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

D3.1 : Secure-by-design orchestration and management & Data plane computation offloading.r1

Lead beneficiary	UEssex	Lead author	Mays AL-Naday, Sumeyya Birtane, Shankha Gupta
Reviewers	Edgardo Montes de Oca (MONT), Sándor LAKI (ELTE)		
Type	R	**Dissemination**	PU
Document version	V1.0	**Due date**	30/06/2025

Co-funded by the European Union

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)	Sumeyya Birtane, Shankha Gupta, Mays AL-Naday (UEssex)
Author(s)	Wissem Soussi, Gökcan Cantali, Gürkan Gür (ZHAW), Péter Vörös, Mohammed Alshawki (ELTE), Konstantinos Pournaras, Kostas Lampropoulos (PNET), Maria B. Safianowska (ISRD), Antonios Lalas, Virgilios Passas, Sarantis Kalafatidis, Nikolaos Makris, Donatos Stavropoulos, Stelios Mpatziakas, Ioanna Kapetanidou, Konstantinos Giapantzis, Georgios Agrafiotis, Thanasis Korakis, Anastasios Drosou (CERTH), Tom Goethals (IMEC), Francesco Paolucci (CNIT)
Reviewers	Edgardo Montes de Oca (MONT), Sándor LAKI (ELTE)
Type	R – Document, Report
Dissemination level	PU – Public
Due date	M18 (30/06/2025)
Submission date	30/06/2025

Document version history

Version	Date	Changes	Contributor (s)
v0.1	02/12/2024	Initial table of contents	Mays AL-Naday, Sumeyya Birtane (UEssex)
v0.2	10/01/2025	Updated ToC and partner assignment	Sumeyya Birtane Mays AL-Naday (UEssex)
v0.3	21/01/2025	Further update to ToC and partner assignment	Sumeyya Birtane (UEssex)
v0.4	04/02/2025	Further update to ToC and partner assignment	Francesco Paolucci (CNIT)
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)
v0.6	20/03/2025	Contribution for Section 2.3, 3.5, 5.2, 6.2, 6.8	Virgilios Passas (CERTH)
v0.7	19/06/2025	Reviewed version	Sàndor Laki (ELTE), Edgardo Montes de Oca (MONT)
v0.75	22/6/2025	Refinement based on review comments	All authors
v0.8	24/06/2025	Near final version	Mays AL-Naday (UEssex)
v0.9	27/06/2025	Quality review	Joachim Schmidt, Leonardo Padial (HES-SO)
v0.95	29/06/2025	Final review and refinements	Antonios Lalas, Virgilios Passas (CERTH) and CERTH team
v1.0	30/06/2025	Final version for submission	Antonios Lalas (CERTH)

Disclaimer

Copyright message

Table of Content

List of acronyms and abbreviations

Abbreviation	Description
5GC	5G Core
AMF	Access and Mobility Management Function
AI	Artificial Intelligence
ATD	Anomaly Traffic Detector
AUSF	Authentication Server Function
CRAN	Cloud Radio Access Network
CTI	Cyber Threat Intelligence
DFE	Decentralized Feature Extraction
DoS	Denial of Service
EMA	exponential moving average
FQDN	Fully Qualified Domain Name
GTP	GPRS Tunnelling Protocol
IDS	Intrusion Detection System
KPI	Key Performance Indicator
LLM	Large Language Model
ML	Machine Learning
MLP	Multi-layer Perceptron
MTD	Moving Target Defense
NFV	Network Functions Virtualization
NSD	Network Service Descriptor
NWDAF	Network Data Analytics Function
OAI	OpenAirInterface
O-RAN	Open Radio Access Network
OSL	OpenSLice
PRB	Physical Resource Block
RAN	Radio Access Network
RC	RAN Control
RIC	RAN Intelligent Controller
RRC	Radio Resource Control
RT-RIC	Real Time RAN Intelligent Controller
SD	Slice Differentiator
SDN	Software-Defined Networking
SLA	Service Level Agreement
SM	Service Model
SMF	Session Management Function
S-NSSAI	Single Network Slice Selection Assistance Information
SSL	Self-Supervised Learning
SST	Slice Service Type

Abbreviation	Description
STIX	Structured Threat Information eXpression
TAXII	Trusted Automated eXchange of Intelligence Information
TCP	Transmission Control Protocol
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

List of tables

Executive summary

The purpose of this deliverable is to present the design principles, architectural components, and initial implementation steps 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 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.

Early implementation strategies 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 testing activities in NATWORK.

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. 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.

The document is structured to provide a clear progression from design to implementation, as follows: The Software Design: Orchestrator(s)the design of orchestrators deployed at different layers of the 6G continuum, covering orchestration at the extreme edge, Cloud Radio Access Network (CRAN), and core edge-cloud continuum enabling 6G core. The Data Plane Computation Offloading Design section explores strategies to optimise latency and energy use, covering offloading classification, Wirespeed AI (WAI) and Decentralised Feature Extraction (DFE), in-network machine learning models, and a Radio Access Network (RAN) security-performance balancer. The Software Design: Orchestration Support Systems section presents security enhancements, including a Moving Target Defence (MTD) framework, decentralised CTI sharing, AI-based behavioural analysis, and a security-performance balancing. The Implementation section describes the deployment of the design components, optimisation algorithms and testbed validations, ensuring secure and efficient 6G operations. Finally, the Conclusions section reflects on the project's strategic orientation and outlines expectations for future milestones in scalable 6G deployments.

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 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 implementation progress 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.
- **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 D3.1 *"Secure-by-design orchestration and management & Data plane computation offloading"* 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 D3.1 "Secure-by-design orchestration and management & Data Plane Computation Offloading" 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.

2. Software Design: Orchestrator(s)

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 NATWORK 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 NATWORK orchestrator services. Conversely, innovations and service orchestration strategies emerging from NATWORK 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.

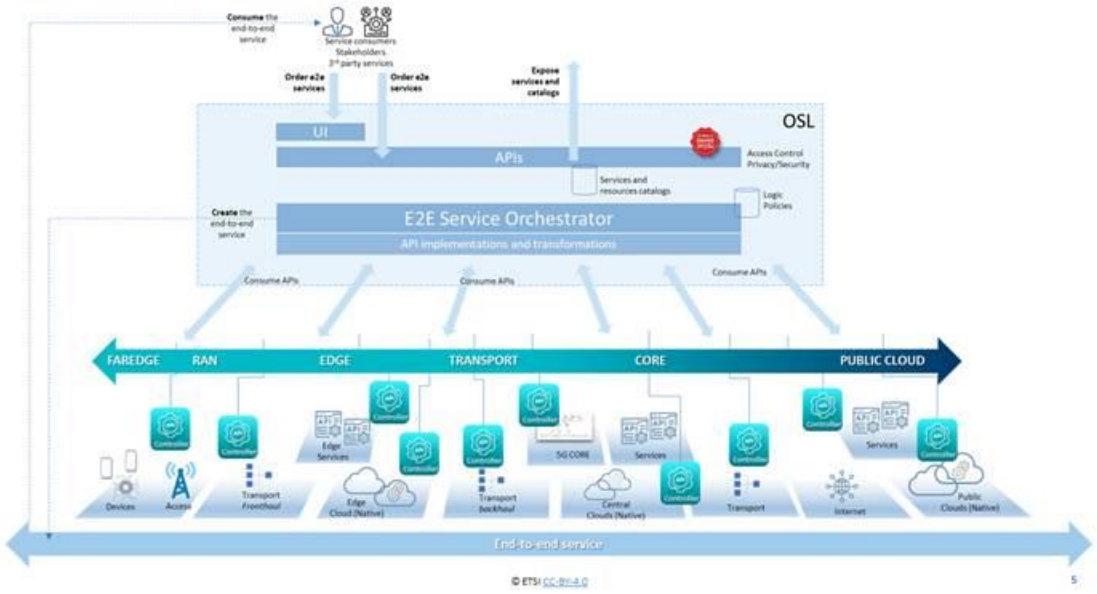

Figure 1: OSL Reference Architecture

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.

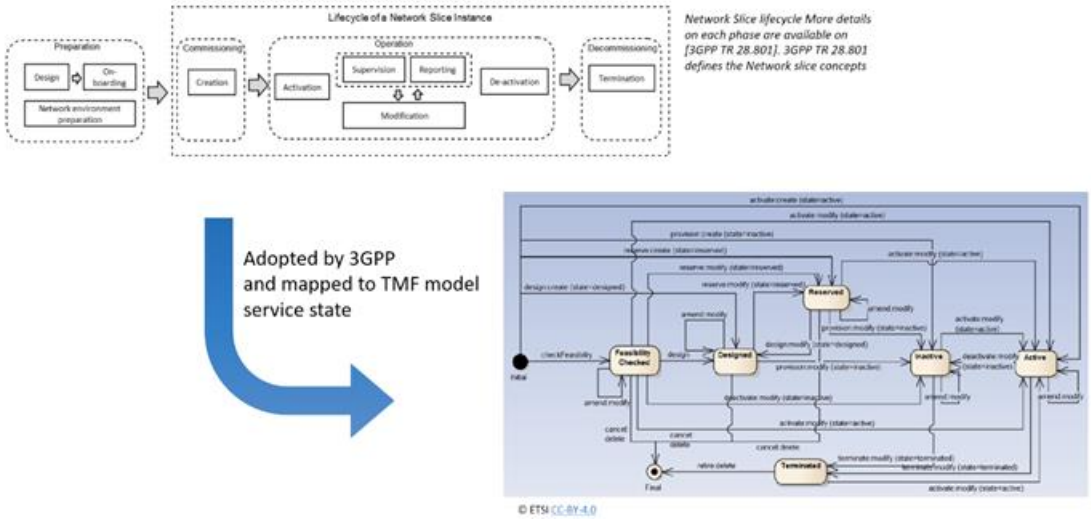

Figure 2: Service Lifecycle Workflow

- **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.

2.2. Orchestration at the Extreme Edge (Feather)

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 NATWORK. Flocky is a newly created 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.

2.2.1. Functional components

Figure 3 shows the relations of 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.

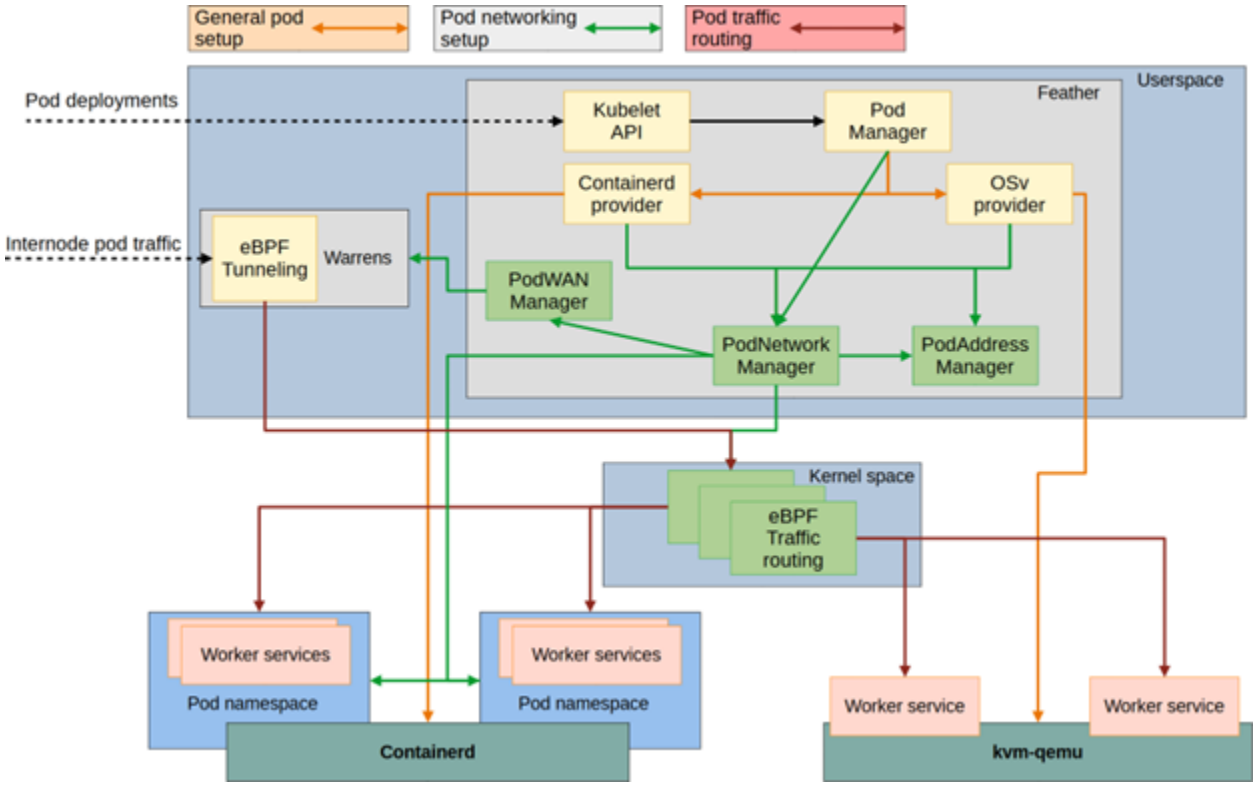

Figure 3: Feather architecture, with main components in yellow (orchestration) and green (networking).

