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

D4.2 Payload security per runtime, intelligent runtime selection and attestation.r2

Lead beneficiary	IMEC	**Lead author**	Tom Goethals
Reviewers	Gürkan Gür (ZHAW), Andy Aidoo (UZH)		
Type	R	**Dissemination**	PU
Document version	V1.0	**Due date**	31/12/2025

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	WP4
Associated task(s)	T4.2
Main Author(s)	Tom Goethals (IMEC)
Author(s)	Shankha Gupta, Mays Al-Naday (UESSEX), Athanasios Papadakis, Asterios Mpatziakas, Vangelis V. Kopsacheilis, Antonios Lalas, Anastasios Drosou (CERTH), Vincent Lefebvre, Mark Angoustures (TSS), Andy Aidoo (UZH), Angelos Lampropoulos, Ioannis Markopoulos (NOVA)
Reviewers	Gürkan Gür (ZHAW), Andy Aidoo (UZH)
Type	R – Document, Report
Dissemination level	PU – Public
Due date	M24 (31/12/2025)
Submission date	31/12/2025

Co-funded by the European Union
Project funded by
UK Research and Innovation

Document version history

Version	Date	Changes	Contributor (s)
v0.1	12/08/2025	Template ready	Tom Goethals (IMEC)
v0.2	12/11/2025	Contributions for sections 1, 4, 5	Tom Goethals (IMEC), Shankha Gupta, Mays Al-Naday (UESSEX), Vincent Lefebvre, Mark Angoustures (TSS)
v0.3	24/11/2025	Draft version with all contributions from partners	Tom Goethals (IMEC), Shankha Gupta, Mays Al-Naday (UESSEX), Athanasios Papadakis, Asterios Mpatziakas, Vangelis V. Kopsacheilis, Antonios Lalas, Anastasios Drosou (CERTH), Vincent Lefebvre, Mark Angoustures (TSS), Andy Aidoo (UZH), Angelos Lampropoulos, Ioannis Markopoulos (NOVA)
v0.4	25/11/2025	Integration and draft review	Tom Goethals (IMEC)
v0.5	26/11/2025	Version ready for review	Tom Goethals (IMEC)
v0.6	05/12/2025	Document review	Gürkan Gür (ZHAW), Andy Aidoo (UZH)
v0.7	11/12/2025	Addressing review comments	Tom Goethals (IMEC)
v0.8	12/12/2025	Refinement based on review comments	All authors
v0.8.5	15/12/2025	Quality check	Joachim Schmidt (HES-SO)
v0.9	15/12/2025	Addressing quality check comments	Tom Goethals (IMEC)
v0.95	23/12/2025	Final review and refinements	Antonios Lalas (CERTH) and CERTH team
v1.0	31/12/2025	Final version for submission	Antonios Lalas (CERTH)

Disclaimer

Copyright message

Contents

Co-funded by the European Union

List of acronyms and abbreviations

Abbreviation	Description
1D-CNN	1D Convolutional Neural Network
AFH	Adaptive Frequency Hopping
AMF	Access and Mobility Management Function
API	Application Programming Interface
CLI	Command Line Interface
CNL	CloudNativeLab
CSI	Channel State Information
DDoS	Distributed Denial of Service
DFL	Decentralized Federated Learning
DoSt	Denial of Sustainability
EDA	Exploratory Data Analysis
FL	Federated Learning
FPGA	Field Programmable Grid Array
GL	Gossip Learning
HTTP	HyperText Transport Protocol
IID	Independent and Identically Distributed
IPC	Inter-Process Communication
LOS	Line Of Sight
ML	Machine Learning
MLP	Multilayer Perceptron
MQTT	Message Queuing Telemetry Transport
NLOS	No Line Of Sight
OAM	Open Application Model
OCI	Open Container Initiative
PCI	Peripheral Component Interconnect
PFCP	Packet Forwarding Control Protocol
QoE	Quality of Experience
RAN	Radio Access Network
REST	Representational State Transfer
RIC	RAN Intelligent Controller
RTSP	Real-Time Streaming Protocol
SDN	Software Defined Network
SINR	Signal-to-Interference-plus-Noise Ratio
SWIM	Scalable Weakly-consistent Infection-style Process Group Membership
TCP	Transport Control Protocol
TEE	Trusted Execution Environment
TPM	Trusted Platform Module
TSDB	Time-Series Database

Co-funded by the European Union 6GSNS Project funded by UK Research and Innovation

Abbreviation	Description
UDP	User Datagram Protocol
UPF	User Plane Function
VM	Virtual Machine
WASI	WebAssembly System Interface
WASM	WebAssembly
WMA	Weighted Moving Average

List of figures

List of tables

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

Executive summary

The deliverable D4.2 “Payload security per runtime, intelligent runtime selection and attestation.r2” is the second and final version of D4.1 “Payload security per runtime, intelligent runtime selection and attestation.r1” and provides a comprehensive overview of the NATWORK aspects enabling secure, flexible-runtime workloads. This is actually a report on the payload security per runtime, the intelligent runtime selection as well as the remote attestation, that derive from NATWORK innovations. The current deliverable (final version) derives from the work performed under the Task 4.2 “AIaaSecS for software payload”.

While containerized software and virtual machines have been employed for years in software orchestration, many security aspects of containers and security optimization through orchestration have been neglected in research and frameworks. On the one hand, containers are merely processes that are slightly isolated but not intrinsically secure. Alternative runtimes such as microVMs and the WebAssembly WASM System Interface (WASI) have become popular, promising superior workload security and optimal performance. Still, their exact contributions are unclear and integrating them into container-based software orchestration is non-trivial. The optimal runtime type should also be chosen for each payload based on security (and other) requirements.

On the other hand, several hardware-based approaches, such as Trusted Execution Environments (TEEs) and Trusted Platform Modules (TPMs), may be leveraged to secure deployed software through remote attestation by certifying nodes for secure and reliable execution of sensitive workloads.

This document covers four aspects of enabling secure, runtime-flexible workloads. First, an overview of the most promising execution formats and runtimes, including unikernels and WASM, is provided. The second aspect, intelligent runtime selection, involves selecting a suitable runtime for a specific workload based on specific parameters and workload properties and unifying network and orchestration aspects required for transparent, runtime-agnostic operation. Additionally, intelligent methods for network resilience are examined. Moreover, the third aspect provides the building blocks for a feasible, flexible remote attestation framework. WASM payload security is discussed similarly to remote node attestation, and beyond during the payload execution. We discuss the core merits of security technology, emphasizing bytecode tampering at runtime. The project addresses this major weakness using a modified WASM runtime and a full-stack remote attestation schema covering both the WASM runtime and the actual executable payload. Finally, an AI runtime anomaly detection system for containerized applications is described that utilizes granular kernel-level monitoring.

1. Introduction

Containerized software and virtual machines have been employed for years in software orchestration, both in the cloud and on the edge. However, research and frameworks have neglected many security aspects of both containers and security optimization through orchestration. This is a significant shortcoming for 6G frameworks, as edge networks are more vulnerable to attack vectors than physically and digitally secured data centers, and 6G inherently spans the entire continuum from the cloud to the edge.

In security terms, containers are merely processes that are slightly isolated from the rest of the operating system but not inherently secure. However, containers are not the only option for executing workloads (i.e., a "runtime"). Alternative runtimes such as microVMs and WebAssembly (WASM) System Interface (WASI) have become popular, promising superior workload security and optimal performance. Still, their exact contributions are unclear, and integrating them into container-based software orchestration is no easy task. Furthermore, there exists the need to define the optimal runtime for each payload based on security (and other) requirements.

On the other hand, several hardware-based approaches, such as Trusted Execution Environments (TEEs) and Trusted Platform Modules (TPMs), can be leveraged to secure deployed software through remote attestation by certifying nodes as secure and reliable for the execution of sensitive workloads.

Considering the heterogeneity of 6G environments and the scalability required of frameworks, this document examines the properties of runtimes and secure environments, as well as how to describe them formally (i.e., intent-based), and how to leverage them through decentralized intelligent orchestration optimally.

1.1. Purpose and structure of the document

This document builds on previous work reported in D4.1, covering the same ongoing aspects researched in T4.2 to enable secure, runtime-flexible workloads. In summary, these are:

- An overview of the most promising execution formats and runtimes, including unikernels and WASM. This includes state-of-the-art capabilities, resource use, and essential security aspects.
- Intelligent runtime selection, which is split into "enabling runtime selection" and "intelligent selection". The former details how various runtimes can be integrated into a uniform structure/API based on the most popular orchestration frameworks, such as

Kubernetes, in preparation for intelligent intent-based selection. The latter considers workload and runtime properties, determining the optimal workload for a specific task and matching it to a suitable execution node.

- Building a remote attestation framework that ensures secure execution of the selected workloads by leveraging node hardware capabilities such as TEE and TPM.
- Demonstrating an AI runtime anomaly detection system for containerized applications that utilizes granular kernel-level monitoring.

The rest of the document is structured according to these four main aspects:

Sections:

2. **Runtime overview & security:** provides a link to the sections of D4.1 that present state-of-the-art security aspects of the most promising execution formats and runtimes for deployable workloads. This includes execution and image payload security for Virtual Machines (VMs), containers, microVMs, and WASM. The inception of WASM technology is recalled in the context of security.
3. **Intelligent runtime selection:** covers the intelligent, intent-based selection of a suitable runtime for a specific workload based on parameters and workload properties, in addition to standardization and unification efforts in terms of networking and orchestration APIs. Finally, this section covers ML-based workload modeling in the context of resource optimization.
4. **Remote Attestation:** describes TEE and TPM-based attestation methods as the building blocks for a scalable, flexible attestation framework. Moreover, WASM payload remote attestation is discussed, exploiting a novel runtime integrity verification scheme through a modified WASM runtime.
5. **Containerized Runtime Protection and AI-driven Payload Security:** details the implementation of an enhanced containerized 5G security testbed designed for real-time runtime protection and experimentation. This includes the integration of granular kernel-level monitoring and live offensive capabilities to establish precise operational baselines and simulate payload attacks. Finally, the section validates a novel Hybrid AI ensemble—combining statistical, neural network, and tree-based models—for the predictive detection of anomalies and resource exhaustion targeting microservices.
6. **Conclusions:** Wraps up the document, reflecting on the project's strategic orientation and establishing expectations for future milestones.

1.2. Intended Audience

NATWORK’s D4.2 “Payload security per runtime, intelligent runtime selection, and attestation.r2” is devised for public use in the context of design and implementation of security services of the NATWORK consortium, comprising members, project partners, and affiliated stakeholders. This document primarily focuses on the fundamental security aspects of software runtimes and hardware attestation required for the project, thereby serving as a referential tool throughout the project's lifespan.

This deliverable is a Public (PU) document. It is primarily intended for the NATWORK consortium, the European Commission, and the wider research and industrial community interested in secure runtime environments and 6G systems.

1.3. Interrelations

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

NATWORK is categorized as a "Research Innovation Action – RIA" project and is methodically segmented into 7 WPs, further subdivided into tasks. With partners contributing to multiple activities across various WPs, the structure ensures clarity in responsibilities and optimizes communication amongst the consortium partners, boards, and committees. The interrelation framework within NATWORK offers smooth operation and collaborative innovation across the consortium, ensuring the interconnection of the diverse expertise from the various entities (i.e., Research Institutes, Universities, SMEs, and large industries), enabling scientific, technological, and security advancements in the realm of 6G. The D4.2 deliverable addresses all activities of the NATWORK project related to T4.2 for the period between D4.1 (M15) and M24. D4.2 reports on activities directly related to T4.2, some of which are related to T3.1 (optimizing selection and embedding of AI security services, i.c.w. UESSEX, CERTH, and ZHAW as main stakeholders) and UC1 (Sustainability and reliability of 6G Slices and services i.c.w. UESSEX, ISRD and TSS).

2. Runtime overview & security

This section provides an overview of the security and functional properties of various runtimes relevant to T4.2 and the NATWORK project as a whole. Starting with VMs, it moves through lightweight alternatives such as containers and microVMs, to promising future runtime formats, including WebAssembly.

2.1. Virtual Machines

In D4.1, VMs were examined for their security properties, compatibility with existing systems and software. VMs benefit from a mature ecosystem and robust hardware-enforced isolation mechanisms to confine compromises, reducing the potential for lateral movement to other VMs. Section 2.2 in D4.1 further introduced unikernels – a lightweight virtualization paradigm that solely retains the essential functions of applications and operating system within a single executable.

Numerous studies have compared properties like startup times, image size, and CPU overhead in selected applications. Generally speaking, unikernels exhibit smaller image sizes and considerably faster startup times compared to VMs [1],[2]. While primitive CPU-bound workloads do not exhibit performance overhead in either VMs or unikernels, more complex tasks suffer from performance penalties in unikernels [2],[3]. Although unikernels may not yet consistently outperform VMs, their potential should not be underestimated, as much of their current limitations stem from their relative immaturity.

2.2. MicroVMs

The examination of microVM properties and security benefits for the NATWORK project was completed in Section 2.2 of D4.1. However, additional performance insights have been collected, which are discussed in detail in section 3.1 of this document.

