, scalable, and interoperable setting.

2.1.3. 5G-Core and SDN related resources

The testbed leverages **Software-Defined Networking (SDN) and 5G technologies** to enable experimentation with next-generation communication infrastructures. It integrates the **OAI 5G Core Network project** for 5G core functionalities, **O-RAN** for open and interoperable radio access, and **OpenDaylight** as an SDN controller to manage and orchestrate programmable networks. Together, these components create a flexible, standards-based environment for designing, testing, and validating 5G services, network slicing, and advanced edge-to-cloud use cases in both research and pre-deployment scenarios.

Building on this foundation, this testbed incorporates a modular microservices-based architecture that supports dynamic scaling, automated lifecycle management, and performance monitoring. AI-driven behavioral analysis models process real-time data to detect anomalies such as DoS attempts, triggering automated responses through the Floodlight SDN controller. This integrated setup provides a realistic platform for validating threat detection accuracy, evaluating mitigation effectiveness, and assessing overall system resilience under controlled attack simulations in next-generation 5G microservice ecosystems.

2.1.4. 5G-SDN Testbed Components Set-Up

2.1.4.1. Multimodal Fusion Approach for Intrusion Detection System for DoS Attacks

Towards the deployment of this component at the 5G-SDN Testbed, the Cloud and Edge Resources and the 5G-Core and SDN related resources will be used.

2.1.4.2. Lightweight SDN-based AI-enabled Intrusion Detection System for Cloud-based Services

Towards the deployment of this component at the 5G-SDN Testbed, the Cloud and Edge Resources and the 5G-Core and SDN related resources will be used.

2.1.4.3. AI-enabled DoS Attack

Towards the deployment of this component at the 5G-SDN Testbed, the Cloud and Edge Resources and the 5G-Core and SDN related resources will be used.

2.1.4.4. Microservice Behavioral Analysis for Detecting Malicious Actions

Towards the deployment of this component at the 5G-SDN Testbed, the Cloud and Edge Resources and the 5G-Core and SDN related resources will be used.

2.1.4.5. Security-performance balancer

Separating user traffic to different servers based on ciphering, integrity, and replay protection algorithms is essential for optimizing performance and enabling efficient use of cryptographic acceleration. Different algorithms (Snow, AES, ZUC) impose varying computational loads on the CPU and hardware accelerators. By directing users with similar algorithmic demands to specific servers by Security Performance Balancer, the network can reduce context-switching overhead and better align traffic with hardware capabilities, such as dedicated crypto engines and accelerators. This minimizes latency, prevents CPU bottlenecks, and ensures consistent throughput, especially under high-load conditions. It also allows tailored tuning of server configurations to match the expected algorithm's workload, improving processing efficiency and overall user experience.

2.1.4.6. Multiagent AI based cybersecurity support system

Towards the deployment of this component at the 5G-SDN Testbed, the Cloud and Edge Resources and the 5G-Core and SDN related resources will be used.

2.2. NITOS Testbed Components Set-Up

The NITOS Testbed Facility offers a comprehensive set of technologies and capabilities that support advanced experimentation. This section provides a detailed description of the testbed

infrastructure, along with the configuration and deployment procedures required for hosting the NATWORK components within the NITOS environment.

2.2.1. NITOS Testbed Infrastructure

NITOS Future Internet Facility is a state-of-the-art, integrated research infrastructure comprising multiple heterogeneous testbeds. It is dedicated to supporting experimentation-driven research in the field of wired and wireless communication networks. The facility is remotely accessible and available to the global research community 24/7. To date, it has been utilized by hundreds of researchers worldwide.

Figure 2: NITOS Testbed

Wireless Experimentation Platform: NITOS provides a highly versatile wireless networking testbed, enabling researchers to conduct real-world experiments across a spectrum of radio technologies and deployment scenarios. The platform features a mix of stationary and mobile nodes, each equipped with multiple wireless interfaces that support technologies such as 5G, Wi-Fi, WiGig, and others. These nodes are deployed across diverse environments, from controlled indoor labs with external RF isolation to complex outdoor spaces where interference and mobility reflect real-world conditions. This configuration allows experimentation at multiple layers of the protocol stack—from physical layer customization to application-layer performance testing—facilitating research in multi-radio access technologies (multi-RAT), spectrum sharing, and next-generation wireless systems. The testbed supports repeatable trials, multi-hop mesh setups, mobility patterns, and heterogeneous interface coexistence, making it ideal for both academic and industrial research.

Cloud and Edge Computing Resources: To support distributed systems and networked applications, the NITOS facility includes a powerful computing cluster composed of multi-core, high-memory servers with substantial storage capacity and high-throughput network interconnects. This infrastructure enables experimentation with a wide range of cloud and edge computing paradigms. The platform supports virtualization (VMs), container orchestration (e.g., Kubernetes, Docker), and bare-metal deployment, allowing researchers to explore topics such as microservice-based architectures, resource offloading, network slicing, NFV, and service chaining. Fast internal and external connectivity facilitates integration with other testbed components—such as wireless or IoT infrastructures—enabling end-to-end experimentation across heterogeneous layers.

Software-Defined Radio (SDR) Capabilities: For researchers interested in custom wireless protocol development and physical layer innovation, NITOS offers access to a rich set of high-end software-defined radio platforms. These SDR devices are fully integrated with the testbed's compute and wireless infrastructure, enabling both isolated lab-scale experiments and over-the-air trials. The available SDRs support advanced configurations such as 4×4 MIMO, channel bandwidths of up to 100 MHz, and high-frequency tunability. Combined with flexible software stacks (e.g., GNU Radio, srsRAN, OAI), they allow full-stack experimentation, from waveform generation and channel modeling to real-time signal processing and PHY/MAC protocol design. The SDR testbed also supports research in spectrum sensing, dynamic access control, and cross-layer optimization.

Programmable Networking (SDN): The NITOS infrastructure is equipped with a fully programmable networking environment that supports software-defined networking across both wired and wireless domains. Utilizing SDN-capable switches and programmable forwarding engines, researchers can design and deploy custom network control policies, routing strategies, and traffic engineering solutions. The testbed supports OpenFlow and advanced data plane programmability via the P4 language, enabling low-level control over packet processing pipelines. This allows for experimentation with novel network functions, telemetry, intent-based networking, and integration with edge computing and IoT environments. The SDN infrastructure is ideal for exploring topics such as network slicing, service function chaining, and security-aware routing in programmable, multi-domain environments.

2.2.2. NITOS Testbed Components Set-Up

2.2.2.1. Slice Orchestration and Slice Management for beyond 5G Networks Component

Towards the deployment of this component at the NITOS Testbed, the Cloud and Edge Computing Resources, the Wireless Experimentation Platform and the SDR Resources will be utilized. More specifically, the following resources will be employed:

- Kubernetes cluster hosted on server
- OAI for RAN and Core Network
- Nodes with 5G connectivity (UEs)

2.2.2.2. Wirespeed traffic analysis in the 5G transport network

Towards the deployment of this component at the NITOS Testbed, the Cloud and Edge Computing Resources, the Wireless Experimentation Platform and the SDN Resources will be utilized. More specifically, the following resources will be employed:

- Netronome Agilio SmartNIC 25Gbps
- Kubernetes cluster hosted on server
- OAI for RAN and Core Network
- GPU for the training the Machine Learning (ML) model

2.3. 5GLab Testbed Components Set-Up

This section presents two independent tracks deployed at Gradiant 5GLab. The first is for AI-based anti-jamming, which runs on the 5G RAN infrastructure and the second is the PKG stack for a sub-THz indoor link.

2.3.1. 5GLab Testbed Infrastructure

The AI-based anti-jamming testbed is designed for detecting and mitigating jamming attacks in a 5G indoor laboratory environment, where the UE, jammer, and gNB are located within 10 meters. The system operates in the n78 band (3.5 GHz, FR1), using USRP B210 devices both to generate the jamming signals and to capture IQ samples of the gNB's received signal.

The jammer transmits software-generated chirp signals (chosen to maximize power efficiency) with configurable parameters such as transmit gain, sweep period and bandwidth. Transmission is handled through the UHD framework.

The gNB runs on BubbleRAN, which provides an SDK to develop xApps for the Near-RT RIC, in this case focused on PRB scheduling. The captured signals are preprocessed to identify the PRBs

affected by jamming. The results of this detection stage are then communicated to the scheduling xApp via a REST API, where they serve as the input parameters for adaptive scheduling decision.

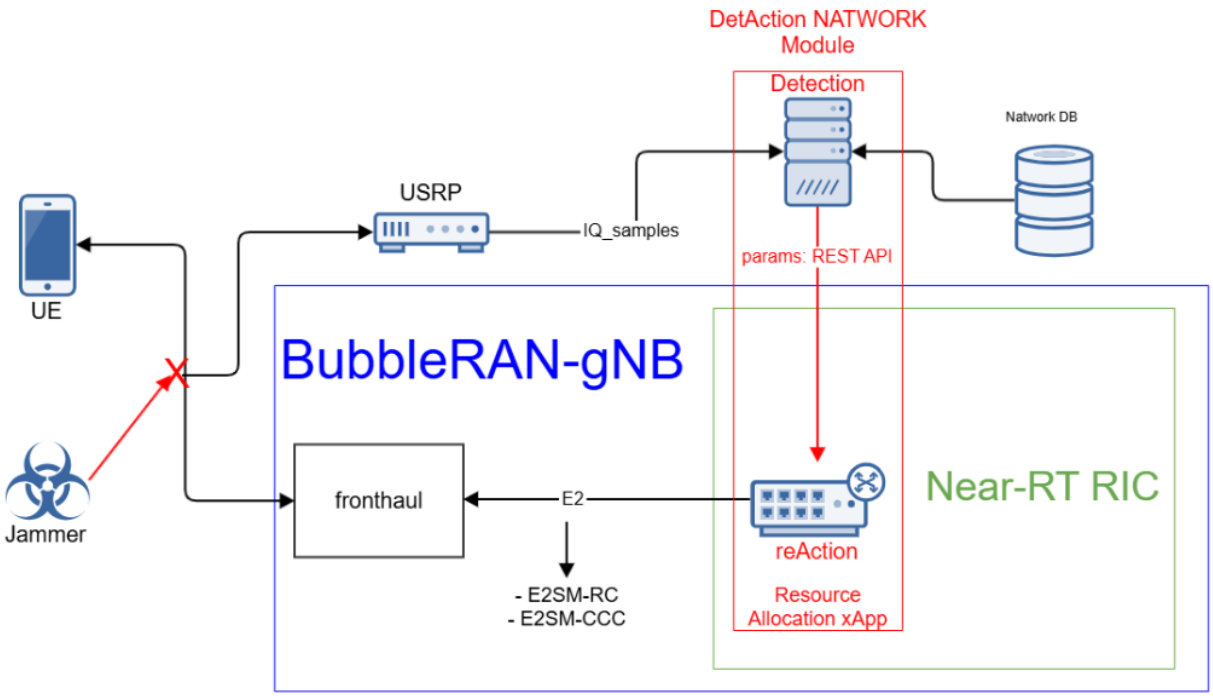

Figure 3: AI-based anti-jamming testbed

The PKG testbed targets OFDM-based physical layer key generation for sub-THz bands in an indoor laboratory environment with separation between Tx and Rx around 10 meters. It consists of three SDR nodes (Alice, Bob & Eve) using USRP B210 devices with specific sub-THz antennas centered at 92.45 GHz on TDD. USRP B210 units provide baseband acquisition and streaming of raw IQ, while external sub-THz up/down-conversion front ends perform the translation to and from 92.45 GHz and handle RF filtering and gain control. Waveform generation, clocking, and data acquisition run on the host via GNU Radio. Channel estimation and feature extraction are executed in software (PC/server with MATLAB) and feed the PKG pipeline: quantization, information reconciliation, and privacy amplification. For learning-based UL/DL prediction, inference is performed by a pre-trained model on a dedicated server connected to the SDR node.

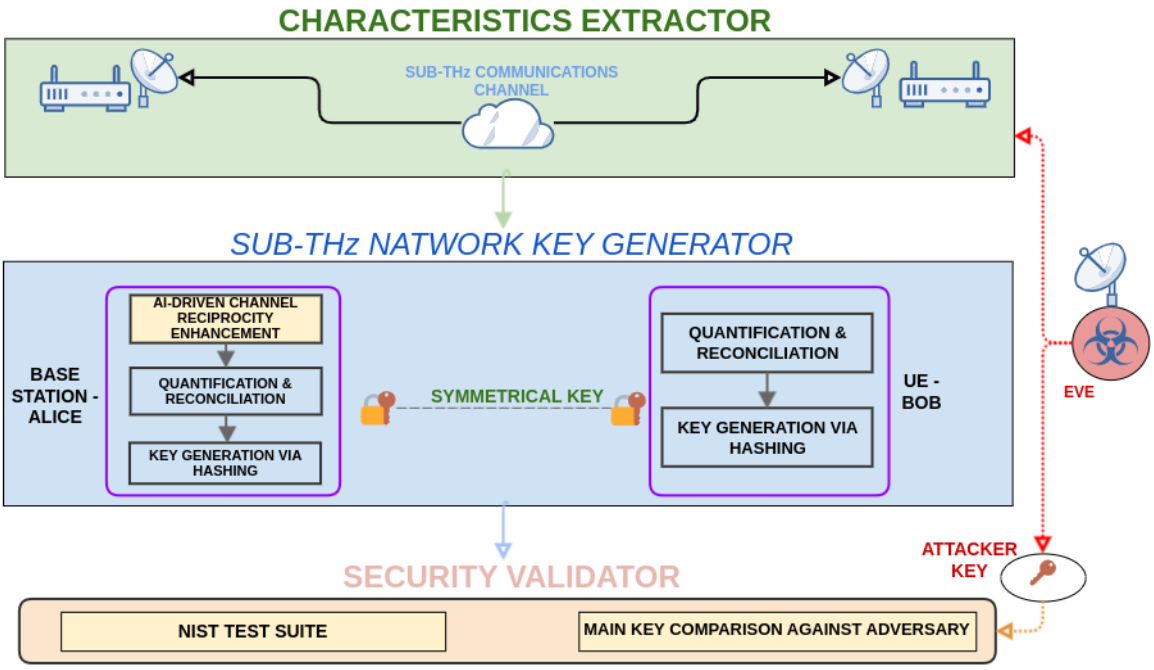

Figure 4: PKG infrastructure-component testbed

2.3.2. 5GLab Testbed Components Set-Up

In this case, DetAction component relies on AI-based anti-jamming testbed infrastructure, meanwhile the following three components belong to the PKG testbed structure.

2.3.2.1. DetAction: Detection and reAction Against Jamming Attacks

The role of this component is to receive, preprocess, localize in the spectrum, and mitigate jamming attacks. It takes IQ samples from a USRP capturing the signal, resamples them to a target data rate, and applies an STFT with a fixed-length window to transform them into the frequency domain. From this representation, spectrum fragments corresponding to 5G RB frequencies are extracted and normalized. These fragments are then evaluated by a CNN to detect whether jamming is present. Finally, the mapping of jammed versus non-jammed RBs is provided to the scheduling xApp, which uses this information to avoid the jammed spectrum while attempting to maintain QoS for the UEs.

2.3.2.2. Characteristics Extractor

This component converts raw IQ into channel characteristics for downstream PKG analytics. It performs synchronization, channel estimation, and feature computation (CSI and related statistics) on data acquired from the sub-THz indoor link. The service generates feature vectors with timestamps and quality flags, finally saves the results in an experiment database. The resulting clean feature sets feed the Key Generation Service and provide channel indicators for

PKG evaluation. Metrics for the adversary (Eve) are also extracted here, ensuring variability across positions and conditions to assess security in the Security Validation component.

2.3.2.3. Key Generator

This component derives a shared symmetric key from reciprocal channel measurements over the sub-THz link. It ingests features from the extractor and executes the PKG pipeline: UL/DL neural network model, followed by quantization, information reconciliation, and privacy amplification. The AI block can enhance reciprocity using measurements at Alice. The operating scope targets indoor trials with a Tx–Rx separation around 10m, with Bob moving to create a dynamic channel. Key results include Key Generation Rate (KGR), Key Disagreement Rate (KDR) for the main link, reconciliation performance, and end-to-end latency.

2.3.2.4. Security Evaluator

This component provides independent verification of key strength and system robustness. It runs statistical tests (NIST test suite), tracks mismatch rates and compares the Alice/Bob keys against an adversary (Eve) baseline. It reports experiment results for the whole PKG pipeline and evaluates them against baseline values of randomness quality, KGR/KDR targets (KPIs), and reproducibility across scenarios.

2.4. ARNO Testbed

2.4.1. ARNO Testbed Infrastructure

The Advanced Research on NetwOrking (ARNO) testbed is a modular and continuously evolving experimental platform that spans the full nick Telco and IT continuum: access, metro, and core networks, as well as edge and cloud domains. Initially conceived for optical networking research, ARNO has matured into a reference infrastructure for programmable networking and in-network intelligence. A central strength of ARNO lies in its programmable hardware ecosystem, which enables researchers to explore fine-grained control, acceleration, and telemetry directly in the data plane. The testbed integrates P4-programmable 100G switches, including the Intel Barefoot Tofino 1, offering line-rate programmability for flexible packet processing and advanced telemetry functions. Complementing these, ARNO features a broad set of SmartNICs and DPUs, such as NVIDIA BlueField-2 and BlueField-3, as well as converged BlueField-2 DPUs with embedded GPUs. These platforms allow the offloading of networking, storage, and security services from CPUs to programmable accelerators, supporting advanced use cases in in-network computing, hardware-assisted orchestration, and AI-driven decision-making at the edge. Additional programmable platforms, including NetFPGA SUME and Xilinx Alveo boards, further extend experimentation into FPGA-based packet processing. ARNO's network core couples this programmable environment with a metro optical infrastructure (ROADMs, packet-optical nodes, coherent transponders, and pluggable optics up to 400 Gbps) and a Calix E7 PON for the access

segment. Two Edgecore switches with Quad Small Form-factor Pluggable Double Density (QSPF-DD) interfaces provide support for 400G ZR/ZR+ and 400G/100G XR pluggable optics, programmable via CMIS specifications.

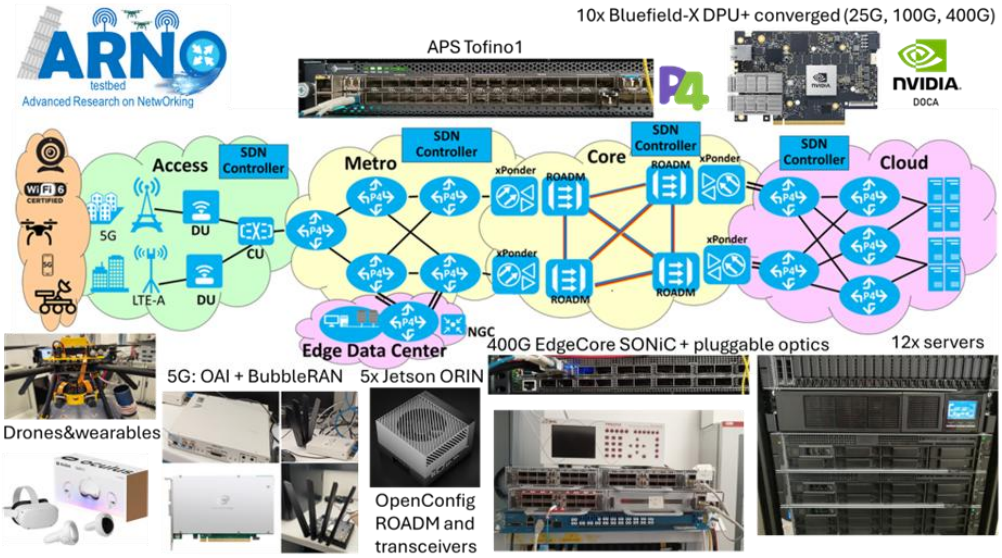

Figure 5: ARNO testbed

For computing, ARNO provides DELL PowerEdge and HP servers equipped with NVIDIA Tesla, V100, A16, and T4 GPUs, FPGAs, and Intel PAC cards, enabling high-performance AI/ML workloads tightly coupled with programmable networking. This synergy allows end-to-end validation of architectures spanning from programmable data planes (Tofino, BlueField, NetFPGA) to cloud-edge orchestration. On the wireless side, ARNO includes an SDN-controlled 5G/6G segment with SDRs (Ettus X310, B210, N310), Quectel development kits, NTN modules, and support for multiple gNB splits (option 2 and 7-1). The testbed runs open and customizable 5G/6G software stacks (OAI, srsRAN), tightly integrated with RAN Intelligent Controllers (RICs) and hardware acceleration for latency-sensitive functions. COTS solutions, such as the BubbleRAN MPX, further complement this environment.

Beyond connectivity, ARNO supports vertical applications through a robotics and XR infrastructure: multi-payload drones, 4WD rovers, Meta Quest 3 and Oculus VR headsets, smart glasses, and humanoid robots enable immersive and mobile scenarios for verticals such as remote driving, training, and telepresence. Traffic generation and monitoring is ensured by Spirent, VIAVI, and software tools like Cisco TRex, enabling experiments up to 400 Gbps line rate.

Finally, ARNO supports federated and secure experimentation through OpenVPN, GRE, IPsec, and BGP tunneling, with proven interoperability across multiple European testbeds.

2.4.2. ARNO Testbed Components Set-Up

2.4.2.1. Wire-speed AI (WAI) and Decentralized Feature Extraction (DFE)

The DFE/WAI components run as security functions inside programmable devices at the data plane of the network (core network and data net network connected to 6G). The main backends considered for setting up in ARNO are the following:

- Bluefield-2 and/or Bluefield-3 SmartNIC encompassing DOCA Flow acceleration capabilities.
- Barefoot Tofino 1 switch.

In the former case, at least a couple of Bluefield-2 are deployed in a packet/optical network encompassing 40/100/200/400 Gigabit Ethernet links. Each device is attested to a hosting server (i.e. Dell PowerEdge R760 or HPE Proliant DL380 Gen11), capable of providing single or dual port traffic capacity up to 100Gb/s or 200Gb/s, depending on the type of card.

The deployment of DFE/WAI functions is done using the OFA Agent container, that may reside either in external controllers, in the hosting server or in the DPU user space.

2.4.2.2. DFE Telemetry

The DFE Telemetry is a network function component deployable in P4 switches, able to extract selected features from selected flows and send them in the form of a Telemetry Report to selected analysis collectors. The setup is envisioned as a P4 software deployment within hardware backends. Software backends are the BMv2 and NIKSS switch utilizing e-BPF technology. The deployment is performed through P4 code instantiation inside the backend, through manual or orchestration-triggered via OFA. At runtime, all P4 flow entry configurations are possible that enable the activation/deactivation of a new telemetry stream, selecting the destination collector. Furthermore, at runtime, all feature selection are configurable dynamically via dedicated flow rule commands.

2.4.2.3. AI-driven Security Monitoring for Anomaly Detection and Root Cause Analysis

The AI-driven Security Monitoring for Anomaly Detection and Root Cause Analysis (RCA) component, developed by MONT, will be adapted and deployed within CNIT's ARNO testbed to leverage its programmable hardware and high-performance AI capabilities. This component aims to provide real-time detection of anomalies and advanced RCA in complex IoT and 6G environments.

The ARNO testbed offers an environment with P4-programmable switches (Intel Tofino), SmartNICs/DPUs (NVIDIA BlueField-2/3), and FPGA accelerators. The MMT probe and Montimage AI Platform (MAIP) will integrate with these programmable data-plane elements to

collect, process, and analyze traffic features in real time. Through in-network telemetry provided by ARNO's DFE/WAI components, the AI-driven monitoring system will ingest fine-grained flow data for anomaly detection. Using advanced ML algorithms (CNNs, reinforcement learning, XAI modules), the component will identify deviations in IoT traffic patterns indicative of DDoS attacks or misconfigurations. RCA functionalities will further correlate anomalies with root causes, such as specific flows, devices, or misbehaving services, and provide explainable outputs to operators.

The deployment will exploit both edge and core domains of ARNO:

- At the data plane, lightweight feature extraction will be performed directly on P4 switches or BlueField SmartNICs.
- At the AI/ML backends, Dell and HPE servers equipped with GPUs (Tesla V100, T4, A16) will run the MAIP models, allowing accelerated inference and scalable training.
- The RCA visualization will be integrated into the ARNO orchestration tools and dashboards, offering explainable insights for operators.

The deployment in ARNO will validate the ability of MONT's component to operate at line-rate monitoring speeds (up to 100–200 Gbps), with low-latency anomaly detection and explainable RCA. This will demonstrate how AI-driven monitoring can be tightly coupled with programmable infrastructures in 6G contexts, ensuring scalability, transparency, and trustworthiness.

2.5. Montimage 5G-IoT Testbed

The 5G-IoT testbed deployed by MONT serves as a dedicated environment to evaluate, integrate, and validate advanced security monitoring solutions in realistic IoT and 5G/6G scenarios. The testbed is designed to support experimentation with high volumes of heterogeneous traffic and provides the infrastructure to simulate IoT devices, gateways, and services under diverse operational conditions, alongside real devices. It enables the deployment of monitoring and analysis components, such as the Montimage Monitoring Tool (MMT) and the Montimage AI Platform (MAIP), ensuring that anomaly detection, root cause analysis, and mitigation strategies can be tested in a controlled yet realistic setting. The following subsections describe the underlying testbed infrastructure and the components set-up, highlighting how they jointly create a flexible and programmable platform for the validation of NATWORK technologies.

2.5.1. Montimage 5G-IoT Testbed Infrastructure

As presented in Figure 6, the 5G-IoT testbed of Montimage constitutes a controlled experimental environment designed to validate anomaly detection, monitoring, and automated response mechanisms in next-generation IoT networks. Its configuration, combining 5G Core functions, IoT devices, and the Montimage Monitoring Tool (MMT), directly supports the objectives of UC#3.1: Enabling anomaly detection using machine learning automated techniques for attack detection.

At the access layer, IoT devices and smartphones connect to the network through base stations supported by Software Defined Radio (SDR). This setup allows the emulation of diverse traffic conditions, including benign operations and malicious behaviors such as Distributed Denial-of-Service (DDoS) attacks initiated by compromised IoT devices. The MMT-Sniffer, deployed at the IoT routing device, captures traffic flows at the earliest possible stage, ensuring that anomaly detection systems receive accurate and representative input data.

The 5G Core (EPC) hosts the principal network functions required for control and data plane operations, including the Access and Mobility Function (AMF), the Session Management Function (SMF), and the User Plane Function (UPF), interconnected with additional modules such as NRF, NEF, PCF, UDM, and AF. This architecture enables full end-to-end data transmission between IoT devices and the external Data Network (DN), thereby ensuring realistic conditions for traffic analysis and attack emulation.

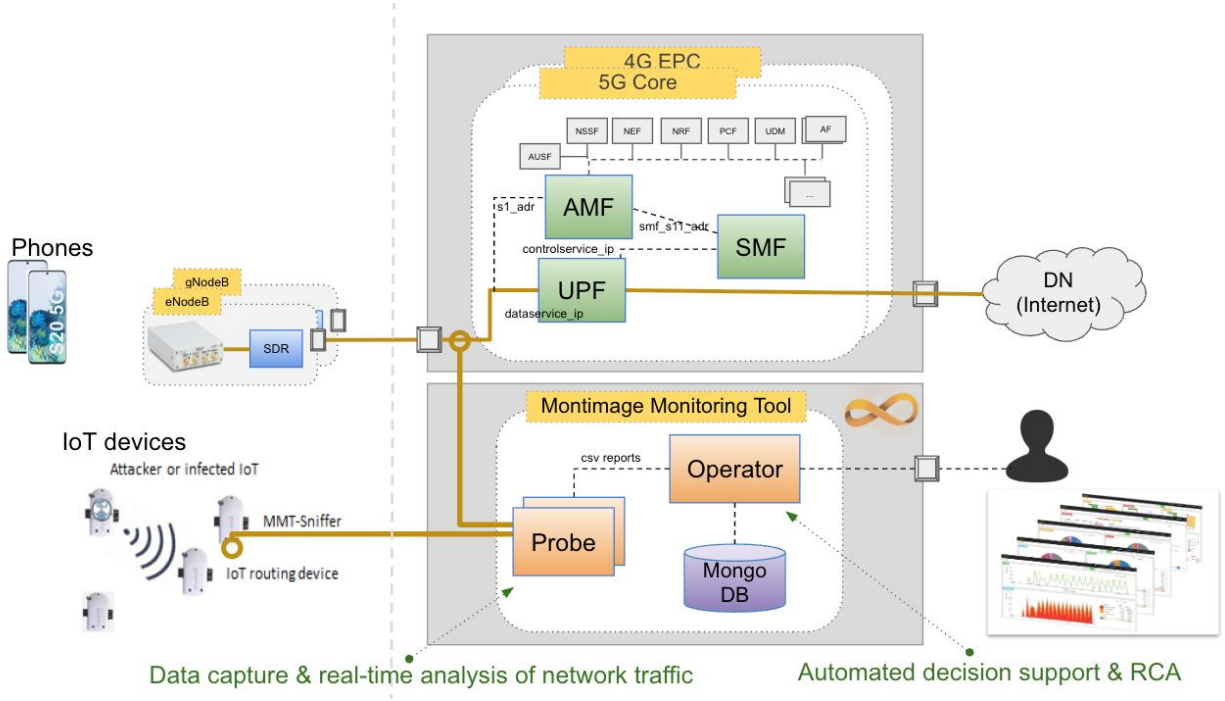

Figure 6: Montimage 5G-IoT testbed architecture

Complementing the 5G Core, the Montimage Monitoring Tool (MMT) is deployed as the main security monitoring component. The MMT Probe collects and analyzes traffic data in real time, while processed results are stored in a MongoDB database for correlation and long-term analysis. The Operator module consolidates insights from the probe and database, enabling root cause analysis, anomaly classification, and alert generation. Results are exposed through dashboards that provide real-time visualization of traffic trends, anomalies, and security events.

AI/ML capabilities sit on top of a feature-extraction and enrichment layer that incorporates protocol semantics, temporal patterns, graph relationships, and device context. The model zoo includes Isolation Forest and autoencoder-based detectors, LSTM for sequential behavior, and graph neural networks for topology-aware reasoning. Robustness is assessed through adversarial testing against evasion and poisoning, while accountability and resilience are tracked with precision/recall, time-to-detect, and mean time to recovery. Explainability is provided through SHAP and counterfactual analyses, summarized by an LLM-based explainer to translate low-level indicators into human-readable causes and mitigation suggestions.

This integrated testbed supports the following experimental capabilities:

- End-to-end monitoring of IoT traffic across access, core, and external networks.
- Emulation of attack scenarios involving infected or compromised IoT devices.
- Application of AI/ML-based anomaly detection to identify suspicious behaviors.
- Automated decision support and Root Cause Analysis (RCA) for mitigation of threats.

Overall, the 5G-IoT testbed provides the experimental foundation for UC#3.1, ensuring that anomaly detection solutions are developed, validated, and optimized under realistic conditions that closely mimic 5G-enabled IoT deployments. Reproducible deployment is facilitated by a Docker-based reference configuration that instantiates the 5G core components, MMT services (Probe, Operator, MongoDB), and example traffic generators. The materials and scripts are available at: https://github.com/montimage-projects/cerberus-edge-configuration.git

2.5.2. 5G-IoT Testbed Components Set-Up

2.5.2.1. AI-driven Security Monitoring for Anomaly Detection and Root Cause Analysis (AI-AD&RCA)

This comprehensive solution leverages the Montimage 5G-IoT Testbed as the experimental foundation for deploying advanced artificial intelligence techniques to enhance security monitoring capabilities in next-generation IoT ecosystems. The framework integrates real-time network traffic analysis with explainable AI methodologies to provide transparent, accountable, and resilient security operations.

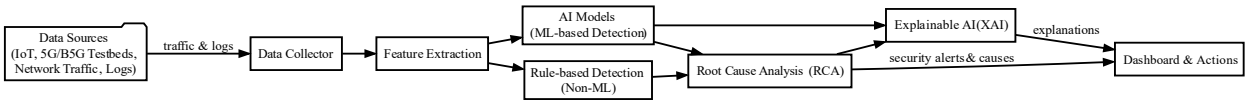

Figure 7: AI-AD&RCA flow diagram

Figure 7 illustrates the AI-driven Security Monitoring framework designed for anomaly detection and root cause analysis (RCA) in IoT and 5G/B5G networks. The framework integrates multiple

layers of monitoring, analysis, and explanation to ensure both rapid detection of threats and transparency in decision-making.

The process begins with data sources, where network traffic from IoT/5G/B5G testbeds and log data from SIEM systems are collected as the primary inputs. These heterogeneous data streams are then processed through the data collection and feature extraction stage, where the raw data is normalized and transformed into meaningful features suitable for analysis.

At the core of the monitoring system are the detection modules, which combine two complementary approaches. On the one hand, AI models provide machine learning-based detection for adaptive and scalable anomaly identification. On the other hand, rule-based detection (e.g., MMT-Security) applies deterministic checks and lightweight pattern matching to capture well-defined or known attack behaviors. Together, these methods enhance detection coverage and robustness.

To ensure that security decisions remain interpretable, the framework integrates an Explainable AI (XAI) layer. This module explains why specific anomalies were flagged, providing interpretable insights that support operator trust, compliance requirements, and accountability in security operations. Once an anomaly is confirmed, the root cause analysis (RCA) module investigates the underlying reasons for the detected issue, whether it stems from a misconfiguration, a targeted cyberattack, or system-level failures.

Finally, all findings are consolidated in a dashboard and action layer, where operators can view security alerts, RCA insights, and recommended mitigation actions. This user-friendly interface ensures that decisions are informed, actionable, and timely.

Overall, this layered architecture provides a holistic approach to anomaly detection and RCA. It combines AI and deterministic methods, enhances explainability, and strengthens resilience against evolving threats, thereby contributing to secure and trustworthy IoT and 5G/B5G networks.

AI-AD&RCA Inplementation

AI-AD&RCA, built on the Montimage Monitoring Tool (MMT) framework, offers flexible deployment options to accommodate diverse operational environments. Deployment can be performed via Docker containers, by building from the source code, or using precompiled packages. All deployment resources are available on Github[1]:

[1] Github repository: https://github.com/Montimage/maip

- Docker Deployment enables quick, consistent, and environment-agnostic setups. Using Docker containers simplifies installation and ensures reliable operation across different platforms, making it an ideal choice for rapid deployment.
- Source Code Installation is intended for environments that require customization or advanced debugging. By building the system directly from source code, operators gain full control over configuration and the internal logic of the platform, enabling fine-tuned adaptations for specific requirements.
- Precompiled Packages offer a fast deployment option for Ubuntu-based systems. The precompiled .deb packages reduce setup complexity and minimize configuration overhead, making them a convenient choice for quick installation without deep technical adjustments.

The core of the tool is a server implemented in ExpressJS, integrating the MMT framework written in C. This includes modules such as MMT-DPI, MMT-Probe, and MMT-Security for real-time feature extraction and traffic analysis. On top of this foundation, the system leverages advanced Python-based machine learning (ML) and explainable AI (XAI) libraries to enable intelligent anomaly detection and root cause analysis.

The server exposes over 60 Swagger APIs that deliver comprehensive services, including real-time feature extraction from network traffic, building and training ML models for anomaly detection, retrieving detailed model metadata, predicting whether network traffic is benign or malicious, and applying various XAI techniques to interpret and explain model predictions. These APIs support the full lifecycle of anomaly detection and root cause analysis, from data ingestion to actionable insights.

On the client side, the system includes a React-based interface that interacts with the server through the Swagger APIs. This interface provides a user-friendly environment for managing models, monitoring anomalies, and visualizing root cause insights. The landing page displays a comprehensive list of both prebuilt and user-defined ML models for anomaly detection, enabling flexible and efficient operational workflows.

Beyond ML-based detection, the system adopts a hybrid detection approach, supporting rule-driven and signature-based detection methods to ensure robust threat detection coverage. Prebuilt security rules define abnormal traffic patterns for known threats, while anomaly-based rules capture expected traffic behaviour, flagging deviations for further investigation. This hybrid architecture enables the system to effectively detect both known threats through signature-based detection and previously unseen anomalies through ML-based analysis, enhancing situational awareness and enabling precise root cause analysis.

2.6. CloudNativeLab Testbed

xNativeLab (previously CloudNativeLab) is an IMEC testbed which allows for fast and user-friendly creation of Kubernetes clusters for teaching and use case evaluation purposes. Kubernetes nodes run on virtual machines, allowing for fast and easy setup as well as teardown. xNativeLab is closely related to the IDlab Virtual Wall, which uses much of the same hardware but allows for baremetal access to devices and servers. xNativeLab provides SSH access through VPN to pre-provisioned servers with a fully set up cluster for various supported Kubernetes versions, including networking plugins. Conversely, the Virtual Wall is accessed through SSH by using jFed (Fed4Fire) or the SLICES project. While xNativeLab is the default evaluation environment for Kubernetes-related projects, the Virtual Wall is a fallback option if baremetal functionality is required.

2.6.1. CloudNativeLab Testbed Infrastructure

xNativeLab is a testbed service designed to simplify experimentation with cloud-native and edge-native frameworks in realistic environments. Traditional research infrastructure often requires extensive manual setup and advanced system administration skills, which can hinder the quick reproduction of experiments. xNativeLab builds on the SLICES research infrastructure, enabling researchers to quickly deploy distributed software frameworks across cloud, edge, and IoT devices while maintaining complete control over the stack for customization. A key component is the xNativeApp, a deployment package that includes both infrastructure and software definitions, allowing researchers to create reproducible and easily deployable research packages. The current implementation of xNativeLab is illustrated in Figure 8.

The Virtual Wall is the main hardware supporting experiments and consists of three parts. Virtual Wall 1 & 2 (206 & 159 nodes, respectively) are legacy testbeds with older hardware; the New Virtual Wall 1 is currently under construction and consists of 186 pcgen7 nodes with:

- 1x 6 core Intel Core i5-9500 CPU (3.00GHz)
- 64GB
- 512GB SSD
- 1 or 5 gigabit nics (+1 control connection)

In all cases, nodes have public IPv6 addresses. IPv4 addresses are LAN only, although public IPv4 addresses can be requested.

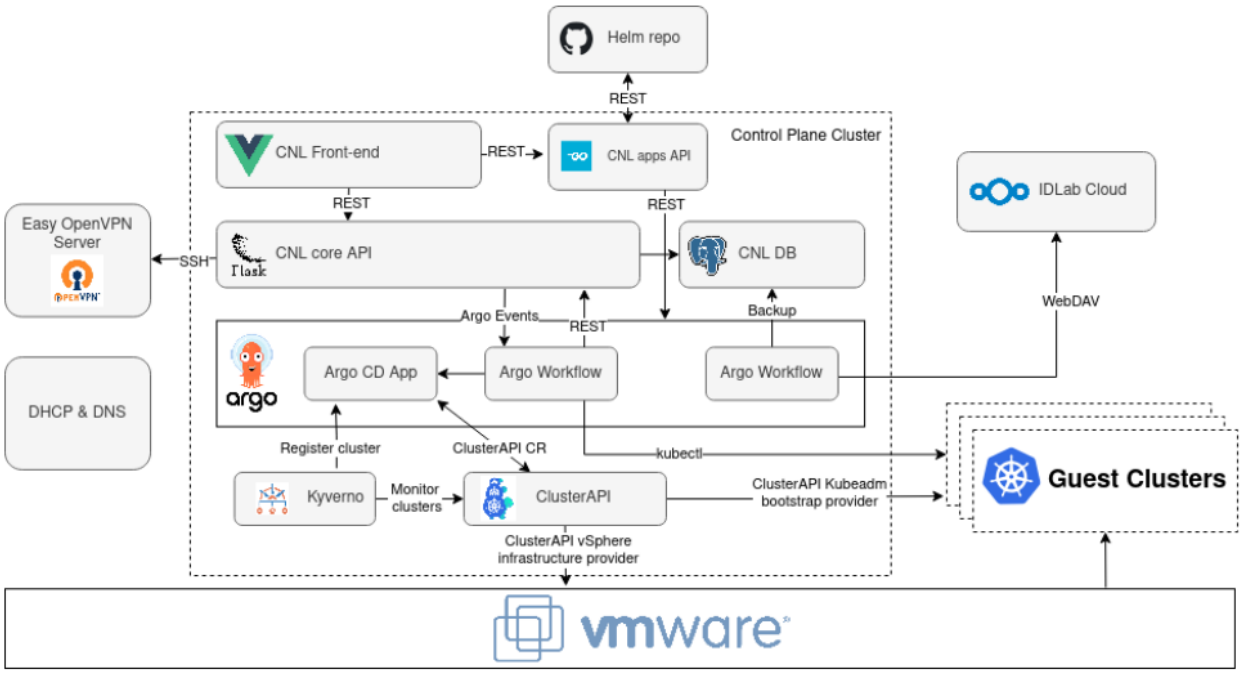

Figure 8: Current xNativeLab implementation

2.6.2. CloudNativeLab Testbed Components Set-Up

The components for each service are deployed differently across CloudNativeLab, the Virtual Wall (baremetal servers), and edge devices as required. While CloudNativeLab is the main testbed and the default for Kubernetes-related components, some components (e.g. Feather) are designed for specifically edge device operation, or require access to Operating System-level resources (e.g. Flocky).

2.6.2.1. TrustEdge

For TrustEdge, the Kubernetes control plane and all cloud-related components are set up in xNativeLab, hosted on a single control plane node. Edge devices that require attestation (e.g. Raspberry Pi, edge servers) are external to xNativeLab. Specifically, as shown in Figure 9:

- The Kubernetes control plane (**Certificates, RBAC, CRDs e.g. EdgeNode)** form the basis of the cluster and are hosted in their natural cloud environment in xNativeLab.
- Due to its tight interaction with the Kubernetes API itself and the need to run in a secure environment, the **Attestation controller** is run in the control plane on xNativeLab.
- The **Registrar/Tenant/Verifier** components form the cloud-based endpoints of TrustEdge for various operations; these interact with the Attestation controller and require secure (controlled) execution, and as such are hosted in the control plane.

- The **KeyLime agent** is the edge agent component of TrustEdge. Each edge device requesting to join the cluster has an agent deployed on it which initiates attestation and further component deployment.
- **Fledge/Feather** is the default Kubernetes agent used by TrustEdge. It runs on each attested edge device, and is securely deployed after attestation and certificate generation.

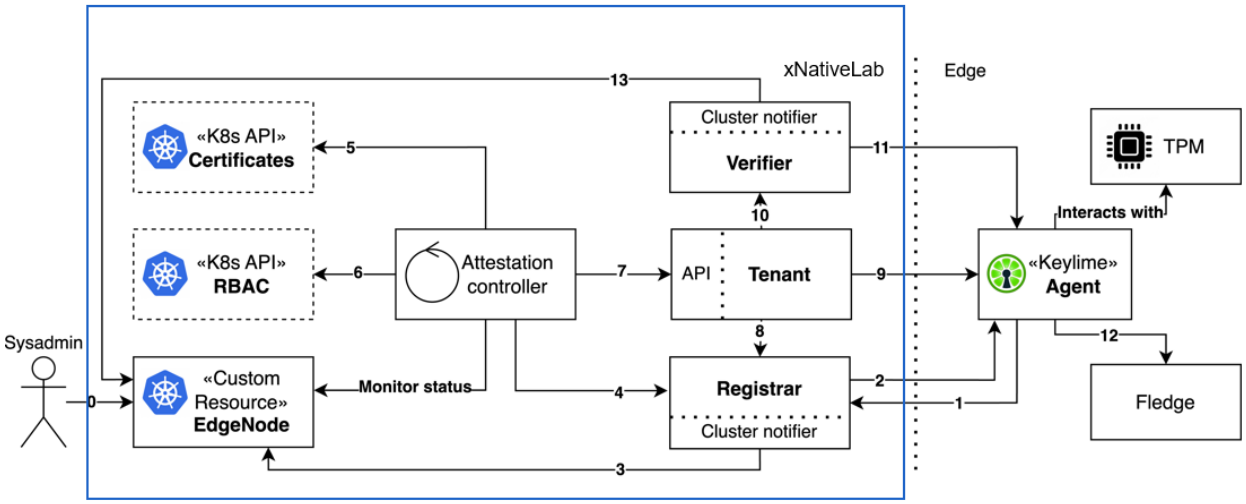

Figure 9: TrustEdge attestation components in Kubernetes and on edge devices

2.6.2.2. Feather

The Feather evaluation setup is divided between xNativeLab VMs and an edge device, specifically a Raspberry Pi 4. xNativeLab runs a Kubernetes cluster with a single control plane node and a worker node, as indicated by Figure 10. No modifications to a default cluster are required, as Feather merely replaces the standard kubelet, behaving according to Kubernetes expectations. Feather itself is deployed on the edge device, replacing the role of a kubelet and CNI plugin. The edge device itself is provisioned with containerd and KVM/Qemu to support the multi-runtime nature of Feather.