At the orchestration level, Flocky (Figure 4Figure 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.

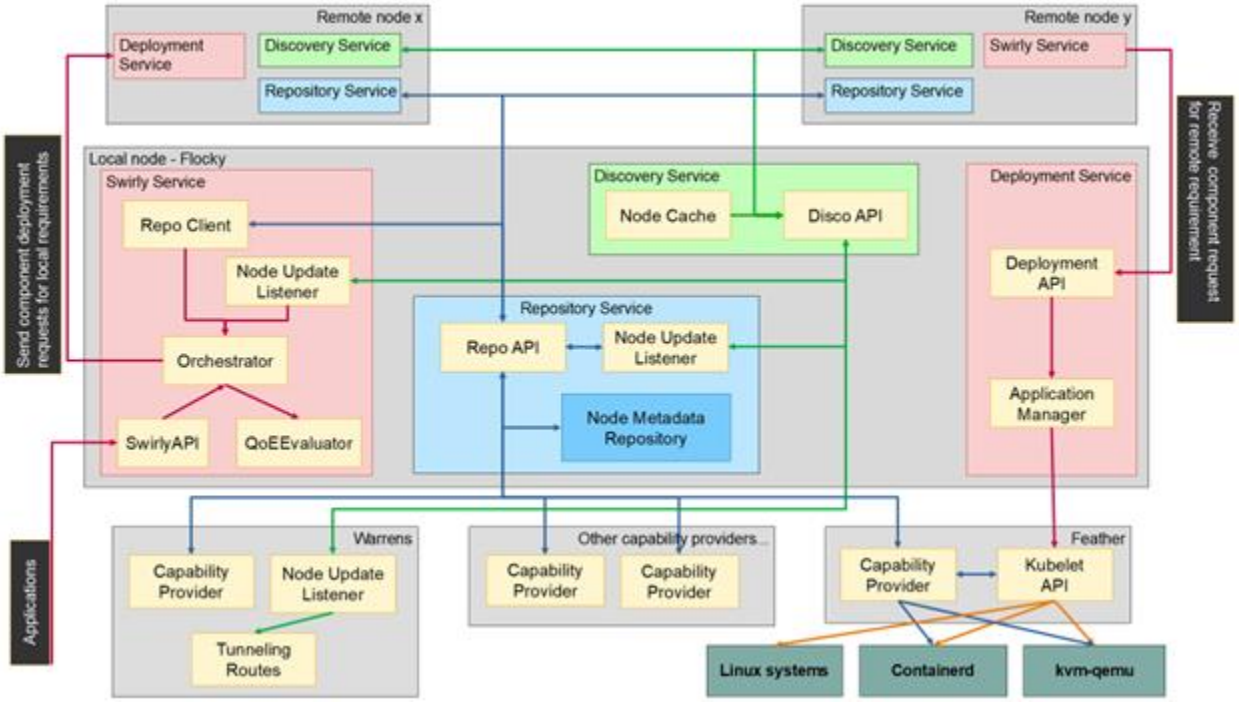

Figure 4: Overview of Flocky services, their main components, and interaction between nodes

2.2.2. Interfaces and Protocols

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.

All Flocky interfaces are REST APIs, and 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.

2.2.3. Data artefacts

- Discovery data: In-memory node catalog used by the Discovery service, containing node names and (public) IP addresses.
- Metadata repository: In-memory OAM metadata store, containing the latest hardware status, running workloads, available runtimes and node properties as reported by each discovered node.
- Latency/Quality of Experience: Used by the Discovery and Swirly services to determine eligible nodes for discovery and deployment, respectively. The latter depends on the chosen optimization parameters and implementation, and relies on information from the Metadata repository.

2.3. Orchestration at the CRAN

This section presents the orchestration mechanisms and architectural design developed within the scope of the NATWORK 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.

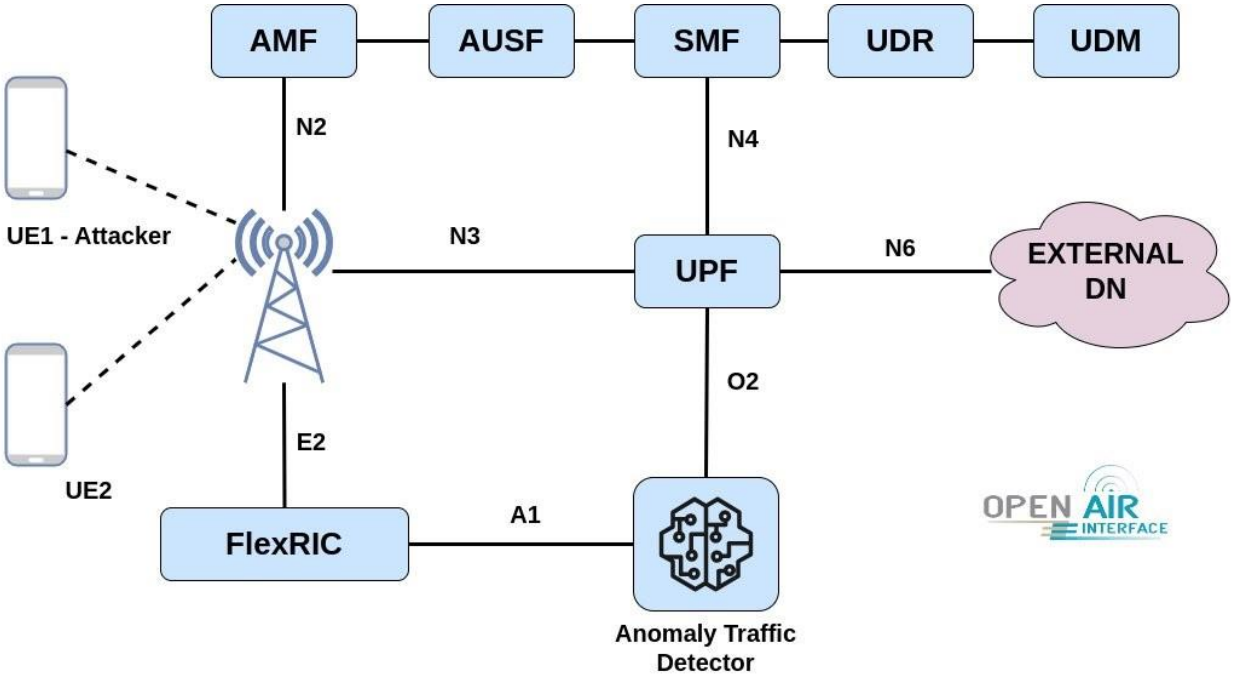

Figure 5: End-to-End Deployment of the AI-Driven Network Intrusion Detection 5G Network

2.3.1. Functional components

The CRAN orchestration framework consists of several key components as illustrated in Figure 5, 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, currently using

Scapy [1] in this first prototype. 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.

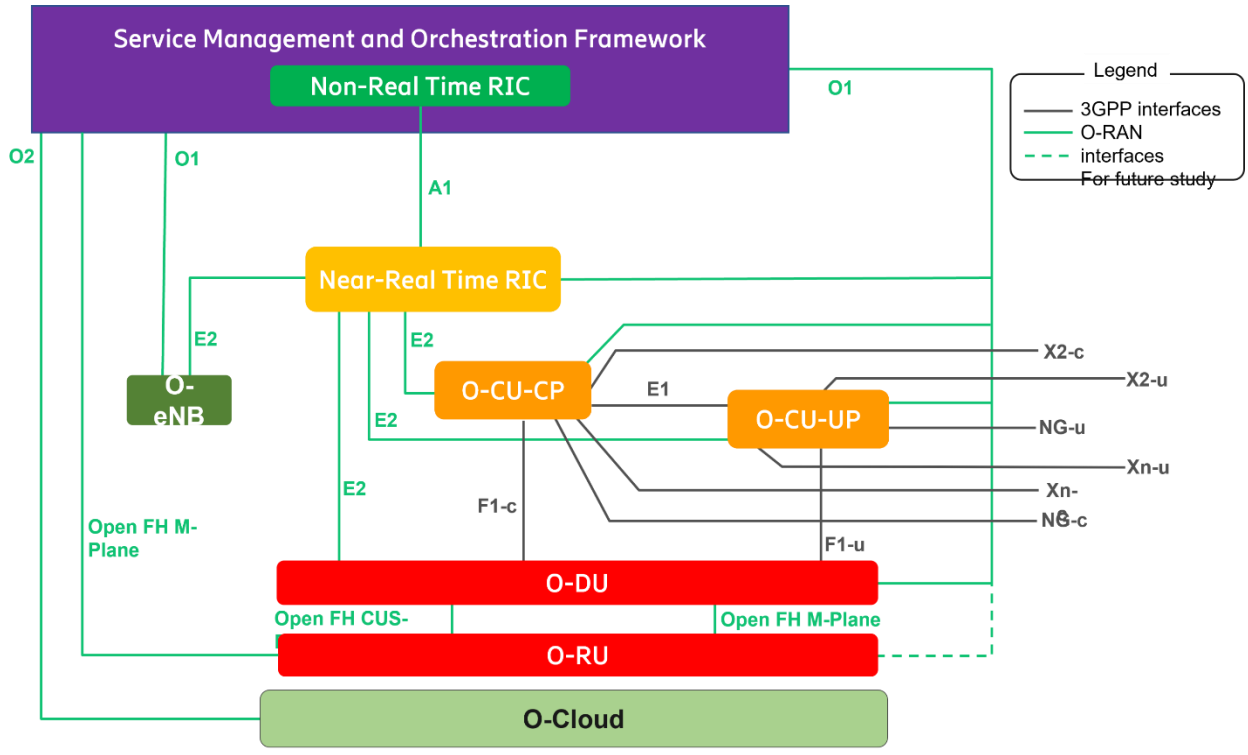

Figure 6: O-RAN Architecture

2.3.2. Interfaces and Protocols

The framework consists of interfaces and protocols to ensure seamless interoperability between components. For clarity, Figure 6 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.

2.3.3. 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.

2.4. 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 [2][2], and initial development of the FORK: A Kubernetes-Compatible Federated Orchestrator [3]. 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. Functional components

The secure-by-Design orchestrator architecture comprises the following key functional components as shown in Figure 7 Figure 7: Secure-by-Design Orchestration Architecture and components:

- **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.