2.3. Containers

The examination of Container properties and security benefits for the NATWORK project, along with a testbed suitable for investigating containerized application security was completed in Section 2.3 of D4.1. However, additional performance insights have been collected, which are discussed in detail in section 5 of this document.

2.4. WebAssembly

The security aspects of WebAssembly (WASM), including the technology's exposure to bytecode tampering during runtime, were examined in D4.1. To mitigate this risk, D3.5 detailed a method for verifying WASM runtime integrity, leveraging an enhanced runtime capable of capturing the WASM bytecode footprint once it is loaded and generating a corresponding quote. We continue with our ongoing roadmap to produce the full-stack WASM remote attestation and runtime integrity verification as defined in D3.5, by leveraging the benefits of containerization as described in Sections 4.2 and 4.3.

3. Intelligent runtime selection

This section discusses continued work on the enablers of intelligent runtime selection, since this deliverable continues the work presented in D4.1. More specifically it elaborates on the following subjects:

  - API unification which ensures that the selectable runtimes are easily interchangeable and accessible through a common network structure.
  - Use of standards, which applies existing standards, e.g., Open Container Initiative (OCI), to other runtimes or implements more generic standards in a unified platform. Further progress has also been made in intent-based workload selection.
  - Network resilience through intelligent anti-DDoS solutions.
  - Although workload modeling and resource optimization are currently limited to workload modeling, they can be easily extended to runtime-dependent properties, as the same workload over different runtimes is likely to have varying resource usage and optimization parameters.
- Resulting from new lines of research in T4.2:
  - Improvements to Gossip Learning,
  - Integrated intent-driven implementation in Flocky to support fully decentralized and organic AI learning dissemination.

3.1. API Unification

Feather [4] was introduced in D4.1 as a Kubernetes-compatible agent that supports multiple runtimes for workloads, fully compatible with OCI standardized images. Feather was the first solution of its kind to generically support multiple runtimes side by side in a single pod, as well as enabling mixed-runtime pod networking. For D4.2, a WebAssembly backend has been integrated into Feather as an alternative to containers and OSv unikernels [1]. Specifically, WasmTime [5] is supported through CLI operation (rather than direct API access). As a reminder, Feather architecture is shown in Figure 1.

[1] https://github.com/cloudius-systems/osv

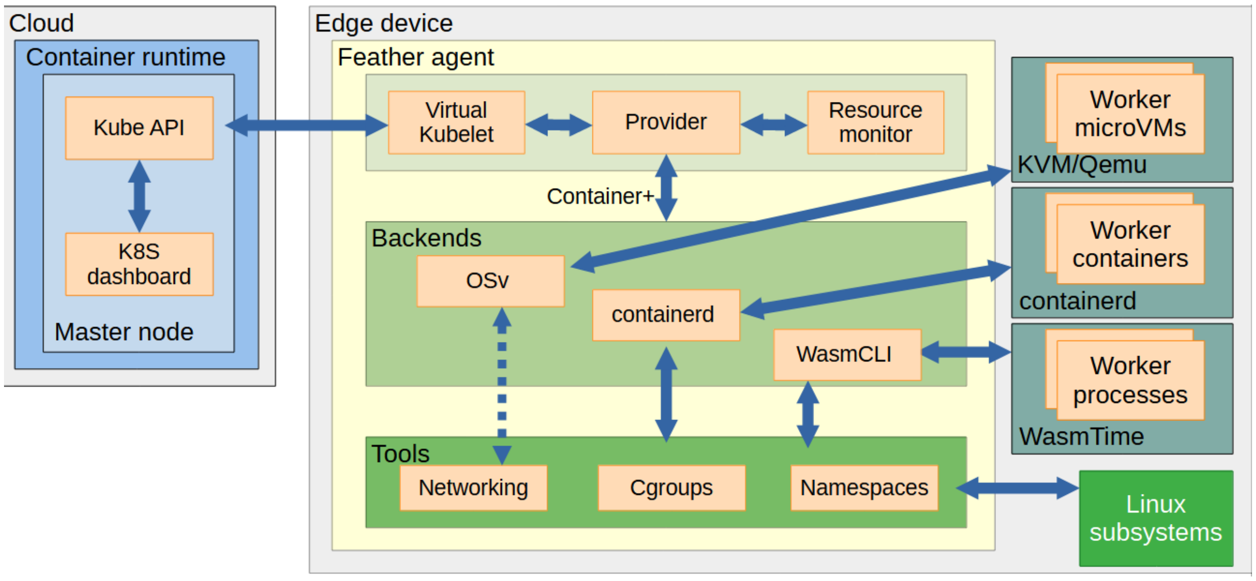


Figure 1: Feather architecture as of D4.2, including WasmTime

The Feather agent (*Cf*. D4.1 for further details) runs on edge devices to provide a low-resource and multi-runtime alternative for the Kubernetes kubelet. It may connect to a Kubernetes cluster and receive pod deployment manifests or operate standalone and load manifests from disk or through a REST API. Received pod manifests are handled by the Provider, which is the Feather-specific implementation of a Virtual Kubelet[2] component. The Provider examines the incoming manifest, determines the correct backend to handle each workload, and constructs Container+ objects to send to the relevant backends. Each Container+ object contains a workload description (i.e., Kubernetes Container definition) and pod-level information, such as IP address, network namespace, and the pod network interface. The backends then execute the required steps to run the workloads, applying networking rules, volume mounts, and resource restrictions as suitable for their respective runtimes.

The WASM backend leverages the Flint tool to package WASM binaries into an OCI image in the same manner as OSv disk images are handled by the OSv backend. The full deployment method is identical to that of containers and OSv unikernels, and previously detailed in D4.1 Figure 3.

Evaluations were performed to gauge various aspects of the current WASM backend. A Rust HTTP server is compiled into different formats, specifically tiny_http for the container and the OSv unikernel, and WASI-http for WASM. Both versions are identical in functionality, differing only slightly in implementation. A single REST endpoint is exposed, which is used to generate load and evaluate performance.

[2] https://virtual-kubelet.io/

Interestingly, image size is by far in favor of WASM workloads, as shown in Figure 2. The size of an “idle” image is also shown, showing the overhead of the target platform itself without the HTTP server. In both cases, the WASM image is over two orders of magnitude smaller than the container image, although some optimization of the Dockerfile may be possible (e.g., an optimized base image). Despite being microVMs containing an entire kernel and all required system libraries, the OSv unikernel images are also several times smaller than the Docker containers, showing excellent optimization.

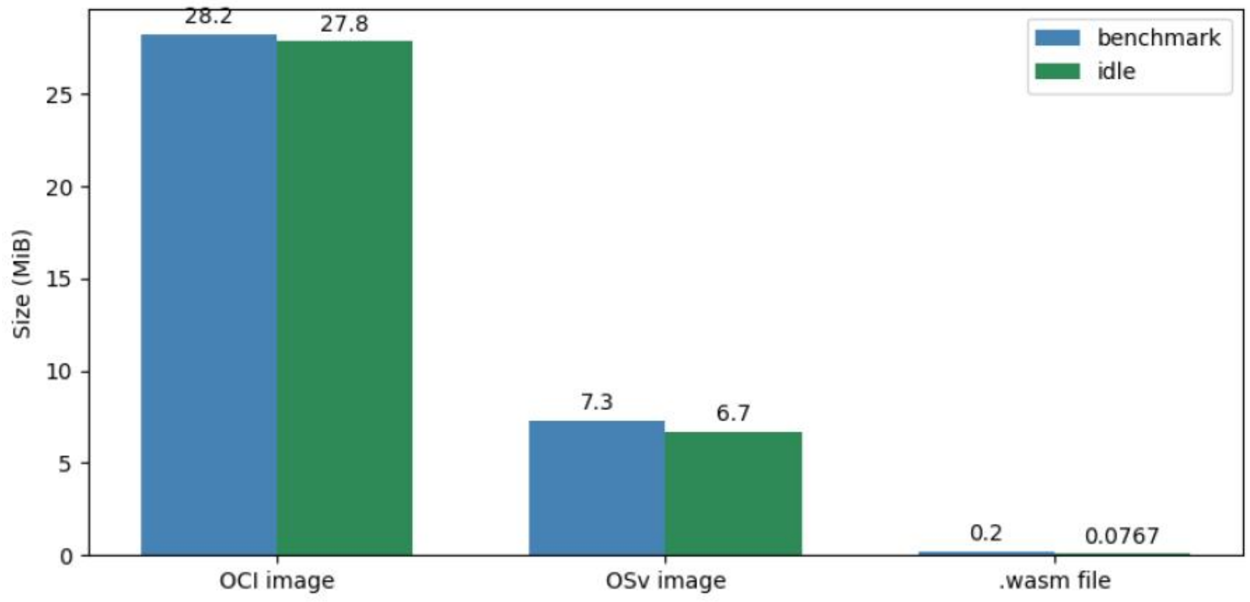


Figure 2: Image size comparison for HTTP server in a container, OSv unikernel, and WASM binary

Figure 3 shows the memory overhead of the various backends, showing a clear advantage for WasmTime; containerd causes a higher memory usage within Feather itself, while QEMU results in a significant memory overhead to start a microVM process. Total memory use is highly dependent on the workload, as shown in D4.1, where memory-intensive processes were determined to be more efficient in a microVM.

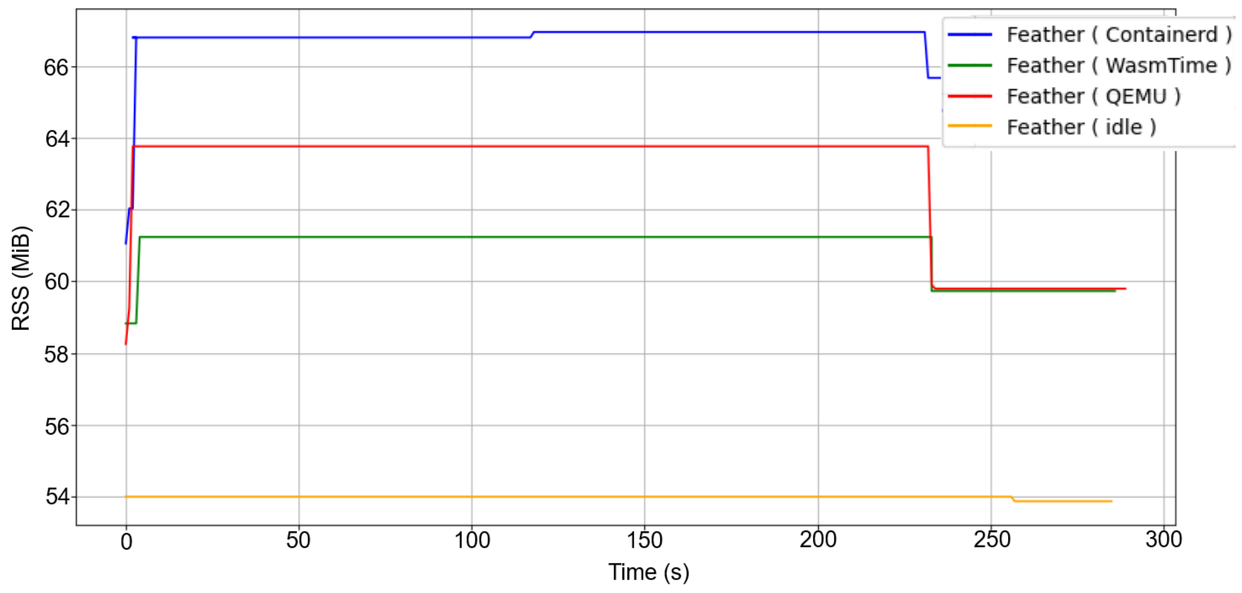


Figure 3: Memory overhead of Feather idle, and after initializing the various backends

Finally, Figure 4 and Figure 5 show the HTTP REST throughput and latency results, respectively. While WASM throughput is exactly on par with native (container) performance, latency is around 50% higher due to slower per-request processing. As observed in previous evaluations (including D4.1 and NATWORK related publications), OSv REST performance is hamstrung at some point, causing exponential latency buildup while throughput is reduced to near zero.