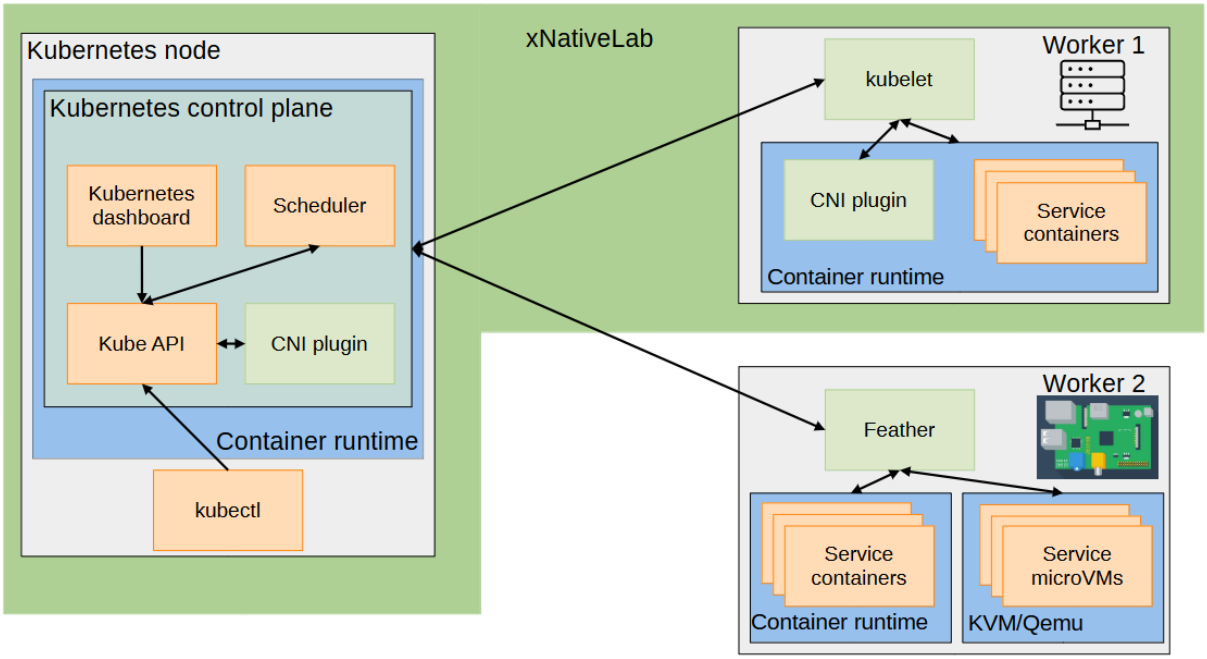

Figure 10: High level overview of Feather components

2.6.2.3. Flocky

Flocky is evaluated on bare-metal machines using the Virtual Wall directly rather than xNativeLab Kubernetes VMs.

For the scalability evaluation, all Flocky services are run on a single server simulating virtual edge devices (discovery + metadata services only, more information in the functional evaluation). The evaluation scenario consists of a straightforward execution and measurement script and warrants no further details.

The functional evaluation consists of five node configurations, each deployed on a single server as per Figure 11. The deployed services are the Flocky Discovery, Metadata, Deployment, and Swirly services, as well as a stubbed Feather interface which handles component deployment.

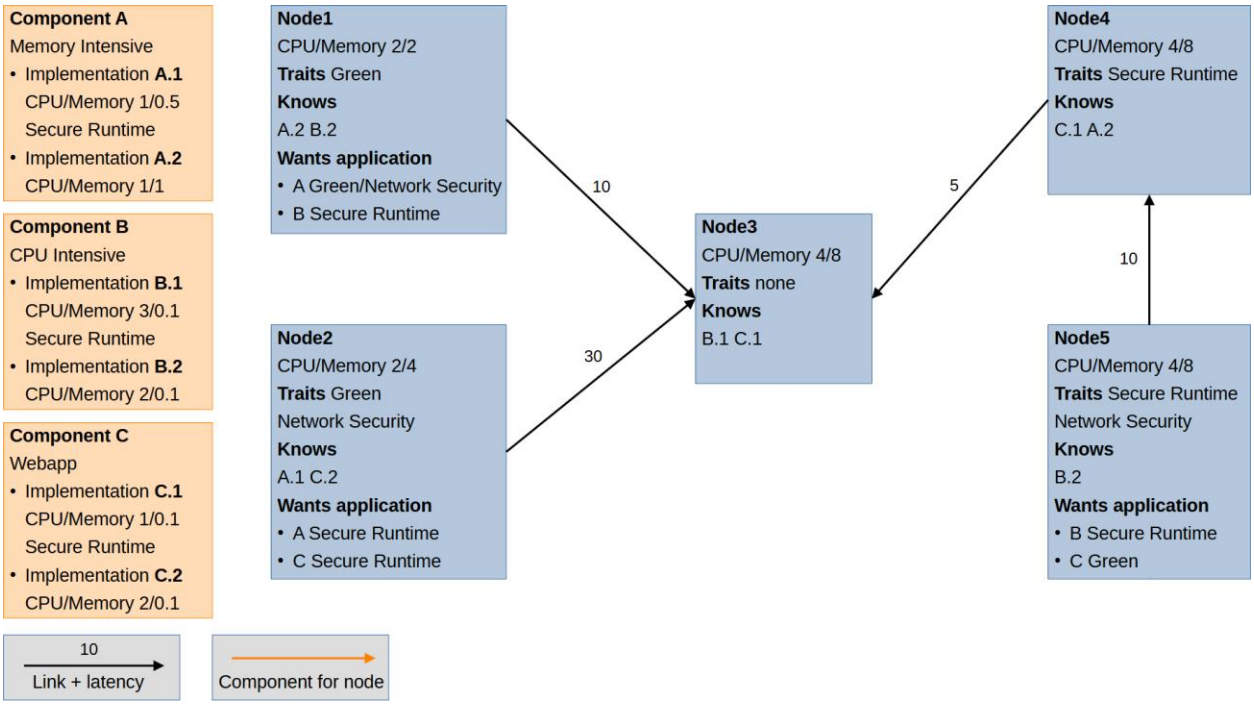

Figure 11: Functional evaluation setup for Flocky

Specifically, these microservices fulfill the following roles:

- **Discovery**: explores the network for other nearby Flocky devices, and provides basic node information for higher level services.
- **Metadata**: subscribes to the Discovery service, gathering node capabilities, resource availability, cluster information and active deployments from each node. Also keeps an up to date index of the local node's capabilities through CapabilityProviders. An example CapabilityProvider is Feather, which provides runtime capabilities for containers and unikernels.
- **Swirly**: comprises an orchestration algorithm and a web service for remote orchestration status updates, which may trigger deployment migrations.
- **Deployment**: a web service translating Open Application Model (Swirly) deployment requests to Kubernetes Deployment manifests, keeping track of deployment status and reporting status updates to Swirly services.
- Stubbed **Feather interface**: rather than allowing full processing of deployments, the Feather REST API is stubbed for this scenario.

2.7. Patras5G-PNET Testbed Components Set-Up

2.7.1. Patras5G-PNET Testbed Infrastructure

The Patras 5G-PNET facility is a private Network for 5G and IoT applications, adopting the Network Slice as a Service (NSaaS) model to provide tailored network slices for verticals to trial use cases and assess KPIs. It operates on licensed and unlicensed spectrum with dedicated SIM cards, supporting end-to-end customized slices across access, transport, and core, including IoT device slicing at the edge. The testbed enables MEC orchestration, mobility management for mobile streaming edge services, and holds an Academic License from the Greek government. It hosts a pre-commercial site with ERICSSON 5G Rel-17 equipment, testing 5G/6G cloud-to-edge scenarios using NOVA's licensed spectrum, with the 5G Core remotely located in Athens.

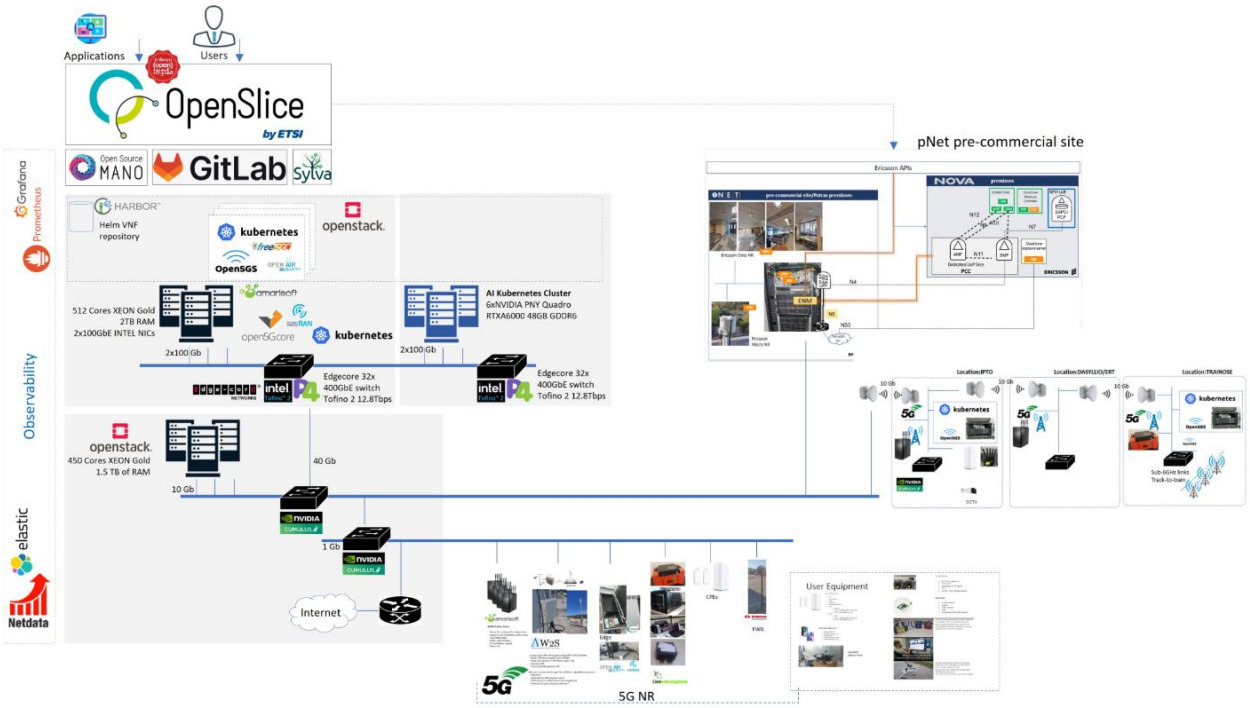

Figure 12: Patras 5G (PNET) facility infrastructure

The **cloud platform** provides 1082 CPUs, 4.5 TB RAM, and 100 TB storage. It includes three AI-enabled servers with 2x NVIDIA PNY QUADRO A6000 48GB GDDR6 GPUs interconnected via NVIDIA NVLINK Bridge. Servers connect via 100GbE/400GbE Edgecore P4 Switches and 10GbE/40GbE NVIDIA Cumulus switches with dual 10GbE NICs DPDK enabled. Kubernetes clusters are available on demand, managed via GitOps (LF Sylva, OpenSourceMANO, Terraform), attached to the 5G System dataplane.

Available 5G Core and EPC solutions that can be orchestrated in the facility include Open5GS in Kubernetes, free5GC in Kubernetes, OpenAirInterface in Kubernetes, AMARISOFT 5GC, Ericsson and RAN solutions from open-source tools such as OAI and SRS.

The facility's **radio equipment** features gNodeB models such as the 4x Amarisoft Classic callboxes supporting 5G SA/NSA, NB-IoT with 3SDR 2X2; 4x4 AW2S panther RU, several USRPs from ETTUS including N310, B210, B205 and LimeSDR cards. P-NET offers indoor and outdoor testing environments located at the university of Patras campus. The indoor site includes RAN equipment from Ericsson (4x DOTs 4479 – 4x4 MIMO) and 1 microcell 5W 4X4 outdoor RU operating FR1 n78.

User equipment includes >20 5G SA standalone smartphones, 4 CPEs, 2 Raspberry Pis equipped with 5G modems, USRPs running OAI ue-softmodem ,plus standalone USB 5G modems for legacy devices (e.g. laptops).

Monitoring is supported by Grafana, Prometheus, NetData, and OSM with VNF telemetry. Prometheus and NetData provide metrics for cloud infrastructure, VNFs, RAN nodes, and energy consumption of compute nodes, switches, 5G gNodeBs, and CPEs, stored in a Prometheus server. Grafana is used for visualization, with Elastic search and Kibana for data collection and visualization.

2.7.2. Patras5G-PNET Testbed Components Set-Up

Figure 13 illustrates the topology of the 5G testbed, with the same TelcoCloud components expected in future 6G networks, where we evaluated the UC4.5 AI-based MTD framework. The setup comprises three cloud environments, operated with OpenStack; one for the Core domain, termed ``Core NFVI", and the other for two distinct Radio Access Points with an Edge domain, the ``Edge NFVI." This deployment implements a distributed UPF architecture, where UPFs are co-located with the base stations (gNBs) in the Edge domains.

The Edge NFVIs include Radio Access elements (UEs and gNBs) and Edge Cloud elements (including the UPF and other possible CNFs). This deployment allows emulating a distributed UPF architecture, where the UPFs are co-located with the gNB on the Edge domain. The Core NFVI hosts the control plane of the 5G Core Network, the subscriber database, and other secondary CNFs for service provision. The Core NFVI also hosts the Slice Manager, and the NFVO implemented with Kubernetes (for CNFs) and OSM (for VNFs). The proposed solutions implemented in the AI-based MTD framework are also hosted in the core NFVI.

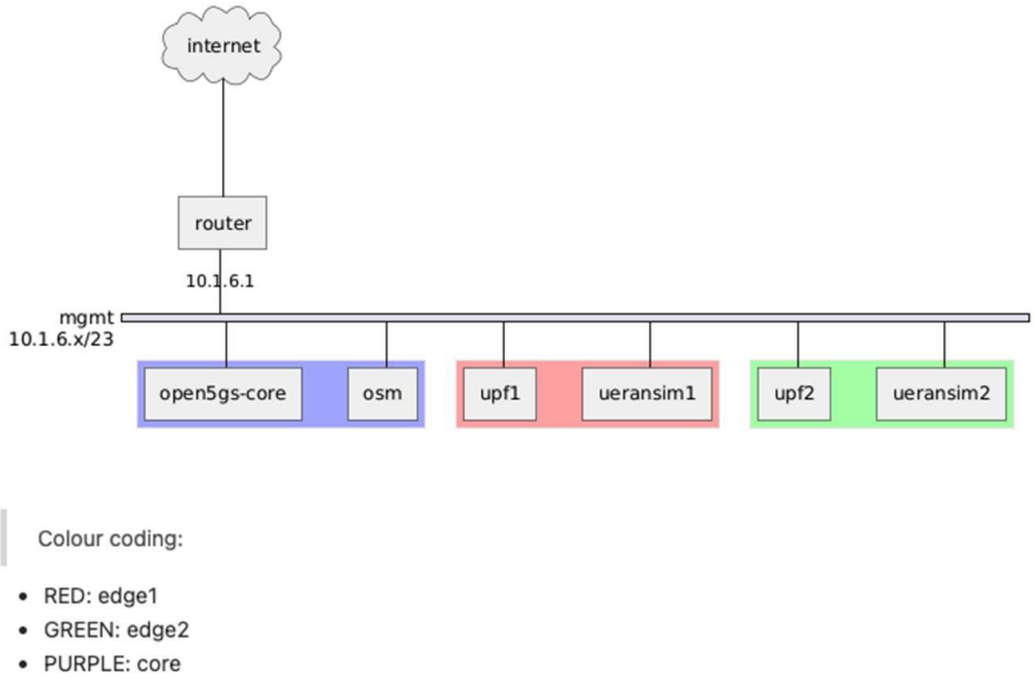

Figure 13: Topology of the TelcoCloud testbed running in PNET testbed.

The 5G Core is implemented with Open5GS, an open-source 3GPP Release-17 compliant 5G core. Open5GS provides the following network functions as discrete services, allowing the separation of the control and data planes: *(i)* AMF, *(ii)* SMF, *(iii)* UPF, *(iv)* AUSF, *(v)* NRF, *(vi)* UDM, *(vii)* PCF, and *(viii*) NSSF.

The RAN and mobile UEs are implemented by UERANSIM, an open-source UE and gNB simulator.

The 5G architecture is Standalone (5G SA). UERANSIM connects to Open5GS via a control interface with the AMF and a user interface to the UPFs. The simulated UEs and gNBs connect via a simulated radio interface. Unlike actual hardware equipment, UERANSIM allows the deployment of a significant number of virtual UEs to test the solution's scalability under an increasing network workload.

Following this structure, as presented in Figure 14: Testbed configuration for the AI-based MTD service, the components of the AI-based MTD service are integrated in the testbed as described below.

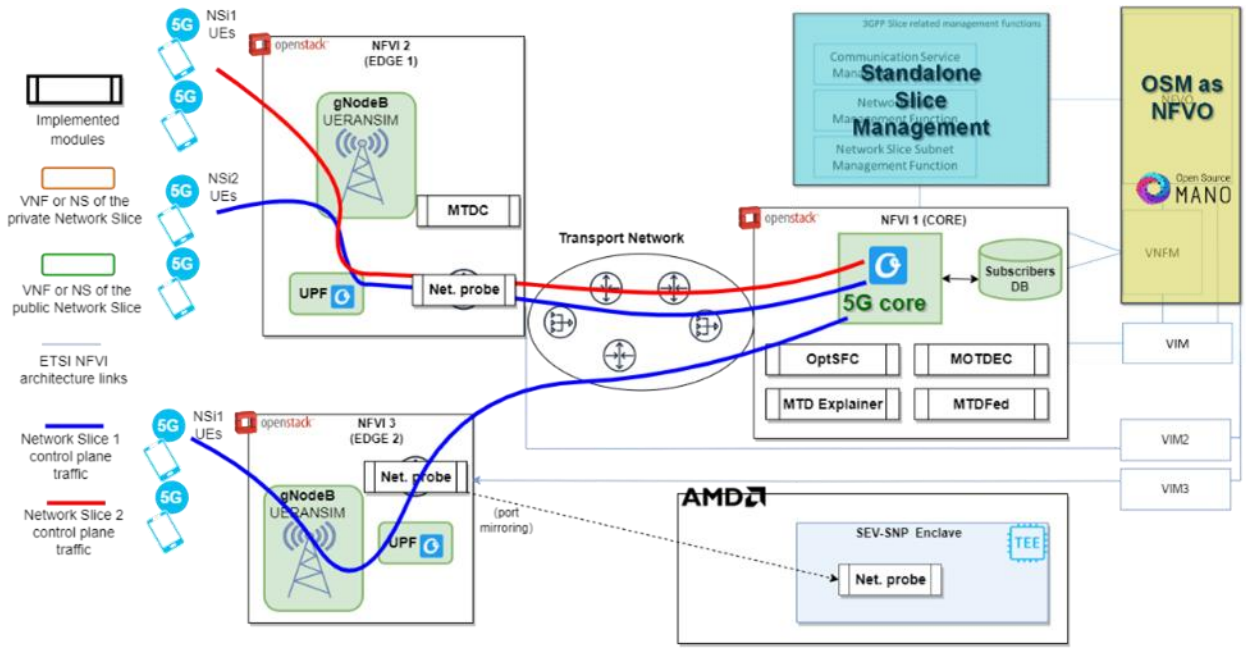

Figure 14: Testbed configuration for the AI-based MTD service

2.7.2.1. MTD Controller

The MTD Controller dynamically executes MTD actions to change the attack surface of Telco Cloud networks while maintaining service continuity. It focuses on interfacing and orchestrating NFV resources with minimal disruption.

It runs in the core domain as a VNF, with a minimal requirement of 8 vCPUs, 16 GB of RAM, and 160 GB of storage. Its northbound interface (NBI) connects it with the MTD Strategy Optimizer (via REST API), receiving from the latter the decisions taken on which MTD operation to perform. Its southbound interface (SBI) is instead interfaced with the NFVO to enforce the operation on the targeted NFV resource. Compatible NFVOs are ETSI OSM for VNFs and Kubernetes for CNFs.

2.7.2.2. MTD Strategy Optimizer

The MTD Strategy Optimizer is the cognitive component of the AI-based MTD service developed to dynamically decide which MTD actions to perform based on the state of the Telco Cloud network. It learns and then applies an optimized MTD strategy balancing security, cost, and QoS/QoE.

For this reason, the MTD Strategy Optimizer is interfaced with various near-real time data sources monitoring the network and collecting metrics to assess the network state. Specifically, the MTD Strategy Optimizer is interfaced via REST API with:

1. The AI-driven security monitoring for anomaly detection and root cause analysis framework, collecting security analytic data.

2. The NFVO collecting resource consumption metrics to estimate the cost of MTD operations based on the targeted NFV resource to reconfigure.
3. Openstack and Kubernetes for architectural and infrastructure information with near real-time information on the running VNFs/CNFs.
4. The MTD Controller, informing it of the MTD operation to enforce.
5. The MTD Explainer receiving multiple data from the MTD Strategy Optimizer, such as the decisions made and the rewards, to explain the policy and MTD strategy learned by the deep-RL model.

2.7.2.3. MTD Explainer

The MTD Explainer uses XAI for deep-RL models and post-hoc explanation techniques to clarify why specific MTD actions (e.g., migration vs. shuffling) for specific CNFs were chosen. For this reason, the MTD Explainer is hosted in the core domain together with the MTD Strategy Optimizer component, with which it is interfaced to interpret the latter's decisions.

2.7.2.4. MTDFed

MTDFed is directly interfaced with the MTD Strategy Optimizer as it enables virtual network operators (VNOs) running local MTD optimizers to collaboratively improve and speed-up the optimization of MTD strategies among participants running local MTD Strategy Optimizer. Using Federated Learning (FL), MTDFed enables collaborative optimization without compromising the confidentiality of the network traffic nor that of the VNOs' deep-RL models.

2.7.2.5. AI-driven Security Monitoring for Anomaly Detection and Root Cause Analysis

The AI-driven Security Monitoring and Root Cause Analysis (RCA) component, developed by MONT, will be integrated into the Patras5G-PNET testbed to provide real-time detection and analysis of anomalies in IoT and 6G network environments. The Patras5G-PNET facility, with its end-to-end slicing capabilities, MEC orchestration, and advanced monitoring infrastructure, offers an ideal environment to validate the component under realistic conditions

The Montimage Monitoring Tool (MMT) and Montimage AI Platform (MAIP) will be deployed at both the Edge NFVI and Core NFVI layers of the testbed. At the edge, lightweight anomaly detection probes (MMT-probe) analyze traffic close to IoT devices and user equipment, enabling low-latency event detection. At the core, MAIP aggregates heterogeneous data streams (network traffic, logs, telemetry) and applies advanced ML models, including CNNs and reinforcement learning, to detect suspicious behaviors and provide RCA. This dual deployment ensures scalability and accuracy across the distributed 6G architecture.

The integration leverages the testbed's Prometheus, Grafana, and Kibana monitoring stack for visualization and correlation of detected events with infrastructure telemetry. Detected

anomalies are further enriched with contextual information from Cyber Threat Intelligence (CTI) sources, enabling actionable insights and reducing false positives. RCA capabilities ensure that when anomalies occur, the component not only signals the event but also identifies the underlying cause (e.g., misconfiguration, DDoS traffic pattern, compromised IoT device).

This deployment contributes directly to UC#3.1 – Enabling anomaly detection using machine learning automated techniques for attack detection, validating its performance under realistic 6G testbed conditions. It supports the NATWORK KPIs on Mean Time to Detect (MTTD), False Positive/Negative rates, and Mean Time to Resolve (MTTR) by embedding monitoring and RCA capabilities into the testbed's edge-to-core continuum.

2.8. NCL Testbed Components Set-Up

2.8.1. NCL Testbed Infrastructure

The Network Convergence Laboratory (NCL) at the University of Essex provides the foundation for the edge–cloud infrastructure used in NATWORK. NCL is a state-of-the-art research facility designed to explore cloud–edge convergence, energy-aware orchestration, and secure networked services. It integrates heterogeneous compute and storage resources with a high-capacity programmable SDN network, enabling experimentation with advanced 6G edge–cloud concepts.

For the testbeds we have two large-scale core cloud clusters, 2 cloudlets, and 4 edge nodes. Each core cluster node is based on AMD EPYC 7352 24-core processors (48 threads), Ubuntu 22.04 LTS, and NUMA-optimized architecture, interconnected via Pica programmable switches for flexible traffic steering. Collectively, the NCL infrastructure integrates over 200+ CPUs, 200+ TB of storage, and a programmable SDN/P4 network with 180 Gbps SDN and 100 Gbps P4 capabilities, providing the aggregate resources across cloud, cloudlet, and edge tiers. Cloudlet nodes serve as intermediate aggregation points, offering localized computing and orchestration closer to the edge. Edge servers extend the continuum further by hosting lightweight CNFs, microservices, and latency-sensitive workloads, while also acting as distributed points for federated learning and cyberattack simulations. The lab interconnects multiple edge-cloud clusters through a dedicated SDN fabric offering high bandwidth, ensuring low latency and high throughput across distributed domains.

A set of containerized user-emulation clusters (user groups) have been deployed to generate application demand across the testbed. These emulation clusters are hosted on machines based on AMD EPYC 7281 16-core processors (32 threads) and run request generators as pods: benign clients that emulate real user behavior and malicious users that produce controlled, oscillating

request patterns. The malicious containers are used to reproduce and evaluate novel cyberattack scenarios such as Denial of Sustainability.

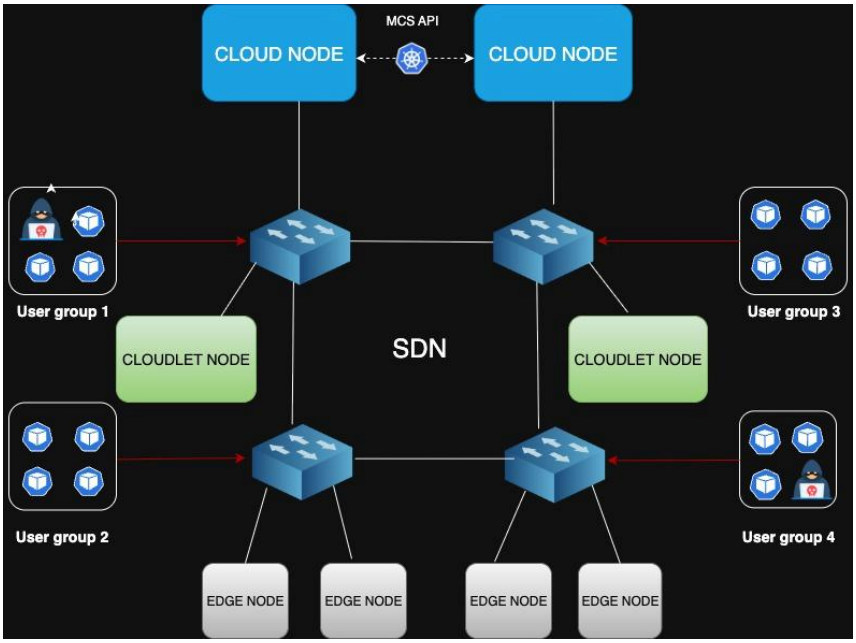

Figure 15: NCL Testbed Infrastructure

Figure 15: NCL Testbed Infrastructur demonstrates a high-level view of the testbed infrastructure, showing a multi-domain edge-to-cloud environment where secure-by-design and energy-aware orchestration, federated learning frameworks, and secure slice management functions are developed and validated.

2.8.2. NCL Testbed Components Set-Up

Figure 16: NCL Testbed Components illustrates the components of the NCL edge–cloud testbed used to evaluate UC1.1, focusing on decentralized orchestration and management of 6G slices under novel cyberattacks such as Denial of Sustainability (DoSt). The setup leverages the FORK orchestrator [2] as a baseline, extended with security-compliant orchestration (sFORK), federated learning, cyber threat intelligence (CTI) integration and monitoring frameworks.

Prometheus telemetry and Grafana dashboards provide detailed observability of CPU and memory resources of CNF services, response times, and energy utilization. Secure-by-design orchestration is delivered via the sFORK framework, comprising global agents, local orchestration agents, CNF managers, slice managers, dependency operators, and AI-powered scheduler, with Kubernetes serving as both orchestration platform and execution environment for distributed CNFs and learning tasks.

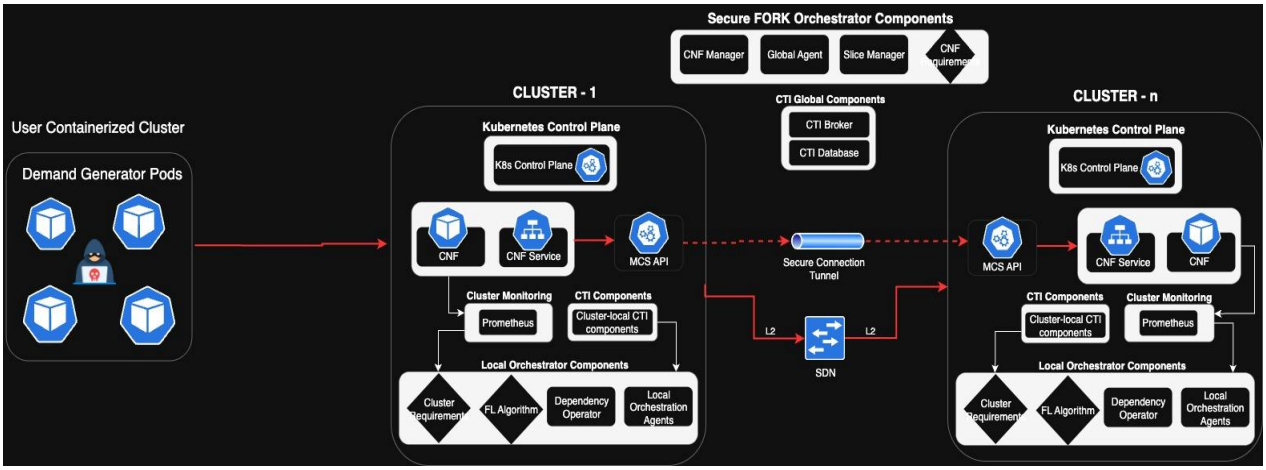

Figure 16: NCL Testbed Components

Security-compliant slice management leverages federated learning agents and the CTI framework, a peer-to-peer Operation Support System (OSS) that collects cluster hygiene metrics and guides resource allocation to mitigate threats such as DoSt attacks while aligning with sustainability goals. Federated learning for edge-to-cloud is done through distributed machine learning agents and shared datasets, with Prometheus TSDB and MinIO integration planned to support persistent storage. Secure inter-cluster connectivity is maintained via Submariner tunnels and the MCS API. Collectively, these components provide monitoring, orchestration, security, and learning capabilities, allowing evaluation of energy efficiency, slice resiliency, and attack mitigation in a realistic multi-domain edge–cloud environment.

2.8.2.1. Energy efficient over edge-cloud

This component focuses on evaluating the energy impact of benign and malicious requests on edge–cloud resources. Containerized demand-generation clusters are used to emulate realistic user traffic, including malicious patterns reproducing Denial of Sustainability (DoSt) attacks. These traffic generators, deployed as Kubernetes pods, stress CPU and memory resources of CNF services by issuing oscillating requests that degrade performance without causing full-service outages. Prometheus telemetry collects fine-grained CPU, memory, and response-time metrics, while Grafana provides visualization and comparative analysis across single- and multi-cluster setups. Together, these components enable measurement of energy utilization, service degradation, and QoS variance under controlled load scenarios, forming the basis for sustainable edge–cloud orchestration strategies.

2.8.2.2. Secure-by-design Orchestration

The secure-by-design orchestrator architecture integrates several key components:

- Global Agent**:** Serves as the central decision-maker, managing global dependency graphs, initiating and monitoring deployments, and negotiating with local orchestration agents.

It evaluates cluster offerings based on security, resource availability, hygiene, and energy sustainability metrics.

- CNF Manager**:** Oversees the lifecycle of Cloud-Native Functions, ensuring that CNFs meet predefined requirements while coordinating with local orchestration agents.
- Slice Manager**:** Orchestrates network slices by tracking their status, dynamically allocating resources, and interacting with global and local agents to ensure efficient slice deployment and monitoring.
- Local Orchestration Agents**:** Operate within each cluster to manage CNF deployments and lifecycle. They report cluster capabilities, including resource availability, hygiene scores, and compliance, to the global agent, and execute deployment decisions in real time.
- Dependency Operator**:** Maintains global dependency graphs that map relationships between CNFs and microservices across clusters. It ensures subgraphs are up-to-date and distributed according to resource availability and security policies.
- AI-Powered Scheduling**:** Applies machine learning models to improve resource allocation and scheduling, providing local agents with insights derived from usage patterns and predicted demand to optimize CNF performance.
- Cluster Requirements**:** Defines cluster-specific requirements for CNF deployment, guiding local orchestration agents to allocate resources, enforce security policies, and meet performance metrics.
- Monitoring**:** Continuously observes the health, performance, and security of CNFs and network slices. Integrated Prometheus telemetry supplies data to both local and global agents for timely decision-making and compliance with security policies.

2.8.2.3. Security-compliant Slice Management

Security-compliant slice management ensures that network slices and CNFs are deployed and operated securely, resiliently, and efficiently across multi-cluster edge–cloud environments. The secure-by-design orchestration framework integrates Cyber Threat Intelligence (CTI)**,** enabling continuous vulnerability awareness and adaptive risk mitigation. The sFORK orchestrator coordinates within its operators and agents, leveraging CTI insights such as hygiene scores and vulnerability mappings to ensure only compliant and trusted deployments occur across clusters.

The CTI Agent collects and shares vulnerability data between clusters, anonymizing sensitive fields while preserving actionable intelligence. Local orchestration agents execute secure deployments, while the Dependency Operator manages CNF interrelations. Prometheus telemetry provides real-time monitoring of CPU, memory, and network metrics, which are

utilized by Federated Learning (FL) agents to support predictive resource management and anomaly detection. Together, these components enable secure and risk aware management of 6G slices across the edge–cloud continuum.

2.8.2.4. Federated Learning for Edge-to-cloud

The Federated Learning (FL) framework enables intelligent, distributed workload prediction and anomaly detection across edge and cloud domains within the 6G core. FL agents operate close to the infrastructure layer, training locally on CNF CPU, memory, and energy utilization metrics to derive cluster-specific insights. A data pipeline periodically extracts telemetry from Prometheus TSDB, batches it, and transfers it to MinIO for persistent storage and offline processing. These datasets comprise custom DoST simulation data, capture a mix of benign and adversarial workload behaviors, forming the foundation for model training across distributed nodes.

Initial benchmarking has been conducted using historical Google cluster traces as baseline datasets for model development and performance validation. Multiple machine learning models were evaluated for workload prediction, with XGBoost achieving the best performance. Accordingly, the framework adopts Federated XGBoost as its baseline, leveraging an XGBoost bagging approach for decentralized training across distributed nodes.

Two models are being developed under this framework: a resource optimization model that predicts workload trends and advises the orchestrator on energy-aware scaling decisions, and a classification model designed to identify and distinguish DoST-induced oscillatory patterns from benign users with legitimate demand surges. Future development will extend this architecture with decentralized publish/subscribe based FL integration within Kubernetes for enabling continuous improvement in workload forecasting, traffic classification, and orchestration resilience against evolving DoST-like threats.

2.9. TSS Testbed infrastructure and Components Set-Up

2.9.1. TSS Testbed Infrastructure

TSS testbed is dedicated to validating CIA-hardening techniques (confidentiality, integrity, Availability) on X86 native, containerized, and WASM workloads. It provides:

Hardware resources:

- Dedicated server with multi-core CPU, 32-64 GB RAM, SSD storage.
- Networking at 1/10 GbE.
- Energy measurement instrumentation for KPI monitoring.

Virtualisation & orchestration:

- Docker & kubernetes clusters for container-based deployments.
- Support for Kubernetes sidecar deployment patterns (e.g., D-MUTRA sidecar for runtime attestation).

Monitoring & analytics stack:

- Prometheus + Grafana for system metrics (CPU, memory, energy).

Security-specific frameworks:

- D-MUTRA blockchain: decentralized, dependency-free mutual remote attestation framework.
- LLVM toolchain for instrumentation of x86 code.
- Modified WASMTIME runtime with integrity checker.
- E9patch for hot patching executable binaries. It allows modifying and instrumenting binaries without requiring source code access.

2.9.2. TSS Testbed Components Set-Up

The testbed is structured into three parts corresponding to the supported format:

2.9.2.1. CIA-hardening of x86 payloads

- **Workload**: ELF-formatted executable MMT-Probe running as the security sensitive native workload defined in Use Case 1.2.
- For practical reasons and representativeness of the test, SECaaS-hardened MMT-Probe, may be tested on a different testbed where it is deployed and running. (e.g., P-NET's, CNIT's or MONT's). This will simplify the operations to generate normal traffic conditions (e.g., by T-Rex), collect performance ratio (e.g., throughput)
- Tests will be made for CIA hardening as follows:
  1. Confidentiality: automatic encryption of ELF code section, decryption at runtime (<3s).
  2. Integrity: injection of Prove/Verify primitives, comparison of runtime bytecode signature vs pre-deployment reference, logging to D-MUTRA.
  3. Availability: monitoring packet-processing routines (timestamps, throughput baseline).
- Workflow:
  1. Build & protect MMT binary via SECaaS pipeline, leveraging our tools aka Systemic and D-MUTRA.
  2. Instrument the hardened MMT binary, with timestamps, possibly triggered by i9patch's trampolines or through a novel LLVM-based inserted probe.
  3. Deploy with blockchain nodes active.

4. Run test traffic (e.g;, simulated traffic/live/PCAP).
5. Compare KPIs vs baseline unprotected MMT.

2.9.2.2. CIA-hardening of containerized payloads

- Workload: MMT-Probe and/or IS-RD's Liquid xAPP deployed in Docker/K8s
- For practical reasons and representativeness of the test, SECaaS-hardened MMT-Probe, may be tested on a different testbed where it is deployed and running. (e.g., IS-RD's or MONT's), simplify the operations to generate normal traffic conditions (e.g., by T-Rex), collect performance ratio (e.g., MMT's throughput)

Tests will be made for CIA hardening as follows:

1. As the state of the art is mature and fulfilled in this area, no container-based integrity test will be implemented.
2. Integrity verification is implemented with a sidecar attached to the workload namespace, monitoring the container's workload memory footprint, and performing runtime attestation with D-MUTRA.
3. Availability: monitoring through sidecar telemetry probes accessing the container's workload elements (e.g., sampled collection of the instruction pointer, sampled collection of the stack trace, sidecar located performance reference payload).
4. To produce these hardenings (i.e., for integrity and availability preservation), sufficient privilege or capability shall be delivered to the sidecar container (e.g., CAP_SYS_PTRACE).

Workflow:

5. Deploy workload as container in Kubernetes cluster.
6. Append D-MUTRA sidecar (Docker Compose / Helm).
7. Run the attestation verification pattern (e.g., cyclic, on-demand).
6. Compare attestation timing & performance penalties Run test traffic (e.g., simulated traffic/live/PCAP).
7. Compare KPIs vs baseline unprotected MMT or Liquid xAPP.

2.9.2.3. CIA-hardening of WASM payloads

Workload: MMT-Probe ported to WASM, executed on modified WASMTIME runtime.

Tests will be made for CIA hardening as follows:

a. Confidentiality: encryption of WASM bytecode, measured decryption delay. This test is pending our WASM hardening feasibility study.

b. Integrity: Runtime signature computed from JIT-serialized blob, compared to reference signature. This test is pending our WASM hardening feasibility study.

- Availability: monitoring through the modified runtime for its own instrumentation. This test is pending our WASM hardening feasibility study.

Procedure:

- Deploy modified WASMTIME with Prove/Verify routines
- Load protected WASM module; run baseline traffic.
- Trigger integrity checks (periodic + on-demand).
- Measure startup delay, attestation cycle, and runtime overhead.

2.10. ISRD Testbed Components Set-Up

2.10.1. ISRD Testbed Infrastructure

Figure 17 illustrates the O-RAN functional architecture. As a central part of cloud-native, virtualized networking solutions, the RAN—specifically within the Liquid RAN framework—comprises several key component groups: the 5G O-DU (Open RAN Distributed Unit), 5G O-CU (Open RAN Centralized Unit), 5G Near-RT RIC (Near-Real-Time Radio Intelligent Controller) integrated with various xApps, and the 5G Core Network (5GC).

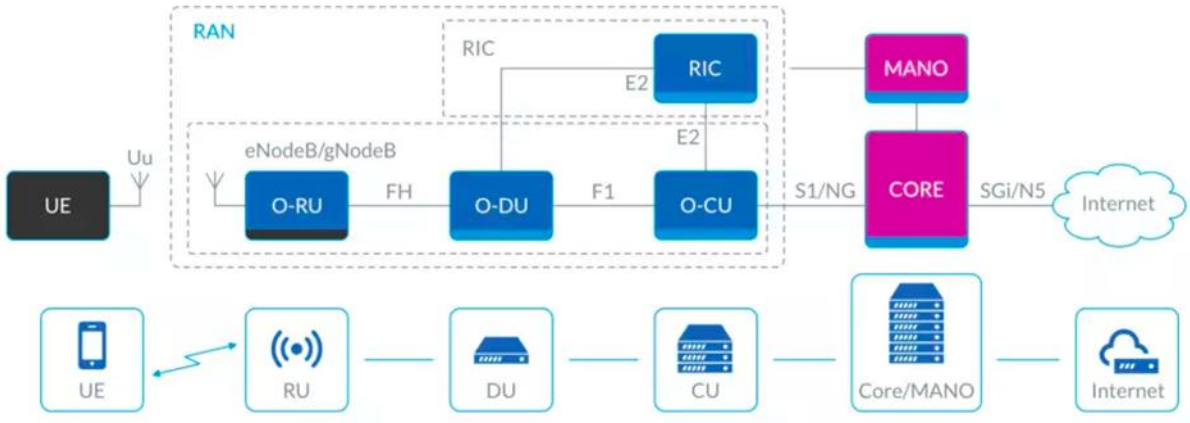

Figure 17: ISRD Testbed Infrastructure

Through the ISRD approach, a flexible allocation and migration of 5G NR protocol stack functions between the O-DU and O-CU are achievable. Both the ISRD Liquid RAN O-CU and O-DU represent advanced implementations of the O-RAN Central and Distributed Units. Similarly, the ISRD Liquid RAN Near-RT RIC embodies an O-RAN-compliant Near Real-Time RAN Intelligent Controller, offering sophisticated logical control and optimization of RAN components (O-DU and O-CU). It

accomplishes this through detailed data collection and responsive actions over the E2 interface, enabling near-real-time management of RAN performance and resources.

2.10.2. ISRD Testbed Components Set-Up

2.10.2.1. JDM-xApp

The JDM-xApp continuously receives real-time metrics from E2 nodes and evaluates patterns of degradation. It can operate in two modes: in the rule-based mode, it applies threshold logic—such as flagging jamming when BLER exceeds 10% or when CQI drops abruptly—triggering mitigation actions that limit MCS levels for specific UEs. This mode is deterministic, simple to deploy, and does not require prior training or data labeling.

For environments where jamming patterns may evolve or exhibit non-linear characteristics, the second, **ML-based mode** approach introduces intelligence through unsupervised learning techniques like clustering or anomaly detection. These models analyze historical metric trends to identify outliers indicative of jamming and classify the severity level. The xApp then dynamically enforces adaptive scheduling policies based on this classification. It also incorporates a feedback mechanism to adjust its behavior over time, thereby improving its resilience to new or stealthy jamming methods. Although more complex, this mode is especially useful in high-mobility and dense deployments, or adversarial environments.

In both operational modes, the JDM-xApp uses the E2 CONTROL interface to issue commands to the O-DU, modifying the scheduler's behavior for affected UEs. This could include restricting MCS levels, altering scheduling priorities, or temporarily offloading traffic. Such targeted control ensures precise jamming mitigation without unnecessarily degrading overall system performance.