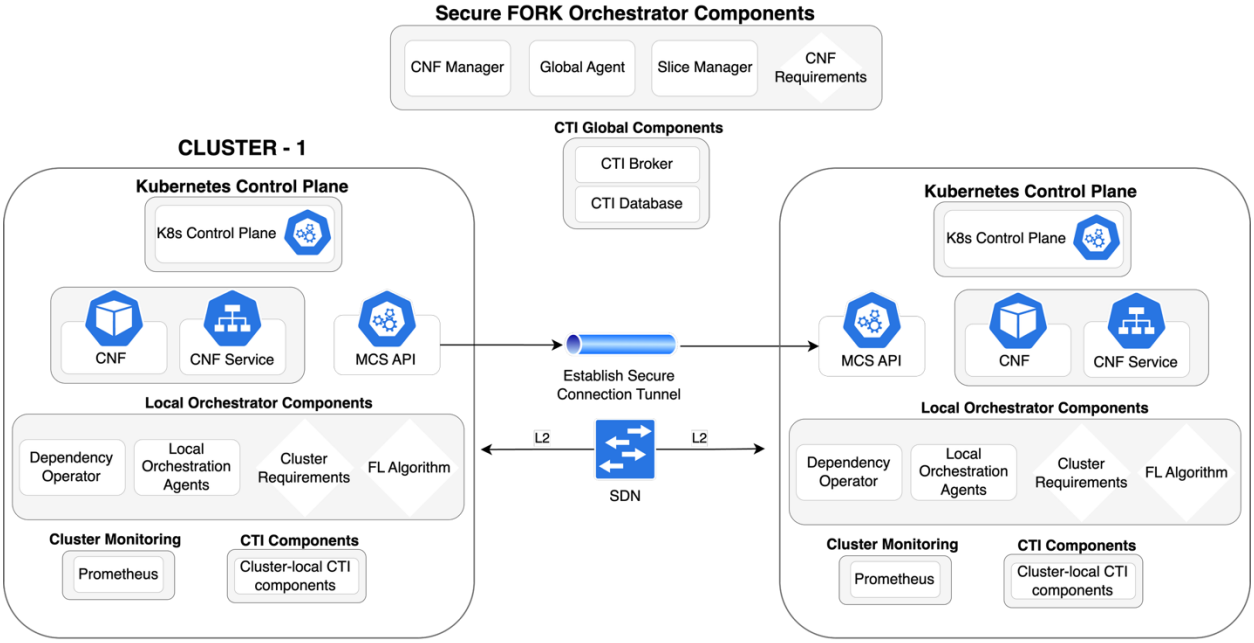

Figure 7: Secure-by-Design Orchestration Architecture and components

The components are presented in Figure 7, 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.

2.4.2. 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.
- **Global Orchestrator API:** A RESTful API enabling the global orchestrator to query resource availability, hygiene scores, and initiate deployments or scaling actions.
- **Cluster Local agent API:** Local agents expose resource metrics, energy scores, and security statuses via a protected endpoint, enabling informed real-time decision-making.
- **Monitoring and Telemetry Interface:** Connects to monitoring tools to gather performance, resource usage, and telemetry data from clusters and deployments. This interface leverages existing monitoring solutions. It reuses existing open-source Prometheus APIs, unlike the other NATWORK-specific interfaces.
- **Machine Learning Interface:** Supports integration with the AI-based learning framework to incorporate predictive insights for slice management decisions

2.4.3. Data artefacts

The orchestration framework manages several key data artefacts critical for secure and efficient operation:

- **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.

3. Data Plane Computation Offloading Design

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.

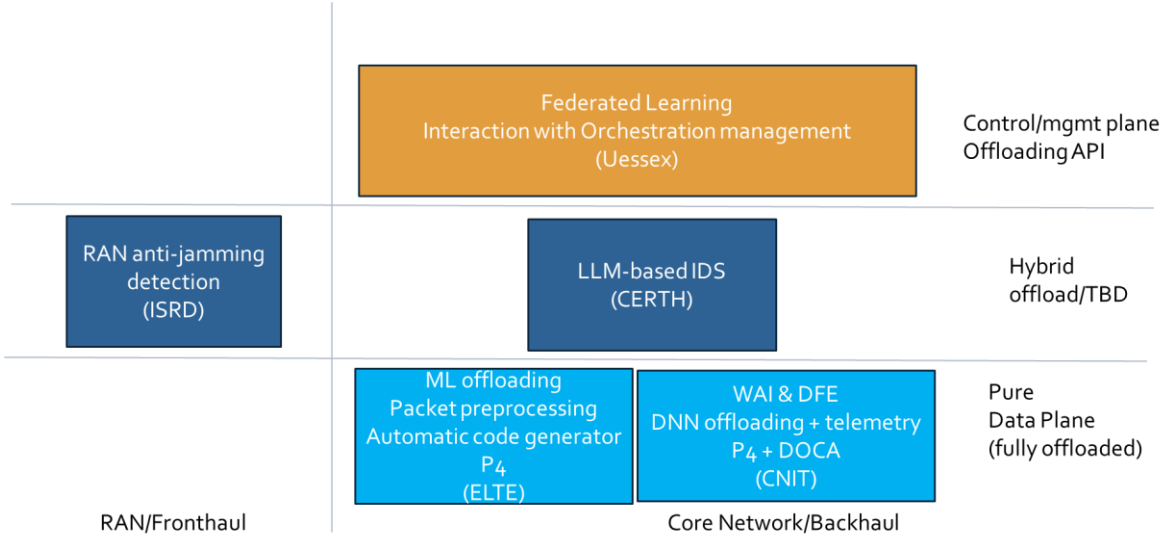

Figure 8: NATWORK Data Plane offloading solutions and flavours

3.1.1. NATWORK Offloading flavours

The chart shown in Figure 8 shows the different NATWORK 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.

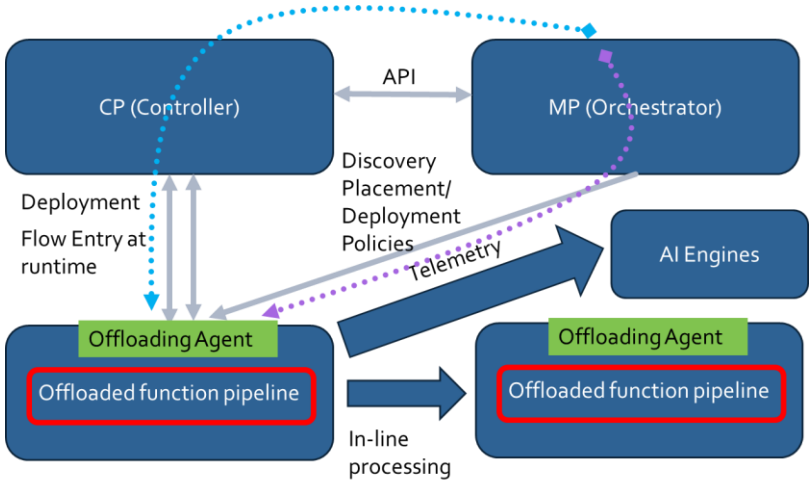

Figure 9: Offloading deployment and runtime configuration options

3.1.2. Deployment and configuration interfaces

A hybrid architecture (Figure 9) 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 10:

- 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.

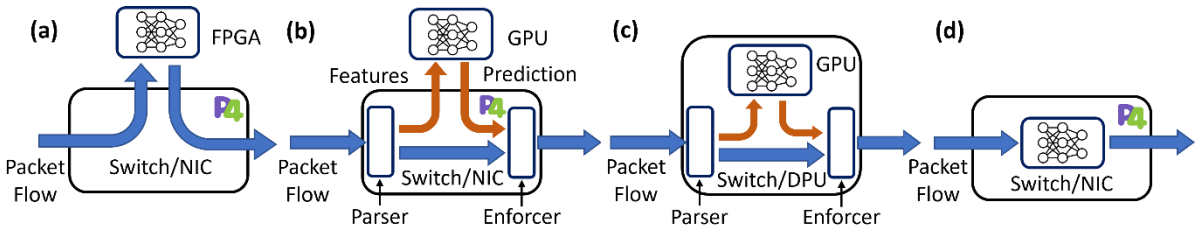

Figure 10: Embedding AI/ML inside programmable switches

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 11.

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.

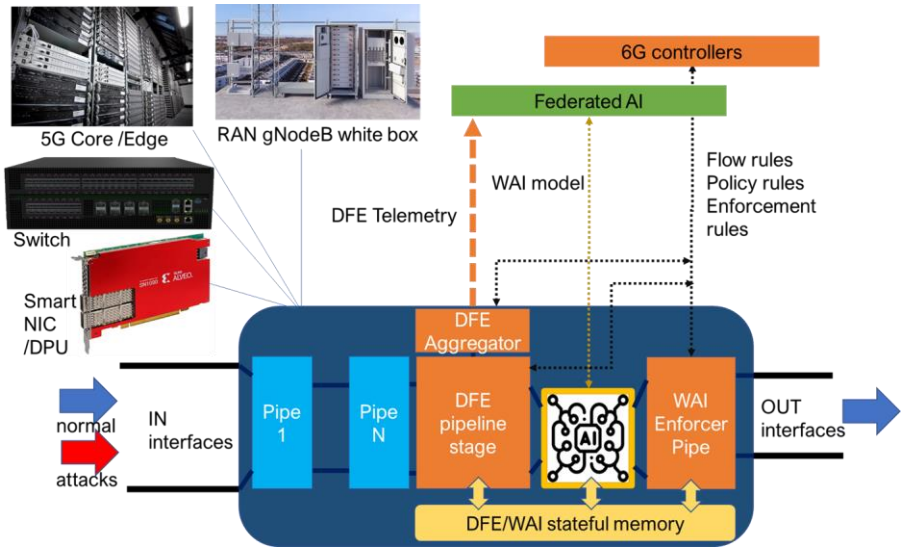

Figure 11: DFE and WAI design submodules

3.2.1. 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 distill 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.

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 12 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.

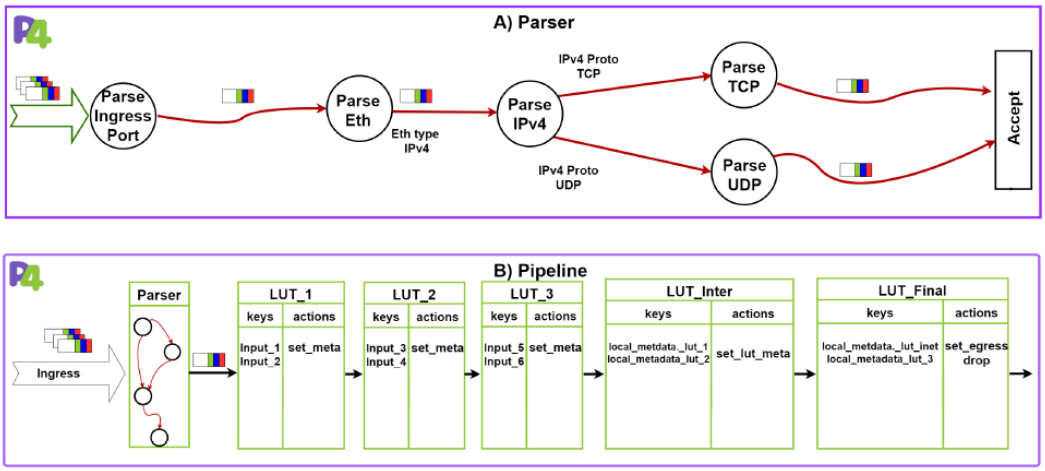

Figure 12: P4 Parser and pipeline for a 6-feature input

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 13A). The P4 pipeline is accordingly composed of five tables (Figure 13B):

- 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).