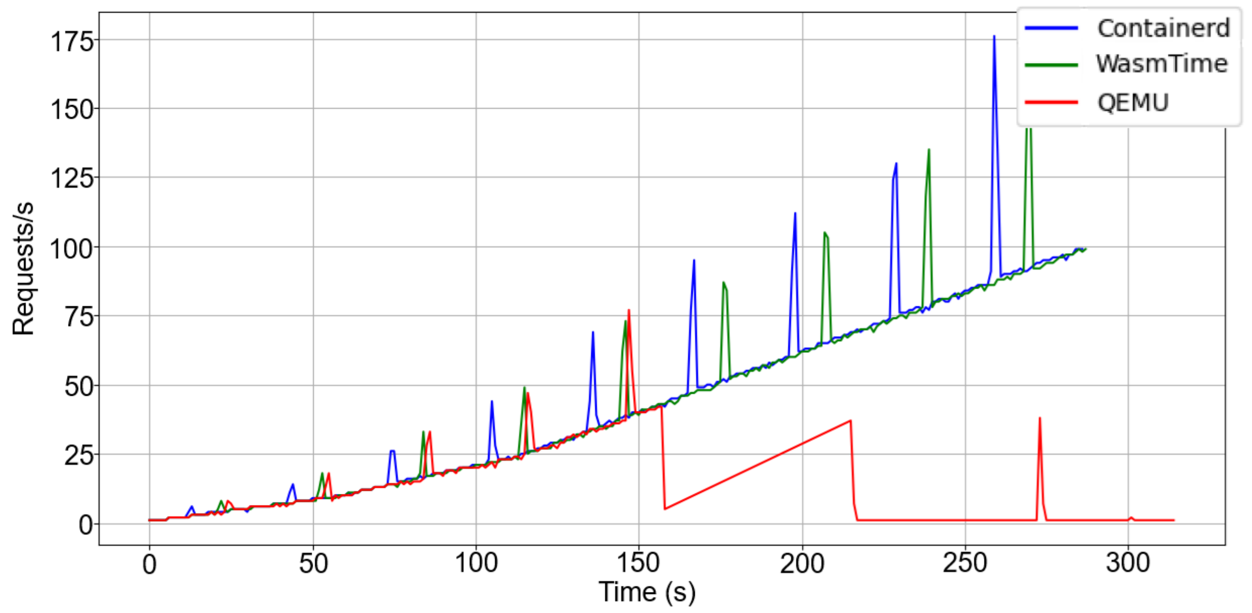


Figure 4: Requests per second handled by HTTP servers in various backends, OSv performance decreases as a result of queuing and threading issues

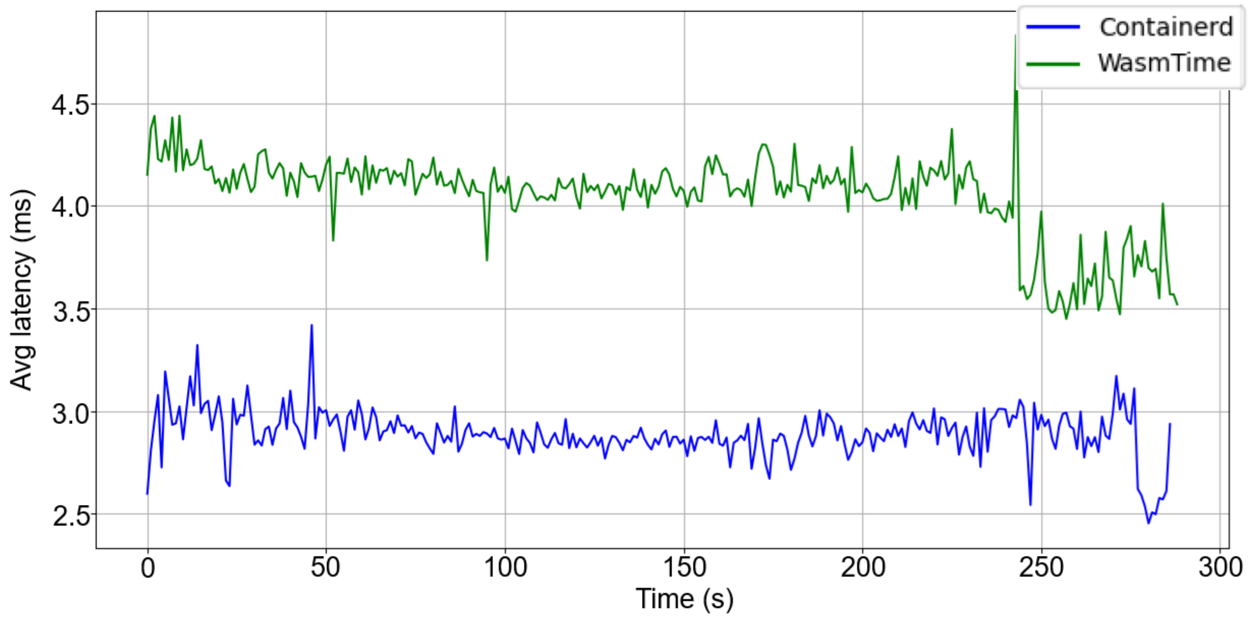


Figure 5: Request latency over time for containers and WASM, OSv latency (not included) is an order of magnitude higher and eventually spirals out of control as requests/s drops

To summarize, WASM promises to be an excellent addition to runtime unification through Feather and useful for intelligent runtime selection depending on the type of workload. Results show that, at least for WasmTime, WASM is useful for storage-constrained devices but slightly slower than containers. However, additional work is required to examine other WASM runtimes (e.g., WAMR, WasmEdge), which may yield better performance. Conversely, WASM preview version support, component model support, and security features are also highly dependent on low-level WASM runtime implementation. As such, this topic merits follow-up research.

Finally, this line of research also revealed some insights into cross-runtime security. When multiple workloads are executed in a single pod, the security gains from different runtimes are often limited to the isolation features provided by the runtime itself. At the network level, there may be important risks that are frequently overlooked in a container-only pod architecture. Specifically, all workloads attached to the same network interface (i.e., containers and WASM) may sniff each other's traffic. While this is also the case in container-only pods, network security may be improved by using microVMs for sensitive traffic. However, even in this case, other workloads can still sniff incoming traffic meant for different workloads. Future work may provide more secure networking approaches at the workload level.

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

3.2. Network Resilience

As part of a broader strategy to strengthen network resilience, the anti-DDoS solution is implemented on a SmartNIC equipped with an FPGA. This hardware platform enables real-time traffic analysis directly on the physical network link, combining machine learning-based DDoS attack detection with high-performance packet processing written in P4. By offloading these functions to the SmartNIC, the system ensures minimal latency, deterministic behavior, and robust protection, even under high-volume attack conditions.

The SmartNIC is deployed within a 5G/6G base station, where its PCI Express interface provides a direct high-speed connection to the host server. Thanks to this architecture, the solution can be extended during a later development phase to monitor traffic from virtual machines and/or containers running on the host itself. This capability enables the system to detect malicious activity initiated by hosted services, identify their sources, and take appropriate mitigation measures. Not only does the platform protect the radio access network from external threats, but it also provides defense in depth by preventing internally generated attacks and quickly alerting operational teams.

Complementary to the previous approach, protecting the network from DDoS attacks, the protection also aims to mitigate jamming attacks. This novel approach uses the current metrics provided by the UE for channel control and quality, such as CSI and SINR. It analyzes them in real-time using an xApp in the near RT-RIC (Real-Time RAN Interface Controller) layer. This approach is fully compliant with the direction of 6G and O-RAN standardization. Preliminary offline analysis of the logs from gNodeB using time series algorithms, such as the Pettitt test, enabled the correct detection of the jamming. Currently, we need to generate more data with different cases, *e.g.*, LOS (Line-Of-Sight) or NLOS (Non-Line-Of-Sight). The advantage of this approach is that we can deploy our solution anywhere near gNodeB and use minimal additional hardware resources, as the metrics are already available. By using these metrics, the goal is to propose a solution such as AFH (Adaptive Frequency Hopping); this way, we utilize the infrastructure already in place. This AFH would allow the UEs to move to different frequencies using the scheduler in an optimal way and therefore improve the communication channel and increase the throughput.

3.3. Use of standards

Compliance of the developments in T4.2 with existing standards and practices is key to long-term compatibility and extensibility. Most efforts to standardize research outputs were documented in D4.1; this section provides details on additions and minor changes that adhere to the same standards.

3.3.1. Kubernetes & Open Container Initiative

Due to its implementation in Feather, the integration of WASM workloads from Section 3.1 is fully compatible with existing Kubernetes clusters and OCI (image) standards. Additionally, the solution is easily integrated with multi-runtime networking using existing means, specifically by attaching the WASM processes to an existing pod network namespace and assigning them a unique IP address, allowing for the complete integration of WASM workloads into multi-runtime pods without modifying the Kubernetes control plane.

3.3.2. Open Application Model

The OAM[3] implementation in Flocky [6] has been extended to AI workloads and Gossip Learning. The general OAM implementation (detailed in D4.1) remains unchanged. Section 3.4.2 further details the architectural changes in Flocky; the relevant concrete additions for OAM are:

- Traits describing node machine learning capability: hardware acceleration, CPU/GPU support for TEEs, and attestation
- Traits indicating a machine learning workload and whether it should use Gossip Learning (including selection of update integration method)

3.4. Intent-based selection

Intent-based selection is achieved through Flocky, which was introduced in D4.1. Flocky is a framework for decentralized, intent-based metadata gathering and orchestration. Its main goal is to allow any (edge) device to deploy an OAM Application consisting of multiple Components to multiple discovered target devices, depending on the requirements of each Component and the target devices' capabilities.

This section details the changes to Flocky architecture from D4.1 and describes how new functionality and methods for decentralized learning have been implemented using Flocky as a basis.

3.4.1. Decentralized learning

The Flocky orchestrator can be leveraged for decentralized learning, a more natural fit for edge computing than Federated Learning (FL), which requires centralized components that may result in (networking and computing) bottlenecks. Gossip Learning (GL) is a more organic approach than

[3] https://oam.dev/

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

FL in which nodes communicate model updates only to their direct neighbors and rely on multi-hop propagation to disseminate their knowledge throughout the network topology. These properties make GL an excellent fit for several aspects of the NATWORK project. However, the basic averaging methods used for model integration in both FL and GL are far from ideal, as the rest of this section will show. Instead, a new approach is devised that has both better and faster convergence, as well as higher global accuracy.

3.4.1.1. Gossip Learning & existing approaches

FL and, by extension, Decentralized Federated Learning (DFL) have been extensively studied [7] for their potential in neural network training. Specifically, a variety of model averaging or aggregation methods can be used, such as Federated Averaging, various Secure Aggregation [8] methods which mitigate privacy risks by obfuscating individual model updates, or Matched Averaging [9] methods aiming to reduce communication overhead. Furthermore, the effects of network topologies [9] and data distribution [11] on the performance of (D)FL have been extensively studied, as well as the mitigation of noisy communications channels [11].

Several methods have been proposed that improve specific security aspects of (D)FL, including Byzantine-Robust FL [13] for adversarial node behavior, or Distributionally Robust averaging [14] which considers Non-Independent and Identically Distributed (Non-IID) data properties. Additionally, some methods aim to improve specific aspects of integration, such as vanishing variance issues [15]. Finally, Xiao et al. argue that straightforward averaging methods are far from ideal for distributed learning [16], showing that despite an increased correlation between model outputs on all nodes, the computed distance between those models stops decreasing. As a result, the models never fully convergence towards an optimum.

GL [17] has been proposed as a fully decentralized variant of DFL, using gossip algorithms to spread model updates across complex topologies. In GL, each node only directly contacts a small set of neighbors to push updates, typically through a gossip protocol such as SWIM [18]. Gossip protocols may also benefit from a push-pull approach [18], which has been shown to offer superior update dissemination compared to push-only approaches as implemented in most DFL approaches. Inheriting many (dis)advantages from DFL, its performance is shown to be on par with FL depending on hyperparameters, network data compression, and integration methods [20].

Existing frameworks, such as EdgeFL [20], offer a variety of options for running FL algorithms in the network edge; however, unlike the proposed framework they do not present a generic orchestration framework that allows ML workloads and GL to be used as components of a larger application.

There are several frameworks that solve more specific FL issues. For example, DeFTA considers the network outdegree of contributing nodes in DFL to weigh their updates [22] and integrates it into a DFL framework. Fedstellar [23] offers a similar framework, with various options to run DFL workloads, showing its effectiveness in attack detection.

3.4.1.2. Custom approach for global & local learning

Delta Sum Learning is devised as a customized approach for GL update integration, aiming to be more efficient than the default averaging approaches used in FL and GL. Examining integration operations, we found that averaging fundamentally contains an implicit factor N (the number of gossip neighbors) in the denominator, which slows down learning and might result in reduced global accuracy:

$$\boldsymbol{W}_{t+1} = \boldsymbol{W}_t + \frac{\sum_{n=0}^{N} k_n \Delta \boldsymbol{W}_{n,t}}{\mu_{kn} N}$$

where **W** is the weights matrix of the local node at a certain timestep, N is the number of neighboring nodes, k is the number of samples for each neighbor, delta w is the model difference, and μ is the standard deviation of sample count across neighbors. Starting from basic model updates using gradient descent on a single node and extending this to an analogous form for multiple nodes, we constructed a model for Delta Sum Learning, which eliminates this factor. The model can be summarized as:

$$\boldsymbol{W}_{a,t_0+T} = \frac{\sum_{n=0}^{N+1} \boldsymbol{W}_{n,t_0}}{N+1} + \lambda(t_0 + T) \sum_{n=0}^{N+1} \Delta \boldsymbol{W}_{n,t_0+T}$$

$$\lambda(t) = \min\left(A + \frac{t}{B}, C\right)$$

With each node forming its new model at t_0 + T by averaging the base models of each neighbor at t_0, and adding the sum of all neighboring model differences from t_0 to t_0+T. The latter is modulated by a gossip learning factor λ, which is time-dependent and requires constants A, B, and C, determined by individual use cases. The learning factor is used to dampen the effects of erratic updates during the initial stages of learning, when updates from different nodes may be highly conflicting due to non-IID data. The time-dependent nature of the factor ensures that in later stages, the combined learning of the topology is favored over local learning, which may otherwise undo the effects of gossip updates.