2.10.2.2. Liquid RAN

Figure 18The deployment depends on the specific end-user needs, but usually consists of multiple instances of O-DU, O-CU. The 5GC, and either a commercial or an USRP-based RU can be also provided if required. Additionally, the deployment can integrate the Near-RT RIC and the relevant xApp(s) such as RAN KPM xApp. All the Liquid RAN components are deployed as containerized applications that can run either directly on bare metal or within Kubernetes environments as pods. Figure 18 depicts a complete deployment with all optional and additional components.

In this architecture, the RAN components interconnect with other 5G network elements as defined below. Liquid RAN supports a wide range of commercial User Equipment (UE) brands such as Oppo, Samsung, and OnePlus. The User Equipment (UE) can connect to either a commercial RU (for example, Benetel 550) or a COTS RU equipped with a USRP (b210) serving as the radio front end, with UEs typically placed inside an RF shield box for test isolation. The O-DU

interfaces with commercial RUs via the Open Fronthaul (Open-FH) interface, while communication with COTS RUs occurs over a proprietary fronthaul link. The O-CU connects to the 5G Core Network (5GC) through standardized N2 and N3 interfaces. Additionally, xApps expose external APIs that provide access to RAN-related metrics, enabling enhanced monitoring and control capabilities.

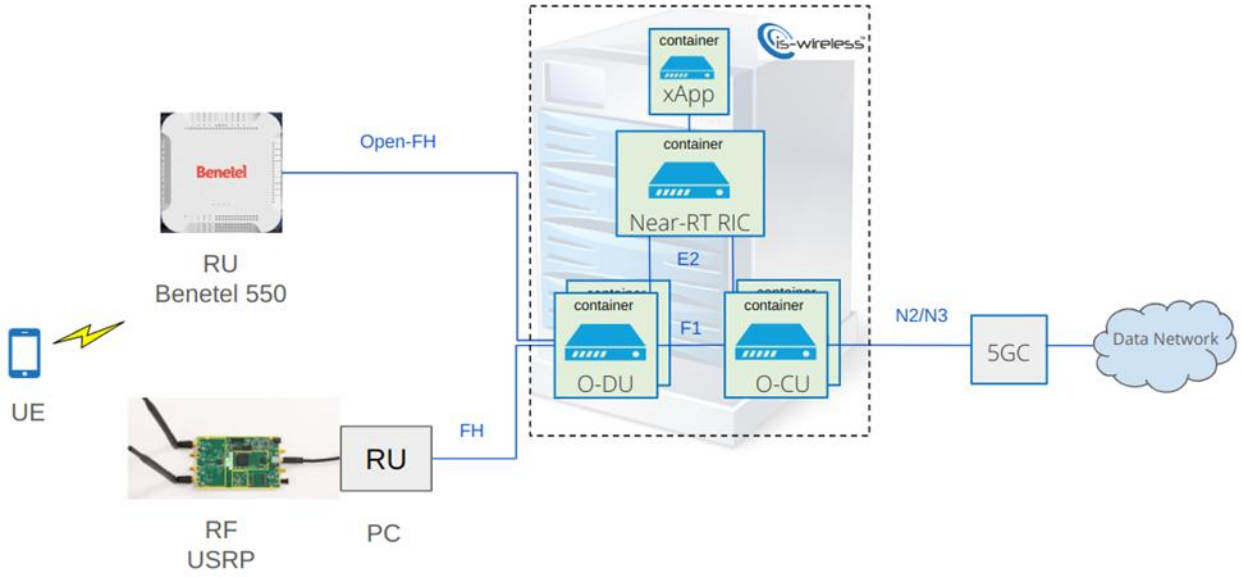

Figure 18: ISRD Testbed setup

Liquid RAN supports various configurations of bands (n77, n78, n79), bandwidths (10-100MHz) and antennas (SISO and MIMO 2x2 DL). For a complete deployment servers with proper computing resources, which depends mostly on the bandwidth used, will be required.

2.10.2.3. Liquid Near-RT RIC

The ISRD Liquid Near-RT RIC serves as an O-RAN–standardized Near-Real-Time RAN Intelligent Controller designed to optimize RAN performance. As shown in Figure 19, it interfaces with E2 nodes—specifically O-DUs and O-CUs—as well as with xApps and the Non-RT RIC, using O-RAN–compliant E2, xApp API, and A1 interfaces, respectively. While the SMO and Non-RT RIC are not part of the ISRD solution, the ISRD Near-RT RIC itself is delivered as a containerized software package built on Docker. Its default deployment method uses Docker Compose, though alternative orchestration options such as Docker Swarm and Kubernetes are also supported, ensuring flexible and scalable deployment in various environments.

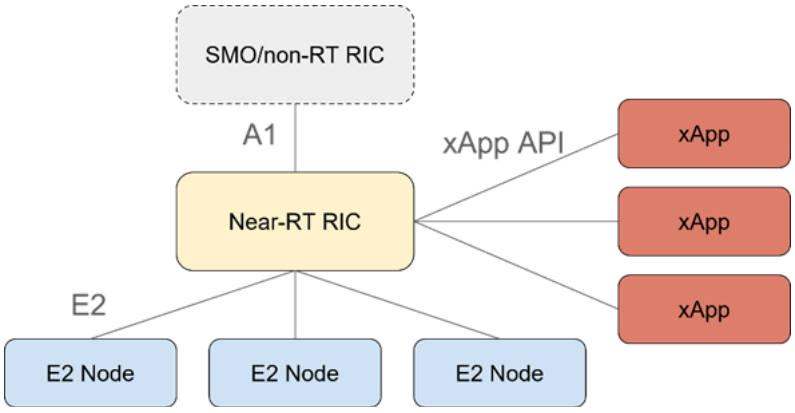

Figure 19: ISRD Liquid Near-RT RIC interfaces

The basic deployment of Liquid RIC follows a Cloud-Native Function (CNF) model using Docker containers managed through Docker Compose. In a single-machine setup, the user operates with a single YAML file for entity management, simplifying network configuration considerably. The deployment process involves pulling the Near-RT RIC Docker images from DockerHub and then configuring the deployment through the `docker-compose.yaml` file. This YAML file defines the container names, their interconnections, and the overall structure of the Near-RT RIC environment. Liquid RIC can be run in two ways, the first being the console and the second being detached mode, running containers in the background.

Figure 20: The main screen of the Liquid Near-RT RIC

The Liquid RIC container and all other containers provide their logs which can be viewed using standard docker commands. The logs are written to both the system console of a respective container and to the dedicated log files. The Liquid RIC offers a Graphical User Interface (GUI) for

both the Near-RT RIC and KPM xApp. Through the GUI the user can observe Near RT RIC and RAN performance monitoring as well as perform xApp control such as addition and removal. The GUI consists of several different views for exploring different dimensions of Liquid RIC. The views are: Home, xApps, xApp, E2 Nodes, E2 Node, Cells, Cell, Logs and a Settings view. Figure 20 shows Home view, which is the main screen.

2.10.2.4. KPM xApp

The KPM xApp is a built-in xApp of the Liquid Near-RT RIC. It subscribes to the measurements (KPMs) from all cells, which are stored in the Valkey database. The KPM collection interval and the reporting interval is 1 second. The KPMs can be viewed both in the Liquid Near-RT RIC GUI and the KPM xApp GUI. The KPM xApp GUI is Graphana based and it is a browser-based application, including several dashboard which support different RAN vendors. The example dashboard is depicted in Figure 21.

Figure 21: Grafana dashboard with ISRD KPMs

2.11. ELTE Testbed Components Set-Up

The ELTE testbed has been designed to emulate a complete 5G network with support for advanced security and monitoring capabilities. It integrates open-source 5G components, emulated radio access, a programmable data network, and blockchain services. This environment provides the basis for experiments in anomaly detection, monitoring, and blockchain-assisted trust mechanisms.

2.11.1. ELTE Testbed Infrastructure

The ELTE testbed provides a realistic and flexible environment for 5G and IoT experimentation, combining open-source 5G components, programmable hardware, cloud infrastructure, and blockchain-enabled security. The 5G Core node is implemented using Open5GS, with two separate deployments to ensure modularity and performance separation. One instance handles the complete suite of control-plane functions, including session management and mobility control, while a second instance is dedicated exclusively to the User Plane Function (UPF), enabling precise traffic management and realistic evaluation of data-plane operations. A dedicated setup is integrated to validate machine learning functions within the data plane, consisting of high-performance servers for model deployment, traffic orchestration, and monitoring, paired with Intel Tofino switches running P4 pipelines enhanced with ML-based classification logic. This configuration allows high-speed packet classification, fine-grained monitoring, and reproducible testing of AI-driven anomaly detection methods, while a Flwr federated learning node supports distributed training and secure aggregation of ML models across the network, ensuring scalability and privacy without centralizing sensitive data.

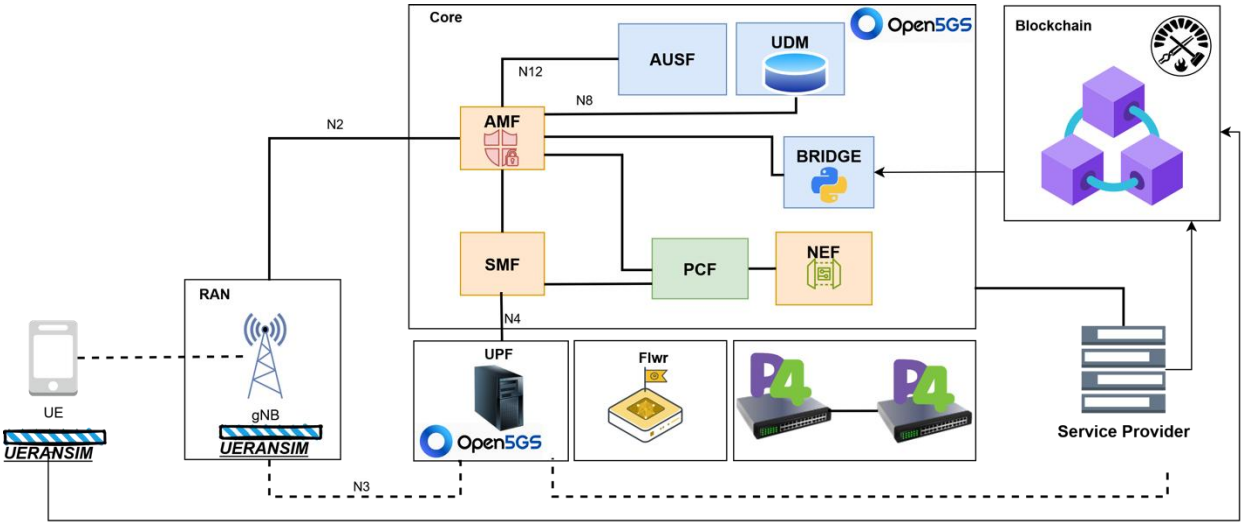

Figure 22: ELTE Testbed infrastructure

The Data Network is built on OpenStack and closely integrated with the Open5GS based UPF, providing a programmable and flexible environment for service deployment, traffic routing, and multi-domain emulation. The cloud infrastructure enables dynamic orchestration of virtualized network functions and application workloads, supporting experiments that combine communication-layer performance with application-layer behaviors. Radio access is emulated through UERANSIM, with separate instances for User Equipment and gNB, allowing scalable emulation of multiple devices and radio interfaces while maintaining compliance with 3GPP

standards. This approach facilitates controlled and repeatable experiments without the need for extensive physical hardware.

A blockchain layer, deployed across all nodes using the Foundry framework, adds distributed trust, tamper-resistant logging, and secure coordination for network operations. It provides a foundation for validating blockchain-assisted security and anomaly detection mechanisms, ensuring transparency, accountability, and resilience against compromised nodes. By integrating these components, the testbed offers a comprehensive platform for experimentation with end-to-end 5G-IoT scenarios, including advanced machine learning validation, programmable data-plane evaluation, distributed AI training, and blockchain-enhanced security services.

2.11.2. ELTE Testbed Components Set-Up

2.11.2.1. End-to-End Security Management

The End-to-End Trust Management in ELTE testbed is designed to provide a realistic and modular platform for evaluating secure authentication and authorization mechanisms for IoT devices within a 5G environment. The testbed integrates core network functions, radio access emulation, a programmable data network, and a blockchain-based trust layer to enable comprehensive end-to-end experimentation. The testbed architecture supports controlled evaluation of device registration, data routing, and secure service access while maintaining compatibility with standard 5G protocols. Key components of the testbed include:

5G Core (**Open5GS**): The core network is implemented using Open5GS, providing essential control-plane functions such as the Access and Mobility Function (AMF), Authentication Server Function (AUSF), and Unified Data Management (UDM). These network functions manage authentication, session establishment, and mobility for IoT devices, ensuring secure and reliable connectivity.

User Plane Function and Data Network (**Open5GS**): The UPF, also implemented via Open5GS, handles user-plane traffic and interfaces with the Data Network. The DN includes an HTTPS-based service provider that receives and responds to IoT device traffic, enabling secure end-to-end data delivery. This separation between control and data planes allows precise measurement of traffic flows and authentication performance.

UE (**UERANSIM**): The UE emulates IoT device functionality. In the physical testbed, a Raspberry Pi running UERANSIM represents an actual IoT node, generating realistic traffic patterns and triggering registration and authentication flows with the 5G Core.

gNB (**UERANSIM**): The gNB represents the radio access node and is emulated using UERANSIM. It establishes the connection between the UE and the 5G Core, handling signaling, session setup, and data forwarding, providing a realistic representation of the radio access network.

Blockchain Layer (**Foundry**): The testbed integrates a permissioned Ethereum blockchain implemented with Foundry, which supports smart-contract-based operations for end-to-end trust establishment. The blockchain records pseudonyms, access policies, and authentication metadata, enabling decentralized verification and reducing reliance on centralized identity databases.

2.11.2.2. Data plane ML

The Data Plane ML validation was carried out on the dedicated testbed hosted at ELTE. The test environment integrates programmable hardware, control infrastructure, and traffic generation capabilities, providing a realistic and reproducible setting for experimentation.

The architecture is illustrated in Figure 22 and consists of the following components:

Two Servers: These provide the control, orchestration, and monitoring layer of the testbed. Their functions include:

- Deploying and updating ML models into the data plane.
- Acting as the control plane, distributing classification rules and managing runtime configurations.
- Generating test traffic for validation.
- Running eBPF-based modules, which can host ML model execution for packet classification and provide fine-grained monitoring.
- Logging classification metadata and collecting performance statistics.

Server Specification:

- **Operating System:** Ubuntu 20.04.6 LTS
- **CPU:** AMD Ryzen Threadripper 1900X, 8 cores / 16 threads, 2.2–3.8 GHz
- **RAM:** 128 GB DDR4
- **Network:** Two Mellanox MT27700 ConnectX-4 Lx 25GbE NICs

Two Intel Tofino Switches: Serving as the programmable data-plane hardware, the switches execute P4 pipelines extended with ML-based classification capabilities. The ML model can be deployed directly on Tofino for line-rate packet classification.

Flexible ML Deployment: The testbed supports running the ML model on the Tofino switches and/or within the servers using eBPF. This allows evaluation of both hardware-accelerated line-rate classification and software-based processing.

Traffic Generation and Monitoring Tools: Deployed on the servers, these tools inject diverse traffic patterns into the network and capture experiment data, ensuring comprehensive visibility of system behavior under benign and malicious traffic conditions.

Operational Workflow:

1. ML-enhanced P4 pipelines are compiled and deployed on the Tofino switches or ML modules are loaded via eBPF on the servers.

2. The servers generate test traffic and run monitoring modules to collect metadata and performance metrics.
3. The control plane dynamically updates the ML models and classification rules as required.
4. The data plane classifies incoming packets and enforces the defined actions in real time.
5. Monitoring infrastructure logs performance data, classification decisions, and robustness outcomes.

This testbed combines hardware-accelerated packet processing on Tofino with flexible eBPF-based ML execution on servers, providing a versatile platform to validate the Data Plane ML component under multiple deployment scenarios.

2.11.2.3. Secure Data Aggregation

The Secure Data Aggregation as part of the ELTE testbed provides a controlled environment to evaluate privacy-preserving and distributed machine learning techniques for IoT networks. It is designed to support the aggregation of data from IoT devices while maintaining data confidentiality and minimizing centralized exposure, enabling experiments on federated learning and secure model training.

The core of the testbed consists of a Flwr node, which orchestrates federated learning workflows and coordinates model updates across connected IoT devices. The node manages training rounds, aggregates local model parameters, and enforces secure communication between the central aggregator and participating devices. This configuration allows testing of both algorithmic performance and system-level behaviors, including latency, scalability, and resilience under realistic workloads. Key aspects of the Flwr node include:

Flwr main server: Acts as the central coordinator for federated learning and secure aggregation, managing training rounds, orchestrating updates from normal clients, and coordinating MPC nodes for privacy-preserving combination of model parameters.

MPC Nodes: Perform secure computations on local model updates from clients, ensuring that individual data or model parameters are never exposed while contributing to the overall aggregated model.

Normal Clients: Represent typical IoT devices generating local data for training. They participate in the federated learning process by providing local updates that are securely aggregated via MPC nodes.

2.12. ZHAW Testbed Components Set-Up

2.12.1. ZHAW Testbed Infrastructure

This is a local testbed used in the initial phase of development and testing of the components of the AI-based MTD framework, namely: the MTD Controller, the MTD strategy optimizer, MTD Explainer, and MTDFed. Depicted in Figure 23, the testbed comprises a core and edge cloud domain, both set up with Openstack. Each domain configures three VMs: one master node and two worker nodes, forming a Kubernetes cluster. These are where the CNFs are running, with the MTD controller enabling the migration between the clusters.

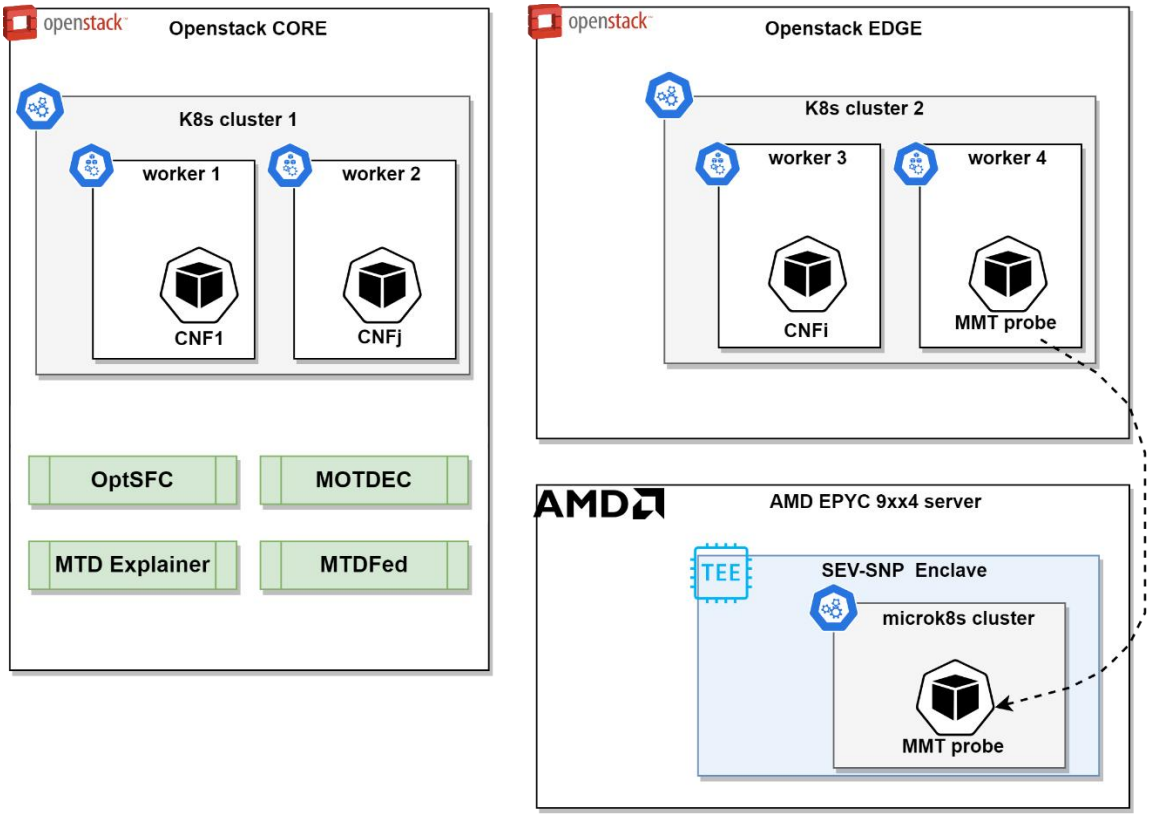

Figure 23 ZHAW local testbed for AI-based MTD framework implementation and testing.

ZHAW's testbed additionally runs an AMD EPYC 9xx4 server for trusted execution environments (TEE), as it is capable of creating SEV-SNP enclaves (Secure Encrypted Virtualization-Secure Nested Paging), distinctive TEE VMs that run with encrypted RAM and CPU registries [1]. Such SEV-SNP VM is used in the testbed and hosts a microk8s Kubernetes node, enabling the MTD Controller's MTD action of TEE encapsulation and decapsulation by transferring the CNFs from the other clusters to microk8s and vice-versa.

2.12.2. ZHAW Testbed Components Set-Up

The components of the AI-based Framework are all developed and tested in this testbed, with further integration in the PNET testbed to operate with a 5G core and relative network functions provided by Open5GS. The components are hosted in the OpenStack core domain, reflecting their position in the Patras 5G testbed.

2.12.2.1. MTD Controller

The MTD Controller maintains the same set-up as described in 2.7.2.1.

2.12.2.2. MTD Strategy Optimizer

The MTD Strategy Optimizer maintains the same set-up as described in 2.7.2.2.

2.12.2.3. MTD Explainer

The MTD Explainer maintains the same set-up as described in 2.7.2.2.3.

2.12.2.4. MTDFed

The MTDFed maintains the same set-up as described in 2.7.2.2.4.

2.13. HES-SO Testbed Components Set-Up

HES-SO is building a testbed that targets two different attacks at the same time: jamming detection and DDoS malicious traffic in a wireless environment. There is a third component addressed which is the Mirai botnet setup which will allow us to create legit malicious traffic for training and validation purpose.

2.13.1. HES-SO Testbed Infrastructure

The experimental infrastructure (Figure 24) is a complete end-to-end 5G testbed comprising a 5G Core Network (5GCN), a 5G Radio Access Network (RAN), and User Equipment (UEs) generating both benign and malicious IoT traffic. The 5GCN is deployed using containerized Open5GS (i.e., Network Slice Selection Function (NSSF), Network Exposure Function (NEF), Network Repository Function (NRF), Policy Control Function (PCF), Unified Data Management (UDM), Application Function (AF), Authentication Server Function (AUSF), Access and Mobility Management Function (AMF), User Plane Function (UPF), Service Communication Proxy (SCP)), while the RAN is provided by a containerized srsRAN project gNodeB. Radio transmission is handled by USRP B210 software defined radios, which serve as both the gNB transceiver and as programmable radio sources for jamming experiments.

Commercial Samsung S23 smartphones, equipped with Osmocom programmable SJA5-9FV SIM cards configured for the local Open5GS network, act as UEs and provide IP connectivity to attached Raspberry Pi (RPI) devices via USB-tethering. The gNB and UEs currently operate in the

N77 band with 20 MHz channel bandwidth. All core components–including Open5GS, srsRAN, the jamming module, traffic capture, and machine-learning pipelines –are deployed as Docker containers. The jamming module is also implemented as a container controlling a dedicated USRP B210 through GNU Radio, at present producing a single-carrier interference signal with tunable frequency and power to disrupt the 5G NR link between the UE and the gNB. Each RPI hosts a set of Docker based containers that act as traffic sources or sinks.

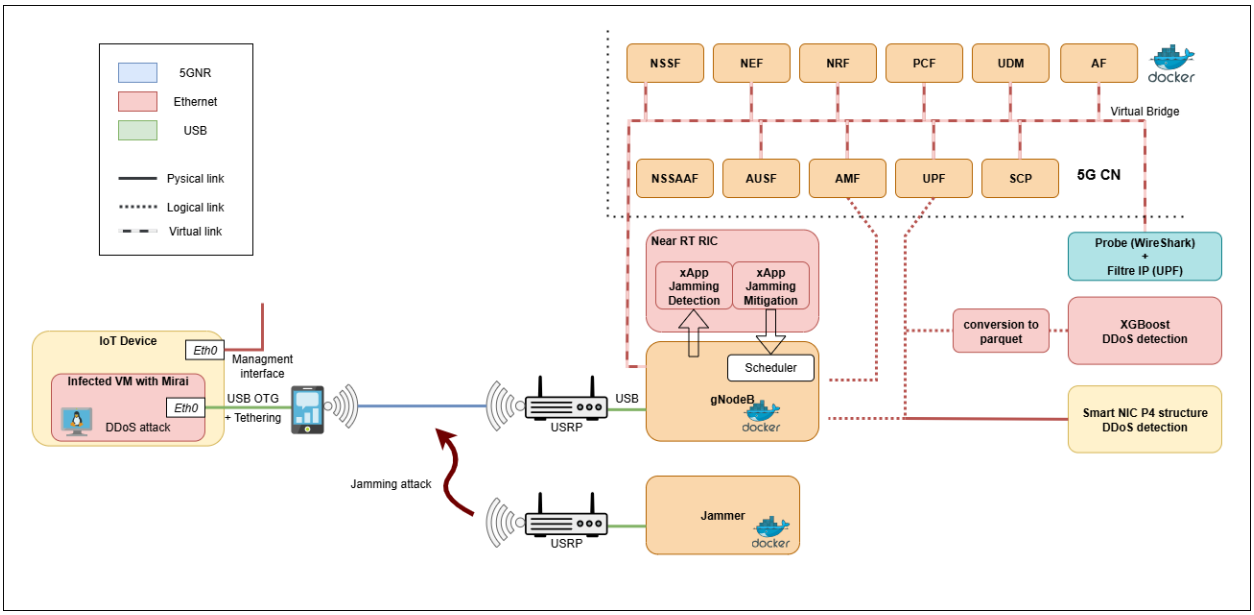

Figure 24 HES-SO full testbed.

We currently support representative benign traffic patterns MQTT by mosquitto, User Datagram Protocol (UDP) with Python socket programming, HyperText Transfer Protocol (HTTP) with nginx/curl, and video streaming with mediamtx/ffplay–as well as an isolated Mirai-infected VM that generates realistic botnet traffic. The benign services include both clients and servers, whereas the Mirai VM only produces malicious flows. All benign traffic services are deployed as lightweight containers, which are directly connected to the test interface exposed by the tethered smartphone and communicate over the same network stack as ordinary IoT endpoints. In contrast, the malicious Mirai instances are executed within fully isolated Quick Emulator (QEMU)-based VMs to ensure an additional layer of containment and prevent any unintended propagation of the malware. The malware VMs are managed through a dedicated Malware Management and Control (MMC) infrastructure, composed of mmc-vm-daemon, mmc-host daemon, and mmc-cli. Control commands for the malware are exchanged via AF_UNIX sockets on the RPI host side and serial ports on the QEMU guest side, enabling fine grained orchestration while maintaining secure separation of malicious code. Traffic of interest for DDoS detection is captured at the N3 interface between the gNB and the UPF, which carries the user-plane flows generated by both benign containers and malicious VMs. Captured packet traces are processed

by CICFlowMeter, which aggregates raw packets into bidirectional flow records enriched with statistical features such as flow duration, inter-arrival times, byte and packet counts, and burst metrics. These feature-rich flow records are stored in the Parquet format16 and consumed by a dedicated container running Python-based XGBoost models. At present, both training and inference are performed offline. However, the pipeline is designed to first operate online at the edge and, in a subsequent phase, to be offloaded to a P4-programmable SmartNIC to enable line-rate inference for faster detection and mitigation.

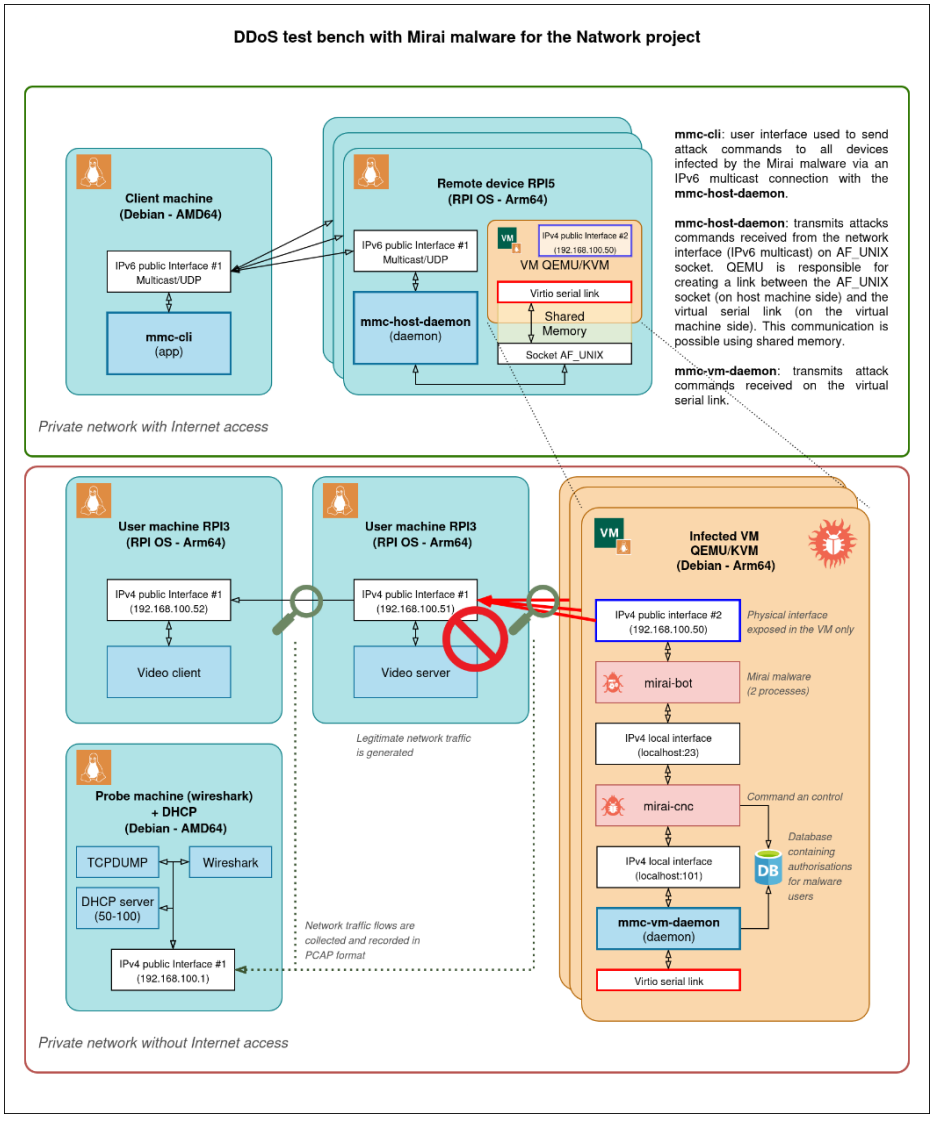

Figure 25 Mirai Malware Control Mechanism

The gNB is also being prepared for future integration with a near-real-time O-RAN RIC, which will enable the deployment of xApps capable of monitoring radio-level metrics to detect jamming in real time and provide control directives back to the gNB scheduler. This planned integration will

allow the system to detect and react to both jamming and DDoS attacks at the gNB in real time, blocking malicious traffic and mitigating radio-level disruptions capabilities particularly relevant for IoT-dominated 5G deployments. Nevertheless, in its current state, the testbed is already able to gather data, train models, and perform offline detection of attacks, providing a robust experimental platform for end-to-end evaluation of 5G-enabled IoT security solutions.

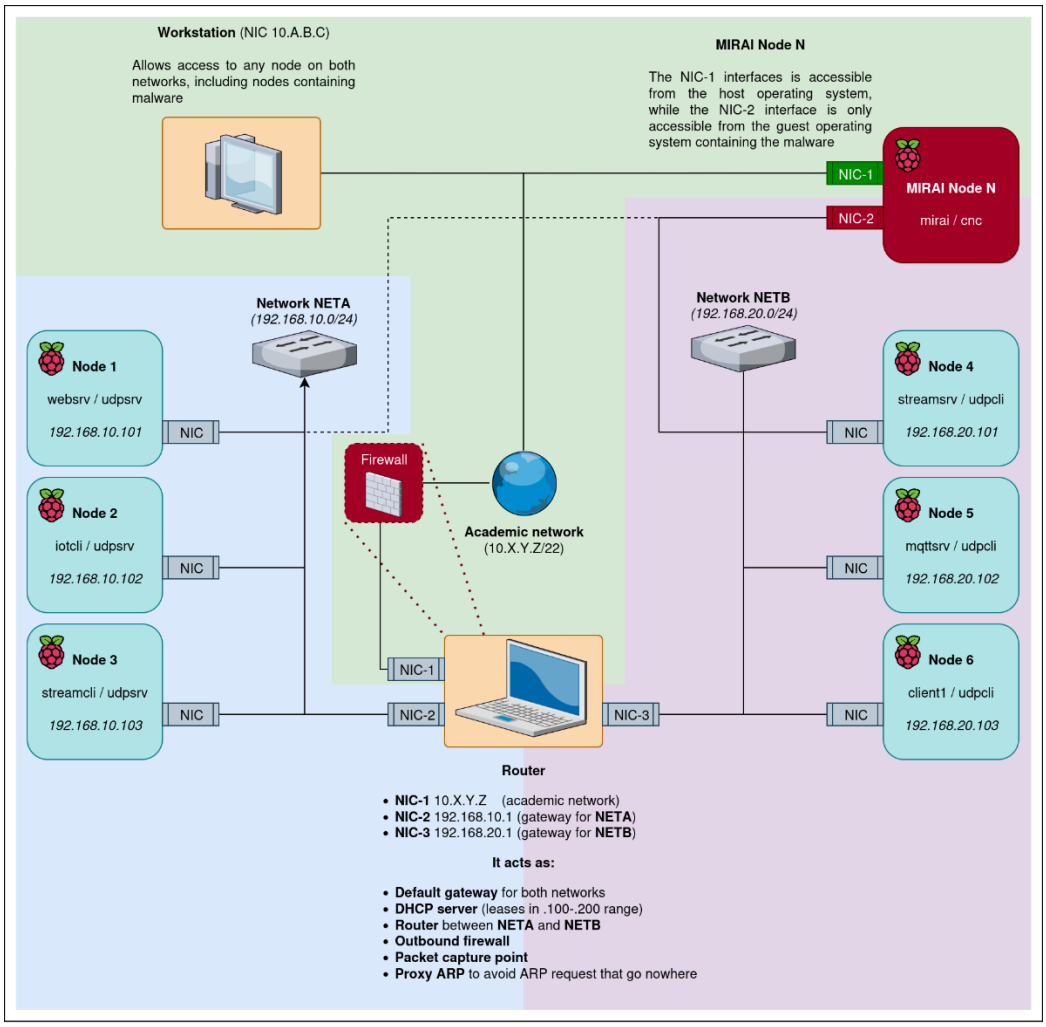

Figure 26 HES-SO Network testbed architecture for Mirai botnet attack generation.

2.13.1. HES-SO Testbed Components Set-Up

As previously mentioned, at HES-SO we are working on a testbed that addresses two attacks simultaneously with two different natures. The very first is to install the Base Station into the computer. This is done by using “docker compose build”. The way we define the yaml file allows us to build the image and prepare the containers all at once. In the second step you need to do the same with the jammer, which is also in the form of a container. Currently, every time we have a modification on the jammer, GNU Radio generates a python script, and we need to rebuild the

container, which is very fast at this point. Currently Near RT RIC is in place but not yet tested. Regarding Mirai-based botnets, further explanation was provided earlier. For the physical setup, we use Raspberry Pi units; on each Pi we install the VM developed for research purposes. Some Raspberry Pis are dedicated to generating benign (healthy) traffic.

In Figure 26 the complete network setup is outlined, where a firewall is in place to not block any possibility of infection towards other equipment's. This approach collects the data stream for training purposes, which will subsequently be deployed on a SmartNIC.

2.14. UZH Testbed Components Set-Up

The UZH testbed provides a reproducible, end-to-end environment to evaluate anomaly detection and explainability in realistic 5G scenarios. The platform consists of two main virtual machines: a Control VM running the free5GC control-plane functions and a Data VM that hosts the user-plane, traffic generation, and packet capture. Radio access is emulated with a software gNodeB and UE using UERANSIM. Traffic flows from UE through gNodeB over N2/N3 into the core (AMF/SMF/UPF), and mirrored packets on the N3 interface feed the IDS/XAI backend. Inference results and explanations are exposed via a REST API to a lightweight dashboard for operators.

2.14.1. UZH Testbed Infrastructure

The infrastructure follows the logical layout in the figures. The Control VM runs free5GC with NRF, NSSF, PCF, NEF, UDM, AUSF, AMF, and SMF. The Data VM hosts UPF and a small mininet topology (h1 for gNB/UE, h2 for UPF, s0 switch, r0 router) to create an isolated GTP-U path toward the core. A port mirror (SPAN/TAP) on the N3 segment continuously captures GTP-U (UDP/2152) and PFCP (UDP/8805) traffic using tcpdump/tshark; optional IPFIX export (nProbe) is available for flow-level analytics. Typical addressing uses separate subnets for N2, N3, and N6; routing is configured so that PDU sessions established by SMF traverse the UPF and out to a data network. Hardware requirements are modest (8–16 vCPU, 32–64 GB RAM, SSD), and all components run on Ubuntu 22.04. The setup supports both benign workloads (ping/HTTP/iperf) and controlled attack scenarios.

2.14.2. UZH Testbed Components Set-Up

Deployment proceeds in three steps. First, free5GC is installed on the Control VM and the control-plane NFs are started; SMF policies and slice parameters are registered in NRF/PCF. Second, the Data VM brings up the mininet topology, starts the UPF, and launches UERANSIM so that the gNB attaches to AMF (N2) and the UE registers and creates a PDU session (N3). Third, data capture and analytics are enabled: mirrored N3/PFCP traffic is written to PCAP (or exported as IPFIX) and passed into the IDS/XAI pipeline. The backend performs parsing and feature extraction (flow

construction with 5-tuple, TEID/QFI, temporal windows; statistics such as packet/byte rates, inter-arrival variance, UL/DL ratios, entropy, and PFCP/NAS counters), then executes binary and multi-class models (Random Forest, XGBoost, CNN/DNN). A simple decision gate provides anomaly scores and labels. Results and explanations are served through /predict and /explain endpoints and visualized on a dashboard showing alert rates, confusion matrix, and feature-importance distributions.

2.14.2.1. Anomaly Detection Explainer

The Explainer component generates human-readable reasons for each flagged anomaly. For tree-based models it uses SHAP TreeExplainer; for neural models it employs KernelSHAP/LIME. Each inference returns the top-k contributing features with sign and magnitude, which are then summarized by a small LLM into domain-aware narratives. Explanation quality is tracked with four KPIs: faithfulness (performance drop under feature deletion), robustness, complexity, and latency. To prevent misleading attributions, the testbed includes adversarial "bad-tests": leakage traps (ensuring explainers do not focus on non-causal identifiers such as IP pools), randomization checks (weights/labels shuffled should yield structureless attributions), micro-drift repeats, and out-of-distribution runs across slices or time-of-day.

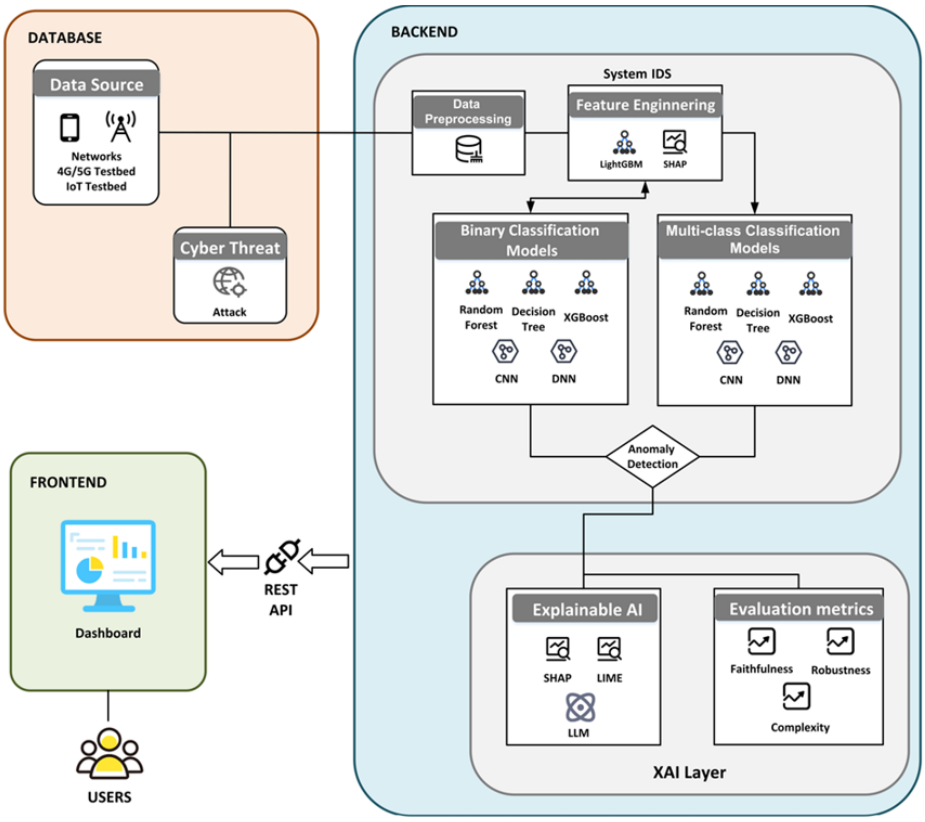

Figure 27: Anomaly Detection Explainer.

Overall, the UZH testbed offers a controlled, programmable, and repeatable environment that links UE/gNB, free5GC core, UPF, data capture, ML-based IDS, and XAI. It enables rigorous evaluation of detection accuracy alongside explanation quality so that operators can understand, trust, and act on anomaly alerts in 5G networks.

3. Dry Run Tests for NATWORK Components

In this section, the validation of the 43 components of the NATWORK project is illustrated. With these sets of dry run test instructions, the component owners verified or will verify in the upcoming period the functionality of the components. Component owners had a close collaboration with testbed owners to identify the infrastructure of the testbed(s) on which the related component had been installed in. Components that were installed in more than one testbed have a shared (single) test report. In D6.3, second version of "System Integration on the testbeds, Pilot installations and implementations", a full validation of the components will be reported.

In the table below, information on the components, the test scenarios and related test cases of each component, and the status of the dry run tests is presented.