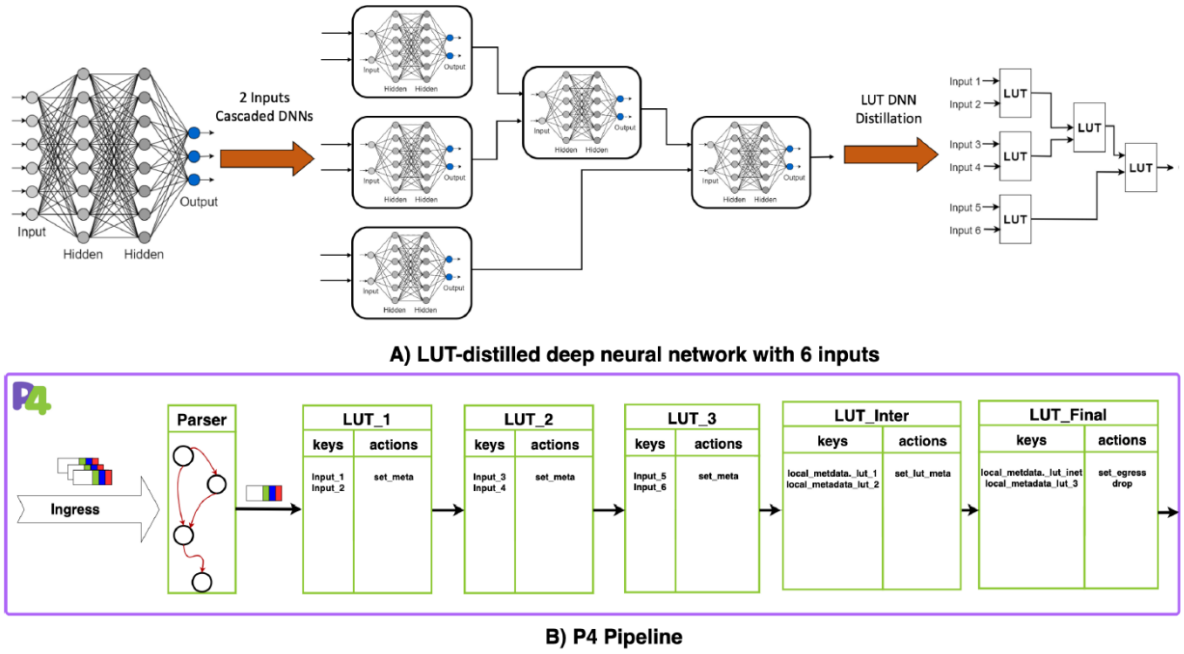

Figure 13: LUT distilled DNN and its P4 pipeline

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 [4] with the NATWORK ack.

3.2.2. 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 14, 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.

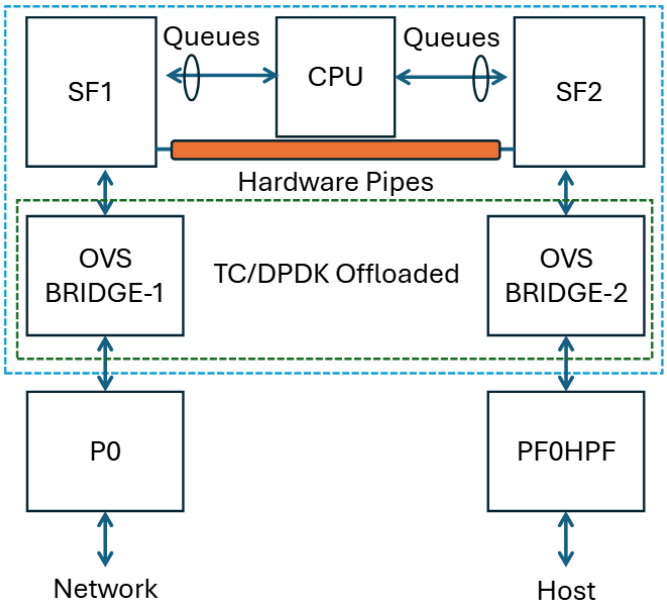

Figure 14: DPU architecture

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.

Not only DOCA Flow APIs do establish hardware pipes at setup but also dynamically manage them, adding or removing entries as needed. These pipes match standard packet fields, including

IP source and destination, L4 protocol, and L4 ports. Additionally, they can assign metadata to packets, accessible by subsequent pipes and ARM cores for further processing.

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 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 [5].

3.3. In-network ML models

Machine learning (ML) in the programmable data plane, particularly with Intel Tofino, 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 15Figure 15. The model uses a federated learning-based approach for deploying machine learning (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: These serve as ground truth sources for training the ML models and monitoring their accuracy.
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.

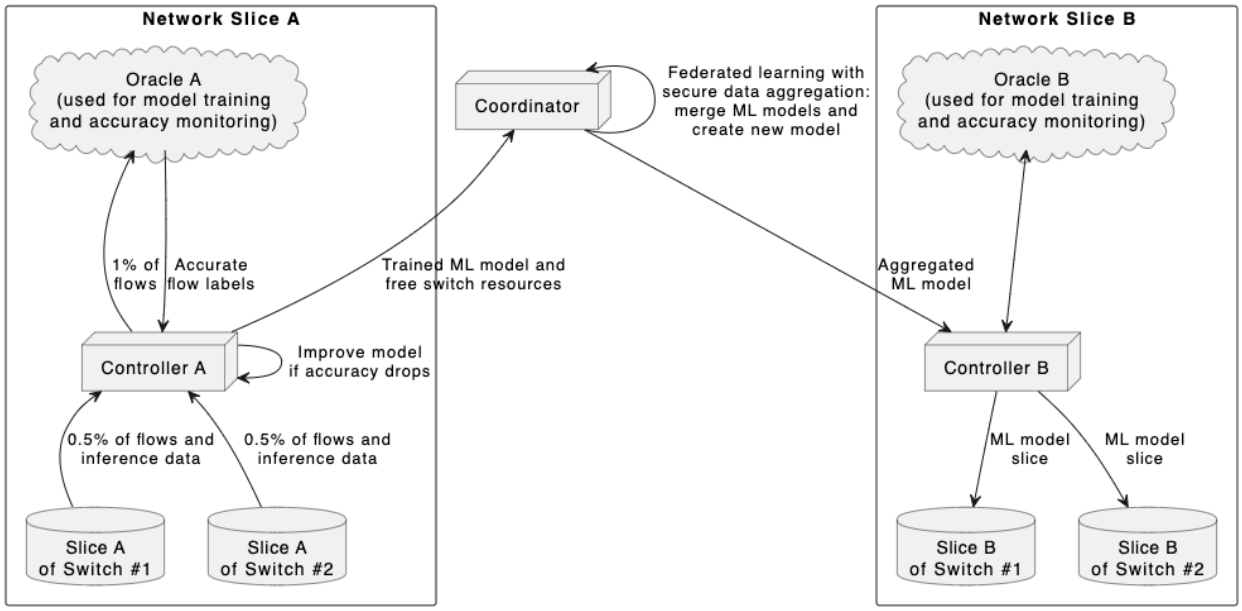

Figure 15: Data plane ML model

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 5.6.

3.4. RAN security-performance balancer

The service aims to balance the performance of radio elements, and the security applied to the radio to ensure the constant availability of radio resources (Figure 16Figure 16). The balancer will consider, on the one hand, the risks of DDoS attacks that occur in the radio interface and, on the other hand, the performance requirements posed to the radio software/hardware due to increased traffic. The risk of the attack considered by the balancer comes from the anti-DDoS xApp which performs attack detection. The main task of the balancer is to understand when the increased performance required is due to an attack in progress or regular peak traffic. The balancer will inform the agents when they should apply deeper packet inspection or when the security controls can be reduced. The 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 standard xApp API.

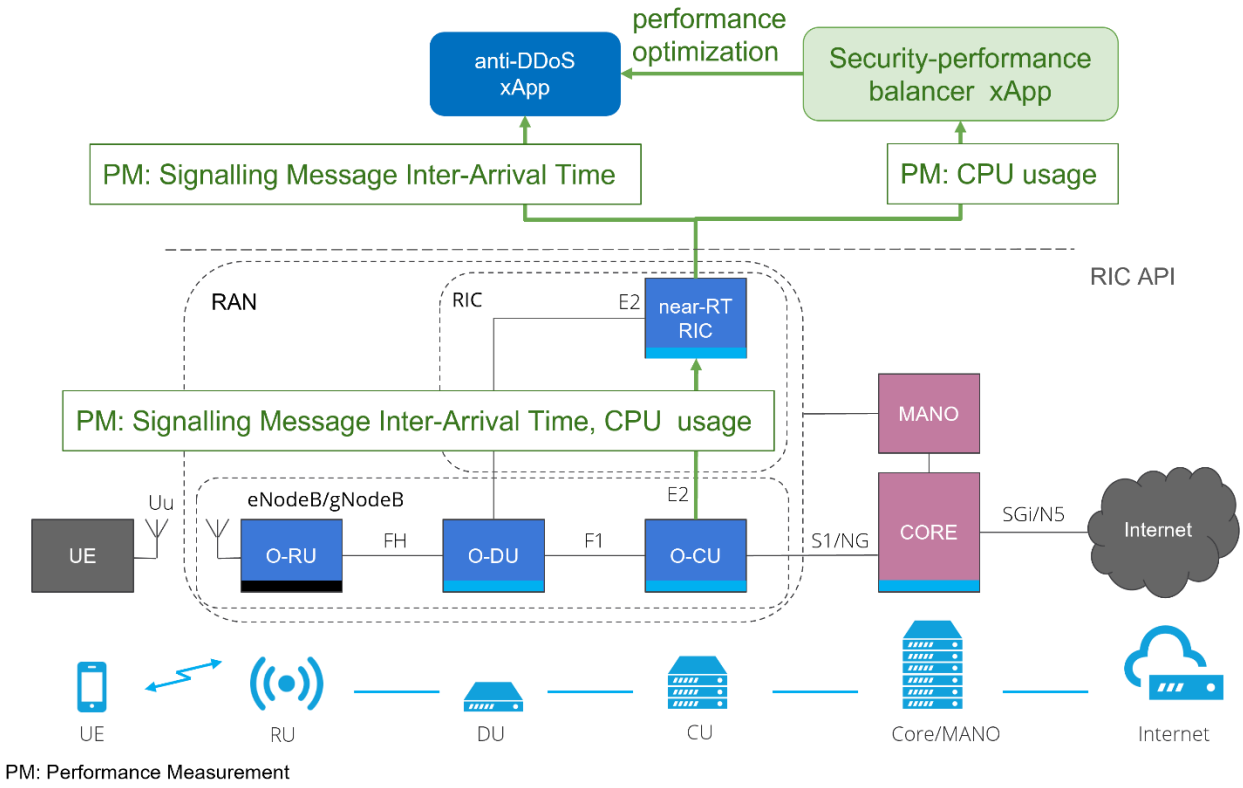

Figure 16: Security-performance balancer architecture

In conclusion, the service acts as a crucial decision-making component within the O-RAN architecture, dynamically adjusting security measures and performance parameters based on real-time threat assessments and traffic demands. By integrating seamlessly with the near-RT RIC through standardized xApp APIs, it provides a robust and adaptive solution to maintain both operational efficiency and resilience against DDoS threats in modern radio networks.

3.5. LLM-based IDS

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.