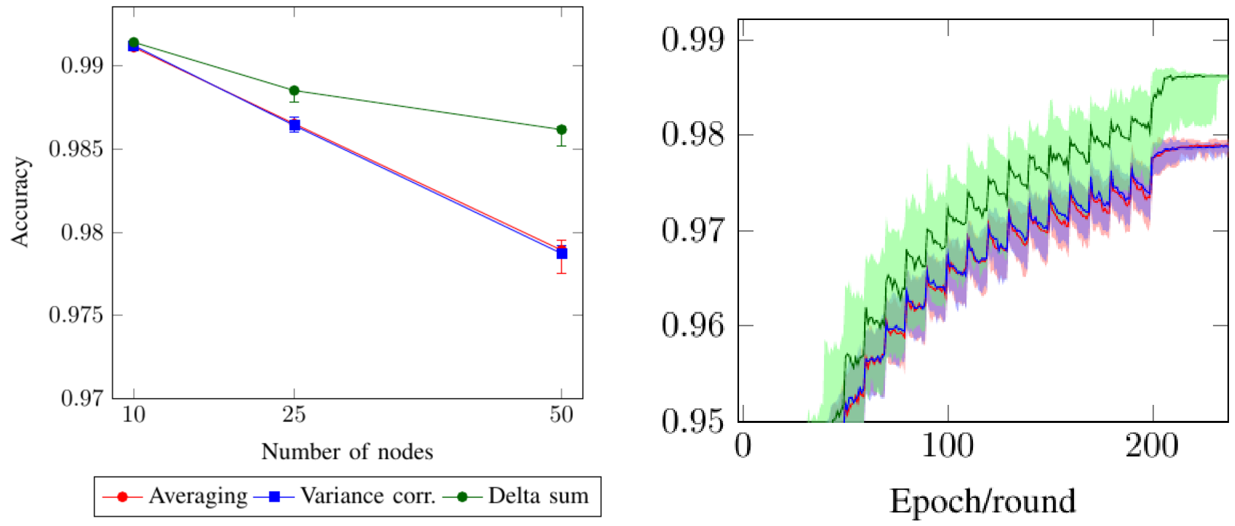


Figure 6a: Accuracy of gossip learning integration strategies for various topology sizes, 11b: range of training accuracy over time for different integration strategies

Figure 6a shows the effectiveness of Delta Sum learning compared to standard Averaging and Variance-corrected Averaging approaches. As the number of nodes in a topology scales, both averaging approaches result in a linear loss of global accuracy, whereas Delta Sum learning results in a logarithmic loss, which may be further mitigated by considering data distribution. Accuracy during the entire training process is shown in Figure 6b, illustrating that Delta Sum not only results in faster learning, but in later stages it also results in far lower accuracy loss during local training epochs which occur between integration rounds of neighboring model updates. Finally, Delta Sum also converges towards the nodes with the highest global accuracies, whereas default averaging strategies converge towards the median nodes.

While this method does not consider the effects of the underlying network topology and non-IID data explicitly, evaluation shows that node connectedness has a low to negligible impact on learning; the Delta Sum accuracy distribution over all nodes is very small despite connectivity ranging from 1 to 7 neighbors, with an average of 3.2 to 4.2 depending on topology size. Finally, Delta Sum learning is compatible with several security improvements to FL and GL, including obfuscation techniques that average or combine several model updates (e.g., Secure Aggregation) to avoid backward analysis of individual data points.

3.4.2. Intent modeling & detection

Figure 7 shows the architectural additions to Flocky to support ML workloads and learning (marked in purple). Concretely, two services have been added:

- The Gossip Service enables generic gossip dissemination of workload data using the SWIM protocol. In the case of ML, this consists of model updates (3). Workload data is handled differently from orchestration metadata (2); the latter is discovered and indexed by the Discovery and Repository services and concerns mostly OAM objects used for orchestration purposes, while the latter is arbitrary data used by workloads, which should stay private to the workloads. To that end, the Gossip Service allows workloads to link data to specific keys, which are not shared with other services and workloads. In future work, the Gossip clusters themselves will also be separated to avoid dissemination of sensitive data to nodes that do not run services related to that data and to optimize network traffic. The Gossip service enables the "Gossip" part of GL and can be used either by the ML service or directly by a workload by specifying Gossip keys and data exchange endpoints (REST or Shared Memory) using workload Traits (GossipTrait or GossipLearningTrait). In the case of REST endpoints, updates are timestamped for easy versioning of data.

- The ML service handles node-level ML capability detection (1), advertising hardware acceleration, TEE support, and supported learning algorithms as node Traits. This service interacts with the OS and hardware using NVIDIA's Management Library (NVML) through the go-nvml library. This library allows fine-grained feature detection, for example, the level of CPU and GPU TEE support and hardware models/types. This service also handles the specifics of deploying and monitoring ML workloads (if any) and applies GossipLearningTraits to workloads that require learning algorithms. These traits specify a model integration algorithm to use and contain configuration that allows the ML service to set up a bridge between workload endpoints (REST or Shared Memory) and the Gossip Service. The supported integration algorithms are standard Averaging, Variance-corrected Averaging, and Delta Sum.

In addition to these services, node resources have been streamlined and are defined uniformly and flexibly across types of resources. This extensibility allows for easier implementation in custom orchestration algorithms (QoE Evaluators as part of the Swirly Service) and resource-agnostic checks in preliminary node selection (Swirly & Deployment Services).

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

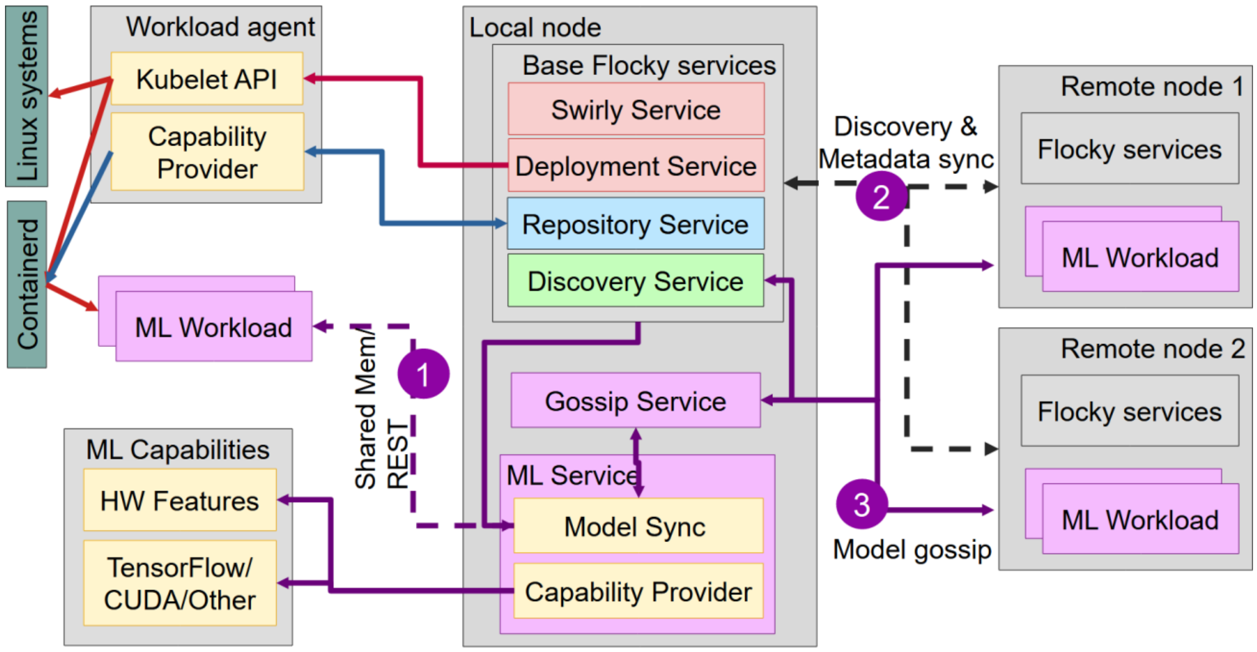


Figure 7: Architecture of Flocky services as of D4.2. For details on Base Flocky services, refer to D4.1 Figure 16

3.5. Data Collection Methodology for ML services

To ensure the development of reliable and representative machine learning models for detecting DDoS attacks, a controlled and fully instrumented testbed is deployed. This environment is designed to reproduce both legitimate network behavior and malicious attack scenarios under realistic conditions.

The testbed consists of a variety of legitimate devices that generate authentic network traffic across multiple protocols, including HTTP, MQTT, RTSP, and UDP. Additional client-server services can be introduced as needed, each deployed in a container using Docker to ensure consistency, portability, and ease of orchestration.

Concurrently, compromised devices infected with a variant of the Mirai malware are integrated into the same test environment to serve as authentic attack sources. These instances of malware are confined to a virtual machine to ensure strict isolation. They are controlled externally without exposing any network interfaces, relying instead on Linux inter-process communication (IPC) mechanisms to prevent unintended propagation. The malware's self-propagation capabilities are disabled by default but can be re-enabled to meet specific experimental needs.

To capture and analyze the resulting traffic, a dedicated probe records all network flows in PCAP format. A fully containerized processing pipeline, incorporating tools such as CICFlowMeter, is then used to convert the raw packets captured into Parquet datasets. These processed datasets serve as the basis for model training and subsequent inference procedures.

This methodology ensures that collected data reflects realistic operational conditions while maintaining a secure, controlled, and ethical experimental environment.

3.6. Decentralized workload modeling & resource optimization

3.6.1. Dost Attack Dataset generation

The Denial of Sustainability (DoSt) [24] attack induces oscillatory and rapidly fluctuating HTTP request patterns, forcing continuous scale-in and scale-out events in Kubernetes-based cloud-native network function (CNF) services. Unlike traditional denial-of-service attacks, DoSt does not crash services but instead causes persistent CPU and memory oscillations, degrades QoS, and causes unsustainable energy consumption.

To support decentralized workload modelling and anomaly detection, custom DoSt datasets are being generated within the edge–cloud testbed in the Network Convergence Lab (NCL) at the University of Essex, using containerized traffic generators that emulate both benign and adversarial user behavior. Prometheus [25] is used to monitor CNF services, capturing service-level, cluster-level, and VM-level telemetry, which also reveals the broader impact of DoSt on Kubernetes components such as the API server and scheduler. Prometheus stores all collected metrics in its time-series database (TSDB) format, enabling high-resolution monitoring of resource behavior.

Prometheus scrapes CPU, memory, and network metrics every 15 seconds from CNF services during these scenarios. The telemetry is ingested into the Prometheus TSDB and periodically exported to MinIO [26] object storage for reliable, scalable dataset retention. These datasets, containing clearly distinguishable benign and DoSt-induced resource patterns, form the foundation for preprocessing, feature engineering, traffic classification, and FL-based resource optimization.

3.6.2. Data Engineering and Preprocessing

Telemetry collected from the testbed is stored in Prometheus TSDB and periodically exported to MinIO object storage as raw Prometheus chunk files, pushed in two-hour intervals. These chunks will be decoded and transformed into structured JSON/CSV formats, forming the basis for clean, model-ready datasets.

While preprocessing the custom DoSt datasets is still in progress, the data engineering workflow has already been validated using historical Google workload traces, which serve as the baseline for developing and testing the ML pipeline. For these traces, data has been structured for two prediction granularities: a 5-minute horizon for fine-grained forecasting and a 40-minute horizon aligned with real orchestration decision intervals. Feature extraction includes chronological ordering, lagged features, rolling statistics, and workload categorization for time-series modeling. Exploratory Data Analysis (EDA) is performed to understand workload distributions, detect anomalies, and derive correlations, establishing a preprocessing strategy that will later be applied consistently to the custom DoSt datasets for both classification and resource-prediction models across centralized and federated setups.

3.6.3. Federated Learning

Federated Learning (FL) is being developed to support decentralized workload prediction and anomaly detection across the edge–cloud environment. Two models are being developed under this framework: a resource-optimization model that predicts workload trends and advises the orchestrator on energy-aware scaling decisions, and a traffic-classification model aimed at identifying DoSt-induced oscillatory patterns and distinguishing them from benign demand surges. These models feed as crucial components into the orchestrator from Task 3.1 in WP3.

Initial benchmarking has been performed using historical Google cluster traces [27] as baseline datasets, where multiple ML models were evaluated for workload prediction; XGBoost consistently delivered the best performance. Based on these findings, Federated XGBoost has been selected as the baseline approach, using a tree-bagging strategy for decentralized training across distributed nodes. Development and testing of these models are currently underway within the testbeds at NCL. We intend to demonstrate both the centralized and decentralized FL architectures as part of the evolving system design.

