



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

D3.2 : Secure-by-design orchestration and management & Data plane computation offloading.r2

Lead beneficiary	CNIT	Lead author	Francesco Paolucci, Rana Abu Bakar
Reviewers	Nasim Nezhadsistani (UZH), Sándor Laki (ELTE)		
Type	R	Dissemination	PU
Document version	V1.0	Due date	31/03/2026



Co-funded by
the European Union

6G SNS

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	WP3
Associated task(s)	T3.1, T3.2
Main Author(s)	Francesco Paolucci, Rana Abu Bakar (CNIT)
Author(s)	Sumeyya Birtane, Shankha Gupta, Mays AL-Naday (UEssex), Wissem Soussi, Gökcan Cantali, Gürkan Gür (ZHAW), Péter Vörös, Mohammed Alshawki (ELTE), Konstantinos Pournaras, Kostas Lampropoulos (PNET), Maria B. Safianowska (ISR), Antonios Lalas, Virgilios Passas, Sarantis Kalafatidis, Nikolaos Makris, Donatos Stavropoulos, Stelios Mpatziakas, Ioanna Kapetanidou, Nikolaos Vakakis, Konstantinos Giapantzis, Georgios Agrafiotis, Maria Anna Pistela, Alexandros Papadopoulos, Vangelis V. Kopsacheilis, Thanasis Korakis, Anastasios Drosou (CERTH), Tom Goethals (IMEC), Marco Lucio Mangiacapre, Layal Ismail (CNIT)
Reviewers	Nasim Nezhadsistani (UZH), Sándor Laki (ELTE)
Type	R – Document, Report
Dissemination level	PU – Public
Due date	M27 (31/03/2026)
Submission date	31/03/2026

Document version history

Version	Date	Changes	Contributor (s)
r1.v0.1	02/12/2024	Initial table of contents	Mays AL-Naday, Sumeyya Birtane (UEssex)
r1.v0.2	10/01/2025	Updated ToC and partner assignment	Sumeyya Birtane Mays AL-Naday (UEssex)
r1.v0.3	21/01/2025	Further update to ToC and partner assignment	Sumeyya Birtane (UEssex)
r1.v0.4	04/02/2025	Further update to ToC and partner assignment	Francesco Paolucci (CNIT)
r1.v0.5	25/02/2025	Filled content for the parts: Section 4.1 - Moving Target Defense (MTD) Framework Section 6.4 - MTD	Gökcan Cantali, Wissem Soussi (ZHAW)
r1.v0.6	20/03/2025	Contribution for Section 2.3, 3.5, 5.2, 6.2, 6.8	Virgilios Passas (CERTH)
r1.v0.7	19/06/2025	Reviewed version	Sándor Laki (ELTE), Edgardo Montes de Oca (MONT)
r1.v0.75	22/6/2025	Refinement based on review comments	All authors
r1.v0.8	24/06/2025	Near final version	Mays AL-Naday (UEssex)
r1.v0.9	27/06/2025	Quality review	Joachim Schmidt, Leonardo Padial (HES-SO)
r1.v0.95	29/06/2025	Final review and refinements	Antonios Lalas, Virgilios Passas (CERTH) and CERTH team
r1.v1.0	30/06/2025	Final version for submission	Antonios Lalas (CERTH)
r2.v0.1	20/01/2026	Differential Table of Contents, partner assignment, differential contributions to be merged	Francesco Paolucci, Rana Abu Bakar (CNIT), all authors
r2.v0.2	26/02/2026	Merged Table of Contents, first round of contributions merging	Francesco Paolucci, Rana Abu Bakar (CNIT), all authors
r2.v0.3	06/03/2026	Final round of contributions, advancements beyond D3.1 subsections, ready for internal review	Francesco Paolucci, Rana Abu Bakar (CNIT), all authors
r2.v0.4	09/03/2026	Reviewed version	Nasim Nezhadsistani (UZH), Sándor Laki (ELTE)

Version	Date	Changes	Contributor (s)
r2.v0.5	11/03/2026	Editor and Authors Revisions	Francesco Paolucci, Rana Abu Bakar (CNIT), all authors
r2.v0.6	18/03/2026	Pre-Quality Check polishing	Francesco Paolucci (CNIT)
r2.v0.6.5	18/03/2026	Pre-quality check ISRD updated contributions	Maria Safianovska (ISRD), Francesco Paolucci (CNIT)
r2.v0.7	20/03/2026	Quality review	Joachim Schmidt, Anna Padée, Leonardo Padial (HES-SO)
r2.v0.8	24/03/2026	Refinement based on quality review comments	Mays AL-Naday (UEssex), Francesco Paolucci (CNIT), Virgilios Passas (CERTH)
r2.v0.9	27/03/2026	Final review and refinements	Virgilios Passas (CERTH) and CERTH Team
r2.v1.0	31/03/2026	Final version for submission	Antonios Lalas (CERTH)

Disclaimer

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or 6G-SNS. Neither the European Union nor the granting authority can be held responsible for them. The European Commission is not responsible for any use that may be made of the information it contains.

While the information contained in the documents is believed to be accurate, the authors(s) or any other participant in the NATWORK consortium make no warranty of any kind with regard to this material including, but not limited to the implied warranties of merchantability and fitness for a particular purpose.

Neither the NATWORK Consortium nor any of its members, their officers, employees, or agents shall be responsible or liable in negligence or otherwise howsoever in respect of any inaccuracy or omission herein.

Without derogating from the generality of the foregoing neither the NATWORK Consortium nor any of its members, their officers, employees, or agents shall be liable for any direct or indirect or consequential loss or damage caused by or arising from any information advice or inaccuracy or omission herein.

Copyright message

© NATWORK Consortium. This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation, or both. Reproduction is authorised provided the source is acknowledged.

Table of Contents

List of acronyms and abbreviations	12
List of figures	15
List of tables	17
Executive summary	18
1. Introduction	19
1.1. Purpose and structure of the document	19
1.2. Intended Audience.....	20
1.3. Interrelations	20
1.4. Advancements with respect to deliverable D3.1.....	21
1.4.1. Advancements in Software Design: Orchestrators (Section 2)	21
1.4.2. Advancements in Data Plane Computation Offloading (Section 3).....	21
1.4.3. Advancements in Orchestration Support Systems (Section 4).....	22
1.4.4. Advancements in Implementation (Section 5)	22
1.4.5. Advancements in Strategies and Optimisation Algorithms (Section 6).....	22
2. Software Design: Orchestrator(s)	23
2.1. Guided design and development by OSL patterns	23
2.1.1. Advancements beyond D3.1	25
2.2. Orchestration at the extreme edge	26
2.2.1. Novelty Beyond state-of-the-art.....	26
2.2.2. Functional components	27
2.2.3. Interfaces and Protocols	31
2.2.4. Data artefacts.....	32
2.2.5. Advancements beyond D3.1	33
2.3. Orchestration at the CRAN	34
2.3.1. Novelty Beyond state-of-the-art.....	34
2.3.2. Functional components	35
2.3.3. Interfaces and Protocols	37

2.3.4.	Data artefacts.....	38
2.3.5.	Advancements beyond D3.1	38
2.4.	Secure-by-Design Orchestration at the Core.....	40
2.4.1.	Novelty Beyond state-of-the-art.....	40
2.4.2.	Functional components	42
2.4.3.	Interfaces and Protocols	43
2.4.4.	Data artefacts.....	44
2.4.5.	Advancements beyond D3.1	44
3.	Data Plane Computation Offloading Design	45
3.1.	Offloading functions in the data plane	45
3.1.1.	NATWORK Offloading flavours	46
3.1.2.	Deployment and configuration interfaces.....	48
3.2.	Wirespeed AI (WAI) and Decentralized Feature Extraction (DFE).....	49
3.2.1.	Novelty Beyond state-of-the-art.....	51
3.2.2.	DFE/WAI in P4 programmable switches	52
3.2.3.	DFE/WAI in NVIDIA Bluefield-2 DPU	56
3.2.4.	Advancements beyond D3.1	59
3.3.	Data plane Machine Learning	60
3.3.1.	Novelty Beyond state-of-the-art.....	60
3.3.2.	Offloading mechanisms and network segment applicability	61
3.3.3.	Functional components	61
3.3.4.	Advancements beyond D3.1	63
3.4.	RAN security-performance balancer	63
3.4.1.	Novelty Beyond state-of-the-art.....	64
3.4.2.	Functional components	65
3.4.3.	Advancements beyond D3.1	66
3.5.	LLM-based IDS.....	67
3.5.1.	Novelty Beyond state-of-the-art.....	67
3.5.2.	Offloading mechanisms and network segment applicability	67

3.5.3.	Functional components	68
3.5.4.	Advancements beyond D3.1	70
4.	Software Design: Orchestration Support Systems	72
4.1.	Moving Target Defense (MTD) Framework	72
4.1.1.	Novelty Beyond State-of-the-Art	73
4.1.2.	MTD Controller	74
4.1.3.	Advancements Beyond D3.1	75
4.2.	Selective Cyber Threat Intelligence (CTI) solution	75
4.2.1.	Novelty beyond State-of-the-art	76
4.2.2.	Functional components	77
4.2.3.	Interfaces and Protocols	78
4.2.4.	Data artefacts.....	79
4.2.5.	CTI Cross-Domain selective Sharing.....	79
4.2.6.	Advancements beyond D3.1	81
4.3.	AI-based Behavioural Analysis service	81
4.3.1.	DFE/WAI	81
4.3.2.	Data plane ML	85
4.3.3.	Advancement beyond D3.1	88
4.4.	Microservice behavioural analysis	89
4.4.1.	Technical Description	89
4.4.2.	Functionalities Provided	90
4.4.3.	Dependencies.....	90
4.4.4.	Algorithms	91
4.4.5.	Interfaces and Protocols	92
4.5.	Security-performance balancer service	93
4.5.1.	Novelty beyond State-of-the-art	93
4.5.2.	Technical description	93
4.5.3.	Functionalities provided	94
4.5.4.	Dependencies.....	95

4.5.5.	Algorithms and workflow.....	95
4.5.6.	Interfaces and protocols	96
4.6.	Blockchain Based Trust Establishment	97
4.6.1.	Novelty Beyond state-of-the-art.....	97
4.6.2.	Technical Description.....	98
4.6.3.	Dependencies.....	99
4.6.4.	Functionalities Provided	99
4.6.5.	Workflow.....	99
4.6.6.	Interfaces and Protocols	100
4.6.7.	Advancement beyond D3.1	100
5.	Implementation	102
5.1.	Orchestration at the extreme edge	102
5.2.	Orchestration at the CRAN	102
5.3.	Orchestration at the Core (sFORK and SCOUT)	106
5.4.	WAI/DFE.....	107
5.5.	Security Performance Balancer	108
5.6.	LLM-based IDS.....	108
5.7.	Data plane ML.....	108
5.8.	MTD Controller	109
5.9.	Blockchain Based Trust Establishment	111
6.	Strategies and Optimisation Algorithms.....	113
6.1.	Orchestration at the extreme edge	113
6.1.1.	Strategies adopted.....	113
6.1.2.	Problem Definition.....	113
6.1.3.	Developed Solution.....	113
6.1.4.	Advancements beyond D3.1	114
6.2.	Orchestration at the CRAN	114
6.2.1.	Strategies adopted.....	114
6.2.2.	Advancements beyond D3.1	115

6.3.	Orchestration at the Core	116
6.3.1.	CTI Cross-Domain selective Sharing.....	116
6.3.2.	Workload Prediction for Scheduling	117
6.3.3.	Advancement beyond D3.1	117
6.4.	Moving Target Defence (MTD)	118
6.4.1.	Container Restore Algorithm for Kubernetes-aware Orchestration	118
6.4.2.	Parallel Live Migration Scheduling Using ML-based Time Prediction	119
6.4.3.	Advancements Beyond D3.1	120
6.5.	DFE/WAI offloading	122
6.5.1.	WAI and DFE for P4 switch DNN	122
6.5.2.	DFE and WAI in DPU-based mitigation	127
6.5.3.	Advancement beyond D3.1	130
6.6.	Data plane ML	131
6.6.1.	Feature Extraction.....	131
6.6.2.	Model Training and Online Learning	132
6.6.3.	Model Encoding for Match-Action Tables	133
6.6.4.	Model Disaggregation	133
6.6.5.	Advancements beyond D3.1	134
6.7.	RAN security-performance balancer	134
6.7.1.	Problem definition	134
6.7.2.	Developed Solution.....	135
6.7.3.	Advancement beyond D3.1	135
6.8.	LLM-based IDS.....	136
6.8.1.	Packet-token embedding optimization	136
6.8.2.	Contrastive learning and flow augmentation process.....	136
6.8.3.	Security policy enforcement	138
6.8.4.	Advancements beyond D3.1	138
7.	Conclusions	140
	References	141

List of acronyms and abbreviations

Abbreviation	Description
5GC	5G Core
AMF	Access and Mobility Management Function
AI	Artificial Intelligence
APT	Advanced Persisting Threat
ATD	Anomaly Traffic Detector
AUSF	Authentication Server Function
CNFs	Cloud-Native Functions
CNN	Convolutional Neural Network
CP	Control Plane
CRAN	Cloud Radio Access Network
CRIU	Checkpoint/Restore In Userspace
CTI	Cyber Threat Intelligence
DDoS	Distributed Denial of Service
DFE	Decentralized Feature Extraction
DN	Data Network
DNNs	Deep Neural Networks
DoS	Denial of Service
DoSt	Denial of Sustainability
DPDK	Data Plane Development Kit
DPU s	Data Processing Units
EI	Edge Intelligence
EMA	exponential moving average
ETSI	European Telecommunications Standards Institute
FL	Federated Learning
FQDN	Fully Qualified Domain Name
GL	Gossip Learning
GTP	GPRS Tunnelling Protocol
IAT	Inter-Arrival Time
IDS	Intrusion Detection System
KNN	K-Nearest Neighbors
KPI	Key Performance Indicator
KPM	Key Performance Monitoring
LiMi	Live Migration
LLM	Large Language Model
LUTs	Lookup Tables
MANO	Management and Orchestration
MEC	Multi-Access Edge Computing
ML	Machine Learning

Abbreviation	Description
MLM	Masked Language Modelling
MLP	Multi-layer Perceptron
MP	Management Plane
MTD	Moving Target Defense
NCL	Network Convergence Laboratory
NFS	Network File System
NFV	Network Functions Virtualization
NIDS	Network Intrusion Detection Systems
NSD	Network Service Descriptor
NWDAF	Network Data Analytics Function
OAFA	Orchestration and Flow Agent
OAI	OpenAirInterface
OAM	Open Application Model
OCI	Open Container Initiative
OFA	Offloaded Function Agent
O-RAN	Open Radio Access Network
OSL	OpenSLice
OSS	Operation Support System
OVS	Open Virtual Switch
PID	Process ID
PRB	Physical Resource Block
QAT	Quantization-Aware Training
RAN	Radio Access Network
RC	Radio Resource Control
RIC	RAN Intelligent Controller
RRC	Radio Resource Control
RSS	Receive Side Scaling
RT-RIC	Real Time RAN Intelligent Controller
SBA	Service-Based Architecture
SD	Slice Differentiator
SDN	Software-Defined Networking
SDO	STIX Data Object
SLA	Service Level Agreement
SM	Service Models
SMF	Session Management Function
S-NSSAI	Single Network Slice Selection Assistance Information
SSL	Self-Supervised Learning
SST	Slice Service Type
STIX	Structured Threat Information eXpression

Abbreviation	Description
TAXII	Trusted Automated eXchange of Intelligence Information
TC	Traffic Control
TCP	Transmission Control Protocol
TEE	Trusted Execution Environment
TMF	TeleManagement Forum
UDM	Unified Data Management
UDR	Unified Data Repository
UE	User Equipment
UPF	User Plane Function
VNF	Virtualized Network Function
WAI	Wirespeed Artificial Intelligence

List of figures

Figure 1: OSL Reference Architecture	23
Figure 2: Service Lifecycle Workflow	25
Figure 3: Feather architecture, with main components in yellow (orchestration) and green (networking).....	29
Figure 4: Overview of Flocky services, their main components, and interaction between nodes	30
Figure 5: Updated Flocky architecture including ML/Gossip services.....	31
Figure 6: End-to-End Deployment of the AI-Driven Network Intrusion Detection 5G Network..	34
Figure 7: O-RAN Architecture	37
Figure 8: Training time and Model accuracy comparison	39
Figure 9: Secure-by-Design Orchestration Architecture and components	43
Figure 10: NATWORK Data Plane offloading solutions and flavours.....	46
Figure 11: Offloading deployment and runtime configuration options	48
Figure 12: Embedding AI/ML inside programmable switches.....	50
Figure 13: DFE and WAI design submodules	51
Figure 14: P4 Parser and pipeline for a 6-feature input	53
Figure 15: LUT distilled DNN and its P4 pipeline	54
Figure 16 WAI final design – A single P4 for DDoS and DDoS adversarial attacks	56
Figure 17: DPU architecture.....	57
Figure 18 DPU Wire-Speed Feature Extraction Final Schema of the software and hardware entities involved in Attack Mitigation.....	59
Figure 19: Data plane ML model.....	63
Figure 20: Security-performance balancer architecture	66
Figure 21: LLM-IDS Inference Pipeline.....	68
Figure 22: LLM-Based IDS Overview	69
Figure 23: Architecture of the MTD framework	72
Figure 24: CTI Solution Architecture and components.....	78
Figure 25: Simple confusion matrix for decision making strategy	80

Figure 26: Sample Sensitivity and Necessity maps for decision making strategy	80
Figure 27: Data plane ML components and interactions	86
Figure 28: Position of the microservice behavioural analysis module and interconnection to other modules.....	90
Figure 29: UDP Flooding Attack Execution	92
Figure 30: New flow control rule (left) and a graphical representation of flow rules (right).....	92
Figure 31: ISRD Liquid Near-RT RIC interfaces.....	94
Figure 32: ISRD Liquid Near-RT RIC general deployment architecture.	94
Figure 33 Security-performance load balancer workflow	96
Figure 34: Pseudonym Generation Monitoring Stack	101
Figure 35: Model Training Evaluation - Accuracy, ROC AUC and F1 Score Comparison	104
Figure 36: Detailed Architecture of the AI-Driven Network Intrusion Detection System	105
Figure 37: CRIU steps	109
Figure 38: ML Classifier + regressor to optimize the live migration of microservice-based applications.....	119
Figure 39: simplified VirtualService configuration.....	121
Figure 40 Kiali traffic visualization during live migration.....	122
Figure 41: DNN to lookup distillation method.....	125
Figure 42: DOCA-based offloaded DDoS mitigator.....	128
Figure 43: Offloaded DDoS mitigator algorithm.....	130
Figure 44: Overview of the contrastive learning and augmentation process	137

List of tables

Table 1: Interface to MTD Controller	75
Table 2: OFA methods.....	84
Table 3: Interface to P4 Runtime	88
Table 4: Interface to SDN Controller to enforce flow rules	92
Table 5: Interface to Microservice Orchestrator to report irregular resource usage and trigger scaling decisions.....	93
Table 6: Interface to monitoring engine to retrieve real-time monitoring data.....	93
Table 7: Interface to O-RAN A1.....	96
Table 8: Interface to O-RAN E2	96
Table 9: Interface to RIC.....	97
Table 10: Interface to Distributed Insertion	100
Table 11: Interface to Distributed Query.....	100
Table 12: Interface to Token Verification	100
Table 13: Training and inference times for various machine learning models	105

Executive summary

The deliverable D3.2 “Secure-by-design orchestration and management & Data plane computation offloading.r2” presents the final design principles, architectural components, and implementations, superseding D3.1, toward achieving secure-by-design orchestration and management of 6G network slices, as well as exploring advanced mechanisms for data plane computation offloading. As part of the NATWORK project’s broader goal to enable secure, sustainable, and resilient 6G services, this document focuses on developing decentralised and intelligent management services capable of responding to emerging security threats while addressing energy efficiency and trust in multi-domain environments.

The report introduces the final versions of decentralised orchestration components, services and algorithms capable of maintaining service continuity under evolving cybersecurity threats while optimising energy consumption across the edge-to-cloud continuum. It also outlines advanced approaches for offloading computation into the network data plane to reduce latency and enhance in-network intelligence.

The deliverable further defines critical supporting frameworks such as decentralised Cyber Threat Intelligence (CTI) exchange, AI-based behavioural analysis, Moving Target Defence (MTD) mechanisms, and security-performance balancer services. These modules work together to strengthen the orchestration platform and dynamically adapt to operational and threat conditions.

The final implementation strategies, in the context of the NATWORK project, are described alongside validation approaches and methodologies to evaluate orchestration effectiveness, energy efficiency, and security performance in the context of 6G networks. The results of this work lay the foundation for upcoming large-scale integration and validation of the NATWORK solution in the final WP6 evaluation and demonstration activities.

1. Introduction

The rapid evolution of 6G networks demands innovative solutions to manage the complexity of massive device connectivity, data-intensive applications, and escalating security threats in edge-to-cloud ecosystems. The deliverable "*Secure-by-design orchestration and management & Data plane computation offloading*" addresses these challenges by presenting a cohesive framework for decentralised orchestration, secure management, and optimised data plane computation offloading. The purpose of this deliverable is to provide a detailed account of the software design, implementation strategies, and validation outcomes for orchestration and offloading in 6G networks. In particular, this version r2 represents the final outcome of Task 3.1 and 3.2 activities on orchestration and data plane offloading solutions, respectively. This version inherits the content of release r1 and includes the final design solutions for components and services implementing security orchestration and data plane offloading aspects of the whole NATWORK solutions, ready for integration and validation in the WP6 Use Case Pilots. Moreover, it includes the novelty beyond the state of the art introduced by each component and service. The document is organised into several sections to guide the reader through NATWORK's key contributions. Following the Executive Summary, this introductory chapter (Section 1) is divided into subsections: Section 1.1 outlines the purpose and structure of the document, Section 1.2 describes the intended audience, and Section 1.3 highlights how the work connects to broader research and development initiatives.

1.1. Purpose and structure of the document

The purpose of the "*Secure-by-design orchestration and management & Data plane computation offloading*" deliverable is to present a comprehensive overview of the NATWORK project's advancements in developing secure, sustainable, and efficient 6G network solutions. It details the final design, implementation, and validation of a decentralised orchestration framework alongside innovative data plane computation offloading strategies to address critical challenges in energy consumption, cybersecurity, and computational efficiency in edge-to-cloud 6G ecosystems. By integrating real-time Cyber Threat Intelligence (CTI), AI-driven analytics, and energy-efficient algorithms, this document demonstrates how the project enables service continuity and aligns with Net-Zero and EU Horizon objectives.

Following the Introduction, which sets the stage for the document's purpose, audience, and its interconnections within the project's framework, the structure continues as follows:

- **Section 2 – Software design: Orchestrators:** Presents the project's orchestration service designs, detailing how secure, decentralized, and intent-compliant orchestration is achieved across extreme edge, CRAN, and core network domains.

- **Section 3 – Data Plane Computation Offloading Design:** Describes the NATWORK strategies for computation offloading, including offloading classifications, Wirespeed AI (WAI), Decentralized Feature Extraction (DFE), and the deployment of in-network machine learning models.
- **Section 4 – Software Design: Orchestration Support Systems:** Presents the NATWORK support services, such as the Moving Target Defense (MTD) framework, CTI selective sharing mechanisms, AI-based behavioral analysis, and the security-performance balancer, which enhance orchestration resilience and adaptability.
- **Section 5 – Implementation:** Describes the final implementation of the orchestration and offloading components.
- **Section 6 – Strategies and Optimisation Algorithms:** Details the strategies and optimization methods used to enable adaptive orchestration, proactive threat mitigation, and energy-efficient service management.
- **Section 7 – Conclusions:** Wraps up the document by reflecting on NATWORK's strategic direction, summarizing achievements, and establishing expectations for the upcoming validation and integration milestones.

1.2. Intended Audience

The Deliverable D3.2 *"Secure-by-design orchestration and management & Data plane computation offloading.r2"* is devised for public use in the context of project management and dissemination/ communication activities of the NATWORK consortium, comprising members, project partners, and affiliated stakeholders. This document mainly focuses on the secure-by-design orchestration, management frameworks, and data plane computation offloading aspects of the project, thereby serving as a referential tool throughout the project's lifespan. Also, the document highlights the strategic blueprint and collective vision of the project, ensuring that all collaborative efforts are harmonised and directed toward the fulfilment of the project's ambitions.

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's 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. The D3.2 "Secure-by-design orchestration and management & Data plane computation offloading.r2" deliverable addresses all activities of the NATWORK project related to the design, development, and validation of secure, resilient, and energy-efficient orchestration frameworks, as well as advanced data plane offloading mechanisms. It interrelates closely with architectural work defined in WP2, security and orchestration advancements from WP3, AI-driven management solutions from WP4, and integration and validation efforts within WP6, ensuring consistency and alignment across the project's technical pillars.

1.4. Advancements with respect to deliverable D3.1

The main advancements are summarized below per section of the document. Moreover, specific subsections, called "Advancements beyond D3.1", detail the advancements in each section of the deliverable for each considered topic. The goal is to clearly describe the innovation of the NATWORK solutions with respect to the state of the art described in deliverable D2.1.

1.4.1. Advancements in Software Design: Orchestrators (Section 2)

The orchestration framework has been extended across the extreme edge, CRAN, and core domains. Feather and Flocky have been enhanced with Gossip and ML services enabling decentralized data exchange and distributed learning at the edge. In the CRAN domain, the framework integrates AI-based anomaly detection with radio resource management, enabling automated mitigation actions validated in an OpenAirInterface-based 5G environment. At the core, the secure-by-design federated orchestrator (sFORK) has reached its final design and implementation stage, enabling distributed service placement across clusters while considering security, resource, and energy constraints.

1.4.2. Advancements in Data Plane Computation Offloading (Section 3)

The data-plane offloading architecture has been finalized to support AI-driven security functions embedded directly in programmable network devices. The Decentralized Feature Extraction (DFE) module enables line-rate traffic feature extraction on DPUs, while Wirespeed AI (WAI) enables machine learning inference inside programmable switches. Additional contributions

extend the framework with data-plane ML pipelines, RAN security-performance balancing mechanisms, and LLM-based intrusion detection approaches, enabling heterogeneous security functions across switches, DPUs, and RAN components.

1.4.3. Advancements in Orchestration Support Systems (Section 4)

Security support services have been extended with improved capabilities for dynamic threat mitigation and cross-domain intelligence sharing. These include the Moving Target Defense (MTD) framework, Selective Cyber Threat Intelligence (CTI) sharing mechanisms, and AI-based behavioural analysis services, enabling coordinated threat detection and mitigation across distributed infrastructures.

1.4.4. Advancements in Implementation (Section 5)

Compared to D3.1, this deliverable reports the final implementation progress of several NATWORK components, including orchestration frameworks, programmable data-plane security functions, and orchestration support services. These components have been integrated and tested in experimental environments including programmable switches, DPU-based mitigation platforms, and 5G testbeds, demonstrating the feasibility of the proposed architecture.

1.4.5. Advancements in Strategies and Optimisation Algorithms (Section 6)

The deliverable also introduces extended and refined strategies and algorithms supporting orchestration and security enforcement. These include improved approaches for distributed orchestration decisions, data-plane machine learning model encoding, WAI/DFE offloading strategies, RAN security-performance balancing, and LLM-based intrusion detection, improving scalability, efficiency, and responsiveness of the NATWORK framework.

2. Software Design: Orchestrator(s)

This section introduces the software design of the orchestration layer, describing how orchestrators enable secure, decentralized, and intent-driven coordination across extreme edge, CRAN, and core network domains.

2.1. Guided design and development by OSL patterns

OpenSlice (OSL) is an open-source operations support system designed to provide support for VNF/NSD onboarding and management. The platform supports TM FORUM OpenAPIs related to Service Catalog Management, Ordering, Resource, and more. It enables NFV developers to onboard and manage VNF and network service artifacts, while allowing vertical customers to browse available service specifications.

While OSL itself is not a direct component of the NETWORK project, it remains highly relevant. The experience and insights gained from OSL in terms of modular design, API integration, and automation have contributed to the development of NETWORK orchestrator services. Conversely, innovations and service orchestration strategies emerging from NETWORK can be applied back into the OSL ecosystem to enhance its capabilities. This mutual influence fosters stronger alignment between open-source frameworks and emerging innovations in 6G orchestration and management.

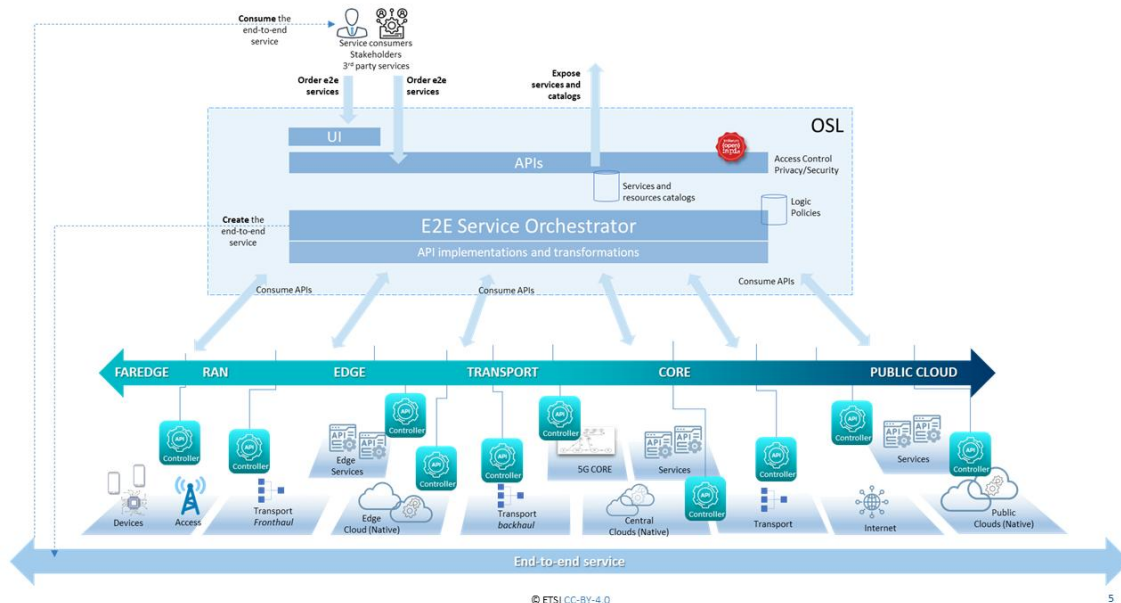


Figure 1: OSL Reference Architecture

OSL design principles pave the way for a modular architecture where each component has a well-defined role, promoting separation of concerns and facilitating easier maintenance and scalability (Figure 1). Key aspects include:

- **Service Catalogs and Specifications:** Services are defined using standardized templates, enabling consistent exposure and management within the Service Catalogs. This approach allows for both predefined network services and the flexibility to define custom configurations.
- **Standardized APIs:** Utilizing TM Forum Open APIs (e.g., TMF 638 for Service Inventory, TMF 641 for Service Ordering) ensures seamless integration with external systems and promotes interoperability across different domains.
- **Automation and Workflow Management:** The orchestration engine supports automated workflows, managing the provisioning, configuration, and lifecycle of network services. This automation is crucial for efficient service delivery and adherence to predefined policies and standards.

OpenSlice focuses on aspects related to 6G slice lifecycle management by supporting the modelling, ordering, and orchestration of services that underpin network slices. For example, it can handle slice templates representing vertical services, manage slice instantiation requests, and interface with lower-layer domain orchestrators responsible for RAN, core or transport slicing. This enables coordination of network slice deployment and assures service-level requirements in an end-to-end manner. These capabilities are complementary to NATWORK orchestration services, which focus on distributing and peering services across clusters where the security requirements of assigned services and hosting clusters are met; thereby, providing secure-by-design end-to-end slice operation over multiple domains.

The orchestrator design inherently supports closer interaction and interfacing between components:

- **End-to-End Service Orchestration:** The orchestrator coordinates with various domain controllers (e.g., SDN, NFV, RAN) to provision and manage services across the entire network stack, from user devices to core networks and cloud services.
- **Lifecycle Management (Figure 2):** Services undergo a comprehensive lifecycle, including provisioning, monitoring, scaling, and decommissioning. OSL provides the framework for

managing these stages effectively, ensuring services operate as intended throughout their lifespan.

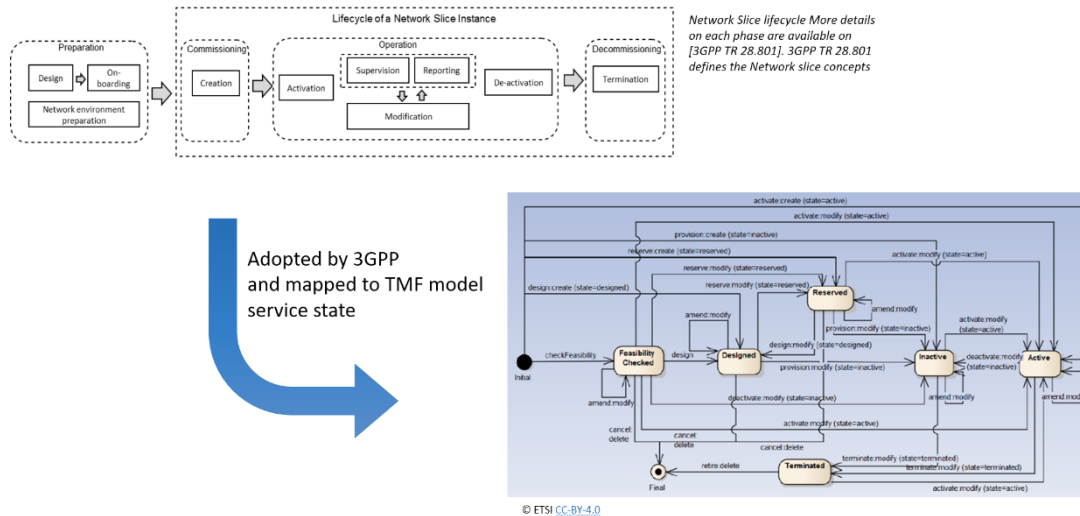


Figure 2: Service Lifecycle Workflow

- **Resource and Service Inventory Management:** Maintaining accurate records of both resources and services is essential. OSL defines mechanisms for tracking and managing these inventories, enabling real-time insights and efficient resource utilization.

By integrating these OSL design principles into the orchestrator design, we establish a robust foundation that promotes efficient service delivery, adaptability to evolving requirements, and seamless integration within the broader network ecosystem.

2.1.1. Advancements beyond D3.1

NATWORK project continues to actively engage with the ETSI SDG OSL (Software Development Group OpenSlice) community through pNET, monitoring its innovations, relevant architectural practices and operational insights across the broader orchestration ecosystem. Further to this, tangible collaboration activities between NATWORK and the OSL investigate how the OSL can operate as an application-level client for the project's sFORK service, by exploring architectural and interfacing approaches. These will allow OSL components to be distributed across multiple different clusters based on security posture, resource availability, sustainability criteria, etc. using sFORK's service management solution.

At this stage, the contribution of NATWORK focuses on defining interoperability mechanisms and architectural integration patterns that would allow OpenSlice to leverage secure, federated service deployment capabilities provided by sFORK. With this collaboration NATWORK will be

exploring opportunities to contribute to the SDG OSL a complementary external service for optimal deployment of OpenSlice.

2.2. Orchestration at the extreme edge

Workload orchestration at the (extreme) edge is achieved through two frameworks developed during the project: Feather, a Kubernetes-compatible agent, leverages multiple runtimes in addition to containers, allowing the ideal execution format for any single deployment (e.g., microVM, container). While microVM support was already present, WASM support and cross-runtime pod networking, as well as Flocky support, have been specifically added for NETWORK. Flocky is a higher-level framework designed as a decentralized alternative to Kubernetes, which leverages Feather as a deployment agent. Flocky is designed around the Open Application Model (OAM), and detects node capabilities (e.g. security options, attestation) which can be used by deployments as required. Both Feather and Flocky were first detailed in D3.1. This section summarizes their design and how new components fit the Feather/Flocky architecture.

2.2.1. Novelty Beyond state-of-the-art

Several aspects of Feather improve beyond the state of the art:

- Unlike standard Kubernetes-based approaches, it transparently supports the mixed use of various (i.e., more than one) non-container runtimes in an extensible framework. The runtime implementations are compatible with Open Container Initiative (OCI) standards, remaining compatible with existing Kubernetes clusters. Furthermore, these runtimes may be combined in a single unit of deployment (Pod), which is unsupported by other orchestrator agents, or Operators/extensions in the case of Kubernetes.
- Its networking features provide a decentralized container network (Warrens) in which each device can bootstrap its own networking infrastructure, and connect to the rest of the cluster. Furthermore, true mixed-runtime pod networking is supported which enables transparent inter- and intra-pod network communication between workloads in different runtimes as if they were deployed as containers.

Flocky decentralizes what is commonly known as the “control plane” of an orchestrator across an entire cluster. As such, the novelty is that each node can dynamically take on specific roles and tasks depending on its capabilities. Node software/hardware capabilities are detected in real-time by the framework (with plug-in capabilities for third party services detecting specific features), modeled using OAM, and advertised to other nodes by exposing a metadata store service. In a client role, nodes need only deploy a discovery service and a Swirly service which matches workloads required by the client to “orchestrator” nodes. In an orchestrator role, nodes can deploy the entire Flocky stack including Feather to support multi-runtime application

deployments. The core Flocky services were presented in D3.1; since then additional Gossip and ML services have been added to support efficient application data dissemination, as well as Edge Intelligence (EI) and Gossip/Federated Learning (GL, FL).