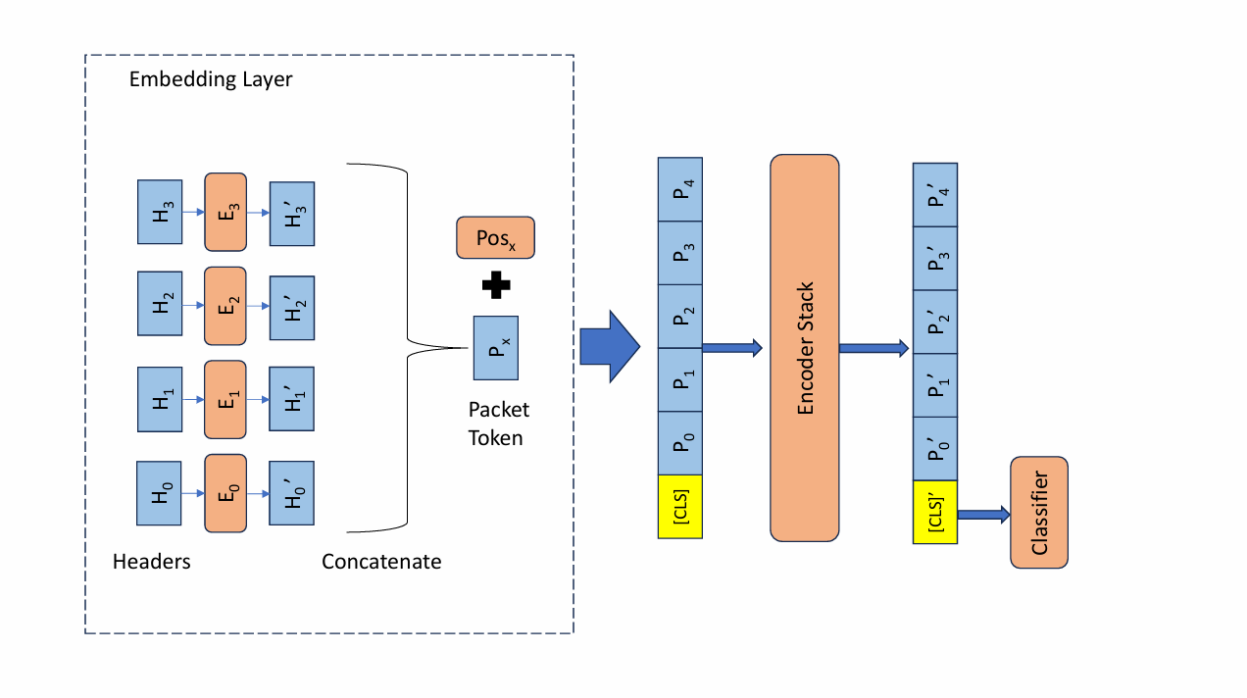

Figure 17: LLM-Based IDS Overview

In Figure 17, we show an overview of the architecture and pipeline for the traffic data processing. In this pipeline we capture traffic that is separated into sequences of packets that are part of a flow identifiable by the 5-tuple of IP protocol, IP addresses, and ports. The features selected for the packet are various packet fields such as the packet length, TCP flags etc. We also extract additional metadata from each packet including a timestamp (relative to the beginning of the flow) as well as direction which is a custom binary flag which replaces the IP address and port

headers in the packet to hide information that irrelevant (and potentially leak label information) during training.

To generate tokens from this sequence, we use a custom embedding layer that translates all raw packet header values into float vectors which are then concatenated together through a linear layer to form a packet token. In these tokens we also add information about the order of each token in the sequence using a positional embedding layer. The encoder stack of our LLM-based IDS takes a sequence of packet tokens as input. The encoder stack of our model is a 4-layer transformer encoder stack with 4 attention heads, and the embedding dimension is 256. In addition to the packet tokens, we also append a special CLS' token at the beginning of the sequence, which serves as the output of the model used for classification. Through our pretraining procedure the transformer encoder stack has been trained to output a representation of the flow packet sequence in the CLS' token.

To classify each flow, we use a classifier module (such as a simple MLP) which can either be trained along with rest of the model during a supervised fine-tuning step or can be trained on its own only to identify malicious flows from the output of our LLM (using a simple MLP or linear classifiers such as Logistic Regression, Random Forest etc.). Unsupervised anomaly detection is also possible using the output of the LLM to detect flows that significantly differ from regular traffic.

4. Software Design: Orchestration Support Systems

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.

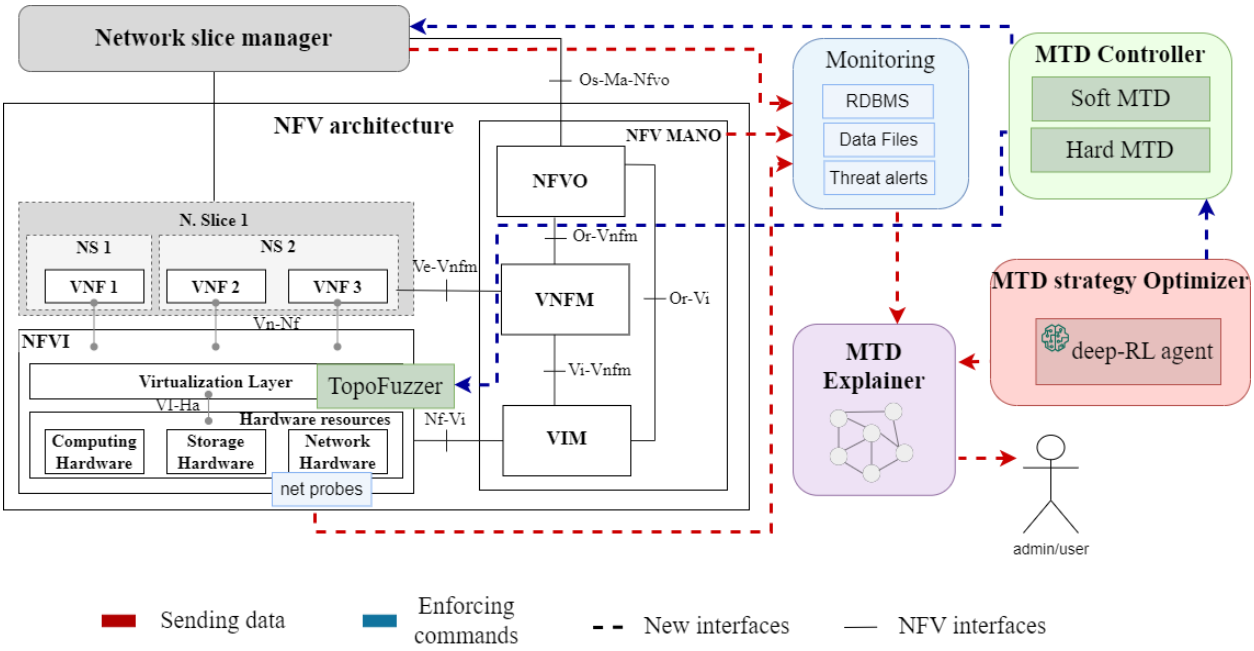

Figure 18: Architecture of the MTD framework

The proposed framework consists of three main components, as shown in Figure 18: Architecture of the MTD framework:

- **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. MTD Controller

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

4.1.1.1. 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.

4.1.1.2. 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**

4.1.1.3. 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.

4.1.1.4. 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.

4.1.1.5. 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.

4.1.1.6. 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.

4.2. 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. 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 19: CTI Solution Architecture and components:

- **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.

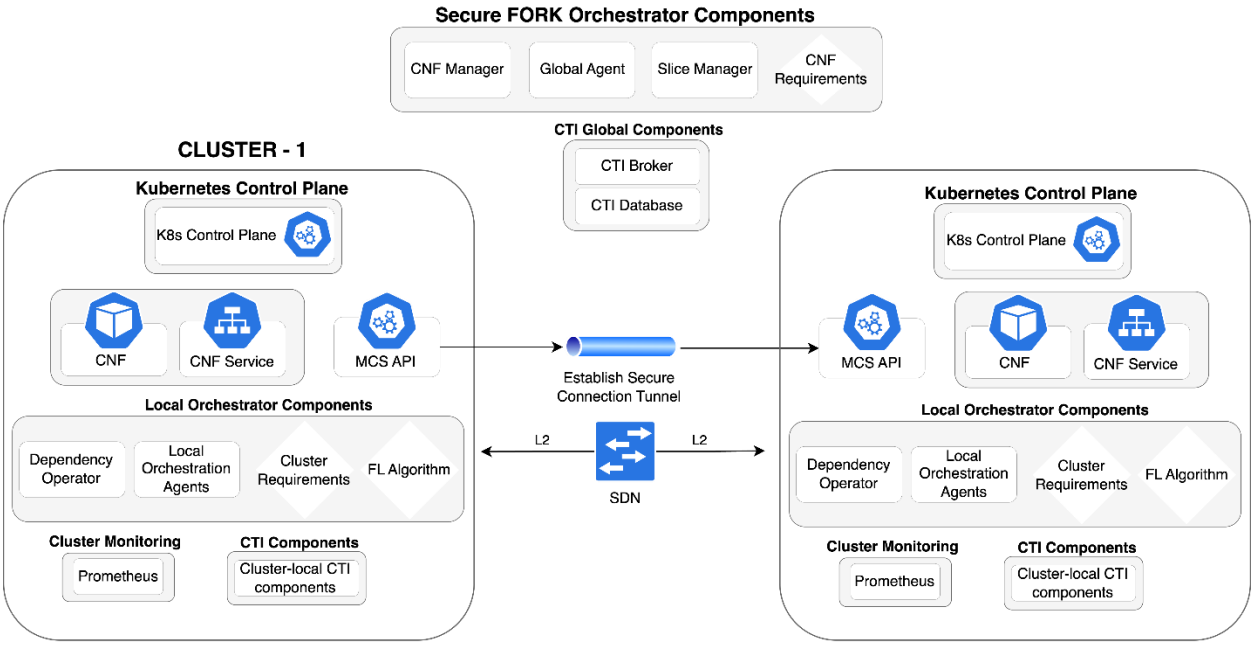

Figure 19: CTI Solution Architecture and components

The components are presented in Figure 19, 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 (Trusted Automated Exchange of Intelligence Information) 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 2.4.1.

4.2.2. 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.

4.2.3. 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.

4.2.4. 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 20: Simple confusion matrix for decision making strategy, 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.

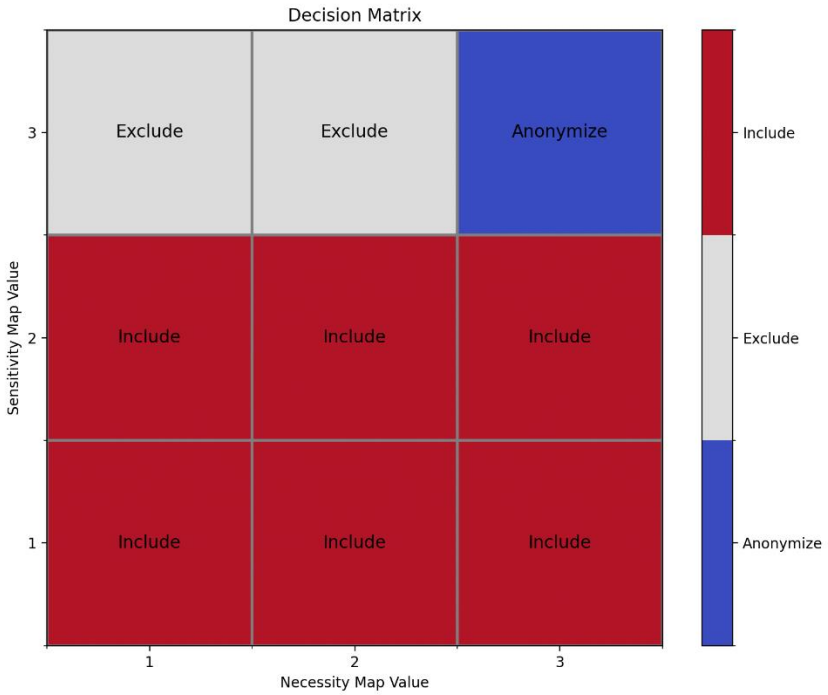

Figure 20: Simple confusion matrix for decision making strategy

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* M*ap* and *Necessity* M*ap* strategies as shown in Figure 21: Sample Sensitivity and Necessity maps for decision making strategy. 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 21: Sample Sensitivity and Necessity maps for decision making strategy

4.3. AI-based Behavioural Analysis service

4.3.1. DFE/WAI

4.3.1.1. Technical Description

DFE (Distributed Feature Extraction) and WAI (Wirespeed AI) 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 current functionalities exposed by the OFA 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

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.

4.3.1.4. Algorithms

In the current implementation, no algorithms are employed in the OFA.

4.3.1.5. Interfaces and Protocols

DFE and WAI communicate with the Offloaded Function Agent (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).

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		{
		"running_containers": [
		"DPU_DFE3",
		"DPU_DFE2",
		"DPU_DFE1"
		]
		}
		
		{
		"message": "Container 'DPU_DFE1' started successfully!"
		}
		
		{
		"message": "Container 'DPU_DFE1' stopped successfully!"
		}

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 22:

- 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 IDS (Intrusion Detection System) running in the control plane. The IDS 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.