Table 2: Components and related information of the dry run tests

#	Component	Test Scenarios / Test Cases	Dry run test status
1	Energy efficient over edge-cloud	Appendix A.1	Partially Tested
2	TrustEdge	Appendix A.2	Fully Tested
3	Feather	Appendix A.3	Fully Tested
4	Flocky	Appendix A.4	Fully Tested
5	Secure-by-design orchestration	Appendix A.5	Fully Tested
6	End-to-End Security Management	Appendix A.6	Fully Tested
7	Slice orchestration and slice management for beyond 5G networks	Appendix A.7	Fully Tested
8	AI-Based RIS configuration	Appendix A.8	Not tested yet
9	ML-based MIMO	Appendix A.8	Not tested yet
10	JASMIN & Filter Mitigation	Appendix A.8	Partially Tested
11	DetAction: Detection and reAction against jamming attacks	Appendix A.9	Fully Tested
12	Security-compliant Slice Management	Appendix A.10	Fully Tested
13	Multimodal Fusion Approach for Intrusion Detection System for DoS attacks	Appendix A.11	Partially Tested
14	Lightweight SDN-based AI-enabled Intrusion Detection System for cloud-based services	Appendix A.12	Fully Tested
15	AI-enabled DoS attack	Appendix A.13	Fully Tested
16	Multiagent AI based cybersecurity support system	Appendix A.14	Partially Tested

#	Component	Test Scenarios / Test Cases	Dry run test status
17	Data plane ML	Appendix A.15	Fully Tested
18	Wire-speed AI (WAI) and Decentralized Feature Extraction	Appendix A.16	Fully Tested
19	Microservice behavioral analysis for detecting malicious actions	Appendix A.17	Partially Tested
20	MTD Controller	Appendix A.18	Partially Tested
21	MTD Strategy Optimizer	Appendix A.19	Fully Tested
22	MTD Explainer	Appendix A.20	Not tested yet
23	AI-driven security monitoring for anomaly detection and root cause analysis	Appendix A.21	Partially Tested
24	Security-performance balancer	Reporting Period 2	Not tested yet
25	DFE Telemetry	Appendix A.22	Fully Tested
26	Secure Data Aggregation	Appendix A.23	Fully Tested
27	Federated Learning for edge-to-cloud	Appendix A.24	Partially Tested
28	MTDFed	Appendix A.25	Fully Tested
29	CIA-hardening of x86 payloads Component	Appendix A.26	Partially Tested
30	CIA-hardening of containerized payloads	Appendix A.27	Not tested yet
31	CIA-hardening of WASM payloads Component	Appendix A.28	Partially Tested
32	JDM-xApp	Reporting Period 2	Not tested yet
33	Liquid RAN	Appendix A.29	Not tested yet
34	Liquid Near-RT RIC	Reporting Period 2	Not tested yet
35	KPM xApp	Reporting Period 2	Not tested yet
36	Characteristics Extractor	Appendix A.30	Fully Tested
37	Key Generator	Appendix A.31	Fully Tested
38	Security Evaluator	Appendix A.32	Fully Tested
39	AI -Based Anomaly Detection Explainer	Appendix A.33	Not tested yet
40	Wirespeed traffic analysis in the 5G transport network	Appendix A.34	Fully Tested
41	Detection and mitigation against jamming attacks	Appendix A.35	Partially Tested

#	Component	Test Scenarios / Test Cases	Dry run test status
42	Setting up of a Mirai botnet	Reporting Period 2	Not tested yet
43	FPGA-based hardware detection of DDoS attacks	Reporting Period 2	Not tested yet

In the following sub-sections, directions on how the related dry run tests were performed or will be performed, are presented. Additional information on the actual tests, including the test scenarios / test cases and the results for the components on which initial dry run tests were performed can be found in the Appendix.

3.1. Energy efficient over edge-cloud

The test procedures and results are recorded in the attached Excel sheet UEssex –Energy-efficient.xlsx in Appendix A.1.

3.1.1. Test procedures / Test cases

CPU utilization and energy consumption (TS01)**:** Benign workloads (TC01) were tested in single- and multi-cluster setups, showing RTT variance due to MCS API overhead.

Adversarial workloads (TC02) reproduced DoST attacks with oscillatory HTTP traffic, causing CPU/memory oscillations, degraded QoS, and increased RTT while keeping services alive. The DoST attack demonstration has been successfully conducted; however, mitigation strategies are still under development in cohernece with Section 3.5 (Secure-by-Design Orchestration) and Section 3.27 (Federated Learning for Edge–Cloud). These mitigation mechanisms have not yet been validated within the testbed and will be integrated and evaluated in subsequent phases.

Monitoring and dataset generation (TS02): Prometheus and Grafana were used to monitor CPU, memory, network, and pod lifecycle metrics during benign and DoST workloads. Time-series data were stored in the Prometheus TSDB and persisted in MinIO, preparing structured datasets for federated learning and anomaly detection.

These tests confirm the impact of DoST on CNF services and establish a dataset pipeline for AI-driven slice management.

3.2. TrustEdge

3.2.1. Test Procedures / Test Cases

The full set of test procedures and results is recorded in the attached Excel sheet IMEC-TrustEdge.xlsx in Appendix A.2. The test was performed in April 2024 and successfully passed.

Startup time: Measures the (added) boot time of the framework from attestation to secure Feather deployment. The average case 20.91s time adds to boot time, around half of which is Feather starting and half is the attestation process

3.3. Feather

3.3.1. Test Procedures / Test Cases

The full set of test procedures and results is recorded in the attached Excel sheet IMEC-Feather.xlsx in Appendix A.3. The tests were performed at various dates from mid-2024 to early 2025 depending on implementation milestones. All tests have been successfully passed.

The first group of tests involves **runtime comparisons** between Docker containers, OSv unikernels, and WASM workloads in WASMTime when deployed through Feather:

Minimal load: Measures the memory overhead of Feather when initializing container, unikernel and WASM backends and deploying an idle workload. The resulting overhead was minimal, and the lowest for WASM (WASMTime).

Application: Measures the overhead of Feather with active containers and unikernels. Measures the resource consumption of a Minecraft server in both container and unikernel format to gauge benefits of runtimes. The results show a CPU penalty for unikernels, while using significantly less memory than a container.

Image size: Measures the relative size of an image for specific functionality (HTTP server) in different runtime formats. WASM resulted in the smallest images, followed by the OSv unikernel and the container.

HTTP performance: Measures performance of an HTTP server in various runtimes. Considers latency as well as raw request throughput using k6 command. The results show a small but significant latency overhead for WASM, while keeping pace with container throughput. OSv unikernel performance was an order of magnitude worse on KVM/Qemu, despite older tests showing good results on XenServer.

The second test suite concerns the performance parameters of **decentralized and multi-runtime networking**:

Decentralization throughput: Measures throughput of the decentralized point-to-point internode part of the networking solution compared to WireGuard. The solution was able to saturate a Gbps physical connection with 10-100 times lower CPU use than WireGuard depending on send/receive and protocol.

Decentralization scalability: Measures throughput of the decentralized point-to-point internode part of the networking solution in 5-node star and ring topologies to evaluate scalability. In the ring topology, WireGuard uses one of the nodes as VPN controller. The results indicate that network throughput depends only on P2P connected nodes, whereas WireGuard performance depends on total topology size.

Multi-runtime throughput: Measures throughput of the multi-runtime (node local) part of the networking solution using a video streaming scenario. The results show a sustained 2.5Gbps throughput using just 1% of a single CPU core, independent of the runtimes used. Furthermore, the solution adds only 10-100µs end-to-end latency compared to Linux-native container-to-container traffic.

3.4. Flocky

3.4.1. Test Procedures / Test Cases

The full set of test procedures and results is recorded in the attached Excel sheet IMEC-Flocky.xlsx in Appendix A.4. The tests were performed in February 2025 and all successfully passed. To summarize, various aspects of the solution framework are gauged:

Functional evaluation: The functionality of the framework is measured in terms of metadata discovered (discovery + metadata services) and required services deployed (orchestration metadata use). Results are as expected with 100% discovery, and all services successfully placed after 2-3 rounds (depending on random factors).

Resource scalability: CPU and memory are measured for topologies from 1 to 150 nodes, for discovery distances from 10 to 20 (ms ping simulated). Memory scales as expected, with very low overhead compared to baseline (16MB base to 21MB at the densest scenario). CPU scaling exactly follows the number of neighbours.

Network scalability: Network traffic is measured for topologies from 1 to 150 nodes, for discovery distances from 10 to 20. Network traffic scales linearly with the number of neighbours, but rises twice as fast as CPU scaling (8x traffic for 4x CPU and 4x neighbours).

Discovery accuracy: Metadata discovery efficiency is measured when each node is assigned 2 random metadata items at start, and a total pool of 100 must be discovered by each node. Results show >99% metadata discovery from 75 nodes and 20 discovery distance upwards, and >96% for smaller, loosely connected topologies. This indicates

single nodes not connected to the topology due to random generation (trivial to fix with starting conditions and configuration for nodes).

Deployment latency: Measures the total deployment latency (end to end) of a two-component service in the Flocky framework, from user action to service deployment. Median case shows 21.1ms response time for deployment on two separate nodes, consisting of ~70% network latency.

3.5. Secure-by-design orchestration

The test procedures and results are recorded in the attached Excel sheet UEssex – secure-by-design-orch.xlsx in Appendix A.5.

3.5.1. Test Procedures / Test Cases

ORCH-TS01 (Security-compliant orchestration): Slice and cluster requirements were defined declaratively with security constraints. Orchestration was triggered, and placement/scaling decisions were monitored to verify compliance. Results confirm that no insecure placement occurred, demonstrating adherence to secure-by-design policies.

ORCH-TS02 (Subgraph communication): Multi-CNF slices with dependency graphs were created. The sFORK policy and strategy components decomposed slice dependencies into subgraphs, which were distributed to local cluster agents. Subgraphs were executed correctly across clusters, confirming proper communication and dependency handling.

These tests validate that sFORK respects slice-level security requirements and enables coordinated orchestration across distributed clusters.

3.6. End-to-End Security Management

The full set of test procedures and results is recorded in the attached Excel sheet ELTE-E2E-Trust.xlsx in Appendix A.6. All tests were executed between August and November 2024, and every case completed successfully.

3.6.1. Test Procedures / Test Cases

Core Connectivity: The 5G Core was started and verified to be fully operational. The gNB connected to the Core via UERANSIM, and the UPF established a stable link through Open5GS. Log files on both ends confirmed successful attachment and session establishment.

End-to-End UE Path: The UE connected through the configured gNB, and the connection propagated correctly through the 5G Core to the UPF and Data Network (DN). The full data path from UE → gNB → Core → UPF was validated, confirming correct forwarding and control-plane behavior.

Blockchain Node Deployment: A Foundry blockchain node was deployed locally after installing Rust, Node.js, and the Foundry toolchain. Health checks using CLI tools (forge test, cast block-number) verified correct operation. Dummy transactions were processed successfully, confirming proper block generation and transaction handling.

Blockchain Integration with 5G Components: The UPF communicated successfully with the blockchain node running in the DN, performing data path and transaction validation. The 5G Core also initiated blockchain transactions and received confirmation responses, demonstrating reliable two-way interaction between the 5G and blockchain layers.

All procedures have been executed as defined in the test cases (E2E_Trust-TS.01–TS.03). No anomalies, disconnections, or processing errors were observed. The integrated E2E Trust component is verified to be stable, interoperable, and ready for full system integration.

3.7. Slice orchestration and slice management for beyond 5G networks

A comprehensive record of the test procedures and their outcomes is provided in the attached Excel file, CERTH-Slice-orchestration-and-management.xlsx in Appendix A.7. All tests conducted to date have been completed successfully.

3.7.1. Test Procedures / Test Cases

A series of tests were conducted under the test scenario Slice-orchestration-management-TS01, aimed at verifying the proper functionality of the NATWORK Slice orchestration and slice management for beyond 5G networks component integrated within a 5G network. The first test case (**TC01**) validated the default behavior of the xAPP under normal network conditions, where traffic generated by a default traffic generator was correctly classified as benign. In **TC02**, the xAPP's detection capabilities were evaluated using malicious traffic from the KDD Cup 1999 dataset, with results confirming accurate classification of attack traffic. Building upon this, **TC03** assessed the xAPP's ability to respond adaptively by reallocating Physical Resource Blocks (PRBs) to limit the impact on slices under attack, demonstrating successful reallocation based on computed anomaly ratios. Finally, **TC04** tested the system's mitigation capability by disconnecting malicious User Equipment (UE) when the anomaly ratio reached 100%, effectively removing the threat from the network. All test cases between **October 2024 and May 2025** were executed successfully, and the expected outcomes were achieved.

3.8. AI-Based RIS configuration

Test scenarios for this component can be found in the attached CERTH-Signal Processing.xlsx in Appendix A.8.

3.8.1. Test Procedures / Test Cases

The goal of this procedure is the determination of the RIS configuration for multi-user scenarios. Procedure consists of the following steps:

- **Step 1:** The receiver and the transmitter will be positioned in Line-of-Sight with the RIS unit.
- **Step 2:** The communication link quality will be measured with the RIS unit out of function in order to use this measurement as baseline.
- **Step 3**: The communication link in case that the user is served standalone will be measured using the optimal RIS configuration.
- **Step 4:** The communication link quality in the multi-user scenario will be measured using the codebook entries multiplexing algorithm for fair beam-splitting.

The prerequisite information is the optimal RIS configurations for the case that each user is served standalone by it. The procedure demands data only extracted by the RIS-testbed. The outcome is the performance per user in multi-user scenario. The current status is that the setup has been prepared and preliminary results are extracted. Mainly, the proposed codebook entries multiplexing algorithms have been initially compared with traditional RIS sharing approach such as segmentation of RIS unit for multiple users and time division multiple access. The evaluation approach will be completed the next period-planned for early 2026.

3.9. ML-based MIMO

ML-based MIMO test procedures are in the attached CERTH-Signal Processing.xlsx in Appendix A.8.

3.9.1. Test Procedures / Test Cases

The procedure described on JASMIN & Filter Mitigation will be evaluated in the case where receiver or/and jammer is equipped with MIMO antennas. In this case, synchronization issues and benefits from MIMO usage will be addressed. The evaluation has been planned for mid-2026, in the period that the evaluation of JASMIN and Filter Mitigation will have been completed for SISO case.

3.10. JASMIN & Filter Mitigation

The test scenarios and the initial test results for the present component appear in the attached CERTH-Signal Processing.xlsx in Appendix A.8.

3.10.1. Test Procedures / Test Cases

The goal of this procedure is the detection of the jamming attack across all main types (constant, periodic, reactive) in real-time within IEEE 802.11p. protocol. Procedure consists of the following steps:

For JASMIN evaluation

- **Step 1:** The dedicated protocol for V2X, IEEE 802.11p, will be simulated in the SDR-based setup.
- **Step 2:** One SDR will be used as a transmitter, one as a receiver and one as a jammer.
- **Step 3:** JASMIN model will be connected with the receiver.
- **Step 4:** The output of JASMIN will be measured in case the jammer is inactive.
- **Step 5:** The output of JASMIN will be measured in case the jammer is active.
- **Step 6:** The outputs in both cases will be evaluated based on the ground truth

For Filter Mitigation

- **Step 1:** The dedicated protocol for V2X, IEEE 802.11p, will be simulated in the SDR-based setup.
- **Step 2:** One SDR will be used as atransmitter, one as a receiver and one as a jammer.
- **Step 3:** The information from the receiver will be directed identically in two sinks; the jammed and the mitigated.
- **Step 4:** In the mitigated sink, the respective filter will be applied and the clear from the attack signals will be stored.
- **Step 5:** For the evaluation, the content of the sinks will be compared.

The prerequisite for mitigation is the perfect synchronization between two SDR ports, a technical task that is ongoing. The procedure demands data only extracted by the SDR-based testbed. The accuracy of the JASMIN model will be evaluated in two cases; clear and jammed signal for an overall evaluation of the model's performance. The mitigation will be evaluated comparing the SNR values before and after the usage of the filter mitigation. The evaluation of JASMIN has been completed. The overall accuracy is 99.92% and is passed the test successfully. For the mitigation part, the evaluation will be completed in early 2026.

3.11. DetAction: Detection and reAction against jamming attacks

3.11.1. Test Procedures / Test Cases

The full details of the test procedures and cases for this component are provided in the attached file **GRAD-DetAction.xlsx** in Appendix A.9, which includes parameters, steps, and dates. All dry-run tests on the component have been successfully completed.

Signal preprocessing validation: The signal acquisition and preprocessing pipeline (resampling, frequency transformation, spectrum fragmentation, and normalization) have been verified.

Detection phase classification validation: The detection phase has been tested in an inference scenario to ensure that the same steps used during training and validation can be executed without errors during testing.

ReAction PRB assignment verification: The reAction algorithm has been validated in a simulation scenario, confirming the allocation of PRBs to UEs, both in the presence and absence of jamming in certain PRBs.

Connection between Detection and ReAction phases verification: The interaction between the detection and reAction phases has been tested in a simulation scenario and confirmed to work as intended.

3.12. Security-compliant Slice Management

The test procedures and results are recorded in the attached Excel sheet UEssex –CTI.xlsx in Appendix A.10.

3.12.1. Test Procedures / Test Cases

CTI-TS01 (CTI exchange in multi-cluster environments): Two Kubernetes clusters with vulnerability scanners and CTI agents were deployed. Applications with different vulnerability profiles were scanned, and bidirectional CTI sharing was enabled. Results confirmed that each cluster received tailored CTI data, with sensitive fields anonymized.

CTI-TS02 (Sensitivity/necessity mapping): Vulnerability scans from multiple clusters were processed by the CTI agent. Metadata fields were selectively anonymized or included in STIX bundles based on risk scores, necessity, and sensitivity mappings. This validated the anonymisation mechanism, with sensitive values hashed and relevant fields preserved.

CTI-TS03 (Hygiene score evaluation): Applications with varying vulnerability severities were deployed, and CTI analysis was used to compute hygiene scores. Clusters with more severe vulnerabilities were shown to have lower hygiene scores, confirming the correctness of the scoring mechanism.

These tests confirm that the CTI framework provides tailored, anonymized intelligence, supports risk-aware orchestration decisions, and delivers accurate hygiene scores for secure-by-design slice management.

3.13. Multimodal Fusion Approach for Intrusion Detection System for DoS attacks

The test procedures and results areis recorded in the attached Excel sheet CERTH-Multimodal Fusion Approach IDS.xlsx in Appendix A.11.

3.13.1. Test Procedures / Test Cases

Step 1 - Deployed 2 docker in the 5G-SDN testbed one for traffic replay and a second one containing the multimodal IDS.

Step 2 - Replay 3 pcap files from open datasets (UNSW-15,5GAD-2022, 5G-NIDD) and log the classification results of the IDS i.e. (a) Traffic Type (Anomalous/Normal), (b) Attack type if anomalous traffic was detected in (a).

Step 3 – Compare the logged results with the ground truth contained in the datasets and compare the 3 KPI described in D6.1 i.e. Probability of DoS Attack Detection, AI-based Intrusion Detection, Probability of False detection.

Results: Probability of DoS Attack Detection > 0.92 (min) in all cases, Probability of False detection < 0.11 (max) in all cases.

3.14. Lightweight SDN-based AI-enabled Intrusion Detection System for cloud-based services

The test procedures and results are recorded in the attached Excel sheet CERTH-SDN IDS.xlsx in appendix A.12. The component was tested in one scenario comprised of six steps presented in the next subsection. All dry-run tests on the component have been successfully completed.

3.14.1. Test Procedures / Test Cases

Step 1 - Deploy 3 dockers in the 5G-SDN testbed, one for attack creation (Kali Linux tools via python scripts), a second one containing the IDS tool and one for Wireshark to capture traffic.

Step 2 - Use the Apache JMeter tool for different traffic patterns and workload performance measurements monitor impact on QoS and OpenAirSim to simulate the UEs and eNB operation

Step 3 – Carry out two types of DoS attacks: (i) a UDP Flooding attack targeting the UPF component; and (ii) an SCTP Flooding attack targeting the AMF component. Log relevant details.

Step 4 – If an attack is detected, log identification of the attack, the attacker's IP, and the message sent to the SDN to mitigate this attack.

Step 5 – Verify that SDN controller has implemented mitigation

Step 6 – Check logs to see detection time.

Results: Both attacks are always detected when using ensemble of models with average time a) 4.8s when using Exponential Moving Average (EMA), b) 5.2s when using MLP DNN, c) 5.6s when using 1D-CNN, d) 5.8s when using ensemble of methods. When the attack was detected, the mitigation action was always successfully implemented in the SDN.

3.15. AI-enabled DoS attack

The test procedures and results are recorded in the attached Excel sheet CERTH-AI-enabled_DoS_attack.xlsx in Appendix A.13. The component was tested in two scenarios, specifically attacking SMF 5G component and attacking AMF 5G component. All dry-run tests on the component have been successfully completed.

3.15.1. Test Procedures / Test Cases

Scenario 1 - Attacking SMF 5G component

Step 1 - Run AI-enabled DoS attack container against SMF component of CERTH's 5G tesbed.

Step 2 - Conduct 1000 episodes in training mode.

Results:

Results follow the expectations: Exponential decline of epsilon value across episodes; Logarithmic/linear growth in rewards after exploration phase completion; Consistent growth in the total number of successful attacks across training; Total percentage of successful attacks at the end of the training process 88.2%.

Scenario 2 – Attacking AMF 5G component

Step 1 - Run AI-enabled DoS attack container against AMF component of CERTH's 5G tesbed.

Step 2 - Conduct 1000 episodes in testing mode.

Results:

Results follow the expectations: Constant and minimal value of epsilon, 0.1; Constant and maximum value of reward, 1000.

Constant and minimal value of epsilon, 0.1; Constant and maximum value of reward, 1000

3.16. Multiagent AI based cybersecurity support system

The test procedures and results are recorded in the attached Excel sheet CERTH-Multiagent_System.xlsx in appendix A.14 . The component was tested in four scenarios for different agents, comprised of multiple steps presented in the next subsection. All dry-run tests on the component have been successfully completed.

3.16.1. Test Procedures / Test Cases

Scenario 1 – E2E module test scenario

Step 1 – Deploy the multiagent AI framework in a 5G testbed containing multiple VNFs (UPF, SMF, AMF) for traffic and control-plane emulation.

Step 2 – Inject a combination of synthetic attack events (DoS, lateral movement, data exfiltration etc.) and benign traffic.

Step 3 – Log all correlation and automated response activities executed by the threat intelligence and automated response agents.

Step 4 – Compare the system's detection and mitigation performance against baseline manual incident response workflows (i.e. a human operator manually mitigating detected attacks).

Expected Results:

Threat correlation accuracy > 0.90 in all cases; average automated response time < 5s; compromised node count reduced by 5-15% compared to baseline.

Scenario 2a - Threat reporting and Insight Agent

Test Procedures / Test Cases

Step 1 – Deploy the LLM-based Threat Insight Agent with access to cybersecurity standards, incident datasets, and network context data.

Step 2 – Run evaluation using a golden dataset high-quality data, question-answer pairs derived from ISO, ENISA, NIST and ETSI references.

Step 3 – Test three prompting strategies (Zero-Shot, One-Shot, Few-Shot) and collect the generated responses.

Step 4 – Evaluate performance using four metrics: Prompt Alignment, Faithfulness, Response Relevancy, and Context Recall.

Results:

Few-Shot prompting yields >10% improvement across all metrics; Faithfulness ≥ 0.85; Response Relevancy ≥ 0.90; Context Recall ≥ 0.80.

Scenario 2b Generate Human-Readable Threat Reports and Actionable Insights

Test Procedures / Test Cases

Step 1 – Deploy the Threat Intelligence Agent in CERTHs' testbed divided into distinct zones (Core, Edge, Access).

Step 2 – Introduce multiple threat events (e.g., DDoS, lateral movement, data exfiltration) within each zone.

Step 3 – Verify that the agent correlates Indicators of Compromise (IOCs) and produces human-readable summaries for each event.

Step 4 – Assess whether generated reports include clear, actionable insights and recommendations tailored to the affected network zone or role.

Step 5 – Validate clarity and accuracy by expert review against ground-truth threat data.

Results:

The system successfully generated contextualized reports summarizing attack type, impact scope, and recommended mitigation steps for various zones of the system. Reports were judged clear and operationally relevant by cybersecurity analysts.

Scenario 3 - IoC Correlation Agent

Test Procedures / Test Cases

Step 1 – Train the SAFE-AE (Suspicious trAffic Filtering and Evaluation AutoEncoder) on normal traffic samples from multiple open and CERTH generated datasets.

Step 2 – Replay mixed normal and anomalous traffic bags in real-time through the model.

Step 3 – Identify suspicious traffic bags and feed them into the LLM for IP-level anomaly interpretation and mitigation advice generation.

Step 4 – Compare SAFE-AE performance against baseline MIL and supervised models using common detection metrics.

Results:

Accuracy = 77.75%; Precision = 82.06%; Recall = 89.58%; F1-Score = 85.66%; detection latency < 1s per bag; false alarm rate < 0.12.

Scenario 4 Coordinate with security orchestration tools

Step 1 – Deploy the Orchestration Coordination Agent equipped with an LLM interface connected to an Open Source Security Orchestration, Automation, and Response (SOAR) platform in CERTHs' testbed .

Step 2 – Simulate detected vulnerabilities and policy breaches across 5G slices (e.g., outdated services, misconfigured firewalls, improper ACLs).

Step 3 – Validate that the agent triggers SOAR-driven actions including:

a) Patching vulnerable services

b) Updating firewall configurations

c) Adjusting access control lists (ACLs)

d) Modifying slice-level security policies

Step 4 – Observe execution traceability and ensure feedback from each action is logged and reintroduced into the agent network for closed-loop adaptation.

Step 5 – Confirm that the secondary LLM generates comprehensive, human-readable documentation of all automated decisions and outcomes

Expected Results:

The Orchestration Coordination Agent effectively executed multi-step mitigation workflows through SOAR integration, maintained full action traceability, and produced detailed natural-language reports summarizing all actions taken and their network impact.

3.17. Data plane ML

3.17.1. Test Procedures / Test Cases

The full set of test procedures and results is recorded in the attached Excel sheet ELTE-Data-Plane-ML.xlsx in Appendix A.15. All tests were executed in August 2025, and every case passed successfully.

Compilation and deployment: The ML-enhanced P4 program was compiled and deployed on both the Tofino hardware target and the eBPF software backend without errors. The binaries loaded correctly, confirming portability across hardware and software environments.

Packet classification: With the ML model preloaded in the pipeline, benign traffic was consistently classified as benign, while malicious traffic samples (including portscan and DDoS flows) were reliably detected and tagged.

Control-plane integration: Model updates pushed from the control plane were successfully loaded into the data plane, and rule enforcement (e.g., dropping malicious traffic while forwarding benign flows) worked as expected.

Robustness: The pipeline handled malformed packets gracefully, classifying them as "unknown" or dropping them without any crash. When executed without a preloaded model, the system defaulted to benign classification, demonstrating stable fallback behavior.

No anomalies or deviations were observed during testing. The results confirm that the Data Plane ML component is stable, functional, and ready for integration.

3.18. Wire-speed AI (WAI) and Decentralized Feature Extraction (DFE)

3.18.1. Test Procedures / Test Cases

The preliminary test procedures and cases for DFE-WAI have been carried out to assess the correct implementation and execution of the first DFE-WAI components. They are in the attached CNIT-DFE-WAI.xlsx in Appendix A.16. The activity followed a methodology in which each test scenario was defined as a high-level functional area, further refined into individual test cases. Each test case included a set of preconditions, the required test data, a detailed description of the execution steps, and a comparison between expected and actual results in order to determine the final status. Distinct scenarios are considered for two main backends: the P4 switch running DFE+WAI and the Bluefield-2 DPU Smart-NIC running DFE.

The first scenario (i.e., DFE-WAI-TS01), concerned the verification of the P4-based deep neural network application embedded as distilled LUT cascade. This Wirespeed AI (WAI) solution is deployed on the Tofino (TNA) target. The tests confirmed the successful compilation of the program and its correct loading on the switch. Further validation showed that the switch was able to forward benign traffic on the appropriate interface while correctly discarding malicious traffic, thereby meeting the functional requirements defined at design stage.

The second scenario (i.e., DFE-WAI-TS02), focused on the compilation and containerization of a DOCA application on the target DPU. The objective was to ensure the absence of compilation errors both when running directly on the DPU and when executed inside a Docker container. The tests demonstrated that the application compiled successfully in both environments, with no errors detected during the build process.

The third scenario (i.e., DFE-WAI-TS03), targeting the DPU, addressed the runtime behavior of the DOCA application. Using GDB, the internal control flow of the application was inspected and confirmed to match the expected behavior without anomalies. Additional validation was

performed by running the application under traffic load and monitoring the DOCA Flow counters. The counters increased consistently with the traffic injected and aligned with the expected results, demonstrating that the runtime behavior of the application was correct and stable.

Across all three scenarios, the expected results coincided with the actual outcomes, and every test case achieved a “Pass” status. No deviations or unexpected issues were observed. The results of these tests confirm the functional readiness and stability of the first release of the DFE-WAI components within the tested scope. The full test descriptions, inputs, and results, are provided in the Annex, where the complete Excel test report is included.

3.19. Microservice behavioral analysis for detecting malicious actions

This test focuses on detecting malicious actions in microservices through behavioral analysis using AI/ML models. A profiling tool captures key performance metrics to establish normal behavior, after which multiple models are evaluated for binary and multiclass anomaly detection and resource prediction. Future testing procedures extend this framework to a full 5G microservice infrastructure with orchestration, SDN integration, continuous monitoring, and automated mitigation through controlled attack simulations. The tests associated with this component are described in CERTH-Microservice Behavioral Analysis for Detecting Malicious Action Component.xlsx and in appendix A.17.

3.19.1. Test Procedures / Test Cases

Step 1 - Deploy a dockerized profiling tool to monitor twelve key metrics across infrastructure, including CPU and memory usage, disk read/write throughput, network traffic, latency percentiles, and error rates, establishing a baseline of normal microservice behavior.

Step 2 - Gather real-time resource usage and performance data from all deployed microservices. Aggregate metrics to detect both gradual deviations (e.g., step increases in load) and sudden anomalies (e.g., spikes in traffic or CPU/memory usage).

Step 3 - Utilize a lightweight 1-D CNN to classify microservice behavior as Normal or Anomalous. Repeat the same step with other AI/ML models such as Multi-Layer Perceptron (MLP), Random Forest, and SVM to collect data to validate the CNN’s effectiveness.

Step 4 - For microservices flagged as anomalous, use a 1-D CNN to identify the specific anomaly type (high CPU, high memory, traffic spike, gradual load increase, high network latency) or mark it as Unknown for further inspection. Repeat the same step with other AI/ML models to collect data and compare their performance with proposed solution.

Step 5 - Perform a proof-of-concept evaluation using an open dataset to evaluate the system's ability to detect the five defined anomaly types, ensuring its' robustness and accuracy of both binary and multiclass models.

Step 6 - Utilize an RNN-based neural network to model CPU and memory consumption of microservices using portions of the open dataset, which includes measurements under normal conditions and various attacks.

Results: The profiling tool successfully captured all twelve metrics and established baseline behaviors. The lightweight 1-D CNN achieved high performance in binary classification and multiclass classification effectively distinguished between anomaly types and unknown patterns. Additionally, resource prediction using an RNN-based model demonstrated strong predictive capabilities for CPU and memory consumption, enabling proactive resource allocation and performance optimization.

Future Test Procedures / Test Cases

Step 1 - Deploy the *Microservice Orchestrator*. Set up a Kubernetes cluster to function as the microservice orchestrator, responsible for automating deployment, scaling, and management of containerized microservices.

Step 2 - Deploy the *5G Core Network* (Free5GC). Implement the 5G core network, which provides a fully containerized and modular implementation of key 5G core functions such as AMF, SMF, and UPF for handling control and user plane operations.

Step 3 - Integrate the component/Microservice monitoring with the *Central SDN Controller* (Floodlight OpenFlow). Deploy and configure the controller to enable centralized network control, efficient traffic management, and optimized resource allocation across the 5G core components.

Step 4 - Set up the *Monitoring Engine* (Prometheus and Grafana) to continuously collect resource metrics from all deployed microservices. This includes CPU utilization, memory usage, disk read/write throughput, and other key performance indicators, providing real time data required for the Microservice Behavioral Analysis module.

Step 5 - Activate the *Microservice Behavioral Analysis Module:* use AI-driven anomaly detection to identify deviations from normal behavior or abnormal traffic patterns. Detected anomalies trigger automated actions through the orchestrator.

Step 6 - Perform controlled attack simulations on the deployed 5G microservice infrastructure to evaluate the responsiveness of detection and mitigation mechanisms. These scenarios will include different attack types e.g. DoS attempts, privilege escalation, and unauthorized access emulations.

Step 7 - Detect potential attacks through *Behavioral Anomaly Analysis*. Utilize the AI-driven anomaly detection framework to identify potential security threats. The two-stage CNN model analyzes real-time telemetry and resource consumption data to distinguish between normal and abnormal behavior, classifying anomalies.

Step 8 - Execute mitigation actions based on detected anomalies: When an anomaly is confirmed, initiate automated mitigation action through the orchestrator and SDN controller.

Results: Controlled attack simulations will confirm the system's resilience, with the two-stage CNN model accurately detecting and classifying threats such as DoS, privilege escalation, and unauthorized access. Confirmed anomalies were promptly mitigated through automated orchestration and SDN actions, isolating affected microservices, rerouting suspicious traffic, and maintaining overall service stability.

3.20. MTD Controller

The test procedures and expected results are recorded in the attached Excel sheet ZHAW-MTD-Controller.xlsx in Appendix A.18.

3.20.1. Test Procedures / Test Cases

The tests for the MTD Controller are organized around two functional goals: (1) verify live/stateful migration of CNFs without session loss, and (2) verify stateless migration (stop-and-recreate) for both VNFs and CNFs. The live-migration scenario (MTD Controller-TS.01 / TC.01) validates that the MTD Controller can coordinate with a container orchestrator (i.e., Kubernetes) and perform a transparent migration of a stateful CNF so that the service remains running and session state is preserved. The stateless scenarios (MTD Controller-TS.02 / TC.01 and TC.02) validate that the MTD Controller can coordinate either with an NFV MANO (for VNFs) or with the container orchestrator (for stateless CNFs) to stop execution on one node and instantiate an equivalent instance on another node, ensuring the function resumes operation on the destination node only.

Each test case requires the same basic cluster preconditions: the MTD Strategy Optimizer must be operational and able to decide migration actions, there must be at least two compute nodes available in the edge-to-cloud continuum. Tests may be exercised either proactively — by waiting for the optimizer's scheduled decision — or reactively by injecting a simulated cyberattack (examples used in the test cases are data-exfiltration and malware infection). These triggers validate both proactive and reactive MTD operations.

The test steps are intentionally simple and observable: initialize the MTD framework, allow the optimizer to decide (or trigger an attack to force a decision), then monitor the orchestration actions and the runtime status of the network function and the nodes involved. Acceptance

criteria and pass/fail conditions should be explicit and measurable. For both tests, pass criteria include: (a) orchestration commands observed in controller logs and orchestrator events, (b) VNF/CNF service running on the target cluster and not on the source cluster after cutover, and (c) continuity of ongoing sessions (no lost sessions or packet-loss spikes exceeding a pre-defined SLA).

3.21. MTD Strategy Optimizer

The test procedures and expected results are recorded in the attached Excel sheet ZHAW-MTD-Strategy-Optimizer.xlsx in Appendix A.19.

3.21.1. Test Procedures / Test Cases

The tests for the MTD Strategy Optimizer are organized around two complementary verification goals: (1) proactive decision-making, where the optimizer autonomously decides to relocate network functions based on monitoring and deep-RL trained optimal policies, and (2) reactive decision-making, where the optimizer responds immediately to detected security incidents. Proactive tests (MTD Strategy Optimizer-TS.01 / TC.01–TC.02) validate that the optimizer consumes monitoring telemetry, reasons about risks and resource consumption (via deep-RL continuous optimization), and issues migration actions for stateless VNFs and live/stateful CNFs. Reactive tests validate the optimizer's ability to quickly detect attack indicators reported by the monitoring tool and to recommend or trigger migrations as a containment/mitigation measure.

All test cases assume an integrated monitoring pipeline: e.g the OSM and Kubernetes orchestrators provide CNF/VNF life-cycle state information to the MTD Strategy Optimizer, while Montimage's MMT monitoring feeds real-time traffic telemetry. Tests are executed with a multi-cluster environment for dry-runs (in the ZHAW testbed), to validate them in an edge-to-cloud continuum scenario. Final evaluations are then done in a 5G network (following 6G Telco-Cloud setup and usage of CNFs and network slices).

Acceptance criteria are: (a) the optimizer emits a migration decision/plan within an allowed decision latency window, (b) the decision contains sufficient metadata (target CNF, MTD operation, and destination), (c) the MTD Controller correctly interprets the request given by the MTD strategy optimizer, (d) decisions that are in conflict with the network state or previously taken decisions are not enforced.

3.22. MTD Explainer

The test procedures and expected results are recorded in the attached Excel sheet ZHAW-MTDFed.xlsx in Appendix A.20.

3.22.1. Test Procedures / Test Cases

The MTD Explainer component is responsible for ensuring transparency and interpretability of the Moving Target Defense (MTD) system's automated decisions. This test scenario (MTD Explainer-TS.01 / TC.01) verifies that, whenever the MTD Strategy Optimizer decides to perform an action—either proactively or reactively—the Explainer can generate a clear, human-understandable rationale describing why that action was taken and how it enhances the system's security posture. The test assumes a fully functional MTD environment with the Strategy Optimizer active and deployed network function (either CNF or VNF) in the edge-to-cloud continuum. The Explainer should already be integrated with the MTD decision pipeline, capable of consuming decision metadata and contextual telemetry. Preconditions also require that monitoring data and the trained deep-RL model are available.

The **expected result** is that the MTD Explainer produces a human-interpretable explanation corresponding to the decision. This explanation should articulate *what* action was taken (e.g., a stateful CNF live migration versus a stateless VNF re-instantiation), *why* the action was necessary (e.g., response to detected attack, mitigation of aging-induced vulnerabilities, or proactive re-randomization of resources), and how it contributes to security improvement. The explanation is presented in natural language suitable for a system operator, auditor, or analyst, and it is evaluated against an expert-knowledge based analysis, checking that: 1. the generated explanation is coherent, accurate, and matches the actual decision taken; and 2. the explanation references the key decision drivers (e.g., detected threat, function age, or MTD Strategy Optimizer's confidence).

3.23. AI-driven security monitoring for anomaly detection and root cause analysis in IoT networks

To validate the effectiveness of the AI-driven anomaly detection and root cause analysis (RCA) framework, a set of test procedures and cases has been defined. These procedures aim to assess the system's performance across different operational and attack scenarios, ensuring compliance with the defined KPIs and overall objectives of NATWORK. Each test is carried out under controlled IoT/6G network conditions, where IoT devices, gateways, and monitoring probes are deployed, and traffic is generated either from real devices or simulated datasets. The following subsections describe the test scenarios in detail. The test procedures and expected results are recorded in the attached Excel sheet MONT-AI-AD-RCA.xlsx in Appendix A.21.

3.23.1. Test Procedures / Test Cases

- Baseline Performance Validation

The first test scenario establishes the baseline performance of the system under normal operating conditions. IoT devices are deployed to generate standard, benign traffic flows, while the monitoring probes (MMT) continuously analyze the traffic and the AI-based detection models (MAIP) process the data. The objective is to validate that no anomalies are falsely reported, while key performance indicators such as latency, throughput, and CPU utilization are collected. This baseline serves as a reference for subsequent tests, allowing us to distinguish between normal variations in traffic and actual anomalies.

- DDoS Attack Detection

The second scenario evaluates the system's ability to detect large-scale DDoS attacks. Using traffic generation tools, SYN flood and UDP flood attacks are launched against IoT gateways and edge nodes. Both ML-based detection rules and traditional non-ML rules are tested. The aim is to measure the mean time to detect (MTTD), with a target of under 5 minutes for ML-based approaches and under 10 milliseconds for MMT's non-ML rule-based detection. This scenario verifies the responsiveness of the anomaly detection system and its capacity to trigger timely alerts under high-volume attack conditions.

- Detection Accuracy: False Positives and Negatives

The third test scenario focuses on the accuracy of anomaly detection, particularly with respect to false positives (FP) and false negatives (FN). Mixed datasets containing both benign traffic and malicious traffic (covering various attack types) are replayed. The goal is to ensure that the detection system raises alerts only for genuine threats while ignoring harmless anomalies. The KPI targets for this scenario are set at less than 1% for both FP and FN rates. The results of this test provide a quantitative measure of the reliability of the AI models and their suitability for large-scale IoT deployments.

- Packet Loss and Performance Impact

The fourth scenario assesses the impact of monitoring and detection on overall network performance. Probes are deployed in environments with constrained bandwidth as well as under high-load conditions to evaluate the Packet Loss Ratio (PLR). The system is expected to maintain a PLR below 0.001%, ensuring that IoT communication remains reliable while security monitoring is active. This scenario is critical for validating that the anomaly detection and RCA framework does not degrade the quality of service or compromise the efficiency of IoT operations, even in resource-limited environments.

- Incident Resolution and Mitigation Time

The fifth scenario evaluates the system's ability to resolve incidents rapidly after detection. Once an attack (e.g., a flooding attack) is launched and identified by the anomaly detection system, the

mitigation mechanisms are activated. These include traffic rerouting, access control enforcement, or container migration strategies, depending on the context. The test measures the mean time to resolve (MTTR), with the objective of reducing this to under 10 minutes. This ensures that service continuity is preserved and that disruptions to IoT applications are minimized during ongoing attacks.

- Root Cause Analysis (RCA) Validation

The sixth and final scenario focuses on validating the RCA module, which is central to UC#3.1's innovation. Several subcases are explored to ensure robustness: (i) in cases of benign misconfigurations (e.g., faulty routing rules), the RCA module must correctly distinguish these from malicious events; (ii) during malicious attacks, such as SYN floods, the RCA module must pinpoint the source, type, and scope of the anomaly; (iii) in scenarios involving compromised IoT devices, the RCA system must attribute suspicious behavior to the responsible node; and (iv) explainable AI (XAI) techniques, combined with LLM-based reporting, must generate human-readable explanations that operators can trust and act upon. This scenario ensures not only technical detection but also usability and transparency for human operators, closing the loop between detection, understanding, and action.

3.24. Security-performance balancer

3.24.1. Test Procedures / Test Cases

The test procedure for the Security Performance Balancer should involve generating user traffic with different security algorithm configurations—such as Snow, AES, and ZUC—while monitoring how the system distributes users across servers based on their ciphering, integrity, and replay protection settings. The test should verify that users employing the same algorithms are grouped onto the same servers, reducing CPU load and maximizing the use of cryptographic accelerators.

3.25. DFE Telemetry

The test procedures and results are recorded in the attached Excel sheet CNIT-DFE-Telemetry.xlsx in Appendix A.22.

3.25.1. Test Procedures / Test Cases

Extensive test sessions have been conducted for the DFE-Telemetry component to verify its correct functionality and performance under different operating conditions. The component has been tested in the P4 BMv2 backend. Further test sessions will be executed when the version for the Tofino TNA backend will be available.

The first scenario (i.e., DFE-Telemetry-TS01), addressed the compilation and deployment of the P4-DFE Telemetry program on the BMv2 software switch. The source code was compiled using the P4C compiler and deployed successfully, with the binary loading correctly in the target environment. The outcome confirmed the readiness of the software implementation and the correctness of the build process.

The second scenario (i.e., DFE-Telemetry-TS02), focused on functional validation within a Mininet environment. A topology consisting of three hosts and three concurrent flows (two UDP and one TCP) was instantiated, and the telemetry program was deployed on the switch. Reports were correctly generated for all flows, confirming the capability of the component to monitor heterogeneous traffic patterns and produce telemetry outputs as expected.

The third scenario (i.e., DFE-Telemetry-TS03), investigated the performance impact in terms of latency and CPU overhead. Comparative experiments were conducted between simple forwarding and telemetry-enabled forwarding using external Spirent traffic generators. Results demonstrated that the latency overhead introduced by the telemetry component remained within the expected range and did not compromise normal forwarding. Similarly, the CPU load remained within acceptable bounds, with differences observed between simple and telemetry-enabled forwarding confirming the correct operation of telemetry features without excessive resource usage.

The fourth scenario (i.e., DFE-Telemetry-TS04), extended the performance evaluation to scalability conditions. By progressively increasing the number of flows from 1 to 10, 100, and 1000, the component was evaluated for both latency and CPU overhead under high-load conditions. The observed increases were moderate and consistent with expectations, demonstrating that the DFE-Telemetry component is capable of scaling effectively with the traffic load while maintaining operational stability.

Across all executed scenarios, the expected and actual results coincided, and all test cases were marked with a “Pass” status. No anomalies or deviations were detected. The results confirm that the DFE-Telemetry component meets its design objectives, both in terms of functional correctness and performance. The full set of detailed test descriptions and measurements is provided in the Annex, where the corresponding Excel test report is included.

3.26. Secure Data Aggregation

The full set of test procedures and results is recorded in the attached Excel sheet ELTE-Data-Aggregation.xlsx in Appendix A.23. All test cases completed successfully.

3.26.1. Test Procedures / Test Cases

The **Basic Flwr Server Startup** tests (Secure_DA-TS.01) confirmed that the Flower (Flwr) environment initializes correctly. The server started successfully, clients connected, and federated training rounds executed without errors. Training completed across multiple rounds, with aggregated global model accuracy improving steadily as expected.

The **SecAgg+ for Secure Aggregation** tests (Secure_DA-TS.02) validated the integration of the SecAgg+ module for privacy-preserving training. Secure key exchange, encryption, and masking were verified in logs. Aggregation succeeded using masked client updates, preserving data privacy while maintaining comparable convergence to the non-secure baseline.