2.2.2. Functional components

Figure 3 shows the relations between the various components in Feather (listing only the relevant ones, excluding implementation details):

- **Pod Manager:** handles pod-level logic, splitting each deployment into individual workloads to be handled by providers. This component provides minimal info to the PodNetwork manager for pod-level networking.
- **Providers:** Each provider represents a single type of workload e.g. containers, unikernels. The API is standardized in an interface, and based on the Open Container Initiative Image Specification for extensibility.
- **Pod WAN/Network/Address Manager:** components for a custom pod networking framework which allows workloads from different runtimes in the same pod to communicate as if they were all container-based.
- **eBPF traffic routing:** traffic routing for the pod networking framework is based on various eBPF programs to enhance performance, handling traffic at the kernel level. Instances and configuration are managed from the PodNetwork manager.
- **Workload runtimes:** while not directly a component of Feather, at least one of these is required on each Feather node for Feather to work correctly; e.g., containerd, KVM.

Specifically, Figure 3 shows how mixed-runtime networking is combined with the Feather architecture: multi-runtime images are created by packaging non-container formats (e.g., OSv unikernels) into an OCI image and adding metadata according to OCI standards. When Feather receives a pod deployment manifest from Kubernetes through the Kubelet API, the Pod Manager (1) pulls the required workload images, examining the metadata to determine the required runtime providers; examples are the Containerd provider and OSv provider components (2). This approach avoids any changes to the Kubernetes cluster itself, or having to customize deployment manifests. The providers only handle workload image management, resource restrictions (e.g., cgroups), and basic runtime parameters to call their runtimes. Importantly, Feather is aimed at low-resource edge devices, and by default provides its own container networking implementation using round-robin pod IP assignment rather than relying on complex APIs and CNI plugins, while blocking default CNI plugin deployment on nodes running Feather to avoid interference.

Mixed-runtime pod networking is enabled at the pod level by interaction of the Pod Manager with the PodNetwork Manager (3), which creates a dedicated bridge interface for each pod using the pod IP address, and a dedicated pod network namespace connected to the bridge. Different implementations of this component exist, matching various network modes (e.g., "legacy" for round-robin IP assignment, or "full multi runtime"). The pod network namespace supports any workload process that can directly access the container network interface inside it (e.g., "cni0" for containers), and can be optionally disabled for performance reasons if no workloads within a pod require it. Any pod workload that requires its own interface, e.g. a TAP device for a qemu VM, is attached to the bridge instead of moving it into the pod network namespace. To support cross-runtime networking, providers must call either a PodNetwork Manager function which assigns a workload to the pod network namespace, or a function which creates a sub-pod ready TAP device for a workload to use; in Feather the Containerd provider adds containers to the pod network namespace, while the OSv provider requests TAP devices.

The PodNetwork Manager and providers interact with the PodAddress Manager (4), which generates IP addresses from a self-assigned address pool. In addition to pod- and workload level network initialization, the PodNetwork manager is responsible for sub-pod networking, the core functionality which enables mixed-runtime networking by treating each pod as a miniature subnet under its assigned pod IP address. To that end, Feather uses custom eBPF traffic routing programs (5) to ensure optimal performance, and to disguise sub-pod networking from the point of view of other pods, as well as from other Kubernetes nodes. The lifecycles and routing tables of these programs are handled by the PodNetwork manager based on provider and PodAddress manager interaction. Finally, the PodNetwork manager also interacts with the PodWAN Manager (6) to determine how external pod traffic should be routed by eBPF programs to the eBPF Tunneling component.

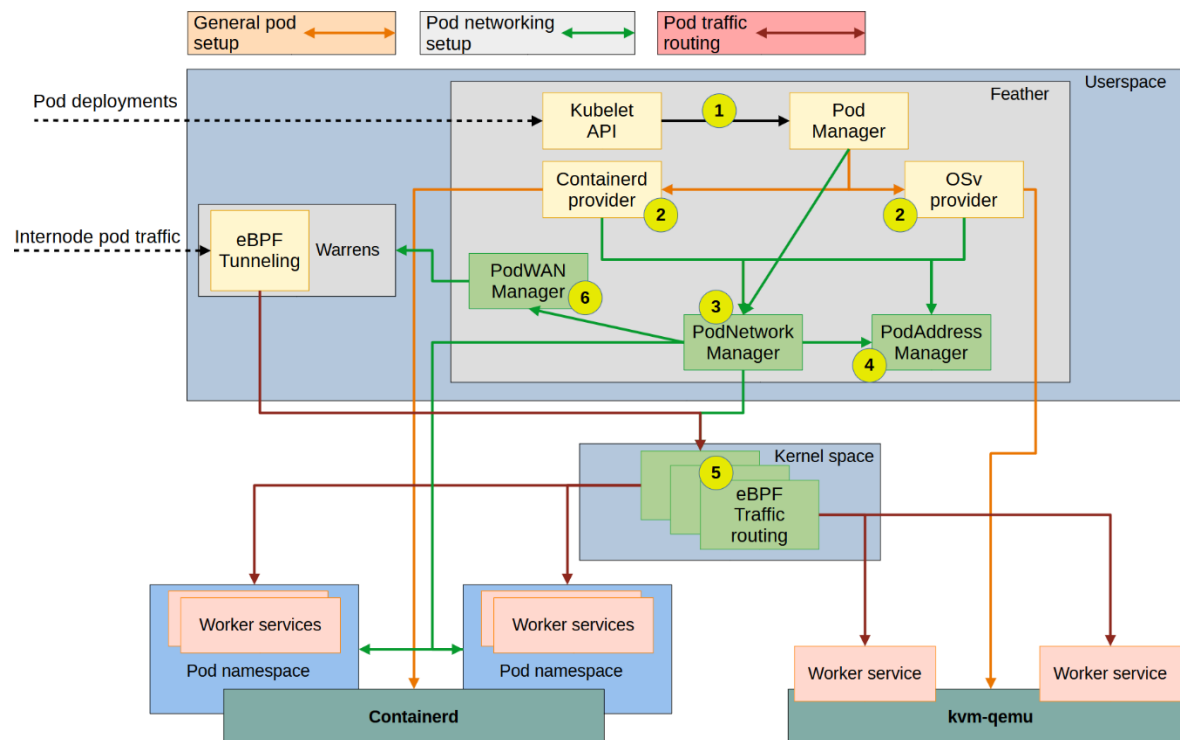


Figure 3: Feather architecture, with main components in yellow (orchestration) and green (networking).

At the orchestration level, Flocky (Figure 4) uses three main services, each of which may be (partially) deployed on any node in a cluster depending on its role:

- **Discovery Service:** responsible for finding other Flocky nodes in the cluster, within a preconfigured latency range. The discovery process is entirely decentralized and resembles gossip-based networks or algorithms. It operates entirely at the network level, gathering only the required properties for node identification and communication. Other services may subscribe to updates from the Discovery service.
- **Metadata Repository/service:** The Repository service gathers additional metadata from discovered nodes based on an extension of the Open Application Model. Specifically, it collects hardware resource status, important node properties (e.g. dedicated hardware, attestation, security), and operational status e.g. applications and detected runtimes. Metadata collection is highly flexible and handled through Capability providers, while the actual metadata is stored in a local repository.
- **Swirly/Deployment services:** workload deployment is split into two separate services as Swirly (orchestration) and Deployment. The Swirly service receives requests for application deployments (i.e. one or more workloads), splits them into multiple parts

based on workload requirements and discovered node capabilities, and deploys each workload on the most suitable node. For orchestration-only nodes, the Deployment service may be ignored, and vice versa.

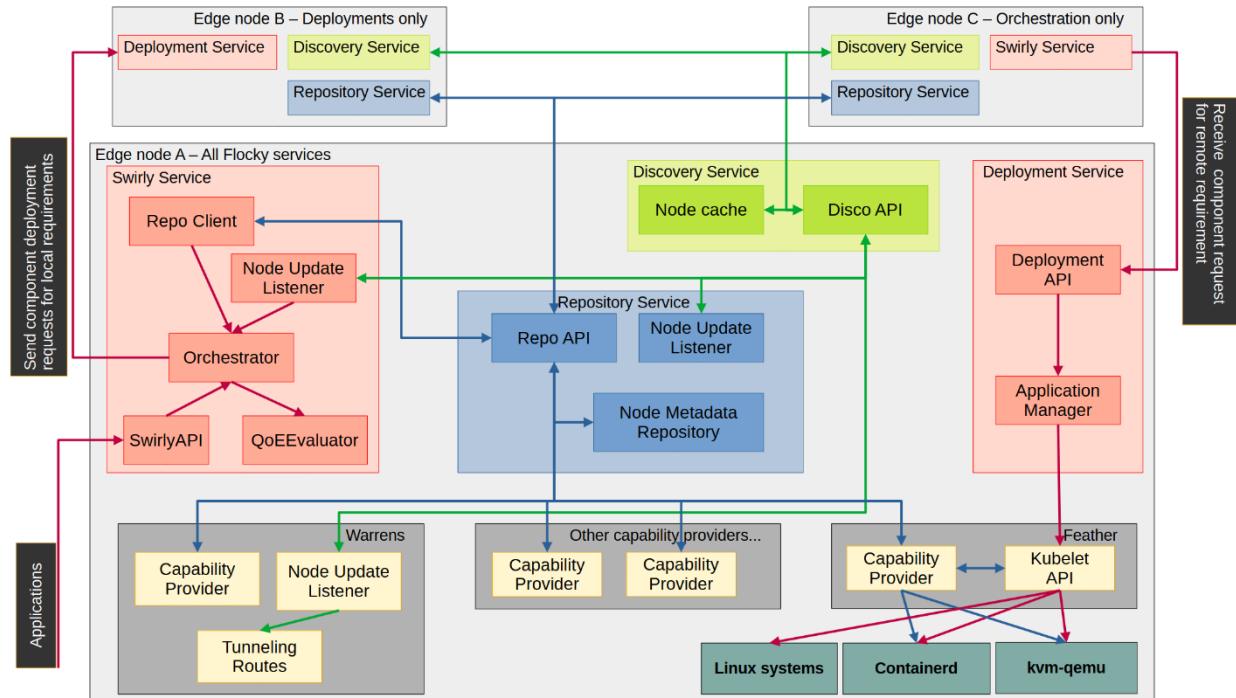


Figure 4: Overview of Flocky services, their main components, and interaction between nodes

Figure 5 shows how the Gossip and ML services have been integrated into the framework as optional services; the core (“Base”) Flocky services are only used as a repository of metadata.

- **Gossip Service:** this service was initially added to enable the creation of one or more Gossip clusters for ML model update dissemination. This service is written in Golang and uses the libopenstorage gossip library, which uses the SWIM protocol. Its main function is to transparently link gossip clusters to REST endpoints (or shared memory) exposed by applications/workloads with minimal configuration and to eliminate redundant gossip component deployments.
- **ML Service:** monitors the hardware capabilities of the system for ML-relevant hardware acceleration features, as well as enabling the deployment of edge learning workloads. Optionally, this service can set up Gossip Learning by linking ML workloads to gossip clusters by offloading configuration to the Gossip Service.
- **General workflow for Flocky extension:** new services may fulfill any number of functions, but should register themselves with the Service API if they can be called from other (third party) extension services. Any specific platform or system capabilities they

provide/describe must be reported through the CapabilityProvider API, while exposed Traits should be registered in the Traits API, which can then be used in deployment manifests and forwarded to the correct services by the Deployment service. These APIs are elaborated in 2.2.3 with examples.

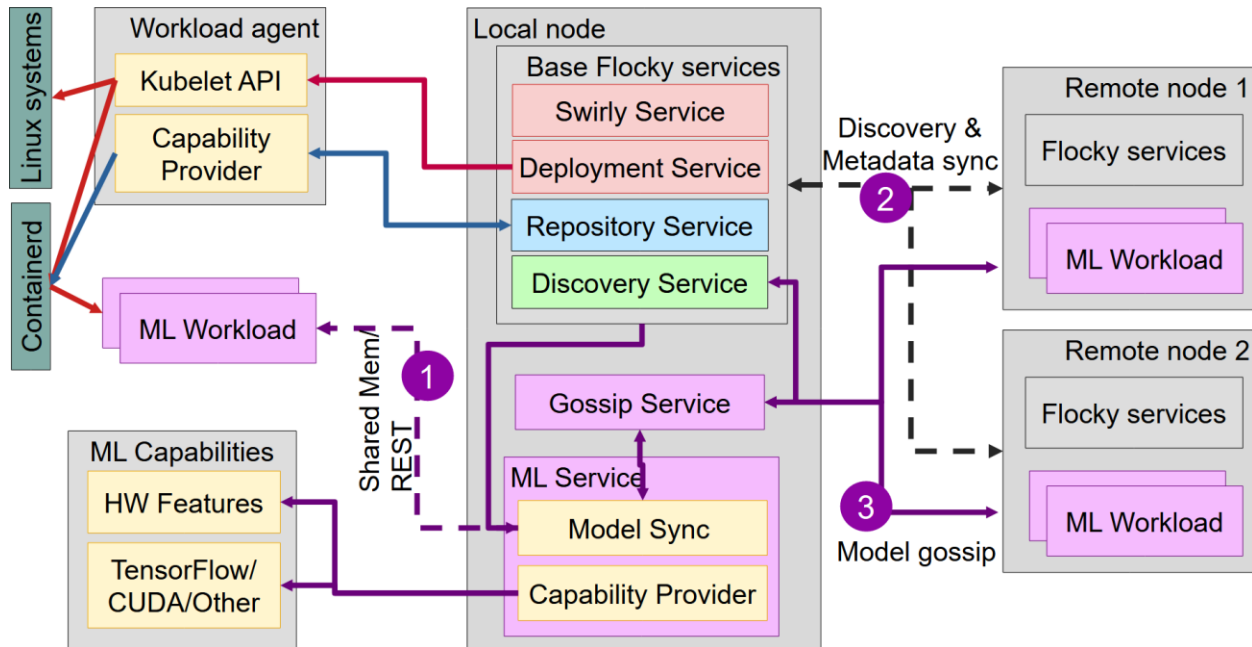


Figure 5: Updated Flocky architecture including ML/Gossip services

2.2.3. Interfaces and Protocols

The following external APIs are defined by Feather and Flocky's standard architecture. Feather offers two interfaces for workload deployment:

- Virtual Kubelet API: A REST API capable of communicating with a Kubernetes cluster; requires authentication data to join the cluster.
- Standalone API: A custom REST API which accepts Kubernetes deployment manifests to be deployed on the local node, primarily used with Flocky.

Flocky may be used to extract information or extend the framework:

- Disco API: offers pull- or subscribe-based methods of receiving node discovery updates.
- Repo API: Offers subscribe-based methods for remote node metadata updates and newly discovered workload types/definitions (i.e. new software pushed into the cluster). Also

provides methods for matching workloads with specific implementations based on requirements, and for fetching remote node metadata. Finally, it supports subscription-based methods to register Capability providers, allowing future components to provide more orchestration functionality.

- Swirly API: Hosted on each node with an orchestrator role; exposes methods exclusively for deploying an OAM application.
- Deployment API: Hosted on each worker node and contains only methods to deploy individual workloads. This API is an exception to the rule and is meant for internal use.

The following internal APIs are defined by the Flocky Repository Service for development of additional services. They are based on OAM objects (Trait, NodeCapabilities) and internal objects (Service). Concretely, OAM objects are used for all outward-facing functionality (i.e., used in application manifests), while internal objects are allowed for node-local API calls.

- Service API: (third party) services can register themselves in Flocky to make themselves available to other services. While similar to DNS, this is a separate system: the registered endpoints consist of a REST API path, including TCP port, rather than a DNS name and IP address.
- CapabilityProvider API: both core and third party services can register themselves as CapabilityProviders, providing updates in the form of NodeCapability objects on request. NodeCapabilities include system resources and applications deployed on a node by default, but may be extended. For example, the ML Service provides available GPU resources.
- Traits & TraitHandler API: (third party) services can register themselves as Trait handlers through this API, allowing the Repository Service to advertise the node as accepting such Traits in manifests, and handling them through the relevant service. For example, the Gossip Service registers itself as a TraitHandler for the discover.flocky.oam.Gossip Trait, which transparently sets up communication channels between a Gossip cluster and either read/write REST endpoints to, or a segment of shared memory with a deployed workload.

2.2.4. Data artefacts

Both Feather <https://github.com/togoetha/feather-multiruntimenetwork> and Flocky <https://github.com/idlab-discover/flocky/> offer evaluation tools in their open source repositories on Github. Feather does not have functional evaluation tools; evaluation is focused on comparative resource use between various Kubernetes-flavored agents and Feather, as well as between different runtimes. Processing speed and networking aspects may also be evaluated.

Flocky offers both functional and scalability evaluation tools. The functional evaluation tool may be started on any node, showing a human-readable translation of the contents of that node's metadata repository. Using customized (or generated) deployment manifests and well-defined hardware environments, Flocky's discovery, modeling and scheduling aspects can be validated in detail. Scalability tools are available for both core Flocky and the Gossip Learning service; generator tools can create topologies for a configurable number and density of nodes, using optional density maps to simulate real-world conditions (e.g. dense cities vs rural areas). The core Flocky generator tool can also generate generic applications, components and node properties to be assigned to/discovered by each node. Follow-up tools use these topologies and node manifests to simulate all nodes by instantiating the required Flocky services, and measuring various resource use as well as functional completeness (e.g. % of all components discovered cluster-wide over time).

2.2.5. Advancements beyond D3.1

To summarize, the ML Service and Gossip Service have been added to Flocky since D3.1. These services enable two major functions of the refactored framework:

- Exchange application data in a distributed way, within dedicated gossip clusters for each application. Nodes can subscribe to these gossip clusters and exchange data between components (e.g., linking a microservice and message queue instances across an entire edge topology).
- Expose ML capabilities to the orchestration and metadata services, which can be used to optimally deploy ML workloads on accelerated hardware. Additionally, this service supports Gossip Learning methods when combined with the Gossip Service, allowing ML workloads to leverage efficient, decentralized edge learning across an entire edge topology.

To enable these services, the Flocky framework has been refactored to better support (third party) extension services, providing some critical new APIs:

- The Service API now registers all Flocky services and their endpoints. Other services can consult the registry to find out how to address extension services.
- The Capability (Provider) API and Trait (Handler) API have been significantly reworked and formalized to allow extension services to modify specific aspects of deployments as required by Traits in deployment manifests, and to expose which non-standard capabilities (resources) and Traits are supported.

2.3. Orchestration at the CRAN

This section presents the orchestration mechanisms and architectural design developed within the scope of the NETWORK project, for managing CRAN in an O-RAN-compliant environment. While the implementation is being validated on CERTH's infrastructure using the OpenAirInterface (OAI) platform, the proposed solution is infrastructure-agnostic and can be deployed on any CRAN setup equipped with an OAI-compatible RF frontend. The framework introduces novel, AI-driven orchestration strategies that integrate network intrusion detection, dynamic resource allocation, and user management. Specifically, the solution employs AI/ML models trained on real-world datasets to classify network traffic and dynamically adjust resource allocation and user management. It identifies malicious users and triggers a Radio Resource Control (RRC) connection release to mitigate their impact on the network while prioritizing legitimate users through end-to-end slicing. Below, we analyze the components of the framework.

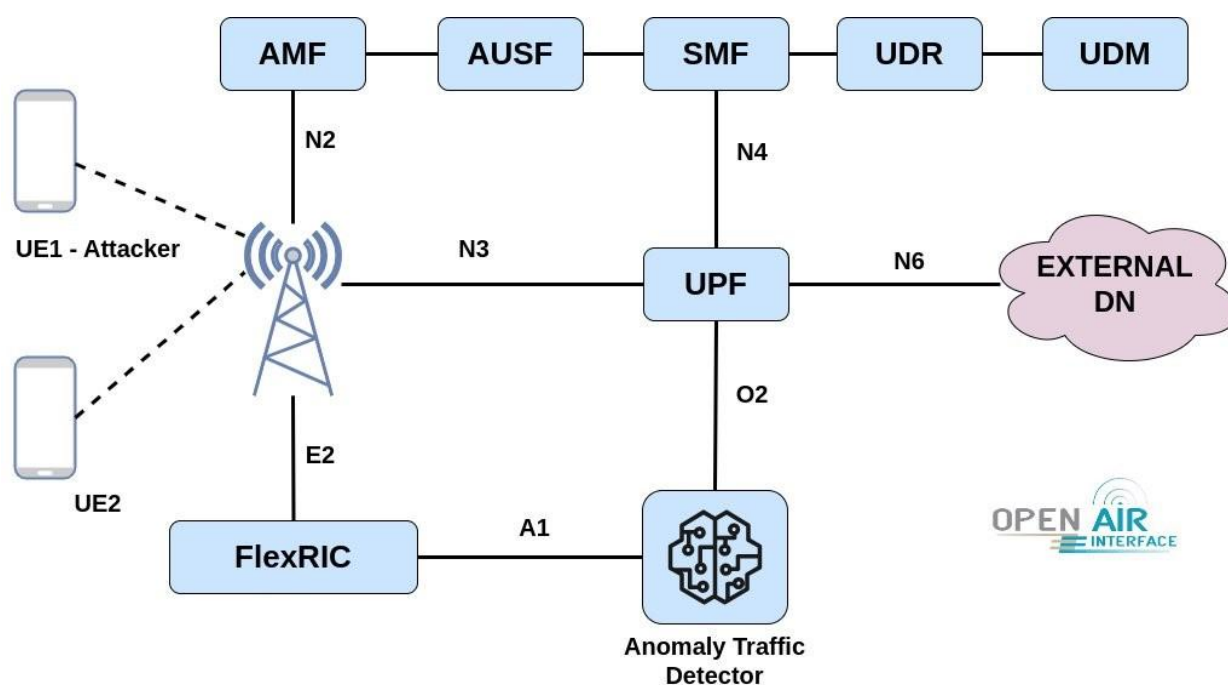


Figure 6: End-to-End Deployment of the AI-Driven Network Intrusion Detection 5G Network

2.3.1. Novelty Beyond state-of-the-art

Although recent research efforts have proposed advanced solutions for Network Intrusion Detection Systems (NIDS) and for Radio Access Network (RAN) control and slicing, these approaches are typically developed in isolation. Existing works on anomaly detection focus

primarily on traffic classification accuracy without incorporating automated network reactions to mitigate detected threats [1][2]. Conversely, RAN control and slicing mechanisms are often implemented without integrating real-time anomaly detection capabilities that can influence resource allocation decisions. As a result, current state-of-the-art solutions lack a unified framework that jointly considers security intelligence and dynamic network control in operational environments.

Furthermore, a significant portion of existing NIDS research relies on simulated environments or assumes the availability of complete feature sets during inference. Such assumptions do not accurately reflect real-world deployment constraints, where partial observability, latency requirements, and standardized interface limitations must be considered. This gap between experimental evaluation and practical deployment limits the applicability of many proposed solutions to operational 5G and beyond networks.

The CRAN orchestrator addresses these limitations by designing, implementing, and validating a real-time network intrusion detection xApp within the O-RAN architecture. Unlike prior approaches, the solution tightly integrates AI-driven anomaly detection with dynamic RAN resource management in a real-world experimental setup. The developed xApp operates within the O-RAN near-Real-Time RIC environment and combines traffic classification, user-level decision making, and radio resource control actions in a closed-loop manner.

Specifically, AI/ML models trained on real-world datasets are used to classify traffic flows and identify malicious users. Upon detection of anomalous behavior, the xApp dynamically enforces mitigation policies, including triggering RRC connection release procedures to isolate malicious users and protect network stability. At the same time, legitimate users are prioritized through adaptive end-to-end slicing mechanisms, ensuring service continuity and quality of experience.

The integration of the solution within the OAI platform, combined with the use of standardized O-RAN interfaces and Service Models provided by FlexRIC, demonstrates a fully operational implementation rather than a simulation-based proof of concept. This practical realization establishes a novel security-aware RAN control framework that simultaneously enhances network resilience, resource efficiency, and user-level service differentiation.

2.3.2. Functional components

The CRAN orchestration framework consists of several key components as illustrated in Figure 6, which consists of the experimental setup. The components are responsible for managing resources, suppressing attacks and optimizing performance. These include:

- **Orchestrator:** The central Docker/Kubernetes-based entity that coordinates real-time network intrusion detection and dynamic resource allocation based on AI/ML models. It

deploys the necessary network functions (gNB, UPF, AMF, SMF, etc.) and facilitates their communication. Core network functions communicate through a service-based architecture (SBA) using Fully Qualified Domain Name (FQDN) resolution, while communication between the Access and Mobility Management Function (AMF) and gNB occurs via Docker bridges.

- **Anomaly Traffic Detector (ATD):** This component resides near the User Plane Function (UPF) and continuously monitors GPRS Tunnelling Protocol (GTP) traffic. After collecting and processing packet data, it classifies it with a Random Forest model and computes the anomaly ratio per User Equipment (UE). This ratio is sent to the xApp via a socket-based interface, functionally representing the A1 interface.
- **O-RAN RIC (Radio Intelligent Controller):** Based on FlexRIC, it serves as the programmable control plane for the RAN. It supports Service Models (SM) such as Key Performance Monitoring (KPM) and Radio Resource Control (RC), and it manages interactions with multiple xApps.
- **xApp:** Upon receiving anomaly ratio metrics from the ATD, the xApp performs two critical actions: 1) reallocates PRBs (physical resource blocks) to prioritize legitimate users (slicing), and 2) triggers an RRC Connection Release for malicious users, effectively disconnecting them from the network. These actions are enforced through the E2 interface using the RC SM.

These components operate in a tightly integrated loop: the ATD observes traffic and sends per-UE anomaly ratios to the xApp; the xApp computes updated slicing or release decisions and applies them via the RIC to the RAN. This feedback loop enables both rapid intrusion response and optimal resource allocation.



The framework consists of interfaces and protocols to ensure seamless interoperability between components. For clarity, Figure 7 illustrates the O-RAN Architecture. The interfaces include:

- **E2 Interface:** Facilitates communication between the near-real-time RAN Intelligent Controller (RIC) and the E2 agent, which is gNodeB, allowing dynamic radio resource management.
- **O1 Interface:** Supports the exchange of management and orchestration data between the non-real-time RIC and network components. In this context, it enables the management of user traffic within the UPF. We propose that the ATD be deployed on the non-RT RIC, parsing user data through the O1 interface, as core network functions may be deployed on a Service Management and Orchestration framework.
- **A1 Interface:** Enables policy-based control from the non-real-time RIC to the near-real-time RIC for AI/ML-driven optimization. Since there is currently no open-source implementation of non-real-time RIC, a socket-based interface was defined between the ATD and the xApp for real-time anomaly classification and policy enforcement.
- **NG Interface:** Connects the 5G Core (5GC) to the gNodeB for control and user-plane traffic handling.

2.3.4. Data artefacts

- **Anomaly Packet Ratio:** Describes the percentage of anomaly packets per UE within a total packet window size N.
- **Packet Flow Data:** Traffic flows from UPFs, analyzed by the ATD to detect anomalies and enforce security policies. The traffic flows contain the following features: Protocol Type (e.g., TCP, SCTP, UDP), Service Type (e.g., HTTP, FTP, SSH), Connection Status Flag (e.g., SF for normal, REJ for rejected, RST for reset) and Source and Destination Byte Counts.
- **Resource Block Allocation Percentage:** This is the security policy/decision metric, and it determines the resource block allocation per UE after the calculation of the anomaly ratio per UE.
- **KPM Data:** Real-time performance metrics, including throughput, latency, and resource utilization, collected via Key Performance Indicator Service Model.

2.3.5. Advancements beyond D3.1

Since the submission of D3.1, significant progress has been achieved both at the algorithmic and system integration levels. The ATD component has been extended and retrained using the more recent and comprehensive CICIDS2017 public dataset [3]. This update allowed the evaluation of the detection framework under more realistic and diverse attack scenarios compared to the datasets previously considered. The use of CICIDS2017 enabled a broader coverage of contemporary intrusion patterns and improved the robustness assessment of the developed models.

Within this extended evaluation phase, four different machine learning models were systematically examined, namely Logistic Regression, Decision Tree, Random Forest, and Gradient Boosting. The comparative analysis considered classification performance alongside computational efficiency in terms of training and inference time, in order to assess suitability for real-time deployment. Among the evaluated approaches, the Decision Tree model demonstrated a particularly favorable balance between detection accuracy and computational overhead, achieving strong performance while maintaining low training complexity and fast inference execution as it is shown in Figure 8.

In parallel to the algorithmic enhancements, the CRAN orchestrator has been validated in a real 5G environment (OAI 5G network). Specifically, the CRAN orchestrator has been deployed at CERTH within the NITOS testbed infrastructure. This advancement demonstrates the feasibility of the proposed orchestration mechanisms under realistic network conditions.

Overall, the advancements beyond D3.1 include improved model generalization through updated datasets, systematic benchmarking of alternative ML models and the successful validation of the orchestration framework within a real 5G testbed environment.

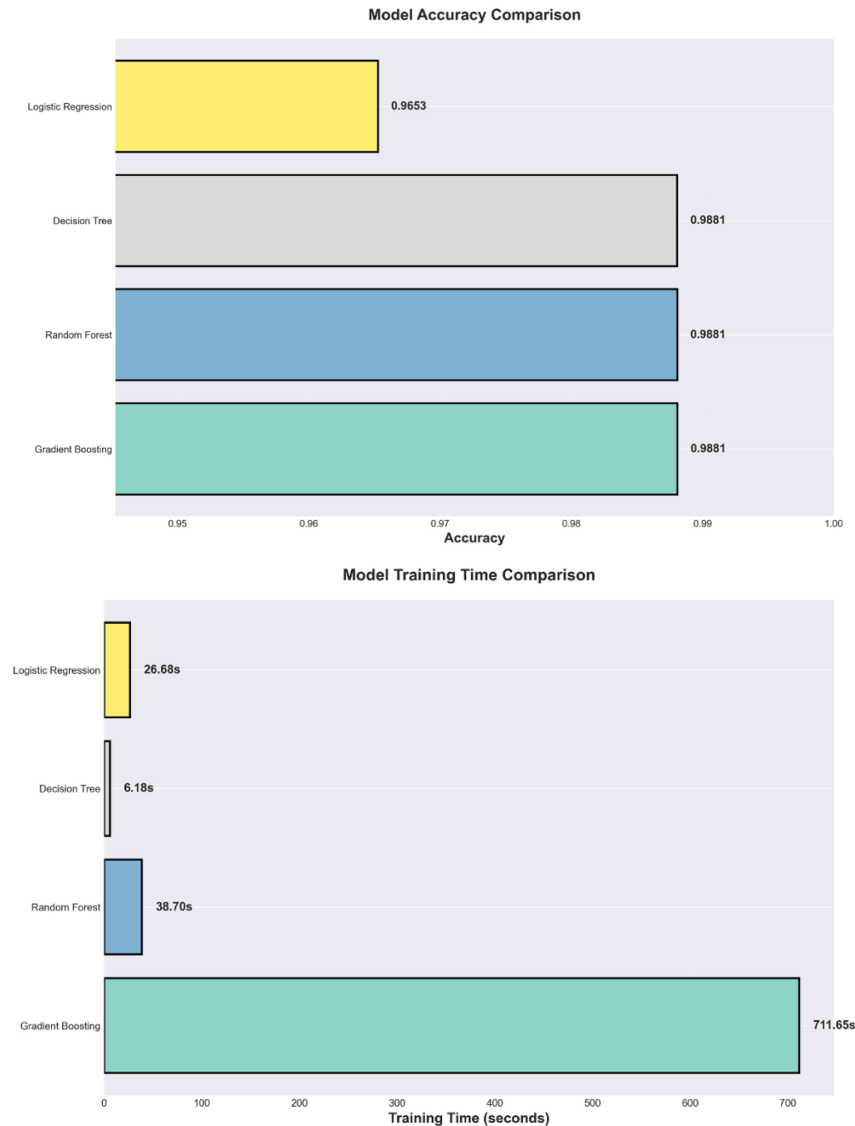


Figure 8: Training time and Model accuracy comparison

2.4. Secure-by-Design Orchestration at the Core

Orchestration at the core network level is pivotal for scalable, secure, and sustainable management of 6G slices. At the core domain of the 6G architecture, orchestration must handle high-scale, multi-tenant environments to ensure secure, efficient, and resilient management of slices and services. The first components have been designed prior to NATWORK in [4], and initial development of the FORK: A Kubernetes-Compatible Federated Orchestrator [5]. The secure-by-design orchestrator, namely secure FORK (sFORK), is under development in NATWORK and will be deployed at the core, serves as a decentralised coordination hub, managing dependency graphs, optimising resource distribution, and ensuring end-to-end security across distributed domains. sFORK will deliver novel capabilities on top of its baseline, including: implementation of global components operated by slice providers, API communication to enable interaction between different clusters and their client (i.e. slice provider), new functional components to gather information regarding the security status and requirements of slice components as well as clusters, and accordingly make distribution as well as deployment decisions. sFork integrates with a peer-to-peer Cyber Threat Intelligence (CTI) sharing solution – developed within NATWORK as an Operation Support System (OSS) - to drive cluster hygiene scores, mitigating threats like Denial of Sustainability (DoSt) attacks while aligning with Net-Zero goals. The following subsections detail the secure-by-design orchestration and the CTI solution components, focusing on their roles, functionalities, interfaces, and data artefacts.

The Secure-by-Design Orchestration framework ensures that deployment and management of network slices, cloud-native functions (CNFs), and associated services are conducted in a secure, dynamic, and sustainable manner across distributed clusters. This section will explain the functional components, interfaces and protocols, and data artefacts of the Secure-by-Design Orchestration framework.

2.4.1. Novelty Beyond state-of-the-art

Secure-by-design FORK (sFORK) is a Kubernetes-compatible federated edge-cloud orchestrator, distinguished from SotA orchestration solutions in a number of aspects:

- **Hierarchical Federation:** sFORK forms a hierarchical orchestration mesh, consisting of a set of agents, distributed across cloud clusters and interact with each other – and Kubernetes in each cluster - through well-defined APIs and descriptors, to coordinate the distribution and connectivity (i.e. stitching across a shared network) of sub-graphs of a service to form a whole service. Agents are split between: a global agent that receives requirements from different stakeholders (clouds and service providers) and pre-process them before being passed to a distribution agent that makes optimisation service distribution decisions, passed back to cluster-local agents. Each of the latter handles

scheduling of their part of the service and collaboratively they put a whole service together. To the best of our knowledge this is pioneering architecture, different from state of the art orchestrators that focus mostly on replica deployment over multiple clusters, rather than the distribution of a service.

- **Multi-Stakeholder:** sFORK, unlike existing solutions, interaction of multiple stakeholders in a 6G ecosystem. That is, to receive input of requirements and constraints from: cloud clusters (potentially different clouds), 6G slice providers, 6G slice operators and potentially even the direct underlying network; process the collective requirements of these stakeholders and accordingly make trade-off based optimised distribution decisions that attempts to maximise the satisfaction of involved stakeholders. This is a novel approach, distinguished from SotA solutions, which typically focus meeting the needs of one of the stakeholders while making assumptions about their counterparts.
- **Security vs Energy intents consideration in decisions:** sFORK is one of first orchestration solutions to incorporate security requirements (intents) and status indicators into the design of a service deployment map, positioning it as a security-by-design service orchestrator. Security requirements are expressed as high-level intents from the different stakeholders. These are then jointly considered when making orchestration decisions. Notably, the security intents of the different stakeholders along with the status indicators can run into conflicts. sFORK is able to recognise such conflicts and take them into account when make orchestration decisions.
- **Edge-cloud native architecture:** sFORK adopts the microservices architecture, featuring highly modular and self-contained controllers and operators within each agent, communicating through well-defined APIs and descriptors. Each controller is designed as lightweight component and able to exchange API messages over IPsec tunnels, established over the shared network connecting the different clusters. In addition, the distribution operator is fully self-contained and pluggable to the orchestrator, allowing modular integration of different distribution strategies and optimisation algorithms. Separately, the CNF manager is yet another modular component that can be located anywhere else in the 6G ecosystem and interact with other orchestrator components through workflow automation tools (e.g. helm charts). The high modularity of components and ability to communicate over a shared network, enable high distribution of sFORK and position it as pioneer edge-cloud native orchestrator.
- **Well-defined APIs and descriptors (data artefacts):** sFORK introduces a new set of descriptors for services, clusters and deployment requests. Distinguished from SotA descriptors, sFORK counterparts enable to provide topology-based representation of services, in the form of service graphs that show the dependencies between CNFs. They further enable to describe service intents for cybersecurity, energy and resources as well

as describe the cybersecurity posture of each CNF and the collective service. Similarly, using sFORK descriptors, cloud clusters are able to define their status indicators as well as constraints on offered services for deployment. These status indicators and intents include: security, energy and resources availability.