3.6.3.1. Centralized

A centralized FL prototype is currently being developed and tested on Kubernetes to validate the training workflow and aggregation logic. In this setup, each FL client trains locally on its own private data, generating incremental XGBoost trees during each round while keeping all raw data fully isolated. A central aggregator then collects these tree updates, merges them into a unified global model using a federated bagging strategy, and shares the updated booster back with the clients for the next training cycle. This ongoing implementation provides an end-to-end proof-of-concept for federated, tree-based learning and establishes a baseline before we transition to more decentralized architectures. The current implementation uses historical Google workload traces for experimentation; the same pipeline will later be replicated using our custom DoSt datasets as they become fully processed and structured for FL training.

3.6.3.2. Decentralized

A fully decentralized FL architecture will be developed as the next stage, as demonstrated in Figure 8. In this design, no central server will be required; instead, each client will exchange model updates directly with other clients through a publish–subscribe mechanism. This approach will support intra-cluster and cross-cluster collaboration with technologies such as Submariner, enabling multi-cluster communication [28]. Model parameters will be propagated in a peer-to-peer fashion to enable resilient, scalable, and privacy-preserving learning suitable for distributed 6G-core environments.

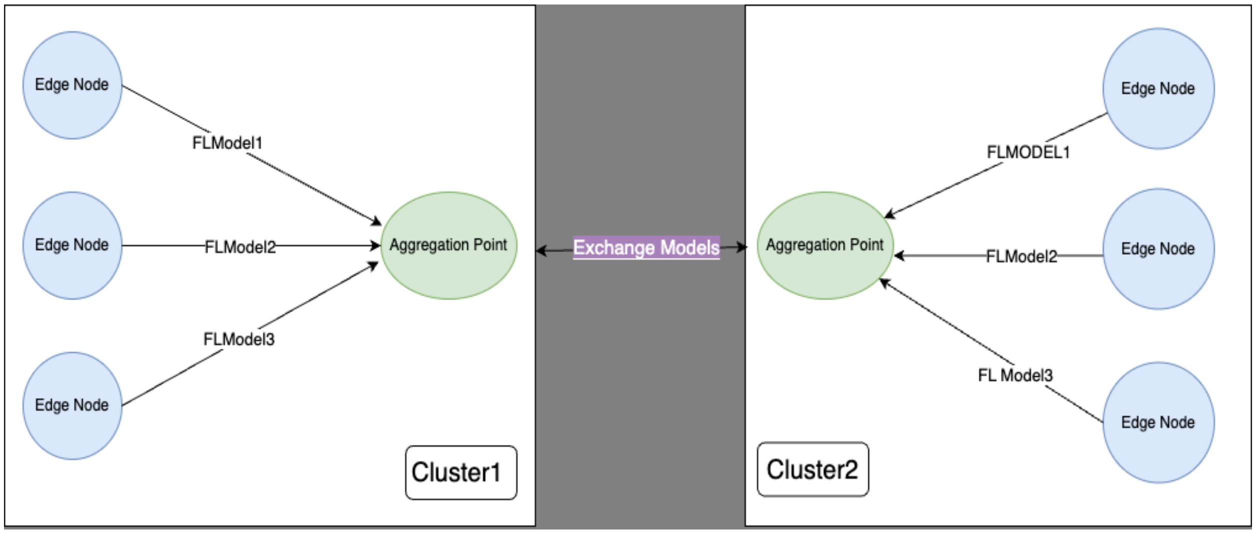


Figure 8: Decentralized Federated Learning

4. Remote Attestation

4.1. TPM-based & TEE-based attestation

Both TPM-based and TEE-based attestation were fully reported in D4.1, in the form of the TrustEdge [29] framework. However, there are plans to continue this line of research in the medium term, aiming to decentralize certain aspects. Most importantly, although certificates are hierarchical by their nature, TrustEdge requires each operation to be validated by the centralized Kubernetes control plane (i.e., the TrustEdge Operator). It should be possible to decentralize the bulk of these operations by forming chains of trust and allowing validated nodes to perform certain peer attestation operations themselves.

4.2. WASM remote attestation

D4.1 detailed how WASM module runtime verification can be achieved through the modification of the WASM runtime. The deliverable also describes the current state-of-the-art for WASM module authentication and how a WASM image remote attestation can be designed using state-of-the-art existing modules.

From D4.1 onwards, the following research directions have been explored, advanced, and remain ongoing toward developing a cloud-native, full-stack remote attestation mechanism for WASM payloads.

4.2.1. NATWORK's full-stack WASM remote attestation

Full-stack WASM remote attestation refers to a dual attestation sequence in which the modified runtime (as produced in D3.5) is first remotely attested before it is used to attest the WASM module itself. The remote attestation of the modified runtime—being a compiled x86 user-space application, possibly placed into a container—can be performed using standard cloud-native remote attestation mechanisms. As stated below, recent developments also consider the containerization of WASM modules. Both directions will be examined to design a scalable, cloud-native, full-stack WASM remote attestation mechanism. The full-stack WASM remote attestation is illustrated in Figure 9.

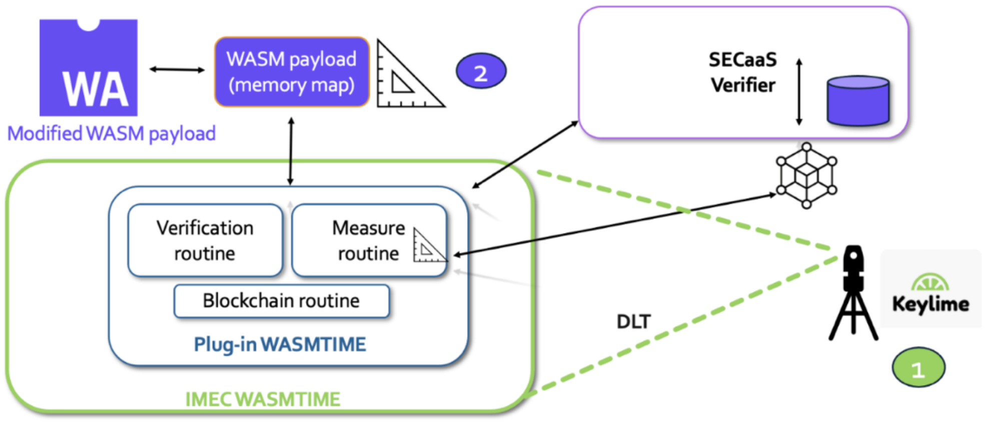


Figure 9: WASM full stack remote attestation

4.2.1.1. Containerization of WASM runtimes

WASM runtimes such as Wasmtime[4], Wasmer,[5] and WasmEdge[6] are commonly deployed inside containers, enabling their integration with container orchestration tools (e.g., Kubernetes[7], Nomad[8]) and deployment tooling, thereby supporting automation aligned with common DevSecOps security policies.

4.2.1.2. Containerization of WASM modules

With the emergence of the WASI[9], new solutions have appeared. Projects such as OCI-compatible WASM images (e.g., shims[10], Krustlet[11]) enable WASM modules to be packaged directly as container images, allowing for the use of container registries, networking, and orchestration ecosystems.

4.2.1.3. Associated challenges and roadmap

Two main challenges arise: scalability and performance. The simplifications offered by containerization and orchestration must not lock the system into a specific runtime nor degrade

[4] WASTIME: https://wasmtime.dev/
[5] Wasmer: https://wasmer.io/
[6] WasmEdge: https://wasmedge.org/
[7] Kubernetes: https://kubernetes.io/
[8] Nomad: https://developer.hashicorp.com/nomad
[9] WASI: https://wasi.dev/
[10] Shims; https://github.com/deislabs/containerd-wasm-shims
[11] Krustlet: https://krustlet.dev/

WASM's near-instant startup times, in accordance with NATWORK's security/performance reconciliation principle.

We want to design a highly scalable solution. A significant obstacle to the adoption of software security solutions stems from the modifications they require in the target software [30]. Unlike software-based confidentiality-preservation techniques, which typically mandate such changes, integrity verification can be designed to avoid them. To achieve this, we place the routines for checking integrity in a separate sidecar, leaving the runtime untouched. By doing this, the workflow is significantly simplified by leveraging container orchestration. We will consider using the sidecar based simplified workflow where the sidecar is automatically appended on our special WASM runtime by the container deployment orchestrator.

Startup times must be short. Our analysis must consider the distinct timelines for the remote attestation of the WASM runtime and the WASM modules. The time to check the runtime is different from the time to check the module. As a matter of fact, the runtime is checked when it is deployed on the platform as system code. The runtime is common to all modules that it will be interpreting. Hence, containerization of the runtime has no negative impact on the module execution. Conversely, containerization of the WASM module introduces an unavoidable overhead. Consequently, our roadmap will prioritize the containerization of the WASM runtime, without disregarding the merits of WASM module containerization. This work will build on our separate ongoing efforts to harden containerized x86 payloads.

It is also worth noting that the 5-6MB expense induced by containerization also fosters the containerization of the runtime (i.e., cost accounted once for all modules) rather than the containerization of the modules (i.e., cost accounted for each module).

Performing integrity verification before execution conflicts with WASM's near-instant cold-start property. To address this, we will consider a novel **attest-after-start** approach that eliminates startup latency. This is enabled directly by our runtime measurement method (as discussed below). The attest-after-start pattern will be applied to WASM modules only, as the attestation of the WASM runtime can be desynchronized and performed in advance.

Finally, we will evaluate the relevance of coupling this work with our platform-agnostic D-MUTRA mutual remote attestation or adopting a simplified client-server layout.

4.3. WASM runtime integrity verification

The runtime integrity directly uses the same method as used for remote attestation. Nuances and specific considerations are detailed below.

4.3.1. Measurement sequence

We will restrict our measurements to the WASM module and consider the measurement of the WASM runtime as already worked out in a different timeline and desynchronized.

4.3.2. Periodic measurements

High-frequency periodic measurements degrade performance because they consume additional CPU resources for integrity verification. We will leverage Linux's cgroups utility to predefine the resource allocation for the runtime sidecar.

The measurement pattern will aggregate several consecutive measurements into a single quote, which will be delivered to the verifier.

4.3.3. On-demand measurements

As an alternative to periodic measurements, on-demand or event-based measurements are performed only when needed. Here too, the application of Linux's cgroups remains beneficial as the measurement routine operates during the WASM module execution.

Co-funded by the European Union 6GSNS Project funded by Schweizerische Eidgenossenschaft Confédération suisse Confederazione Svizzera Confederaziun svizra Swiss Confederation Federal Department of Economic Affairs, Education and Research EAER State Secretariat for Education, Research and Innovation SERI UK Research and Innovation

5. Containerized Runtime Protection and AI-driven Payload Security

In D4.1, a scalable Container-based Security Testbed was presented. Building upon the infrastructure explained there, this version shows expansions made, i.e. additional services that enhance experimentation, monitoring, penetration testing, and attack-defense validation. While an AI-powered monitoring and mitigation tool was introduced and its operation demonstrated in a microservices-based 5G core environment (free5GC/Open5GS), the present extension integrates several complementary containers to improve the current environment's simulation capabilities and expands the data utilized by the AI models.

To address the objective of protecting software at the runtime level, an enhanced containerized experimentation framework was established to serve as a robust baseline for analysing software payload integrity. This infrastructure integrates a granular monitoring stack utilizing Prometheus and Grafana, which extends beyond standard resource tracking to visualize deep kernel-level indicators. By deploying live offensive components, directly alongside the 5G core services, the environment facilitates the controlled simulation of payload manipulation actions and resource exhaustion attacks, such as high-rate UDP flooding. This configuration enables the derivation of precise idle resource baselines, which are experimentally proven to be essential for distinguishing typical operational behaviour from anomalies during the software lifecycle, thereby securing the execution environment against dynamic threats.

Complementing this runtime monitoring, the proposed attack detection mechanism fulfils the requirement for intelligent, automated security optimization and workload suitability analysis. The proposed approach implements a hybrid AI ensemble that unifies statistical methods with deep neural networks using ensemble methods. Experimental validation confirmed that this ensemble strategy achieves high precision and reliability, superior to single-model approaches, effectively securing the software payload by dynamically identifying malicious variations in real-time workloads.