The **MPC-Based Secure Aggregation** tests (Secure_DA-TS.03) demonstrated that secure multi-party computation (MPC) was correctly initialized using the MP-SPDZ library. End-to-end MPC aggregation completed successfully, ensuring that no single party accessed individual client data. Performance overhead remained within acceptable limits compared to SecAgg+.

All procedures under Secure_DA-TS.01–TS.03 executed successfully. The secure aggregation mechanisms (SecAgg+ and MPC) functioned as intended, ensuring privacy-preserving federated learning with stable training and reliable performance. The component is verified to be functional, secure, and ready for integration into the larger system.

3.27. Federated Learning for edge-to-cloud

The test procedures and results are recorded in the attached Excel sheet UEssex – Federated Learning edge-cloud.xlsx in Appendix A.24.

3.27.1. Test Procedures / Test Cases

TS01 (Centralized ML benchmarking): Historical Google cluster traces were pre-processed with feature engineering techniques (e.g., lag features, rolling statistics) to extract CPU and memory patterns. We prepared datasets for model training through two distinct methods: fine-level granularity for detailed patterns, and orchestration-focused aggregation for peak demand planning. Models including ARIMA, LSTM, and XGBoost were trained with hyperparameter tuning and evaluated against real workload traces. This benchmark validated the predictive framework and provided a baseline for orchestration use cases.

TS02 (Baseline federated learning framework): Google workload traces were partitioned across multiple nodes to emulate distributed edge–cloud training. Local XGBoost models were trained independently on each node, with predictions aggregated using a bagging-based FL approach. Results confirmed the feasibility of decentralized learning, showing comparable performance to centralized ML, while demonstrating scalability for edge-to-cloud scenarios.

These tests collectively establish a validated baseline for predictive workload modeling, serving as the foundation for future FL extensions with custom DoST datasets and MinIO-based persistent data storage.

3.28. MTDFed

The test procedures and expected results are recorded in the attached Excel sheet ZHAW-MTDFed.xlsx in Appendix A.25.

3.28.1. Test Procedures / Test Cases

The MTDFed component enables collaborative training of the MTD Strategy Optimizer across multiple VNOs through an FL approach. The goal of these tests is to verify that distributed optimizers deployed across different networks can jointly train a more accurate and resilient global model without sharing network monitoring data. Three test cases are defined to evaluate basic functionality and the integration of privacy-preserving mechanisms such as Multi-Party Computation (MPC) and Differential Privacy (DP).

The first test case (MTDFed-TS.01-TC.01) validates the baseline functionality of MTDFed without privacy mechanisms. With at least three VNOs deployed across edge nodes and an active aggregator in the core network, the test triggers federated learning across several rounds and observes the aggregation and convergence process. The expected outcome is that the global MTD Strategy Optimizer model progressively improves its performance compared to the local models, demonstrating successful synchronization and aggregation across VNOs. The second test case (MTDFed-TS.01-TC.02) extends this setup by enabling secure aggregation through MPC. This test ensures that individual model updates from VNOs remain confidential during the learning process. While the procedure mirrors the baseline test, the key validation point is that the aggregator correctly aggregates encrypted updates without accessing any private model information. The third test case (MTDFed-TS.01-TC.03) assesses the use of Differential Privacy within the MTDFed framework. Here, each VNO introduces noise to local updates before sending them to the aggregator, ensuring privacy protection even against potential reconstruction attacks. The test verifies that the system converges successfully and that the global model remains functional, measuring the possible reduced accuracy due to the privacy noise.

3.29. CIA-hardening of x86 payloads Component

The test scenarios and test cases for this component can be found in the attached Excel sheet TSS-CIA hardening x86 payloads.xlsx in Appendix A.26.

3.29.1. Test Procedures / Test Cases

Testing the CIA-hardening of x86 payloads techniques will go through several independent test procedures and test cases, each reflecting a specific hardening technique related to the elevation of payload confidentiality, integrity, and availability. The associated tests will be worked out using MMT probe, a security-related payload, producing continuous anomaly detection. Several testbeds can be considered as this payload will be installed at PNET, CNIT or MONTIMAGE. The testbed will be selected to place the payload in working conditions as an important KPI is the (low) performance penalty induced by the hardening, which can be precisely measured in real working conditions.

- Confidentiality preservation

MMT confidentiality preservation will be elevated by a SECaaS automatic operation which encrypts the text section of its ELF-formatted .exe or .so file. This operation protects MMT against static analysis, discovery of potential vulnerabilities, or intellectual property violations. The test procedure first checks the effectiveness of the security promise, then looks at the latency caused by the decryption (below 3 sec) realized before code execution and finally validates the negligible performance impact during execution. A baseline will be collected on an unprotected MMT. Timestamps will be implemented and used for the timing measurement. The associated tests are SECaaS-Conf-x86-TS.01/02/03.

- Integrity preservation

MMT integrity preservation will be evaluated when the code is on-boarded (i.e., remote attestation) and during its execution (i.e., runtime integrity verification). To get these security attributes, a SECaaS operation will be carried out before deployment to (i) inject Prove, Verify and DLT communication primitives into the code, (ii) build a reference measurement of the augmented payload (i.e., hash of the memory footprint, used as a reference for future integrity checks). .

D-MUTRA blockchain based mutual remote attestation framework will be leveraged as it is dependency-free, enabling MMT code deployment anywhere. This setup workflow will be modified if MMT is deployed as a container, obviating the SECaaS operation. Leveraging *Docker compose* deployment utility, a sidecar will be appended on MMT namespace, hence getting a visibility on its memory footprint. The test procedure will first check the effectiveness of the security promise, trigger tampering, and check its detection. Then, the remote attestation cycle timing (up to the creation of a block) will be measured. Then the procedure will check the performance impact of periodic or event-based (i.e., on-demand) MMT integrity verification during its execution. A baseline will be collected on an unprotected MMT. Timestamps will be

implemented and used for the timing measurement. The associated tests are SECaaS-Int-x86-TS.01/02/03.

- Availability preservation

MMT availability preservation will be evaluated when the code is executing, checking the relevance of the monitoring (i.e., providing an accurate measure of MMT performance in terms of processed packets) as well as the performance penalty caused by the monitoring. A baseline will be collected on an unprotected MMT. Several types of measurement will be worked out, based on the measurement of the call frequency of MMT's packet processing routine or based on the time to execute a reference code block, structurally defined to be independent from historical past processing. A baseline will be collected on an unprotected MMT. Timestamps will be implemented and used for the timing measurement. For simplicity in finding the correct locations for these timestamps' insertion, we will consider LLVM compilation framework which delivers the function names (ie, symbol) removing the opacity of assembly code. The associated tests are SECaaS-Avail-x86-TS.01/02/03

3.30. CIA-hardening of containerized payloads

The test procedures and expected results are recorded in the attached Excel sheet TSS-CIA hardening Containers payloads.xlsx in Appendix A.27.

3.30.1. Test Procedures / Test Cases

The related test scenarios and test cases for this component have been recorded and displayed in the appendix section of this document. The evaluation of this component is going to be performed in the PR2 between months M28 to M31. Two payload alternatives will be considered (i.e., MONT's MMT or ISRD's Liquid xApp). For simplicity, if practicable, we will use the same workload as stated for x86 workload above, notably the modified version with inserted timestamps.

- Confidentiality preservation

No specific test will be worked out as the state of the art fulfilling this need is mature in this respect (i.e., encryption of container OCI image OCI v1 spec). To advance the state of the art, it is required to operate at the executable level (i.e., x86 executable) inside the container and as offered for x86 payloads above.

- Integrity preservation

The implementation consists of setting up a sidecar which operates aside the container, sharing the same namespace PID and additionally allocated with CAP_SYS_PTRACE Linux capability. With

these two conditions, the sidecar will access and measure a defined x86 executable memory footprint resident in the container. By means of Linux *cgroups* resource management, we will be able to restrict the resource consumed by the sidecar, hence the impact on the container performance. The associated tests are SECaaS-Int-Cont-TS.01/02.

- Availability preservation

With the same memory access conditions as defined above, the sidecar will be able to collect sampled runtime information (i.e., instruction pointer states, stack trace) as well as executing an specific performance reference code to assess if the code is current executing (or idle), if the platform resource is under stress and tentatively assess the performance ratio of the code versus a reference performance. he associated tests are SECaaS-Avail-Monit-TS.01/02.

3.31. CIA-hardening of WASM payloads Component

The test scenarios and test cases for this component can be found in the attached Excel sheet TSS-CIA hardening WASM payloads.xlsx in Appendix A.28.

3.31.1. Test Procedures / Test Cases

Based on the feasibility study by TSS on WASM hardening, the CIA hardening as defined for x86 above will be defined and processed. The tests will be worked out with representative WASM workloads. If applicable and agreed, one of them could be MONT's MMT (compiled in Web Assembly bytecode). Some of the associated tests will directly depend on a feasibility study stating the possibility to harden a WASM module against CIA attacks. At the current stage, we have demonstrated that runtime WASM module integrity can be verified.

- Confidentiality preservation

Comparable techniques to those used for x86 are applied to WASM workloads, aiming at encryption of sensitive sections and protection against static reverse engineering. The associated tests are SECaaS-Conf-WASM-TS.01/02/03.

- Integrity preservation

The modified WASMTIME runtime computes a runtime signature of the WASM bytecode and compares it with the pre-deployment reference signature of the same module. Any mismatch indicates tampering of the WASM payload. This process is integrated with the D-MUTRA blockchain to enable decentralized validation and secure record keeping. The associated tests are SECaaS-Int-WASM-TS.01/02/03.

- Availability preservation

WASM runtime performance monitoring will be worked out by instrumenting the WASM runtime. For simplicity, we will instrument the x86 runtime with timestamps inserted before compilation). Additionally, we will explore, for containerized runtime implementations, if and how the sidecar monitoring layout as stated above can be considered. The associated tests are SECaaS-Avail-WASM-TS.01/02/03.

3.32. JDM-xApp

3.32.1. Test Procedures / Test Cases

The JDM xApp performs the choice between the basic AMC algorithm and AMC-based Jamming Detection and Mitigation algorithm. The test procedure shall verify the correctness of choice of the algorithm, i.e. in the jamming scenario the chosen algorithm should be AMC-based Jamming Detection and Mitigation and in the no-jamming scenario (normal operation) the chosen algorithm should be basic AMC.

In the first release, the two algorithms are implemented and tested independently in Liquid RAN. In the next release, their choice will be controlled by the JDM-xApp.

3.33. Liquid RAN

3.33.1. Test Procedures / Test Cases

In the first release, the AMC-based Jamming Detection and Mitigation algorithm implemented in the scheduler is tested. The test procedures shall verify system robustness under jamming on different channels. In the jamming on PRACH test case, the test verifies that UE is able to attach. In the jamming on control and shared channels (UL and DL) It verifies if RRC CONNECTED is maintained. The detailed test descriptions are reported in ISRD-Anti-jamming.xls in Appendix A.29.

3.34. Liquid Near-RT RIC

3.34.1. Test Procedures / Test Cases

The near-RT RIC (Near-Real-Time RAN Intelligent Controller) is a component of the O-RAN architecture that enables intelligent control and optimization of the RAN within a timescale of 10ms to 1s. It hosts applications (xApps) that use near real-time data to manage RAN functions. The RIC testing verifies correct mounting of xApp to the RIC, that RIC has connectivity with RAN over E2 interface and whether RIC can correctly send and receive messages to and from RAN. Specifically, The test procedures should verify whether RIC correctly passes the KPMs to the JDM –xApp and correctly passes the algorithm choice message from the JDM-xApp to the Liquid RAN.

3.35. KPM xApp

3.35.1. Test Procedures / Test Cases

The built-in Liquid Near-RT RIC KPM xApp subscribes to KPMs for all cells and stores them in the Valkey database. The test procedure should verify that the KPM xApp correctly subscribes to the KPMs (e.g.: CSI, Reference Signal Received Power (RSRP), CQI, HARQ feedback, ACK/NACK patterns and BLER), correctly stores KPMs in the database every 1-second.

3.36. Characteristics Extractor

3.36.1. Test Procedures / Test Cases

The full details of the test procedures and cases for this component are provided in the attached file **GRAD-Characteristics_Extract.xlsx** in Appendix A.30, which includes parameters, steps, results and dates. All dry-run tests on the component have been successfully completed.

This component is responsible for correctly extracting channel measurements for key generation. It covers the proper generation, transmission, and reception of OFDM samples, as well as the correct extraction of I/Q samples. In this setup, signal acquisition is completed without issues.

It also builds the input dataset for the channel-prediction neural network: a portion of the extracted data is set aside to train the model and to quantify channel variations accordingly. Two test scenarios were conducted. one with two nodes, the main Alice–Bob link, and another with three nodes, Alice–Bob plus an eavesdropper (Eve), with measurements taken from Eve's position.

3.37. Key Generator

3.37.1. Test Procedures / Test Cases

The full details of the test procedures and cases for this component are provided in the attached file **GRAD-KeyGen.xlsx** in Appendix A.31, which includes parameters, results steps, and dates. All dry-run tests on the component have been successfully completed.

The key-generation pipeline comprises the following steps: acquire raw I/Q samples; run them through the AI DL-UL prediction model; quantize I/Q samples into bits; perform information reconciliation; and hash the reconciled bits to derive the final key. Disagreements between the two ends are evaluated with the corresponding metrics.

On the attached file two test blocks were executed, OFDM-TDD and OFDM-FDD, for both, we collected the primary metrics: KDR (Key Disagreement Ratio) and the ML model MAE (Mean Absolute Error), validating the correct component functionality and performance.

3.38. Security Evaluator

3.38.1. Test Procedures / Test Cases

The full details of the test procedures and cases for this component are provided in the attached file **GRAD-SecurityVal.xlsx** in Appendix A.32, which includes parameters, steps, results and dates. All dry-run tests on the component have been successfully completed.

The Security Validator evaluates the security of the PKG pipeline. First, it takes the generated keys and subjects them to the NIST statistical test suite. Second, it validates security against an eavesdropper scenario (Eve), assessing leakage and advantage under the chosen threat model.

3.39. AI -Based Anomaly Detection Explainer

3.39.1. Test Procedures / Test Cases

The full set of test procedures and results is recorded in the attached Excel sheet UZH-Anomaly Detection Explainer.xlsx in Appendix 33. All tests were executed in August 2025, and every executed case passed successfully. The explainer service was built and deployed without errors. Health checks and REST endpoints responded correctly in the target runtime, confirming a clean rollout.

Alert ingestion & schema validation: Valid IDS alert payloads were accepted and processed, while malformed or schema-incompatible payloads were rejected with informative errors, as expected.

Explanation generation: For malicious traffic samples, the component produced consistent, human-readable explanations capturing salient features/evidence. For benign flows, no explanations were emitted, matching the design intent.

Control-plane integration: Control-plane updates (e.g., playbook selection and policy signaling) were ingested correctly. Actions were logged and traceable end-to-end, with no unintended configuration changes.

Robustness & fallback: Under malformed inputs, missing model artifacts, or dependency timeouts, the system failed gracefully returning safe defaults, preserving service availability, and avoiding crashes or stalls.

No anomalies or deviations were observed during testing. The results confirm that the Anomaly Detection Explainer Component is stable, functional, and ready for integration.

3.40. Wirespeed traffic analysis in the 5G transport network

The full set of test procedures and results is recorded in the attached Excel sheet CERTH-Wirespeed-traffic-analysis.xlsx in Appendix A.34. All tests that have been executed so far are completed successfully.

3.40.1. Test Procedures / Test Cases

A set of test scenarios was carried out to evaluate the functionality, classification accuracy, and control-plane integration of the **Wirespeed Traffic Analysis system** based on P4-programmable SmartNICs and the CERTH Intrusion Detection System (IDS). Under **Test Scenario TS01**, the goal was to validate the compilation and deployment pipeline. In **TC01**, the P4 program was successfully compiled and deployed on an Agilio SmartNIC, confirming correct loading of the binary. In **TC02**, the integration between the P4 data plane and the CERTH IDS was tested, verifying that the IDS could correctly parse and process ingress traffic forwarded by the Agilio SmartNIC in a live 5G network environment.

Test Scenario TS02 focused on packet classification capabilities. In **TC01**, the system correctly identified and tagged all benign traffic, while **TC02** validated its performance on malicious inputs using the CICIDS2017 dataset, with accurate detection of threat flows.

Finally, **Test Scenario TS03** evaluated control-plane integration. In **TC01**, the CERTH IDS produced inferences based on traffic characteristics, which were then translated into runtime rules (e.g., drop malicious, forward benign) and successfully applied by the P4 controller. The verification confirmed that benign traffic was forwarded, and malicious traffic was dropped as expected.

All test cases that were executed between **September 2024 and September 2025** resulted in a **Pass** status, demonstrating that the system operates reliably across the entire pipeline—from compilation and deployment to real-time detection and mitigation.

3.41. Detection and mitigation against jamming attacks (HES-SO)

3.41.1. Test Procedures / Test Cases

In this service-component HES-SO has made significant progress. The reader can check the different tests performed in the attached file HES-SO_Jamming.xlsx in Appendix A.35, all of them had passed correctly. At the current state the testbed is in place, gNodeB and 5G CN are running in containers, jammer is also functional running itself in a container as well and UE (+ SIM card) are also well configured. The testbed is also running the near RT RIC (Real Time RAN Interface Controller) but not yet tested. The metrics collected on the gNodeB provided by the UE have been analyzed only offline using the logs provided by gNodeB (srsRAN) but not by an xApp which

has not been developed yet. There is still an open discussion regarding how to report to the scheduler to provide mitigation against jamming.

3.42. Setting up of a Mirai botnet.

As part of the service component, the HES-SO team successfully implemented a complete and reproducible Mirai environment. The deployment is based on physical machines hosting isolated virtual machines running the Mirai binary and its command infrastructure (CNC). Secure orchestration of agents is ensured by a host-guest communication channel based on QEMU/KVM (AF_UNIX socket on the host side — virtual serial device on the guest side), controlled by dedicated daemons (mmc-host-daemon, mmc-vm-daemon) and an administration client (mmc-cli) discovered via IPv6 link-local multicast. This device makes it possible to generate varied and controlled attack scenarios while ensuring the strict isolation of malicious traffic from the institutional network. The integration of this experimental bench with the pre-processing pipeline (CICFlowMeterV4 → CSV/Parquet) facilitates the production of traces that can be used for model training and evaluation.

3.42.1. Test Procedures / Test Cases

The XGBoost model trained on the CIC-DDoS2019 dataset achieves an overall classification accuracy of 94.2%. Frequent attacks and benign traffic, such as NTP, SYN, and TFTP, are detected with F1 scores close to 0.99–1.00, demonstrating the model's excellent performance on majority classes.

However, rare classes, such as DrDoS_MSSQL (F1 = 0.18) and DrDoS_LDAP (F1 = 0.31), as well as some WebDDoS attacks, show much lower performance. This highlights the model's sensitivity to class imbalances and the importance of balanced sampling or resampling methods to improve generalization.

Inference on Mirai's local traces reveals that TCP SYN attacks are detected but often mislabeled, and that benign traffic, particularly RTSP streams, can be misclassified as attacks. This observation shows a mismatch between the distribution of training data and operational traces, limiting the model's ability to generalize to Mirai variants.

These results highlight two key points: the need to build representative Mirai datasets and to explore more robust modelling approaches, including temporal or hybrid features. The testbed developed is thus useful for generating realistic Mirai scenarios, enabling the improvement of detection models and the evaluation of cross-domain generalization.

3.43. FPGA-based hardware detection of DDoS attacks

At present, progress on this component of the service remains limited. The Smart-NIC is connected to a PC via the PCIe interface. Although work has been carried out on this interface, no relevant DDoS detection model has yet been implemented.

Development is currently underway, with the first step focused on processing network packets using the P4 architecture. This fundamental step is essential to ensure that the Smart-NIC can efficiently handle network traffic and provide the hooks necessary for future functionality.

Once the packet processing framework is stable and fully functional, the next phase will be to integrate AI-based models for DDoS attack detection. This will enable real-time identification and mitigation of attacks directly on the Smart-NIC, leveraging the hardware acceleration capabilities provided by the P4 pipeline.

In summary, the project is progressing step by step: first, a robust P4-based packet processing infrastructure is being set up, and then AI is gradually being integrated.

4. Attacks

Modern mobile and cloud-native infrastructures — particularly those that underpin 5G, B5G and emerging 6G systems — support an ever-expanding set of services and rely on a complex stack of protocols, virtualized network functions, and orchestration mechanisms. This complexity increases both the attack surface and the difficulty of robustly evaluating defensive measures. The datasets presented in this chapter are intentionally diverse and realistic: they capture low-level network traces (pcap), per-flow and short-window feature summaries (CSV / Parquet), time-domain signal samples for wireless jamming research, and labelled scenarios that span classical volumetric DoS and port-scanning, protocol-specific exploits, automated brute-force campaigns, and advanced AI-assisted attack strategies. Together, they provide a comprehensive resource for the development, testing and benchmarking of detection, mitigation, and resilience techniques across networking, wireless signal processing, and cloud/OT (Operational Technology) domains.

There are three motivating goals behind these datasets. First, to provide realistic, labeled datasets that reflect attacks actually observed or plausibly executed against modern mobile and cloud-native infrastructures (e.g., attacks mirrored at N3/N6 in a 5G testbed or protocol abuses in HTTP/2/SCTP). Second, to support multi-layer research: from packet/header-level anomaly detection and per-flow ML classifiers to radio-domain jamming detection and CNF-level (cloud-native function) resilience strategies. Third, to enable reproducible experimentation for both classical ML workflows (training/validation/test splits over CSV/Parquet feature sets) and signal-processing research (IQ sample datasets for JASMIN training). Each dataset is packaged to facilitate immediate use: raw captures for protocol analysts and forensic researchers, and ML-ready, per-flow and 1-second window aggregates for model builders.

4.1. DoS attacks and port scans

Captured on the UZH mini-5G testbed with mirroring at N3 (GTP-U) and N6 (IP). Includes benign baseline plus labeled attacks: ICMP/UDP/SYN/HTTP floods, slow-rate DoS; SYN scan, TCP-connect scan, UDP scan. Delivered as raw pcapng and ML-ready CSV/Parquet (per-flow & 1-s window features: rate, burstiness, inter-arrival stats, TCP flags, unique ports, entropy; TEID/inner proto for N3). Labels via run_id/time-window; privacy: payload stripped, IPs anonymized, TEIDs remapped. Intended for UC3.1 ML/XAI training/validation.

4.1.1. Testbed & Service Mapping

The UZH testbed provides a direct link between simulated network attacks and the service components designed to analyze them. Attack datasets, such as the one containing DoS attacks and port scans, are generated within controlled testbed scenarios. This is achieved by running

malicious traffic from the emulated UE, which then traverses the software-based gNodeB and the free5GC core.

The traffic is captured at key interfaces, primarily at the N3 interface between the gNodeB and the UPF, using a port mirror. These captured packets are then mapped directly to the IDS/XAI backend service. This service ingests the data, performs feature extraction, and feeds it into machine learning models for anomaly detection. If an anomaly is detected, the inference results are passed to the Anomaly Detection Explainer component, which generates a human-readable explanation for the alert. This creates a clear, end-to-end mapping from a specific attack scenario on the testbed to the corresponding detection and explanation services.

4.1.2. Dataset preparation.

First, a selection of both benign baseline traffic (e.g., ping, HTTP, iperf) and specific, labeled attack traffic (e.g., ICMP/UDP/SYN floods, port scans) is generated within the testbed. This raw traffic is captured as `pcapng` files.

Next, the data undergoes preprocessing. To ensure privacy and prevent data leakage, payloads are stripped, IP addresses are anonymized, and Tunnel Endpoint Identifiers (TEIDs) are remapped. The core of the preparation involves feature extraction, where the raw packets are transformed into structured formats like CSV or Parquet. This process constructs network flows and calculates features over 1-second windows. Key extracted features include:

- Packet and byte rates
- Burstiness and inter-arrival statistics
- TCP flags and unique port counts

Finally, the data is structured and labeled using `run_id` and time windows, making it ready for model training and validation.

4.1.3. Training and Validation

The prepared datasets are central to the training and validation of the entire anomaly detection service. The ML-ready CSV/Parquet files, containing detailed features and corresponding labels, are used to train various detection models, including Random Forest, XGBoost, and neural networks (CNN/DNN).

The validation process is twofold. First, the IDS model's performance is evaluated using standard metrics, with results like the confusion matrix visualized on the operator's dashboard. This confirms the model's accuracy in distinguishing between benign and malicious traffic from the dataset. Second, the Anomaly Detection Explainer component undergoes rigorous assessment.

For each correctly identified attack, the quality of the generated explanation is measured against three KPIs: faithfulness, robustness, and complexity.

4.2. AI-DoS attack

This Use Case aims to formulate a variety of AI-based Attacks dedicated to B5G/6G infrastructure with enhanced capabilities. Indicatively, an AI-powered DoS attack will use Reinforcement Learning algorithms to identify and exploit weaknesses in the network that can be used to flood it with traffic or cause it to crash. Also, a protocol fuzzer will be utilized to identify vulnerabilities in the network protocols and test them for potential exploitation. Finally, it will have the ability to apply techniques that will bypass IDS detection making it more effective in attacks.

4.2.1. Testbed & Service Mapping

AI-DoS attack tool applies DoS attacks against different components of CERTH'S 5G testbed. During the tool's dry run, AMF and SMF components were targeted, however the tool is able to target any system utilizing TCP, UDP and SCTP protocols.

4.2.2. Dataset preparation.

Since AI-DoS attack is based on Reinforcement Learning, it does not need training and validation data. Instead, the training process takes place through the interaction with the environment in which the attacks are applied. This means that the attack strategy is adapted according to the feedback received from the target. However, the artifacts of AI-DoS attack can be used to formulate a dataset to train the 5G IDS tools.

4.2.3. Training and Validation

The AI-DoS attack model training is based on rewards received from taking certain actions and evaluating the impact of those actions on the target. A pre-defined number of episodes determines the training duration. Each one of the episodes, concludes after 10 rounds, or after successful DoS attack. Throughout this process, AI-DoS attack model is learning how to effectively deteriorate the QoS of the target, by measuring several factors such as latency, throughput and packet loss. The validation involves applying the AI-DoS attack on a target using the pre-trained wights acquired from the training process.

4.3. DoS attacks and Brute Force attacks

The following section presents a number of DoS attacks that are available in CERTHs' SDN testbed and will be utilized for UC4.4.

4.3.1. Testbed & Service Mapping

A DoS attack targeting the Stream Control Transmission Protocol (SCTP) is a cyberattack strategy focused on disrupting an existing SCTP session or connection, thereby hindering communication between two services. There are several techniques attackers might employ to achieve this disruption. In this use case, the attack targets the SCTP protocol by attempting to set up numerous SCTP sessions.

The HTTP/2 Slow Get Flooding attack is a variant of a DoS attack specifically tailored to exploit the HTTP/2 protocol. In this attack, the malicious user initiates numerous connections to the target server and deliberately keeps these connections active for an extended period by gradually sending incomplete requests. The strategy behind this slow-paced submission of requests is to occupy server resources indefinitely, preventing the server from closing the connection due to inactivity.

The HTTP/2 Ping Flooding attack exploits the "PING" frame feature of the HTTP/2 protocol, which is designed to measure the minimum round-trip time between the client and the server. Attackers execute this type of attack by dispatching an excessive number of PING frames to the target server in quick succession. The primary goal of this attack is to deplete the resources of the server. By inundating the server with these frames, the attacker forces it to allocate a significant amount of its processing capacity and bandwidth to handle and respond to each PING request. This excessive demand on the server's resources can lead to a slowdown or even a complete halt in its ability to serve legitimate requests, effectively disrupting the service for genuine users.

The HTTP/2 Slow Get Flooding attack is a variant of a DoS attack specifically tailored to exploit the HTTP/2 protocol. In this attack, the malicious user initiates numerous connections to the target server and deliberately keeps these connections active for an extended period by gradually sending incomplete requests. The strategy behind this slow-paced submission of requests is to occupy server resources indefinitely, preventing the server from closing the connection due to inactivity.

A Brute Force SSH is an adversarial access technique that aims to overwhelm or compromise an SSH service by repeatedly attempting to authenticate with many different username/password or key combinations until a valid credential is found or the server becomes unable to respond to legitimate clients. Attackers typically open numerous SSH connections or rapidly iterate through credential lists—either from a single source or distributed across many hosts—to exhaust the target's authentication subsystem, CPU and connection-tracking resources, and logging/storage capacity. The consequence can be degraded service or complete denial of remote administrative access, increased load on intrusion-detection/logging systems, and the potential for unauthorized access if weak credentials exist. In the testbed use case, this attack is modeled by

launching a high rate of authentication attempts against the SSH daemon to observe how the SDN environment detects, isolates, and mitigates excessive connection attempts and credential-guessing behavior.

A container offers these attacks by wrapping multiple scripts written in python. The SSH brute force attack is based on the functionalities offered by an open-source tool called Hydra.

4.3.2. Dataset preparation.

Currently separate attacks and benign traffic were recorded separately in pcap files and then converted to flows to be saved as csv to be utilized by the various components developed for the project.

For the next period, this process will be unified to create a more complete dataset. Normal traffic with multiple UEs will be emulated for 24 hours. For each attack, at least one hour of malicious traffic will be collected. Finally, data will be collected during simultaneous execution of the various attacks. The attacks will be carried out against 5G Core VNF (AMF, UPF) and other dockerized services e.g. an ngix server, and microservices e.g. a python based microservice offering access to an LLM app via a REST API.

4.3.3. Training and Validation

The datasets produced by these attacks will be utilized for training algorithms utilized by the following components:

- Lightweight SDN-based AI-enabled Intrusion Detection System for cloud-based services presented in section 3.14
- Multimodal Fusion Approach for Intrusion Detection System for DoS attacks presented in 3.13
- Multiagent AI based cybersecurity support system presented in section 3.16.

4.4. OT/ICS attacks

In this section, we present an adversary intrusion chain against an OT/ICS testbed composed of a vulnerable web application, Apache Tomcat management interface, OpenPLC controllers, and a SCADA front end (ScadaBR). The scenario leverages two representative vulnerabilities: CVE-2021-44228 (Log4Shell) to obtain remote code execution via crafted log payloads, and CVE-2009-3548 (default/weak Tomcat credentials and exposed manager application) enabling WAR upload and remote deployment. Combined, these weaknesses enable attackers to deliver payloads to engineering hosts, run loaders, propagate laterally, and interact with PLCs/HMIs — enabling both data exfiltration and process manipulation. This section describes how the testbed maps to attack

datasets, how datasets are prepared, and how they are used for training and validating detection / attribution models and assessing service-component impact.

4.4.1. Testbed & Service Mapping

The testbed is organized into distinct network segments with clearly defined roles. The perimeter or DMZ hosts an externally reachable web application and an Apache Tomcat instance that exposes the manager webapp and user-deployable WAR endpoints; these services represent the publicly accessible attack surface and are the intended targets for Log4Shell injections and WAR upload abuse. The engineering/IT segment contains one or more engineering workstations running management and development tools; these hosts collect logs and can be targeted by phishing and remote code execution, becoming footholds for deeper compromise. The control network or OT segment comprises OpenPLC instances that emulate programmable logic controllers and ScadaBR as the SCADA/HMI application; together these components expose process variables, registers, and operator interfaces that attackers can read or manipulate. Finally, external infrastructure—represented in the testbed by a simulated attacker C2 server and exfiltration endpoints—models typical command-and-control and data exfiltration channels.

Mapping the attack flow to concrete services and artifacts clarifies what artifacts should be captured during experiments: reconnaissance produces DMZ port-scan and web-fingerprinting artifacts in perimeter access logs and gateway logs; initial access yields HTTP requests containing Log4Shell payloads, logged WAR upload events in Tomcat, and suspicious authentication attempts visible in application logs; execution produces process creation traces, shell or PowerShell invocations, scheduled task creation, and outbound C2 beacons; lateral movement shows up as remote service usage and authentication traces (RDP/SMB/WinRM) and credential dump artifacts in host logs; command-and-control manifests as periodic beacons, DNS or HTTPS anomalies, and proxy logs; and impact is recorded in OpenPLC register writes, ScadaBR operator alerts, setpoint changes, and historian telemetry. To support reproducible analysis, the recommended artifact inventory includes PCAP network captures, flow exports, Tomcat and web server logs, host telemetry (e.g., Sysmon/Windows Event logs), OpenPLC register snapshots and read/write traces, ScadaBR event logs, IDS/SIEM alerts, and a label set tying timestamps to ground-truth attack steps. An explicit topology and metadata package (IP mappings, time synchronization details, campaign identifiers) is required so that each captured artifact can be traced to the appropriate testbed service and action.

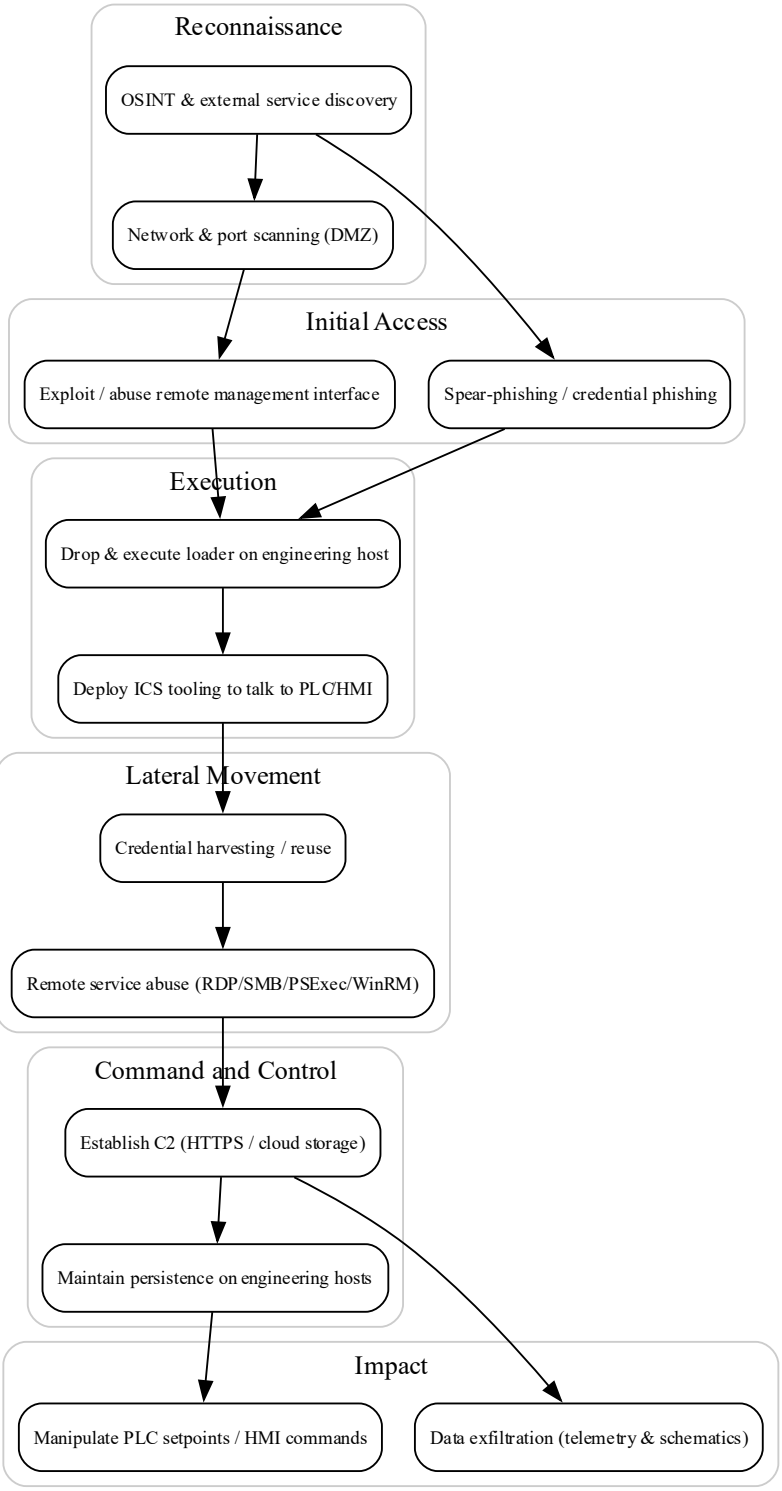

Figure 28: OT/ICS attacks based on Log4Shell and Tomcat vulnerabilities

4.4.2. Dataset preparation.

Dataset preparation for this scenario is guided by three design goals: faithful ground-truth alignment, multi-modal capture, and label richness. A synchronized ground truth timeline is essential: all hosts, PLCs, and capture appliances must be NTP-synchronized and every attacker action annotated with precise start and end times so that artifacts can be labeled at event and window granularity.

The preparation workflow begins by defining distinct attack campaigns that vary payload formats, deployment filenames, timing, lateral paths, and persistence mechanisms; each campaign receives a unique identifier so datasets can be partitioned without leakage. Before executing attacks, baseline captures of normal operation are recorded for a meaningful period in order to characterize benign behavior. During each campaign the team collects raw PCAPs, exports or derives flow records, captures application logs (Tomcat access and error logs, catalina.out), gathers host telemetry such as process creation and command-line activity, and records PLC and SCADA telemetry including setpoint writes and sensor readings. Immediately following attack runs, recovery traces are captured to document remediation activity. Labeling is performed using the ground truth timeline: every artifact is annotated with an attack stage (recon, initial access, execution, lateral movement, C2, impact), the technique or vulnerability used (for example CVE-2021-44228 for Log4Shell injections), the campaign identifier, the affected asset, and a confidence score where appropriate. Labels are produced at both event level—for supervised detection—and at window level for time-series anomaly detection.

Preprocessing derives features appropriate to each modality: for network data, flow durations, byte/packet statistics, inter-packet timing, TLS/J A3 fingerprints, and HTTP header and payload heuristics; for host telemetry, process tree features, parent process names, command-line entropy, newly created binaries and scheduled task events; for application logs, request path patterns, presence of JNDI strings or other injection patterns, and abnormal POST sizes; and for PLC telemetry, deltas in setpoint values and frequency of control writes. Sliding windows of multiple sizes (e.g., 1 s, 10 s, 60 s) are computed to produce sequences for temporal models, and normalization is applied using statistics computed on the baseline training set only. Because attack activity is typically sparse relative to baseline, the preparation stage also addresses class imbalance: options include oversampling attack windows, synthetic augmentation of network traces, or conservative use of SMOTE-style methods for tabular representations.

Finally, datasets are partitioned by campaign into training, validation, and test sets—keeping entire campaign traces in a single split to prevent leakage—and packaged with raw and processed data, labels, topology, campaign definitions, and ingestion scripts to ensure reproducibility.

4.4.3. Training and Validation

Training and validation pipelines are designed to achieve several goals: reliable detection of attack activity across modalities, accurate attribution to the implicated service or component, estimation of impact on process variables, and robust generalization to unseen campaigns and variations. Model classes span unsupervised anomaly detectors (e.g., autoencoders, Isolation Forests) for cases with limited labels; supervised classifiers such as Random Forests, XGBoost, or deep temporal models (LSTM, Temporal CNNs, Transformers) for labeled window detection; and regression models for quantifying process deviations. Temporal sequence models capture behavioral dependencies over time, and multi-modal fusion strategies—either feature-level concatenation or decision-level ensembles—are used to combine signals from network, host, and PLC data. Attribution and root-cause analyses are framed as multi-label classification problems that predict both the attack stage and the target asset; explainability techniques such as SHAP or LIME are recommended to highlight which features drive detections and to support operator investigation. Training proceeds with careful feature selection informed by domain knowledge, campaign-level cross-validation (leave-one-campaign-out) to evaluate generalization, and hyperparameter optimization via grid search or Bayesian methods using validation campaigns. Class imbalance is managed with class weighting, focal loss for neural models, or oversampling; evaluation emphasizes metrics robust to imbalance such as precision, recall, F1, and area-under-the-precision-recall curve (AUC-PR), as well as operational metrics such as false positives per asset per day and detection latency. Validation strategies include testing on unseen campaigns with different payload encodings to simulate zero-day conditions, domain-shift tests that vary baseline operational loads to assess resilience to concept drift, and ablation studies to quantify the contribution of each modality. In practical terms, a training pipeline ingests aligned windows of flows, host features, and PLC time series, normalizes them on training baseline statistics, and trains a temporal classifier with campaign-held-out validation; model checkpoints and early stopping guard against overfitting, and score calibration on validation data yields per-asset thresholds that meet operational false positive targets. For deployment, per-service threshold tuning, alert prioritization through cross-modal fusion (for example, correlating a suspicious WAR upload with a spawned process and subsequent PLC write to raise high-priority incidents), retention of raw forensic captures, and an operator feedback loop for continual learning are recommended to maintain effectiveness over time.

4.5. DoS, Port Scans, and OWASP ZAP Scans

Attacks such as DoS/DDoS and port scan (i.e., *nmap*) are among the most common attack types that are identified by Intrusion Detection Systems (IDS). While DoS/DDoS attacks aim to bring down a system, port scan activities can reveal services that are exposed to external networks, helping to identify attack vectors. Furthermore, while less commonly used than the previous two,

the OWASP Zed Attack Proxy (ZAP) tool [3] can be used to perform special scans that target web applications, attempting to dig out web-specific vulnerabilities.

Mitigation: A machine learning-based IDS that trains on a dataset comprising such attacks would be effective against such malicious activity. Once the model learns how to differentiate these attacks from benign flows, a high detection rate could be reached.

4.5.1. Testbed & Service Mapping

The dataset is generated and tested on the OpenStack-based cloud environment of ZHAW. A Kubernetes cluster with 5 nodes is used, each of which is an Ubuntu-based VM running on OpenStack, and Cilium is leveraged as the Container Network Interface (CNI). A web application consisting of four components (backend, frontend, external service, and database) is deployed on this Kubernetes cluster, using multiple namespaces to simulate the multi-tenancy aspect of modern web applications. While the web application creates benign flows across components, malicious pods are injected into the cluster to generate attack traffic.

DoS: A pod spins up in each namespace and sends heavy amounts of HTTP load towards a randomly elected benign pod in the same namespace

Port Scan: A pod spins up in a randomly selected namespace and conducts a TCP SYN scan on the entire subnet of the pod.

ZAP Scan: A pod spins up in each namespace and uses the ZAP tool to scan the benign frontend pods in the same namespace.

4.5.2. Dataset preparation.

The benign web application has been ran on the testbed for about 2 hours, and waves of attacks of different types (DoS/Port Scan/ZAP Scan) have been triggered on regular intervals. Moreover, during this period, the packet-level network traffic, both benign and malicious, passing through every node in the Kubernetes cluster have been gathered into a central Open Telemetry backend to create a tabular CSV data, which comprises the raw dataset. Since the ratio of attack data to benign data has been quite high for an anomaly detection dataset, a subsampling mechanism has been used to bring the ratios to a more realistic setup.

Afterwards, the raw dataset is preprocessed by using various feature encoders, such as IP encoder, string encoder, boolean encoder and number encoder. By leveraging feature engineering, another new feature called *diversity_index* has been constructed; to indicate how diverse a specific network packet is within a certain time window. Finally, based on the similarity of network packets with each other, a connected graph is generated, which is used as the actual training/validation test.

4.5.3. Training and Validation

As the IDS model, a Graph Convolutional Network (GCN) is leveraged with the following properties:

- Three convolution layers
- One dropout layer
- ReLU as activation functions
- Softmax as the output function

The constructed graph-formed dataset is fed into this model for both training and validation purposes. The nodes of the graph dataset are randomly split into training, validation, and testing in 80%, 10%, and 10%, respectively. The validation split is used to prevent overfitting, and the testing split is used to report the final model's performance. Random Forest and SVM are chosen as baseline models, and it has been shown that the proposed GCN-based IDS managed to achieve 99.9% of accuracy and F1-score in both single-class classification (i.e., anomaly vs benign) and multi-class classification (i.e., detecting attack class specifically) tasks.

4.6. DoSt Attack

The Denial of Sustainability (DoSt) attack targets cloud-native services by overwhelming them with oscillating request patterns that continuously trigger rapid scale-in and scale-out events. Unlike traditional DDoS, DoSt does not fully take services offline but instead causes excessive CPU and memory utilization, leading to gradual QoS degradation and unsustainable energy consumption. In Kubernetes environments, this oscillatory load places significant strain on the control plane, as frequent scheduling and resource reallocation are required to maintain service availability. The subtle nature of the attack makes it harder to detect, as services appear "alive" while their performance and efficiency deteriorate over time.