2.4.2. Functional components

The secure-by-Design orchestrator architecture comprises the following key functional components as shown in Figure 9:

- **Global Agent:** Acts as the central decision-maker, responsible for managing global dependency graphs, initiating and monitoring deployments, and negotiating with local orchestration agents. It evaluates cluster offerings based on a variety of factors such as hygiene, security, resource availability, and energy sustainability metrics.
- **CNF Manager:** Manages the lifecycle of Cloud-Native Functions (CNFs), including their deployment, scaling, and monitoring. It ensures that the CNFs adhere to the defined requirements and interacts with local orchestrators to execute deployments.
- **Slice Manager:** Handles the orchestration of network slices, dynamically tracks slice status, and ensures that resources are allocated efficiently to meet slice-specific requirements. It interfaces with the global agent and local orchestration agents to dynamically deploy slices based on demand and available resources and monitor them.
- **Local Orchestration Agents:** Operate within each cluster to manage the actual deployment and lifecycle of CNFs. These agents are responsible for exposing cluster capabilities, such as resource availability, hygiene scores, and compliance data, to the global agent. They execute deployment decisions and provide real-time status updates back to the global agent.
- **Dependency Operator:** Responsible for dynamically creating and maintaining global dependency graphs, mapping the relationships between microservices and CNFs across clusters. It guarantees that these dependencies are up-to-date and that the correct subgraphs are distributed across clusters based on resource availability and security criteria.
- **AI-Powered Scheduling:** Utilizes machine learning models to enhance resource allocation and scheduling decisions within clusters. By analysing patterns obtained from cluster components and predicting future demands, it provides the local orchestration agents with intelligent insights to optimize the use of available resources.
- **Cluster Requirements:** Defines and communicates the specific requirements for deploying CNFs in each cluster, ensuring that local orchestration agents are aware of the needs for resource allocation, security, and performance metrics. This confirms that deployment actions comply with cluster-specific constraints.

- **Monitoring:** Continuously tracks the health, performance, and security status of CNFs and network slices. Monitoring data is provided to both local and global orchestration agents for insights, enabling timely adjustments to have optimal performance and compliance with security policies. Prometheus integration into the orchestrator follows a modular approach, either an API call wherever online interaction is required or by accessibility to a common data point for offline interactions.

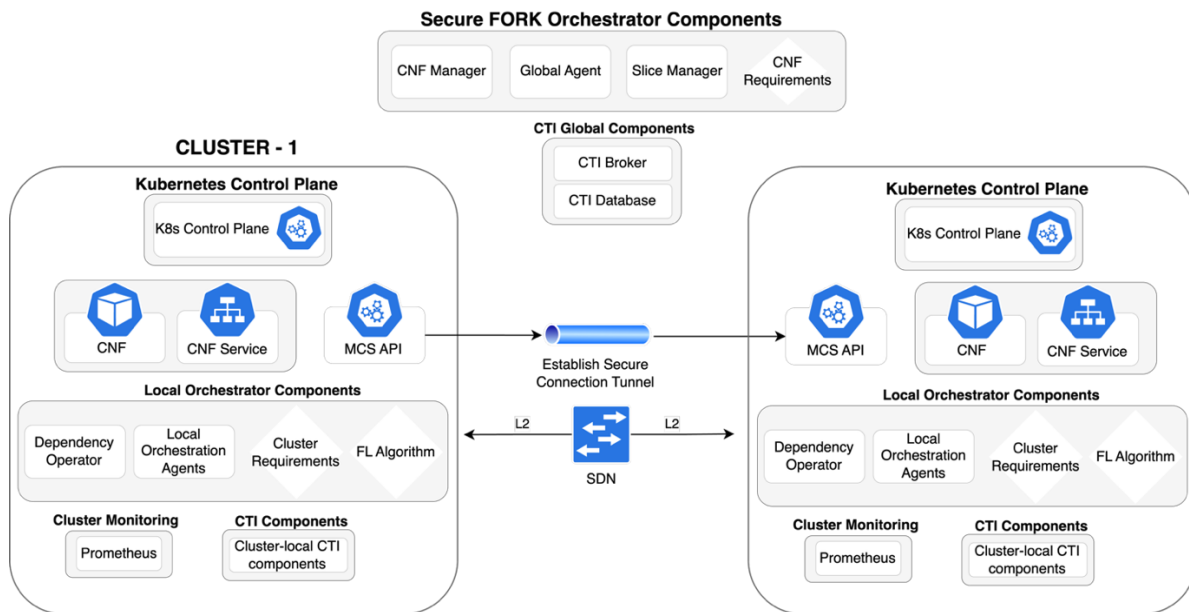


Figure 9: Secure-by-Design Orchestration Architecture and components

The components are presented in Figure 9, which demonstrates the secure-by-design orchestration framework and CTI solution. The interaction between these components enables secure 6G slice management. The FORK orchestrator's Global Agent coordinates with the Slice Manager and CNF Manager to deploy CNFs, leveraging CTI insights provided by the CTI Agent, which collects and shares vulnerability data. Local Orchestration Agents execute deployments, the Dependency Operator maps CNF relationships, the FL Algorithm extracts and processes both CTI data and monitoring/telemetry metrics, and Monitoring feeds real-time telemetry to the local orchestration agents and FL algorithm. Together, the system enables dynamic, adaptive, and secure orchestration of 6G slices.

2.4.3. Interfaces and Protocols

The orchestration framework relies on well-defined interfaces for communication:

- **Slice Management API:** It facilitates communication between the slice manager and other components, such as the orchestrator, CNF manager, and local cluster agents. It enables operations like workload migration, scaling, and reconfiguration.

- #### 2.4.4. Data artefacts

- **Global Dependency Graphs:** Graphs describing the interrelations between the CNFs in a 6G slice, including dependencies, scaling policies, and preferred cluster placements.
- **Cluster Capability Descriptors:** Structured documents (YAML/JSON) detailing available resources, security posture, energy sustainability scores, and supported CNF profiles for each cluster.
- **CNF Deployment Templates:** Secure YAML templates and Helm charts specifying how CNFs should be deployed, including security settings, resource limits, affinity rules, and upgrade strategies.

Compared to the state described in D3.1, sFORK design and development is now complete and moving to testing and validation phase. The final design and functional components are aligned with that described before and implemented PoC has now been pushed to the Network Convergence Lab (NCL) Github space (<https://github.com/NetworkConvergenceLab>), over sFORK private repository (outlined in Section 5.3). The solution and its data artefacts is now in testing and validation stage to prepare:

- a scientific publication submission,
- a demonstration, intended for EUCNC 2026
- Pilot validation for UC1.1

3. Data Plane Computation Offloading Design

The final design developed in Task 3.2 addresses this challenge by extending and refining the offloading security functions directly into programmable data plane elements, transforming network devices into active, intelligent enforcement points. Security services and network functions are implemented as programmable pipelines on P4 switches, SmartNICs, DPUs, FPGA-based accelerators, RAN traffic balancers.

3.1. Offloading functions in the data plane

As network security threats continue to evolve, traditional security mechanisms that rely on centralized processing in software-defined infrastructures or in dedicated management-based collectors and platforms are becoming insufficient. To address this challenge, the activities carried out in Task 3.2 focus on the architectural design and implementation of security services, microservices, and network functions as fully programmable data plane pipelines. By leveraging the capabilities of data plane programmability for different types and variants of backends and in different domains of the 6G architecture, the proposed approach aims to offload security functionalities directly into network devices, enabling high-performance, low-latency threat detection, efficient mitigation mechanisms and overall improved security.

A key aspect of this approach is the development of Decentralized Feature Extraction (DFE) for AI-based security functions. This allows for the real-time analysis of network traffic at the device level, enabling advanced security features such as AI-driven traffic pattern prediction, anomaly detection, and federated learning-based threat mitigation. The goal is to reinforce security mechanisms within 6G networks by embedding intelligence directly into programmable network elements, preventing attacks from propagating beyond the data plane.

To achieve real-time security enforcement, the activity also focuses on the design and implementation of Wirespeed AI (WAI) models. These AI-driven security mechanisms will be optimized for execution on programmable hardware, such as SmartNICs and FPGA-based accelerators, ensuring that security functions operate at full line rate without introducing latency overhead. By embedding AI models into network processing units, the system will be capable of dynamically identifying and mitigating security threats as they emerge.

Additionally, the implementation of security pipelines leveraging hardware acceleration enhances the efficiency of network threat detection and response. Such pipelines are designed to utilize high-performance computing resources within network infrastructure, enabling inline security processing that adapts to diverse attack vectors in real time.

To facilitate AI-driven security enforcement, the task will also involve the development of data plane code generators for AI training and feature telemetry. This includes leveraging tools such as P4RROT to enable automated feature extraction and telemetry collection, supporting continuous learning and refinement of security models. By integrating AI-powered feature collection mechanisms directly into the data plane, the system will be able to adapt security policies dynamically based on evolving network conditions.

The main objective of this initiative is to identify and block Advanced Persistent Threats (APTs) at the data plane level, before they reach cloud-based AI collectors or centralized security management systems. The key challenge lies in designing a programmable security framework that can dynamically adapt to network anomalies and heterogeneous attack events in real time. The proposed approach will ensure that security enforcement mechanisms remain proactive and responsive, leveraging AI and high-performance networking technologies to protect 6G networks from emerging cyber threats. By integrating AI-driven security functionalities directly within programmable data plane elements, this initiative will enable a highly efficient, autonomous, and adaptive security architecture, ensuring that threats are mitigated at the earliest possible stage, without impacting overall network performance.

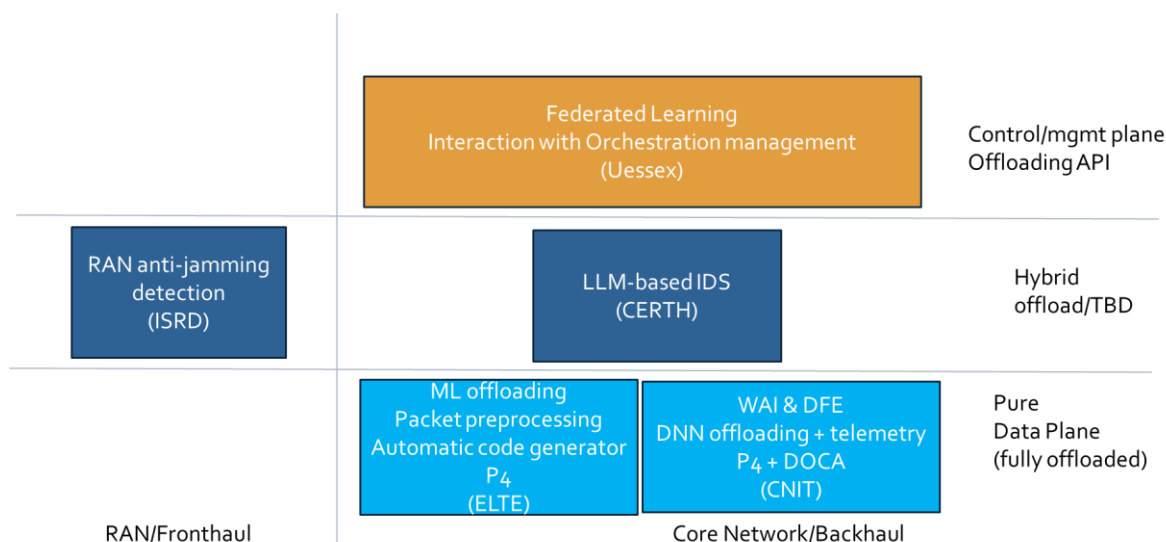


Figure 10: NETWORK Data Plane offloading solutions and flavours

3.1.1. NETWORK Offloading flavours

The chart shown in Figure 10 shows the different NETWORK solutions proposed in the context of Task 3.2 for the data plane offloading of network functions focused on cybersecurity. Each solution has been placed in the graph based on two different classifications: 1) the 6G

domain/segment where the solution is deployed and applied, and 2) the type of offloading implemented.

About the 6G segments, most of the envisioned solutions are conceived to be run in the backhauling or within the Core Network functionalities. These solutions mainly rely on the wirespeed analysis of packets in the proximity of the UPF – outside or inside, depending on the considered programmability stack (i.e., whether GTP tunnelling is considered or not). One solution, conversely, is based on the processing of the radio signal in the RAN gNB segment, targeting anti-jamming detection.

About the offloading flavour, the classification has been performed based on the degree of offloading, as follows:

1. Control/management plane offloading API: such solutions enable communication between orchestrators/controllers to the involved data plane devices to discover, configure, activate and dynamically tune the behavior of offloaded network functions. Such APIs are designed and developed in strict collaboration with Task 3.1 dedicated to security orchestrators.
2. Hybrid offload: the offloaded function may reside partially in the data plane as pure pipeline or into dedicated control plane backends. As an example, the LLM-based intrusion detection system (IDS) is partially deployed in external GPUs, not directly involved in the data plane pipelines. In addition, the anti-jamming detection is based on specific xAPPs retrieved by the Near-real time RIC.
3. Pure data plane offloading: Full offloading refers to the embedding of security network functions implemented as pipelines or chain of pipelines inside data plane backends. This includes either software-based containers running accelerated pipelines (e.g., eBPF, XDP, DPDK) or hardware-based backends such as SmartNIC, programmable switches, or FPGA. In this case, network functions requiring the adoption of AI are designed to run AI tasks inside the backend. If AI engines are not available (e.g., programmable switches do not onboard GPUs), a transformation of the pipeline is implemented to embed the logic of the selected AI algorithm. WAI and ML offloading solutions are envisioned and presented for different backends.

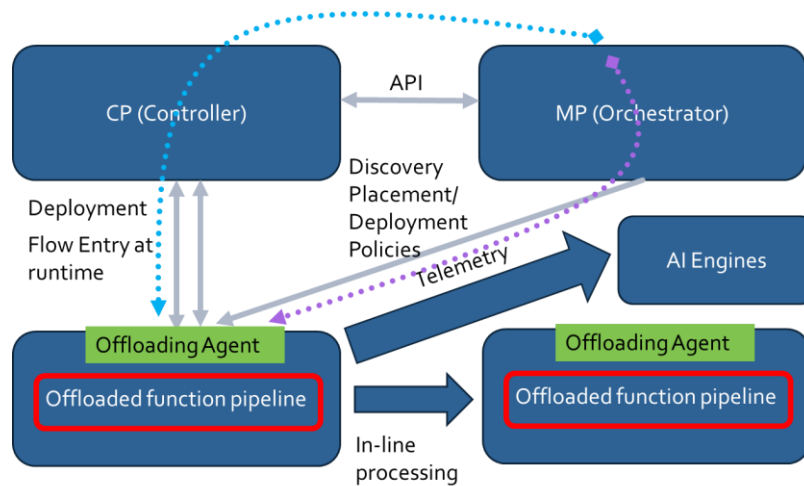


Figure 11: Offloading deployment and runtime configuration options

3.1.2. Deployment and configuration interfaces

A hybrid architecture (Figure 11) is adopted to support scalable, flexible, and dynamic deployment of security functions in 6G networks. It combines SDN-like control plane (CP) mechanisms with cloud-native management plane (MP) orchestration. This hybrid architecture ensures that security services can be efficiently deployed and orchestrated based on their specific operational requirements—whether they need to be tightly integrated within the network infrastructure or implemented as cloud-based applications.

The CP mechanism (shown as a purple line) follows an SDN-like paradigm, where security functions are implemented as programmable pipelines that operate at the data plane level. In this model, a centralized P4-based controller is responsible for deploying the security pipeline and configuring flow rules dynamically. This approach is particularly well-suited for network-centric backends, where real-time traffic enforcement, advanced packet processing, and network telemetry provide continuous security enforcement. Regarding its implementation, the control plane leverages P4 backends to dynamically manage security policies, optimize traffic routing, and enforce fine-grained access control mechanisms. A centralized SDN controller orchestrates updates and reconfigurations, ensuring adaptive threat response.

The advantages of this mechanism are the following:

1. Low-latency packet processing directly at the network level.
2. Enhanced control over security rules via SDN programmability.
3. Real-time traffic analysis and mitigation.

This approach is particularly relevant for network-oriented research groups and institutions, such as CNIT and ELTE, which focus on SDN-based network programmability and secure traffic management.

The Management Plane (MP) – Cloud-Native Approach adopts a cloud-native methodology, where security functions and applications are deployed as containerized workloads running within environments such as Kubernetes pods or Docker containers. This approach is well-suited for function/app backends, where security services need to scale elastically, integrate with cloud-based AI engines, and interact with software-driven network environments. Security services are designed as microservices, packaged into Kubernetes or Docker containers, and orchestrated dynamically based on demand. This allows seamless integration with cloud-native AI models, data processing pipelines, and federated security mechanisms. The advantages are the following:

1. High scalability and elasticity for security functions.
2. Integration with cloud AI-based anomaly detection and mitigation.
3. Simplified deployment and management through Kubernetes orchestration.

This model is best suited for function-oriented security applications developed by ISRD and CERTH, focusing on AI-enhanced security mechanisms, cloud-native microservices, and distributed threat intelligence frameworks.

In the following sections, we provide an introduction and design details for the different NATWORK data plane offloading solutions.

3.2. Wirespeed AI (WAI) and Decentralized Feature Extraction (DFE)

Enabling ML-driven functions in network devices remains challenging due to the distributed and non-linear computations required by Deep Neural Networks (DNNs). Unlike decision trees or support vector machines, DNNs demand specialized processing capabilities. We outline four architectural approaches to integrating DNNs within network functions, as shown in Figure 12:

- External DNN Processing: The switch/NIC matches and forwards selected packets to an external device (e.g., FPGA) for DNN inference. While feasible, this approach introduces delays and power inefficiencies due to inter-device communication.
- Feature Extraction at the Switch/NIC: To optimize processing, the switch/NIC extracts ML-relevant features in real time, reducing the computational load on an external device (e.g., GPU). However, packets still require buffering until inference is completed.

- Integrated GPU within the Switch/NIC: Emerging hardware integrates a dedicated GPU alongside programmable ASICs, enhancing processing speed and minimizing inter-device communication overhead. However, GPUs remain energy-intensive.
- Fully Offloaded DNN Processing: The proposed approach offloads the entire ML pipeline to a programmable switch/NIC, leveraging match-action tables for in-network inference. This eliminates external dependencies, reduces buffering needs, and ensures low-latency processing at wire speed. Furthermore, integrating the ML model directly into network hardware improves energy efficiency compared to GPU-based setups.

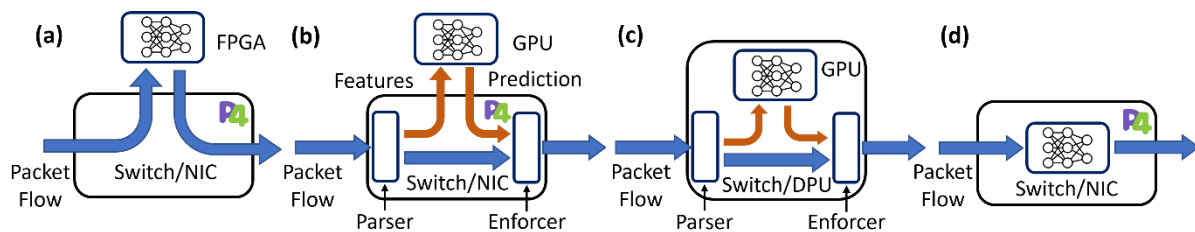


Figure 12: Embedding AI/ML inside programmable switches

The Decentralized Feature Extraction (DFE) and the Wirespeed AI (WAI) have been designed to cover all these approaches. In particular, the DFE operates with all the approaches to extract the desired features from packet trains and flows received and processed by the network element. Depending on the selected approach, the DFE extracts the relevant features and, if needed, provides feature telemetry to external consumers (i.e., (a) and (b) in the figure) or to internal devices (i.e., (c)). Alternatively, it may act as the first pipeline stages of a fully data plane embedded solution including WAI (i.e., (d)). This is the most interesting and challenging case, described in Figure 13.

Packets are received by ingress interfaces, parsed and DFE-analyzed. The DFE pipeline stage is in charge of extracting the selected features used internally to feed WAI. The figure also shows the possibility of exporting such extracted features as a telemetry stream to feed external collectors and consumers. This last design is implemented and evaluated as an additional component in Task 4.3.

Stateful memory is exploited to store and update stateful features (i.e., features related to the history of a session/flow/connection or aggregated information averaged in time windows). Then, a specific pipeline stage is dedicated to onboard WAI. WAI implements the input-output logic of a ML model without necessarily reproducing the full ML structure. Depending on the design, the model may be hardcoded in the WAI or configured as a list of control-plane flow

entries. In the last case, this WAI can be dynamic, i.e., it can be configured at runtime to change the ML network function. Flow rules, policy rules, and enforcement rules are configurable.

The WAI stage returns the output of the considered ML model, while the next WAI enforcer pipe maps the WAI output to a list of actions. As an example, if the classification output of the DDoS mitigator WAI stage is Boolean (i.e., “attack”, “non-attack”), the enforcer may implement a block/discard action on attack-tagged packets, or a forward port action towards a firewall analyser.

This functional design may be applied to several backends: programmable switches, smart NICs, white boxes, software-based switches. In the following, we report the activities involved in implementing WAI and DFE on a programmable switch and on a NVIDIA Bluefield-2 DPU.

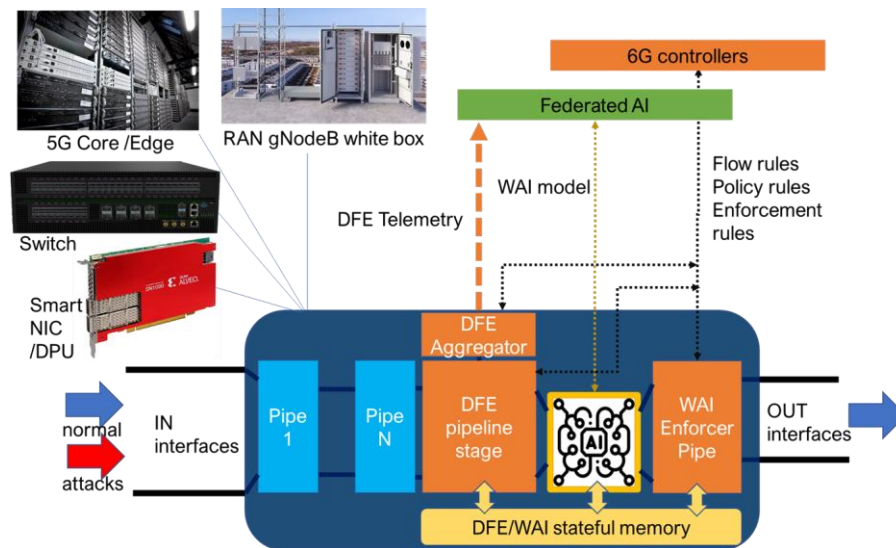


Figure 13: DFE and WAI design submodules

3.2.1. Novelty Beyond state-of-the-art

The proposed solution advances beyond the state of the art by introducing a fully in-data-plane, wire-speed feature extraction architecture that tightly integrates programmable DPU capabilities with performance-aware refinements. Unlike existing approaches that rely on host-based monitoring or hybrid software-hardware pipelines, the DFE component operates entirely within the DPU data plane through optimized DOCA Flow primitives, eliminating CPU involvement and avoiding costly memory transfers between host and accelerator. This architecture enables true per-flow feature extraction at line rate, leveraging hardware-maintained state tables to continuously observe TCP connection attempts and derive lightweight, attack-relevant indicators such as SYN rates, connection state transitions, and fine-grained flow statistics. By extracting features directly from hardware state without packet redirection to software paths, the system

removes traditional bottlenecks associated with interrupt handling, kernel networking stacks, and user-space analytics engines. Furthermore, the design introduces performance-oriented refinements that guarantee strict isolation between monitoring and forwarding functions. Non-suspicious traffic is processed entirely in hardware and forwarded at full bandwidth with zero additional latency, while only suspicious flows trigger enhanced inspection mechanisms. This selective, hardware-enforced processing model enables sustained operation at tens of millions of packets per second, even under adversarial conditions, without degrading quality of service for legitimate users.

About WAI in P4 switches, the design advances by explicitly embedding a DNN inside a P4 switch using the LUT distillation method and addressing an often-overlooked consequence of deploying neural inference directly in the data plane: the vulnerability to adversarial examples. While LUT distillation makes it feasible to execute DNN decisions as match-action lookups, it also creates a high-value target where even subtle, protocol-compliant header perturbations can induce misclassification. The final design contributes three defense designs that remain compatible with match-action pipelines and therefore preserve the “inline” nature of detection. A further novelty is turning an implementation constraint into a security lever: because LUT distillation already relies on integer/quantized inputs, we systematically exploit quantization and simple data-plane primitives (lookup, range matching, XOR/CRC, logical aggregation) to harden the classifier under multiple attacker knowledge assumptions (white-, gray-, and black-box).

3.2.2. DFE/WAI in P4 programmable switches

Deploying Deep Neural Networks (DNNs) inside programmable data planes poses a significant challenge due to the limited arithmetic and memory capabilities of switching hardware. To overcome these limitations, we propose a method to distil a trained, integer-quantized DNN into a series of lookup tables (LUTs), enabling fast, predictable inference through P4 match-action logic. This approach allows us to embed complex decision logic directly into a switch pipeline, supporting low-latency ML-based packet processing without external acceleration [10].

The core idea of our method is to reduce the inference phase of an integer-quantized DNN to a deterministic match-action operation. The procedure begins with a fully trained, quantized DNN. Given that input features and network weights are represented as integers, every possible input combination can be mapped to a corresponding DNN output. This exhaustive mapping enables the DNN to be transformed into a static LUT: each entry matches a specific combination of inputs and stores the associated output. For an input vector composed of features encoded in n and m bits, the LUT requires 2^{n+m} entries. While this process is lossless in terms of model accuracy, the exponential growth in memory with the number of input bits quickly becomes a bottleneck. To address this, we adopt a hierarchical cascaded design.

Instead of implementing a monolithic LUT, we construct a network of 2-input DNNs, each distilled into smaller LUTs. Input features are grouped in pairs and passed through their respective 2-input models. The outputs are then recursively paired and processed by higher-level models, ultimately producing a final output. This layered design drastically reduces the memory footprint: for instance, a 4-input DNN (8-bit features, 1 output) would require a monolithic 4 GB LUT; with the cascaded method, three 64 KB LUTs suffice (192 KB total).

Our implementation targets a P4_16 programmable switch (e.g., Barefoot Tofino). The solution is composed of two primary components:

- P4 Parser and Stateful Stage: Extracts ML-relevant features from incoming packets (DFE).
- P4 Pipeline: Executes the cascaded LUT inference using match-action tables (WAI).

The parser is configured to support common networking headers and extract integer-encoded features for DNN inference. It includes stages for parsing Ethernet, IPv4, TCP/UDP, and optionally application-specific headers (e.g., GTP for mobile core networks). Each parsed field is stored in metadata registers accessible by the pipeline.

Figure 14 illustrates the parser design. It begins with the parse_ethernet state, followed by parse_ipv4, and then TCP/UDP stages where transport-level features (e.g., TCP window size, UDP length) are extracted. These become the input features for the cascaded LUTs.

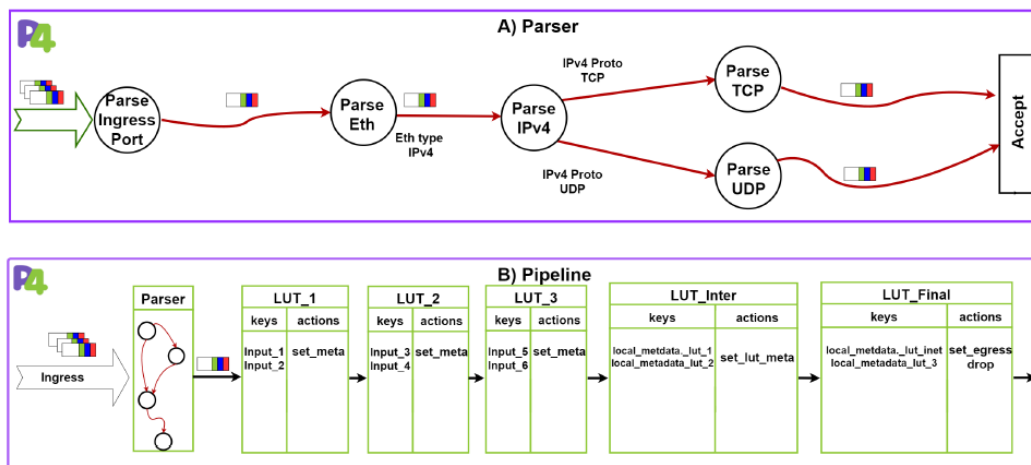


Figure 14: P4 Parser and pipeline for a 6-feature input

The pipeline structure directly mirrors the cascaded LUT architecture. Consider a DNN with 6 input features and 1 output, realized as a cascade of five 2-input distilled LUTs (Figure 15A). The P4 pipeline is accordingly composed of five tables (Figure 15B):

- LUT_1, LUT_2, LUT_3: First layer tables, each taking two input features and producing intermediate metadata.
- LUT_Inter: Second layer table, processing the outputs of the first layer.
- LUT_Final: Last-stage table, producing the final classification and applying the action (e.g., forward, drop).

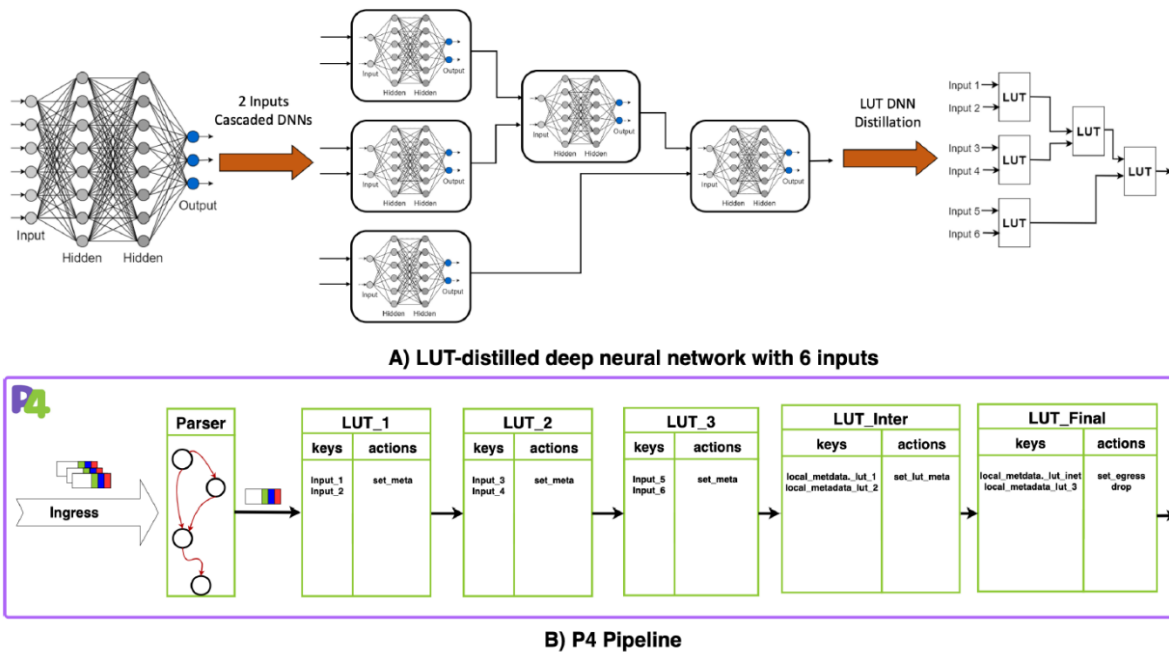


Figure 15: LUT distilled DNN and its P4 pipeline

Each table performs an exact match on either packet features or intermediate results stored in *local_metadata*. The action for each match is to set metadata fields using the *set_meta* or *set_lut_meta* actions. In the final stage, *set_egress* or *drop* actions are triggered based on the classification output.

This modular design can easily be extended to support different ML models and additional features. By updating the parser, new fields can be extracted without altering the pipeline logic. Similarly, updating the LUTs enables rapid deployment of retrained models, making this approach suitable for dynamic environments such as intrusion detection or anomaly classification in IoT.

While the cascaded LUT method enables fast and deterministic DNN inference, several trade-offs must be considered:

- Memory Scaling: Each additional input feature increases the LUT depth logarithmically but not exponentially. Still, total memory usage must be budgeted carefully.

- Integer Encoding: Input features must be quantized and encoded as integers; floating-point inputs are not supported.
- Training Overhead: Cascading introduces training complexity, as multiple small models must be trained and distilled. However, inference latency remains constant—one table lookup per layer.

Despite this, the system allows for real-time, inline ML inference in switching hardware, a major milestone for in-network computing.

The algorithms of the offloaded P4 DNN in P4 switches are described in Section 6.5.1. All the details of the design, the implementation and the results are reported in a journal publication [10] with the NATWORK ack.

3.2.2.1. *Final design (robustness to adversarial attacks)*

The final WAI component consists of an in-switch classifier (“Attack Detector”) that operates on a small set of TCP header fields and produces a binary decision (benign vs. attack). The internal P4 pipeline is shown in Figure 16. At packet arrival, the data plane extracts four features—TCP flags, data offset, sequence number, and acknowledgment number—and encodes them into a compact representation suitable for table keys. Exact matching is used for fields with limited cardinality (e.g., flags and data offset), while broader numerical fields (sequence/ack) are discretized through range matching to map them into fixed-width values. Inference is implemented via LUT distillation with a hierarchical structure. Instead of a single monolithic lookup, the DNN is represented as a cascade of three tables: two first-stage LUTs each process two features and output an intermediate value, and a third LUT consumes the two intermediate outputs to emit the final binary decision. This decomposition reduces the key width per stage and keeps the overall pipeline aligned with standard match-action processing. On top of this baseline detector, the system defines three alternative defense mechanisms that can be deployed without changing the fundamental in-switch processing model: (i) reduced-precision inference, (ii) an adversarially trained auxiliary detector running in parallel, and (iii) embedded input randomization implemented with additional lookup tables. The implemented WAI tools are described in Section 5.4. The algorithms are described in Section 6.5.1. The results are reported in the conference work in [11] with the NATWORK ack.

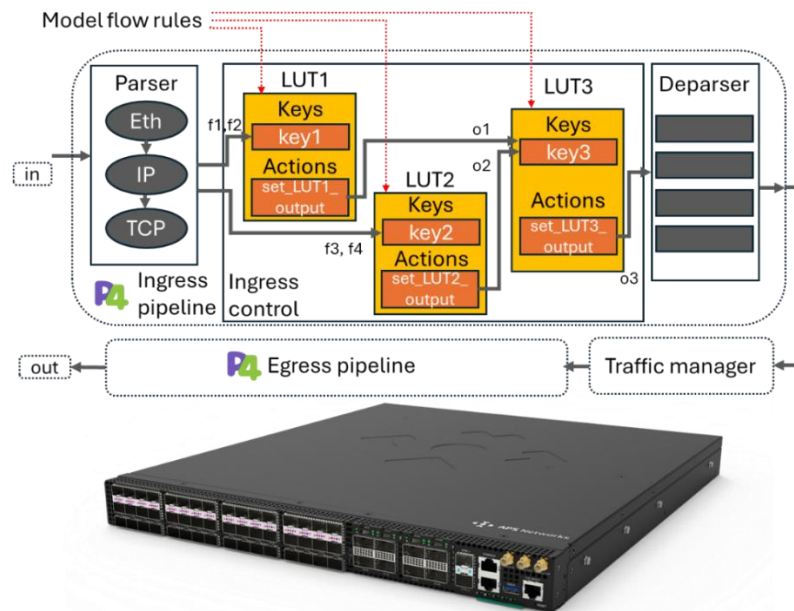


Figure 16 WAI final design – A single P4 for DDoS and DDoS adversarial attacks

3.2.3. DFE/WAI in NVIDIA Bluefield-2 DPU

Distributed Denial-of-Service (DDoS) attacks remain a significant cybersecurity threat, disrupting legitimate access to services. Among various attack strategies, the TCP SYN flood attack is particularly effective, overwhelming target servers by exploiting the TCP handshake mechanism. Traditional mitigation techniques, such as rule-based filtering and machine learning-based approaches, often introduce high latency and fail to respond effectively to large-scale attacks. To address these challenges, this offloading activity proposes a novel DDoS mitigation system leveraging programmable Data Processing Units (DPUs) to offload attack detection and mitigation processes from the host system. By utilizing hardware acceleration and intelligent flow-based filtering, real-time attack prevention is achieved while maintaining high network performance.

The proposed mitigation system is built on the architecture of programmable DPUs, specifically leveraging the DOCA Flow framework to define hardware-accelerated packet processing pipelines. The system is implemented on a SmartNIC equipped with a multi-core ARM CPU and programmable packet processing pipelines. The architecture consists of several processing stages that work together to detect and mitigate attacks before they impact the host system.

At the core of the system is a structured sequence of processing pipelines that efficiently classify, filter, and handle network traffic. Packets enter the DPU through either the physical network port or the host interface and are first processed by the root pipe, which identifies and filters non-

IPv4 traffic. Once packets pass this initial stage, they are examined by a blacklist pipe that instantly drops traffic originating from previously identified malicious sources. The control pipe plays a crucial role in dynamically classifying packets based on TCP flags and forwarding them for further analysis.

The DPU architecture, shown in Figure 17, efficiently intercepts ingress traffic from both network and host, processes it, and forwards it to the egress port. Packets enter through either P0 (network-facing Physical Port) or pf0hpf (host-facing Physical Function). These interfaces connect to hardware-offloaded Open-Virtual-Switch (OVS) bridges using Traffic Control (TC) or Data Plane Development Kit (DPDK) optimizations to prevent software switch bottlenecks. The bridges link P0 and pf0hpf to the DPU-internal Scalable Functions, SF1 and SF2, which are lightweight functions deployed on a parent PCIe function. They access the parent's capabilities and resources while maintaining dedicated queues (txq, rxq), allowing multiple services to run concurrently.