5.1. Testbed updates and setup

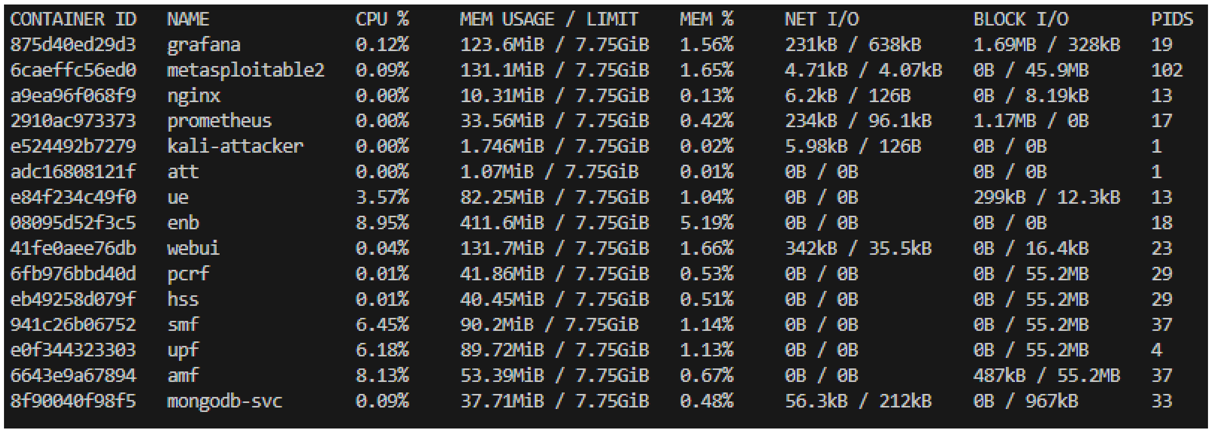
To support extended experimentation, penetration testing, monitoring, and visualization, five new Docker containers were introduced since D4.1:

1. Kali Linux [12] is a Debian-based penetration testing distribution widely used for cybersecurity research. It contains tools for network reconnaissance, exploitation, and security auditing. Within our testbed, it is used within a controlled environment to generate attacks and abnormal traffic.
2. Prometheus [13] is an open-source monitoring and alerting system designed for highly dynamic and containerized environments. In our setup, Prometheus collects resource metrics (e.g., CPU, memory, network I/O) from all active containers, enabling quantitative evaluation of our infrastructure's behavior before, during, and after attacks.
3. Grafana[14] is an analytics and visualization platform that provides dynamic dashboards for time-series data. Linked with Prometheus, it allows real-time visualization of container resource consumption and visually validating attack detection and mitigation.
4. NGINX[15] is a high-performance web server and reverse proxy commonly employed in microservices and cloud architectures. In our deployment, NGINX serves two purposes: (i) as a loadable service that can be monitored under different traffic conditions and (ii) as an additional potential target for simulated attack scenarios.
5. Metasploitable2 [16] is a deliberately vulnerable Linux-based virtual machine used for security training, exploit testing, and penetration-testing research. Integrating Metasploitable2 into the testbed enables controlled exploitation activities and vulnerability-driven experiments that complement the flood-based DoS scenarios that we are planning to implement.

Figure 10 presents the full set of containerized components in CERTH's testbed, including 5G core services (AMF, UPF, etc.), the SDN infrastructure, the AI-powered monitoring stack, and the newly added complementary containers. The figure also displays the idle resource consumption of each container as measured by Docker stats, providing CPU, memory, I/O, and network usage under normal, attack-free conditions. This baseline is crucial for distinguishing typical operational behavior from anomalies detected by the AI-based monitoring system during experiments.

[12] https://www.kali.org
[13] https://prometheus.io
[14] https://grafana.com
[15] https://nginx.org
[16] https://docs.rapid7.com/metasploit/metasploitable-2

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



```
CONTAINER ID   NAME              CPU %    MEM USAGE / LIMIT     MEM %    NET I/O           BLOCK I/O         PIDS
875d40ed29d3   grafana           0.12%    123.6MiB / 7.75GiB    1.56%    231kB / 638kB     1.69MB / 328kB    19
6caeffc56ed0   metasploitable2   0.09%    131.1MiB / 7.75GiB    1.65%    4.71kB / 4.07kB   0B / 45.9MB       102
a9ea96f068f9   nginx             0.00%    10.31MiB / 7.75GiB    0.13%    6.2kB / 126B      0B / 8.19kB       13
2910ac973373   prometheus        0.00%    33.56MiB / 7.75GiB    0.42%    234kB / 96.1kB    1.17MB / 0B       17
e524492b7279   kali-attacker     0.00%    1.746MiB / 7.75GiB    0.02%    5.98kB / 126B     0B / 0B           1
adc16808121f   att               0.00%    1.07MiB / 7.75GiB     0.01%    0B / 0B           0B / 0B           1
e84f234c49f0   ue                3.57%    82.25MiB / 7.75GiB    1.04%    0B / 0B           299kB / 12.3kB    13
08095d52f3c5   enb               8.95%    411.6MiB / 7.75GiB    5.19%    0B / 0B           0B / 0B           18
41fe0aee76db   webui             0.04%    131.7MiB / 7.75GiB    1.66%    342kB / 35.5kB    0B / 16.4kB       23
6fb976bbd40d   pcrf              0.01%    41.86MiB / 7.75GiB    0.53%    0B / 0B           0B / 55.2MB       29
eb49258d079f   hss               0.01%    40.45MiB / 7.75GiB    0.51%    0B / 0B           0B / 55.2MB       29
941c26b06752   smf               6.45%    90.2MiB / 7.75GiB     1.14%    0B / 0B           0B / 55.2MB       37
e0f344323303   upf               6.18%    89.72MiB / 7.75GiB    1.13%    0B / 0B           0B / 55.2MB       4
6643e9a67894   amf               8.13%    53.39MiB / 7.75GiB    0.67%    0B / 0B           487kB / 55.2MB    37
8f90040f98f5   mongodb-svc       0.09%    37.71MiB / 7.75GiB    0.48%    56.3kB / 212kB    0B / 967kB        33
```

Figure 10: Docker resource monitoring during the idle state, showing the stats of all containers

5.1.1. Monitoring Interface

The integration of Prometheus and Grafana provides detailed, real-time visualization of containerized system metrics. The Grafana dashboards display CPU usage (including busy and system load), memory consumption (RAM used and cores), I/O percentage, network traffic, and disk space, along with basic resource indicators for each container. These visualizations offer clear insights into system behavior under normal conditions and illustrate the impact of simulated attacks. An example of the dashboard is shown in Figure 11.

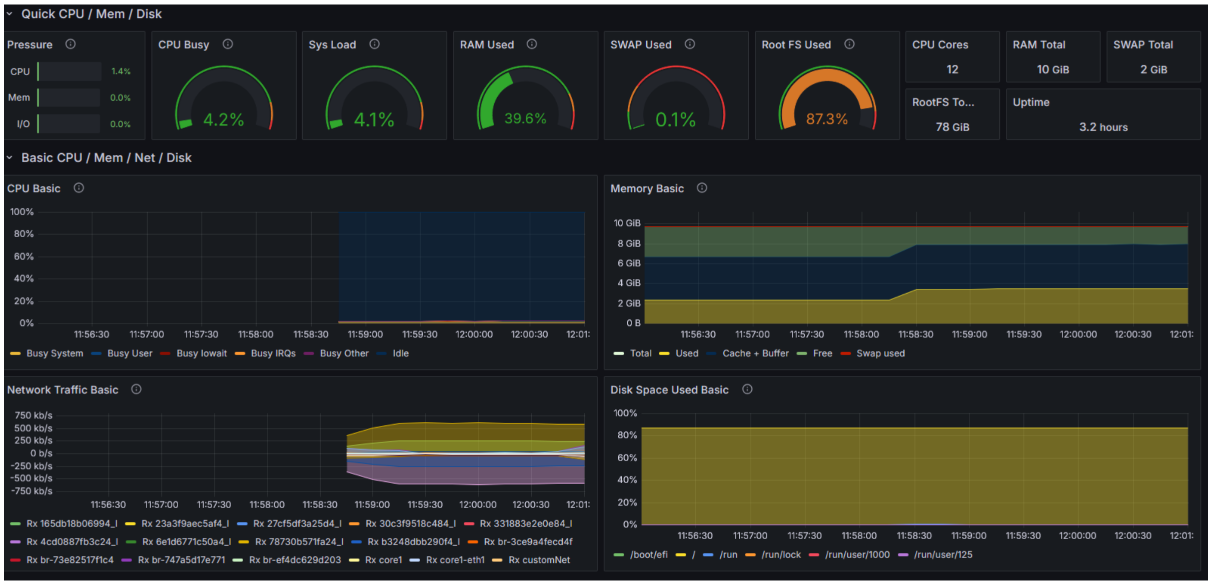


Figure 11: Grafana dashboards showing Prometheus container metrics, including CPU load, memory usage, I/O, network traffic, and disk space

5.2. Proof-of-Concept Attack Experiment

To evaluate the responsiveness and accuracy of the AI-powered monitoring and mitigation system, a proof-of-concept attack experiment was conducted against the AMF component of the 5G core. An attack container was used to generate a high-rate UDP flooding attack targeting the AMF's network interface at IP address 192.187.3.2 and port 5868, which is commonly associated with signaling traffic. Tools such as custom Python scripts or well-known traffic generators, like hping3[17], can be used to generate this type of flood traffic.

During the attack, the system's resource usage changed significantly. As shown in Figure 12, during the attack, the system's resource consumption increased dramatically. The docker stats output shows that the attacker container (att) reached an extremely high CPU load of 277.67%, indicating the intensive packet generation process used to flood the AMF. This sharp rise is accompanied by noticeable spikes in other components directly affected by the traffic: the eNB container increased to 12.70% CPU, while the AMF itself rose to 14.13%, reflecting the stress imposed by the large volume of incoming packets. These deviations clearly contrast with the baseline idle metrics presented in Figure 10.

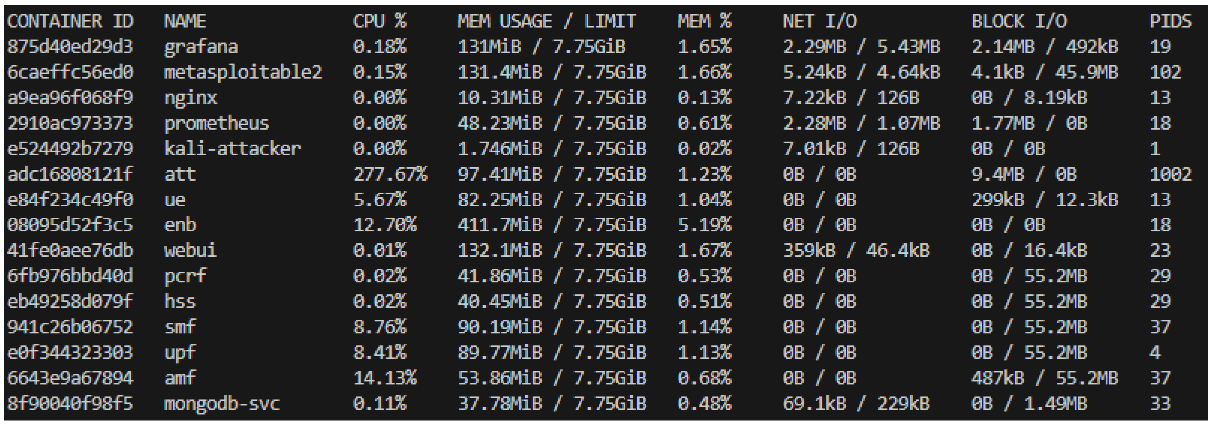
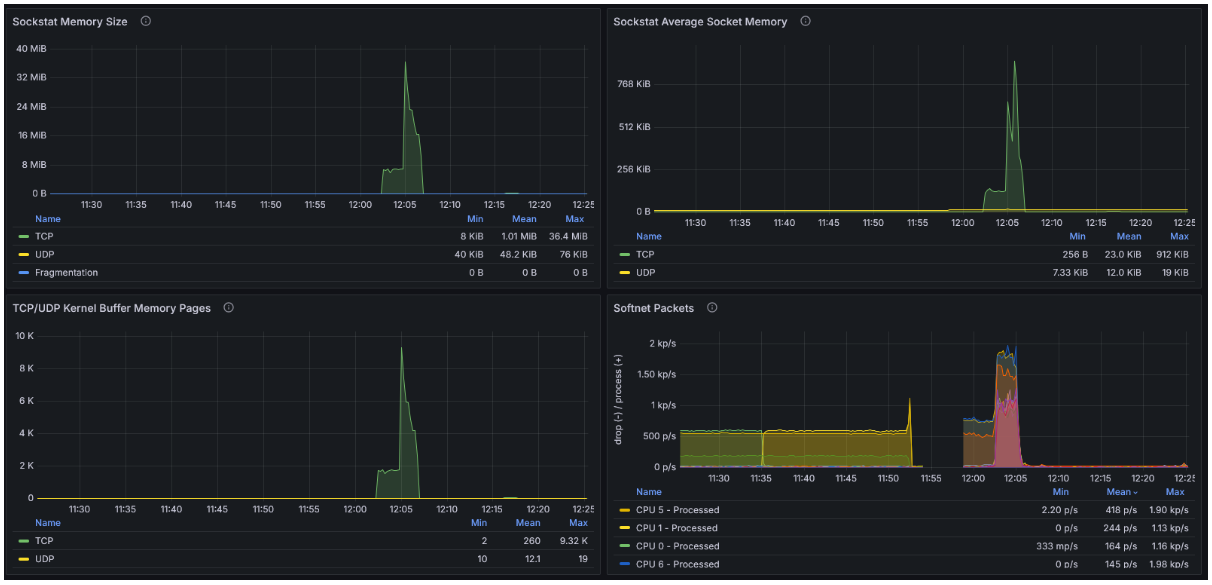

```
CONTAINER ID   NAME              CPU %     MEM USAGE / LIMIT     MEM %    NET I/O           BLOCK I/O        PIDS
875d40ed29d3   grafana           0.18%     131MiB / 7.75GiB      1.65%    2.29MB / 5.43MB   2.14MB / 492kB   19
6caeffc56ed0   metasploitable2   0.15%     131.4MiB / 7.75GiB    1.66%    5.24kB / 4.64kB   4.1kB / 45.9MB   102
a9ea96f068f9   nginx             0.00%     10.31MiB / 7.75GiB    0.13%    7.22kB / 126B     0B / 8.19kB      13
2910ac973373   prometheus        0.00%     48.23MiB / 7.75GiB    0.61%    2.28MB / 1.07MB   1.77MB / 0B      18
e524492b7279   kali-attacker     0.00%     1.746MiB / 7.75GiB    0.02%    7.01kB / 126B     0B / 0B          1
adc16808121f   att               277.67%   97.41MiB / 7.75GiB    1.23%    0B / 0B           9.4MB / 0B       1002
e84f234c49f0   ue                5.67%     82.25MiB / 7.75GiB    1.04%    0B / 0B           299kB / 12.3kB   13
08095d52f3c5   enb               12.70%    411.7MiB / 7.75GiB    5.19%    0B / 0B           0B / 0B          18
41fe0aee76db   webui             0.01%     132.1MiB / 7.75GiB    1.67%    359kB / 46.4kB    0B / 16.4kB      23
6fb976bbd40d   pcrf              0.02%     41.86MiB / 7.75GiB    0.53%    0B / 0B           0B / 55.2MB      29
eb49258d079f   hss               0.02%     40.45MiB / 7.75GiB    0.51%    0B / 0B           0B / 55.2MB      29
941c26b06752   smf               8.76%     90.19MiB / 7.75GiB    1.14%    0B / 0B           0B / 55.2MB      37
e0f344323303   upf               8.41%     89.77MiB / 7.75GiB    1.13%    0B / 0B           0B / 55.2MB      4
6643e9a67894   amf               14.13%    53.86MiB / 7.75GiB    0.68%    0B / 0B           487kB / 55.2MB   37
8f90040f98f5   mongodb-svc       0.11%     37.78MiB / 7.75GiB    0.48%    69.1kB / 229kB    0B / 1.49MB      33
```