4.6.1. Testbed & Service Mapping

The DoSt attack scenarios are deployed within the NCL edge–cloud testbed to assess system resilience and sustainability under adversarial workloads. Containerized demand-generation clusters emulate benign and malicious user behavior, generating oscillatory HTTP traffic that stresses CNF CPU and memory resources without causing service downtime. Prometheus telemetry collects fine-grained resource metrics, stored in the TSDB and later persists in MinIO for processing. The custom DoSt datasets generated in Service 1 – Component 1(Energy-Efficient over edge-cloud) will be used within Service 15 – Component 2 (Distributed federated Learning across edge-cloud) to train distributed models across the edge–cloud continuum for predictive scaling and anomaly detection. Currently public google trace datasets have been used to build the baseline of Federated Learning.

4.6.2. Dataset preparation.

Currently public google trace datasets have been used to build the baseline of Federated Learning. Two models will be trained using the prepared datasets — one for resource prediction to inform scaling decisions in Service 3 (orchestrator), and another for traffic classification to distinguish benign from DoSt-induced behavior. Preprocessing and structuring will support both models, ensuring aligned feature extraction for resource trends and anomaly patterns. The DoSt dataset is generated from Prometheus telemetry, which scrapes raw CPU, memory, and network metrics every 15 seconds. These metrics are stored in the TSDB and periodically exported to MinIO object storage for scalable and persistent dataset management. For training and testing, historical Google workload traces are currently used, while custom DoSt datasets are being prepared to support future model training and validation in realistic attack scenarios.

4.6.3. Training and Validation

Benchmarking of ML models for workload prediction has been completed, and federated XGBoost is currently used for resource prediction using Google workload traces as the baseline for distributed resource prediction across edge and cloud nodes, enabling the orchestrator to make energy-aware and secure scaling decisions. Training and validation will be performed across distributed nodes using partitioned datasets to evaluate model consistency and accuracy. A traffic classification model to distinguish between benign and DoSt-induced patterns will be developed in future stages.

4.7. Mirai botnet attack

Although Mirai was first identified several years ago, it continues to pose a significant threat to internet service providers (ISPs) by generating large-scale distributed denial-of-service (DDoS) attacks. The modular nature and constant evolution of this botnet allow it to adopt new attack strategies and propagation techniques, making it a recurring topic of research and security.

At HES-SO, our aim is to create representative datasets which reflect the diversity of Mirai attack patterns observed in real-world scenarios. These datasets will serve as a basis for evaluating and improving detection models capable of recognizing classic and emerging variants of Mirai-generated DDoS traffic.

4.7.1. Testbed & Service Mapping

The Mirai trace sets were generated in a controlled environment at the HES-SO facilities. The test bench is based on several Raspberry Pi 5 devices, each of them running Mirai in an isolated virtual machine. The bots are controlled via a takeover mechanism that allows the botnet to be taken

over and various attack scenarios to be reproduced while maintaining the security and reproducibility of the experiments.

At this stage, local traces are not yet combined with public datasets (e.g. CIC-DDoS2019) — this remains a future option to enrich the diversity of samples and evaluate the generalisability of the models. The primary intention is to produce traces that accurately reflect Mirai's specific behaviour, without mixing traffic generated by other attack tools (HPING3, LOIC, HOIC, etc.), to obtain a representative repository of Mirai variants.

The types of traffic reproduced may include TCP SYN floods, UDP amplifications and other vectors observed in Mirai campaigns; the testbed allows the intensity, synchronisation and composition of attacks to be adjusted to study realistic cases.

The associated service components are:

S9-C1: *Setting up of a Mirai botnet, so as to be able to probe and analyze it*.

S9-C2: *Developing a FPGA-based hardware device capable of detecting various types of DDoS attacks*.

The current development phase focuses primarily on Mirai traffic, providing a concrete use case for testing detection and inference strategies.

4.7.2. Dataset preparation.

For the training and testing phases, we are currently using the CIC-DDoS2019 dataset, which is publicly available on KaggleHub. This dataset was originally generated using the CICFlowMeter tool developed by the Canadian Institute for Cybersecurity (CIC) and retains a total of 77 flow-based features.

Our locally generated datasets consist of raw PCAP captures produced by the Mirai test bench. These captures are processed using CICFlowMeterV4 to extract network flows and export them to CSV format, which are then converted to Parquet files for efficient storage and processing.

So as to ensure compatibility with the training data, we keep an exact alignment of features: the same 77 features are kept in the same order as those in the CIC-DDoS2019 dataset. A mapping procedure is also applied to harmonize class labels, as minor discrepancies may arise between versions of CICFlowMeter (e.g., differences in naming conventions or label coding).

This pre-processing pipeline ensures that locally captured Mirai traces can be seamlessly integrated into the same analytical framework as the reference dataset, facilitating future cross-dataset evaluations and domain adaptation experiments.

4.7.3. Training and Validation

As aforementioned, the training and validation processes are currently being conducted using the CIC-DDoS2019 dataset, with XGBoost selected as the reference model. For this initial phase, the original feature set of the public dataset has been retained to ensure comparability with existing benchmarks. The dataset was divided into training and test subsets, and standard evaluation metrics were applied to assess model performance.

The experimental results show that the model achieves an overall classification accuracy of 94.2%. High-volume attack types and benign traffic, including NTP, SYN, and TFTP flows, are detected with near-perfect F1 scores ranging from 0.99 to 1.00. Conversely, rare classes such as DrDoS_MSSQL (F1 = 0.18) and DrDoS_LDAP (F1 = 0.31), as well as some WebDDoS flows, show significantly lower detection performance. These disparities underscore the model's sensitivity to class imbalance and the need to implement strategies to address data bias, such as resampling or cost-sensitive training.

Analysis of locally captured Mirai traces revealed that, although TCP SYN-based attacks are often detected, their class labels are frequently misattributed. Furthermore, harmless application-level traffic, particularly RTSP streaming flows, can sometimes be misclassified as attack traffic. This indicates a discrepancy between the CIC-DDoS2019 training data and actual Mirai traffic, which affects the model's ability to generalize novel attack behaviours.

Overall, these results outline two key points: the need to create dedicated, well-labelled datasets specific to Mirai, and the potential benefits of exploring more advanced modelling approaches, such as architectures incorporating temporal or hybrid features. The developed test bench therefore plays a crucial role in producing realistic Mirai attack scenarios, which will support future efforts to improve detection robustness and cross-domain generalization.

4.8. Data for JASMIN training and evaluation

This dataset contains time-domain signal representations (I/Q samples) for the IEEE 802.11p protocol, covering all supported modulation schemes: BPSK, QPSK, 16-QAM, and 64-QAM. It includes two scenarios: clean signal and jamming attack. The data was generated using an SDR setup with varying distances between transmitter, receiver, and jammer, and across different SNR levels.

4.8.1. Testbed & Service Mapping

An SDR IEEE 802.11p (V2X) setup with three USRP B210s at 5.9 GHz—Tx, Rx, and a jammer—drives the experiments; each USRP (omni antennas, USB 3.0) is controlled via GNU Radio on NVIDIA Jetson Orin hosts, with Tx/Rx PHY based on WiMe and the jammer mirroring the Tx flowgraph while injecting white Gaussian noise; I/Q packets (128 complex samples) are captured via API to a shared DB under varying Tx–Rx–jammer distances to emulate a vehicular service (base-station ↔ AV) and enable real-time jamming detection.

4.8.2. Dataset preparation.

Two splits are used: (i) training with clear (unjammed) packets across BPSK/QPSK/16-QAM/64-QAM and wide SNRs (including <0 dB), and (ii) evaluation with both clear and jammed packets; jamming combines reactive and periodic modes (jammer activates on signal detect and aligns with Tx periodicity), yielding highly positive-SNR jammed bursts; per-modulation counts and SNR stats are reported (e.g., evaluation: clear—BPSK 2539, QPSK 4259, 16-QAM 3063, 64-QAM 1292; jammed—BPSK 8465, QPSK 9098, 16-QAM 16539, 64-QAM 22978) and the full dataset is released on Zenodo.

4.8.3. Training and Validation

Training uses only unjammed data: an LSTM (input 128×2 I/Q, 128 units, dropout 0.5, Adam, early stopping) is tuned across SNRs, and per-modulation Isolation Forest for outlier detections (150 trees) are trained on features derived from constellation distances (RSE-based, quadrant reduction, Manhattan distance, intra-packet point permutation) to set contamination per scheme; at run-time, windows of P = 21 packets (10 MHz front-end) yield >3.5k decisions/s, and on the evaluation split JASMIN attains 99.92% overall accuracy (perfect for BPSK/16-QAM/64-QAM; QPSK ≈ 99.6%), with OD outlier rates on jammed data of ~87% (BPSK), ~76% (QPSK), ~94% (16-QAM), and ~97% (64-QAM).

4.9. Eavesdropping attack on PKG

We study a passive eavesdropper (Eve) targeting the AI-Enhanced Physical Key Generation in sub-THz service by recording UL/DL pilots in Gradiant 5GLab and mirroring the public PKG steps to attempt key replication. The evaluation quantifies attack ineffectiveness via KDR across distance, position, and mobility, and assesses that gains from the AI reciprocity module on the main link do not translate into any advantage for Eve.

4.9.1. Testbed & Service Mapping

We evaluate the service AI-Enhanced Physical Key Generation in sub-THz under a passive eavesdropping threat in an indoor laboratory TDD setup. The legitimate endpoints, Alice and Bob, follow this pipeline: the Characteristics Extractor derives channel features from uplink and

downlink pilots; on Alice's side, the Network Key Generator applies AI-driven channel-reciprocity enhancement, then quantization, information reconciliation, and key generation via hashing. On the other hand, Bob runs the symmetric branch (quantization, reconciliation, and hashing) to obtain the same secret key. The adversary, Eve (USRP B210), is a third node co-located in the testbed that records the sub-THz UL/DL transmissions between Alice and Bob without injecting traffic. Eve locks onto the signals using standard pilot-aided timing and frequency estimation (no shared clock or common RF front-end is assumed) and moves along the legitimate path, including sub-wavelength displacements, to seek positions where her channel observations approach Bob's. The attack targets the entire PKG pipeline by mirroring the public steps: channel-measurement extraction, quantization, reconciliation and privacy amplification.

4.9.2. Dataset preparation.

The attacker dataset consists of controlled sub-THz TDD captures collected at Eve's location. For each capture, Eve records the uplink and downlink pilot signals transmitted by Alice and Bob and derives complex channel estimates, amplitude and phase across OFDM subcarriers, for both directions, together with timestamps and scenario tags. The campaign spans multiple distances, positions (including sub-wavelength offsets along the Alice–Bob path), and mobility conditions in the indoor lab to ensure diversity. The attacker pipeline relies solely on Eve's observations and publicly exchanged messages.

4.9.3. Training and Validation

The training stage applies only to the PKG service AI-driven channel reciprocity enhancement. Using Alice/Bob pilot observations, the model is trained to improve UL–DL reciprocity on the main link (by mapping UL to DL features), thereby increasing bit agreement before reconciliation. This training does not involve Eve's data and does not provide the attacker with any advantage. Validation focuses on security against passive eavesdropping. The primary metric is the Key Disagreement Rate (KDR) between Eve's final key and the legitimate final key after reconciliation and privacy amplification. Equivalently, KDR can be computed against Bob, since Alice and Bob are designed to agree post-reconciliation. Ineffective eavesdropping yields KDR close to 0.5 and an attacker secret-bit rate effectively zero. We report KDR across distances, positions, and mobility scenarios, emphasizing that the attack is most effective only at very short separations (sub-wavelength scales) and in TDD conditions that favor reciprocity. Even then, at practical separations the resulting keys remain indistinguishable to Eve. KDR thus acts as the service-level assessment under attack, and by comparing variants with and without the AI reciprocity module it also provides a component-level assessment, confirming that legitimate-link gains do not translate into any advantage for Eve.

4.10. Jamming attack

Jamming attacks are a common form of active interference in wireless communications, aimed at degrading or completely denying service. As mentioned in D5.1, there are several different types of jammers depending on their strategy, but most of them pose a trade-off between energy consumption and effectiveness. One of the ways for a jammer to increase its efficiency relies on the knowledge of the specific signal it wants to attack. In our case, we will assume the reasonable scenario in which the jammer knows the 5G band that we are using and attacks a portion of the 5G spectrum, affecting some of the available PRBs but not all at the same time, as that would imply a high energy consumption.

The jammer will use previously generated chirp signals and transmit them using a USRP. This is not a dataset per se, as the amount of different chirp signals is not so large, as the center frequency and gain are selected using the USRP.

4.10.1. Testbed & Service Mapping

In the 5GLab testbed (specifically within the AI anti-jamming subsystem), jamming attacks are generated with a USRP to disrupt legitimate communications between a UE and the gNB in the 5G n77 band. The jammer's bandwidth is configurable so that it can target particular PRBs within the 5G band. The controlled injection of these attacks is an integral part of the testbed and is required to validate the DetAction component.

4.10.2. Dataset preparation.

The dataset used by the jamming-detection component is built from previously captured chirp signals recorded under a variety of sample rates, bandwidths and channel conditions. Samples are split into two categories: non-jammed signals (i.e., 5G communications plus background noise) and jamming signals (chirps), which may appear either in isolation or superimposed on 5G traffic. Note that the testbed jammer transmits pre-generated chirp waveforms via the USRP; this collection is not treated as a conventional training dataset in the sense of a large, diverse corpus of signals, as there are only a limited number of distinct synthetic chirps generated. The center frequency and transmit gain of the jammed chirps are configured directly on the USRP.

4.10.3. Training and Validation

The DetAction detection phase is trained using the dataset described above. Preprocessing extracts spectral fragments that correspond to PRB-sized blocks (using 180 kHz PRBs according to the 5G numerology) from the captured 5G and jamming signals. These fragments are used to train a convolutional neural network (CNN) that currently achieves AUC = 0.97, F1-score = 0.98 and accuracy = 0.95 in validation. Validation is performed on a held-out set of captured signals

not used during training, and the system will be further tested in the 5GLab testbed with additional signals emitted by the jamming USRP and captured in real time.

5. Conclusions

D6.2 “System Integration on the testbeds, Pilot installations and implementations.r1” deliverable, addressed the set-up of NATWORK’s testbeds, the installation of the components that were identified in previous stages of NATWORK project, and the initial validation of the components through dry run tests. In addition, several attacks were identified and, through the Attack Generation System, were emulated on NATWORK’s components. By determining these attacks, also the related datasets per attack were identified.

During the initial assessment of verifying the NATWORK’s components, 14 testbeds by thirteen 13 partners were set up. Testbed owners provided their testbeds on time, allowing dry run tests to be executed in a thorough way. Specific components were set up in more than one testbed. At the Use Case (UC) level, UC functionalities were shared across multiple testbeds. Preliminary tests were conducted for the corresponding components of NATWORK, identifying that all components were successfully installed in the related testbed or testbeds. Moreover, mature components were validated through a set of test scenarios identifying that NATWORK components are ready for NATWORK’s trials in controlled lab environments.

The dry run tests that were performed during this period, focused on the verification of the mature components and the results are presented in this deliverable. Validation of the components was performed by the component owners. Nonetheless, during the validation of the components, a solid collaboration between component owners and testbed owners was performed. The report of the results is presented per component. This also applies in cases where a component was installed in more than one testbed; a single report per component is created. The actual results from the dry run tests of the components can be found in the Appendix of this document.

T6.2 “Testbed integration & attack generation system” also had an objective to identify and emulate specific attacks towards NATWORK system, through the Attack Generation System. These attacks have been triggered against the related NATWORK components, and the related results have been reported in this deliverable. Initial security breaches in the overall NATWORK framework have been recorded in D6.2 when these attacks were forwarded to NATWORK.

5.1. Next steps

The present deliverable represents the first version of “System Integration on the testbeds, Pilot installations and implementations”. A second version of this report will be submitted in M32 when T6.2 concludes, and T6.3 is about to be completed in M33. This successive report will provide an additional outlook on the status of NATWORK’s components from the T6.2 perspective. Further tests will be performed, and additional attacks will be identified and

emulated towards NATWORK components. Adjustments to the testbeds can be applied when needed. Any adjustments will be also reported in the second version of “System Integration on the testbeds, Pilot installations and implementations”.

In the second report, more information from T6.3 “Use Cases Trials and Demonstration” will also be presented. The finalized definition and set-up of the controlled environments for each use case will be described. This task will also execute the use case trials in those prepared NATWORK environments, evaluate their initial results and demonstrate them. In that regard, the attack generation system from T6.2 will be used to the greatest extent possible. Moreover, experimental data on system performance, security metrics, and end-user feedback will be gathered. By doing so, an extensive evaluation of NATWORK’s outcomes will be performed. Therefore, the use case trials that will be carried out through the timeline of T6.3 will be reported in D6.3 deliverable. Overall, the two versions of this report jointly detail the testbed infrastructure and the related NATWORK for the upcoming pilot trials and the evaluation of the NATWORK system.

6. References

[1] Sev-Snp, A. M. D. (2020). Strengthening VM isolation with integrity protection and more. *White Paper, January*, *53*(2020), 1450-1465.

[2] Ejaz, S., & Al-Naday, M. (2024, March). FORK: A Kubernetes-compatible Federated Orchestrator of Fog-native applications over multi-domain edge-to-cloud ecosystems. In *2024 27th Conference on Innovation in Clouds, Internet and Networks (ICIN)* (pp. 57-64). IEEE.

[3] Zed Attack Proxy (ZAP), https://www.zaproxy.org/ [Accessed: 09.10.2025]

Appendix

In the appendix, the actual test scenarios per component are presented. Specific components have been already verified during months M1 and M22 of NATWORK project timeline. The actual dry run test results of these components can be found in the related tables below. In addition, test scenarios and test cases that have been already identified for the related components and are currently under evaluation (no results for these scenarios have been achieved yet) are also displayed in the current report. Components that the related test scenarios have not yet been identified and the results of components that are under evaluation will be presented in the second version of "System Integration on the testbeds, Pilot installations and implementations" which is deliverable D6.3 due M32.

In the information below, the current status of the test scenarios and test cases per component are illustrated.

A.1 Energy efficient over edge-cloud

Project Name:	**NATWORK**
Component Name:	**Energy-efficient/delay-aware orchestration**
Created by:	**UEssex**
Date of creation:	**01.09.2025**
Filename:	**UEssex-Energy-efficient.xlsx**

Test scenario ID	Test scenario	Tets case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
CPU utilization measure energy consumption - TS01	Verify benign workload in single- and multi-cluster setups	CPU utilization measure energy consumption - TS01-TC01	Single-Cluster vs Multi-Cluster traffic, impact of service discovery (MCS API)	Containerized application and proxy deployed in Kubernetes clusters	1. Deploy application (single + multi) 2. Send benign traffic across clusters 3. Measure RTT and compare	Simulated benign containerized workload	Multi-cluster shows higher RTT variance due to MCS API overhead	01/06/2025	Multi-cluster shows higher RTT variance due to MCS API overhead + virtualisation overhead	Pass
CPU utilization measure energy consumption - TS01	Demonstrate Denial of Sustainability (DoST) attack	CPU utilization measure energy consumption - TS01-TC02	Generate oscillating HTTP requests to cause CPU/memory oscillation and QoS degradation (harder to detect)	Containerized application and proxy deployed in Kubernetes clusters	1. Deploy service and proxy 2. Simulate containerized end users generating oscillatory HTTP traffic 3. Measure RTT, CPU/memory oscillations, QoS degradation	DoST workload	Service remains alive but shows high oscillations, scaling pod resources in/out quickly degrading QoS and	18/06/2025	Service remains alive but shows high oscillations, scaling pod resources in/out quickly degrading QoS and longer RTTs	Dost attack demonstration – Pass Mitigation – Not tested yet

Test scenario ID	Test scenario	Tets case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
							longer RTTs			
Monitoring and Dataset generation using DosT attack to feed Machine learning - TS02	Monitor and log system resources for AI-driven analysis	Monitoring and Dataset generation using DosT attack - TS02-TC01	Prometheus-based monitoring of CPU, memory, network TX/RX, and pod lifecycle stats	Containerized application and Prometheus deployed on Kubernetes clusters	1. Deploy Prometheus & Grafana in cluster 2. Visualise CPU/memory/network telemetry during attack 3. Store time-series data in TSDB	Prometheus telemetry - DosT workload traffic (benign/malicious)	Structured telemetry datasets stored, visualized via Grafana dashboards, and prepared for federated learning	18/06/2025	Structured telemetry datasets stored, visualized via Grafana dashboards, and prepared for federated learning	Pass

A.2 TrustEdge

Project Name:	**NATWORK**
Component Name:	**TrustEdge**
Created by:	**UGent**
Date of creation:	**23.09.2025**
Filename:	**IMEC-TrustEdge.xlsx**

Test scenario ID	Test scenario	Tets case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
Boot-TS02	(Added) boot time of the framework from attestation to secure Feather deployment	Base-TS01-TCO1	Measures the total time added to device boot time by the framework	Running Kubernetes cluster	1. Prepare evaluation edge node 2. Repeatedly reboot & connect to Kubernetes cluster for attestation 3. Measure attestation overhead & Feather start time	N/A	<60 second added time	10/04/2024	Average case 20.91s time added to boot, around half of which is Feather starting and half is the attestation process	Pass

A.3 Feather

Project Name:	**NATWORK**
Component Name:	**Feather**
Created by:	**IMEC**
Date of creation:	**23.09.2025**
Filename:	**IMEC-Feather.xlsx**

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
Runtime comparison benchmark -TS01	Compare unikernels (OSv/KVM), containers, and/or WASM	Minimal - TS01-TCO1	Measures the overhead of Feather for containers, unikernels	Feather agent up and running	1. Prepare test device (containerd installation, KVM-qemu) 2. Start	Idle workload images, deployment manifest	Slight increase in memory use compared to idle	15/04/2025	Minimal (<1%) CPU use, memory overhead 7-13MB with the lowest	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
	(WASMTime) performance under different scenarios		and WASM by deploying an idle workload to all backends		Feather agent 3. Deploy workload 4. Run evaluation & metrics script		Feather, no CPU consumption.		overhead for WASMTime	
Runtime comparison benchmark -TS01	Compare unikernels (OSv/KVM), containers, and/or WASM (WASMTime) performance under different scenarios	Application -TS01-TCO2	Measures the overhead of Feather with active containers and unikernels. Measures the resource consumption of a Minecraft server in both container and unikernel format to gauge benefits of runtimes.	Feather agent up and running, suitable workload images prepared.	1. Prepare test device (containerd installation, KVM-qemu) 2. Start Feather agent 3. Deploy workload 4. Run evaluation & metrics script	Workload images, deployment manifest	Significant rise in memory use, unknown benefits for either runtime	/	No memory overhead for Feather w.r.t. minimal scenario. Huge (>30%) performance overhead for KVM-based unikernel, but also uses 30% less memory than containerized MC server.	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
Runtime comparison benchmark -TS01	Compare unikernels (OSv/KVM), containers, and/or WASM (WASMTime) performance under different scenarios	Images - TS01-TCO3	Measures the relative size of an image for specific functionality (HTTP server) in different runtime formats.	N/A	1. Prepare image contents (compilation) 2. Build image (Docker, Flint, ...) 3. List image size	Workload images	Smaller WASM image, likely larger microVM image compared to container	15/04/2025	WASM image smallest (0.2KB), followed by OSv unikernel (7.3MB) and native (container image) 28.2MB.	Pass
Runtime comparison benchmark -TS01	Compare unikernels (OSv/KVM), containers, and/or WASM (WASMTime) performance under different scenarios	HTTP - TS01-TCO4	Measures performance of an HTTP server in various runtimes. Considers latency as well as raw request throughput using k6 command.	Feather agent up and running, suitable workload images prepared. All workloads reachable through container networking.	1. Prepare test device (containerd installation, KVM-qemu) 2. Start Feather agent 3. Deploy workloads 4. Run evaluation & metrics script	Workload images, deployment manifest	WASM performance similar to native. Previous OSv versions produced unikernels on par with native on XenServer, but overloading HTTP traffic was a weak spot and can cause bottlenecks and latency spikes on KVM-Qemu. This test case uses a	15/04/2025	WASMTime latency 50% higher than native; request/s keeps pace with native. OSv on KVM latency 500% higher than native, requests/s eventually crashes when latency skyrockets. Unfortunate, but somewhat expected.	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
							newer version.			
Container network performance-TS02	Performance of decentralized multi-runtime container networking solution.	Throughput-TS02-TC01	Measures throughput of the decentralized P2P internode part of the networking solution compared to WireGuard.	Feather agent up and running, suitable evaluation images prepared (HTTP server, video streaming server). All nodes mutually reachable on public IPs.	1. Prepare test devices (containerd installation, KVM-qemu) 2. Start Feather agents on all devices 3. Start internode networking processes, validate eBPF running 4. Deploy workloads 5. Run evaluation & metrics scripts	Workload images, deployment manifests, HTTP request scenario	Lower CPU use than alternatives (WireGuard), 1Gbps internode traffic possible	15/05/2024	CPU use 10-100 times lower than WireGuard in same setup, 1Gbps physical connection saturated (~910Mbps + packet and communication overhead)	Pass
Container network performance-TS02	Performance of decentralized multi-runtime container networking solution.	Scalability-TS02-TC02	Measures throughput of the decentralized P2P internode part of the networking solution in 5-node star and ring	Feather agent up and running, suitable evaluation images prepared (HTTP server, video streaming server). All	1. Prepare test devices (containerd installation, KVM-qemu) 2. Start Feather agents on all devices 3. Start	Workload images, deployment manifests, HTTP request scenario	Scaling dependent on number of neighbours, not total topology size.	15/05/2024	Star topology shows similar throughput as WireGuard. Ring topology shows ~910Mbps traffic for all nodes (saturated physical	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
			topologies to evaluate scalability. In the ring topology, WireGuard uses one of the nodes as VPN controller.	nodes mutually reachable on public IPs.	internode networking processes, validate eBPF running 4. Deploy workloads 5. Run evaluation & metrics scripts				interface) and ~230Mbps for WireGuard.	
Container network performance-TS02	Performance of decentralized multi-runtime container networking solution.	Throughput-TS02-TC03	Measures throughput of the multi-runtime (node local) part of the networking solution.	Feather agent up and running, suitable evaluation images prepared (HTTP server, video streaming server). Feather multi-runtime network solution running (configuration).	1. Prepare test device (containerd installation, KVM-qemu) 2. Start Feather agent, validate eBPF running 3. Deploy workloads 4. Run evaluation & metrics script	Workload images, deployment manifests, HTTP request scenario	<1% of single CPU core per Gbps interpod traffic, multi-Gbps traffic possible	12/07/2024	Added latency <100µs, average 2.5Gbps traffic sustained in video streaming scenario independent of endpoint runtimes, 1%-1.5% of single CPU core.	Pass

A.4 Flocky

Project Name:	**NATWORK**
Component Name:	**Flocky**
Created by:	**UGent**
Date of creation:	**23.09.2025**
Filename:	**IMEC-Flocky.xlsx**

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
Functional-TS01	Functional evaluation of the framework	Base-TS01-TCO1	The functionality of the framework is measured in terms of metadata discovered (discovery + metadata services) and required services deployed (orchestration metadata use)	N/A	1. Prepare test devices, config files 2. Start Flocky services in order 2.a Node monitoring service (one node) 2.b Deployment & Flocky services 2.c Discovery services 2.d Metadata services 3. Deploy workloads 4. Evaluate situation from node monitoring service after 5	Service config files, deployment manifests	100% metadata discovery, all services successfully placed after <5 rounds	10/02/2025	100% discovery, all services successfully placed after 2-3 rounds (depending on random factors)	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
					discovery rounds					
Scalability-TS03	Scaling efficiency of the framework	Network-TS02-TCO1	Network traffic is measured for topologies from 1 to 150 nodes, for discovery distances from 10 to 20 (ms ping simulated)	N/A	1. Prepare test device, copy config files 2. Start simulation script with node and distance parameters 3. Run metrics gathering script	Service config files	O(neighbours) scaling or less, where neighbours is roughly total nodes * discovery distance	10/02/2025	Network traffic follows O(neighbours) scaling, but rises twice as fast as CPU scaling.	Pass
Scalability-TS03	Scaling efficiency of the framework	Resource-TS02-TCO1	CPU and memory are measured for topologies from 1 to 150 nodes, for discovery distances from 10 to 20 (ms ping simulated)	N/A	1. Prepare test device, copy config files 2. Start simulation script with node and distance parameters 3. Run metrics gathering script	Service config files	O(neighbours) resource scaling or less	10/02/2025	Memory scales as expected, with very low overhead compared to baseline (16MB base -> 21MB at densest scenario). CPU scaling exactly follows	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
									O(neighbours).	
Scalability-TS03	Scaling efficiency of the framework	Discovery-TS02-TCO1	Metadata discovery efficiency is measured for topologies from 1 to 150 nodes, for discovery distances from 10 to 20 (ms ping simulated). Each node is assigned 2 random metadata items at start, and a total pool of 100 must be discovered by each node.	N/A	1. Prepare test device, copy config files 2. Start simulation script with node and distance parameters 3. Run metrics gathering script	Service config files	>99% for topologies where all nodes have at least 2 connections, graceful degradation for sparsely connected topologies	10/02/2025	>99% metadata discovery from 75 nodes and 20 discovery distance upwards, >96% for smaller, loosely connected topologies. This indicates no bifurcation, but single nodes not connected to the topology due to random generation (trivial fix with starting conditions for nodes in realistic and simulated scenarios)	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
Latency-TS03	(Additional) latency added by the framework	Base-TS03-TCO1	Measures the total deployment latency (end to end) of a two-component service in the Flocky framework, from user action to service deployment.	Flocky framework running on two nodes (requester and deployer)	1. Start Wireshark 2. Execute repeated deployment calls 3. Gather latency data from Wireshark and Flocky instrumentation	Deployment manifest	<2.5 second response time excluding image pull and deployment (Kubernetes average with standard reconciliation loop)	10/02/2025	Median case 21.1ms response time for deployment on two separate nodes, consisting of ~70% network latency.	Pass

A.5 Secure-by-design orchestration

Project Name:	**NATWORK**
Component Name:	**Secure-by-design Orchestration**
Created by:	**UEssex**
Date of creation:	**24.09.2025**
Filename:	**UEssex-secure-by-design-orch.xlsx**

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
ORCH-TS01	Validate Secure-by-Design orchestration	ORCH-TS01-TC01	Check if orchestration respects declared security requirements of a 6G Slice and Clusters	Orchestrator (sFORK), Slice requirements and Cluster requirements declarative inputs	1. Define the slice requirements with security constraints 2. Define the clusters' requirements with security constraints 3. trigger orchestration by introducing the inputs declaratively 4. Monitor whether placement/scaling respects constraints	Slice, Slice Requirements and Clusters requirements manifests	Orchestrator decisions comply with Secure-by-Design policies (no insecure placement)	01/07/2025	Orchestrator decisions comply with Secure-by-Design policies (no insecure placement)	Pass
ORCH-TS02	Subgraph communication within orchestration	ORCH-TS02-TC01	Verify if sFORK core component interacts with cluster local orchestrators and disseminates the subgraphs	Orchestrator (sFORK) policy and strategy components to create dependency graphs alined with the slice and slice requirements.	1. Create slice with multiple dependent CNFs 2. sFORK Policy and strategy components decompose the dependency graph, create subgraphs aligned with the slice requirement	Slice manifest with the defined CNF dependency, Slice Requirements and Clusters requirements manifests	Subgraphs are correctly shared and executed across clusters, CNF dependencies are set	01/07/2025	Subgraphs are correctly shared and executed across clusters, CNF dependencies are set	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
					and cluster requirement inputs 3. Distribute the subgraphs to related clusters 4. Observe communication with cluster local agents					

A.6 End-to-End Security Management

Project Name:	**NATWORK**
Component Name:	**E2E Trust Establishment**
Created by:	**ELTE**
Date of creation:	**01.09.2025**
Filename:	**ELTE-E2E-Trust.xlsx**

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
E2E_Trust-TS.01	Verify main connections	E2E_Trust-TS.01-TC.01	gNB connects to the 5G Core	The 5G Core should be up and running, listening to newly joined gNB	Step 1: 5G core starts and running. Listening to the connections Step 2: gNB through UERANSIM	N/A	gNB connection to 5G Core. Updating the log files on both nodes	Aug-24	Connection stable.	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
					performs the preliminary configuration and connects to the relevant 5G Core					
E2E_Trust -TS.01	Verify main connections	E2E_Trust-TS.01-TC.02	UPF connects to the 5G Core	The 5G Core should be up and running	Step 1: 5G core starts and running. Listening to the connections Step 2: UPF through Open5GS performs the preliminary configuration and connects to the relevant 5G Core	N/A	Link between 5G Core and external UPF. Updating the log files on both nodes	Sept-24	UPF registration with 5G Core completed successfully.	Pass
E2E_Trust -TS.01	Verify main connections	E2E_Trust-TS.01-TC.03	UE connects to the UPF	The 5G Core should be up and running. The gNB should be connected to the 5G Core. Link to the UPF should	Step 1: gNB starts and running. Listening to the connections Step 2: UE connects to the specific gNB through the releveant configuration Step 3:	N/A	Full running system, link from the UE through gNB, to the revelenat UPF and DN	Sept-24	Traffic between UE and DN verified through UPF tunnel.	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
				be already established	through 5G Core, the connection is directed to the UPF, which will update the log files					
E2E_Trust-TS.02		E2E_Trust-TS.02-TC.01	Check the blockchain connection	Required dependencies for Foundry (e.g., Rust, Node.js, Foundry toolchain) are installed. Network connectivity is available.	Step 1: Deploy the Foundry blockchain node in a standalone mode (local devnet or testnet). Step 2: Verify the node starts successfully by checking process status and listening ports. Step 3: Run a simple health check using Foundry CLI (e.g., forge test or cast block-number) to ensure the	N/A	Foundry blockchain node is successfully established and running. Node responds to CLI/API queries. Dummy transaction is processed and confirmed in the local chain. Logs show successful startup, block production, and transaction handling.	Oct-24	Foundry blockchain node is successfully established and running.	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
					node is responsive.					
E2E_Trust-TS.03	Blockchain Connetion to other nodes	E2E_Trust-TS.03-TC.01	Blockchain DN node runs	Foundy Blockchain is running. UPF and DN (in the same node, different processes) are connected	Step 1: UPF establishes a data path to the DN where the Foundry blockchain node is running. Step 2: DN node performs a test data transaction.	N/A	Successful communication and transaction handling between UPF and DN blockchain node.	Oct-24	UPF successfully routed data traffic to DN.	Pass
E2E_Trust-TS.03	Blockchain Connetion to other nodes	E2E_Trust-TS.03-TC.02	Blockchain 5G core node runs	Foundy Blockchain is running.	Step 1: 5G Core initiates a blockchain request (e.g., send transaction, query block). Step 2: The Foundry blockchain processes the request and logs the interaction in its transaction records. Step 3: 5G Core receives the blockchain	N/A	Successful blockchain request/response cycle between 5G Core and blockchain node.	Nov-24	Successful blockchain request/response	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
					response confirming successful addition.					

A.7 Slice orchestration and slice management for beyond 5G networks

Project Name:	**NATWORK**
Component Name:	**Slice orchestration and slice management for beyond 5G networks**
Created by:	**CERTH**
Date of creation:	**01.09.2025**
Filename:	**CERTH-Slice-orchestration-and-management.xlsx**

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
Slice-orchestration-management-TS01	Verify proper functionality of the component	Slice-orchestration-management-TS01-TC01	Default operation of 5G network	xAPP with the ML model is loaded	1. Deploy 5G network and xAPP 2. Send test traffic	Default Traffic Generator	Default classification (benign)	11/2024	Default classification (benign)	Pass
Slice-orchestration-management-TS01	Verify proper functionality of the component	Slice-orchestration-management-TS01-TC02	Verify the accuracy of the xAPP in the detection of malicious traffic	xAPP with the ML model is loaded	1. Deploy 5G network and xAPP 2. Send traffic from dataset with attacks	KDD Cup 1999 dataset with attacks	Packets classified as malicious	02/2025	Packets classified as malicious	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
Slice-orchestration-management-TS01	Verify proper functionality of the component	Slice-orchestration-management-TS01-TC03	Identify attack on the network and reallocate PRBs on the slices in order to limit the slice under attack	xAPP with the ML model is loaded	1. Deploy 5G network and xAPP 2. Send traffic from dataset with attacks 3. xAPP calculates the anomaly ratio and reallocate the PRBs on the slices	KDD Cup 1999 dataset with attacks	Reallocation of PRBs on the slices of the network	04/2025	Reallocation of PRBs on the slices of the network	Pass
Slice-orchestration-management-TS01	Verify proper functionality of the component	Slice-orchestration-management-TS01-TC04	Disconnect malicious UE from network	xAPP with the ML model is loaded	1. Deploy 5G network and xAPP 2. Send traffic from dataset with attacks 3. xAPP calculates the anomaly ratio and reallocate the PRBs on the slices 4. xAPP disconnects the malicious UE when anomaly ration reaches 100%	KDD Cup 1999 dataset with attacks	Malicious UE disconnecting from the network	05/2025	Malicious UE disconnecting from the network	Pass

A.8 Signal Processing Services

Components: AI-Based RIS configuration / ML-based MIMO / JASMIN & Filter Mitigation

Project Name:	**NATWORK**
Component Name:	**Signal Processing services**
Created by:	**CERTH**
Date of creation:	**01.09.2025**
Filename:	**CERTH-Signal Processing.xlsx**

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
Signal Processing-TS01	AI-based RIS configuration	Signal Processing-TS01-TC01	Determine the RIS configuration for multi-user scenarios	Optimal RIS configurations for the case that each user is served standalone by it	Step 1: The receiver and the transmitter will be positioned in Line-of-Sight with the RIS unit. Step 2: The communication link quality will be measured with the RIS unit out of function in order to use this measurement as baseline. Step 3: The communication link in case that the user is served standalone will be measured using the optimal RIS configuration. Step 4: The	Extracted by RIS-testbed	Performance per user in multi-user scenario	Early 2026	Not available yet	Not evaluated yet

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
					communication link quality in the multi-user scenario will be measured using the codebook entries multiplexing algorithm for fair beam-splitting.					
Signal Processing-TS02	ML-based MIMO	Signal Processing-TS01-TC02	Verify the results of JASMIN & Filter Mitigation in MIMO setups for receiver and jammer	MIMO antennas in testbed	Step 1: The dedicated protocol for V2X, IEEE 802.11p, will be simulated in the SDR-based setup. Step 2: One SDR will be used for transmitter, one as receiver and one as jammer. Step 3: JASMIN model will be connected with the receiver. Step 4: The output of JASMIN will be measured in case the jammer is inactive. Step 5: The output of JASMIN will be measured in case the jammer is active. Step 6: The outputs in both cases will	Extracted in SDR-testbed	Accuracy of the JASMIN model in clear and jammed data. SNR enhancement before and after the filter mitigation application	Mid 2026	Not available yet	Not evaluated yet

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
					be evaluated based on the ground truth					
Signal Processing-TS03	JASMIN & Filter Mitigation	Signal Processing-TS01-TC03	Detect the attack across all main types (constant, periodic, reactive) in real-time in IEEE 802.11p.	For JASMIN none. For Filter Mitigation, synchronization of the SDR ports is required	Step 1: The dedicated protocol for V2X, IEEE 802.11p, will be simulated in the SDR-based setup. Step 2: One SDR will be used for transmitter, one as receiver and one as jammer. Step 3: JASMIN model will be connected with the receiver. Step 4: The output of JASMIN will be measured in case the jammer is inactive. Step 5: The output of JASMIN will be measured in case the jammer is active. Step 6: The outputs in both cases will be evaluated based on the ground truth	Extracted in SDR-testbed	Accuracy of the JASMIN model in clear and jammed data. SNR enhancement before and after the filter mitigation application	For JASMIN 05/2025. For fliter mitigation planned for early 2026	JASMIN: 99.92%. Filter Mitigation: not available yet	JASMIN: pass. Filter Mitigation: not evaluated yet

A.9 DetAction: Detection and reAction against jamming attacks

Project Name:	**NATWORK**
Component Name:	**DetAction**
Created by:	**GRADIANT**
Date of creation:	**11/09/2025**
Filename:	**GRAD-DetAction.xlsx**

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
DetAction-TS.01	Signal preprocessing validation	DetAction-TS.01-TC.01	Signal DB connection verification	Signal DB running	1. Connect Detection phase with signal DB	N/A	Connection succesful	01/07/2025	Connection succesful	Pass
DetAction-TS.01	Signal preprocessing validation	DetAction-TS.01-TC.02	Signal DB captured signals loading verification	Signal DB connected to Detection phase	1. Obtain signals from the DB using their sign-meta parameters 2. Load the signals' samples	Sigmf-meta from the signals and sigmf-data to load	Signals' IQ samples loaded	01/07/2025	Signals' IQ samples loaded	Pass
DetAction-TS.01	Signal preprocessing validation	DetAction-TS.01-TC.03	Signal resampling verification	Signal IQ samples correctly loaded	1. Using sigmf-meta parameters, obtain the IQ samples original sample rate 2. Resample that signal to the desired sample rate	Loaded IQ samples from sigmf-data	Signal resampled to final rate	01/07/2025	Signal resampled to final rate	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
DetAction-TS.01	Signal preprocessing validation	DetAction-TS.01-TC.04	Signal spectrum fragmentation	Signal IQ samples correctly resampled	1. Obtain signal spectrum using FFT. 2. Split that signal into fragments of the desired length	Resampled IQ samples	Signal spectrum fragments obtained	01/07/2025	Signal spectrum fragments obtained	Pass
DetAction-TS.01	Signal preprocessing validation	DetAction-TS.01-TC.05	Spectrum fragments normalization	Spectrum fragments generated	1. Obtain all fragments from the used signals 2. Obtaing metrics for normalization. 3. Normalize all fragments using said metrics	Spectrum fragments	Spectrum fragments normalized	01/07/2025	Spectrum fragments normalized	Pass
DetAction-TS.02	Detection phase classification validation	DetAction-TS.02-TC.01	Inference classification	Detection phase algorithm previously trained.	1. Obtain IQ samples of the captured signal. 2. Preprocess the signal, obtaining the final spectrum fragments. 3. Assign each fragment its location in the spectrum, simulating a PRB location. 4. Classify	Captured signal.	Inference process runs without errors	01/08/2025	Inference process runs without errors	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
					each of the fragments and show the results for each PRB					
DetAction-TS.03	ReAction PRBs assignation verification	DetAction-TS.03-TC.01	No PRB being jammed	Algorithm priority set (p.e: RoundRobin)	1. Fix a number of UEs and their data rates. 2. Set all PRBs as available (there is no jamming). 3. Let the reAction algorithm assign each PRBs to the UEs	N/A	PRBs are assigned to UEs without any malfunction	01/08/2025	PRBs are assigned to UEs without any malfunction	Pass
DetAction-TS.03	ReAction PRBs assignation verification	DetAction-TS.03-TC.02	Some PRBs being jammed	Algorithm priority set (p.e: RoundRobin)	1. Fix a number of UEs and their data rates. 2. Set a percentage of the PRBs as jammed. 3. Let the reAction algorithm assign each PRBs to the UEs	N/A	PRBs are assigned to UEs without any malfunction	01/08/2025	PRBs are assigned to UEs without any malfunction	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
DetAction-TS.04	Connection between Detection and reAction phases verification	DetAction-TS.03-TC.01	Simulation of connection between Detection and reAction	Detection and reAction working	1. Initiate Detection phase and make it receive and classify a signal. 2. Use its output as input in the reAction phase, simulating a interface between them 3. The reAction phase uses the received output from Detection as its input	N/A	Simulated reAction phase receives and uses correctly the output of the Detection phase	01/08/2025	Simulated reAction phase receives and uses correctly the output of the Detection phase	Pass

A.10 Security-compliant Slice Management

Project Name:	**NATWORK**
Component Name:	**CTI-Driven Selective Sharing**
Created by:	**UEssex**
Date of creation:	**24.09.2025**
Filename:	**UEssex-CTI.xlsx**