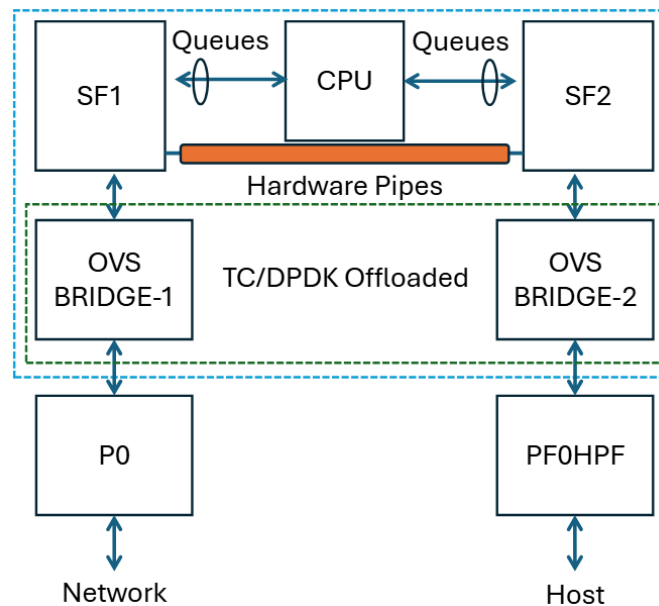


Figure 17: DPU architecture

Packets are then processed through programmable hardware pipes defined via DOCA Flow APIs. These pipes determine packet handling—whether to drop, modify, forward to the CPU, or steer directly to another SF, bypassing the CPU. The DDoS mitigation application runs on the DPU CPU and directly interfaces with SF1 and SF2. During environment initialization, hardware pipes are created to direct packets to the SF queues when processing is needed. A Receive Side Scaling (RSS) mechanism distributes traffic across queues, and the application retrieves packets using DPDK APIs in polling mode, parsing and forwarding them accordingly. Assigning one ARM core per tx-rx queue pair is recommended to avoid race conditions and performance issues.

For SYN flood detection, the system incorporates two specialized pipelines: the SYN pipe and the SYN miss pipe. The SYN pipe monitors incoming SYN packets and applies metadata-based tracking to detect suspicious patterns, leveraging hardware counters to track per-flow statistics. If an IP address shows an abnormal SYN-to-TCP ratio, it is flagged as a potential attacker and blacklisted. The SYN miss pipe is responsible for handling previously unseen IPs, ensuring that all new sources are added to the monitoring system for further evaluation.

To ensure high performance and scalability, the architecture is designed to minimize CPU intervention. This is an essential design requirement that avoids the implementation of a low-performance offloading service. Hardware pipes manage most packet classification and filtering tasks, while only a limited number of packets require host processing. The system dynamically updates flow records, blacklists, and counter thresholds to adapt to evolving attack patterns in real time. By leveraging the DPU’s built-in hardware acceleration, the mitigation system is capable of handling high-speed network traffic at line rate while maintaining low latency and high efficiency. The DFE software tools are described in Section 5.4. The details of the offloaded DDoS mitigator in DPU are described in Section 6.5.2. All the details of the design, the implementation and the results are reported in [12].

The final proposed design is shown in Figure 18 and adopts a hybrid offloading approach, where packet-level processing is primarily executed on the DPU data plane, while higher-layer analysis and control actions are delegated to software components running on the host. In particular, L3/L4 traffic monitoring, feature extraction, and mitigation actions are offloaded to the DPU using DOCA Flow and hardware-accelerated switching, enabling wire-speed operation without involving host CPU resources in the steady state. Host-side components are activated only when application-layer analysis is required or when control decisions (e.g., blocking requests) must be communicated to the DPU. This separation allows the system to combine high performance with functional flexibility, while keeping the offloaded logic limited to operations that benefit most from hardware acceleration.

From a network deployment perspective, the solution is applicable to scenarios where the DPU is positioned in-line between an external or untrusted network segment and a protected host or service. Typical examples include data center edge servers, cloud ingress nodes, and virtualized network functions exposed to public traffic. The L3/L4 defense operates transparently on all

incoming traffic and does not require changes to the hosted applications, making it suitable for general-purpose deployments. The optional host-side L7 component can be placed on the same server or on a remote system, as long as it can communicate block requests to the DPU, allowing the solution to be adapted to different network topologies and operational constraints.

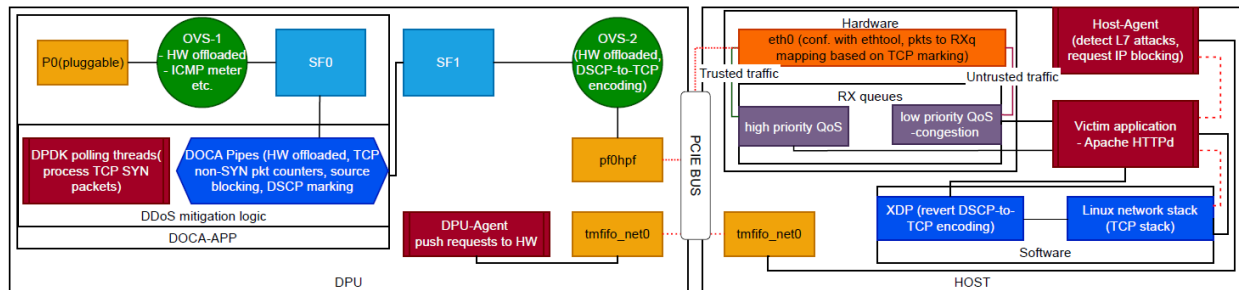


Figure 18 DPU Wire-Speed Feature Extraction Final Schema of the software and hardware entities involved in Attack Mitigation

The L7 defense integration is conceptually simple shown on bottom part of a specialized application runs on the host server and recognizes the behavior of clients by inspecting application specific metrics and activities, and, whenever it notices anomalous activities, it sends a request via a UDP packet to the DPU agent to request the block the identified source IP. UDP was chosen because of its simplicity, since it allows us to easily handle block requests from a virtually unlimited number of host agents. Indeed, application layer attacks change as the L7 protocol changes or, even, depending on the application version (e.g., attacks based on CVE might harm some application's versions but not others), so it might be easier to deploy multiple independent host agents, each specialized for a different attack category. This final overall design is currently submitted as journal version to Computer Networks.

3.2.4. Advancements beyond D3.1

The final design presented in this section extends the capabilities described in D3.1 by introducing several architectural and functional improvements to both the DFE and WAI components. For the DFE, the system evolves into a fully wire-speed feature extraction solution implemented directly in the DPU data plane through DOCA Flow primitives. This refinement enables per-flow monitoring and extraction of attack-relevant TCP features—such as SYN rates, connection states, and flow statistics—directly from hardware-maintained state tables, eliminating host CPU involvement in the steady state. Furthermore, the design introduces host-based communication agent to mitigate L7 attacks. Host-side components are activated only for higher-layer analysis or control actions, enabling flexible integration of application-layer defenses.

For the WAI component, the advancements focus on improving the robustness and deployability of in-switch machine learning inference. While the previous design demonstrated the feasibility of LUT-distilled neural network inference within match-action pipelines, the final solution

addresses the vulnerability of such classifiers to adversarial attacks, trying to attack the AI model. The design introduces three complementary defense mechanisms—reduced-precision inference, adversarially trained auxiliary detection, and input randomization—implemented using data-plane-compatible primitives such as lookup tables, range matching, and simple logical operations. These mechanisms leverage the quantized nature of LUT distillation to enhance resilience against adversarial manipulation while preserving line-rate processing and compatibility with programmable switch architectures. In addition, the classifier implementation is refined through a hierarchical LUT structure that decomposes the neural inference process across multiple tables, reducing key width and maintaining alignment with standard match-action pipeline constraints. Overall, the advancements beyond D3.1 improve scalability, robustness against adversarial attacks, and practical deployability across programmable data plane platforms.

3.3. Data plane Machine Learning

Running Machine Learning (ML) efficiently on programmable hardware and software targets is challenging due to limited arithmetic capabilities, restricted memory, and the strict per-packet processing model. Building on these constraints, we propose an ML-enabled data plane architecture designed for real-time and adaptive attack detection.

3.3.1. Novelty Beyond state-of-the-art

The proposed ML-enabled data plane architecture advances the state-of-the-art in three major ways:

Fully online learning in the control plane triggered by in-network feedback: Unlike prior ML-in-data-plane approaches that rely on pre-trained models or periodically retrained offline datasets, our system performs continuous online learning. Features extracted at line rate are sampled and streamed to the control plane, where new random forest models are trained, validated, and deployed on demand. This eliminates cold-start phases and enables rapid adaptation to emerging attack types.

As-soon-as-possible inference for early packet-level decisions: Instead of waiting for full-flow aggregation, the data plane performs incremental inference after each packet is received. Separate models are trained and encoded for different packet counts (e.g., 1st, 2nd, ... packet), enabling early detection of high-speed attacks and reducing average classification latency.

Hardware-aware model encoding and disaggregated deployment: The architecture supports heterogeneous P4 targets—ranging from CPU-based eBPF switches to ASIC-based Intel Tofino—by applying custom decision tree encodings. A disaggregation mechanism allows distributing model components across multiple switches while preserving classification correctness and minimizing per-switch resource usage.

The design ensures that no single component must carry the full ML workload, allowing switches to continue performing forwarding and other essential functions without exceeding hardware resource budgets (SRAM/TCAM).

3.3.2. Offloading mechanisms and network segment applicability

The proposed architecture supports three complementary offloading mechanisms within the data plane, categorizing the computational tasks based on their role in the security pipeline:

Stateful Feature Extraction Offloading: This mechanism enables persistent per-flow feature computation—such as packet length statistics, TCP flag counters, and port-related information—directly inside the switch pipeline. By utilizing register-based storage, the system avoids the latency of control-plane polling for basic telemetry.

Inference Offloading: This process compiles the decision logic of Random Forest classifiers into match-action tables tailored to hardware constraints. On eBPF targets, the model is implemented through depth-wise table chaining; on Intel Tofino hardware, each decision tree is embedded into a single table using range-matching capabilities.

Partial Learning Offloading: While the heavy training process remains in the control plane, this mechanism delegates flow sampling and feature collection to the data plane. This enables continuous online learning and model refinement with minimal communication overhead.

This ML-enabled data plane pipeline is designed for flexible deployment across different network segments:

- **At the Network Edge:** Switches perform immediate feature extraction and early inference for newly observed flows, ensuring timely detection and data collection at the point of entry.
- **In Aggregation or Core Switches:** The architecture supports spreading decision-tree components across several hops (model disaggregation). This minimizes the ML resource footprint on individual devices while maintaining high classification accuracy across the network fabric.

3.3.3. Functional components

Machine learning in the programmable data plane presents unique challenges due to the architectural constraints of programmable network switches. Tofino, which can be programmed in the P4 (Programming Protocol-independent Packet Processors) language, is designed for high-speed packet processing with stage-based pipelines but lacks the general-purpose computational capabilities required for complex ML tasks. Unlike CPUs and GPUs featuring dedicated tensor cores and floating-point processing units, Tofino’s architecture prioritizes efficiency in packet

forwarding over extensive computation. As a result, executing ML directly on the switch is highly constrained by the available processing power.

Another major challenge is the limited memory available within Tofino. ML models typically require significant storage for parameters, feature representations, and intermediate computations. However, Tofino primarily provides SRAM and TCAM memory, which are designed for fast packet classification rather than storing large ML models. Additionally, the switch lacks native support for floating-point arithmetic, making it difficult to implement models that rely on high-precision numerical operations.

Feature extraction, which is a critical step in many ML applications, is also difficult to implement efficiently in the data plane. Traditional ML models process features derived from entire packet flows, whereas Tofino operates at the per-packet level, making it challenging to aggregate information across multiple packets. Moreover, P4 lacks support for iterative computations, such as loops or complex mathematical operations, which are commonly used in preprocessing and normalization steps. Consequently, implementing ML-based anomaly detection, traffic classification, or congestion control within the switch requires significant workarounds to extract meaningful features from packets in real time.

To overcome these challenges, we propose a ML model as illustrated in Figure 19. The model uses a federated learning-based approach for deploying ML in the data plane across multiple network slices. Slices A and B are two distinct network slices, each with its own ML oracle, controller, and switches. The key components of this approach are:

1. Oracles (IDS): These serve as ground truth sources for training the ML models and monitoring their accuracy, typically implemented via an external Intrusion Detection System.
2. Controllers: Each network slice has a controller that receives a small percentage of flows from the switches. The controller updates the ML model if accuracy drops and offloads computational work from the switches.
3. Federated Learning via a Coordinator: Instead of training ML models individually on each switch, the Coordinator aggregates models from both network slices through secure data aggregation, producing an improved global ML model. This updated model is then sent back to the controllers for deployment.
4. Switches: Each switch is responsible for processing packets using a lightweight ML model piece. Only a small fraction of flows is sent to the controller to improve the model without overloading switch resources.

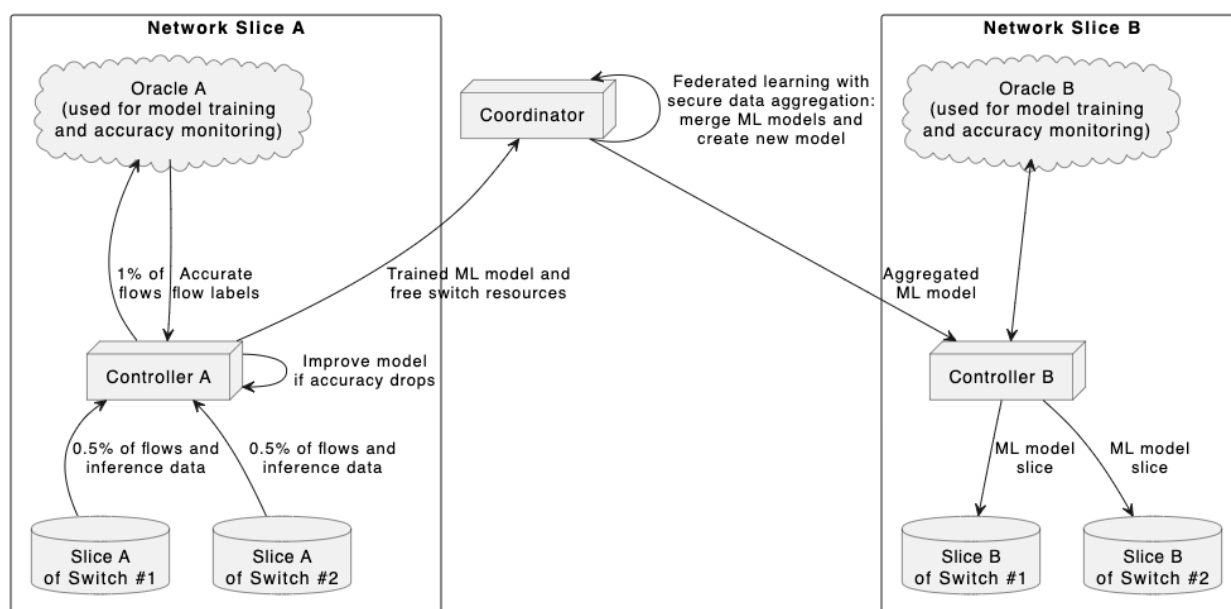


Figure 19: Data plane ML model

While the architectural limitations pose significant challenges for direct ML deployment in the data plane, a federated learning approach with coordinated control-plane support offers a practical and scalable solution for enabling intelligent network functions. For implementation details and access to the codebase, refer to Section 5.7.

3.3.4. Advancements beyond D3.1

As detailed in this section, the current architecture builds upon the foundations laid in D3.1 by introducing some technical additions that enhance the original data plane ML concept. A major addition is the "as-soon-as-possible" inference strategy, which enables incremental classification based on early packet counts—such as the first or second packet—to reduce detection latency significantly. The framework now also supports heterogeneous targets, extending functionality beyond Intel Tofino ASICs to include eBPF-based CPU switches through specialized model encoding techniques like depth-wise table chaining. Furthermore, the architecture provides a more granular definition of offloading mechanisms, specifically categorizing tasks into stateful feature extraction, inference, and partial learning offloading.

3.4. RAN security-performance balancer

This section presents the RAN security-performance balancer as a novel, O-RAN-compliant solution for managing heterogeneous cryptographic processing within the RAN. Building on recent advances in AI/ML-driven RAN optimization, the proposed approach introduces crypto-algorithm awareness, hardware-aware resource mapping, and predictive provisioning of CU-UP

resources based on traffic and ciphering characteristics. It leverages standard 3GPP security negotiation procedures and integrates with the Near-RT RIC control loop to enable algorithm-specific traffic steering and efficient utilization of heterogeneous compute capabilities. The section further details the functional components of the Load Balancer xApp and outlines the key advancements beyond D3.1, emphasizing a practical, standards-aligned implementation within existing O-RAN architectures. Additional details on the service implementation and the underlying algorithms are provided in Sections 5.5 and 6.7, respectively.

3.4.1. Novelty Beyond state-of-the-art

Recent research on AI/ML in O-RAN has predominantly focused on radio-centric optimization problems such as load balancing across cells and RUs, slice-level QoS prediction, or UE association control using bandit learning [13], GNNs, DRL, or federated meta-learning techniques (e.g., multi-agent load balancing in O-RAN and slice QoS optimization frameworks) [14]. These works demonstrate the effectiveness of ML for predicting PRB utilization, congestion states, or traffic steering decisions within the RAN domain [15][16]. However, they do not consider the heterogeneity of cryptographic processing load introduced by different 3GPP ciphering algorithms (e.g., 128-NEA1/SNOW 3G vs 128-NEA2/AES) nor the implications of hardware acceleration availability on CU-UP resource allocation.

The proposed solution advances beyond this SOTA in three key dimensions. First, it introduces **crypto-algorithm awareness as a first-class load balancing dimension**. Unlike prior ML-based RAN optimization that treats user traffic as homogeneous, this approach exploits the outcome of the Access Stratum security negotiation (RRC Security Mode Command) at the CU-CP to classify traffic by ciphering algorithm before user-plane data flows. This enables algorithm-specific steering to dedicated CU-UP pools. No surveyed O-RAN ML work explicitly leverages the negotiated ciphering algorithm as a resource steering parameter.

Second, the solution performs **hardware-aware mapping of predicted traffic to heterogeneous crypto resources**. Existing studies optimize radio or slice resources without modeling per-algorithm computational cost. In contrast, the proposed framework integrates algorithm-dependent cycles-per-byte models and hardware acceleration capabilities (e.g., AES-NI, QAT, SmartNIC) into the control loop. The Near-RT RIC thus allocates AES-dominant traffic to acceleration-enabled pools while isolating SNOW traffic to CPU-optimized pools. This transforms the problem from generic load balancing to heterogeneous compute orchestration, a dimension absent in current O-RAN ML literature.

Third, the framework incorporates **predictive, risk-aware provisioning of algorithm-specific CU-UP pools**. Rather than reactive scaling based solely on instantaneous load, the gradient boost-based predictor forecasts both total traffic and ciphering algorithm mix, generating estimates to

satisfy SLA targets. Prior works in O-RAN primarily apply ML for reactive optimization (e.g., PRB reallocation or UE association), without coupling probabilistic forecasts to proactive compute pool scaling. By integrating uncertainty-aware forecasts with hysteresis-controlled scaling policies, the proposed architecture reduces SLA violations and improves resource efficiency under time-varying AES/SNOW ratios.

In summary, while existing SOTA demonstrates ML-driven radio and slice optimization in O-RAN, it does not address security-processing heterogeneity. The proposed solution fills this gap by combining crypto-algorithm awareness, hardware-aware compute mapping, and predictive risk-based provisioning within a standards-compliant Near-RT RIC control loop.

3.4.2. Functional components

The proposed Load Balancer xApp consists of the following main functional components deployed within an O-RAN-compliant architecture, shown in Figure 20. First, a KPI Collection and Feature Builder in the Near-RT RIC subscribes via E2 (e.g., E2SM-KPM) to per-algorithm traffic counters, CU-UP CPU metrics, RAT distribution indicators, and session statistics. It aggregates these inputs into time-windowed feature vectors (e.g., lagged AES/SNOW ratios, total throughput, mobility indicators) suitable for ML inference. Second, a gradient boost-based Prediction component performs multi-horizon forecasting of total traffic and ciphering algorithm mix (AES vs SNOW), optionally producing probabilistic outputs to support risk-aware decisions. Third, a Crypto-Aware Capacity Controller component of the Load Balancer xApp converts predicted per-algorithm throughput into required compute resources using calibrated cycles-per-byte models and knowledge of available hardware acceleration. It determines CU-UP pool sizing and generates steering policies that bind UEs to algorithm-specific CU-UP instances based on the ciphering outcome negotiated at CU-CP. Finally, a Model Training and Policy Governance function hosted in the Non-RT RIC /SMO or offline performs periodic offline training, model validation, lifecycle management, and provides A1 policy guidance (e.g., risk levels, scaling limits, guardrails). Together, these components form a closed-loop, predictive, and hardware-aware control system that optimizes security-processing capacity in real time while remaining aligned with O-RAN interfaces and operational constraints.

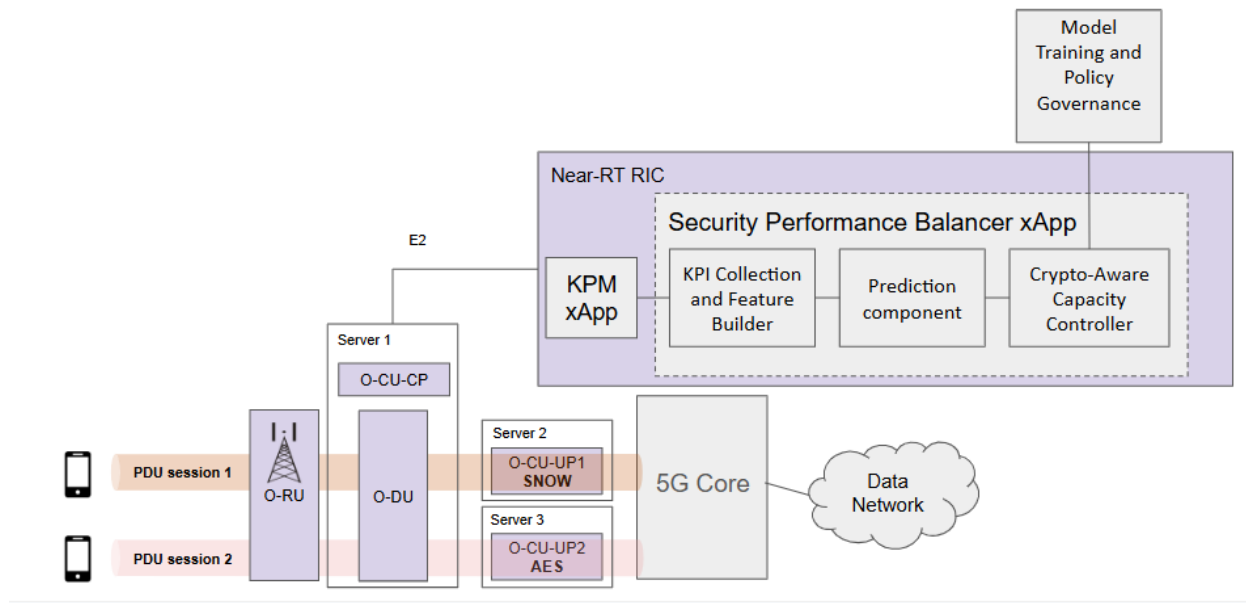


Figure 20: Security-performance balancer architecture

3.4.3. Advancements beyond D3.1

In this report we present a new concept for the Security Load Balancer driven by more practical approach to O-RAN compliant implementation. The RAN security-performance balancer described in D3.1 primarily focuses on dynamic adaptation of security mechanisms versus performance constraints, through computation offloading, orchestration, or runtime adjustment of security functions within a shared infrastructure. Its emphasis is on balancing security strength and performance overhead by relocating or scaling security processing components, at a more abstract orchestration level. In contrast, the proposed Security Load Balancer introduces a more practical and implementation-oriented deviation: instead of dynamically changing security levels or offloading encryption functions, it leverages the already standardized 3GPP security negotiation at CU-CP (RRC Security Mode Command) and deterministically separates user-plane traffic based on the negotiated ciphering algorithm into dedicated CU-UP instances. This approach avoids modifications to security procedures or algorithm selection logic and does not require complex offloading frameworks. It operates fully within existing E1AP signaling and CU-CP/CU-UP split mechanisms making it directly deployable in current O-RAN architectures. Practically, this reduces integration complexity, eliminates the need for cross-domain orchestration changes, and enables immediate exploitation of hardware crypto acceleration (e.g., AES-NI or dedicated crypto engines) by binding homogeneous algorithm traffic to tuned CU-UP pools. Therefore, while D3.1 addresses security-performance trade-offs through architectural flexibility and offloading concepts, the current solution provides a lightweight,

standards-aligned, and operationally feasible mechanism that can be incrementally introduced into commercial deployments with minimal architectural disruption.

3.5. LLM-based IDS

3.5.1. Novelty Beyond state-of-the-art

The proposed LLM-based IDS introduces a transformer-driven approach for traffic flow representation learning that advances beyond conventional network intrusion detection mechanisms. Traditional IDS solutions typically rely either on signature-based detection or on supervised machine learning models trained on handcrafted statistical flow features [6][7][8]. Such approaches often require labeled datasets and suffer limited generalization across heterogeneous network environments.

In contrast, the proposed solution leverages a transformer encoder architecture inspired by BERT [9] and applies self-supervised pretraining on unlabeled network traffic. Through contrastive learning objectives, the model learns to identify structural similarities and dissimilarities between traffic flows without requiring attack annotations. This enables the extraction of generalized and transferable traffic representations that are not tightly coupled to a specific dataset or network topology.

A key innovation of the approach lies in the packet-level tokenization process, where individual packets within a flow are transformed into structured tokens analogous to words in natural language processing. This design preserves temporal ordering and fine-grained packet characteristics instead of compressing flows into aggregated statistical summaries. Furthermore, to prevent label leakage and infrastructure-specific memorization, sensitive identifiers such as IP addresses and ports are abstracted through a directional binary flag, ensuring that the model focuses on behavioral patterns rather than static identifiers.

The resulting architecture provides a unified framework capable of supporting supervised classification, semi-supervised learning, and unsupervised anomaly detection using the same learned flow embeddings. This combination of self-supervised transformer-based sequence modeling at packet granularity constitutes a clear advancement over existing ML-based and statistical IDS approaches.

3.5.2. Offloading mechanisms and network segment applicability

The LLM-based Intrusion Detection System (LLM-IDS) has been designed with a flexible deployment model that supports distributed execution across the cloud–edge continuum. Given the computational complexity of transformer-based architectures, especially during training and

representation learning phases, the system distinguishes between resource-intensive operations and latency-critical inference tasks in order to enable efficient offloading strategies.

The self-supervised pretraining phase, which involves large-scale traffic datasets and contrastive learning objectives, is executed in centralized cloud infrastructures equipped with GPU acceleration. This stage requires substantial computational resources and benefits from elastic scaling capabilities available in cloud environments.

For operational deployment, inference can be offloaded closer to the network edge to satisfy real-time detection requirements. Once pretrained, the LLM encoder can operate in a lightweight inference mode where flow embeddings are generated and classified with reduced computational overhead. This enables deployment in edge nodes such as Multi-access Edge Computing (MEC) platforms or 5G gNB edge servers. By performing detection near the traffic source, the system reduces response latency and avoids unnecessary backhaul traffic toward centralized security platforms.

3.5.3. Functional components

To extract useful features for intrusion detection from raw packet sequences, CERTH has developed an LLM-based IDS based on the BERT transformer encoder architecture. Our model has been pretrained on unlabeled data traffic using self-supervised training methods including contrastive learning. Through our pretraining procedure our model learns to recognize similar and dissimilar flows enabling generalizable intrusion detection across diverse traffic domains. The inference pipeline is presented in Figure 21.

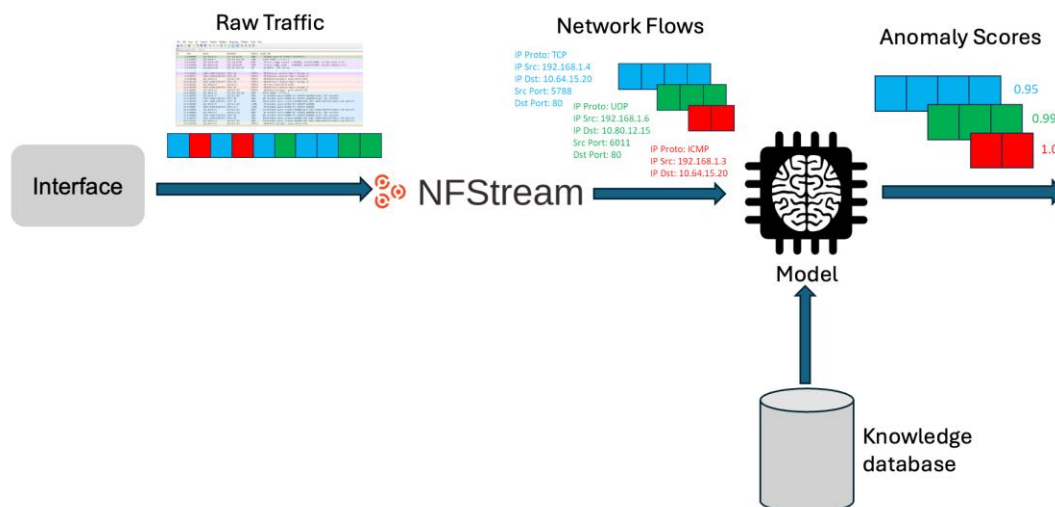


Figure 21: LLM-IDS Inference Pipeline

In Figure 22, we show an overview of the architecture and pipeline for the traffic data processing. The architecture of the LLM-based IDS is organized into a modular pipeline consisting of traffic acquisition, feature extraction, tokenization, sequence encoding, and classification. Thus, the LLM-based IDS consists of the following functional modules:

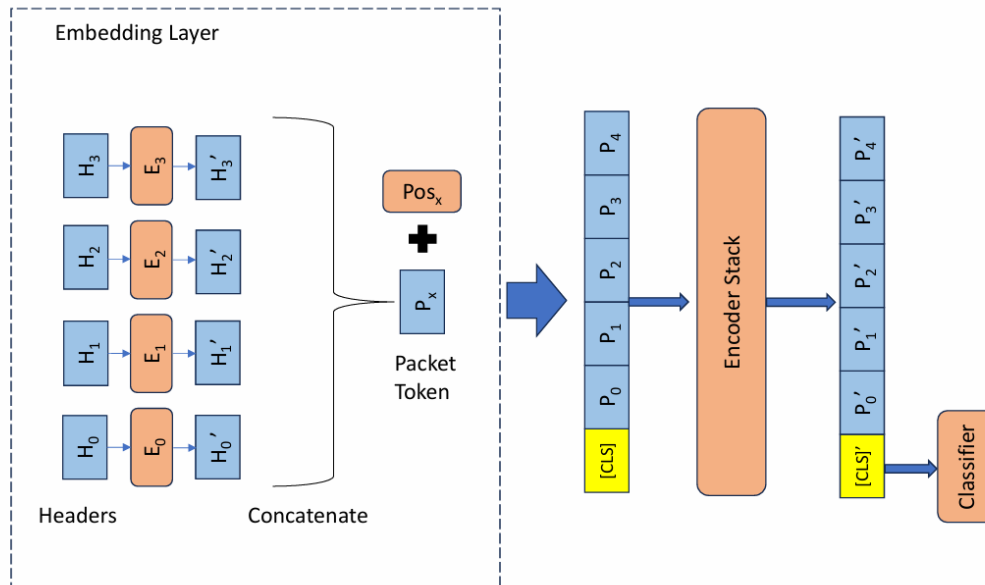


Figure 22: LLM-Based IDS Overview

Traffic Capture Module

The traffic capture module collects raw packets and groups them into flows based on the standard 5-tuple definition comprising IP protocol, source and destination IP addresses, and source and destination ports. Each flow is then represented as an ordered sequence of packets.

Packet Feature Extraction Module

For each packet within a flow, relevant header fields are extracted, including packet length and TCP flags. Additional metadata is derived, such as a timestamp relative to the beginning of the flow. To avoid the inclusion of potentially misleading identity information, IP addresses and port numbers are not directly embedded; instead, a custom binary directional flag is introduced to indicate packet direction within the flow.

Tokenization and Embedding Layer

A custom embedding layer transforms the raw packet features into dense vector representations. The individual feature embeddings are concatenated and projected through a

linear transformation to form a single packet token. Positional embeddings are added to encode the sequential order of packets within the flow.

Transformer Encoder Stack

The core of the system consists of a four-layer transformer encoder stack with four attention heads per layer and an embedding dimension of 256. A special classification token is appended at the beginning of each sequence, serving as the aggregated representation of the entire flow. During pretraining, the transformer learns to encode the behavioral characteristics of the packet sequence into this classification token representation.

Classification/Detection Module

For detection, the output corresponding to the classification token is passed to a classifier module. This module may consist of a multilayer perceptron trained jointly during supervised fine-tuning, or alternatively, the embeddings may be exported and used by conventional classifiers such as logistic regression or random forests. The same embeddings can also support unsupervised anomaly detection by identifying flows that significantly deviate from learned representations of benign traffic.

3.5.4. Advancements beyond D3.1

Since the submission of D3.1, additional experimentation and refinements have been conducted to further investigate potential improvements to the LLM-based IDS training pipeline. In particular, we explored the incorporation of Masked Language Modelling (MLM), a pretraining strategy commonly used in BERT-based architectures, as a complementary self-supervised learning objective alongside the contrastive learning framework already employed in our approach.

The motivation behind this investigation was to examine whether MLM could enhance the quality of the learned packet-flow representations by encouraging the model to reconstruct partially masked packet tokens within a sequence. In natural language processing tasks, MLM has been shown to improve contextual understanding by forcing the model to infer missing tokens based on surrounding context. Therefore, it was considered a potential augmentation technique that might further strengthen the semantic representations of traffic flows before applying contrastive learning.

To evaluate this hypothesis, MLM was introduced as an additional pretraining step prior to the contrastive learning phase. The model was trained to predict masked packet tokens within flow sequences using the same packet tokenization scheme described in the previous sections. The

resulting representations were then used as initialization for the contrastive training stage and subsequently evaluated in the intrusion detection task.

However, experimental evaluation showed that the inclusion of MLM did not lead to measurable improvements in detection performance or representation quality when combined with the existing contrastive learning framework. Furthermore, the additional training stage increased the computational complexity and overall training time without providing a clear benefit in terms of classification accuracy or generalization.

Based on these findings, the MLM pretraining step was not included in the final training pipeline of the LLM-based IDS. The results suggest that the contrastive learning objective already provides sufficiently strong representation learning for the packet-flow sequences considered in NATWORK.

4. Software Design: Orchestration Support Systems

This section presents the NATWORK support services related to security orchestration and data plane offloading, highlighting key mechanisms such as Moving Target Defense, selective CTI sharing, AI-based behavioral analysis, and security-performance balancing to enhance orchestration resilience and adaptability.

4.1. Moving Target Defense (MTD) Framework

The proposed MTD Framework is responsible for enhancing the security level of network functions across the edge-to-cloud spectrum. It enforces both pro-active and reactive actions, mainly entailing the re-instantiation and live migration of VNFs and CNFs, which could be either a live (*i.e.*, stateful) migration or stateless migration depending on the current state of the environment and the target resources. These operations can also be performed in an inter-slice manner, allowing a VNF/CNF to be moved not only to different domains, *e.g.* from the core node to an edge node, but also to a different network slice whenever necessary. During such operations, advanced forensic analysis can be performed by duplicating the transferred checkpoint image for a static image security scan or for running an isolated deployment in a sandboxed environment for analysis. In addition, IP shuffling and port shuffling are also provided as MTD actions to further strengthen the security of NFV-based Telco Cloud networks.