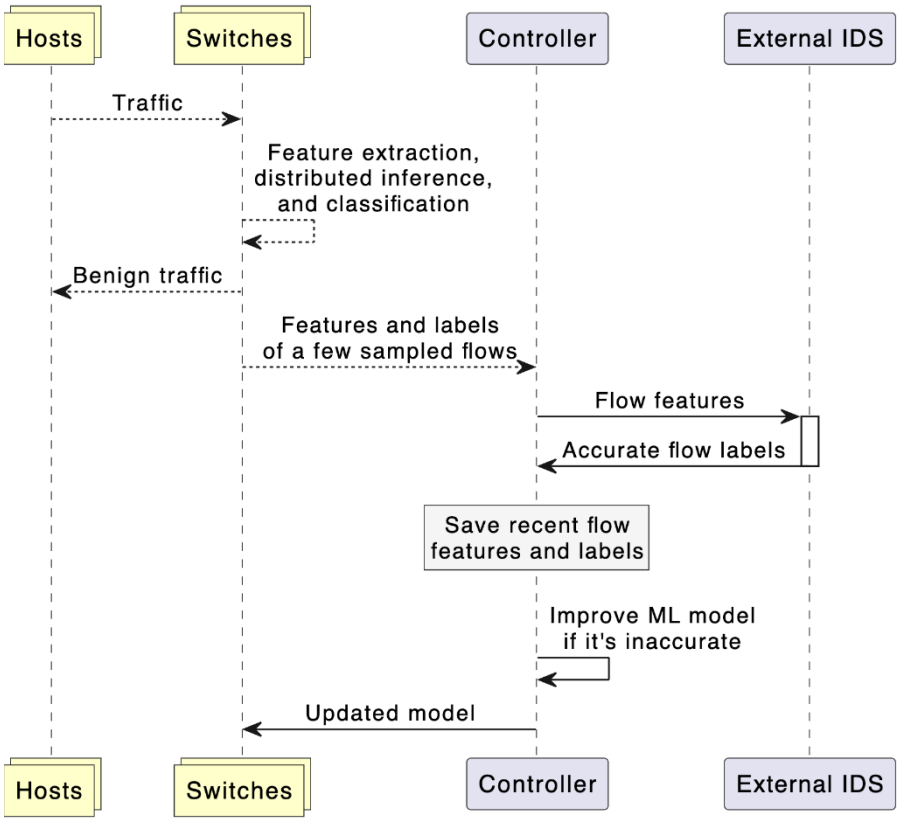

Figure 22: Data plane ML components and interactions

4.3.2.2. Functionalities Provided

- Feature Extraction: Switches parse incoming packets and extract predefined features useful for traffic classification.
- In-Network Inference: Lightweight random forest inference is performed on the switch to classify packets in real time.
- Traffic Sampling: A small subset of traffic flows is forwarded to the control plane to enable model validation and updates.

4.3.2.3. Dependencies

- Programmable Data Plane: Requires switches supporting the P4 language (e.g., Intel Tofino) for feature extraction and inference logic.

- Control Plane Infrastructure: Includes a controller capable of model training, monitoring, and deployment.
- IDS Software: Must be capable of receiving mirrored traffic and classifying it using complex ML models.

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. Microservice behavioural analysis

The NATWORK 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.

4.3.3.1. Technical Description

The Microservice behavioural analysis module performs continuous microservice performance monitoring to ensure efficient operation of the system. In NATWORK, 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 Intrusion Detection System (IDS), automated scaling decisions and elasticity actions are triggered to maintain optimal resource utilization and prevent service degradation. Figure 23 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.

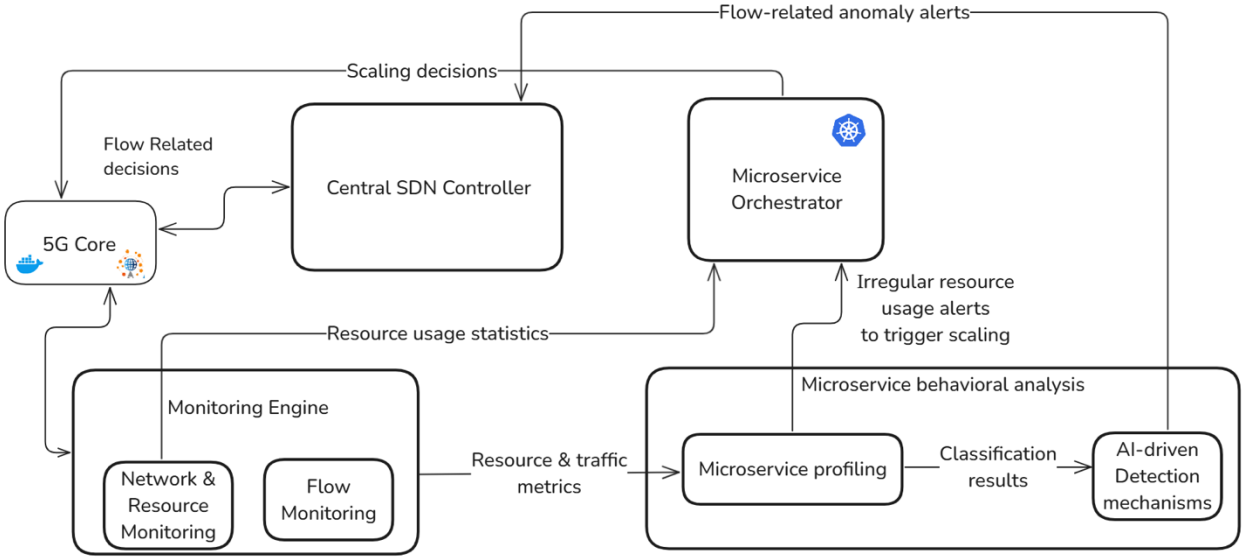

Figure 23: Position of the microservice behavioural analysis module and interconnection to other modules

4.3.3.2. 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

4.3.3.3. 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 [6] 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.

4.3.3.4. 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 24: UDP Flooding Attack Execution and Figure 25: New flow control rule (left) and a graphical representation of flow rules (right).

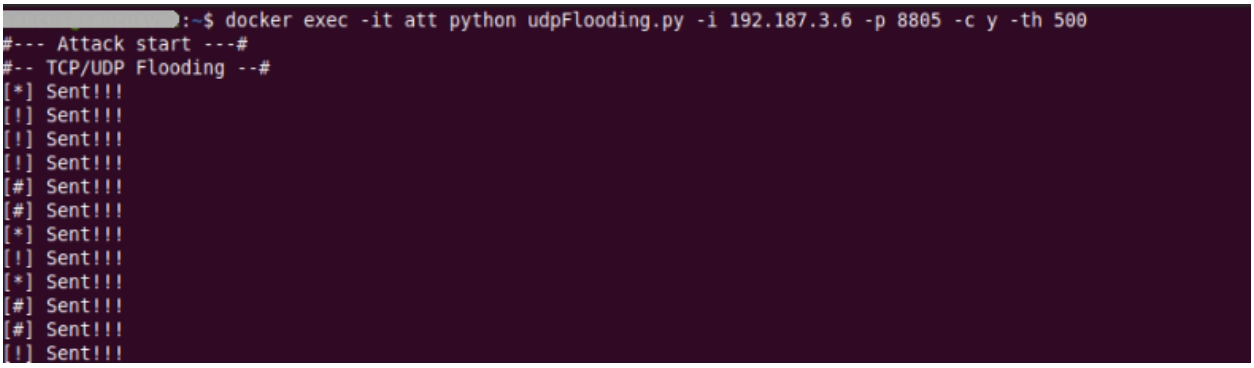

Figure 24: UDP Flooding Attack Execution

Figure 25: New flow control rule (left) and a graphical representation of flow rules (right)

4.3.3.1. 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	-

4.4. Security-performance balancer service

4.4.1. 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 26. 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.

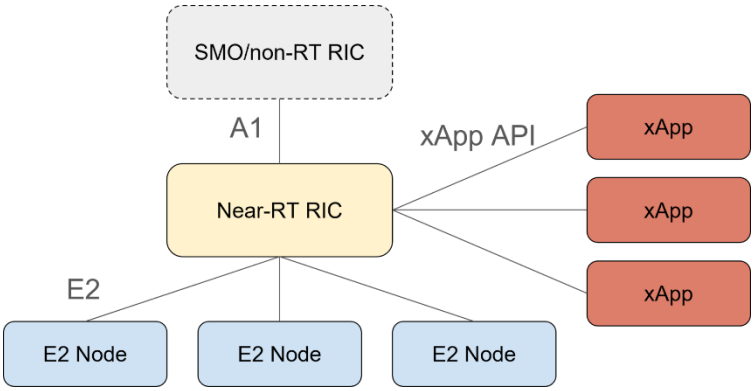

Figure 26: ISRD Liquid Near-RT RIC interfaces.

The Liquid Near-RT RIC general deployment architecture is depicted in Figure 27. 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.

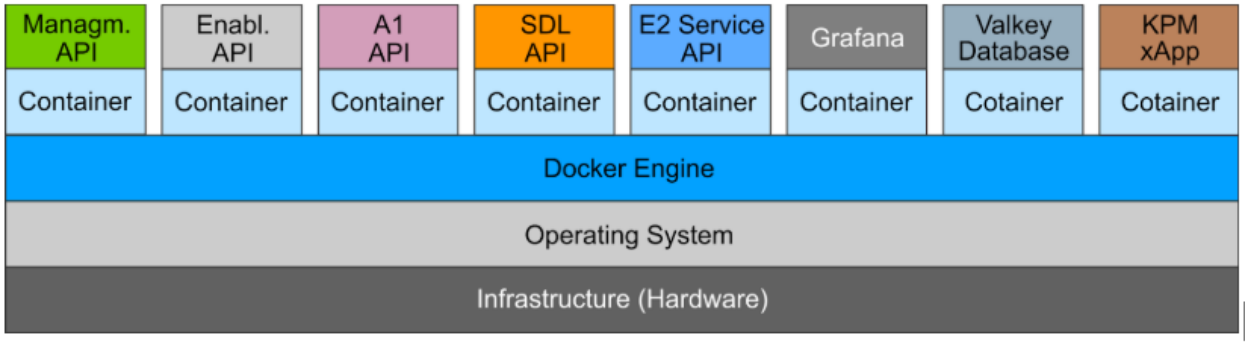

Figure 27: ISRD Liquid Near-RT RIC general deployment architecture.

4.4.2. 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).
- **Security and performance optimization:** optimizes the security xApp parameters, e.g., inter-arrival time Performance Measurement (PM) reporting frequency, based on the target performance.
- **Node control**: sends a control message to xApp to modify its operation parameters.

4.4.3. Dependencies

Security-performance balancer service relies on the following components:

- **ISRD anti-DDoS xApp** is a Python-based xApp that detects a UE attack on 5G RAN based on the RRC Signaling Message Inter-arrival Time PM and disconnects the attacking UEs. The Security-performance balancer controls this component to limit its load on the CPU while maintaining a high DDoS attack detection rate.

- **ISRD Liquid RAN:** ISRD Liquid RAN consists of the following. O-RAN Central Unit (O-CU) is a logical node hosting RRC [7], SDAP [8] and PDCP [9] protocols. O-RAN Distributed Unit (O-DU) is a logical node hosting RLC [10]/MAC [11]/High-PHY [12] layers based on a lower-layer functional split [12]. 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 [13].

- **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 [13]. The A1 interface, connecting the SMO layer [14] 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.

4.4.4. Algorithms

The main task of the balancer is to understand when the increased performance required is due to an attack in progress or due to regular peak traffic. The system processes a dataset of n samples, each with m feature measurements, and classifies them into k predefined categories using a balancing mechanism. When a radio's resource usage spikes, the system uses inference methods like Naïve Bayes to classify new, unseen samples. Unlike standard Bayesian classification that assumes all classes are equally represented, this approach adjusts classification thresholds to improve decision accuracy. The balancer also dynamically instructs agents to either intensify inspection (e.g., deep packet inspection) or relax security controls based on classification outcomes.

4.4.5. 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 [15]. 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 [13].
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 [16].
Input	Reporting parameters from anti-DDoS xApp.
Output	Change parameters of anti-DDoS xApp.

4.5. Blockchain Based Trust Establishment

Current 5G authentication mechanisms involve main core network functions such as AMF and AUSF for trust establishment, which, while secure, are not designed for repeated or federated end-to-end validation. 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 evident in trust-sensitive IoT applications 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 allows devices to authenticate directly with service providers after initial authentication and registration in the core, reducing reliance on the 5G core and supporting scalable, trustless access control in distributed IoT environments.