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
CTI-TS01	Verify CTI data exchange in multi-cluster environments	CTI-TS01-TC01	Multi-cluster bidirectional CTI exchange	Two Kubernetes clusters deployed with Trivy scanner and CTI components	1. Deploy apps with varying vulnerability profiles in all clusters 2. Enable bidirectional CTI sharing 3. Monitor data flow and filtering	Mixed applications (WordPress, Jenkins, Redis) vulnerability reports	Each cluster receives tailored CTI based on necessity maps, sensitive data anonymized	01/05/2025	Each cluster receives tailored CTI based on necessity maps, sensitive data anonymized	Pass
CTI-TS02	Validate sensitivity/necessity mapping mechanism	CTI-TS02-TC01	Selective inclusion/anonymisation of vulnerability fields ib the CTI data	Clusters with varying sensitivity and necessity mappings values, vulnerabilities with different risk score values	1. Scan cluster applications for vulnerabilities 2. Process through CTI Agent 3. Inspect and analyse CTI STIX formatted output	Vulnerabilities of deployed applications in the clusters	Each metadata field in a vulnerability data anonymized/included in the STIX bundle using risk score, sensitivity and necessity decision making mechanisms	01/05/2025	Anonymised values are replaced with hash, otherwise the values are included in the final CTI	Pass
CTI-TS03	Evaluate hygiene score calculation	CTI-TS03-TC01	Verify hygiene score reflects cluster risk posture	Multiple clusters with vulnerability	1. Introduce applications with different vulnerability profiles with	Vulnerability datasets with high/med	Clusters with more severe vulnerabilities show	01/05/2025	Clusters with more severe vulnerabi	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
				scanners deployed	different CVSS severities 2. Run CTI agent analysis 3. Calculate hygiene score per cluster	ium/low CVSS scores	lower hygiene scores		lities show lower hygiene scores	

A.11 Multimodal Fusion Approach for Intrusion Detection System for DoS attacks

Project Name:	**NATWORK**
Component Name:	**Multimodal Fusion Approach for Intrusion Detection System for DoS attacks**
Created by:	**CERTH**
Date of creation:	**01.09.2025**
Filename:	**CERTH-Multimodal Fusion Approach IDS.xlsx**

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
Multimodal-IDS-TS01	Verify proper functionality of the component	Multimodal-IDS-TS01-TC01	Deploy 2 docker in the 5G-SDN testbed one for traffic replay and a second one containing the multimodal IDS.	Pre-condition 1: Prior SECaaS processing the instructions are not encrypted.	a)Deploy Dockers b) Check Deployment step (docker service ls)	N/A	Containers Succesfully deployed	Reporting Period 1	Containers Succesfully deployed	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
Multimodal-IDS-TS01	Verify proper functionality of the component	Multimodal-IDS-TS01-TC02	Replay 3 pcap files from open datasets and log the classification results of the IDS i.e. (a) Traffic Type (Anomalous/ Normal), (b) Attack type if anomalous traffic was detected in (a).	Multimodal-IDS-TS01-TC01 succesful	a) Replay pcap files b) Check that traffic is monitored c) Check that logs are correctly produced	UNSW-15,5GAD-2022, 5G-NIDD	Result logs logs are correctly produced	Reporting Period 1	Result logs logs are correctly produced and stored in IDS docker	Pass
Multimodal-IDS-TS01	Verify proper functionality of the component	Multimodal-IDS-TS01-TC03	Compare the logged results with the ground truth contained in the datasets and compare the 3 KPI described in D6.1 i.e. Probability of DoS Attack Detection, AI-based Intrusion Detection, Probability of False detection.	Multimodal-IDS-TS01-TC02 succesful	a) Retrieve logs from docker. b) Run python script to calculate KPIS	UNSW-15,5GAD-2022, 5G-NIDD	Mean Probability of DoS Attack Detection > 80% , Mean Probability of false detection < 10%	Reporting Period 1	Probability of DoS Attack Detection > 0.92 (min) in all cases, Probability of False detection < 0.11 (max) in all cases.	Pass for Probability of DoS Attack Detection, Fail for False detection in UNSW-15

A.12 Lightweight SDN-based AI-enabled Intrusion Detection System for cloud-based services

Project Name:	**NATWORK**
Component Name:	**Lightweight SDN-based AI-enabled Intrusion Detection System for cloud-based services**
Created by:	**CERTH**
Date of creation:	**01.09.2025**
Filename:	**CERTH-SDN IDS.xlsx**

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
SDN-IDS-TS01	Verify proper functionality of the component	SDN-IDS-TS01-TC01	Deploy 3 dockers in the 5G-SDN testbed one for attack creation (Kali Linux tools via python scripts), a second one containing the IDS tool and one for Wireshark to capture traffic.	N/A	a)Deploy Dockers b) Check Deployment step (docker service ls)	N/A	Containers Succesfully deployed	Reporting Period 1	Containers Succesfully deployed	Pass
SDN-IDS-TS01	Verify proper functionality of the component	SDN-IDS-TS01-TC02	Use the Apache JMeter tool for different traffic patterns and workload performance measurements monitor impact on QoS and OpenAirSim to simulate the UEs	SDN-IDS-TS01-TC01	a) Start Jmeter b) Start OpenAirSim c) Check wireshark logs to verify traffic is created	N/A	Traffic succesfully created	Reporting Period 1	Verified that traffic succesfully created	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
			and eNB operation							
SDN-IDS-TS01	Verify proper functionality of the component	SDN-IDS-TS01-TC03	Carry out two types of DoS attacks: (i) a UDP Flooding attack targeting the UPF component; and (ii) an SCTP Flooding attack targeting the AMF component. Log relevant details.	SDN-IDS-TS01-TC02	a) Start attack scripts b)Check wireshark and IDS logs	N/A	Attacks succesfully started	Reporting Period 1	The logs verify that the attacks were successfully started	Pass
SDN-IDS-TS01	Verify proper functionality of the component	SDN-IDS-TS01-TC04	If an attack is detected log identification of the attack, the attacker IP and the message sent to the SDN to mitigate this attack.	SDN-IDS-TS01-TC03	Check wireshark and IDS logs	logs from SDN - IDS-TS01 - TC03	Verify that attacks are detected. Attack time and detection time (minus detection time) should agree.	Reporting Period 1	Verified that attack and detection time match. Both attacks arealways detected when using ensemble of models.	Pass
SDN-IDS-TS01	Verify proper functionality of the component	SDN-IDS-TS01-TC05	Verify that sdn controller has implemented mitigation	SDN-IDS-TS01-TC04	Check IDS logs and SDN rules table	N/A	Commands sent from the IDS should be present in the Table	Reporting Period 1	verified that commands sent from the IDS are present in the Rule Table	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
SDN-IDS-TS01	Verify proper functionality of the component	SDN-IDS-TS01-TC06	Calculate detection times	SDN-IDS-TS01-TC03	Check wireshark and IDS logs to calculate detection time	logs from SDN-IDS-TS01-TC03	No baseline. Detection Time KPI calculated	Reporting Period 1	Both attacks are always detected when using ensemble of models with average time a) 4.8 s when using Exponential Moving Average (EMA), b) 5.2s when using MLP DNN, c) 5.6s when using1D-CNN, d) 5.8 when using ensemble of methods. When the attack was detected, the mitigation action was always successfully implement-ted in the SDN.	Pass

A.13 AI-enabled DoS attack

Project Name:	**NATWORK**
Component Name:	**AI-enabled DoS attack**
Created by:	**CERTH**
Date of creation:	**01.09.2025**
Filename:	**CERTH-AI-enabled_DoS-Attack.xlsx**

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
AI-enabled-DoS-TS01	Attacking SMF 5G component	AI-enabled-DoS-TS01-TC01	Run AI-enabled DoS attack container against SMF component of CERTH's 5G tesbed	N/A	a)Run Docker container specifying the target IP (IP of SMF) and the mode (training mode) b) Check successfull deployment	N/A	Container succesfully deployed	Reporting Period 1	Container succesfully deployed	Pass
AI-enabled-DoS-TS01	Attacking SMF 5G component	AI-enabled-DoS-TS01-TC02	Conduct 1000 episodes in training mode	AI-enabled-DoS-TS01-TC01 succesful Host capability to open at least 210 parallel threads	Wait until 1000 episodes are successfully completed: - Evaluate progression of GORGOs' learning across episodes - Evaluate frequency of successful DoS attacks on the SMF service	N/A - Does not require training and validation data but can adapt its policy based on the execution environment	Exponential decline of epsilon value across episodes Logarithmic/linear growth in rewards after exploration phase completion Consistent growth in the total number of successful attacks across training Total percentage of successful attacks at the end of the training	Reporting Period 1	Actual results follow the expected results Total percentage of successful attacks at the end of the training process: 88.2%	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
							process over 80%			
AI-enabled-DoS-TS02	Attacking AMF 5G component	AI-enabled-DoS-TS02-TC01	Run AI-enabled DoS attack container against AMF component of CERTH's 5G tesbed	The container includes pre-trained weights of the AI model	a)Run Docker container specifying the target IP (IP of AMF) and the mode (training mode) b) Check successfull deployment	N/A	Container succesfully deployed	Reporting Period 1	Container succesfully deployed	Pass
AI-enabled-DoS-TS02	Attacking AMF 5G component	AI-enabled-DoS-TS02-TC02	Conduct 1000 episodes in testing mode	AI-enabled-DoS-TS02-TC01 successful Host capability to open at least 210 parallel threads	Wait until 1000 episodes are successfully completed: - Evaluate progression of GORGOs' pre-trained learning across episodes	N/A	Constant and minimal value of epsilon Constant and maximum value of reward	Reporting Period 1	Actual results follow the expected results: Average epsilon constantly 0.1 Reward constantly 1000	Pass

A.14 Multiagent AI based cybersecurity support system

Project Name:	**NATWORK**
Component Name:	**Multiagent AI based cybersecurity support System**
Created by:	**CERTH**
Date of creation:	**01.09.2025**
Filename:	**CERTH-Multiagent_System.xlsx**

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
MultiAgent-TS01	E2E Module Test Scenario	MultiAgent-TS01-TC01	Deploy the multiagent AI framework in CERTH 5G-SDN testbed	5G-SDN testbed operational	a) Deploy multiagent AI framework. b) Verify all VNFs (UPF, SMF, AMF) are active. c) Ensure data and control plane communication is established.		Component successfully deployed and communicating.	Reporting Period 2 (Future Work)	Deployment successful and system stable.	To be reported in next iteration of the deliverable
MultiAgent-TS01	E2E Module Test Scenario	MultiAgent-TS01-TC02	Inject synthetic attack events (DoS, lateral movement, data exfiltration) and benign traffic.	MultiAgent-TS01-TC01 successful.	a) Use attack simulation scripts to generate malicious traffic. b) Inject benign background traffic. c) Monitor system response.	Synthetic traffic dataset.	System logs correlation activities and detects anomalies.	Reporting Period 2 (Future Work)	Attacks detected, logs correctly generated.	To be reported in next iteration of the deliverable
MultiAgent-TS01	E2E Module Test Scenario	MultiAgent-TS01-TC03	Compare detection and mitigation performance against	MultiAgent-TS01-TC02 successful.	a) Execute both automated and manual response workflows. b) Compare detection time	Performance logs and manual response data.	Automated response faster (<5s) with improved accuracy	Reporting Period 2 (Future Work)	Automated response averaged 4.7s with 10% fewer	To be reported in next iteration of the

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
			manual workflows.		and mitigation success.		and fewer compromised nodes (5–15% reduction).		compromised nodes.	deliverable
MultiAgent-TS02a	Threat Reporting and Insight Agent	MultiAgent-TS02a-TC01	Deploy LLM-based Threat Insight Agent with access to cybersecurity standards, datasets, and context data.	Knowledge base prepared and access permissions granted.	a) Deploy agent. b) Connect to knowledge sources. c) Verify proper initialization.	ISO, ENISA, NIST, and ETSI standard documents.	Agent successfully deployed and connected to knowledge sources.	Reporting Period 1	Agent operational and dataset integration verified.	Pass
MultiAgent-TS02a	Threat Reporting and Insight Agent	MultiAgent-TS02a-TC02	Evaluate prompting strategies (Zero-Shot, One-Shot, Few-Shot).	MultiAgent-TS02a-TC01 successful.	a) Run tests using three prompting modes. b) Collect generated responses.	Golden dataset with Q/A pairs.	Few-Shot prompting outperforms other modes (>10% improvement).	Reporting Period 1	Few-Shot performance exceeded 10% improvement across all metrics.	Pass
MultiAgent-TS02b	Generate Human-Readable Threat Reports and Actionable Insights	MultiAgent-TS02b-TC01	Deploy Threat Intelligence Agent and simulate threats in multiple network zones.	Network zones (Core, Edge, Access) configured in testbed.	a) Deploy agent. b) Inject DDoS, lateral movement, and data exfiltration events. c) Observe agent behavior.	5G-NIDD open Dataset	Agent generates zone-specific summaries and mitigation recommendations.	Reporting Period 1	Reports generated with clear actionable insights per zone.	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
MultiAgent-TS03	IoC Correlation Agent	MultiAgent-TS03-TC01	Train SAFE-AE model on normal traffic samples and validate with mixed data.	Dataset (normal and anomalous traffic) preprocessed and labeled.	a) Train SAFE-AE model. b) Evaluate on unseen traffic bags. c) Feed suspicious outputs into LLM.	5G-NIDD and CERTH datasets.	Model detects anomalous bags accurately and generates detailed IP-level interpretations.	Reporting Period 1	Accuracy 77.75%, Precision 82.06%, Recall 89.58%, F1=85.66%, latency <1s. Detailed and coherent report to mitigate anomalies produced	Pass
MultiAgent-TS04	Coordinate with Security Orchestration Tools	MultiAgent-TS04-TC01	Deploy Orchestration Coordination Agent and validate SOAR integration.	SOAR platform deployed and API keys configured.	a) Deploy agent. b) Connect LLM to SOAR APIs. c) Verify communication.	SOAR configuration files and credentials.	Agent successfully communicates with SOAR platform.	Reporting Period 2 (Future Work)	Connection established, responses received from SOAR.	Pass
MultiAgent-TS04	Coordinate with Security Orchestration Tools	MultiAgent-TS04-TC02	Trigger SOAR-driven actions for vulnerability and policy management.	MultiAgent-TS04-TC01 successful.	a) Simulate vulnerabilities (outdated services, misconfigured ACLs). b) Verify automatic patching, firewall updates, ACL adjustments, and policy modifications. c) Review logs for	Simulated vulnerability dataset.	All actions executed with full traceability and feedback loop to agent network.	Reporting Period 2 (Future Work)	SOAR executed all tasks; feedback incorporated into system logs.	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
					execution traceability.					

A.15 Data plane ML

Project Name:	**NATWORK**
Component Name:	**Data Plane ML**
Created by:	**ELTE**
Date of creation:	**01.09.2025**
Filename:	**ELTE-Data-Plane-ML.xlsx**

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
Data-Plane-ML-TS01	Verify compilation and deployment	Data-Plane-ML-TS01-TC01	Compile ML P4 program for Tofino target	P4 source code of ML component available	1. Run compiler for Tofino target 2. Deploy to Tofino switch	ML-enhanced P4 program	Compilation successful, binary loads on Tofino	Add date of execution	Add actual result	Add status
Data-Plane-ML-TS01	Verify compilation and deployment	Data-Plane-ML-TS01-TC02	Compile ML P4 program for software backend (ebpf)	P4 source code of ML component available	1. Run compiler for ebpf target 2. Deploy to software backend	ML-enhanced P4 program	Compilation successful, binary loads in ebpf	08/2025	Compilation successful, binary loads in ebpf	Pass
Data-Plane-ML-TS02	Verify packet classification	Data-Plane-ML-TS02-TC01	Validate benign traffic classification	Model weights pre-loaded in pipeline	1. Send benign traffic flows 2. Collect classification metadata	Packets with normal traffic features	All packets classified as benign	08/2025	All packets classified as benign	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
Data-Plane-ML-TS02	Verify packet classification	Data-Plane-ML-TS02-TC02	Validate malicious traffic classification	Model weights pre-loaded in pipeline	1. Send portscan / DDoS flow samples 2. Collect classification metadata	Packets with malicious traffic features	Packets correctly tagged as malicious	08/2025	Packets correctly tagged as malicious	Pass
Data-Plane-ML-TS03	Verify control-plane integration	Data-Plane-ML-TS03-TC01	Verify model update from control plane	Control plane interface accessible	1. Push new ML model weights via control plane 2. Verify pipeline reload	Updated ML weights	New weights loaded, classification matches updated model	08/2025	New weights loaded, classification matches updated model	Pass
Data-Plane-ML-TS03	Verify control-plane integration	Data-Plane-ML-TS03-TC02	Verify rules from control plane reflect classification results	ML model deployed, control plane connected	1. Push control rules (e.g. drop on malicious) 2. Send mixed traffic	Benign + malicious traffic	Benign forwarded, malicious dropped	08/2025	Benign forwarded, malicious dropped	Pass
Data-Plane-ML-TS04	Verify robustness of functional pipeline	Data-Plane-ML-TS04-TC01	Handle invalid packet features gracefully	P4 pipeline running	1. Send malformed packets with missing/invalid feature fields	Malformed feature packets	Packets dropped or classified as “unknown”, no crash	08/2025	Packets dropped or classified as “unknown”, no crash	Pass
Data-Plane-ML-TS04	Verify robustness of functional pipeline	Data-Plane-ML-TS04-TC02	Pipeline runs with empty model	Model not preloaded	1. Run pipeline without loading weights 2. Send test traffic	Traffic with no model loaded	Default classification (benign)	08/2025	Default classification (benign)	Pass

A.16 Wire-speed AI (WAI) and Decentralized Feature Extraction (DFE)

Project Name:	**NATWORK**
Component Name:	**DFE-WAI**
Created by:	**CNIT**
Date of creation:	**01.09.2025**
Filename:	**CNIT-DFE-WAI.xlsx**

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
DFE-WAI-TS01	Tofino DNN: Verify compilation, deployment and functionality	DFE-WAI-TS01-TC01	Compile P4 DNN program for Tofino (TNA target)	P4-DNN source code available; Intel P4 Studio/Tofino compiler installed; target environment configured.	1.Run compiler for Tofino target. 2.Deploy to Tofino (TNA target).	P4-DNN program	Compilation successful, binary loads to TNA	07/2025	Compilation successful, binary loads in TNA	Pass
DFE-WAI-TS01	Tofino DNN: Verify compilation, deployment and functionality	DFE-WAI-TS01-TC02	Validate the P4 switch forwards benign and drop malicious	Validation dataset (assumed) pre-loaded in the pipeline.	1.Push control rules (e.g forward malicious to different interface). 2.Send mixed traffic	Benign + malicious traffic	Benign forwarded to one interface and malicious to another interface	07/2025	Benign forwarded to one interface and malicious to another interface	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
DFE-WAI-TS02	DPU DOCA: Verify absence of compilation errors and generate executables	DFE-WAI-TS02-TC01	Compile DOCA application	DOCA 2.9 installed on the deployment DPU; Ubuntu 22.04 installed; GCC and other compilation tools available	1. Upload app source code on DPU; 2. Run compiler on DPU with libraries installed on the system and sources available	DOCA application source	Compilation successful, absence of compilation errors and warnings	07/2025	Compilation successful, absence of compilation errors and warnings	Pass
DFE-WAI-TS02	DPU DOCA: Verify absence of compilation errors and generate executables	DFE-WAI-TS02-TC02	Generate DOCA application inside Docker container	Docker installed on the target DPU; source code available; Docker image nvcr.io/nvidia/doca/doca:2.9.1-devel available; hugepages allocated on the DPU	1. Upload app source code on DPU 2. Prepare Dockerfile to specify App compilation inside docker container 3. Execute Dockerfile on DPU to verify correct compilation	DOCA application source	Compilation successful, absence of compilation errors and warnings	07/2025	Compilation successful, absence of compilation errors and warnings	Pass
DFE-WAI-TS03	DPU DOCA: Test application correct behavior	DFE-WAI-TS03-TC01	Use GDB to inspect app behavior and check correctnesoff	App compiled and available on a test DPU; GDB available on the DPU; DPU connected to a simple traffic generator (very low pps); hugepages allocated on the DPU	1. Run application under GDB 2. Generate test traffic 3. Check if handling of incoming traffic matches expected behavior	DOCA application executable; legitimate traffic; rogue traffic	App control flow matches expected behavior, no anomaly observed	07/2025	App control flow matches expected behavior, no anomaly observed	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
			results under limited load							
DFE-WAI-TS03	DPU DOCA: Test application correct behavior	DFE-WAI-TS03-TC02	Use conditionally compiled code to observe packets match on DOCA Flow pipes	App compiled with extra logging functionalities and available on a test DPU; GDB available on the DPU; DPU connected to a traffic generator; hugepages allocated on the DPU	1. Run application on DPU 2. Run traffic generator 3. Observe DOCA Flow pipe counters increases under the test traffic 4. Check counter values against the expected results	DOCA application executable; legitimate traffic; rogue traffic	DOCA Flow pipe counters increase as expected, proving packet match conditions have been correctly specified	07/2025	DOCA Flow pipe counters increase as expected, proving packet match conditions have been correctly specified	Pass
DFE-WAI-TS03	DPU DOCA: Test application correct behavior	DFE-WAI-TS03-TC03	Stress test	App compiled and available on a test DPU; DPU connected to Cisco T-Rex traffic generator for malicious traffic (TCP SYN DDoS emulator) and a source of legitimate traffic (regular HTTP	1. Start the Apache HTTP server on the DPU's host 2. Start DOCA application 3. Start legitimate traffic and check requests are not blocked 4. Start TCP SYN DDoS attack with T-Rex	DOCA application executable; legitimate traffic; rogue traffic	Legitimate traffic is unaffected by DDoS attack, except for a very short transient timespan in which the attack is	07/2025	Legitimate traffic is unaffected by DDoS attack, except for a very short transient timespan in which the attack is	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
				requests); Apache HTTP server running on the DPU's host; hugepages allocated on the DPU	5. Check TCP SYN DDoS traffic is quickly blocked by the DPU app 6. Check legitimate traffic latency increment is negligible while DDoS traffic is active but blocked by the DPU app		the attack is detected and blocked; all the DDoS sources are quickly blocked by the DOCA app		detected and blocked; all the DDoS sources are quickly blocked by the DOCA app	
DFE-WAI-TS03	DPU DOCA: Test application correct behavior	DFE-WAI-TS03-TC04	DOCA application tested within Docker container	Docker image previously generated containing DOCA app available; Docker installed on the DPU; Apache HTTP server installed on the DPU's host; traffic generators available; hugepages allocated on the DPU	1. Start the Apache HTTP server on the DPU's host 2. Start DOCA application within Docker container 3. Generate legitimate and rogue traffic 4. Check performance outcomes do not differ from what is observed when Docker is not involved	Docker image containing DOCA application executable; legitimate traffic; rogue traffic	Observed performance is not lower than what is observed when Docker is not used	07/2025	Observed performance is not lower than what is observed when Docker is not used	Pass

A.17 Microservice behavioral analysis for detecting malicious actions

Project Name:	**NATWORK**
Component Name:	**Microservice Behavioral Analysis for Detecting Malicious Action Component**
Created by:	**CERTH**
Date of creation:	**09/10/2025**
Filename:	**CERTH-Microservice Behavioral Analysis for Detecting Malicious Action Component.xlsx**

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
MBADA-TS01	Verify functionality of the profiling and malicious detection component	MBADA-TS01-TC01	Deploy a dockerized profiling tool to monitor twelve key metrics across infrastructure, including CPU and memory usage, disk read/write throughput, network traffic, latency percentiles, and error rates, establishing a baseline of normal microservice behav	N/A	a) Deploy the profiling tool container b) Verify service status c) Confirm metric collection from all nodes	N/A	Profiling tool successfully deployed	Reporting Period 1	Profiling tool successfully deployed	Pass
MBADA-TS01	Verify functionality of the profiling and malicious detection component	MBADA-TS01-TC02	Gather real-time resource usage and performance data from all deployed microservices. Aggregate metrics to detect both gradual deviations (e.g., step increases in load) and sudden	MBADA-TS01-TC01	a) Start real-time monitoring b) Generate workload on microservices c) Check metric aggregator and logs for deviations	Real-time metric stream	Data succesfully gathered	Reporting Period 1	Data successfully collected	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
			anomalies (e.g., spikes in traffic or CPU/memory usage).							
MBADA-TS01	Verify functionality of the profiling and malicious detection component	MBADA-TS01-TC03	Utilize a lightweight 1-D CNN to classify microservice behavior as Normal or Anomalous. Repeat with MLP, Random Forest, and SVM for validation.	MBADA-TS01-TC02	a) Train and deploy CNN classifier b) Execute same dataset with MLP, RF, and SVM c) Compare detection outputs	Collected performance metrics dataset	Correct data classification	Reporting Period 1	Accurate classification of system behavior	Pass
MBADA-TS01	Verify functionality of the profiling and malicious detection component	MBADA-TS01-TC04	For microservices flagged as anomalous, use 1-D CNN to identify anomaly type (CPU, memory, traffic spike, load increase, latency) or mark as Unknown. Repeat with other AI/ML models to compare performance.	MBADA-TS01-TC03	a) Run anomaly classification model b) Record outputs for each anomaly type c) Compare CNN results with other models	Anomaly-labeled dataset	Proper identification of anomalies	Reporting Period 1	Anomalies correctly identified	Pass
MBADA-TS01	Verify functionality of the profiling and malicious detection	MBADA-TS01-TC05	Perform a proof-of-concept evaluation using an open dataset to assess binary and multiclass detection	MBADA-TS01-TC04	a) Load open dataset b) Evaluate binary and multiclass models c) Record F1,	Open dataset	Correct evaluation of the dataset	Reporting Period 1	Datasets accurately evaluated	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
	component		robustness and accuracy.		precision, recall metrics					
MBADA-TS01	Verify functionality of the profiling and malicious detection component	MBADA-TS01-TC06	Utilize an RNN-based model to predict CPU and memory consumption using the open dataset under normal and attack conditions.	MBADA-TS01-TC02	a) Train RNN model on resource usage data b) Evaluate predictions against real measurements c) Analyze prediction accuracy	Open dataset	Accurately predict CPU and memory consumption	Reporting Period 1	CPU and memory consumption accurately predicted	Pass
MBADA-TS02	Verify proper functionality of the 5G Microservice behavioral analysis Environment	MBADA-TS02-TC01	Deploy the Microservice Orchestrator. Set up Kubernetes cluster to function as the microservice orchestrator, responsible for automating deployment, scaling, and management of containerized microservices.	N/A	a) Deploy Kubernetes cluster b) Configure orchestration services c) Verify all nodes are active	N/A	Orchestrator successfully deployed	Reporting Period 2	Orchestrator deployed and operational	-

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
MBADA-TS02	Verify proper functionality of the 5G Microservice behavioral analysis Environment	MBADA-TS02-TC02	Deploy the 5G Core Network. Implement the 5G core network with containerized functions such as AMF, SMF, and UPF for handling control and user plane operations.	MBADA-TS02-TC01	a) Deploy Free5GC containers b) Validate service startup c) Check connectivity among core components	Free5GC container logs	5G core network running properly	Reporting Period 2	5G core network fully operational	-
MBADA-TS02	Verify proper functionality of the 5G Microservice behavioral analysis Environment	MBADA-TS02-TC03	Integrate the Central SDN Controller. Deploy and configure the controller to enable centralized network control, efficient traffic management, and optimized resource allocation across the 5G core components.	MBADA-TS02-TC02	a) Deploy SDN controller b) Configure OpenFlow rules c) Verify communication with 5G core components	Controller configuration files	SDN controller integrated correctly	Reporting Period 2	SDN controller integrated correctly	-
MBADA-TS02	Verify proper functionality of the 5G Microservice behavioral analysis Environment	MBADA-TS02-TC04	Set up the Monitoring Engine to continuously collect resource metrics from all deployed microservices. This includes CPU utilization, memory usage, disk throughput, and other KPIs, providing real-time data required for the	MBADA-TS02-TC03	a) Deploy Monitoring Engine b) Configure data collectors c) Validate metric collection from all nodes	Collected metrics dataset	Monitoring Engine captures metrics	Reporting Period 2	Monitoring Engine captures metrics	-

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
			Behavioral Analysis module.							
MBADA-TS02	Verify proper functionality of the 5G Microservice behavioral analysis Environment	MBADA-TS02-TC05	Activate the Microservice Behavioral Analysis Module to profile microservices and detect deviations using AI-driven anomaly detection. Detected anomalies trigger automated orchestration actions.	MBADA-TS02-TC04	a) Activate analysis module b) Run behavior profiling c) Validate anomaly detection outputs	Metric data from Monitoring Engine	Behavioral Analysis identifies anomalies	Reporting Period 2	Behavioral Analysis identifies anomalies	-
MBADA-TS02	Verify proper functionality of the 5G Microservice behavioral analysis Environment	MBADA-TS02-TC06	Perform controlled attack simulations on the deployed 5G microservice infrastructure to evaluate detection and mitigation mechanisms (e.g., DoS, privilege escalation, unauthorized access).	MBADA-TS02-TC05	a) Launch simulated attack scenarios b) Monitor behavioral responses c) Record detection and mitigation times	Attack simulation scripts	System detects and handles attacks	Reporting Period 2	System detects and handles attacks	-
MBADA-TS02	Verify proper functionality of the 5G Microservi	MBADA-TS02-TC07	Detect potential attacks through Behavioral Anomaly Analysis using a two-stage CNN model to classify normal and	MBADA-TS02-TC06	a) Run Behavioral Anomaly Analysis b) Collect classification	Telemetry and resource cons	CNN correctly classifies attacks	Reporting Period 2	CNN correctly classifies attacks	-

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
	ce behavioral analysis Environment		abnormal microservice behaviors.		results c) Evaluate accuracy of CNN model	umption data				
MBADA-TS02	Verify proper functionality of the 5G Microservice behavioral analysis Environment	MBADA-TS02-TC08	Execute mitigation actions based on detected anomalies: initiate automated mitigation through the orchestrator and SDN controller.	MBADA-TS02-TC07	a) Trigger mitigation process b) Verify orchestration and SDN responses c) Confirm service stability post-mitigation	Detected anomaly logs	Mitigation applied; services stable	Reporting Period 2	Mitigation applied; services stable	-

A.18 MTD Controller

Project Name:	**NATWORK**
Component Name:	**MTD Controller**
Created by:	**ZHAW**
Date of creation:	**27.08.2025**
Filename:	**ZHAW-MTD-Controller.xlsx**

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
MTD Controller-TS.01	Verify functionality: Live CNF Migration	MTD Controller-TS.01-TC.01	Testing the end-to-end live migratio	* The MTD Strategy Optimizer should be up and running	Step 1: Initiate MTD Framework Step 2-A: Wait for a sufficient amount of	N/A	The MTD Controller should communicate with the Container Orchestrator (e.g.,	Reporting Period 1	Successful comunication with Kubernetes	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
			n process of a CNF	* At least an existing and **STATEFUL** CNF is already deployed in the edge-to-cloud continuum * The MTD Controller should be operating in a cluster with at least two nodes	time until the MTD Strategy Optimizer proactively decides to perform the migration Step 2-B: Alternatively, trigger a cyberattack (e.g., data exfiltration) to force the MTD Strategy Optimizer to reactively decide to perform a migration Step 3: Observe closely the CNF status and on which node/device it is operating		Kubernetes) to perform the live migration. Afterwards, the CNF should be migrated to a new node and still be running, while preserving the past session		and CNF live migration.	
MTD Controller-TS.02	Verify functionality: Stateless VNF Migration	MTD Controller-TS.02-TC.01	Testing the end-to-end stateless migration process of a VNF	* The MTD Strategy Optimizer should be up and running * At least an existing and **STATELESS** VNF is already deployed in the edge-to-cloud continuum	Step 1: Initiate MTD Framework Step 2-A: Wait for a sufficient amount of time until the MTD Strategy Optimizer proactively decides to perform the migration Step 2-B: Alternatively, trigger a cyberattack (e.g., malware infection) to force the MTD Strategy Optimizer to	N/A	The MTD Controller should communicate with the NFV MANO (e.g., OSM) to stop the execution of the VNF on the old node. Then, the same VNF should be instantiated from scratch in another node. In the end, the VNF should be running on the new node,	Reporting Period 1	Successful comunication with OSM and stateless VNF live migration.	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
				* The MTD Controller should be operating in a cluster with at least two nodes	reactively decide to perform a migration Step 3: Observe closely the VNF status and on which node/device it is operating		not on the old node.			
MTD Controller-TS.02	Verify functionality: Stateless CNF Migration	MTD Controller-TS.02-TC.02	Testing the end-to-end stateless migration process of a CNF	* The MTD Strategy Optimizer should be up and running * At least an existing and **STATELESS** CNF is already deployed in the edge-to-cloud continuum * The MTD Controller should be operating in a cluster with at least two nodes	Step 1: Initiate MTD Framework Step 2-A: Wait for a sufficient amount of time until the MTD Strategy Optimizer proactively decides to perform the migration Step 2-B: Alternatively, trigger a cyberattack (e.g., malware infection) to force the MTD Strategy Optimizer to reactively decide to perform a migration Step 3: Observe closely the CNF status and on which node/device it is operating	N/A	The MTD Controller should communicate with the Container Orchestrator (e.g., Kubernetes) to start a CNF replica from an authenticated image in a node, then stop the execution of a previous replica on the old node.	Early 2026	Not available yet	Not evaluated yet

A.19 MTD Strategy Optimizer

Project Name:	**NATWORK**
Component Name:	**MTD Strategy Optimizer**
Created by:	**ZHAW**
Date of creation:	**26.08.2025**
Filename:	**ZHAW-MTD-Strategy-Optimizer.xlsx**

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
MTD Strategy Optimizer-TS.01	Verify functionality: Pro-Active Decisions	MTD Strategy Optimizer-TS.01-TC.01	Testing the stateless migration of a VNF as a proactive measurement	* A monitoring tool is integrated into the system, which would feed the MTD Strategy Optimizer (e.g., MONT MMT tool) * At least an existing VNF is already deployed in the edge-to-cloud continuum	Step 1: Initiate MTD Framework Step 2: Monitor the network environment via the monitoring tool Step 3: Wait for some time so the age of deployed VNF is older	Real-time network data from Step 2	Eventually, the MTD Strategy Optimizer should make a decision to migrate the VNF to a new location	Reporting Period 1	-	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
MTD Strategy Optimizer-TS.01	Verify functionality: Pro-Active Decisions	MTD Strategy Optimizer-TS.01-TC.02	Testing the live migration of a CNF as a proactive measurement	* A monitoring tool is integrated into the system, which would feed the MTD Strategy Optimizer (e.g., MONT MMT tool) * At least an existing CNF is already deployed in the edge-to-cloud continuum	Step 1: Initiate MTD Framework Step 2: Monitor the network environment via the monitoring tool Step 3: Wait for some time so the age of deployed CNF is older	Real-time network data from Step 2	Eventually, the MTD Strategy Optimizer should make a decision to migrate the CNF to a new location	Reporting Period 1	-	Pass
MTD Strategy Optimizer-TS.02	Verify functionality: Reactive Decisions	MTD Strategy Optimizer-TS.02-TC.01	Testing the stateless migration of a VNF as a reactive measurement upon a cyberattack	* A monitoring tool is integrated into the system, which would feed the MTD Strategy Optimizer (e.g., MONT MMT tool) * At least an existing VNF is already deployed in the edge-to-cloud continuum	Step 1: Initiate MTD Framework Step 2: Monitor the network environment via the monitoring tool Step 3: Trigger a malware infection attack on the existing VNF Step 4: Observe how the MTD	Real-time network data from Step 2	Upon the detection of the attack, the MTD Strategy Optimizer should immediately make a decision to migrate the VNF to a new location	Reporting Period 1	this test case is evaluated following the detection of an intrusion and/or tampering attack	Pass Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
					framework reacts					
MTD Strategy Optimizer-TS.02	Verify functionality: Reactive Decisions	MTD Strategy Optimizer-TS.02-TC.02	Testing the live migration of a CNF as a reactive measurement upon a cyberattack	* A monitoring tool is integrated into the system, which would feed the MTD Strategy Optimizer (e.g., MONT MMT tool) * At least an existing CNF is already deployed in the edge-to-cloud continuum	Step 1: Initiate MTD Framework Step 2: Monitor the network environment via the monitoring tool Step 3: Trigger a data exfiltration or intrusion attack on the existing CNF Step 4: Observe how the MTD framework reacts	Real-time network data from Step 2	Upon the detection of the attack, the MTD Strategy Optimizer should immediately make a decision to migrate the CNF to a new location	Reporting Period 1	this test case is evaluated following the detection of an intrusion and/or tampering attack	Pass

A.20 MTD Explainer

Project Name:	**NATWORK**
Component Name:	**MTD Explainer**
Created by:	**ZHAW**
Date of creation:	**27.08.2025**
Filename:	**ZHAW-MTD-Explainer.xlsx**

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
MTD Explainer-TS.01	Verify functionality: Human explanation of an action decided by the MTD Strategy Optimizer	MTD Explainer-TS.01-TC.01	Testing the explainability of a decision made by the MTD Strategy Optimizer	* The MTD Strategy Optimizer should be up and running * At least an existing CNF/VNF is already deployed in the edge-to-cloud continuum * The MTD Framework should be operating in a cluster with at least two nodes	Step 1: Initiate MTD Framework Step 2-A: Wait for a sufficient amount of time until the MTD Strategy Optimizer proactively decides to perform the migration Step 2-B: Alternatively, trigger a cyberattack (e.g., data exfiltration) to force the MTD Strategy Optimizer to reactively decide to perform a migration Step 3: Retrieve the output of the MTD Explainer with regard to the operation	N/A	The MTD Explainer should provide a humanly interpretable explanation on why the MTD Strategy Optimizer decided the specific action (e.g., stateless or stateful migration) and how this action helps the system in terms of security.	Mid 2026	Not available yet	Not evaluated yet

A.21 AI-driven security monitoring for anomaly detection and root cause analysis in IoT networks

Project Name:	**NATWORK**
Component Name:	**AI-driven security monitoring for anomaly detection and root cause analysis in IoT networks**
Created by:	**MONT**
Date of creation:	**27.08.2025**
Filename:	**MONT-AI-AD-RCA.xlsx**

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
IoT-Sec-TS1	Establish baseline IoT traffic monitoring in normal operating conditions.	IoT-Sec-TS1-TC1.1	Monitor and collect traffic from IoT devices under normal operation.	IoT testbed operational, MMT probe deployed, MAIP model not trained yet.	1- Deploy IoT devices in testbed. 2- Generate normal traffic (sensor data, control messages). 3- Capture traffic via MMT probe.	Normal IoT network traffic logs.	Clean dataset with no anomalies; system records traffic correctly.	9/25/2025	Clean dataset with no anomalies; system records traffic correctly.	Pass
IoT-Sec-TS2	Detect different types of DDoS attacks on IoT devices.	IoT-Sec-TS2-TC2.1	SYN flood detection.	ML model trained on normal and SYN flood traffic.	1- Launch SYN flood against IoT gateway. 2- Capture traffic with MMT probe. 3- Run ML anomaly detection.	Attack traffic + benign traffic mix.	SYN flood detected within <5 minutes (ML rule) or <10ms (MMT rule).	9/25/2025	3.5 minutes for ML-based rules and 89 ms for non ML-based rules	Partially pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
		IoT-Sec-TS2-TC2.2	UDP flood detection	Attack traffic + benign traffic mix.	1- Launch UDP flood against IoT gateway. 2- Capture traffic with MMT probe. 3- Run ML anomaly detection.	Attack traffic + benign traffic mix.	UDP flood detected within <5 minutes (ML rule) or <10ms (MMT rule).	9/25/2025	3.5 minutes for ML-based rules and 3 ms for non ML-based rules	Pass
		IoT-Sec-TS2-TC2.3	ICMP flood detection	Attack traffic + benign traffic mix.	1- Launch ICMP flood against IoT gateway. 2- Capture traffic with MMT probe. 3- Run ML anomaly detection.	Attack traffic + benign traffic mix.	ICMP flood detected within <5 minutes (ML rule) or <10ms (MMT rule).	9/25/2025	3.5 minutes for ML-based rules and 5 ms for non ML-based rules	Pass
IoT-Sec-TS3	Validate detection accuracy. False Positive / False Negative Evaluation	IoT-Sec-TS3-TC3.1	Evaluate FP rate <1%.	Trained model, clean dataset.	Run IDS on large clean dataset.	100% benign traffic.	<1% alerts triggered.	9/25/2025	0%	Pass
		IoT-Sec-TS3-TC3.2	Evaluate FN rate <1%.	Model trained with labeled attacks.	Inject multiple attack samples.	Mix of attacks (SYN, UDP, ICMP floods).	FN <1%.	9/25/2025	1.50%	Fail

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
IoT-Sec-TS4	Adaptive anomaly threshold setting using reinforcement learning.	IoT-Sec-TS4-TC4.1	System adjusts threshold during traffic spike without misclassification.	RL thresholding enabled.	1- Generate sudden benign traffic spike. 2- Monitor threshold adaptation. 3- Verify alerts.	Traffic burst without attack.	No false alert triggered; threshold auto-adjusted.	9/25/2025	0%	Pass
IoT-Sec-TS5	Validate system's mitigation response once attack detected.	IoT-Sec-TS5-TC5.1	Automatic traffic filtering after DDoS detection.	Detection system active, mitigation module connected.	1- Launch SYN flood. 2- Wait for detection alert. 3- Observe mitigation (traffic dropped, route blocked).	Attack traffic.	Mitigation triggered <10 minutes; attack neutralized.	9/25/2025	~ 18 minutes	Fail
IoT-Sec-TS6	Validate AI-driven RCA in identifying the underlying cause of anomalies.	IoT-Sec-TS6-TC6.1	Detect anomaly due to a misconfigured IoT device (not an attack).	Device configured with incorrect routing rule.	1- Deploy IoT device with misconfigured route/firewall. 2- Generate normal traffic. 3- System detects anomaly. 4- RCA module analyses anomaly.	Traffic deviating due to misconfiguration.	Anomaly detected, RCA output = "Device misconfiguration, not malicious".	9/25/2025	Anomaly detected, RCA output = "Device misconfiguration, not malicious".	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
		IoT-Sec-TS6-TC6.2	Identify root cause of detected anomaly as DDoS.	SYN flood attack launched on IoT gateway.	1- Launch SYN flood. 2- Anomaly detected. 3- RCA module correlates alerts (e.g., multiple flows, repeated requests).	Attack traffic.	RCA output = "SYN Flood attack detected at gateway".	9/25/2025	RCA output = "SYN Flood attack detected at gateway".	Pass
		IoT-Sec-TS6-TC6.3	Identify a compromised IoT device as the anomaly source.	Malware-infected IoT device generating abnormal traffic.	1- Infect IoT device with simulated malware (e.g., Mirai sample). 2- Device sends abnormal traffic. 3- Anomaly detected. 4- RCA correlates anomaly to specific device ID.	Device-originating attack traffic.	RCA output = "Compromised IoT device X, abnormal outbound traffic".	9/25/2025	RCA output = "Compromised IoT device X, abnormal outbound traffic".	Pass
		IoT-Sec-TS6-TC6.4	Provide explainable RCA report for detected anomaly.	Any anomaly detected (e.g., DDoS).	1- Run detection + RCA. 2- LLM/XAI generates natural language explanation.	Anomaly logs.	Operator receives clear report (e.g., “Traffic spike caused by SYN flood attack from IP X”).	9/25/2025	Operator receives clear report (e.g., “Traffic spike caused by SYN flood attack from IP X”).	Pass

A.22 DFE Telemetry

Project Name:	**NATWORK**
Component Name:	**DFE-Telemetry**
Created by:	**CNIT**
Date of creation:	**01.09.2025**
Filename:	**CNIT-DFE-Telemetry.xlsx**