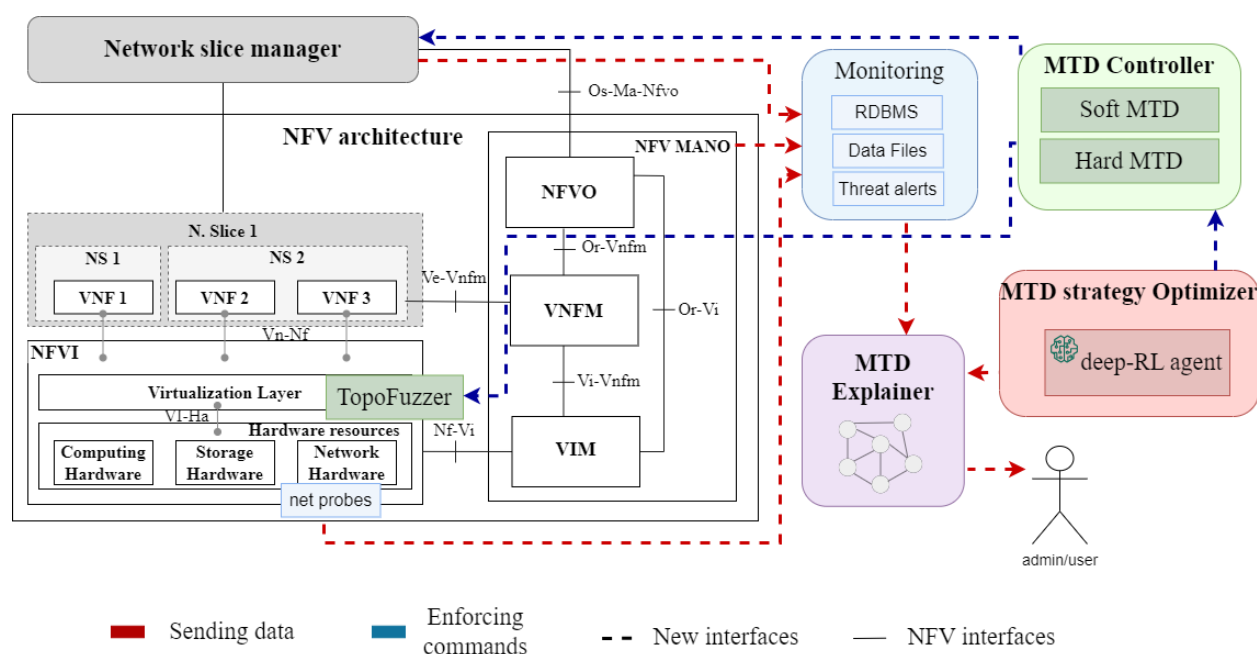


Figure 23: Architecture of the MTD framework

The proposed framework consists of three main components, as shown in Figure 23:

- **MTD Controller** is the enforcer component that executes the MTD actions proposed by the MTD Strategy Optimizer, via direct interaction with the NFV MANO and the Network Slice Manager, supporting the orchestration of the resources on the infrastructure.
- **MTD Strategy Optimizer** is mainly responsible for determining the details of the MTD actions such as the optimal frequency (for pro-active cases) or the necessary trigger (for reactive cases), along with the internal mechanisms to utilize (e.g. live migration).
- **MTD Explainer** helps the service owners to gain insight on performed MTD actions, by generating a human-interpretable explanation of why such action was necessary to be conducted.

This deliverable focuses on the first component, the MTD Controller, described in the following section, while its algorithms are further described in Section 6.4.

4.1.1. Novelty Beyond State-of-the-Art

The main novelty of the proposed MTD Framework lies in enabling stateful live migration of CNFs with clients end-to-end session persistence through Istio service mesh integration. Traditional MTD approaches either require service restarts or rely on stateless VNF migrations that drop active connections. This work solves that limitation by implementing session-based traffic redirection, allowing CNFs to be moved while maintaining active sessions and minimizing downtime. The framework extends this capability to multi-cluster environments, including migration to Trusted Execution Environment (TEE) clusters, which is not addressed in existing MTD systems for future Telco Cloud networks.

Beyond migration, the framework combines multiple defense mechanisms, namely, IP shuffling and port shuffling, within a single orchestration system spanning the edge-to-cloud spectrum. This integrated approach is novel in that it allows selecting and combining different MTD actions based on threat conditions, rather than relying on a single defense technique. The separation between the MTD Strategy Optimizer (which determines what actions to take) and the MTD Controller (which executes them) provides a clean abstraction that simplifies implementation and allows policy decisions to evolve independently from execution logic.

A third contribution is the enhanced real-time visibility into traffic behaviour during and after migration. The MTD framework collects metrics across multiple domains, providing a unified view of the network state during MTD operations.

4.1.2. MTD Controller

The MTD Controller is responsible for applying the MTD actions determined by the MTD Strategy Optimizer.

4.1.2.1. *Technical description*

The MTD Controller, as part of its responsibilities, mainly interacts with other internal components such as the MTD Strategy Optimizer, and external B5G/6G components such as NFV MANO and the Network Slicing Manager. MTD Controller provides an API endpoint to be called by the MTD Strategy Optimizer whenever an action is determined to be necessary by the latter component. The MTD Controller handles the migration actions by communicating with external components via their APIs.

4.1.2.2. *Functionalities provided*

The following functionalities are provided by the MTD Controller:

- Executing a *live* migration for a **CNF**
- Executing a *stateless* migration for a **VNF**
- Executing a *stateless* migration for a **CNF**
- Performing IP shuffling operations for both **VNF** and **CNF**
- Performing port shuffling operations for both **VNF** and **CNF**

4.1.2.3. *Dependencies*

The MTD Controller depends on the following components:

- The MTD Strategy Optimizer, another component of the proposed MTD Framework.
- The NFV MANO: in our 5G testbed we use the open source implementation maintained by the makers of the NFV standard, the European Telecommunications Standards Institute (ETSI), namely ETSI OSM (Open Source MANO).
- Kubernetes as NFVO: to orchestrate MTD actions on CNFs, Kubernetes is required as the MTD controller is interfaced to its API for operations such as CNF live migration.

4.1.2.4. *Algorithms*

By offloading the algorithmically complex and heavy parts to the MTD Strategy Optimizer component, the proposed framework allows MTD Controller to be a simpler component which is responsible for executing the policies decided by the former component.

4.1.2.5. *Technologies*

For the MTD Controller component, the following technologies are used:

- Python as the programming language for the application logic.

- Kubernetes API for CNF orchestration.
- OSM API for VNF orchestration.

4.1.2.6. *Interfaces and Protocols*

The following interfaces are provided by the MTD Controller:

Table 1: Interface to MTD Controller

EnforceMTDAction	
Description	Enforce the MTD action determined by the MTD Strategy Optimizer
Input	Details of the determined MTD action (possibly in JSON format).
Output	Acknowledgement (positive or negative) of the performed MTD action.

4.1.3. Advancements Beyond D3.1

In this section we report the final design of the MTD Controller with extended functionalities with respect to D3.1.

The MTD Controller extends the functionalities defined in D3.1 with the following new ones:

- End-to-end session persistence during CNF live migration.
- Graphical visibility of near-real time and historical network traffic and redirection during migration.

New dependencies are added to fulfill the above-mentioned functionalities, namely:

- Istio service mesh is added to implement the traffic redirection and end-to-end session persistence during live migration.
- Open-source Prometheus and Kiali frameworks are used for the graphical visibility of the network traffic and its redirection.

The GUI of the MTD Controller implements a new UI for more granular selection of the CNF and the destination cluster for the multi-cluster live migration functionality. Furthermore, the interfaces provided by dependencies' GUIs, namely Prometheus and Kiali, are integrated in the MTD Controller.

4.2. Selective Cyber Threat Intelligence (CTI) solution

The CTI solution is a middleware component developed to assess and advertise the security posture and operational health of clusters in multi-domain environments. Its primary purpose is to enable trust-informed orchestration by serving as a mechanism for gauging the trustworthiness of a cluster, based on real-time security telemetry and vulnerability insights.

Rather than acting solely as a data exchange platform, the CTI solution functions as a dynamic trust assessment tool, continuously evaluating the hygiene of clusters and using that assessment to influence orchestration decisions—such as whether to place or migrate services to a given domain.

Designed as a decentralized and adaptive framework, the CTI system collects vulnerability reports from local scanners or integrated K8s resources, monitors changes in the threat landscape, and shares filtered and policy-compliant threat intelligence across domains. This information enables clusters to make informed, risk-aware decisions about resource placement, thereby reinforcing secure-by-design principles across the orchestration pipeline. The system operates via a publish-subscribe architecture, allowing for selective and policy-driven CTI dissemination between trusted peers. The CTI solution enhances the overall resilience and sustainability of the 6G network by aligning placement decisions with hygiene scores—derived from local vulnerability telemetry. The following sections outline the functional components, interfaces, and data artifacts that enable this capability.

4.2.1. Novelty beyond State-of-the-art

Our CTI solution, *SCOUT: Confidentiality-compliant Cyber Hygiene Observability of Vulnerability Threats*, is a novel solution and first to enable systematic expression of cybersecurity posture of a cloud cluster through: monitoring of vulnerabilities in deployed services, structuring CTI as indicator of the cybersecurity posture (*hygiene*) of a cloud cluster, adapting the amount of intelligence included in CTI to meet and confidentiality needs of its publisher before exchanged with other cloud peers (subscribers). The novelty of SCOUT particularly include:

- **Agentic cyber hygiene monitoring and assessment:** SCOUT is an agent-based solution, comprised of multiple agents, each deployed in a cloud cluster. An agent features a vulnerability monitor, using off-the-shelf scanner (here we adopt open-source Trivy), interacting with SCOUT vulnerability operator and CTI agent. The latter publish CTI of the cluster vulnerabilities while subscribe to CTI of other clusters. Published CTI is maintained by a Trusted Automated Exchange of Intelligence Information (TAXII) server and its backend database.
- **Edge-cloud native architecture:** similar to sFORK, SCOUT is a highly modular solution, having components self-contained and use well-defined APIs and output files to interact with each other. Moreover, although the TAXII protocol follows traditional central client-server approach, the application-/cluster-centric design of SCOUT database, allow to distribute the server side over multiple points, each focused on CTI of different applications and/or clusters.

- **Adaptive information exposure (confidentiality-compliant):** SCOUT pioneers in enabling adaptive reduction of CTI data to meet the confidentiality needs of the publisher and the necessity for intelligence of the subscribers. To do so, SCOUT introduces novel metadata models: the *sensitivity* and *necessity* maps. The former rates on a pre-defined scale the sensitivity of data for each key in a STIX Data Object (SDO) to the publisher, while the necessity map rates the need for the data of this key, by the subscriber. SCOUT introduces a novel data selection strategy and algorithm that takes these conflicting requirements into account and decides based on a trade-off between them what data to omit from a SDO, before publishing it.

4.2.2. Functional components

The CTI component in each cluster/domain assesses cluster hygiene scores and shares the CTI data. Reconfiguration actions are triggered by signals generated from the CTI component's analysis of real-time hygiene scores. These insights guide reconfiguration processes, enabling the continuity of dynamic policy enforcement. The CTI component communicates findings to the Kxs control plane, which can prevent deployments that fail hygiene score requirements or security checks. It alerts the orchestrator when cluster hygiene scores fall below acceptable thresholds. This interaction allows the CTI component to integrate with the Kxs control plane, supporting dynamic security assessments and policy enforcement. It facilitates real-time monitoring, workload adjustments, and security compliance while optimizing the overall performance and reliability of the network.

The CTI service at the core includes the following key functional components as shown in Figure 24:

- **Vulnerability Operator:** Interfaces with local vulnerability scanners and telemetry collectors (e.g., Prometheus, custom security tools) to translate raw data into structured CTI formats.
- **CTI Agent:** Deployed in each cluster to collect, process, and publish local vulnerability data and threat information. Manages the subscription and dissemination of CTI data between clusters and orchestrators. Ensures that only authorized and policy-compliant data is exchanged.
- **CTI Policy Module:** Defines and enforces sharing rules based on cluster-specific confidentiality, privacy, and trust policies. Controls adaptive filtering of CTI data.
- **CTI Analytics Module:** Processes received threat intelligence and computes metrics to compute Cluster Hygiene Scores and feed orchestration logic.

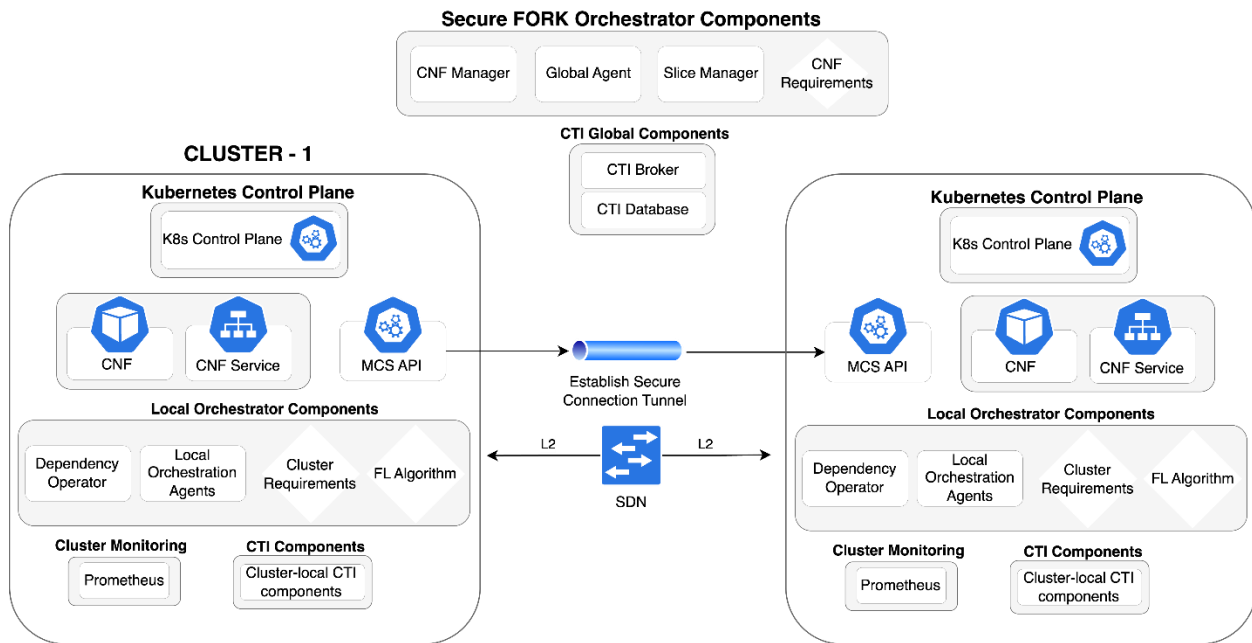


Figure 24: CTI Solution Architecture and components

The components are presented in Figure 24, which illustrates the key elements of the CTI solution. The interaction between these components enables CTI creation and sharing mechanisms effectively. The CTI agent is responsible for creating CTI data in STIX format and sharing it using TAXII protocol. The CTI Agent uses the CTI Policy module to dynamically filter shared data based on the sensitivity of the threat intelligence, the trustworthiness of the requesting cluster, and pre-defined compliance rules. It enforces sensitivity and necessity algorithms to analyse, examine and prepare each vulnerability metadata before sharing it with the other party. This component also calculates the CNF and overall cluster hygiene scores. Cluster hygiene scores directly influence CNF placement and migration decisions, enabling proactive risk mitigation and maintaining slice continuity. It also ensures that the data are structured for sharing with Local orchestration agents previously introduced in Section 2.4.2.

4.2.3. Interfaces and Protocols

The CTI system communicates using standardised and interoperable protocols and APIs, including:

- **TAXII:** Protocol for sharing STIX-based intelligence between CTI Agents, Brokers, and Orchestrators.
- **CTI Hygiene Score Reporting interface:** Receives cluster/domain hygiene scores and supports integration with dynamic security assessment tools.

- **RESTful APIs:** Lightweight interfaces for control, subscription management, and metadata exchange between CTI components and orchestration modules.
- **STIX data model:** for CTI vulnerability representation and sharing across clusters.

4.2.4. Data artefacts

The CTI solution manages and exchanges the following key data artefacts:

- **CTI datasets:** Final CTI datasets to share with receiving parties. These will be contributed as open-source.
- **Vulnerability Reports:** Data collected from scanners summarizing the vulnerabilities present in active services.
- **Cluster Hygiene Scores:** Quantitative representation of a cluster's security posture, based on the number and severity of known vulnerabilities.
- **Policy Metadata:** Definitions of what type of CTI metadata can be shared, filtered, or withheld based on security and privacy considerations.

4.2.5. CTI Cross-Domain selective Sharing

In a multi-domain 6G environment, sharing CTI information across clusters must be carefully managed to ensure that only relevant information is exchanged without exposing sensitive or confidential data. To address this, each domain defines a local data model of both the necessity and sensitivity of the information contained in CTI artefacts. To guide selective sharing, we introduce a confusion matrix that classifies CTI data based on its necessity and sensitivity. This matrix, shown in Figure 25, provides the basis for determining what should be fully shared, anonymized, or withheld, depending on the specific trust policies and compliance requirements of each domain.

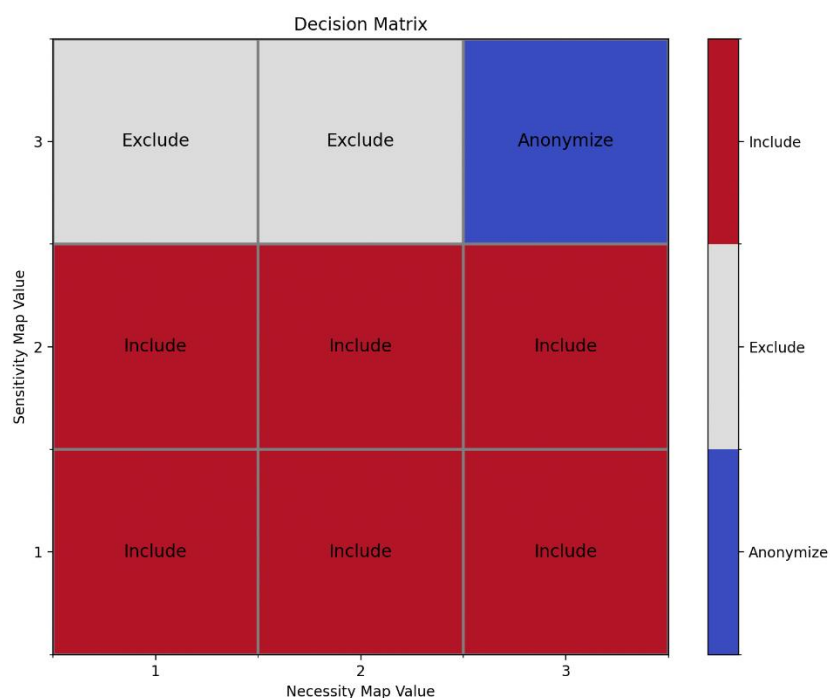


Figure 25: Simple confusion matrix for decision making strategy

Building upon this classification, we have developed a scoring mechanism that quantifies the decision process for each CTI data element. This mechanism evaluates the risk and utility of sharing certain fields and is being refined as part of a broader strategy currently under development for publication. It utilises two maps named as *Sensitivity Map* and *Necessity Map* strategies as shown in Figure 26. In addition, a policy model is implemented to formalize these decisions. It allows each domain to express sharing preferences in a structured format, which is enforced at runtime by the CTI Policy Manager. This ensures consistent and policy-compliant sharing behaviour across clusters. These efforts support a context-aware, privacy-preserving CTI exchange model, enabling NATWORK to achieve secure, adaptive orchestration and management of 6G slices across federated domains.

```
necessity_map:
  fixedVersion: 1
  installedVersion: 2
  lastModifiedDate: 3
  links: 2
```

```
sensitivity_map:
  fixedVersion: 1
  installedVersion: 3
  lastModifiedDate: 2
  links: 2
```

Figure 26: Sample Sensitivity and Necessity maps for decision making strategy

4.2.6. Advancements beyond D3.1

SCOUT design, development and evaluation are now complete. Particularly, post D3.1, final improvements to the functional components of SCOUT were made, along with updates to the sharing policy and unit testing the solution in lab-based unit tests to validate operation and draw performance analysis, which have been described in a scientific paper submission. The solution has been submitted for publication to Springer Journal on Cloud Computing and currently under review. Since D3.1, SCOUT is further being prepared for demonstrator to showcase the solution in EUCNC2026 and future relevant venues.

4.3. AI-based Behavioural Analysis service

4.3.1. DFE/WAI

4.3.1.1. *Technical Description*

DFE and WAI are software components designed to enable high-performance, intelligent, and adaptive security functions within distributed environments in the data plane with different backends. They leverage real-time telemetry and AI-driven threat analytics to detect and mitigate network-based attacks efficiently. DFE and WAI communicate with the Offloaded Function Agent (OFA) API to exchange function discovery, backend state information, requests for activation or deactivation of functions, and change mitigation rules. This interaction enables efficient flow policy configuration adaptive AI model updates based on threat intelligence, and dynamic enforcement of mitigation policies in response to changing attack patterns. Additionally, DFE and WAI expose a REST API that allows external systems to interact with and manage their functionalities through well-defined endpoints.

4.3.1.2. *Functionalities provided*

The functionalities exposed by the OFA controlling DFE at DPUs are the following:

1. Discovery of active DPU instances and their capabilities
2. Activation of accelerated VNF running in the data plane
3. Deactivation of accelerated VNF in the data plane

Moreover, activation, deactivation and configuration of WAI pipelines in P4 switches are added. Integrating Tofino programmable switches into OFA enables WAI capabilities to operate directly in the network data plane. This allows AI-driven decisions to be enforced at line rate through P4-based mitigation programs, reducing reaction latency and avoiding unnecessary traffic redirection to external compute resources. Through this integration, OFA can deploy and activate P4 pipelines, dynamically install or update match-action rules, and adapt forwarding behavior in response to WAI outputs. This supports real-time actions such as traffic filtering, flow blocking,

selective redirection to DPUs, and service chaining. The possibility to run different WAI functions with the same P4 code (i.e., changing only the flow entries) enables a very fast real-time defensive reaction response.

4.3.1.3. *Dependencies*

DPU are configured and run with NVIDIA DOCA Flow SDK. However, at the OFA level, the only dependency needed for communication is a REST API Client able to send get, post and delete methods to the OFA.

the service directly manages Tofino P4 programs, it includes the P4 compilation toolchain dependency. This requires access to the vendor-specific compiler, a build environment aligned with the target switch runtime version, and lifecycle management of the generated pipeline artifacts such as compiled binaries and configuration files. The service must also depend on the Tofino runtime and control-plane interface. This includes integration with vendor runtime APIs (BRFT API) and/or CLI tooling to install, update, and delete match-action table entries, as well as to read counters, registers, and pipeline state. Additional dependencies concern configuration and rollout tooling. The service must support secure transport mechanisms such as SSH or TLS, implement credential and key management, enforce role-based access control, and provide versioning and rollback capabilities for both pipeline artifacts and rule sets. Finally, observability and monitoring capabilities are required. The service must be able to collect telemetry and counters, verify that mitigation rules are correctly installed and active, and detect inconsistencies between the intended policy and the actual switch state.

4.3.1.4. *Algorithms*

The proposed framework integrates DFE and WAI services with the OFA using an Apache Camel-based integration layer. The algorithm coordinates the discovery of available Data Processing Units (DPUs), the lifecycle management of accelerated functions, and the deployment of mitigation policies in the programmable data plane.

Apache Camel serves as the orchestration backbone for integration routes that handle service discovery, function activation and deactivation, telemetry processing, and mitigation enforcement. Through these routes, the framework dynamically interacts with OFA using REST APIs and propagates mitigation decisions to both DPUs and programmable switches.

The algorithm operates in four main phases.

Discovery Phase.

The system periodically queries the OFA/discovery endpoint to obtain information about active DPU instances, their capabilities, and the currently running containers. This step provides an

updated view of available BlueField-2 and BlueField-3 resources capable of executing DFE functions.

Function Lifecycle Management.

Based on the requested mitigation policy or orchestration command, the system selects an appropriate DPU target and invokes OFA REST endpoints to activate or deactivate the required accelerated function. Activation requests start the corresponding DFE container, while deactivation requests stop the running function.

Mitigation Enforcement.

When WAI components detect anomalous traffic patterns, mitigation policies are translated into programmable data-plane actions. The framework ensures that the appropriate P4 pipeline is deployed on the Tofino switch and installs or updates match-action rules that implement the required enforcement behavior, such as traffic filtering, flow blocking, or selective redirection toward DPUs.

Verification and Monitoring.

The system continuously monitors telemetry and runtime counters from DPUs and switches to ensure consistency between the intended policy and the actual deployed configuration. In the event of inconsistencies or failures, corrective actions, such as reinstalling the rule or redeploying the function, can be triggered.

Inputs:

1. PolicyIntent: high-level mitigation intent (e.g., SYN flood detection, flow blocking, selective redirection)
2. TelemetryStream: DFE/WAI telemetry + alerts (push or pull)
3. OFA_BASE_URL: OFA HTTPS endpoint
4. AuthToken/Cert: credentials for OFA and switch control-plane
5. SwitchTarget: Tofino runtime endpoint (BRFT/CLI/SDK)
6. DPUTargets: set of DPU nodes (BF2/BF3), each running containers

Outputs:

1. Active DFE containers on DPUs (DOCA Flow enforced)
2. Active WAI pipeline on Tofino (P4 pipeline + match-action rules)
3. Consistent state across OFA ↔ DPUs ↔ Switch

4.3.1.5. *Interfaces and Protocols*

DFE and WAI communicate with the OFA API to exchange real-time telemetry data, attack detection alerts, and mitigation rules. The REST API provides an interface for external systems to interact with and manage DFE and WAI functionalities. The following endpoints facilitate seamless integration (see Table 2). The final architecture follows a layered model separating orchestration, control, and data-plane enforcement. At the northbound level, OFA exposes a REST API over HTTPS. WAI or orchestration services interact with OFA using standard HTTP methods (GET, POST, DELETE) to deploy mitigation profiles, install or remove rules, activate P4 programs, and retrieve status or telemetry. Authentication and authorization are handled through secure mechanisms (e.g., tokens or certificates). At the southbound level toward Tofino, besides DPUs, OFA communicates also with the switch runtime via vendor-specific control-plane APIs. This interface enables loading P4 pipeline artifacts, managing match-action table entries, and reading counters and state information over secure management channels. Toward DPUs, OFA (or its agent) configures flow rules using the NVIDIA DOCA Flow SDK, translating high-level WAI policies into pipe definitions and rule entries. In the data plane, standard Ethernet/IP/TCP or GTP-encapsulated (in the case of 5G Core Network segment) protocols are preserved, while enforcement is implemented through P4 match-action processing and optional flow steering/block.

4.3.1.6. *Target NATWORK Use Case and Pilot evaluation*

The target use case for the evaluation of the service utilizing DFE/WAI functionalities and the OFA Agent is Pilot 4.1. In this pilot a collaborative scenario between three NATWORK services is envisioned to discover and block dynamic attacks at the data plane using this service, while monitoring the anomalies through the P4-based Analytics service, encompassing DFET module, and providing real-time security monitoring service to update DFE/WAI policies through the AI-driven anomaly detection and root cause analysis component.

Table 2: OFA methods

Endpoint	Method	Description
/discovery	GET	Discovers active DPU instances and their capabilities.
/startddosfunction	POST	Activates SYN flood mitigation detection.
/stopddosfunction	DELETE	DELETE, Stops the running mitigation function.
JSON message format		<pre>{ "running_containers": ["DPU_DFE3", "DPU_DFE2", "DPU_DFE1"] }</pre>

Endpoint	Method	Description
		<pre>{ "message": "Container 'DPU_DFE1' started successfully!" }</pre> <pre>{ "message": "Container 'DPU_DFE1' stopped successfully!" }</pre>

4.3.2. Data plane ML

The proposed data-plane offloaded machine learning (ML) model is a hybrid architecture combining in-network processing with control-plane intelligence to achieve scalable and efficient traffic classification.

4.3.2.1. Technical Description

The following components are used to construct our proposed data-plane offloaded ML model. An overview of the architecture and the interactions between the components can be seen in Figure 27:

- Programmable switches, responsible for extracting features from packets, executing in-network random forest inference, and forwarding a small fraction of the traffic to the control plane.
- An Intrusion Detection System running in the control plane. The IDS (serving as the ML Oracle described in Section 3) does not suffer from the memory and computational limitations of in-network solutions and can provide more accurate classification results for a great number of traffic patterns, at the cost of increased latency and reduced throughput.
- A controller, which supervises the in-network inference. By using the IDS to classify the traffic samples sent by the switches, the controller can detect when the accuracy of the in-network inference declines and can train and deploy an improved classification model as a response.

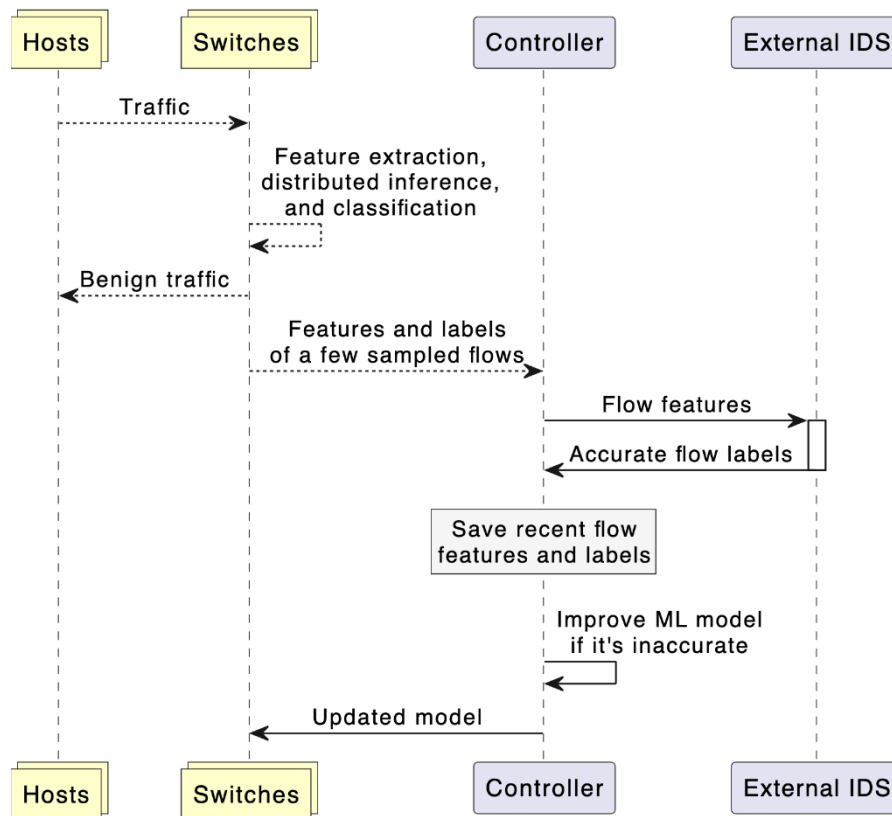


Figure 27: Data plane ML components and interactions

4.3.2.2. Functionalities Provided

Flow Identification and State Management Our design uses a bidirectional 5-tuple hash to ensure forward and reverse traffic map to the same flow ID. This ID serves as an index for hardware registers:

- TNA Pipeline: Uses RegisterAction blocks for atomic read-modify-write operations, storing metadata like timestamps, roles, and feature counters.
- PSA Pipeline: Implements identical logic via macro-generated primitives. Due to PSA's support for larger integer operations, it also computes inline statistical features like moving averages using bit-shift smoothing.

Stateful Feature Extraction Per-flow statistics are updated in the data plane on every packet arrival:

- Metrics: Tracking packet lengths (min/max/sum), TCP flag counters (SYN/ACK/RST), and flow-length counters with overflow protection.

- **Target Specifics:** The PSA version extends this to include Inter-arrival Time (IAT) features. TNA prioritizes pipeline efficiency by minimizing stages, while PSA uses explicit read-update-write blocks for higher arithmetic flexibility.

Inference Pipeline and Decision-Tree Execution Inference is driven by a distributed Random Forest model embedded directly in the switch tables:

- **Encoding:** TNA utilizes range-match tables with ordered keys to fit hardware constraints. PSA uses equivalent match-action tables with more flexible key layouts.
- **Logic:** Each tree contributes a certainty score to metadata fields. A final evaluation table determines the winning label once enough trees have executed or the flow reaches a sufficient packet count.

Label Handling and Packet Treatment Once a flow is classified, the result is stored in a persistent per-flow register to bypass inference for subsequent packets. The switch then applies one of three behaviors:

1. **Drop:** For malicious traffic.
2. **Internal Forwarding:** For service chaining or routing.
3. **Endpoint Delivery:** Stripping inference headers and restoring original EtherTypes.

Reporting and Telemetry A lightweight in-band telemetry mechanism checks flow IDs against a reporting policy. When triggered, a ML reporting header is inserted, containing the flow ID, features, label, and certainty scores. For TNA, an internal metadata carrier header is also used to bridge inference results between the ingress and egress pipelines.

4.3.2.3. *Dependencies*

- **Programmable Data Plane:** P4-capable targets (e.g., Intel Tofino, eBPF) supporting stateful registers and match-action pipelines.
- **Control Plane Infrastructure:** A centralized or distributed controller for model orchestration, P4Runtime management, and real-time model deployment.
- **IDS Logic:** A classification engine (acting as the ML Oracle) to provide ground-truth labels for online learning.

4.3.2.4. *Algorithms*

- **In-Network Random Forest Inference:** A decision-tree-based classifier tailored for P4 and switch constraints (e.g., integer arithmetic, limited memory).
- **Accuracy Monitoring:** Periodic evaluation of in-switch inference accuracy using IDS-classified samples.

- **Model Retraining Algorithm:** Triggers when inference accuracy falls below a threshold; uses collected data to update the ML model.

4.3.2.5. *Interfaces and Protocols*

Table 3: Interface to P4 Runtime

P4 Runtime API	
Description	Provides a control interface for programming and managing P4-based switch behavior, including feature extraction and ML inference logic.
Input	Protobuf or JSON-based configuration describing tables, actions, and match fields.
Output	Confirmation of configuration changes or error messages from the switch.

4.3.3. Advancement beyond D3.1

4.3.3.1. *DFE/WAI*

The final DFE/WAI service extends the functionality introduced in D3.1 by evolving from a DPU-focused mitigation framework to a unified orchestration solution capable of managing both DPU-based feature extraction and AI-driven mitigation directly in programmable network switches. In addition to discovering and activating DFE mitigation functions on DPUs through the OFA Agent, the service now supports the deployment, activation, and configuration of WAI pipelines on Tofino P4 switches. This enables AI-based detection decisions to be enforced directly in the network data plane at line rate, significantly reducing reaction latency and avoiding the need to redirect traffic to external processing systems. The final design also introduces dynamic control capabilities, allowing OFA to deploy P4 pipelines, update match-action rules, and adapt forwarding behavior in response to WAI outputs. This supports real-time mitigation actions such as filtering, blocking, selective redirection to DPUs, and service chaining. Moreover, the system allows different WAI detection functions to run using the same P4 program by updating rule entries only, enabling very fast policy updates without recompiling the data-plane pipeline. Compared to D3.1, the architecture also incorporates a more complete operational framework including P4 compilation toolchain management, runtime control of Tofino switches, secure configuration mechanisms, and monitoring capabilities to verify rule deployment and collect telemetry.

4.3.3.2. *Data Plane ML*

As detailed in the preceding section, the current iteration of the data plane ML model introduces technical additions that expand upon the baseline architecture established in D3.1. Beyond the initial high-level components, the model now includes a comprehensive flow identification and state management system utilizing bidirectional 5-tuple hashing to ensure consistency across

forward and reverse traffic. The functionality for stateful feature extraction has been enhanced to include a broader range of metrics, such as TCP flag counters and Inter-arrival Time features, with specific implementations tailored for both TNA and PSA architecture pipelines. Another addition is the fine-tuning of the inference pipeline and decision-tree execution logic, which now details how certainty scores are aggregated and how winning labels are determined through evaluation tables. Furthermore, the architecture now explicitly defines label handling and packet treatment workflows, allowing the switch to perform autonomous actions—such as dropping malicious traffic or internal service chaining—once a flow is classified.

4.4. Microservice behavioural analysis

The NETWORK B5G architecture follows a microservice-based approach. Microservices architecture is a fundamental enabler of flexible and scalable 6G network services. Unlike monolithic applications, in microservice-based applications, network functions are decomposed into smaller, independent components that operate autonomously, allowing for scalable deployment, real-time adaptability, and efficient resource management, making them well-suited for dynamic network environments. In the following section a module that monitors the performance of microservices in a continual manner to ensure the efficient operation of system is presented.

4.4.1. Technical Description

The Microservice behavioural analysis module performs continuous microservice performance monitoring to ensure efficient operation of the system. In NETWORK, this involves leveraging runtime metric collectors and packet sniffers to continuously track key performance indicators, such as CPU and memory usage, ingress/egress traffic, etc. The module analyses real-time monitoring data, to determine whether microservices meet predefined performance requirements and if abnormal traffic flows occur in the network. If deviations are detected by an AI-based IDS, automated scaling decisions and elasticity actions are triggered to maintain optimal resource utilization and prevent service degradation. Figure 28 shows the position of the microservice behavioural analysis module and its interconnection to other modules. In particular, this module comprises microservice profiling techniques and AI-driven anomaly detection mechanisms for enhanced microservice profiling and threat detection. It interacts with the monitoring engine to collect real-time data on microservices resource usage and traffic metrics, the microservice orchestrator to trigger scaling decisions dynamically based on detected anomalies and the SDN controller to enforce mitigation actions.