Figure 12: Docker resource monitoring during the UDP flooding attack, showing CPU usage spikes for the attacker, the eNB, and the AMF containers

Further evidence of the attack's impact is captured in the Grafana monitoring dashboard. As illustrated in Figure 13,key kernel-level indicators such as sockstat memory size, average socket memory, TCP/UDP kernel buffer usage, and softnet packet counts experienced dramatic spikes during the attack window from approximately 12:03 to 12:07. These patterns reflect the stress placed on the networking subsystem as it attempts to handle abnormal traffic surges.

[17] https://www.kali.org/tools/hping3/

Co-funded by the European Union 6GSNS Project funded by Schweizerische Eidgenossenschaft Confédération suisse Confederazione Svizzera Confederaziun svizra Swiss Confederation Federal Department of Economic Affairs, Education and Research EAER State Secretariat for Education, Research and Innovation SERI UK Research and Innovation



Name	Min	Mean	Max
TCP	8 KiB	1.01 MiB	36.4 MiB
UDP	40 KiB	48.2 KiB	76 KiB
Fragmentation	0 B	0 B	0 B

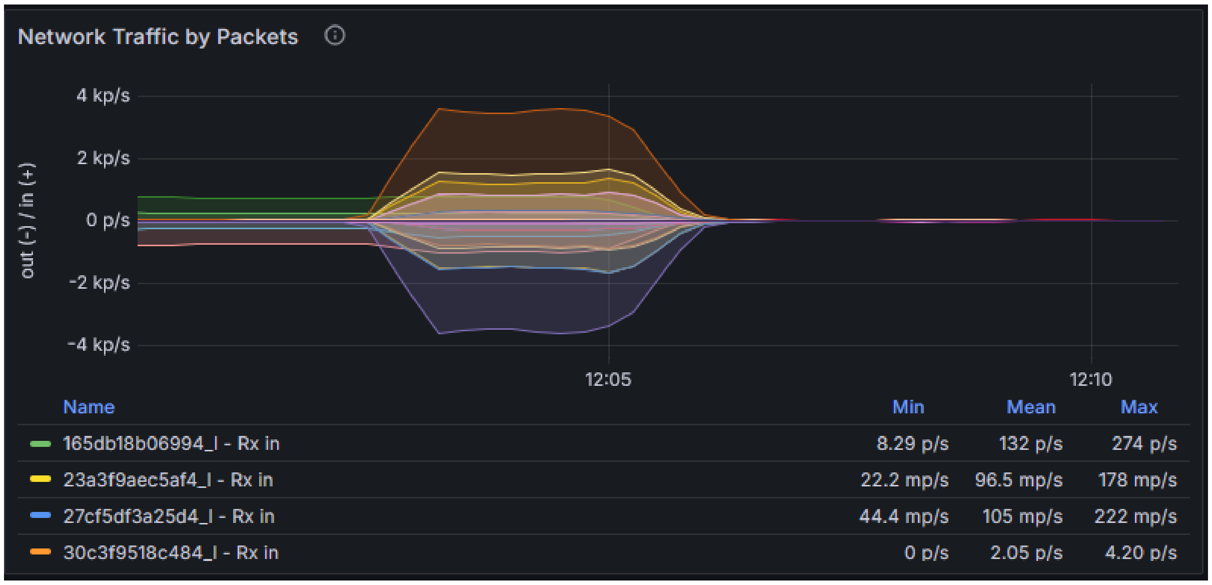
Name	Min	Mean	Max
TCP	256 B	23.0 KiB	912 KiB
UDP	7.33 KiB	12.0 KiB	19 KiB

Name	Min	Mean	Max
TCP	2	260	9.32 K
UDP	10	12.1	19

Name	Min	Mean ⌄	Max
CPU 5 - Processed	2.20 p/s	418 p/s	1.90 kp/s
CPU 1 - Processed	0 p/s	244 p/s	1.13 kp/s
CPU 0 - Processed	333 mp/s	164 p/s	1.16 kp/s
CPU 6 - Processed	0 p/s	145 p/s	1.98 kp/s

Figure 13: Grafana dashboard visualization of kernel-level network metrics

An additional visualization is provided in Figure 14, which depicts network traffic volume in packets as recorded by Grafana. This figure clearly shows the spike corresponding to the onset of the flood attack. Toward the end of the attack window, the graph approaches an almost flat line, indicating that system resources become increasingly unreachable or overloaded, which prevents further packet processing and reflects the AMF's state.



Name	Min	Mean	Max
165db18b06994_l - Rx in	8.29 p/s	132 p/s	274 p/s
23a3f9aec5af4_l - Rx in	22.2 mp/s	96.5 mp/s	178 mp/s
27cf5df3a25d4_l - Rx in	44.4 mp/s	105 mp/s	222 mp/s
30c3f9518c484_l - Rx in	0 p/s	2.05 p/s	4.20 p/s

Figure 14: Grafana network traffic graph illustrating packet volume

5.3. Comparison to State of the Art (SotA) Practices

The proposed architecture improves upon standard 5G security testbed practices by integrating real-time offensive capabilities, detailed kernel-level monitoring, and hybrid AI ensemble detection strategies within a unified containerized environment. Contemporary State of the Art (SoA) testbeds frequently rely on pre-generated datasets or isolated simulations to validate security mechanisms [30]. In contrast, this work expands the containerized 5G security testbed to include live offensive and vulnerable components directly alongside the 5G core. By integrating Kali Linux for dynamic traffic generation and Metasploitable2 for vulnerability research, the environment facilitates the establishment of a precise idle resource baseline. This baseline is critical for distinguishing typical operational behaviour from anomalies in real-time, addressing the limitations of static simulation environments that often fail to capture the stochastic nature of live network attacks [31].

Furthermore, the monitoring capabilities implemented in this framework address the visibility gaps typical of standard container management setups. While conventional monitoring often restricts itself to basic CPU and memory consumption metrics, this system utilizes Prometheus and Grafana to visualize deep kernel-level indicators, including sockstat memory size, TCP/UDP kernel buffer usage, and packet counts. This granular visibility provides crucial insights into the stress placed on the networking subsystem during attacks, validating impacts—such as resource exhaustion at the kernel level—that high-level metrics often fail to detect [32]. This approach ensures that the impact of attacks is accurately captured and correlated with specific attack vectors.

Regarding anomaly detection, this work mitigates the limitations of single model approaches common in current literature. Standard intrusion detection systems often rely exclusively on either statistical heuristic, which are prone to high false-positive rates, or deep learning models, which can lack interpretability [33]. To address these issues, this framework implements a hybrid ensemble strategy [34]. For binary classification, a Hard Voting mechanism combines the statistical Weighted Moving Average (WMA) with AI-based MLP and 1D-CNN models, effectively filtering out false positives by requiring consensus among the distinct detection methodologies. This is complemented by a Soft Voting ensemble for multiclass scenarios, which aggregates probability vectors from Neural Networks and Random Forests. By leveraging the global pattern recognition of neural networks alongside the strict decision boundaries of tree-based models, this method ensures high accuracy across diverse attack vectors, such as AMF and eNB flooding.

5.4. AI detection analysis

The analysis was conducted on PCAP files and resource consumption logs, capturing both normal and abnormal 5G core network traffic. The abnormal traffic included two specific attack scenarios: a UDP flooding attack targeting the AMF and a PFCP flooding attack against the UPF.

5.4.1. Binary classification results

To evaluate anomaly detection capabilities, three complementary strategies were employed: statistical detection using a Weighted Moving Average (WMA) and AI-based detection using neural networks, specifically a Multilayer Perceptron (MLP) and a 1D Convolutional Neural Network (1D-CNN). Each model independently produced predictions for every traffic window.

We propose an ensemble by combining AI models with the WMA to leverage their complementary strengths: reducing false positives and improving robustness. To utilize the WMA, we normalize packet size from an integer in KB to a range of (-1,1) and calculate a baseline mean packet size of -0.33 for normal traffic. The following packet size range was used for the algorithm: [-0.42, -0.24]. Table 1, Table 2 and Table 3, summarize the classification performance of each method and the ensemble approach.

Table 1: MLP classification detailed results

Class	Precision	Recall	F1-Score	Support
Normal	0.96	1.00	0.98	85639
Abnormal	1.00	0.89	0.94	34592
Accuracy	-	-	0.97	120231

Table 2: CNN classification detailed results

Class	Precision	Recall	F1-Score	Support
Normal	0.96	1.00	0.98	85639
Abnormal	1.00	0.89	0.94	34592
Accuracy	-	-	0.97	120231

Table 3: WMA classification detailed results

Class	Precision	Recall	F1-Score	Support
Normal	0.93	0.98	0.95	85639

Class	Precision	Recall	F1-Score	Support
Abnormal	0.93	0.83	0.88	34592
Accuracy	-	-	0.93	120231
Macro Avg	0.93	0.90	0.91	120231
Weighted Avg	0.93	0.93	0.93	120231

A Hard Voting mechanism is used for the ensemble. This is generally more robust for combining neural networks with statistical heuristics. It operates in three different stages:

1. Collection Stage: We collect the binary predictions (0 for Normal, 1 for Attack) for every single test sample from all three models: P_{MLP}, P_{CNN}, P_{WMA}
2. Voting Stage: For every traffic window, we sum the votes:
    - $Sum_{votes} = P_{MLP} + P_{CNN} + P_{WMA}$
3. Decision stage:
    - If $Sum_{votes} \geq 2$: The ensemble predicts Attack (1).
    - Otherwise: It predicts Normal (0).

This helps filter out false positives. For example, if the statistical WMA model falsely flags a packet as an attack because it was slightly large, but both Neural Networks (which understand the context/sequence) predict it as normal, the ensemble will correctly classify it as Normal.

The results demonstrate that the AI-based models (MLP and 1D-CNN) consistently achieve high precision, recall, and F1-scores in detecting abnormal traffic in the examined scenario, surpassing the statistical WMA baseline, which is more sensitive to small deviations but prone to false positives. Integrating all three approaches through a Hard Voting Ensemble enhances the detection reliability by filtering out isolated misclassifications and combining the strengths of both neural networks and statistical heuristics. This hybrid ensemble approach maintains high accuracy across both normal and attack scenarios, including UDP flooding on the AMF and PFCP flooding on the UPF. Overall, these findings underscore the effectiveness of combining statistical and AI-based methods for robust anomaly detection in complex 5G core network environments. The proposed approach will be further validated with more complex attacks.

5.4.2. Multiclass classification results

Initially, each model outputs a probability distribution summing to 1.0 for the three classes (normal traffic, AMF flooding, eNB flooding). For the Neural Networks (MLP & CNN), the raw output of the network is a logit (a raw score). The Softmax function is applied to convert these scores into probabilities. For the Random Forest, tree-based models calculate the percentage of trees in the forest that voted for a specific class. After the probabilities are calculated for every single test sample, the probability vectors from all three models are utilized to calculate the mean.

$$P_{ens} = \frac{P_{MLP} + P_{CNN} + P_{RF}}{3}$$

The final class is selected by finding the index of the class with the highest average probability.