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
DFE-Telemetry-TS01	Verify compilation and deployment	DFE-Telemetry-TS01-TC01	Compile P4-DFE Telemetry program for software switch (BMV2 target).	P4-DFE Telemetry source code available, BMv2 installed, and p4c compiler configured.	1.Run P4C compiler. 2.Deploy to BMV2 target.	P4-DFE Telemetry program.	Compilation successful, binary loads to BMV2.	12/2024	Compilation successful, binary loads to BMV2.	Pass
DFE-Telemetry-TS02	Validate functional behavior	DFE-Telemetry-TS02-TC01	Run P4-DFET program in Mininet with three hosts and three collectors.	Mininet topology with three hosts and three collectors, flow rules to extract different features from different flows.	1. Start Mininet . 2. Push flow rules. 3.Generate Two UDP flows and one TCP flow	Two UDP flows and one TCP flow	- Reports generated for 3 flows with different sizes. - Reports delivered to correct collectors (C1, C2, C3).	12/2024	- Three reports generated for 3 flows with different sizes. - Reports delivered to correct collectors (C1, C2, C3).	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
DFE-Telemetry-TS03	Measure latency and CPU overhead	DFET-Telemetry-TS03-TC01	Compare P4-DFET vs. simple forwarding (latency).	Spirent N4U traffic generator connected-Single 10 Mbps flow configured.	1. Run baseline simple forwarding. 2. Run P4-DFET with 1 feature 3. Run P4-DFET with all features. 4. Measure latency.	Spirent-generated 10 Mbps flow	Latency overhead remains within expected range.	3/2025	- Simple forwarding: 510.84 µs - DFET (1 feature): 697.79 µs - DFET (all features): 733.147 µs	Pass
DFE-Telemetry-TS03	Measure latency and CPU overhead	Data-Telemetry-TS04-TC02	Compare P4-DFET vs. simple forwarding (CPU load).	Spirent N4U traffic generator connected-Single 10 Mbps flow configured.	1. Run baseline simple forwarding. 2. Run P4-DFET with 1 feature 3. Run P4-DFET with all features. 4. Monitor CPU load.	Spirent-generated 10 Mbps flow	CPU load values remains within expected range.	3/2025	- Simple forwarding: 26% - DFET (1 feature): 63% - DFET (all features): 93.3%	Pass
DFE-Telemetry-TS04	Evaluate scalability (for both latency and CPU load)	Data-Telemetry-TS04-TC01	P4-DFET with varying number of flows (1, 10, 100, 1000).	Spirent configured for multiple flows-Total load fixed at 10 Mbps.	1. Configure Spirent for 1, 10, 100, 1000 flows. 2. Enable P4-DFET with single/all features. 3. Measure latency and monitor CPU load.	Spirent-generated traffic (1–10-100-1000) flows at 10Mbps.	Latency and CPU load values increase moderately with increasing the number of flows.	3/2025	Latency and CPU load values increase moderately with increasing the number of flows.	Pass

A.23 Secure Data Aggregation

Project Name:	**NATWORK**
Component Name:	**Secure Data Aggregation**
Created by:	**ELTE**
Date of creation:	**01.09.2025**
Filename:	**ELTE-Data-Aggregation.xlsx**

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
Secure_ DA-TS.01	Basic Flwr Server Startup	Secure_ DA-TS.01-TC.01	Add test case description	Python environment with flwr installed. Dataset and client simulation scripts prepared.	Step 1: Start the Flwr server. Step 2: Launch multiple simulated Flwr clients. Step 3: Monitor logs for connection establishment.	MNIST/ HAR datasets	Server starts successfully and listens for clients. Clients connect to server and begin training rounds.	Aug-24	Flwr server initialized. Logs confirmed client registration and start of training session.	Pass
Secure_ DA-TS.01	Basic Flwr Server Startup	Secure_ DA-TS.01-TC.02	Federated Training Run	Python environment with flwr installed. Dataset and client simulation scripts prepared.	Step 1: Run training for N rounds. Step 2: Monitor client participation and aggregation logs. Step 3: Collect model accuracy after each round.	Partitioned dataset across clients.	Training completes successfully for all rounds. Aggregated global model improves over time.	Aug-24	Training completed successfully.	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
Secure_ DA-TS.02	SecAgg+ for Secure Aggregation	Secure_ DA-TS.02-TC.01	Secure Aggregation Setup	Flwr framework running. SecAgg+ integration module enabled in server and clients.	Sdtep 1: Start Flwr server with SecAgg+ enabled. Step 2: Launch multiple clients with SecAgg+ enabled. Step 3: Verify key exchange and masking steps in logs.	Small dataset for testing (dummy vectors).	SecAgg+ successfully initialized between server and clients. Clients exchange encrypted masks.	Aug-24	SecAgg+ initialization completed.	Pass
Secure_ DA-TS.02	SecAgg+ for Secure Aggregation	Secure_ DA-TS.02-TC.02	Secure Federated Training Run	Flwr framework running. SecAgg+ integration module enabled in server and clients.	Step 1: Run federated training with N clients. Step 2: Each client contributes encrypted model updates. Step 3: Server aggregates masked updates. Step 4: Verify that only aggregated result is revealed, not individual updates.	Partitioned dataset across clients.	Aggregation succeeds even with masked updates. Individual client updates remain private. Global model converges similarly to non-secure baseline.	Aug-24	Aggregation succeeded. Individual client updates remained private.	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
Secure_ DA-TS.03	MPC-Based Secure Aggregation	Secure_ DA-TS.03-TC.01	MPC Initialization	Flwr server and clients running. MPC library MP-SPDZ integrated. SecAgg+ enabled as base protocol.	Step 1: Initialize MPC session among clients and server. Step 2: Verify secure multi-party key generation. Step 3: Ensure each client is assigned a secret share of the aggregation protocol.	Partitioned dataset across clients.	MPC session established successfully. All clients and server confirm participation in secure computation.	Aug-24	MPC session established successfully	Pass
Secure_ DA-TS.03	MPC-Based Secure Aggregation	Secure_ DA-TS.03-TC.02	End-to-End MPC Aggregation	Flwr server and clients running. MPC library MP-SPDZ integrated. SecAgg+ enabled as base protocol.	Step 1: Clients compute local model updates. Step 2: Updates are secret-shared via MPC. Step 3: Server executes MPC aggregation. Step 4: Aggregated result revealed without exposing individual updates. Step 5: Verify logs and performance (latency, resource usage).	Partitioned dataset across clients.	Aggregated model update computed securely via MPC. No single party has access to individual client data. Performance overhead acceptable compared to SecAgg+.	Aug-24	Aggregated model update computed securely via MPC. No single party had access to individual client data.	Pass

A.24 Federated Learning for edge-to-cloud

Project Name:	**NATWORK**
Component Name:	**AI for optimized scheduling (edge-cloud)**
Created by:	**UEssex**
Date of creation:	**01.09.2025**
Filename:	**UEssex-Federated Learning edge-cloud.xlsx**

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
Benchmarking ML algos to predict workloads in cloud - TS01	Benchmark centralized ML models for workload prediction	Benchmarking ML algos to predict workloads in cloud - TS01-TCO1	Train and evaluate ML models (ARIMAX, LSTM, XGBoost) on historical Google cluster traces as baseline and further improve with feature engineering & hyperparameter tuning	Pre-processing and feature engineering of Google cluster traces	1. Extract and clean CPU/memory features from traces 2. Prepared datasets for model training through two methods: fine-level granularity for detailed patterns, and orchestration-focused aggregation for peak demand planning.3. Apply feature engineering (e.g., lag features, rolling statistics) 4. Train ARIMA, LSTM, XGBoost models with	Google cluster trace dataset (historical)	Trained ML models should provide accurate workload predictions, with expected accuracy in a reasonable range across algorithms	16/08/2025	Predictive framework validated; best accuracy (able to predict spikes in data) achieved ~70–75% (XGBoost); demonstrates robustness for orchestration	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
					hyperparameter tuning 5. Evaluate against actual workloads					
Baseline federated learning Framework - TS02	Implement baseline federated learning for historical workload prediction using Google workload traces	Baseline federated learning Framework - TS02-TC01	Evaluate FL architecture (bagging with XGBoost) on distributed Google trace partitions, and testing with hyperparameter tuning	Pre-processing and partitioning of Google traces across multiple nodes	1. Partition traces across nodes 2. Train local XGBoost models on each node 3. Aggregate centrally (currently) via FL bagging approach 4. Compare FL predictions vs. ground truth workloads on test set	Partitioned Google cluster traces	FL setup should demonstrate feasibility of decentralized training, with prediction accuracy lower than centralized ML but within an acceptable range	15/09/2025	FL baseline validated; decentralized training feasible with comparable accuracy to centralized ML	Initial testing done using Google Cluster traces(Pass) Not yet tested on custom Dost Data

A.25 MTDFed

Project Name:	**NATWORK**
Component Name:	**MTDFed**
Created by:	**ZHAW**
Date of creation:	**27.08.2025**
Filename:	**ZHAW-MTDFed.xlsx**

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
MTDFed-TS.01	Verify functionality: Aggregation across Virtual Network Operators (VNOs)	MTDFed-TS.01-TC.01	Testing basic MTDFed without any privacy-preservation mechanism	* The MTD Framework should be up and running, with at least three VNOs across edge nodes * At least a set of existing VNFs and CNFs is already deployed in the edge-to-cloud continuum * The aggregator should be up and running in the core network	Step 1: Initiate MTD Framework Step 2: Trigger MTDFed so that VNOs collaboratively train the MTD Strategy Optimizer Step 3: Wait until several round of federated learning occurs Step 4: Observe closely the aggregation process until convergence Step 5: Investigate how the performance of individual MTD Strategy Optimizer models changes over time	N/A	The MTDFed component should yield a more accurate MTD Strategy Optimizer via federated learning over multiple VNOs. The performance of the global model should be investigated across rounds for gaining further insight.	Reporting Period 1	MTDFed tested and preliminary results show improvements of the MTD strategy over single-agent training	Pass
MTDFed-TS.01	Verify functionality: Aggregation across Virtual Network	MTDFed-TS.01-TC.02	Testing secure MTDFed via MPC	* The MTD Framework should be up and running, with at least three	Step 1: Initiate MTD Framework Step 2: Trigger MTDFed so that VNOs collaboratively	N/A	The MTDFed component should yield a more accurate MTD Strategy Optimizer via federated	Reporting Period 1	MTDFed tested and local model confidentiality enabled	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
	Operators (VNOs)			VNOs across edge nodes * At least a set of existing VNFs and CNFs is already deployed in the edge-to-cloud continuum * The aggregator should be up and running in the core network * For secure aggregation, MPC should be deployed and configured in the system	train the MTD Strategy Optimizer Step 3: Wait until several round of federated learning occurs Step 4: Observe closely the aggregation process until convergence Step 5: Investigate how the performance of individual MTD Strategy Optimizer models changes over time		learning over multiple VNOs. In addition to the previous test, the aggregator should be debugged to ensure that it is not aware of the individual models from VNOs.			
MTDFed-TS.01	Verify functionality:	MTDFed-	Testing secure	* The MTD Framework should be	Step 1: Initiate MTD Framework	N/A	The MTDFed component should yield a	Reporting Period 1	MTDFed tested and differential	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
	Aggregation across Virtual Network Operators (VNOs)	TS.01-TC.03	MTDFed via DP	up and running, with at least three VNOs across edge nodes * At least a set of existing VNFs and CNFs is already deployed in the edge-to-cloud continuum * The aggregator should be up and running in the core network	Step 2: Trigger MTDFed so that VNOs collaboratively train the MTD Strategy Optimizer Step 3: Wait until several round of federated learning occurs Step 4: Observe closely the aggregation process until convergence Step 5: Investigate how the performance of individual MTD Strategy Optimizer models changes over time		more accurate MTD Strategy Optimizer via federated learning over multiple VNOs. In addition to the previous test, the performance of the global model will likely be lower to some extent due to the use of differential privacy.		privacy overhead measured.	

A.26 CIA-hardening of x86 payloads Component

Project Name:	**NATWORK**
Component Name:	**SECaaS**
Created by:	**TSS**
Date of creation:	**10.09.2025**
Filename:	**TSS-CIA hardening x86 payloads.xlsx**

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
SECaaS_ConFx86-TS.01	Verify Confidentialtiy resilience x86	SECaaS_ConFx86-TS.01-01	Check that workload file shows encrypted instruction text section	Pre-condition 1: Prior SECaaS processing the instructions are not encrypted.	Step 1: Check code section prior SECaaS operation Step 2: Check code section after SECaaS operation	Two standard x86 native ELF formated workloads	Visual Inspection	01/12/2024	x86 text section is encrypted with AES256, hence cannot be reversed and analyzed through static analysis	Pass
SECaaS_ConFx86-TS.02	Verify Confidentialtiy preservation technique impact on latency at start	SECaaS_ConFx86-TS.02-01	Check that the decryption of the instructions (prior their are loaded and executed) is done in a given time slot	Pre-condition 1: Timestamped code available, enabling to assess with precision the latency at start	Step 1: Timestamped original code is used to assess the time to reach the second timestamp Step 2: Timestamped protected code latency at start is measured	Two standard x86 native ELF formated workload	Time to decrypt and start the executable is below KPI 1.3.2 threshold (ie, 3 seconds)	01/12/2024	Measurements show that the difference between unprotected code and protected code latency at start is average at 120 msec in average (90-150msec). This is sufficiently below KPI 1.3.2 to consider that this KPI is met	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
SECaaS_ConFx86-TS.03	Verify Confidentiality preservation impact on performance	SECaaS_ConF-TS.03.01	Check the impact on protected code performance	Timestamping the workload to assess the performance penalty	Step 1: Timestamps placed on the original code to assess the performance penalty Step 2: Timestamps placed at the same locations in the protected code	Two standard x86 native ELF formated workload	Runtime performance degradation (% compared to baseline execution speed)	01/12/2024	After the decryption step (which creates a latency at start, measured in SECaaS_ConFx86-TS.02 just above, no sensible and measurable performance degradation is generated, simply because there is no change on the instructions once decrypted.	Pass
SECaaS_Intx86-TS.01	Verifiy the integrity preservation is effective; This will be done by leveraging D-MUTRA integrity verification solution	SECaaS_Intx86-TS.01-01	Check that a tampering is detected	Produce a tampering on the memory footprint	Step 1: Timestamps placed on the original code to assess the performance penalty Step 2: Timestamps placed at the same locations in the protected code	Two standard x86 native ELF formated workload	Tampering is detected	01/12/2024	Our integrity check hashes the integral text section of the executable. A single modified bit modifies the hash	Pass
SECaaS_Intx86-TS.02	Measure the integrity preserva	SECaaS_Intx86-TS.02-01	Measure the performance	Timestamping the workload to assess	Step 1: Timestamps placed on the original	Two standard x86 native ELF	5% (ie,50% of 10% for both	01/12/2024	Two techniques are used to diminish the performance impact (i) spread over time	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
	tion impact on performance. This will be done by leveraging D-MUTRA integrity verification solution		degradation	the performance penalty	code to assess the performance penalty Step 2: Timestamps placed at the same locations in the protected code	formated workload	integrity and monitoring)		hashing technique with adjustable idle time between two incremental operations (of the hashing) and (ii) affecting a cgroup resource allocation applied on the measuring thread	
SECaaS_Intx86-TS.03	Measure the remote attestation cycle. This will be done by leveraging D-MUTRA integrity verification solution	SECaaS_Intx86-TS.03	Measure a remote attestation full cycle (up to blockchain block creation)	Timestamping the workload to assess the performance penalty	Step 1: Timestamps placed on the original code to assess the performance penalty Step 2: Timestamps placed at the same locations in the protected code	Two standard x86 native ELF formated workload	1 second	01/12/2024	Pass	Pass
SECaaS_Availx86-TS.01	Assess the usability of self contain	SECaaS_Availx86-TS.01-01	For finite lifetime workload: Check the relevance	Monitoring the x86 by placing timestamps.	For finite lifetime workload only: Timestamp	Two standard x86 native ELF	Define a method easing the setup of probe	01/07/2026	Current technique, as explored in DESIRE-6G project is not satisfactory as it implies	RP2

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
	ed performance monitoring for x86 workloads.		of inserting timestamps at the entry point and exit. (ie, the difference between the timestamps bring the workload performance ratio) For infinite lifetime workload: Check the relevance of inserting trampolines with timestamps at duly defined CFG locations. Consider the call frequency of the relevant block and	Generate stress on the CPU.	the workload at entry and exit point. For infinite lifetime workload only: Step 1: LLVM-powered probe insertion x86 binary compilation Step 2: Stressing the CPU Step 3: Measure the call frequency or reference block time to execute to assess the workload performance ratio	formated workload	insertion, enabling the extraction of relevant CFG-inserted performance probes, for the characterization of the workload speed of operation.		modifications on the payloads through tampoline and timestamp insertions. The technique has developed two types of measurement (ie, call frequency and time to execute) on identified code blocks which requires a hybrid (ie, static + dynamic) analysis of the code. In Natwork, we intend to elevate these results by offering a simplify this technique (notably to boost the success factor **CSF 25= SECaaS friendliness**. In use case UC4.6, **we will explore the possibility to leverage LLVM in the compilation chain** of Montimage to ease the setting of timestamps, with a GUI enabling the user to select the instrumented	

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
			time to execute a plugged reference block to assess the performance.						function. This work will be in RP2	
SECaaS_Availx86-TS.02	Measure the performance impact of performance monitoring for x86 workloads	SECaaS_Availx86-TS.02-01	For finite lifetime workload: Check the relevance of inserting timestamps at the entry point and exit. (ie, the difference between the timestamps bring the workload performance ratio) For infinite lifetime workload: Check the relevance of inserting	Timestamping the workload to assess the performance penalty	Step 1: Prepare two variants: LLVM-powered probe insertion x86 binary compilation and unmonitored variant Step 2: Stressing the CPU Step 3: Measure the performance difference	Two standard x86 native ELF formated workload	Less than 10% (when cumulated with runtime integrity verification)	01/07/2026	Current technique, as explored in DESIRE-6G project is not 100% satisfactory as it implies modifications on the payloads through tampoline and timestamp insertions. The technique has developed two types of measurement (ie, call frequency and time to execute) on identified code blocks which requires a hybrid (ie, static + dynamic) analysis of the code. In Natwork, we intend to simplify this technique (notably to boost the success factor **CSF 25= SECaaS friendliness**. In use	RP2

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
			trampolines with timestamps at duly defined CFG locations. Consider the call frequency of the relevant block and time to execute a plugged reference block to assess the performance.						case UC4.6, **we will explore the possibility to leverage LLVM in the compilation chain** of Montimage to ease the setting of timestamps, with a GUI enabling the user to select the instrumented function. This work will be in RP2	

A.27 CIA-hardening of containerized payloads

Project Name:	**NATWORK**
Component Name:	**SECaaS**
Created by:	**TSS**
Date of creation:	**10.09.2025**
Filename:	**TSS-CIA hardening Containers payloads.xlsx**

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
SECaaS_InT-Cont-TS.01	For containerized x86 worklods, under the novel workflow-friendly scheme dubbed Drop and Attest, with an ON-DEMAND or periodic verifications, t**est the efficiency of the runtime integrity verifications**. Orchestration by K8s for a fully automated workflow.	SECaaS_InT-Cont-TS.01-01	Check that a tampering is detected	Produce a tampering on the memory footprint	1/ Produce a tampering produced during the execution of the workload 2/ Check the tampering detection	Production of the test on MMT probe, Liquid xAPP	Tampering is detected	01/04/2026	To be done in RP2	
SECaaS_InT-Cont-TS.02	For containerized x86 worklods, under the novel workflow-friendly scheme dubbed Drop and Attest, with ON-DEMAND or periodic verifications, test the efficiency of the runtime integrity verification.**Measure the integrity preservation impact on performance.**	SECaaS_InT-Cont-TS.02-01	Measure the performance degradation induced by the integrity verification in one of the two verification patterns (ie, on-demand, periodic)	Timestamping the workload to assess the performance penalty	Step 1: Timestamps placed on the original code to assess the performance penalty Step 2: Timestamps placed at the same locations in the protected code	Production of the test on MMT probe, Liquid xAPP	5%	01/04/2026	To be done in RP2	
SECaaS_Monit- Cont-TS.01	For containerized x86 worklods, with a sidecar mounted monitoring agent with privilege access on its linked container	SECaaS_Monit-Cont-TS.01-01	1/ Deliver sidecar with sufficient system privilege	Check sidecar container access permission to the	Step 1: Timestamps placed on the original code to assess the	Production of the test on MMT probe,		01/07/2026	To be done in RP2	

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
	memory, with an ON-DEMAND or periodic verifications. **The test shall demlonstrate the usability and effectiveness of the execution monitoring (e.g., delivery of proof of execution by sampling over the Instruction Pointer Register, assessment of the execution environment resource congestion, more accurate performance measurement).**		2/ Define different execution conditions (ie, resource congestion, modified data distribution (hence leading to a different cache hit rate) 2/ Execute the containerized workload under the different execution conditions as stated in 2/, 3/ Extract the monitoring elements corresponding to what is actually made possible	main container's memory stack	performance penalty Step 2: Timestamps placed at the same locations in the protected code	Liquid xAPP				

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
SECaaS_Monit-Cont-TS.02	Measure the performance impact of performance monitoring for Container workloads	SECaaS_Monit-Cont-TS.02-01	1/ Instrument an x86 containerized function 2/ Collect the performance metrics during execution	Check sidecar container access permission to the main container's memory stack	The test wil be done according to the result of SECaaS_Monit- Cont-TS.01 just above	Production of the test on MMT probe, Liquid xAPP	5%	01/07/2026	To be done in RP2	

A.28 CIA-hardening of WASM payloads Component

Project Name:	**NATWORK**
Component Name:	**SECaaS**
Created by:	**TSS**
Date of creation:	**10.09.2025**
Filename:	**TSS-CIA hardening WASM payloads.xlsx**

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result
SECaaS_ConFWasm-TS.01	**The test will be defined according to our feasibility study** . Verify Confidentiality resilience WASM	SECaaS_ConFWasm-TS.01-01	Check that workload file shows encrypted instruction text section	Pre-condition 1: Prior SECaaS processing the instructions are not encrypted.	Step 1: Check code section prior SECaaS operation Step 2: Check code section after SECaaS operation	Two standard WASM modules	Visual Inspection	01/04/2026	WASM confidentiality preservation technique has not been implemented yet, will be done in RP2

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result
SECaaS_ConFWasm-TS.02	**The test will be defined according to our feasibility study** Verify Confidentiality preservation technique impact on latency at start	SECaaS_ConFWasm-TS.02-01	Check that the decryption of the instructions (prior their are loaded and executed) is done in a given time slot	Pre-condition 1: Timestamped code available, enabling to assess with precision the latency at start	Step 1: Timestamped original WASM module is used to assess the time to reach the second timestamp Step 2: Timestamped protected WASM module latency at start is measured	Two standard WASM modules	Timing below threshold	01/04/2026	WASM confidentiality preservation technique has not been implemented yet, will be done in RP2
SECaaS_ConFWASM-TS.03	**The test will be defined according to our feasibility study**. Verify Confidentiality preservation impact on performance	SECaaS_ConFWasm-TS.03.01	Check the imperceptible impact on protected code performance	Timestamping the workload to assess the performance penalty	Step 1: Timestamps placed on the original code to assess the performance penalty Step 2: Timestamps placed at the same locations in the protected code	Two standard WASM modules	Performance degradation (% compared to baseline execution speed)	01/04/2026	WASM confidentiality preservation technique has not been implemented yet, will be done in RP2
SECaaS_Int-WASM-TS.01	Verify the integrity preservation is effective	SECaaS_Intx86-TS.01-01	Check that a tampering is detected	Produce a tampering on the memory footprint	Step 1: Timestamps placed on the original code to assess the performance penalty	Two standard x86 native ELF formatte	Tampering is detected	01/12/2024	pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result
					Step 2: Timestamps placed at the same locations in the protected code	d workload			
SECaaS_Int-WASM-TS.02	Measure the integrity preservation impact on performance	SECaaS_Int x86-TS.02-01	Measure the performance degradation	Timestamping the workload to assess the performance penalty	Step 1: Timestamps placed on the original code to assess the performance penalty Step 2: Timestamps placed at the same locations in the protected code	Two standard x86 native ELF formatted workload	5%	01/04/2026	To be done in RP2
SECaaS_Int - WASM-TS.03	Measure the remote attestation cycle	SECaaS_Int x86-TS.03-01	Measure a remote attestation full cycle (up to blockchain block creation)	Timestamping the workload to assess the performance penalty	Step 1: Timestamps placed on the original code to assess the performance penalty Step 2: Timestamps placed at the same locations in the protected code	Two standard x86 native ELF formatted workload	3 sec	01/04/2026	To be done in RP2

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result
SECaaS_AvailWASM-TS.01	**This results from the feasability study for fully support WASM**. In case of success, the test scenario will integrate means to generate stress on the CPU for the modification of the monitored runtime execution speed.	SECaaS_AvailWASM-TS.01-01	Instrument the WASMTIME runtime or the WASM module, typically by setting timestamps on the main bytecode instruction interpretation routine inserted in a loop, or alternatively by collecting call frequency of routine.		**The test will be defined according to our feasibility study**			01/04/2026	WASM Monitoring technique has not been implemented yet, will be done in RP2.
SECaaS_AvailWASM-TS.02	T**his results from the feasibility study for fully support WASM**. This test will assess the impact performance induced by the runtime monitoring as set in SECaaS_AvailWASM-TS.01	SECaaS_AvailWASM-TS.02-01	Two variants of the WASM runtime shall be compared in performance when executing a set of typical WASM modules. An average performance impact shall be defined.		**The test will be defined according to our feasibility study**			01/04/2026	WASM Monitoring technique has not been implemented yet, will be done in RP2**.**

A.29 Liquid RAN

Project Name:	**NATWORK**
Component Name:	**Liquid RAN**
Created by:	**ISRD**
Date of creation:	**2025/10/10**
Filename:	**ISRD-Anti-jamming.xlsx**

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
1.1	Jamming of PRACH channel	1.1	Verify system robustness under jamming on PRACH channel	• 5G NR SA cell active (n78 20 MHz) • Jammer calibrated & isolated • UE in Flight Mode ON state	1. Record baseline KPIs (RSRP, RSRQ). 2. Increase jammer power stepwise (-40 → 0 dBm). 3. UE Flight Mode OFF 4. Observe PRACH and RRC state. 5. Remove jammer → measure recovery.	Jam freq: 3.50 GHz ± 120 kHz; Power steps: 5 dB	UE should be able to attach	Reporting Period 2		
1.2	Jamming of DL control channel	1.2	Verify system robustness under jamming on PBCH / PDCCH channel	• 5G NR SA cell active (n78 20 MHz) • UE attached and UDP DL data call • Jammer calibrated & isolated	1. Record baseline KPIs (RSRP, TPUT, BLER). 2. UE attached to the network; UDP DL TP - 10Mbps 3. Increase jammer power stepwise (-40 → 0 dBm). 4. Observe PDCCH decode and RRC state. 5. Remove jammer → measure recovery.	Jam freq: 3.50 GHz ± 120 kHz; Power steps: 5 dB	RRC CONNECTED maintained	Reporting Period 2		

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
1.3	Jamming of DL shared channel	1.3	Verify system robustness under jamming on PDSCH channel	• 5G NR SA cell active (n78 20 MHz) • UE attached and UDP DL data call • Jammer calibrated & isolated	1. Record baseline KPIs (RSRP, TPUT, BLER). 2. UE attached to the network; UDP DL TP - 100Mbps 3. Increase jammer power stepwise (-40 → 0 dBm). 4. Observe PDSCH decode and RRC state. 5. Remove jammer → measure recovery.	Jam freq: 3.50 GHz ± 120 kHz; Power steps: 5 dB	RRC CONNECTED maintained	Reporting Period 2		
1.4	Jamming of DL shared channel with max TP	1.4	Verify system robustness under jamming on PDSCH channel with max TP	• 5G NR SA cell active (n78 20 MHz) • UE attached and UDP DL data call • Jammer calibrated & isolated	1. Record baseline KPIs (RSRP, TPUT, BLER). 2. UE attached to the network; UDP DL TP - 120Mbps 3. Increase jammer power stepwise (-40 → 0 dBm). 4. Observe PDSCH decode and RRC state. 5. Remove jammer → measure recovery.	Jam freq: 3.50 GHz ± 120 kHz; Power steps: 5 dB	RRC CONNECTED maintained	Reporting Period 2		
1.5	Jamming of DL shared channel for multiple UEs	1.5	Verify system robustness under jamming on PDSCH for	• 5G NR SA cell active (n78 20 MHz) • UEs attached and UDP DL data call	1. Record baseline KPIs (RSRP, TPUT, BLER). 2. UEs attached to the network; UDP DL TP - 100Mbps per every UE 3. Increase jammer	Jam freq: 3.50 GHz ± 120 kHz; Power steps: 5 dB	RRC CONNECTED maintained	Reporting Period 2		

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
			multiple UEs	• Jammer calibrated & isolated	power stepwise (-40 → 0 dBm). 4. Observe PDSCH decode and RRC state. 5. Remove jammer → measure recovery.					
1.6	Jamming of UL control channel	1.6	Verify system robustness under jamming on PUCCH channel	• 5G NR SA cell active (n78 20 MHz) • UE attached and UDP UL data call • Jammer calibrated & isolated	1. Record baseline UL KPIs (RSRP, TPUT, BLER). 2. UE attached to the network; UDP DL TP - 1Mbps 3. Increase jammer power stepwise (-40 → 0 dBm). 4. Observe PUCCH decode and RRC state. 5. Remove jammer → measure recovery.	Jam freq: 3.50 GHz ± 120 kHz; Power steps: 5 dB	RRC CONNECTED maintained	Reporting Period 2		
1.7	Jamming of UL shared channel	1.7	Verify system robustness under jamming on PUSCH channel	• 5G NR SA cell active (n78 20 MHz) • UE attached and UDP UL data call • Jammer calibrated & isolated	1. Record baseline UL KPIs (RSRP, TPUT, BLER). 2. UE attached to the network; UDP UL TP - 10Mbps 3. Increase jammer power stepwise (-40 → 0 dBm). 4. Observe PUSCH decode and RRC state. 5. Remove jammer → measure recovery.	Jam freq: 3.50 GHz ± 120 kHz; Power steps: 5 dB	RRC CONNECTED maintained	Reporting Period 2		

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
1.8	Jamming of UL shared channel for multiple UEs	1.8	Verify system robustness under jamming on PUSCH for multiple UEs	• 5G NR SA cell active (n78 20 MHz) • UEs attached and UDP UL data call • Jammer calibrated & isolated	1. Record baseline UL KPIs (RSRP, TPUT, BLER). 2. UEs attached to the network; UDP UL TP - 2Mbps per every UE 3. Increase jammer power stepwise (-40 → 0 dBm). 4. Observe PUSCH decode and RRC state. 5. Remove jammer → measure recovery.	Jam freq: 3.50 GHz ± 120 kHz; Power steps: 5 dB	RRC CONNECTED maintained	Reporting Period 2		
1.9	Jamming of DL and UL shared channels	1.9	Verify system robustness under jamming on PDSCH and OUSCH channels	• 5G NR SA cell active (n78 20 MHz) • UEs attached and UDP bidirectional data call • Jammer calibrated & isolated	1. Record baseline KPIs (RSRP, TPUT, BLER). 2. UE attached to the network; UDP Bidirectional TP - UDP DL 10Mbps / UDP UL 2Mbps 3. Increase jammer power stepwise (-40 → 0 dBm). 4. Observe PDSCH / PUSCH decode and RRC state. 5. Remove jammer → measure recovery.	Jam freq: 3.50 GHz ± 120 kHz; Power steps: 5 dB		Reporting Period 2		

A.30 Characteristics Extractor

Project Name:	**NATWORK**
Component Name:	**Characteristics Extractor**
Created by:	**GRADIANT**
Date of creation:	**09/11/2025**
Filename:	**GRAD-CharacteristicsExtract.xlsx**

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
CharExt-TS.01	2-Node extraction capability validation	CharExt-TS.01-TC.01	Alice-Bob Link enabled	1) 2-node setup operational (Alice and Bob) 2) Environment configured at the agreed frequencies	1) Configure the system in TDD and FDD 2) Generate UL/DL traffic in different scenarios 3) Capture and extract Alice–Bob measurements	Alice-Bob measurements	The setup provides realistic UL and DL behavior consistent with expected channel characteristics	15/04/2025	Alice and Bob channel measurements (I/Q samples for UL & DL) were extracted correctly and the training dataset has been generated.	Pass
CharExt-TS.02	3-Node extraction capability validation	CharExt-TS.01-TC.01	Eve link enabled	1) 3-node setup operational (Alice, Bob and Eve) 2) Environment configured at the	1) Configure the system in TDD and FDD 2) Generate UL/DL traffic in different scenarios 3) Capture and extract Eve	Eve measurements	The setup provides realistic Eve behavior consistent with expected channel characteristics	15/04/2025	Alice, Bob and Eve I/Q samples have been extracted correctly and the model validation and eavesdroppe	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
				agreed frequencies	measurements as a passive receiver				r datasets have been generated.	

A.31 Key Generator

Project Name:	**NATWORK**
Component Name:	**Key Generator**
Created by:	**GRADIANT**
Date of creation:	**09/11/2025**
Filename:	**GRAD-KeyGen.xlsx**

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
KeyGen-TS.01	Reciprocity Evaluation in TDD	KeyGen-TS.01-TC.01	ML prediction (UL to DL)	1) Channel Characteristics extracted from Alice-Bob link in TDD 2) ML model for reciprocity trained for TDD	1) Capture UL channel 2) Apply ML model to predict DL from UL 3) Compare predicted DL and measured DL	UL channel estimates, predicted DL, measured DL	The ML model try to reproduce DL behavior with acceptable accuracy than the old link	15/05/2025	For the OFDM-TDD 95 GHz scenario without noise, the neural network reaches a low validation MAE of 1.24×10^{-2} after 127 epochs.	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
KeyGen-TS.01	Key Generation Performance in TDD	KeyGen-TS.01-TC.02	Generation and disagreement test	1) DL extracted and predicted DL obtained from the TDD model 2) Quantification and Reconciliation models enabled to generate the key in TDD	1) Generate Alice-Bob TDD keys 2) Apply quantification and reconciliation blocks 3) Compute KGR and KDR	Alice–Bob bitstreams in TDD	The component is capable of produce keys and measure the generation and disagreement ratios in TDD	15/05/2025	The KDR after quantization is kept under 1% for the OFDM-TDD 95GHz scenario without noise.	Pass
KeyGen-TS.02	Reciprocity Evaluation in FDD	KeyGen-TS.01-TC.01	ML prediction (UL to DL)	1) Channel Characteristics extracted from Alice-Bob link in FDD 2) ML model for reciprocity trained for FDD	1) Capture UL channel 2) Apply ML model to predict DL from UL 3) Compare predicted DL and measured DL	UL channel estimates, predicted DL, measured DL	The ML model tries to reproduce DL behavior with acceptable accuracy than the old link	15/05/2025	For the OFDM-FDD 94/95 GHz scenario without noise, the neural network reaches a low validation MAE of 1.86 × 10−2 after 143 epochs.	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
KeyGen-TS.02	Key Generation Performance in FDD	KeyGen-TS.02-TC.02	Generation and disagreement test	1) DL extracted and predicted DL obtained from the FDD model 2) Quantification and Reconciliation models enabled to generate the key in FDD	1) Generate Alice-Bob FDD keys 2) Apply quantification and reconciliation blocks 3) Compute KGR and KDR	Alice–Bob bitstreams in FDD	The component is capable of produce keys and measure the generation and disagreement ratios in TDD	15/05/2025	The KDR after quantization is kept under 1% for the OFDM-FDD 94/95 GHz scenario without noise.	Pass

A.32 Security Evaluator

Project Name:	**NATWORK**
Component Name:	**Security Evaluator**
Created by:	**GRADIANT**
Date of creation:	**09/11/2025**
Filename:	**GRAD-SecurityVal.xlsx**

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
SecVal-TS.01	Security Validation in TDD for sub-THz	SecVal-TS.01-TC.01	Randomness Test	1) Pipeline operational in TDD 2) Bitstreams exported from main link	1) Generate a set of PKG Keys in TDD 2) Run NIST Test 3) Collect results and observations	Received Alice-Bob key bitstreams in TDD.	The generated PKG TDD key bitstreams show statistical properties consistent with a truly random sequence	23/05/2025	The Frequency test of NIST Test Suite has been run and passed with a sequence length equal to 256 and over 100000 binary sequences.	Pass
SecVal-TS.01	Security Validation in TDD for sub-THz	SecVal-TS.01-TC.02	Eavesdropping Test	1) Operational main link between Alice and Bob in TDD 2) Operational attacker capable of eavesdropping passively. 3) Bitstreams exported from all the components	1) Generate a set of PKG Keys from Alice-Bob and Eve in TDD 2) Compare both bitstreams and evaluate the level of disagreement.	Received Alice-Bob and Eve keys in TDD	Eve cannot reconstruct or correlate with the main key for TDD link.	23/05/2025	Eve Key Disagreement Ratio is over 45% for OFDM-TDD 95GHz scenario.	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
SecVal-TS.02	Security Validation in FDD for sub-THz	SecVal-TS.02-TC.01	Randomness Test	1) Pipeline operational in FDD 2) Bitstreams exported from main link	1) Generate a set of PKG Keys in FDD 2) Run NIST Test 3) Collect results and observations	Received PKG key bitstreams in FDD.	The generated PKG FDD key bitstreams show statistical properties consistent with a truly random sequence	23/05/2025	The Frequency test of NIST Test Suite has been run and passed with a sequence length equal to 256 and over 100000 binary sequences.	Pass
SecVal-TS.02	Security Validation in FDD for sub-THz	SecVal-TS.02-TC.02	Eavesdropping Test	1) Operational main link between Alice and Bob in FDD 2) Operational attacker capable of eavesdropping passively. 3) Bitstreams exported from all the components	1) Generate a set of PKG Keys from Alice-Bob and Eve in FDD 2) Compare both bitstreams and evaluate the level of disagreement	Received Alice-Bob and Eve keys in FDD	Eve cannot reconstruct or correlate with the main key for FDD link.	23/05/2025	Eve Key Disagreement Ratio is over 45% for OFDM-FDD 94/95 GHz scenario.	Pass

A.33 AI -Based Anomaly Detection Explainer

Project Name:	**NATWORK**
Component Name:	**Anomaly Detection Explainer Component**
Created by:	**UZH**
Date of creation:	**23.09.2025**
Filename:	**UZH-Anomaly Detection Explainer.xlsx**

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
XAI-UZH-TS01	Verify compilation and deployment	XAI-UZH-TS01-TC01	Build container, deploy service, verify health and endpoints	Docker/Pod runtime available; ML-IDS and MMT running; network reachability between services.	1) Build image 2) Deploy container 3) Check liveness/readiness 4) Curl REST/gRPC endpoints 5) Tail logs for errors	Container image & config (env vars, model registry URL).	Service healthy; endpoints registered; no startup errors.	Reporting Period 2	Deploy ok; no errors	
XAI-UZH-TS02	Verify alert ingestion and schema validation	XAI-UZH-TS02-TC01	Accept valid IDS alert payloads and validate JSON schema	MMT generates features; ML-IDS emits alerts; schema registry available.	POST valid alert to /explain; observe 200 OK and explanation doc ID	Sample alert JSON (valid schema).	Accepted, 200 OK; payload stored; no drops.	Reporting Period 2	schema valid; payload stored; no drops.	
XAI-UZH-TS02	Verify alert ingestion and schema validation	XAI-UZH-TS02-TC02	Handle malformed payloads gracefully	Same as TC01.	POST malformed payload; observe 400/422; WARN logged; service remains healthy	Sample alert JSON with missing/invalid fields.	Returns 4xx; no crash; pipeline unaffected.	Reporting Period 2	warnings logged; pipeline stable	

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
XAI-UZH-TS03	Verify explanation generation for malicious traffic and silence for benign	XAI-UZH-TS03-TC01	Produce explanation for DDoS burst alert	UE registered; traffic generator available; IDS model loaded.	Trigger short DDoS burst; wait for IDS alert; fetch explanation JSON and dashboard card	Replayable pcap: DDoS burst.	Top-k features, confidence, and operator summary rendered; mitigation hint available.	Reporting Period 2	DDoS explanation shown; top-k features	
XAI-UZH-TS03	Verify explanation generation for malicious traffic and silence for benign	XAI-UZH-TS03-TC02	Produce explanation for port scan alert	As above.	Trigger port scan; collect explanation outputs	Replayable pcap: TCP/UDP scan.	Explanation generated; correct factors (e.g., unique dst ports, bursts) highlighted.	Reporting Period 2	Port-scan explanation correct; key factors highlighted.	
XAI-UZH-TS03	Verify explanation generation for malicious traffic and silence for benign	XAI-UZH-TS03-TC03	Ensure no explanations for benign-only traffic	As above.	Run benign HTTP/MQTT/ICMP flows for 5 mins	Benign traffic set.	No alerts → no explanations.	Reporting Period 2	no alerts	
XAI-UZH-TS05	Measure fidelity and stability of explanations	XAI-UZH-TS05-TC01	Fidelity via perturbation/insertion-deletion test	Access to model scoring API; background dataset available.	Mask top-k features progressively; measure score delta (AUC)	Background dataset; captured alerts.	Fidelity ≥ 0.90 (target).	Reporting Period 2	Fidelity AUC 0.93	

A.34 Wirespeed traffic analysis in the 5G transport network

Project Name:	**NATWORK**
Component Name:	**Wirespeed traffic analysis in the 5G transport network**
Created by:	**CERTH**
Date of creation:	**01.09.2025**
Filename:	**CERTH-Wirespeed-traffic-analysis.xlsx**

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
Wirespeed-Traffic-analysis-TS01	Verify compilation and deployment	Wirespeed-Traffic-analysis-TS01-TC01	Compile P4 program for Agilio SmartNIC	P4 pipeline running	1. Compile P4 program 2. Deploy to Agilio SmartNIC	P4 program	Compilation successful, binary loads to Agilio SmartNIC	07/2024	Compilation successful, binary loads to Agilio SmartNIC	Pass
Wirespeed-Traffic-analysis-TS01	Verify compilation and deployment	Wirespeed-Traffic-analysis-TS01-TC02	Interconnection of CERTH IDS with the P4 Runtime	P4 pipeline running	1. Deploy 5G network 2. Deploy P4 program 3. Deploy CERTH IDS	Default traffic generator	CERTH IDS successfully parse and handle the ingress traffic from Agilio P4 SmartNIC	11/2024	CERTH IDS successfully parse and handle the ingress traffic from Agilio P4 SmartNIC	Pass
Wirespeed-Traffic-analysis-TS02	Verify packet classification	Wirespeed-Traffic-analysis-TS02-TC01	Validate benign traffic classification	Component running	1. Send benign traffic flows 2. Collect classification metadata	Default traffic generator	All packets classified as benign	02/2025	All packets classified as benign	Pass
Wirespeed-Traffic-analysis-TS02	Verify packet classification	Wirespeed-Traffic-analysis-TS02-TC02	Validate malicious traffic classification	Component running	1. Send traffic flows from the dataset 2. Collect	CICIDS2017 dataset	Packets correctly tagged as malicious	04/2025	Packets correctly tagged as malicious	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
					classification metadata					
Wirespeed-Traffic-analysis-TS03	Verify control-plane integration	Wirespeed-Traffic-analysis-TS03-TC01	Verify rules from control plane reflect classification results	Component running	1. CERTH IDS inference on the traffic flows 2. Push control rules (e.g. drop on malicious, forward on benign) 3. Verify on the P4 controller that the rules were applied	Benign + malicious traffic	Benign forwarded, malicious dropped	09/2025	Benign forwarded, malicious dropped	Pass

A.35 Detection and mitigation against jamming attacks

Project Name:	**NATWORK**
Component Name:	**Jamming detection and mitigation**
Created by:	**HES-SO**
Date of creation:	**01/04/2025**
Filename:	**HES-SO_Jamming.xlsx**

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
Jamming	Validating testbed	hesso-jamming-TS01-TC01	Single UE connected to testbed to validate correct behaviou	The addition of the SIM card into the WebUI to allow access to	Step 1: gNodeB and srsRAN containers running.	N/A	Connection succesful	15/04/2025	Pass	Pass

Test scenario ID	Test scenario	Test case ID	Test case	Pre-conditions	Test steps	Test data	Expected result	Execution date	Actual result	Status (Pass/Fail)
			r of testbed.	internet to that user.						
Jamming	Validating jammer	hesso-jamming-TS01-TC02	Single UE connected and USRP B210 radiating	Idem than TS01-TC01. Jammer correctly installed. Sometimes USRP B210 gives problem as gNodeB and GNURadio fights for the same USRP.	Step 1: Run the gNodeB and the 5G CN. Step 2: Wait for the UE to be connected. Step 3: Run the jammer container	Logs available inside the container gNodeB (srsRAN)	SINR & CQI dropping heavily and immediately after the jammer starts.	28/05/2025	Precisely we can report SINR dropping and other metrics stopped to report due to the communication fail.	Pass
Jamming	Validating jammer	hesso-jamming-TS01-TC03	Two UE connected and USRP B210 radiating	Idem than TS01-TC02. Adding the second SIM card.	Identical steps apart	Logs available inside the container gNodeB (srsRAN)	SINR & CQI dropping heavily and immediately after the jammer starts.	Not yet.	Add actual result	-