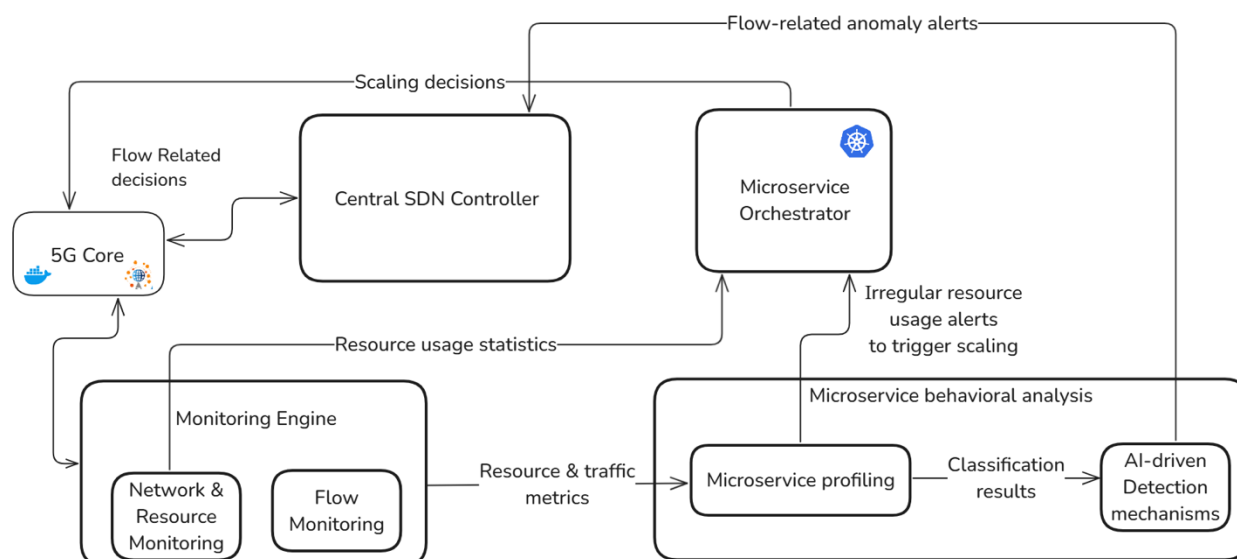


Figure 28: Position of the microservice behavioural analysis module and interconnection to other modules

4.4.2. Functionalities Provided

By microservice profiling and behavioural analysis techniques, the following functionalities are supported:

- Capture network traffic at runtime, utilizing packet-captured (Pcap) files
- Tracking and observation of active flows
- Real-time monitoring of computational and network resource usage
- Flow traffic profiling
- Real-time detection of outliers and identification of anomalies
- Alarm triggering for irregular flows or resource usage
- Automated, near-real time enforcement of flow control rules

4.4.3. Dependencies

NATWORK's microservice behavioral analysis modules are tightly integrated with and dependent on the real-time monitoring tools being developed in the context of Task 4.2 and documented in deliverable D4.1. Indicatively:

- NATWORK relies upon Software-Defined Networking (SDN) for microservices programming and considers Kubernetes for microservice scheduling and orchestration.
- NATWORK builds upon open-source solutions like free5GC [17] for the deployment of 5G networks in compliance with 5G standards.
- For collecting traffic data to perform microservice profiling, packets sniffers, such as tcpdump are being utilized.

4.4.4. Algorithms

While microservices offer significant advantages, their distributed nature makes them inherently more vulnerable to threats such as denial-of-service (DoS) attacks, privilege escalation, and unauthorized access. Given these risks, security is another critical aspect, making behavioural analysis essential in microservice-based architectures to detect anomalies and provide protection against potential breaches.

To effectively analyse microservice behaviour, it is necessary to monitor their performance both on a temporal and a periodic basis, aiming to identify any deviations from normal operation. On the one hand, tracking how network traffic patterns change over time allows for capturing anomalies that may evolve gradually or threats that are identifiable only by analysing a certain period of time. On the other hand, sudden changes in microservice behaviour, such as unexpected spikes in ingress traffic or unusual increases in resource consumption, may indicate malicious activity and have to be contained immediately.

To address these challenges, microservice profiling techniques and AI-based intrusion detection mechanisms are employed to analyse system behaviour in real-time and identify anomalous behaviours. More specifically, two attack mitigation mechanisms are supported: online attack detection based on an exponential moving average (EMA) function and an AI-based intrusion system. The former allows for dynamic and responsive monitoring of data streams, quickly identifying outliers that could indicate abnormal conditions. The AI-powered system continuously analyses telemetry monitoring data to establish behavioural models of normal microservice operations and b) traffic and flow related data. A 1D Convolutional Neural Network (CNN) is utilized for handling the computational resource monitoring data: By profiling CPU, memory, and network usage under typical conditions, the system can classify traffic as normal or irregular, identify deviations that indicate potential ongoing attacks or unexpected system behaviour that aims at exhausting the network resources.

Concerning traffic related data, the AI-powered system processes data input in the form of PCAP files. The AI training follows two distinct modalities: (i) by extracting network flows from each PCAP file and encoding them as integers, and (ii) by leveraging statistical and temporal features derived from each flow, such as packets per second, average packet count, and the time gaps between consecutive packets. For each modality, a dedicated AI model is employed: for the first, a Fully Connected Multilayer Perceptron (MLP) and a CNN for the second. These models are independently trained to learn patterns within the respective data types, aiming to identify network behavior indicative of different attack types. During inference, the system processes live PCAP-based network data and feeds them to the AI-based IDS analyzing it to detect and alert on any signs of malicious activity. The two employed mechanisms are particularly adept at

recognizing abnormal resource consumption patterns, allowing early threat detection and prompt intervention.

Upon detecting suspicious activity, NATWORK applies adaptive flow control and other automated mitigation measures to prevent potential threats and ensure network stability. An example of online UDP Flooding attack detection and mitigation is shown in Figure 29 and Figure 30.

```

:~$ docker exec -it att python udpFlooding.py -i 192.187.3.6 -p 8805 -c y -th 500
#--- Attack start ---#
#-- TCP/UDP Flooding --#
[*] Sent!!!
[!] Sent!!!
[!] Sent!!!
[!] Sent!!!
[!] Sent!!!
[#] Sent!!!
[#] Sent!!!
[*] Sent!!!
[*] Sent!!!
[*] Sent!!!
[#] Sent!!!
[#] Sent!!!
[!] Sent!!!

```

Figure 29: UDP Flooding Attack Execution



Figure 30: New flow control rule (left) and a graphical representation of flow rules (right)

4.4.5. Interfaces and Protocols

The following interfaces are provided by the Microservice behavioural Analysis:

Table 4: Interface to SDN Controller to enforce flow rules

Interface to SDN Controller	
Description	Interface to SDN Controller to enforce flow rules
Input	Json containing a new flow rule (src-ip and port, action enforced)
Output	Json containing the status of the enforcement (success/fail)

Table 5: Interface to Microservice Orchestrator to report irregular resource usage and trigger scaling decisions

Interface to Microservice Orchestrator	
Description	Interface to Microservice Orchestrator to report irregular resource usage and trigger scaling decisions
Input	Json containing the resource alert (microservice name, resource usage fields, scaling action)
Output	Json containing the status of the enforcement (success/fail)

Table 6: Interface to monitoring engine to retrieve real-time monitoring data

Interface to monitoring engine	
Description	Interface to monitoring engine to retrieve real-time monitoring data
Input	Network and resource utilization data
Output	-

4.5. Security-performance balancer service

4.5.1. Novelty beyond State-of-the-art

The section presents the Security-performance balancer service, an O-RAN-compliant near-RT RIC xApp that enables crypto-aware management of user-plane processing within the RAN. The service builds upon the Security-performance-balancer component introduced in Section 3.4.1 and adopts its novel approach to handling heterogeneous cryptographic workloads through algorithm-aware traffic steering and resource allocation. The following subsections describe the technical implementation within the ISRD Liquid Near-RT RIC platform, the main functionalities and dependencies of the service, as well as its operational workflow, interfaces, and integration within the O-RAN ecosystem. Additional details on the service implementation and the underlying algorithms are provided in Sections 5.5 and 6.7, respectively.

4.5.2. Technical description

The Security-performance balancer service is implemented as a near-Real-Time RAN Intelligent Controller (near-RT RIC) xApp that is compliant with the O-RAN architecture. It communicates with the near-RT RIC via the standard xApp API. ISRD provides a proprietary implementation of the near-RT RIC called Liquid Near-RT RIC which can operate with ISRD proprietary Liquid RAN or other O-RAN compliant RANs. The architecture and interfaces of a Liquid Near-RT RIC are shown in Figure 31. The Near-RT RIC connects to the E2 nodes (O-DUs and O-CUs), xApps and the Non-RT RIC over O-RAN compliant E2, xApp API and A1 interfaces, respectively. The SMO/non-RT RIC is not included in the ISRD solution but can be provided by 3rd party.

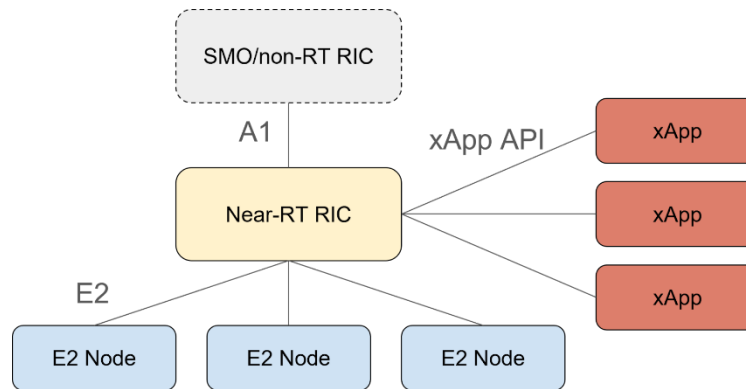


Figure 31: ISRD Liquid Near-RT RIC interfaces.

The Liquid Near-RT RIC general deployment architecture is depicted in Figure 32. Docker Compose is the default deployment method, but Docker Swarm or Kubernetes deployment is also possible. It includes the following Docker containers:

- Management API: Oversees the deployment, configuration, and lifecycle management of xApps within the RIC
- Enablement API: Provides APIs for xApps to interact with essential RIC services
- A1 API: Manages A1 node connections and maintains the state of A1 interfaces.
- SDL API: Provides an API for shared data access among RIC components and xApps
- E2 Service API: Manages E2 node connections and maintains the state of E2 interfaces.
- Grafana: Includes Grafana for Key Performance Measurement presentation.
- Valkey Database: key-value data store
- KPM xApp: It is a proprietary ISRD xApp providing standardized O-RAN KPMs.

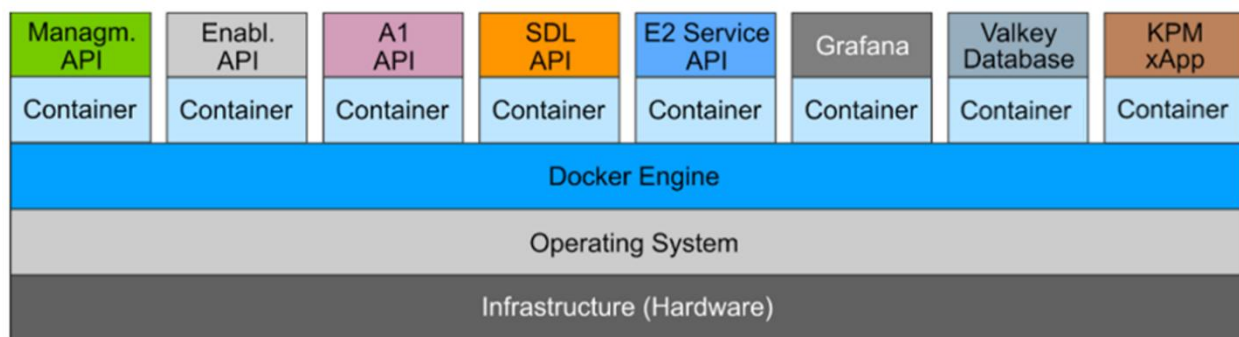


Figure 32: ISRD Liquid Near-RT RIC general deployment architecture.

4.5.3. Functionalities provided

The service provides the following main functionalities:

- **Performance monitoring:** receives infrastructure performance parameters such as CPU load from RAN L2/L3 nodes (i.e., O-DU/O-CU).
- **Crypto-aware provisioning:** steers UE traffic to algorithm-specific CU-UP instances based on the negotiated ciphering algorithm in order to optimize cryptographic processing performance and hardware utilization.

4.5.4. Dependencies

Security-performance balancer service relies on the following components:

- **ISRD Liquid RAN:** ISRD Liquid RAN consists of the following. O-RAN Central Unit (O-CU) is a logical node hosting RRC [18], SDAP [19] and PDCP [20] protocols. O-RAN Distributed Unit (O-DU) is a logical node hosting RLC [21]/MAC [22]/High-PHY [23] layers based on a lower-layer functional split [23]. O-DU and O-CU provide performance measurements, e.g., CPU usage, to the Security-performance balancer xApp via the standard O-RAN E2 interface [24].
- **ISRD Liquid near-RT-RIC:** O-RAN near-real-time RAN Intelligent Controller (near-RT-RIC) is a logical function that enables near-real-time control and optimization of O-RAN elements and resources via fine-grained data collection and actions over E2 interface [24]. The A1 interface, connecting the SMO layer [29] with the near-RT-RIC, enables the SMO to provide Policy Guidance, known as A1 Policies, to control non-real time functions of the near-RT-RIC. The Security-performance balancer service resides in the near RT-RIC as an xApp and communicates with the RIC platform via the standard API.
- **ISRD KPM xApp:** a proprietary ISRD xApp providing standardized O-RAN KPMs.

4.5.5. Algorithms and workflow

CU UP Selection algorithm: cu_up_ciphering_algorithm

CU-CP shall find a CU-UP that is assigned to a specific ciphering algorithm based on the UE bearer request. CU-CP shall assign a specific ciphering algorithm to a CU-UP that hasn't been assigned for an algorithm, if there are no suitable CU-UP found in the current list of candidates for the bearer that is being setup. The CU-UP selection happens during the DRB Setup procedure. The exchange of relevant messages is shown in Figure 33. After CU-CP performs bearer setup it assigns CU-UP to the algorithm used.

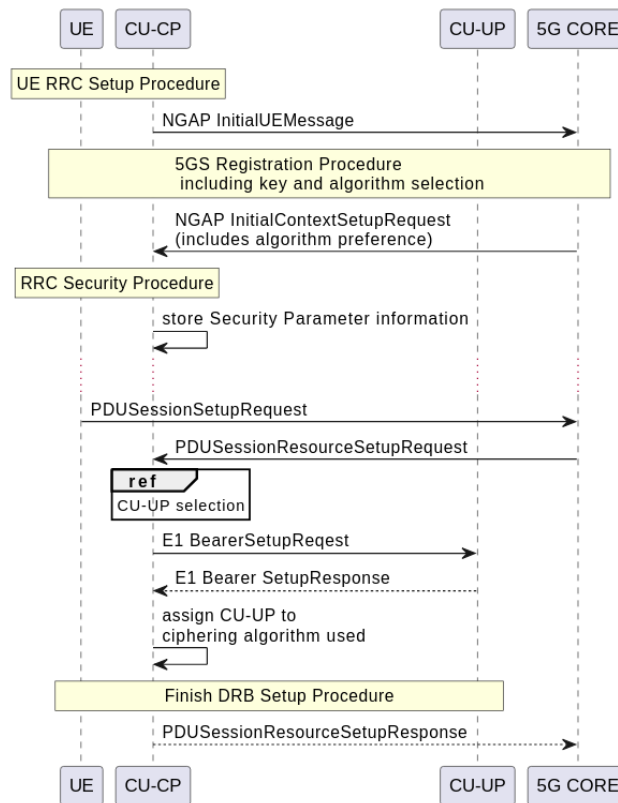


Figure 33 Security-performance load balancer workflow

4.5.6. Interfaces and protocols

Table 7: Interface to O-RAN A1

Standard O-RAN Interface A1	
Description	A1 enables policy-driven guidance of near-RT-RIC applications/functions. Its Policy functions are Orchestration and Automation functions for non-real-time intelligent management of RAN functions. It supports JSON [30]. A1 Policy enables the automation of Security-performance balancer parameters configuration from the SMO layer.
Input	Configure service parameters using the O-RAN A1 policy in the JSON format. Create, update, query, and delete policy. Subscribe to policy status and feedback notifications.
Output	Policy status and feedback notifications.

Table 8: Interface to O-RAN E2

Standard O-RAN Interface E2	
Description	The E2 interface is an open interface between two end points, i.e., the near-RT RIC and the so-called E2 nodes, i.e., DUs and CUs in 5G. E2 allows the near-RT RIC to control procedures and functionalities of the E2 nodes [24].

Standard O-RAN Interface E2	
Input	Performance Measurement: CPU usage.
Output	Subscribe to Performance Measurement: CPU usage.

Table 9: Interface to RIC

RIC APIs (Standard O-RAN Interface)	
Description	The Near-RT RIC APIs are a collection of interfaces providing Near-RT RIC platform services to xApps [31].
Input	Reporting parameters from anti-DDoS xApp.
Output	Change parameters of anti-DDoS xApp.

4.6. Blockchain Based Trust Establishment

Current 5G authentication mechanisms involve main core network functions including AMF and AUSF for trust establishment, which results in significant latency during repeated or federated end-to-end validation [25]. This becomes problematic in scenarios where devices interact with multiple external service providers, as the centralized approach introduces delays and potential bottlenecks. These challenges are particularly clear in a trust-sensitive IoT application such as smart manufacturing and smart cities. To address this, the NATWORK system integrates blockchain technology with the 5G authentication process, enabling a decentralized and transparent trust establishment. This service enables devices to authenticate directly with service providers after the initial authentication and registration in the 5G core. By the reusage possibility within a predefined validity period, devices can establish sessions, perform rapid re-authentication, and access external services without repeating the full core-network authentication procedure. This reduces reliance on centralized functions and supports scalable trust establishment process in 6G networks.

4.6.1. Novelty Beyond state-of-the-art

This module introduces a pseudonym-based trust establishment mechanism that enables multi-purpose use of a single pseudonym within a bounded validity period. To the best of our knowledge, the pseudonyms in the existing blockchain-based 5G authentication frameworks [26][27][28] are generated to provide anonymity during a single interaction, such as UE or vehicle verification. After the authentication procedure, new pseudonyms or authentication runs are typically required for subsequent interactions. Therefore, after establishing a new session, the same pseudonym may not be utilized in accessing additional services, or communicating with another network entity. Other cases usually require either the generation of a new pseudonym or the execution of another full authentication procedure.

The proposed module addresses these limitations by enabling controlled reuse of a pseudonym across multiple related interactions within a defined validity window. This allows the same pseudonym to temporarily be as a persistent and privacy-preserving identity, instead of limiting it to a single instance or usage. As a result, the same pseudonym supports session establishment, fast re-authentication within the 5G network, and secondary authentication toward external service providers. This approach reduces authentication delay and signalling overhead by avoiding frequent pseudonym regeneration and repeated full authentication procedures. In addition, it enables more efficient integration between network-level authentication and service-level access control, while preserving user privacy.

4.6.2. Technical Description

The NATWORK combines standard 5G components with blockchain elements to support blockchain-based trust establishment. It operates on the existing 5G core architecture, preserving its functions while introducing permissioned blockchain for enhanced trust. Key components include the UE, which registers with the 5G network and records a pseudonym on the permissioned blockchain, and the gNB node, which handles the connections. The AMF manages device registration and triggers the blockchain-based process, while the AUSF and UDM perform identity checks. External services in the Data Network (DN) interact with blockchain mechanisms for further authentication, ensuring secure and efficient service access. The service involves five key components and their based technologies working together to establish the trust between the UE and the service provider.

- 5G Core: Utilizing Open5GS, it provides central network control, including functions such as AMF, AUSF, and UDM, which are responsible for authentication and mobility management for IoT devices.
- UPF and DN: The User Plane Function (UPF) also uses Open5GS and connects to the DN, facilitating data routing between IoT devices and service providers.
- UE: Emulating the UE functionality, a machine with UERANSIM is used in the physical testbed to simulate an IoT node.
- gNB: The gNB acts as the RAN node, utilizing UERANSIM to establish radio access and communication between the UE and 5G Core.
- Blockchain: Implemented by Foundry, the permissioned blockchain serves as the distributed ledger that integrates smart contracts to handle various parts of the end-to-end trust establishment process.

4.6.3. Dependencies

This module represents one of the components in S3-S-C2: End-to-End Security Management in NATWORK. The main components of S3-S-C2 related to establishing the trust have been described in D4.3 and D4.4 deliverables. The main dependencies of S3-S-C2 are the following:

- **Blockchain:** Part of the trust establishment and authorization process is replaced with an Ethereum-compatible permissioned blockchain. This provides a decentralized, transparent, and integrity-provided mechanism for device authentication management. It consists of a permissioned Ethereum Blockchain and a smart contract.
- **Bridge:** It is a vital component which acts as a communication bridge between the 5G core network and the permissioned blockchain. The main function of this bridge is to listen to the AMF function inside the 5G core, derive the pseudonym associated with the registration, and to write authentication and access control status to the blockchain via Web3 interfaces.

4.6.4. Functionalities Provided

As a combination of T3.1 and T4.3 described in D4.3 and D4.4 in NATWORK, this module provides the following functionalities:

- **Decentralized Authentication:** It securely verifies the devices across IoT environments using end-to-end authentication. This reduces the chances of unauthorized access by ensuring each identity is trusted and verified.
- **Access Control Management:** It applies strict security policies to manage the UEs that can access specific services in the system that are provided by various service providers. Only authenticated UEs are granted the right level of access to services.
- **Privacy-preserving Identity Management:** It generates secure, privacy-preserving tokens that represent UE anonymized identities and their access policies. These tokens help maintain anonymity while enabling trusted interactions, minimizing the exposure of sensitive identity data.

4.6.5. Workflow

The Bridge generates the pseudonym based on the authentication results that has been handled by the AMF function and add it to the permissioned blockchain. The Service Provider which simulates external applications, leverages the permissioned blockchain for offloading identity verification. It handles authentication requests, verifies pseudonyms via a smart contract, and uses cryptographic signatures for challenge-response interactions. Then, a short-term token will be issued for low-latency access without the need for repeated authentication. By utilizing a

permissioned blockchain to manage device credentials and access control policies, this service ensures verifiable, immutable identity assertions with reduced reliance on centralized systems. Additionally, the Bridge facilitates seamless interaction between the core and the blockchain, ensuring compatibility while maintaining privacy through pseudonym-based identification.

4.6.6. Interfaces and Protocols

The following interfaces are provided by this service.

Table 10: Interface to Distributed Insertion

DistributedInsertion	
Description	Registers the UE by storing its pseudonym and associated access policy in the blockchain.
Input	Identifier, request payload.
Output	Acknowledgement of success/fail regarding the performed insertion (registration).

Table 11: Interface to Distributed Query

DistributedQuery	
Description	Authenticates a UE against a service provider using token-based verification.
Input	Identifier, request payload contains the pseudonym issued for the UE and service provider details.
Output	Acknowledgement of success/ unauthorized access attempt regarding the performed query (authentication).

Table 12: Interface to Token Verification

TokenVerification	
Description	Verifies a service token issued to the UE and validates access using blockchain.
Input	Identifier, request payload contains the token to be verified and the associated UE details.
Output	Acknowledgement of success/ Invalid token or UE pseudonym regarding the performed action (verification).

4.6.7. Advancement beyond D3.1

The previous version of the module in D3.1 has been improved by developing a comprehensive monitoring stack. In this regard, as shown in Figure 34 Prometheus has been utilized to gather numeric metrics and Grafana to create near real-time dashboard visualizations. Since Prometheus is designed to scrape data from long-running services at regular intervals, a Prometheus Pushgateway has been designed that allows the test scripts to push their metrics

immediately after an iteration is finished. Prometheus then collects the data from the Pushgateway.

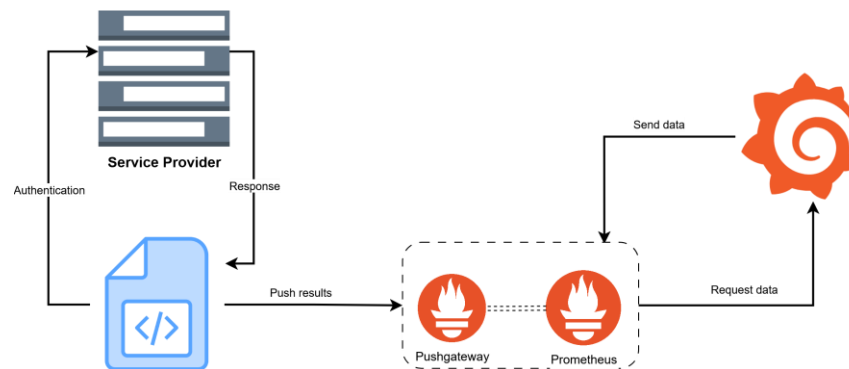


Figure 34: Pseudonym Generation Monitoring Stack

The current updates of the module also strengthens the protocol by establishing a cryptographic linkage between the 5G authentication context in the core and the blockchain identity. This is achieved through a structured derivation process that anchors the blockchain credentials to the 3GPP security framework. In particular, the design introduces a state-synchronized key derivation mechanism based on a standard HMAC-based key derivation function, ensuring that the generated cryptographic material is deterministically derived from the authenticated 5G session. As a result, any blockchain identity used by the device can be privately verified to a valid primary authentication event, preventing unauthorized entities from registering identities that are not rooted in the 5G security context.

In addition, the protocol introduces an identity construction mechanism that combines elliptic curve based public key generation with cryptographic binding of identity metadata. The derived key material is used to generate a verifiable public identity, while a trusted network component associates this identity with a pseudonym identifier and signs the resulting binding before it is recorded on the permissioned blockchain. This combined association ensures that the pseudonym, the public key, and the session context remain cryptographically linked, preventing substitution or remapping attempts and enabling reliable verification by blockchain-based services.

5. Implementation

This section describes the final implementation releases of the orchestration and offloading components, detailing their software tools and modules and operational realization within the overall system architecture.

5.1. Orchestration at the extreme edge

The Gossip and ML services are implemented within the Flocky/OAM framework detailed in D3.1. No notable changes were made apart from the increased flexibility described by the Services, CapabilityProvider and Trait (Handler) APIs. Both the Gossip and ML services are written in Golang, using open source libraries for Gossip (libopenstorage with SWIM protocol) and shared memory mapping, respectively. Examples of ML workload implementation are provided in Python, including a shared memory approach supporting Gossip Learning which is compatible with the Gossip/ML services and their Traits. Weight update processing and integration is performed in Python for optimisation purposes (e.g., GPU accelerators), and will be ported to a reusable library in the future.

Feather repository:

- Main repository: <https://github.com/togoetha/feather-multiruntimenetwork>
- Documentation: README.md in repository
- License: Apache 2.0 (open source)
- Important setup scripts: in repository
- Publication: published in IEEE Access, <https://ieeexplore.ieee.org/document/11274535>

Flocky repository:

- Main repository: <https://github.com/idlab-discover/flocky/>
- Documentation: README.md in repository
- License: Apache 2.0 (open source)
- Important setup scripts: in repository
- Publications: published in IEEE Transactions on Services Computing, <https://ieeexplore.ieee.org/document/11237421>, follow-up accepted for presentation at NOMS 2026

5.2. Orchestration at the CRAN

The core network functions are based on OAI and deployed as microservices within Docker containers. These functions include fundamental 5G core components such as the AMF, Authentication Server Function (AUSF), Session Management Function (SMF), Unified Data Repository (UDR), Unified Data Management (UDM), and multiple User Plane Functions (UPFs),

each configured with distinct Single Network Slice Selection Assistance Information (S-NSSAI) values. An S-NSSAI configuration consists of a Slice Service Type (SST) and a Slice Differentiator (SD), enabling an end-to-end slicing mechanism where a UE can access multiple slices through the same gNB. Each slice binds to a specific service type, adhering to predefined Service Level Agreements (SLAs).

Since UE traffic passes through GTP tunnels within the UPFs, these functions play a crucial role in detecting abnormal behaviours and analysing user demands. Recognizing the significance of traffic data, 3GPP has introduced the Network Data Analytics Function (NWDAF) to collect and analyze core network statistics. However, the O-RAN architecture does not yet integrate NWDAF. To address this gap, we propose positioning NWDAF within the non-RT RIC, allowing it to process core network data via the O1 interface when deployed within a Service Management and Orchestration framework. NWDAF could subsequently apply traffic policies and send analytical summaries to the RT RIC via the A1 interface, enabling real-time control through xApps that manage RAN resources dynamically.

As an open-source implementation of NWDAF is currently unavailable, we developed a custom solution named the Anomaly Traffic Detector (ATD). This network function monitors UPF traffic and analyzes packets, effectively serving as an NWDAF substitute. The ATD is integrated with the FlexRIC-based RT RIC, chosen for its minimal computational overhead and compliance with O-RAN specifications [32]. FlexRIC provides an E2 agent, near-RT RIC, and an xApp development framework. In our setup, OAI's gNB acts as the E2-Agent, while the xApp we developed utilizes FlexRIC's SDK to infer RAN functionalities from the E2-Agent, with a primary focus on the RAN Control SM. The ATD continuously monitors traffic at the UPF, classifying UEs based on their IP addresses and associated S-NSSAI values. It intercepts packets in real time, extracting the necessary features for classification. After collecting an initial set of N packets, the ATD preprocesses the data and feeds it into the trained Random Forest model. The model employs a sliding window mechanism, analyzing batches of 30 packets at a time to ensure near real-time classification while mitigating false positives.

Beyond traffic analysis, the ATD incorporates a ML module to differentiate between benign and malicious traffic. The ML model was trained using the KDDCUP'99 dataset, a widely recognized benchmark for network intrusion detection systems [33]. This dataset comprises over 4 million training instances and approximately 311,029 test samples, containing a diverse range of features related to network connections, including packet header details and content-based attributes.

For our classification model, we selected five critical features:

- **Protocol Type** (e.g., TCP, SCTP, UDP)
- **Service Type** (e.g., HTTP, FTP, SSH)

- **Connection Status Flag** (e.g., SF for normal, REJ for rejected, RST for reset)
- **Source and Destination Byte Counts**

These features were chosen for their significance in distinguishing between normal and malicious traffic and their compatibility with real-time packet analysis via Scapy. Additionally, the dataset includes four types of attack labels: Probing Attack, Remote-to-Local Attack, Denial of Service (DoS) Attack, and User-to-Root Attack. The preprocessing pipeline included:

- **Label Conversion:** Transforming multi-class labels into a binary classification—1 for attacks, 0 for normal traffic.
- **Flag Standardization:** Converting dataset-specific flag values to formats recognized by Scapy.
- **Feature Selection:** Extracting packet-level features relevant to real-time classification.
- **Encoding and Scaling:** Applying OneHotEncoder for categorical variables and MinMaxScaler for numerical values to normalize data.

Following preprocessing, we trained multiple ML models using TensorFlow, including Random Forest, One-Class SVM, Local Outlier Factor, K-Nearest Neighbors (KNN), and Autoencoders. Performance evaluations led us to select Random Forest due to its superior accuracy and efficient training/inference times as it is shown in Figure 35 and Table 13. Upon detecting anomalies, the ATD reports the per-UE anomaly percentage to the xApp, which then executes RAN control countermeasures. It processes incoming messages from ATD clients, extracting UE identifiers, S-NSSAI values, and the associated anomaly ratios.

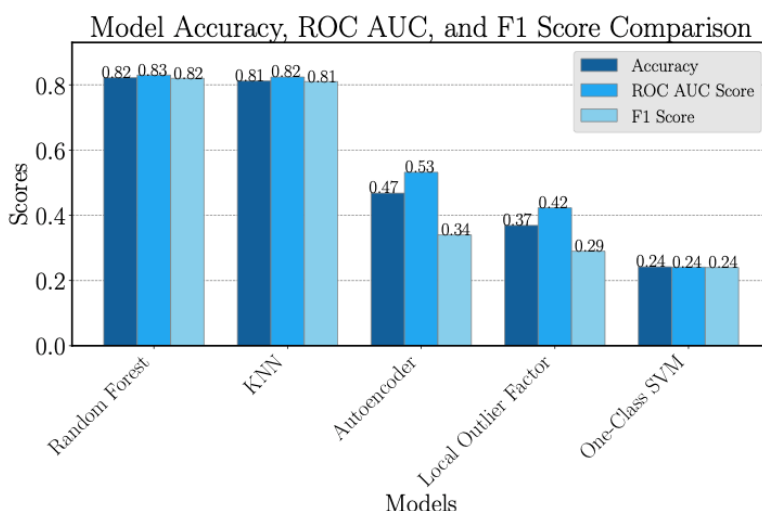


Figure 35: Model Training Evaluation - Accuracy, ROC AUC and F1 Score Comparison

Table 13: Training and inference times for various machine learning models

Model	Train Time (s)	Infer Time (ms)
Random Forest	4.36	3.4
One-Class SVM	125.98	6.14
Local Outlier Factor	0.77	0.6
KNN	5.51	9.14
Autoencoder	181.46	23.11

The detailed operation of our framework is illustrated in Figure 36. The ATD unit continuously monitors UPF traffic and classifies clients based on their IP and S-NSSAI values. It manipulates each packet in real-time, extracting the necessary features that our ML model was trained on. After collecting the first N packets, the ATD preprocesses these features and feeds them into the Random Forest classifier. Then the Random Forest by applying a sliding window mechanism processes N=30 packets at a time, classifying the traffic as benign or malicious. The reason we selected 30 packets-window is to reduce infer/prediction times as close to real-time and avoid false outliers in the classification with a larger input range. Finally, the ATD sends the anomaly percentage per UE to the xApp for the RAN Control and countermeasures.

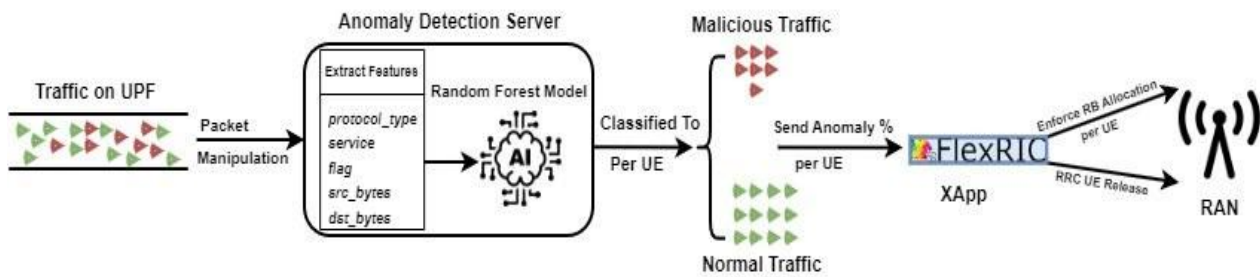


Figure 36: Detailed Architecture of the AI-Driven Network Intrusion Detection System

Code repository:

- Main repository: <https://github.com/teo-tsou/oai-anomaly-detection/>
- Documentation: [README.md](#) in repository
- License: Apache 2.0 (open source)
- Publication: <https://doi.org/10.1145/3636534.3697311>

5.3. Orchestration at the Core (sFORK and SCOUT)

The SCOUT system is implemented as a container-based architecture composed of 4 main components: Vulnerability operator, CTI agent, CTI broker and a MongoDB database. It has a backend API and frontend UI. The backend is developed in Python using the Flask framework, and it handles the ingestion, processing, and transformation of vulnerability reports into STIX 2.1 format, which are then published to a TAXII 2.1 server. The frontend is built with React and TypeScript, styled using the Mantine UI framework, and served using Vite.

FORK system is also implemented as a container-based architecture. It deploys its dependency operator and orchestration and connection components to ensure cluster connectivity and application dependency. Both components are containerized and deployed in Kubernetes for ease of deployment and scalability. Communication between the UI and API is handled via RESTful HTTP endpoints. The system can be deployed either manually using Github repo implementation steps in Kubernetes with kubectl or using automated scripts.

SCOUT Code repository (CTI framework):

- Main repository: <https://github.com/NetworkConvergenceLab/scout>
- Documentation: [README.md](#) in repository
- License: Apache 2.0
- Publication: in submission

FORK Code repository:

- Main repository: <https://github.com/NetworkConvergenceLab/fork>
- Documentation: [README.md](#) in repository
- License: Apache 2.0 (open source)
- Publication: <https://doi.org/10.1109/ICIN60470.2024.10494435>

sFORK PoC implementation is now complete, featuring a set of: cluster-local agents, global (slice) agent, a distribution agent and CNF manager facilitated by Harbor container image registry service.

sFORK Code repository:

- Main repository: <https://github.com/NetworkConvergenceLab/sfork>
- Documentation: [README.md](#) in repository
- License: Apache 2.0 (open source)
- Publication: being drafting.

5.4. WAI/DFE

The software implementing the WAI and DFE functionalities at both the DPU and the P4 switches is tightly coupled with the backend capabilities of the adopted hardware platforms. On the DPU side, the solution is implemented using NVIDIA DOCA SDK version 3.2, targeting both BlueField-2 DPUs and BlueField-2 Converged Cards. The DDoS mitigation logic is implemented as a DOCA Flow-based VNF developed in C/C++, leveraging hardware offloading for flow steering, packet filtering, and statistics collection. Compared to the previous deliverable, the implementation has been updated to align with the DOCA 3.2 programming model, while preserving the same functional architecture and processing pipeline.

On the P4 side, the data-plane implementation remains unchanged from the previous deliverable. A baseline P4 program composed of a cascade of flow tables is used to evaluate P4 capabilities regarding stateful memory usage and SRAM/TCAM resource requirements. The program is compiled and analyzed using P4 Insight and P4 Studio (SDE 9.7), targeting the APS Tofino switch deployed at the CNIT laboratories. The coordination between WAI, DFE, and the underlying data-plane elements is managed by the OFA (Orchestration and Flow Agent), which serves as the unified control-plane integration point across heterogeneous enforcement domains. OFA exposes a northbound REST API over HTTPS, enabling WAI components or orchestration services to deploy mitigation profiles, install or remove rules, activate P4 programs, and retrieve status or telemetry using standard HTTP methods. Secure access is enforced through authentication and authorization mechanisms. At the southbound interface, OFA interacts with programmable switches and DPUs via backend-specific control-plane APIs. For Tofino switches, it manages P4 pipeline artifacts and match-action tables through the vendor runtime, while for DPUs, it translates high-level WAI policies into DOCA Flow pipe definitions and rule entries for direct enforcement on the SmartNIC.