The proposed ensemble combines Neural Networks, which search for global patterns and complex feature interactions, with a Random Forest, which looks for strict decision boundaries based on feature thresholds. This approach is applicable since we have more than two classes and allows covering cases where a model might be correct but hesitant, or incorrect but with low confidence. The detailed classification results for the multiclass scenario are presented in Table 4 to Table 8.

Table 4: MLP classification detailed results

Class	Precision	Recall	F1-Score	Support
Normal	0.9567	1.0000	0.9778	47577
AMF	0.9954	0.7916	0.8819	10101
eNB	0.9557	0.9468	0.9512	9117
Accuracy (total)	-	-	0.9612	66795

Table 5: CNN classification detailed results

Class	Precision	Recall	F1-Score	Support
Normal	0.9576	0.9999	0.9783	47577
AMF	0.9914	0.8061	0.8892	10101
eNB	0.9678	0.9455	0.9565	9117
Accuracy (total)	-	-	0.9631	66795

Table 6: Random Forest classification detailed results

Class	Precision	Recall	F1-Score	Support
Normal	0.9582	0.9999	0.9786	47577
AMF	0.9968	0.8023	0.8890	10101
eNB	0.9612	0.9509	0.9560	9117
Accuracy (total)	-	-	0.9633	66795

Table 7: Ensemble (Soft Voting) classification detailed results

Class	Precision	Recall	F1-Score	Support
Normal	0.9576	1.0000	0.9784	47577
AMF	0.9978	0.7993	0.8876	10101
eNB	0.9605	0.9504	0.9555	9117
Accuracy (total)	-	-	0.9629	66795

Table 8: Ensemble Confusion Matrix classification detailed results

	Pred: Normal	Pred: AMF	Pred: eNB
True: Normal	47577	0	0
True: AMF	1671	8074	356
True: eNB	434	18	8665

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

6. Conclusions

The deliverable D4.1 "Payload security per runtime, intelligent runtime selection and attestation.r1" presented a comprehensive overview of the NATWORK aspects enabling secure, flexible-runtime workloads. Deliverable D4.2 "Payload security per runtime, intelligent runtime selection and attestation.r2" reported on the incremental improvements in each of these aspects over the next reporting period from M15 (D4.1 submission) to M24.

While containerized software and virtual machines have been employed for years in software orchestration, many security aspects of containers and security optimization through orchestration have been neglected in research and frameworks. The combination of D4.1 and this document highlights these neglected factors, summarizing the work performed in each of them.

From the security perspective, relevant runtimes are presented in D4.1, including VMs, microVMs, unikernels, containers, and WebAssembly. Each of these is examined in detail, providing an overview of essential security advantages and disadvantages compared to native processes and/or containers. Additional considerations are presented in this document as required for ongoing work in attestation frameworks (TrustEdge and D-MUTRA), as well as ML-based security improvements for VMs and Docker containers, and the use of non-container runtimes in orchestration (e.g., WASM).

Network and runtime API unification provides a uniform way of selecting and deploying workloads independently of the runtime executing it. This paves the way for intelligent runtime selection, consisting of intent-based metadata gathering and deployment, which can be combined with an in-progress ML-based approach for workload modeling for optimal orchestration. D4.1 provided a foundation for this unification, and this document has added additional runtimes, as well as components and frameworks for GL and FL that leverage the unification framework. These efforts feed into the "optimizing selection" part of T3.1 "Secure-by-design federated slice orchestration and management", along with UC1, for sustainable and reliable 6G services.

The progress concerning remote attestation was presented based on TPM and TEE, which are capable of securing workloads at runtime and may be integrated as a feature within intelligent runtime/node selection at deployment time, aiding with the reliability aspect of UC1.

In the context of containerized environments, the reported work validated that combining granular kernel-level monitoring with a hybrid AI voting ensemble significantly enhances the detection of payload-targeted attacks. This establishes a critical predictive security baseline, demonstrating that intelligent, multi-model detection strategies can effectively secure runtime execution against dynamic resource exhaustion and manipulation. These aspects will be demonstrated in UC4.

References

[1] R. Behravesh, E. Coronado and R. Riggio, "Performance Evaluation on Virtualization Technologies for NFV Deployment in 5G Networks," 2019 IEEE Conference on Network Softwarization (NetSoft), Paris, France, 2019, pp. 24-29, doi: 10.1109/NETSOFT.2019.8806664.

[2] Plauth, M., Feinbube, L., Polze, A. (2017). A Performance Survey of Lightweight Virtualization Techniques. In: De Paoli, F., Schulte, S., Broch Johnsen, E. (eds) Service-Oriented and Cloud Computing. ESOCC 2017. Lecture Notes in Computer Science(), vol 10465. Springer, Cham. https://doi.org/10.1007/978-3-319-67262-5_3

[3] Van Rijn, V., & Rellermeyer, J. S. (2021, December). A fresh look at the architecture and performance of contemporary isolation platforms. In Proceedings of the 22nd International Middleware Conference (pp. 323-335).

[4] T. Goethals, M. De Clercq, M. Sebrechts, F. De Turck, and B. Volckaert, "Feather: Lightweight Container Alternatives for Deploying Workloads in the Edge," in CLOSER, 2024, pp. 27–37.

[5] eddine Khelifa, S., Bagaa, M., Messaoud, A. O., & Ksentini, A. (2024, February). Case study of WebAssembly Runtimes for AI Applications on the Edge. In 2024 Global Information Infrastructure and Networking Symposium (GIIS) (pp. 1-6). IEEE.

[6] T. Goethals, M. Sebrechts, M. Al-Naday, F. D. Turck and B. Volckaert, "Flocky: Decentralized Intent-based Edge Orchestration using Open Application Model," in IEEE Transactions on Services Computing, doi: 10.1109/TSC.2025.3631387.

[7] L. Yuan, Z. Wang, L. Sun, P. S. Yu, and C. G. Brinton, "Decentralized Federated Learning: A Survey and Perspective," IEEE Internet of Things Journal, vol. 11, no. 21, pp. 34617–34638, 2024, doi: 10.1109/JIOT.2024.3407584.

[8] H. Fereidooni et al., "SAFELearn: Secure Aggregation for private FEderated Learning," in 2021 IEEE Security and Privacy Workshops (SPW), 2021, pp. 56–62. doi: 10.1109/SPW53761.2021.00017.

[9] S. Shukla and N. Srivastava, "Federated matched averaging with information-gain based parameter sampling," in Proceedings of the First International Conference on AI-ML Systems, in AIMLSystems '21. Bangalore, India: Association for Computing Machinery, 2021. doi: 10.1145/3486001.3486225.

[10] S. Wang, D. Li, J. Geng, Y. Gu, and Y. Cheng, "Impact of network topology on the performance of DML: Theoretical analysis and practical factors," in IEEE INFOCOM 2019-IEEE conference on computer communications, IEEE, 2019, pp. 1729–1737.

[11] H. Zhu, J. Xu, S. Liu, and Y. Jin, "Federated learning on non-IID data: A survey," Neurocomputing, vol. 465, pp. 371–390, 2021.

Co-funded by the European Union 6GSNS Project funded by Schweizerische Eidgenossenschaft Confédération suisse Confederazione Svizzera Confederaziun svizra Swiss Confederation Federal Department of Economic Affairs, Education and Research EAER State Secretariat for Education, Research and Innovation SERI UK Research and Innovation

[12] V. P. Chellapandi, A. Upadhyay, A. Hashemi, and S. H. Żak, "Decentralized Federated Learning: Model Update Tracking Under Imperfect Information Sharing," in 2024 IEEE International Conference on Big Data (BigData), 2024, pp. 7697–7706. doi: 10.1109/BigData62323.2024.10825274.

[13] M. Fang et al., "Byzantine-Robust Decentralized Federated Learning," in Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, in CCS '24. Salt Lake City, UT, USA: Association for Computing Machinery, 2024, pp. 2874–2888. doi: 10.1145/3658644.3670307.

[14] Y. Deng, M. M. Kamani, and M. Mahdavi, "Distributionally robust federated averaging," in Proceedings of the 34th International Conference on Neural Information Processing Systems, in NIPS '20. Vancouver, BC, Canada: Curran Associates Inc., 2020.

[15] Y. Tian, Z. Al-Ars, M. Kitsak, and P. Hofstee, "Vanishing Variance Problem in Fully Decentralized Neural-Network Systems." 2024. [Online]. Available: https://arxiv.org/abs/2404.04616

[16] P. Xiao, S. Cheng, V. Stankovic, and D. Vukobratovic, "Averaging Is Probably Not the Optimum Way of Aggregating Parameters in Federated Learning," Entropy, vol. 22, no. 3, 2020, doi: 10.3390/e22030314.

[17] L. Giaretta and S. Girdzijauskas, "Gossip Learning: Off the Beaten Path," in 2019 IEEE International Conference on Big Data (Big Data), 2019, pp. 1117–1124. doi: 10.1109/BigData47090.2019.9006216.

[18] Das, I. Gupta, and A. Motivala, "SWIM: scalable weakly-consistent infection-style process group membership protocol," in Proceedings International Conference on Dependable Systems and Networks, 2002, pp. 303–312. doi: 10.1109/DSN.2002.1028914.

[19] Srivastava, T. J. Maranzatto, and S. Ulukus, "Age of Gossip with the Push-Pull Protocol," in ICASSP 2025 - 2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2025, pp. 1–5. doi: 10.1109/ICASSP49660.2025.10890446.

[20] Hegedűs, G. Danner, and M. Jelasity, "Decentralized learning works: An empirical comparison of gossip learning and federated learning," Journal of Parallel and Distributed Computing, vol. 148, pp. 109–124, 2021, doi: https://doi.org/10.1016/j.jpdc.2020.10.006.

[21] H. Zhang, J. Bosch, and H. H. Olsson, "Enabling efficient and low-effort decentralized federated learning with the EdgeFL framework," Information and Software Technology, vol. 178, p. 107600, 2025, doi: https://doi.org/10.1016/j.infsof.2024.107600.

[22] Y. Zhou, M. Shi, Y. Tian, Q. Ye, and J. Lv, "DeFTA: A plug-and-play peer-to-peer decentralized federated learning framework," Information Sciences, vol. 670, p. 120582, 2024, doi: https://doi.org/10.1016/j.ins.2024.120582.

[23] E. T. M. Beltrán et al., "Fedstellar: A Platform for Decentralized Federated Learning," Expert Systems with Applications, vol. 242, p. 122861, 2024, doi: https://doi.org/10.1016/j.eswa.2023.122861.

Co-funded by the European Union
Project funded by
UK Research and Innovation

[24] Aldhyani, T. H., & Alkahtani, H. (2022). Artificial intelligence algorithm-based economic denial of sustainability attack detection systems: Cloud computing environments. Sensors, 22(13), 4685.

[25] Turnbull, J. (2018). Monitoring with Prometheus. Turnbull Press.

[26] Ni, Y., Denolle, M. A., Fatland, R., Alterman, N., Lipovsky, B. P., & Knuth, F. (2024). An object storage for distributed acoustic sensing. Seismological Research Letters, 95(1), 499-511.

[27] Tirmazi, M., Barker, A., Deng, N., Haque, M. E., Qin, Z. G., Hand, S., ... & Wilkes, J. (2020, April). Borg: the next generation. In Proceedings of the fifteenth European conference on computer systems (pp. 1-14).

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

[29] Thijsman, J., Sebrechts, M., De Turck, F., & Volckaert, B. (2024, July). Trusting the Cloud-Native Edge: Remotely Attested Kubernetes Workers. In 2024 33rd International Conference on Computer Communications and Networks (ICCCN) (pp. 1-6). IEEE.

[30] Ali, M. (2016). *User resistance in IT: A literature review.* (International Journal of Information Management / related venue)

[31] M. A. Hoque, M. H. Bhuyan, R. C. Baishya, D. K. Bhattacharyya, and J. K. Kalita, "A survey on distributed denial-of-service attack mitigation for 5G and beyond," IEEE Open Journal of the Communications Society, vol. 6, pp. 5841–5867, 2025.

[32] P. Scalise, M. Boeding, M. Hempel, H. R. Sharif, J. Delloiacovo, and J. Reed, "A systematic survey on 5G and 6G security considerations, challenges, trends, and research areas," Future Internet, vol. 16, no. 3, p. 67, Mar. 2024.

[33] H. Aktolga and O. Alay, "AI-driven container security approaches for 5G and beyond: A survey," ITU Journal on Future and Evolving Technologies, vol. 4, no. 2, pp. 363–384, Aug. 2023.

[34] M. A. Ganaie, M. Hu, A. K. Malik, M. Tanveer, and P. N. Suganthan, "Ensemble deep learning: A review," Engineering Applications of Artificial Intelligence, vol. 115, p. 105151, Oct. 2022.

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