4.5.1. Technical Description

The NATWORK system 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 blockchain for enhanced trust. Key components include the User Equipment (UE), which registers with the 5G network and records a pseudonym on the blockchain, and the gNodeB 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 Data Network (DN), facilitating data routing between IoT devices and service providers.

- UE: Emulating the User Equipment (UE) functionality, a Raspberry Pi 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 blockchain serves as the distributed ledger that integrates smart contracts to handle various parts of the end-to-end trust establishment process.

4.5.2. Dependencies

This module represents the main component in S3-S-C2: End-to-End Security Management in NATWORK. It requires the following:

- Blockchain: Part of the authorization database is replaced with an Ethereum-compatible permissioned blockchain. This provides a decentralized, transparent, and integrity-safeguarded 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 blockchain. The main function of this bridge is to listen to the log of 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.

4.5.3. Functionalities Provided

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

- Decentralized Authentication: It securely verifies both users and 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.

4.5.4. Algorithms & Workflow

The Service Provider which simulates external applications, leverages the blockchain for offloading identity verification. It handles authentication requests, verifies pseudonyms via a smart contract, and uses cryptographic signatures for challenge-response interactions. This module issues short-term tokens for low-latency access without the need for repeated authentication. By utilizing a permissioned blockchain to manage device credentials and access control policies, NATWORK 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. This approach aligns with zero-trust principles, improving security, decentralization, and latency in IoT services.

4.5.5. 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 (possibly in JSON format).
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 token issued for the UE and service provider details. (possibly in JSON format).
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 (possibly in JSON format).
Output	Acknowledgement of success/ Invalid token or UE pseudonym regarding the performed action (verification).

5. Implementation

5.1. Orchestration at the Extreme Edge (Feather)

In Feather, Providers are implemented for both Containerd (Containers) and OSv (unikernels on KVM). This distinction is made based on metadata fields compliant with the OCI specfication, which are ignored by non-Feather agents, specifically feather.backend (container, OSv) and feather.runtime (containerd, KVM). This allows runtimes to support multiple image formats, and for image formats to run on different runtimes if possible.

Non-container images are created by including the workload VMdisk/image as a container layer, and setting the required metadata. This mechanism is not like the Docker approach of multi-platform images, and requires different image names per runtime, which is solved at a higher level by Flocky. Some features may be limited depending on the chosen runtime, for example due to limitations with OSv/KVM, only local read-only mounts are supported for unikernels at this point. Kubernetes secrets and mounts are handled by boot scripts added to an OCI (unikernel) image through a custom tool “Flint”.

Multi-runtime networking in Feather is supported in both Kubernetes clusters and in standalone mode, although in Kubernetes Feather defaults to “legacy” network operation, which assigns unique addresses to each container (instead of per pod) and assumes only a single container per pod. Additionally, both IPv4 and IPv6 addressing schemes are developed.

For Flocky, Capability providers are implemented by Feather (hardware resources, running applications, runtime features) and Warrens or a suitable VPN (network security features). Remote attestation may also be registered as a separate Capability provider if present. Various intents (Traits) are defined to allow workloads to request deployment with (among others) Green energy, QoE limits, specific (secure) runtimes, attestation-capable nodes and specific resource limits.

QoE calculation is open to implementation; currently implemented methods involve static calculation based on node metadata properties and soft node-to-workload matching. Planned ML-based calculation allows online learning of QoE properties from user preferences based on gathered metadata.

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: in review, preprint 10.13140/RG.2.2.13816.35847

Flocky repository:

- Main repository: https://github.com/togoetha/flocky
- Documentation: README.md in repository
- License: Apache 2.0 (open source)
- Important setup scripts: in repository
- Publication: in review

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 User Equipment (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 using Scapy [1], 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 [17]. 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 (RC) 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 machine learning (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 [18]. 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. 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.

The detailed operation of our framework is illustrated in Figure 28. The ATD unit utilizing Scapy, 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.

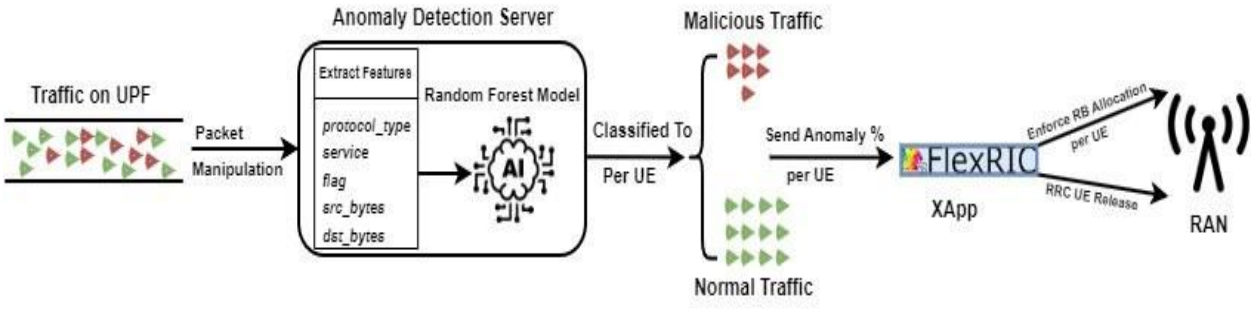

Figure 28: Detailed Architecture of the AI-Driven Network Intrusion Detection System

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

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

sFORK 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

5.4. WAI/DFE

The software implementing WAI and DFE at the DPU and P4 switches is based on the backend features of the hardware platform. In the case of the DPU, a DOCA Flow VNF has been implemented in C++ to realize the DDoS mitigator offloading program, based on DOCA libraries version 2.9. In the case of P4, a baseline P4 program featuring a cascade of flow tables has been employed to test the P4 capabilities in terms of stateful memory and SRAM/TCAM requirements, utilizing the P4 Insight and the P4 Studio SDE 9.7 tools provided along with the APS Tofino switch available at the CNIT laboratories.

Currently, the only available open-source software repository is the P4 DNN Distillation method implementation and assessment software: The software repositories (DOCA programs, P4 programs) have not been released as open source. The plan is to release them on the next deliverables.

Code repository:

- Main repository: https://github.com/emiliopaolini/P4NN_journal
- License: Apache 2.0 (open source)
- Publication: https://doi.org/10.1109/OJCOMS.2024.3411071

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. In-network 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: submitted to IEEE Globecom 2025, under review

5.7. 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 [19]. During this process, CRIU performs various steps as shown in Figure 29.

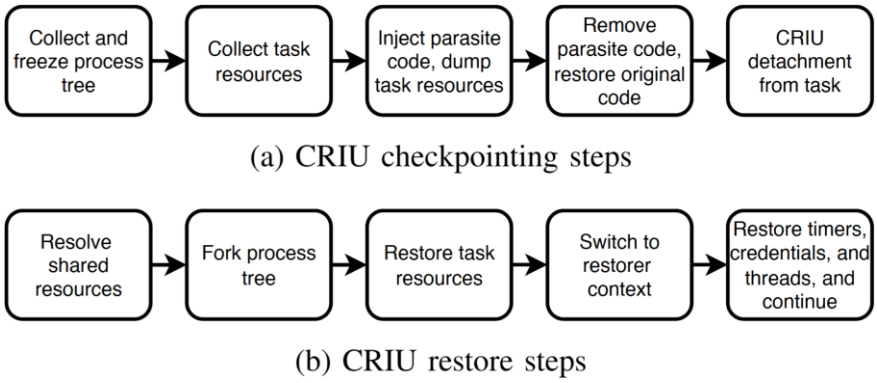

Figure 29: CRIU steps

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: in review

5.8. 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: under review

6. Strategies and Optimisation Algorithms

6.1. Orchestration at the Extreme Edge (Feather)

6.1.1.1. 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.

6.1.1.2. 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.

6.1.1.3. 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 [20] discovery mechanism to enable decentralized node metadata discovery, including supported runtimes and available applications. With a pluggable orchestration system allowing for different preferences and algorithms per node. The deployment algorithm for an application is summarized as follows:

- Split into components (individual workloads)
- For each component
    - Find suitable implementations in Metadata repository based on Traits (intents)
    - (Optionally) rank implementations by preference
    - For each implementation

- Check if a suitable one (same or higher Traits) is already deployed on a remote node
  - Use if found
- Find acceptable nodes using Metadata repository
- Rank acceptable nodes by Quality of Experience (QoE) according to chosen definition/implementation (QoE evaluator)
- In order of ascending QoE, attempt deployment of the component implementation on each node until successful

The “Find acceptable nodes” and “Rank by QoE” stages can be extended by implementing a “QoE evaluator” interface and configuring a specific Flocky instance to use that implementation.

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.

6.3. 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.

6.4. Moving Target Defence (MTD)

The proposed Moving Target Defence (MTD) framework introduces two algorithms designed to optimize stateful live migration (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 5.7, 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 live migration (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 LiMi Scheduling Using ML-based Time Prediction

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 30, 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.

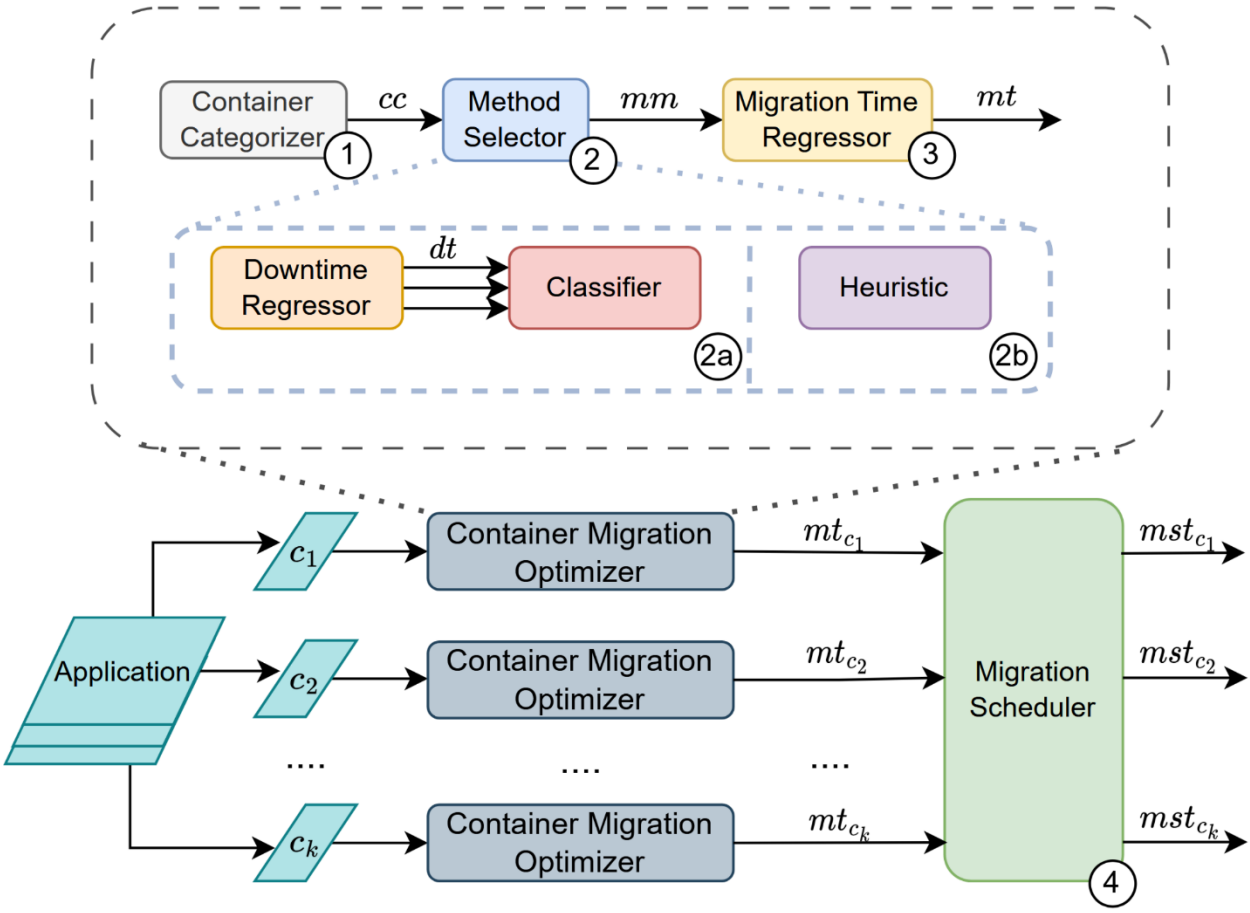

Figure 30: ML Classifier + regressor to optimize the live migration of microservice-based applications

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.