Regarding software availability, none of the WAI-related P4 implementations, including the DDoS detector and the adversarial defense mechanisms, are currently released as open source. Their publication is restricted due to ongoing research activities and planned extensions.

The DOCA-based DFE implementation has been finalized and is ready for release; it will be made publicly available after publication of the associated scientific results.

Code Repositories:

DOCA DFE repository: https://github.com/engranaabubakar/NATWORK_DFE

WAI repository: https://github.com/engranaabubakar/P4_DDoS

OFA Agent: https://github.com/engranaabubakar/DPU_OFA_Agent

DNN models: https://github.com/emiliopaolini/P4NN_journal

License: Apache License 2.0 (open source, upon release)

5.5. Security Performance Balancer

The Security Performance Balancer is implemented as an O-RAN compliant xApp and shipped with ISRD Liquid RAN and Liquid Near-RT RIC as a commercial product package. The product is provided to our customers on a per-license basis and as such there is no public repository available.

5.6. LLM-based IDS

The LLM-based IDS has been implemented as a modular and extensible pipeline that encompasses traffic preprocessing, representation learning, and detection components. It relies on modern deep learning frameworks supporting transformer architectures and GPU acceleration. The preprocessing module captures raw traffic traces and organizes packets into flows based on the 5-tuple definition. Each flow is transformed into a sequence of packet tokens, where packet-level features and metadata are embedded into dense vectors augmented with positional encodings. The system exposes programmatic interfaces that allow integration with external orchestration and monitoring components.

Code repository:

- Main repository: https://github.com/koukipp/contrastive_transformers_ids
- License: Apache 2.0 (open source)
- Publication: <https://doi.org/10.48550/arXiv.2505.08816>

5.7. Data plane ML

Our In-network ML repository is structured around a flexible development and simulation environment. It includes tools for compiling and deploying P4 programs for both eBPF and Intel Tofino targets. It supports launching a Mininet-based testbed, simulating traffic using PCAP files, and orchestrating various components such as controllers, oracles, and coordinators. Key components include a Docker setup for managing dependencies, Python scripts for model coordination and evaluation, and a shared library for synchronizing constants between P4 and Python. The design emphasizes modularity and experimentation, making it easy for prototyping secure and scalable in-network ML systems.

- Main repository: <https://github.com/P4ELTE/Natwork-DataPlaneML>
- Documentation: README.md in repository
- License: Apache 2.0 (open source)

- Publication: In-Network Attack Detection Using Disaggregated Inference & Online Learning - IEEE Globecom 2025

5.8. MTD Controller

The MTD controller operates the parallel live migration (LiMi) of containers and microservices as an MTD operation to enhance the security of an NFV orchestration platform, specifically Kubernetes orchestration for CNFs, dynamically changing the attack surface of the cloud native systems both in the edge and core domains. Within MTD, container migration can serve as one of the approaches to achieving this dynamic shift by relocating workloads and disrupting potential attack vectors. Additionally, such migration can be useful for isolating an infected container by moving it to a secure cluster, allowing deeper analysis when an unknown attack occurs. This isolation helps contain the breach, preventing the infection of other applications, unauthorized access, and exfiltration of data.

The proposed migration approach leverages Kubernetes orchestration, the CRIU library, a network file system (NFS), and a local container image registry. CRIU (Checkpoint/Restore In Userspace) is a Linux-based software tool capable of freezing a running process, container, or application and creating a checkpoint of its current state, saving it to the disk. This checkpoint can then be transferred to and restored on any host, allowing the application to resume its work as when it was frozen on the source host. The Kubelet checkpoint API of the Kubernetes orchestrator provides the mechanism for creating these checkpoints. This API allows its users to initiate a checkpoint, which captures the complete state of a container, including its memory, process information, and file system data. The resulting checkpoint can later be used to restore the container to its exact previous state. Triggering the *kubelet* API initiates checkpoint creation through the container runtime in use. The runtime forwards this request to its lower-level components, which, upon receipt, utilize CRIU to carry out the checkpointing process [34]. During this process, CRIU performs various steps as shown in Figure 37.

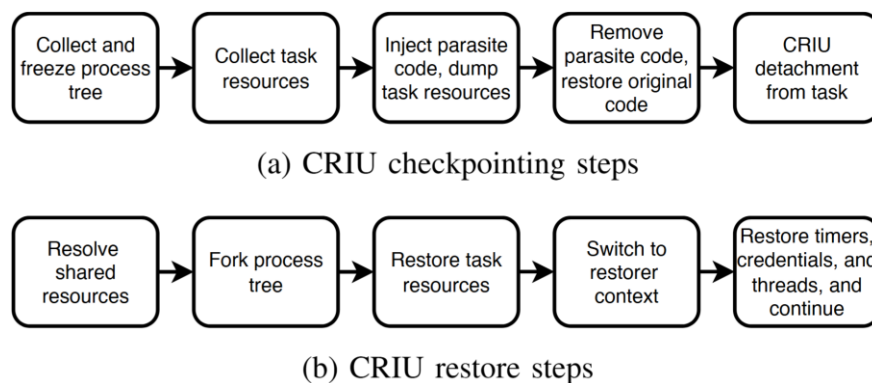


Figure 37: CRIU steps

During the checkpoint process CRIU injects a “parasite code” into the container to collect process IDs (PID), task resources, file descriptors, including open files and sockets, registers, and other essential task parameters. The checkpoint can then be found as an archive file on the source node. CRIU then removes the parasite code and detaches itself from the container’s process.

In contrast to checkpointing, restoring a container is currently not possible via the Kubelet API. Thus, the MTD controller uses the algorithm and phases defined in Section 6.4. to restore the container on the destination cluster while keeping the high-level Kubernetes orchestration aware of the changes.

Benchmarking CNFs have been developed to test the MTD controller’s performance using various containers, each specialized to handle resource-intensive tasks. They are designed to evaluate migration performance under different service requirements, as detailed below:

1. CPU: The CPU benchmarking application is a program that simulates state transitions via a series of states. In each of its four states, a CPU-intensive computation is performed, beginning in State 1 (S1). By calculating 2 to the power of 256 10 million times in a loop, a sustained high CPU load is created for an extended period. Once the task is completed in a given state, the program transitions to the next state, repeating the process. When the task in the last state is finished, it transitions to S1 again, thus continuously repeating the workload while tracking the number of completed cycles and current state.
2. Memory: To perform migrations of applications that are running memory-intensive tasks, a local *redis* in-memory cache is deployed. For comparison, three instances with three different amounts of entries are migrated. 10'000, 100'000 and 1'000'000 key-value pairs are written into the database.
3. Disk: To explore the impact of disk usage on the proposed migration, a MongoDB instance is used, storing its data in BSON documents, a binary representation of JSON files.

Code Repository

- Main repository: <https://github.com/RinchenKolo/ContMigration>
- Documentation: README.md in repository
- License: Apache 2.0 (open source)
- Publication: <https://doi.org/10.1109/ICC52391.2025.11161983>

The implementation of the two new features described in Section 4.1.3 resolves the previous challenge of integrating traffic redirection into container migration by leveraging Istio, where CNFs are migrated with their sidecar proxies and traffic is seamlessly switched post-migration by patching VirtualService routing rules for minimal service disruption. Prometheus and Kiali

provide comprehensive observability through metrics collection and visual traffic graphs, with Prometheus federation enabling multi-cluster monitoring from a central backend to validate correct redirection and workload integration.

1. Main repository: <https://github.com/wsoussi/CubeMig>
2. Documentation: [README.md](#) in repository
3. License: Apache 2.0 (open source)
4. Publication: being drafting.

5.9. Blockchain Based Trust Establishment

The implementation provides a general guide in setting up and validating a 5G testbed environment designed to demonstrate blockchain based trust establishment between IoT devices and service providers. The setup leverages components such as Open5GS (5G core), HTTPS Server (service provider network), UERANSIM (radio simulation), Raspberry Pi (IoT UE), and Foundry (blockchain):

Testbed Initialization

Install and configure Open5GS, UERANSIM, and blockchain tools.
Set up UE, and gNB and 5G Core interaction.

Software Stack Setup

Deploy core NFs: AMF, AUSF, UDM, SMF, UPF.
Configure blockchain (via Foundry) and smart contract for trust attestation.

UE Registration Flow

UE initiates first-time network registration through gNB.
AMF/AUSF authenticate and record pseudonym/trust info to blockchain.

Blockchain Verification

IoT service provider queries blockchain using pseudonym.
On successful attestation, mutual authentication is completed.

Validation Criteria

Establishment of trust without further core network involvement.
Successful cryptographic verification and secure channel setup.

Code Repository

- Main repository: <https://github.com/elte-cybersec/E2E-5G-Trust>

- Documentation: README.md in repository
- License: Apache 2.0 (open source)
- Publication: <https://doi.org/10.23919/SoftCOM66362.2025.11197380>

6. Strategies and Optimisation Algorithms

This section details the strategies and algorithmic optimization methods used to enable adaptive orchestration, data plane offloading solutions, proactive threat mitigation, and energy-efficient service management.

6.1. Orchestration at the extreme edge

6.1.1. Strategies adopted

Orchestration in the edge is divided into two main aspects: flexibility, to support non-container workloads and highly heterogeneous device topologies, and decentralization, which requires a level of node and workload modeling not supported by common APIs such as Kubernetes. The solution is presented in terms of Feather, which uses eBPF and backend detection to support multi-runtime pods (networking) and to advertise its capabilities, and Flocky, which uses Open Application Model (OAM) to enable decentralized metadata discovery and orchestration.

6.1.2. Problem Definition

Two critical aspects must be solved to enable effective orchestration in the edge:

- Device specifications and user/service requirements are far more varied than in the cloud. This presents an opportunity to support various types of workloads other than containers depending on device capabilities but also requires suitable capability modelling for optimal orchestration.
- A decentralized solution is more suitable for edge orchestration; therefore, a suitable framework should be constructed that supersedes existing solutions such as the Kubernetes API, allowing devices and workloads to be modeled at a high level for decentralized metadata discovery and exchange.

6.1.3. Developed Solution

Feather leverages Virtual Kubelets, combined with eBPF traffic routing and extensible backend (i.e. containerd, OSv) implementations to support various types of workloads in a single orchestrator agent for Kubernetes/OAM deployments.

Flocky extends OAM and the Swirly discovery mechanism to enable decentralized node metadata discovery, including supported runtimes and available applications. The highly pluggable orchestration system allows for different preferences and algorithms per node. The orchestration algorithm has “Find acceptable nodes” and “Rank by QoE” stages that can be extended by

implementing a “QoE evaluator” interface and configuring a specific Flocky instance to use that implementation.

6.1.4. Advancements beyond D3.1

Further development of the decentralized Flocky framework into edge intelligence requires both a service to enable ML workloads within the existing orchestrator, and a method of exchanging weights updates between nodes running the same models. Solutions for these aspects need to follow the OAM model developed for Flocky, and the generic Flocky framework for discovery and orchestration.

The ML service enables ML workloads by extending application deployment manifests with optional ML Traits. These Traits are applied to workloads by the service to ensure, for example, hardware acceleration and ML-side mechanisms for distributed weight updates (i.e., Gossip Learning). The service also exposes the necessary hardware and system information in the metadata store to enable suitable node selection during the deployment process.

The Gossip service enables the communication aspect of edge intelligence; operating within the Flocky framework it handles gossip cluster creation and attaches various workloads to the relevant clusters for data exchange, for example weights update distribution for ML workloads.

6.2. Orchestration at the CRAN

6.2.1. Strategies adopted

To effectively manage the CRAN, we leverage containerized network functions and intelligent control mechanisms, ensuring optimal resource allocation and user management. The strategies involve utilizing OAI core and RAN network components, deploying them as microservices using Docker. Also, by using FlexRIC for the Real-Time RIC and integrating AI-based anomaly detection xApp to enhance network security and performance.

6.2.1.1. Problem Definition

Sophisticated attacks compromise data integrity, user privacy, and overall network functionality. In our experimental setup we demonstrate that a DoS attack can disrupt key 5G core components such as the UPF, leading to failures within the RAN causing permanent bufferfloats. Beyond network disruptions, these security threats result in inefficient resource utilization, higher operational costs, and increased recovery efforts.

6.2.1.2. Developed Solution

To counter these challenges, we propose a solution that integrates real-time anomaly detection, adaptive resource management, and user traffic monitoring. The xApp leverages AI/ML models

trained on real-world datasets to classify network traffic and dynamically allocate resources and suppress malicious users. It detects malicious behaviour, triggering RRC connection terminations to protect the network while prioritizing legitimate users via end-to-end slicing. This implementation, built within the OAI platform, utilizes standardized O-RAN interfaces and Service Models from FlexRIC. The functionality of the xApp is described below:

Algorithm: xApp Functionality

1. **Initialization:**
 - Establish a connection with RT-RIC and subscribe to the RC SM.
 - Accept incoming ATD client connections.
 - Initialize data structures for tracking UE activity.
2. **Monitoring and Data Processing:**
 - Receive and parse ATD messages.
 - Extract UE ID, S-NSSAI values, and anomaly ratios.
 - Continuously update UE-related records.
3. **Dynamic Resource Allocation:**
 - Calculate PRB assignments based on anomaly scores.
 - Apply scaling mechanisms to maintain fair resource distribution.
4. **Threat Mitigation and Countermeasures:**
 - Identify and classify UEs exhibiting malicious behavior.
 - Reduce PRB allocation for flagged users and redistribute resources.
 - If a UE reaches 100% anomaly ratio, trigger RRC connection release.

6.2.2. Advancements beyond D3.1

The core strategies and optimization mechanisms for the CRAN orchestration framework remain consistent with those described in D3.1. The overall architecture continues to rely on the integration of containerized OAI RAN and core components, the FlexRIC near-real-time RIC platform, and the AI-driven anomaly detection pipeline to support security-aware resource orchestration. The fundamental control logic of the xApp, including anomaly-based resource allocation and mitigation actions at the RAN level, has not been modified. Overall, the advancements beyond D3.1 mainly concern system integration, operational validation, and stability improvements of the orchestration framework, rather than fundamental changes to the underlying orchestration strategy.

6.3. Orchestration at the Core

6.3.1. CTI Cross-Domain selective Sharing

Sharing CTI data requires caution, as it must include relevant threat details without exposing sensitive or confidential information. If not properly filtered, such data could reveal system-sensitive information and lead to security risks. The Selective CTI sharing mechanism applies CTI policy to examine each vulnerability data obtained through the security scanner. These policies determine which metadata can be included in the CTI package. It filters the vulnerability metadata using data anonymisation and exclusion. After this process, the CTI agent generates the CTI data with the selected vulnerability metadata and complies with the STIX data serialisation standard. The CTI component in each cluster/domain assesses cluster hygiene scores and shares the CTI data. Reconfiguration actions are triggered by signals generated from the CTI component's analysis of real-time hygiene score. These insights guide reconfiguration processes, enabling the continuity of dynamic policy enforcement. The CTI component communicates findings to the orchestration components in local clusters and control planes, which can prevent deployments that fail hygiene score requirements or security checks. It alerts the orchestrator when cluster hygiene scores fall below acceptable thresholds. It facilitates real-time monitoring, workload adjustments, and security compliance while optimising the overall performance and reliability of the network.

6.3.1.1. *Problem Definition: Sensitivity vs. Necessity*

In CTI sharing, organisations face a trade-off between protecting sensitive information and ensuring the utility of shared data. CTI data often includes sensitive information that makes organisations hesitate to share and collaborate. If exposed, highly sensitive indicators may pose privacy, reputational, or security risks. However, omitting such details can significantly reduce the usefulness and actionability of the threat intelligence. The challenge lies in determining which information is essential (necessary) for the receiving party, while minimising the exposure of sensitive data. This tension between sensitivity and necessity forms a core problem in secure and effective CTI exchange.

6.3.1.2. *Developed Solution*

To address this challenge, we designed a dynamic and adaptive mapping framework that assigns a *sensitivity score* and a *necessity score* to each data field within the shared intelligence. Each vulnerability is assessed individually; vulnerability metadata is utilised to assign relative scores. This dual-scoring system enables granular control over what is shared, allowing CTI Agents to prioritise data that is critical for defence while withholding or anonymising overly sensitive fields. These scores are used in the decision-making algorithm to determine the appropriate sharing

policy for each piece of information. This ensures a balanced, policy-driven exchange of CTI that supports collaboration without compromising security or privacy.

6.3.2. Workload Prediction for Scheduling

6.3.2.1. *Problem Definition: workload prediction including anomalous*

In large-scale distributed networks, efficient workload scheduling becomes increasingly complex. Traditional schedulers might often react to immediate resource usage without anticipating upcoming demands. This can lead to poor performance, especially when there are sudden spikes in traffic or abnormal activity. Without the ability to predict workloads, the system may either over-allocate resources (wasting energy) or under-allocate (SLA loss). The key challenge is to forecast future load and adjust scheduling decisions in advance while also handling unpredictable behaviour like Denial of Sustainability (DoSt) attacks or traffic bursts.

6.3.2.2. *Developed Solution: AI prediction*

To address this challenge, we designed a lightweight, AI-driven workload prediction component as a microservice that can be integrated into the orchestration and scheduling layer. This component exposes a standardized API, allowing the orchestrator to query predictions on key workload indicators—such as node-level usage trends or traffic surges—based on historical telemetry or CTI events. It receives telemetry data inputs (e.g., CPU and memory metrics) and returns short-term forecasts for resource demand across nodes. These forecasts assist the orchestrator in taking preemptive actions— scaling workloads or adjusting placements to energy-efficient zones—before overload or performance degradation occurs. The prediction service also supports scenarios involving anomalous patterns, helping distinguish between typical usage fluctuations and potential DoSt-like anomalies. This ensures better performance, lower energy usage, and improved resilience in dynamic network environments.

6.3.3. Advancement beyond D3.1

sFORK lightweight: deterministic optimisation strategy that compares a slice (service) requirements against status indicators for participating clusters and vice versa, and subsequently make a placement decision on a CNF level, resulting in a distribution map on a service level. The algorithm is intended to be light, fast-responding and posed as the default to include with sFORK.

Federated learning for workload prediction and intelligent sFORK decisions: our ML-based workload prediction has been further developed into a federated learning solution, and currently undergoing lab-based testing. The transformation to federated workload prediction, enable different clusters to influence placement decisions based on cluster-local knowledge as well as global ecosystem one.

6.4. Moving Target Defence (MTD)

The proposed Moving Target Defence (MTD) framework introduces two algorithms designed to optimize stateful LiMi performance for CNFs:

- I. Container Restore Algorithm for Kubernetes-aware Orchestration: This ensures seamless state restoration during migration while maintaining Kubernetes cluster constraints.
- II. Parallel LiMi Scheduling Using ML-based Time Prediction: this leverages ML to predict and optimize migration timing across multiple concurrent instances.

Each of the following subsections outlines the problem being addressed and the corresponding algorithm developed to solve it.

6.4.1. Container Restore Algorithm for Kubernetes-aware Orchestration

As previously described in Section 5.8, the CRIU library is partially integrated to Kubernetes, with *kubelet* API allowing the creation of a checkpoint of a running container but missing an API request to equivalently restore the container from the checkpoint on the destination cluster.

To solve this issue, the MTD controller defines a process/algorithm for the restore phase of a stateful container LiMi performed as an MTD operation. A live migration in this context refers to the process of transferring a stateful containerized application running in a Kubernetes pod from one cluster to another with the goal of minimal downtime (i.e., the time an application is unavailable), while preserving the application's state. Thus, the restore process starts with having a container checkpoint and is summarized as follows:

1. Create a checkpoint using the kubelet API.
2. Change the permissions of checkpoint files to enable non-root users to restore the container.
3. Convert the checkpoint into a container image.
4. Push the new image to a local or remote registry (e.g., Dockerhub).
5. Apply a prepared YAML file that pulls the new image.

This above-described algorithm is implemented as a shell script. The first part of the script searches for a pod name, which must be given as an argument when running the program. Once the existence of the pod is confirmed, a checkpoint is created by using the kubelet API's POST command. The checkpoint file is then saved to the NFS. As the file is saved with permissions allowing modifications only by root users, the next part of the algorithm changes the permissions of the above-mentioned file. Once non-root users can change the file, *buildah* is used to convert the checkpoint file into an image and push it to the local registry. Once the image is successfully

pushed, the new image is pulled by Kubernetes via a YAML file, which contains the necessary information using the pod's original name.

At the end of the migration script, the remote migration controller applies a predefined YAML file to launch the previously checkpointed container. Kubernetes is then instructed to wait for the pod to be ready, and once it is, let it run for five seconds. After that, the *StatefulSet* and its corresponding pod are deleted, and the execution cycle starts over again.

6.4.2. Parallel Live Migration Scheduling Using ML-based Time Prediction

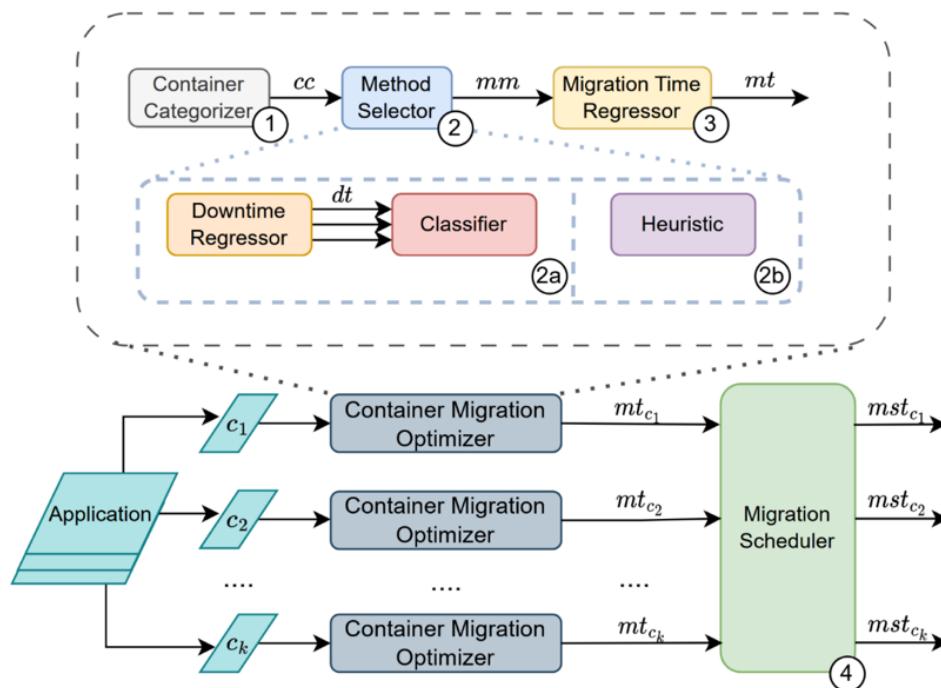


Figure 38: ML Classifier + regressor to optimize the live migration of microservice-based applications

Due to the interdependency of components (e.g. the back-end part may need to have a persistent connection with the DB instance), it is crucial to determine the order of migration for each component, so that the migration time would be minimized. Furthermore, the workload of each component also affects the migration process, requiring a careful estimation of how long it will take for each container to be migrated and functional in the destination location. The problem in this case is to develop an accurate estimation method for the migration time of each component and consequently, a proper scheduler to prioritize the migration of certain components for the least service disruption.

As depicted in Figure 38, the first part of the solution, the Container Migration Optimizer, develops an ML-based classifier selecting the best migration method per container to minimize

the migration time and service downtime of the container, based on its workload type. To this end, a tool for collecting metrics of a running container is required to be connected to the MTD framework, providing the resource utilization of the corresponding container and feeding it to the ML classifier or heuristic model (formed from statistical analysis of the dataset used for training the classifier). Once the migration method is selected, the second part of the solution, the Migration Scheduler, considers the estimated migration time for each container, based on a developed regressor estimating the total migration time. The solution, then, provides a schedule plan for the microservice-based application, presenting the time to start the migration for each component, aiming to minimize the total migration time.

6.4.3. Advancements Beyond D3.1

As an addition to the implemented solution in D3.1, D3.2 adds the implementation of the two new features described in Section 4, detailed in the following subsections.

6.4.3.1. *Integration of traffic redirection into the existing migration flow*

Integrating traffic redirection into the container migration flow was one of the main challenges that remained unsolved in the previous project. While container migration itself was functional, redirecting live traffic to a migrated workload in a controlled manner was not addressed. This project resolves this limitation by integrating Istio-based traffic management into the migration process. The migration flow assumes that applications are deployed in Istio enabled namespaces. As a result, application pods are automatically injected with an Istio sidecar proxy. During migration, the container is restored into another Istio enabled namespace in the target cluster, ensuring that the sidecar is created alongside the restored container. Once the migration is completed, the system waits until the restored container reaches a ready state before applying traffic redirection.

Traffic redirection is implemented using Istio *DestinationRule* and *VirtualService* resources, which must be defined prior to migration. The Destination Rule specifies service subsets based on application version labels, while the *VirtualService* controls how traffic is routed between these subsets. After migration, traffic redirection is performed by issuing a patch request to the *VirtualService*, updating the routing rules to point to the migrated instance. An example of a simplified *VirtualService* configuration is shown in Figure 39.

```

1 apiVersion: networking.istio.io/v1beta1
2 kind: VirtualService
3 spec:
4   http:
5     - route:
6       - destination:
7         host: demo-app
8         subset: v1
9         weight: 100

```

Figure 39: simplified VirtualService configuration

After restoring the container in the target cluster, the application version label is updated from v1 to v2. Traffic redirection is then achieved by modifying the routing rule to forward traffic to the v2 subset. This approach allows traffic to be switched deterministically and with minimal downtime once the migrated workload is read.

6.4.3.2. Traffic monitoring using Prometheus and Kiali

Prometheus and Kiali are used to observe and validate traffic behavior during and after container migration. Prometheus collects time series data from services and infrastructure components, while Kiali provides a visual interface for observing and analyzing Istio service mesh traffic. Together, they offer both quantitative metrics and an intuitive overview of service interactions within the mesh.

Using Prometheus and Kiali is particularly useful in the context of traffic redirection and mirroring. They provide visibility into where traffic is routed, how requests are distributed across services and clusters, and whether migrated workloads receive traffic as expected. This makes it possible to validate the correctness of traffic management decisions without relying solely on application level logs.

Kiali user interface:

Kiali provides a graphical representation of the Istio service mesh, visualizing services, workloads, and their interactions as a directed graph. This view helps to understand the current mesh topology and to observe traffic flows between services and across cluster boundaries. During migration, Kiali is used to verify that traffic is redirected or mirrored correctly and that the migrated workload is properly integrated into the mesh. Figure 40 shows the Kiali traffic graph, illustrating traffic flowing to workloads in both Cluster 1 and the SEV-SNP cluster.

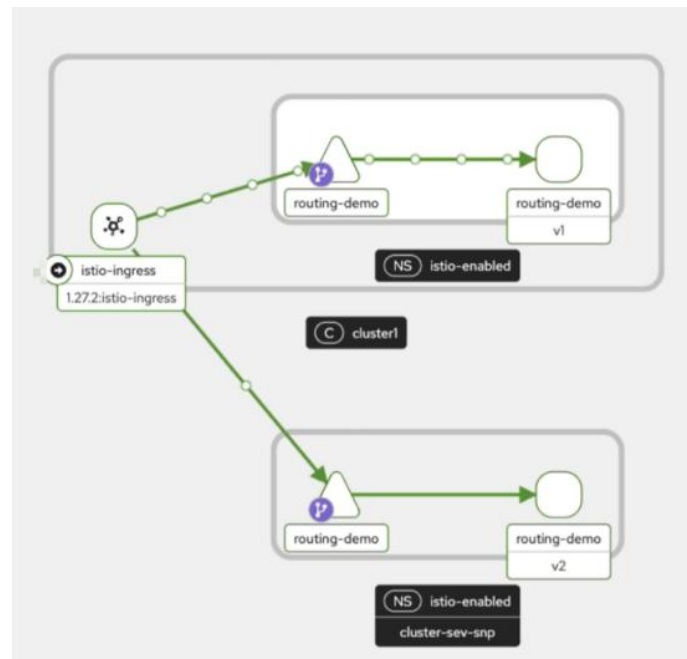


Figure 40 Kiali traffic visualization during live migration

Prometheus federation:

A limitation of Kiali is that it can only read metrics from a single Prometheus instance. In a multi cluster setup, this requires each cluster to run its own local Prometheus instance while exposing aggregated metrics to a central Prometheus server. This approach is commonly referred to as Prometheus federation. In this project, the core cluster hosts the primary Prometheus instance, which acts as the central metrics store. Other clusters, such as the SEV-SNP cluster for TEE confidential CNFs, runs a local Prometheus instance that collects cluster local metrics and exposes them to the primary Prometheus through federation. This setup allows Kiali to visualize traffic across clusters while relying on a single Prometheus backend.

6.5. DFE/WAI offloading

This subsection provides the technical details, from the algorithmic point of view, of the DFE and WAI offloading solutions presented in Section 3.2.

6.5.1. WAI and DFE for P4 switch DNN

6.5.1.1. Problem definition

Current programmable switches face two key limitations: (1) restricted parallelization and (2) hardware backends that lack full P4 language support, leading to suboptimal latency performance. For example, software-based P4 DNN implementations achieve intra-switch latencies nearly an order of magnitude higher than standard pipelines like forwarding and

steering. This stems from hardware vendors prioritizing fast memory access for lookup tables over computational resources like ALUs. While fully in-network ML processing is conceptually possible, conventional methods struggle to deploy DNNs within the data plane. DNNs rely on multiply-accumulate operations and nonlinear functions, but programmable ASICs in commercial P4 switches lack both floating-point and integer arithmetic support. Although DNNs can tolerate low-precision inputs when trained appropriately, the inability to perform even integer-based multiply-accumulate operations prevents their direct deployment. While feature extraction can be efficiently implemented on a programmable ASIC backend, adapting the DNN function requires a complete remapping to bypass ALU-dependent operations.

To enhance WAI robustness against adversarial attacks while preserving compatibility with LUT-based in-switch inference, three complementary strategies are adopted. First, the perturbation space is constrained through input quantization. By reducing the numerical precision of input features, the granularity available for adversarial manipulation is limited, making fine-grained optimization less effective. Second, the classifier's decision boundary is reshaped through adversarial training. By explicitly incorporating adversarially generated samples into the training dataset, the model learns to internalize adversarial directions and adjust its loss landscape accordingly. Third, controlled stochasticity is introduced into the inference process. By injecting pseudo-random perturbations inside the data plane prior to lookup, the attacker's optimization objective is transformed from a deterministic problem into one defined over an expectation, thereby increasing attack complexity and reducing transferability. All three strategies are designed to remain fully compatible with programmable match-action pipelines and LUT-based DNN distillation.

To enhance WAI robustness against adversarial attacks while preserving compatibility with LUT-based in-switch inference, three complementary strategies are adopted. First, the perturbation space is constrained through input quantization. By reducing the numerical precision of input features, the granularity available for adversarial manipulation is limited, making fine-grained optimization less effective. Second, the classifier's decision boundary is reshaped through adversarial training. By explicitly incorporating adversarially generated samples into the training dataset, the model learns to internalize adversarial directions and adjust its loss landscape accordingly. Third, controlled stochasticity is introduced into the inference process. By injecting pseudo-random perturbations inside the data plane prior to lookup, the attacker's optimization objective is transformed from a deterministic problem into one defined over an expectation, thereby increasing attack complexity and reducing transferability. All three strategies are designed to remain fully compatible with programmable match-action pipelines and LUT-based DNN distillation.

Let x denote the input feature vector extracted from a network packet, y its ground-truth label, and $L(\cdot)$ the classifier loss function.

In an untargeted adversarial attack, the attacker seeks a perturbation δ , subject to a constraint $\|\delta\| < \epsilon$, that maximizes the classification loss:

$$\max_{\|\delta\| < \epsilon} L(x + \delta, y)$$

In a targeted attack, the objective becomes:

$$\min_{\|\delta\| < \epsilon} L(x + \delta, y_{target})$$

In white-box settings, the attacker can exploit full knowledge of the model parameters and apply gradient-based optimization techniques such as the Elastic-Net Adaptive Attack (EAD), which minimizes a composite distortion metric of the form:

$$\beta \|\delta\|_1 + \|\delta\|_2^2$$

In black-box scenarios, decision-based attacks such as the Boundary Attack approximate the classifier's decision boundary through iterative querying.

The baseline 8-bit LUT-distilled model achieves high accuracy on clean traffic. However, under white-box EAD attacks, classification accuracy rapidly decreases as perturbation magnitude increases, stabilizing around 40% at higher distortion levels. Under black-box Boundary attacks, accuracy drops from above 85% to approximately 50%. These results demonstrate that LUT distillation alone does not provide intrinsic adversarial robustness.

The key challenge is therefore to modify the attacker's optimization landscape while maintaining compatibility with data-plane processing constraints.

6.5.1.2. *Developed solution*

To enable DNN deployment in hardware pipelines lacking arithmetic capabilities, we propose distilling a trained, integer-quantized DNN into a LUT. This approach transforms inference into a match-action operation by encoding integer inputs as LUT addresses and storing precomputed outputs for all possible input combinations. As illustrated in Figure 41, a network with two inputs of n and m bits forms a compound address of $n+m$ bits, generating 2^{n+m} LUT entries. This method extends to multi-input networks by concatenating all inputs into a single key. The process is lossless and preserves model accuracy. Compared to existing table-based quantization strategies available in the literature, our approach embeds the entire DNN within a LUT rather than merely accelerating operations. However, it presents challenges: (i) memory usage grows exponentially with input bit-width, (ii) inputs must be integer-encoded (with floating-point representation restricted to hidden layers), and (iii) the LUT scales linearly with the number of output variables.

Memory constraints affect key size and the number of entries that can be defined, impacting scalability.

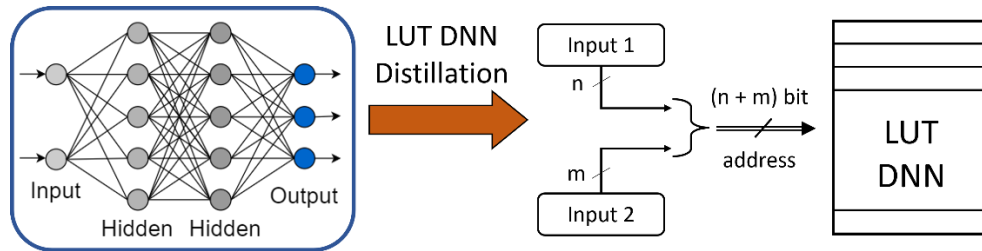


Figure 41: DNN to lookup distillation method

Despite these limitations, this method enables DNN deployment in P4 switches without restrictions on model complexity or type. It generalizes to other ML algorithms if inputs and outputs are quantized accordingly. In networking scenarios, where data types are less complex than images, quantization constraints are more manageable. Additionally, larger DNNs can be trained to enhance accuracy without affecting inference speed, as lookup time remains constant. Retraining is also feasible—updated models can be distilled and deployed by simply refreshing the LUT entries.

Transforming neural networks into LUTs presents a significant challenge due to the precision and wide range of 32-bit floating point numbers. This large input-output space makes it impractical to create LUTs directly, as the memory and computational requirements become infeasible. Quantization addresses this issue by reducing the precision of inputs and outputs, mapping continuous values to a smaller, discrete set. This significantly reduces the LUT size and makes LUT-based computation more viable, especially in resource-constrained environments such as embedded systems or programmable network devices.

To maintain accuracy under limited precision, we adopt Quantization -Aware Training (QAT), where the impact of quantization is accounted for during training. In this approach, inputs are quantized using a dedicated quantizer q_{input} , while weights remain in full precision. The quantized layer computes the activation y as follows: $y = \sigma(f(w, q_{\text{input}}(x)) + b)$

where w represents the weights, x is the input, f is the layer operation, σ is the activation function, and b is the bias. We employ DoReFa quantizers for their flexibility in specifying bit-widths, making them well-suited for hardware-efficient implementations of neural networks.

To include adversarial attacks in the detection, we report the three developed solutions, along with evaluation results referred to the accuracy of the proposed models against adversarial attacks.

Reduced-Precision Defense.

The first solution reduces input quantization precision from 8 bits to 6, 4, and 2 bits. This discretization reduces the effective perturbation search space, limiting the ability of adversarial optimization to exploit small feature variations.

Experimental results show that while the baseline 8-bit model degrades to approximately 40% accuracy under strong EAD perturbations, the 6-bit configuration maintains roughly 60–70% accuracy. The 4-bit model stabilizes near 75–80%, and the 2-bit model remains close to 80% even at high perturbation magnitudes. Similar improvements are observed under black-box Boundary attacks, where reduced-precision models significantly outperform the baseline. Although extreme quantization slightly reduces clean accuracy, it substantially enhances adversarial robustness.

Adversarial Training Defense

The second solution introduces an additional classifier trained on both legitimate and adversarially generated samples. This reshapes the effective loss surface by incorporating adversarial directions directly into the learning process.

In gray-box scenarios, the adversarially trained model maintains approximately 99% accuracy across perturbation magnitudes. Even in white-box conditions, accuracy remains between 85% and 91%, significantly higher than the approximately 40% plateau observed in the baseline model. Under black-box Boundary attacks, robustness exceeds 98% across perturbation levels. This approach provides the strongest robustness improvement among the evaluated strategies.

Embedded Randomization Defense

The third solution introduces pseudo-random perturbations ρ_r into the feature representation before LUT lookup. This transforms the attacker's objective into an expectation-based optimization problem. For untargeted attacks, the objective becomes:

$$\mathbb{E}_{\rho_r}[L(x + \delta + \rho_r, y)]$$

and for targeted attacks:

$$\mathbb{E}_{\rho_r}[L(x + \delta + \rho_r, y_{target})]$$

By forcing the attacker to optimize over stochastic behavior, the difficulty and computational cost of adversarial generation increase significantly.

With embedded randomization enabled, accuracy under white-box EAD attacks stabilizes between 86% and 89%, and exceeds 90% in gray-box scenarios. Under black-box Boundary attacks, accuracy remains around 87–88%, compared to approximately 50% in the baseline case.

While clean accuracy is slightly reduced, robustness improvements are consistent and substantial across all threat models.

6.5.2. DFE and WAI in DPU-based mitigation

6.5.2.1. *Problem definition*

A prevalent method used in DDoS attacks is the TCP SYN flood attack. In this type of attack, the target server is overwhelmed with many TCP SYN (synchronize) packets. The server, following the standard TCP handshake process, allocates resources and responds with SYN-ACK (synchronize-acknowledgment) packets. However, the attackers deliberately do not send the final ACK packet, preventing the connection from being completed. As a result, the server's TCP session table becomes exhausted, rendering it unable to process legitimate connection requests and effectively denying service to authorized users.

To address DDoS attacks, various detection and mitigation techniques have been proposed, including rule-based, signature-based, commercial solutions, ML, anomaly-based, and flow-based approaches. Despite their effectiveness, these methods often introduce non-negligible latency in detecting and responding to attacks. This delay can be particularly problematic when attack rates are high, as a substantial portion of the attack traffic may remain unmitigated for a critical period, worsening the impact on the target system.

A first DFE/WAI design and implementation fully offloaded in a DPU using DOCA libraries faces the complexity of minimizing the number of ARM core processing calls (which are relatively slow), while maximizing the number of operations that can be done using hardware accelerators (high speed).

6.5.2.2. *Developed solution*

To manage the stateless nature of hardware pipes, a stateful control logic is deployed on the ARM cores. Written in C, it dynamically orchestrates and updates hardware pipes by inserting, modifying, and removing entries as needed. This approach ensures that attack detection remains adaptive while leveraging hardware acceleration to minimize latency. Metadata tagging is used to track packet flow, enabling real-time decision-making regarding traffic handling.

The DDoS mitigator logic is deployed on the ARM cores and written in C programming language, including all the available C libraries to properly fill, update, query, and finally remove entries in the pipes. This distribution of tasks between the main two components of the fully offloaded system, namely the ARM cores and the offloading hardware, is explicitly described in Figure 42.

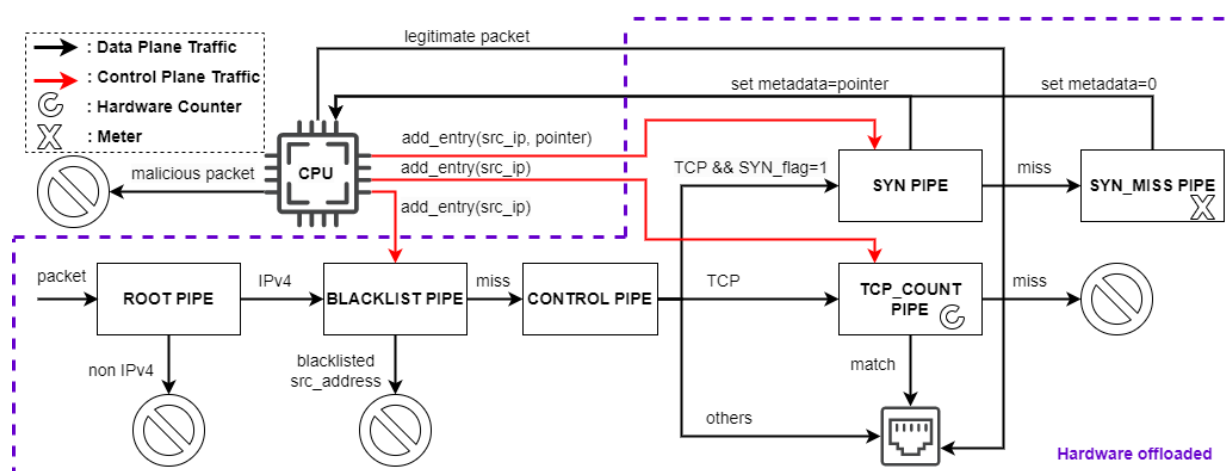


Figure 42: DOCA-based offloaded DDoS mitigator

Packets entering the DPU through port P0 are first processed by SF1 and the root pipe, which filters out non-IPv4 traffic. The blacklist pipe immediately drops packets from known malicious sources. The control pipe evaluates remaining traffic, directing TCP packets to the appropriate processing path. Non-TCP packets are forwarded to SF2 to reach the host.

If a packet has the SYN flag set and remains under the rate limiter threshold, it is processed in the SYN pipe. Otherwise, it is sent to the SYN_MISS pipe for further analysis. The TCP_COUNT pipe updates per-source statistics, tracking the number of legitimate TCP packets. If no matching entry is found, the packet is dropped to prevent unauthorized access. The system employs a rate limiter to regulate the number of SYN packets sent to the host, preventing CPU overload.

Additionally, all outgoing traffic from the host is forwarded through a dedicated hardware pipe from SF2 to SF1, reducing processing overhead and ensuring efficient data flow.

A multi-threaded application on the ARM cores handles real-time SYN flood detection and mitigation. The pseudo-code of the application is detailed in Figure 43. Each thread polls assigned RX queues via DPDK libraries, processing only packets requiring in-depth inspection. If a packet comes from a previously unmonitored source, an IP record is created to track SYN counts and register the entry ID in the TCP_COUNT and SYN pipes. Metadata values are updated to facilitate efficient lookup and tracking.

For already monitored sources, the system retrieves statistics and recalculates the SYN-to-TCP ratio. If the ratio exceeds a predefined threshold, the source is blacklisted, ensuring subsequent packets are dropped at the hardware level without involving the CPU. The system continuously updates and removes outdated entries to free resources for new traffic.

By efficiently distributing tasks between hardware pipes and ARM cores, this architecture maintains high throughput and low latency, effectively mitigating large-scale SYN flood attacks in real time.

The experimental setup consists of multiple traffic generator servers, a high-capacity switching infrastructure, and a target server equipped with a DPU hosting the DFE functionality. Legitimate and malicious traffic sources are connected through a 100 Gb/s data center switch, which forwards traffic to the DPU-enabled server. The DPU operates inline between the external network and the protected host, enabling decentralized observation of incoming traffic and real-time feature extraction at the network edge.

Traffic Generation and Attack Scenarios

Two main traffic classes were considered to validate the DFE behavior. Legitimate traffic was generated by a Linux server emulating multiple independent clients, each assigned a distinct IP address and network namespace. Application-level traffic was produced using standard benchmarking tools to generate sustained and realistic connection patterns. DDoS traffic was generated by a separate server using a high-performance traffic generator capable of producing high-rate TCP SYN flood attacks. This allowed controlled stress testing of the DFE component under increasing attack intensities while maintaining stable legitimate traffic flows.

Measurement and Monitoring Approach

A key requirement of the testbed was the ability to accurately measure the traffic processed and filtered by the DPU. To this end, the switching infrastructure mirrored both ingress and egress traffic to a dedicated monitoring server. Packet capture was performed using DPDK-based tools, enabling lossless capture of high-rate traffic that cannot be reliably observed using standard kernel-based tools. On the DPU-enabled server, hardware-assisted mirroring was used to duplicate post-filtering traffic toward a dedicated host interface. This approach allowed precise evaluation of which packets were blocked by the DFE logic, and which were delivered to the host. All mirroring operations were handled in hardware and did not introduce measurable performance penalties during the experiments.

DPU Configuration for DFE Validation

The DPU was configured to dedicate most of its ARM cores exclusively to the execution of the DFE application, minimizing interference from other system processes. Feature extraction, traffic classification, and mitigation decisions were performed directly on the DPU using hardware-accelerated flow tables and counters. This configuration ensured that the evaluation reflected the decentralized nature of the DFE approach, where each DPU independently extracts features and enforces mitigation policies at wire speed. The results obtained from this testbed are used

to assess the effectiveness, scalability, and performance of the decentralized DFE solution. In the evaluated scenario, the attack was generated by approximately 590,000 sources, with a TCP SYN credit set to 10 packets per source. This configuration requires the DPU to process 11 TCP SYN packets per source before issuing a blocking rule, resulting in a total of approximately 6.49 million packets that must be handled to fully mitigate the attack. The experimental results show that the attack is completely blocked within 2.84 seconds. This implies that the DOCA-based DFE application sustained an average processing rate of approximately 2.29 million packets per second. Although the observed throughput is lower, this behavior is explained by temporary congestion occurring at the SF1 interface, where polling threads are unable to immediately forward processed packets. This effect is likely related to the large number of flow rules dynamically inserted into the DPU flow tables during the initial phase of the attack, which can transiently impact forwarding resources. A deeper investigation of this behavior is considered outside the scope of the present deliverable and is left for future analysis.

Algorithm 1 Application Logic pseudo-code

```

1: while true do
2:   poll RX queue for RX_packets
3:   if #RX_packets  $\neq$  0 then
4:     for packet in RX_packets do
5:       parse the packet
6:       if meta = 0 then
7:         start IP monitoring
8:         add packet to TX_packets
9:       else
10:        retrieve IP record from meta
11:        increase IP sent #SYN
12:        query #PKTs HW counter
13:        if #SYN/#PKTs  $\geq$  TH then
14:          blacklist the IP
15:          drop the packet
16:        else
17:          add packet to TX_packets
18:        end if
19:      end if
20:    end for
21:    send TX_packets to TX_queue
22:  end if
23:  manage entry aging
24: end while

```

Figure 43: Offloaded DDoS mitigator algorithm

6.5.3. Advancement beyond D3.1

The developed DFE solution validates decentralized feature extraction and mitigation directly on DPUs through a dedicated high-speed testbed emulating realistic DDoS scenarios. In this architecture, the DPU operates inline between the external network and the protected host, extracting traffic features and enforcing mitigation policies independently of centralized analysis

or host-based packet processing. Experiments combining legitimate traffic with large-scale TCP SYN flood attacks show that the DPU can process approximately 6.49 million packets generated by about 590,000 attack sources and fully mitigate the attack within 2.84 seconds, sustaining an average processing rate of around 2.29 million packets per second. These results confirm that decentralized, hardware-assisted feature extraction can effectively detect and mitigate attacks at the network edge.

For the WAI component, three complementary strategies were developed to improve robustness of in-switch ML inference against adversarial attacks. Reduced-precision defense limits the perturbation space by lowering input quantization, adversarial training improves resilience by incorporating adversarial samples during training, and embedded randomization introduces stochastic perturbations before LUT-based inference. Experimental results show consistent robustness improvements across white-box, gray-box, and black-box attack scenarios while remaining compatible with match-action data-plane execution.

6.6. Data plane ML

Our proposed model employs as-soon-as-possible in-network inference and online learning to enable accurate, low-latency network attack detection. Additionally, data plane program disaggregation is used to ensure that switches retain sufficient resources for other essential tasks, such as packet routing.

6.6.1. Feature Extraction

6.6.1.1. *Problem definition*

The classification of network flows—determining whether they represent an attack or benign traffic—is based on various features derived from individual packets within the flows. These features include parameters such as current and maximum packet length, the number of times different TCP flags were set, and the source and destination ports. In order to compute some of these features, the system requires persistent storage of flow-specific data.

6.6.1.2. *Developed solution*

This persistence is implemented using registers indexed by the hash of the flow’s identifier (5-tuple). However, this approach introduces the risk of hash collisions, so that different flows can inadvertently share the same register locations, resulting in data corruption.

To mitigate this risk, we implemented flow timeouts: flows that have been inactive for at least 30 seconds are considered terminated, allowing their register slots to be reset when a new flow is assigned.

6.6.2. Model Training and Online Learning

6.6.2.1. *Problem definition*

Pre-trained models and pre-initialized datasets often present significant challenges in the context of programmable network hardware. Large models require significant memory and computational resources, which exceed the capabilities of network hardware. Conversely, smaller or more compact models may lack the necessary accuracy to provide reliable results, making them ineffective for critical tasks such as security applications or traffic optimization.

This creates a trade-off between resource efficiency and classification precision, with neither extreme providing a fully viable solution for real-time, high-performance network operations.

6.6.2.2. *Developed solution*

Our proposed model operates entirely through online learning, eliminating the need for pre-trained models or pre-initialized datasets. The control plane is responsible for training models using sampled features received from programmable switches, the architecture and component details can be seen on Figure 19 and Figure 27.

These switches randomly select a fraction of their flows for monitoring and forwarding the extracted features—and classification results, if available—to the control plane. Flow selection is based on a combination of the flow identifier and a random number initialized at switch startup.

To support real-time inference, flow features must be collected after each received packet. This ensures that separate classifier models can be trained for different flow lengths.

The controller caches flow features and classification results received from the switches for a few minutes. These features are then forwarded in batches to an external IDS, which provides highly accurate flow labels. Thus, the proposed online learning approach relies on delayed supervision from an external IDS, rather than pre-labeled datasets or human feedback. The external IDS performs network attack detection with high accuracy, though at the cost of reduced throughput and increased latency compared to in-network approaches. The labels provided by the IDS serve as the ground truth, while the in-network classification results act as predicted labels. These true and predicted labels are used to assess the accuracy or F1-score of the in-network classifier.

The controller regularly trains new random forest models using the sampled features and true labels. The training approach follows a methodology, where a separate random forest is trained for each subflow (i.e., the first N packets of a flow). A decision from a random forest is only accepted if its certainty exceeds a predefined threshold, and only sufficiently accurate random forests are incorporated into the final model. This approach enables early classification for some flows after just a few packets, while others may require more packets to achieve the necessary confidence level. Once a new model is trained, it undergoes an evaluation phase, where it

classifies the collected features and its predictions are compared to the true labels. If the new model surpasses the performance of the currently deployed in-network model by a predefined threshold (e.g., a 1% improvement in accuracy), it is deployed to the switches, replacing the previous model.

6.6.3. Model Encoding for Match-Action Tables

6.6.3.1. *Problem definition*

Software pipelines (eBPF) cannot perform efficient range matching, requiring long table dependency chains to evaluate decision trees. On the other hand, hardware pipelines (Tofino) support range matching but strictly limit the number of range-match keys available per stage. Furthermore, real-time inference must operate at line rate while supporting flows of varying lengths, requiring the system to select the correct ML model based on the flow's current packet count.

6.6.3.2. *Developed solution*

To address these constraints, we employ target-aware encoding methods. For eBPF targets, each depth level of a decision tree is mapped to a separate match-action table. These table entries represent tree nodes, while the actions perform threshold comparisons to either determine a final verdict or forward processing to the next table in the chain.

In contrast, Tofino switches support range match keys but do not allow long sequences of match-action tables where earlier results are required before executing subsequent tables. Consequently, we applied the encoding method from prior research, where each decision tree is embedded into a single match-action table. Each feature is assigned a range match key, and each decision tree leaf corresponds to a table entry that specifies the required feature ranges for classification. However, this approach limits the number of features that can be used, as Tofino switches only support a limited number of range match keys per table.

Since our approach relies on as-soon-as-possible inference, separate random forests must be trained and encoded for different flow lengths. To facilitate this, each match-action table includes an additional key that identifies which random forest to use. This identifier is determined using a separate table that maps the flow's packet count to the corresponding random forest identifier.

6.6.4. Model Disaggregation

6.6.4.1. *Problem definition*

The accuracy of network attack detection can be improved by increasing the number of decision trees within each random forest or by extending the depth of decision trees. However, both approaches increase the model's resource requirements, such as memory. Since switches must

also allocate resources for other tasks like packet forwarding, it is crucial to limit the resource consumption of in-network attack detection.

6.6.4.2. *Developed solution*

If flows traverse multiple switches within the internal network, the per-switch resource requirements can be reduced by partitioning the P4 program and distributing its components across multiple switches. In our random forest-based approach, this means embedding different decision trees of the forest in different switches. However, certain components must be present in all switches—particularly those at the network edge—so that flow features can be computed or received from upstream switches.

Determining which decision trees should be deployed on which switches is beyond the scope of this work. While decision trees within a random forest can operate independently, edge switches need access to the verdicts of all decision trees to compute the final classification result. To facilitate this, an extra header is added to packets upon entering the internal network and removed when exiting. This header stores information about pending decision trees and already computed verdicts.

6.6.5. Advancements beyond D3.1

The fundamental problem definitions and developed solutions remain the same as in D3.1. This section introduces minor refinements and technical justifications regarding eBPF and Tofino hardware constraints, with no significant updates to the underlying model.

6.7. RAN security-performance balancer

This section addresses the challenge of managing user-plane cryptographic processing in disaggregated 5G architectures with CU-CP/CU-UP split. Due to the different computational characteristics of 3GPP ciphering algorithms and their interaction with heterogeneous hardware acceleration, current CU-UP selection—being agnostic to the negotiated algorithm—leads to inefficient resource usage and performance variability. To address this, the section introduces a machine learning–based solution deployed in the Near-RT RIC, which forecasts algorithm-specific traffic demand and derives corresponding compute requirements. The following subsections describe the problem formulation and the proposed predictive approach based on gradient boosting techniques.

6.7.1. Problem definition

In disaggregated 5G architectures with CU-CP/CU-UP split, user-plane encryption is performed at the CU-UP using the ciphering algorithm negotiated during the RRC Security Mode Command. Different 3GPP ciphering algorithms (e.g., AES, SNOW, ZUC) impose significantly different

computational loads and interact differently with available hardware acceleration mechanisms (e.g., AES-NI, crypto engines). In current implementations, CU-UP selection is typically agnostic to the negotiated ciphering algorithm, leading to heterogeneous algorithm traffic being processed within the same CU-UP instance. This can result in inefficient CPU utilization, reduced cache efficiency, increased context-switching overhead, and unpredictable performance under high load, especially when software-only and hardware-accelerated algorithms coexist. The core problem addressed by the proposed solution is the lack of algorithm-aware user-plane traffic steering, which prevents optimal alignment between ciphering workload characteristics and underlying compute capabilities.

6.7.2. Developed Solution

The ML method used in the proposed solution is based on gradient boosting decision trees, e.g. LightGBM, deployed as an inference component in the Near-RT RIC to forecast ciphering-algorithm-specific traffic demand. The objective of the model is not to select the ciphering algorithm itself (which remains governed by 3GPP security procedures), but to predict the future distribution of traffic across negotiated algorithms (e.g., AES vs SNOW) and the resulting compute demand per CU-UP pool.

The forecasting problem can be formulated as a supervised regression task with time-series characteristics. Two related targets are modeled: (i) total PDCP user-plane throughput and (ii) the proportion or absolute throughput associated with each ciphering algorithm. From these predictions, the system derives expected per-algorithm compute requirements. LightGBM is preferred because it handles structured, tabular telecom KPIs effectively, supports nonlinear feature interactions, performs well with moderate data volumes typical of RAN deployments, and allows efficient inference suitable for Near-RT operation.

Input features include lagged throughput values, historical AES/SNOW ratios, hour-of-day and day-of-week indicators, RAT distribution (e.g., LTE vs NR share), attach rates, mobility indicators, and recent CU-UP CPU utilization. These features capture both temporal patterns (diurnal behavior, traffic bursts) and structural drivers of algorithm mix variation (e.g., LTE fallback events increasing SNOW share). The model is trained offline in the Non-RT RIC or SMO using historical measurements collected via E2 interfaces, and the trained model artifact is periodically deployed to the Near-RT RIC for inference.

6.7.3. Advancement beyond D3.1

The problem definition and developed solution are new in this deliverable and have been proposed to support the new Security-performance balancer design as described in Section 3.4.

6.8. LLM-based IDS

In recent developments within deep learning, self-supervised learning (SSL) has gained prominence as a powerful alternative to traditional supervised approaches, particularly through the adoption of Transformer-based architectures and large language models (LLMs). These models demonstrate strong capabilities in extracting rich and generalizable representations from unlabeled data, thereby reducing reliance on extensive manually annotated datasets. In this context, we introduce an IDS built upon the BERT architecture, a bidirectional Transformer model originally tailored for natural language understanding tasks. The BERT model's encoder-centric design, coupled with its bidirectional self-attention mechanism, enables it to capture complex contextual relationships within structured, sequential inputs, making it especially well-suited for traffic classification tasks in network security. Unlike generative LLMs such as GPT, which are optimized for text generation, BERT's architecture inherently supports discriminative learning objectives, offering a more suitable foundation for accurate and efficient real-time threat detection.

6.8.1. Packet-token embedding optimization

6.8.1.1. Problem Definition

LLMs and Transformer-based models in general are powerful tools for processing sequential data. However, as the number of tokens in a sequence increases, so does the training and inference time, as transformer attention computation time scales quadratically with the number of tokens. Treating each packet header or the raw bytes of the packet as individual tokens can quickly increase the number of tokens that need to be processed.

6.8.1.2. Developed Solution

To optimise the processing time of our model we employed the packet-token based embedding procedure described in Section 3.5 which significantly reduces the number of tokens that our LLM-based IDS has to process. In addition to this, we implement a time limit as well as a limit to the number of packets to 32 for each flow to keep the packet sequences short. This does not impact the performance of our model.

6.8.2. Contrastive learning and flow augmentation process

6.8.2.1. Problem Definition

As the digital landscape becomes more interconnected, the frequency and severity of zero-day attacks have significantly increased, leading to an urgent need for innovative IDS. Machine Learning-based IDS that learn from the network traffic characteristics and can discern attack patterns from benign traffic offer an advanced solution to traditional signature-based IDS. However, they heavily rely on labeled datasets, and their ability to generalize when encountering

unseen traffic patterns remains a challenge. To provide a generalizable baseline model for intrusion detection we devised a self-supervised contrastive learning process on unlabelled raw packet sequences, as a pretraining task for the encoder stack of the LLM.

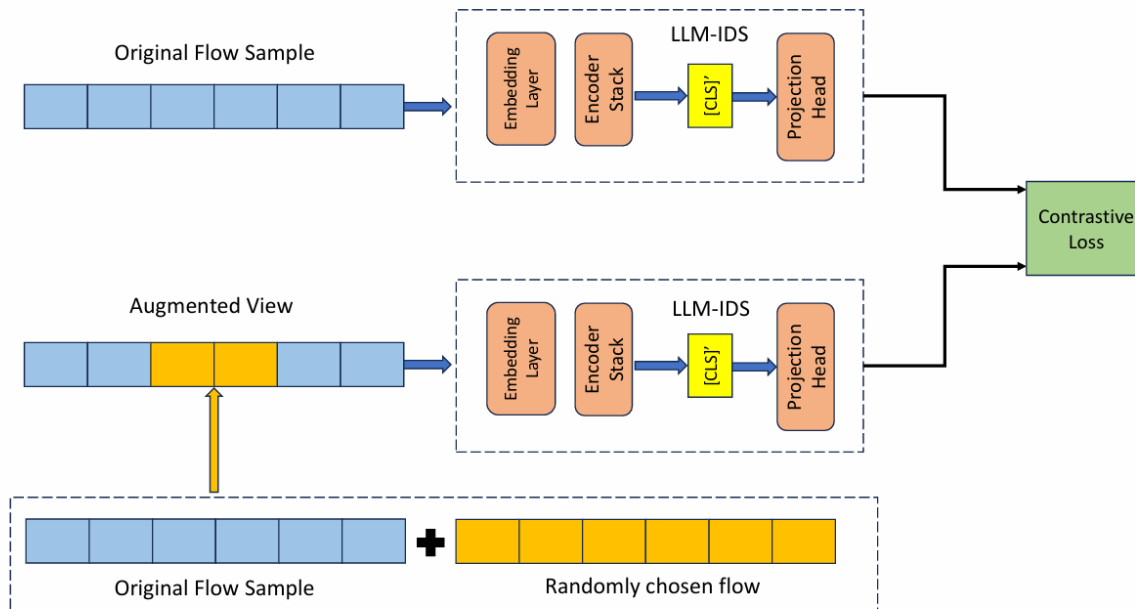


Figure 44: Overview of the contrastive learning and augmentation process

6.8.2.2. Developed Solution

The objective of the contrastive learning task is to learn meaningful representations by bringing the representations of similar flows closer to each other in the embedding space while at the same time pushing apart flow representations that are dissimilar. To achieve this, we have to generate flow samples that are relatively similar to each other for the contrastive learning process. We call these artificially generated samples augmented views of an original flow. To create these augmented views, we devised a simple procedure which is illustrated in Figure 44. To create similar pairs of flow packet sequences we create a new packet sequence by mixing up the packets of an original flow sequence with those of a randomly chosen flow that has the same length as the original one. From the original flow sample, we select a random patch of continuous packets and replace them with the packets that are in the same positions from the randomly chosen flow. The original sample and the augmented view are then forwarded to our LLM-IDS to generate the representation for each sequence in the output of the CLS' token. Finally, the CLS' token is forwarded to a projection head which in our case is a simple 2-layer MLP, which is used to train our model on the contrastive loss objective function, by attracting the views of the original sample and the augmented view, while repelling the representation of the original flow

with all other flows in a mini-batch. After the pretraining procedure, the projection head is discarded and replaced with the classification module as noted in Section 3.5.

6.8.3. Security policy enforcement

6.8.3.1. Problem Definition

In order to enforce appropriate security policies on potentially malicious traffic, it is essential to accurately track and identify packet flows. This requires maintaining stateful information related to the origin and characteristics of each flow. Specifically, flow identifiers such as the 5-tuple (source IP, destination IP, source port, destination port, and protocol) must be extracted and monitored in real time. Without such contextual information, it becomes challenging to reliably associate traffic with specific entities or apply precise mitigation strategies, especially in environments where threats may be dynamic, stealthy, or distributed.

6.8.3.2. Developed Solution

To mitigate security threats, the proposed solution introduces a modular, ML-assisted traffic analysis pipeline integrated with an IDS. This pipeline leverages a traffic classifier, trained using publicly available datasets such as CIC-IDS 2017 [36] and UNSW-NB15 [37], to identify and flag malicious or anomalous flows in real time. The IDS acts as the enforcement component, which cross-references the provided metadata with ongoing traffic patterns and applies predefined security policies.

These policies may include:

- I. Blocking incoming or outgoing traffic from the identified malicious IPs.
- II. Rate limiting traffic to prevent potential DoS attacks.
- III. Generating alerts or logs for further forensic analysis.

The classifier is designed to be lightweight and extensible, enabling real-time inference. To handle possible attacks when the classifier identifies a flow as malicious it proceeds to inform the IDS that monitors network traffic, with the 5-tuple that is associated with the flow, so that the IDS can act against the malicious IPs by implementing a security policy. This solution provides a coordinated and context-aware approach to detecting and mitigating threats, offering better performance, reduced false positives, and greater adaptability to emerging network attack patterns.

6.8.4. Advancements beyond D3.1

The core problem definitions and developed solutions described in this section remain largely unchanged compared to those presented in D3.1. The overall architecture of the LLM-based IDS, including the packet-token embedding mechanism, the contrastive learning framework for self-

supervised pretraining, and the security policy enforcement pipeline, continues to constitute the primary strategy for traffic representation learning and intrusion detection. No major modifications were introduced to the underlying model architecture or to the optimization algorithms governing the training process.

7. Conclusions

The deliverable D3.2 “Secure-by-design orchestration and management & Data plane computation offloading.r2” presented the final integrated view of NATWORK’s secure orchestration and programmable data plane capabilities, outlining both software architecture and final implementation results, ready for integration and validation in the WP6 evaluation and demonstration activities. As 6G architectures grow increasingly complex, dynamic, and performance-critical, NATWORK proposes a cohesive solution that bridges orchestration, security, and intelligent automation across all layers of the network—from extreme edge to core.

Collectively, these technologies form a holistic, modular, and interoperable foundation for secure, sustainable, and scalable 6G network orchestration and management. This work not only addresses today’s limitations in network security, efficiency, and responsiveness but also sets the stage for future extensions through federated intelligence and continuous optimization. The services and solutions, described in the previous sections in their final form, will be integrated in the final Use Cases evaluation and demonstration activities, according to the provisional workplan, to reach their projected technology readiness level. This will enable them to be validated through real-world use cases and testbeds in the final phase of the NATWORK project, meeting their expected KPIs, providing technical maturity, integration feasibility and impact across diverse 6G verticals.

References

- [1] B. M. Xavier, M. Dzaferagic, D. Collins, G. Comarela, M. Martinello, & M. Ruffini, "Machine learning-based early attack detection using open ran intelligent controller." In ICC 2023- IEEE International Conference on Communications, 2023, pp. 1856-1861
- [2] A. Chiejina, B. Kim, K. Chowhdury and V. K. Shah. 2024. System-level Analysis of Adversarial Attacks and Defenses on Intelligence in O-RAN based Cellular Networks. In Proceedings of the 17th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '24). Association for Computing Machinery, New York, NY, USA, 237–247
- [3] I. Sharafaldin, A. H. Lashkari and A. A. Ghorbani, "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization", 4th International Conference on Information Systems Security and Privacy (ICISSP), Portugal, January 2018
- [4] M. AL-Naday, V. Karagiannis, T. De Block and B. Volckaert, "Federated Scheduling of Fog-Native Applications Over Multi-Domain Edge-to-Cloud Ecosystem," *2023 19th International Conference on Network and Service Management (CNSM)*, Niagara Falls, ON, Canada, 2023, pp. 1-7, doi: 10.23919/CNSM59352.2023.10327839.
- [5] S. Ejaz and M. Al-Naday, "FORK: A Kubernetes-Compatible Federated Orchestrator of Fog-Native Applications Over Multi-Domain Edge-to-Cloud Ecosystems," *2024 27th Conference on Innovation in Clouds, Internet and Networks (ICIN)*, Paris, France, 2024, pp. 57-64, doi: 10.1109/ICIN60470.2024.10494435.
- [6] S. Guthula, N. Battula, R. Beltiukov, W. Guo, and A. Gupta, "netFound: Foundation Model for Network Security," arXiv preprint arXiv:2310.17025, 2023.
- [7] X. Han, S. Cui, S. Liu, C. Zhang, B. Jiang, and Z. Lu, "Network intrusion detection based on n-gram frequency and time-aware transformer," *Computers & Security*, vol. 128, p. 103171, 2023.
- [8] M. A. Ferrag, M. Ndhlovu, N. Tihanyi, L. C. Cordeiro, M. Debbah, T. Lestable, and N. S. Thandi, "Revolutionizing Cyber Threat Detection with Large Language Models: A privacy-preserving BERT-based Lightweight Model for IoT/IIoT Devices," *IEEE Access*, 2024.
- [9] J. Devlin, M. W. Chang, K. Lee, & K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding." In Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, vol. 1, pp. 4171-4186.
- [10] E. Paolini, L. de Marinis, D. Scano and F. Paolucci, "In-Line Any-Depth Deep Neural Networks Using P4 Switches," in *IEEE Open Journal of the Communications Society*, vol. 5, pp. 3556-3567, 2024
- [11] G. Zingrillo et al., "Assessing the Robustness of In-Switch Neural Networks Against Adversarial DDoS Attacks," 2025 30th OptoElectronics and Communications Conference

- (OECC) and 2025 International Conference on Photonics in Switching and Computing (PSC), Sapporo, Japan, 2025, pp. 1-3, doi: 10.23919/OECC/PSC62146.2025.11111457.
- [12] S. Hinic, R. A. Bakar, A. Marotta and F. Paolucci, "Wire-speed DDoS Attack Mitigation using Hardware Acceleration of Programmable DPUs," GLOBECOM 2024 - 2024 IEEE Global Communications Conference, Cape Town, South Africa, 2024, pp. 1197-1202, doi: 10.1109/GLOBECOM52923.2024.10901169.
 - [13] C. Lai, L. Shen and K. Feng, "Intelligent Load Balancing and Resource Allocation in O-RAN: A Multi-Agent Multi-Armed Bandit Approach," arXiv preprint arXiv:2303.14355v1, 2023
 - [14] O. Orhan, V. N. Swamy, T. Tetzlaff, M. Nassar, H. Nikopour, S. Talwar, "Connection-Management xAPP for O_RAN RIC: A Graph", arXiv preprint arXiv:2110.07525v2, 2021
 - [15] V. Kasuluru, L. Blanco, C. J. Vaca-Rubio, E. Zeydan, "On the Impact of PRB Load Uncertainty Forecasting for Sustainable Open RAN," arXiv preprint arXiv:2407.14400v1, 2024
 - [16] M. M. H. Qazzaz, A. Salama, M. Hafeez, S. A. R. Zaidi, "Joint Optimisation of Load Balancing and Energy Efficiency for O-RAN Deployments," arXiv preprint arXiv:2509.09343v1, 2025
 - [17] Free5GC, Open Source Core Network Implementation. Online: <https://free5gc.org> (Accessed 2025/06/25)
 - [18] 5G; NR; Radio Resource Control (RRC); Protocol specification (3GPP TS 38.331 version 18.3.0 Release 18) Online: <https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3197> (Accessed 2024/12/27).
 - [19] LTE; 5G; Evolved Universal Terrestrial Radio Access (E-UTRA) and NR; Service Data Adaptation Protocol (SDAP) specification (3GPP TS 37.324 version 18.0.0 Release 18) Online: <https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3282> (Accessed 2024/12/27).
 - [20] 5G; NR; Packet Data Convergence Protocol (PDCP) specification (3GPP TS 38.323 version 18.3.0 Release 18) Online: <https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3196> (Accessed 2024/12/27).
 - [21] 5G; NR; Radio Link Control (RLC) protocol specification (3GPP TS 38.322 version 18.1.0 Release 18) Online: <https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3195> (Accessed: 2024/12/27). 5G; NR; Radio Link Control (RLC) protocol specification (3GPP TS 38.322 version 18.1.0 Release 18) Online: <https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3195> (Accessed: 2024/12/27).

- [22] 5G; NR; Medium Access Control (MAC) protocol specification 3GPP TS 38.321 V18.4.0 (2024, December).
- [23] O-RAN WG4: O-RAN Control, User and Synchronization Plane Specification. (2024, October). Online: <https://specifications.o-ran.org/download?id=738> (Accessed 2024/12/24).
- [24] O-RAN E2 General Aspects and Principles (E2GAP) 6.0, O-RAN.WG3.E2GAP-R004-v06.00 (2024, October).
- [25] Crugnola, F. (2025). Evaluating and simulating end-to-end latency in 5G environments.
- [26] Chow, M. C., & Ma, M. (2022). A secure blockchain-based authentication and key agreement scheme for 3GPP 5G networks. *Sensors*, 22(12), 4525.
- [27] Mohammed, B. A., Al-Shareeda, M. A., Alsadhan, A. A., Al-Mekhlafi, Z. G., Sallam, A. A., Al-Qatab, B. A., ... & Alayba, A. M. (2024). Efficient blockchain-based pseudonym authentication scheme supporting revocation for 5G-assisted vehicular fog computing. *IEEE Access*, 12, 33089-33099.
- [28] Haddad, Z. (2023). Blockchain-enabled anonymous mutual authentication and location privacy-preserving scheme for 5G networks. *Journal of King Saud University-Computer and Information Sciences*, 35(6), 101458.
- [29] O-RAN WG1: O-RAN Decoupled SMO Architecture 3.0. (2024, October). Online: <https://specifications.o-ran.org/download?id=716> (Accessed: 2024/12/23).
- [30] O-RAN A1 Interface: Application Protocol 4.03, O-RAN.WG2.A1AP-R004-v04.03 (Accessed: 2025/05/20).
- [31] O-RAN WG3: O-RAN Near-RT RIC APIs specification 2.0. (2024, June). Online: <https://specifications.o-ran.org/download?id=659>. (Accessed: 2025/05/20).
- [32] R. Schmidt, M. Irazabal, and N. Nikaein (2021). FlexRIC: an SDK for next-generation SD-RANs. In *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '21)*. Association for Computing Machinery, New York, NY, USA, pp. 411–425.
- [33] I. Obeidat, N. Hamadneh, M. Al-kasassbeh, and M. Almseidin. *Intensive Preprocessing of KDD Cup 99 for Network Intrusion Classification Using Machine Learning Techniques*, 2018.
- [34] H. Schmidt, Z. Rejiba, R. Eidenbenz, and K.-T. Förster, “Transparent fault tolerance or stateful applications in Kubernetes with checkpoint/restore,” in *2023 42nd international Symposium on Reliable Distributed Systems (SRDS)*, pp. 129–139, 2023.
- [35] T. Goethals, F. De Turck & B. Volckaert, Near real-time optimization of fog service placement for responsive edge computing. *J Cloud Comp* **9**, 34 (2020). <https://doi.org/10.1186/s13677-020-00180-z>
- [36] I. Sharafaldin, A. H. Lashkari, A. A. Ghorbani et al., “Toward generating a new intrusion detection dataset and intrusion traffic characterization.” *ICISSp*, vol. 1, pp. 108–116, 2018.

- [37] N. Moustafa and J. Slay, “UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set),” in 2015 military communications and information systems conference (MilCIS). IEEE, 2015, pp. 1–6.