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 lookup table (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 31, a network with two inputs of *n* and *m* bits forms a compound address of *n+m* bits, generating 2^{m+n} 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.

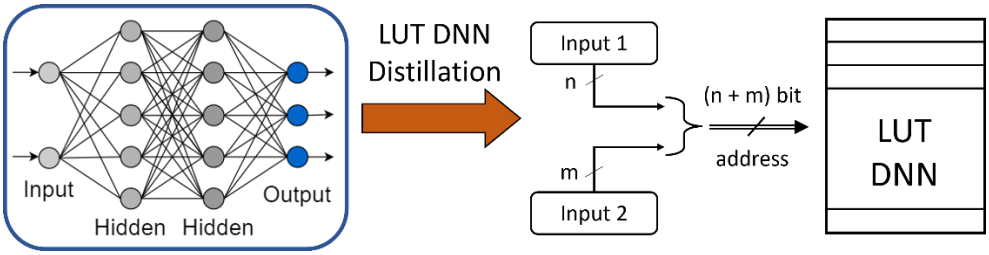

Figure 31: DNN to lookup distillation method

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 lookup tables (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: $y = \sigma(f(w, q\_input(x)) + b)$

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.

6.5.2. DFE and WAI in DPU-based mitigation

6.5.2.1. Problem definition

A prevalent method used in Distributed Denial-of-Service (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, Machine Learning (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 32.

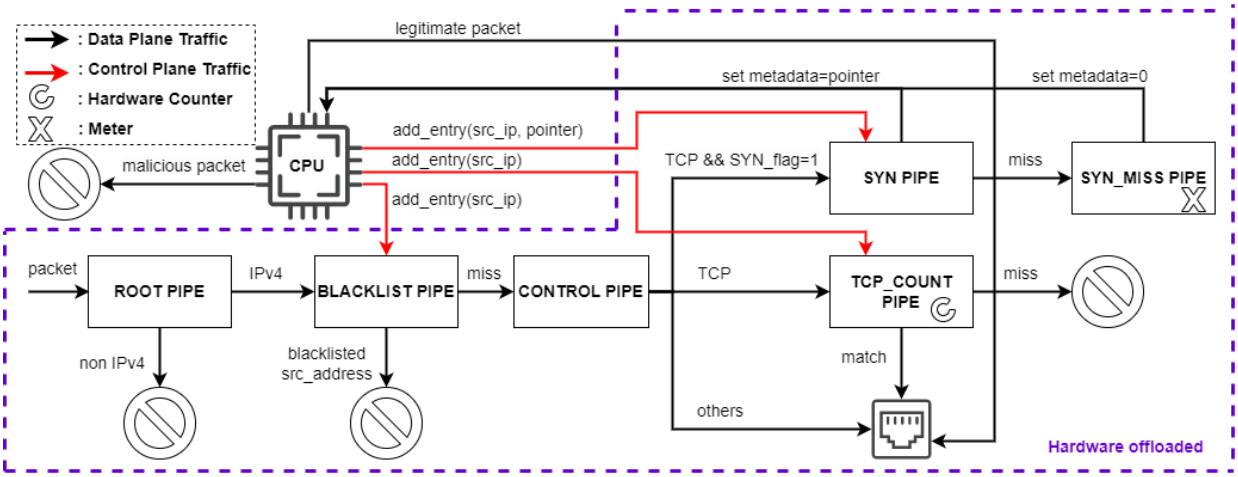

Figure 32: DOCA-based offloaded DDoS mitigator

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 33. 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.

Algorithm 1 Application Logic pseudo-code

```
while true do
    poll RX queue for RX_packets
    if #RX_packets ≠ 0 then
        for packet in RX_packets do
            parse the packet
            if meta = 0 then
                start IP monitoring
                add packet to TX_packets
            else
                retrieve IP record from meta
                increase IP sent #SYN
                query #PKTs HW counter
                if #SYN/#PKTs ≥ TH then
                    blacklist the IP
                    drop the packet
                else
                    add packet to TX_packets
                end if
            end if
        end for
        send TX_packets to TX_queue
    end if
    manage entry aging
end while
```

Figure 33: Offloaded DDoS mitigator algorithm

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.

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.

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 Intrusion Detection System (IDS), which provides highly accurate flow labels. 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. Model Encoding for Match-Action Tables

6.6.2.3. Problem definition

The P4 data plane programming language supports multiple types of programmable switches. We tested our model on two different P4 targets: CPU-based eBPF switches and Intel Tofino hardware switches. These targets have distinct limitations, requiring different approaches to embedding machine learning models.

The match-action tables of eBPF-based switches do not currently support range matching (i.e., mapping actions to numerical value intervals).

6.6.2.4. Developed solution

Therefore, we used a model encoding method, where each depth level of each decision tree corresponds to a separate match-action table. The table entries represent decision tree nodes, while the table actions compare feature values to thresholds, determine verdicts, or forward processing to the next match-action table.

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.

6.6.3. Model Disaggregation

6.6.3.1. 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.

6.6.3.2. 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.

6.7. RAN security-performance balancer

The core idea of our approach to security is to balance the performance of radio and edge elements and the security added to the radio for the constant availability of the radio resources. The balancer will consider, on the one hand, the risks that appeared in the radio interface and, on the other hand, the performance requirements posed to the radio software/hardware due to increased traffic. The main task of the balancer is to understand when the increased performance required is due to an attack in progress or due to regular peak traffic.

6.7.1. Problem definition

Modern radio access networks (RANs) face the dual challenge of maintaining high performance under dynamic traffic conditions while also ensuring robust security against evolving threats such as DDoS attacks. Traditional static security configurations can either underperform during legitimate traffic peaks or overburden the system when reacting to benign conditions. This creates a critical need for a dynamic mechanism that can intelligently distinguish between increased traffic caused by legitimate usage and that triggered by malicious activities.

The core problem addressed in this work is the need to balance the performance of radio and edge elements with the security mechanisms applied to the radio interface, in order to ensure the constant availability of radio resources.

6.7.2. Developed Solution

We will consider an observed data set with n data samples, where each sample contains the measurements of m features observed during a given time interval. Then, the balancer will classify each sample in known classes of interest. In the moment that the radio increases the consumption of resources, the balancer will apply inference (e.g., Naïve Bayesian) for deciding when an unobserved sample Xn+1 may be instantiated to one of the classes. Normally, Bayesian classifier design tries to make the classification process “balanced” as if all classes were represented by a non-zero number of samples in training set X. Our approach will consist of

adjusting the threshold for the Bayesian classifier to classify the sample in one of the classes. The balancer will inform the agents when they should apply deeper packet inspection or when the security controls can be reduced.

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.

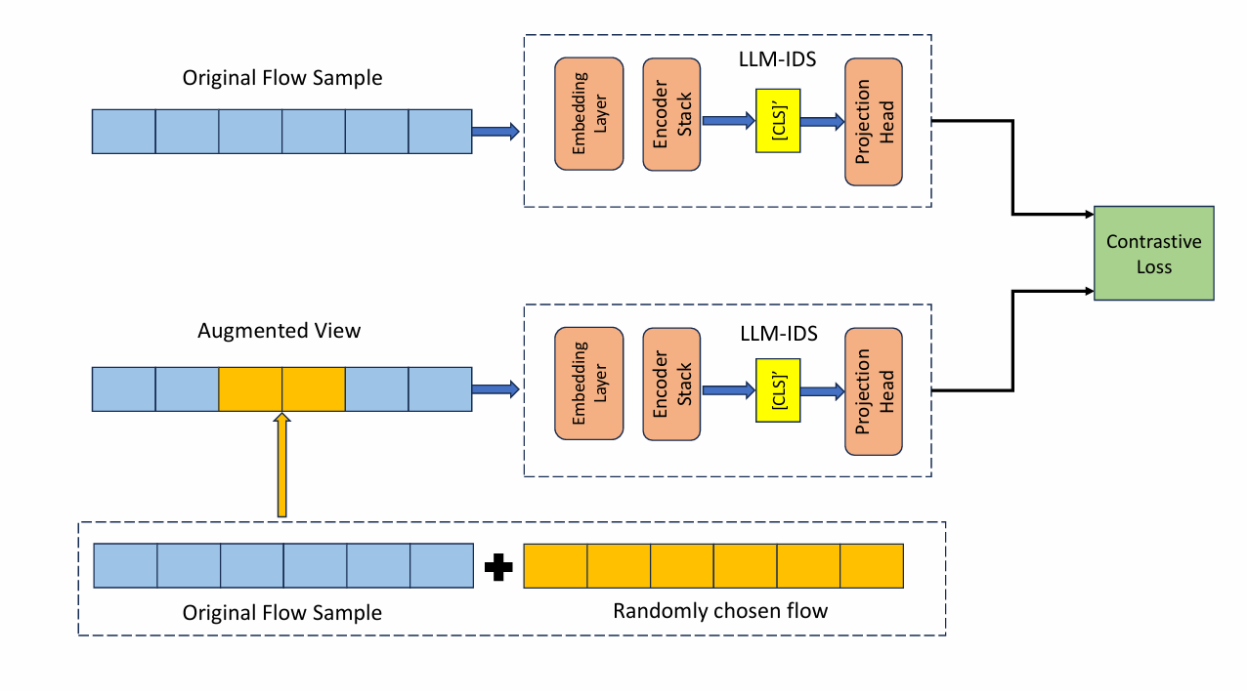

Figure 34: Overview of the contrastive learning and augmentation process

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 34. 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 [21] and UNSW-NB15 [22], 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.

7. Conclusions

The deliverable D3.1 “Secure-by-design orchestration and management & Data plane computation offloading” presented the first integrated view of NATWORK’s secure orchestration and programmable data plane capabilities, outlining both software architecture and initial implementation results. 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 above will continue to develop in the upcoming months of the project, 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 next phases of the NATWORK project, meeting their expected KPIs, providing technical maturity, integration feasibility and impact across diverse 6G verticals.

8. References

[1] Rohith Raj S, Rohith R, Minal Moharir, and Shobha G., “SCAPY - A powerful interactive packet manipulation program.” In 2018 International Conference on Networking, Embedded and Wireless Systems (ICNEWS), pages 1–5, 2018.

[2] 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.

[3] 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.

[4] 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

[5] 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.

[6] Free5GC, Open Source Core Network Implementation. Online: https://free5gc.org (Accessed 2025/06/25)

[7] 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).

[8] 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).

[9] 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).

[10] 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).

[11] 5G; NR; Medium Access Control (MAC) protocol specification 3GPP TS 38.321 V18.4.0 (2024, December).

[12] 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).

[13] O-RAN E2 General Aspects and Principles (E2GAP) 6.0, O-RAN.WG3.E2GAP-R004-v06.00 (2024, October).

[14] 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).

[15] O-RAN A1 Interface: Application Protocol 4.03, O-RAN.WG2.A1AP-R004-v04.03 (Accessed: 2025/05/20).

[16] 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).

[17] Robert Schmidt, Mikel Irazabal, and Navid 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.

[18] Ibrahim Obeidat, Nabhan Hamadneh, Mouhammd Al-kasassbeh, and Mohammad Almseidin. Intensive Preprocessing of KDD Cup 99 for Network Intrusion Classification Using Machine Learning Techniques, 2018.

[19] 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.

[20] Goethals, T., De Turck, F. & Volckaert, B. 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

[21] 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.

[22] 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.