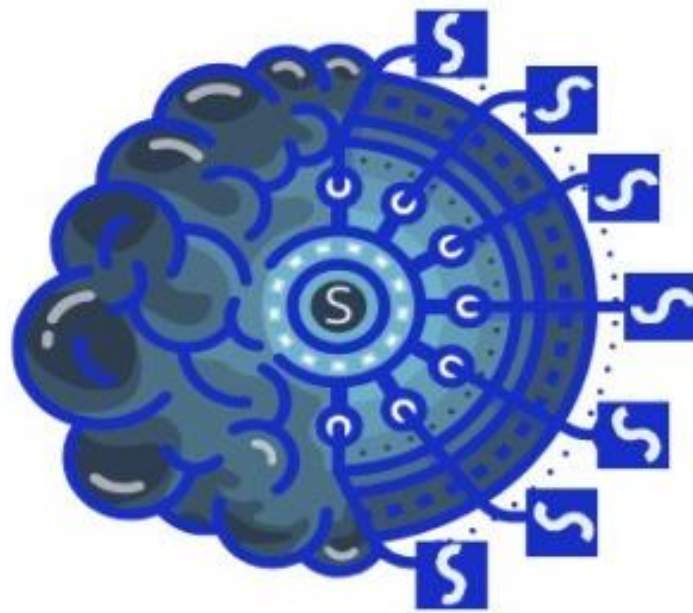


**SCALABLE AND QUANTUM RESILIENT
HETEROGENEOUS EDGE COMPUTING
ENABLING TRUSTWORTHY AI**



SMARTY



This project is supported by the European Union's HORIZON-JU-IA under grant agreement No. 101140087

Deliverable D4.1: Design and preliminary implementation of secure networking solutions

Editor	F. Cugini (CNIT)
Contributors	CNIT, UPM, SSSA, CMPG, ISRD, TKI, SCO, IQU, TUM, NVIL, MLNX, ITL, ISRD, COGN, TUB
Version	1.0
Date	February 11 th , 2025
Distribution	Public

DISCLAIMER

This document contains proprietary information belonging to the SMARTY consortium members, governed by the rights, obligations, and terms outlined in Grant Agreement number 101140087. The SMARTY consortium's activities are funded by the European Commission.

This document and the information within it may not be used, copied, duplicated, reproduced, modified, or shared in any form with third parties, in whole or in part, without the prior written consent of the SMARTY consortium members. If such consent is granted, appropriate acknowledgment of the document's authors and all relevant copyright notices must be clearly referenced. In the event of any infringement, the consortium members reserve the right to pursue appropriate legal action.

The content of this document represents solely the views of the authors and does not necessarily reflect the views of the European Commission. Neither the SMARTY consortium as a whole nor any individual member guarantees the suitability, accuracy, or risk-free nature of the information provided. No liability is accepted for any loss or damage incurred as a result of using this information.

The information is provided "as is," without guarantees or warranties of fitness for any particular purpose. Users assume full responsibility and liability for any use of the information.

REVISION HISTORY

Revision	Date	Responsible	Comment
0.1	Oct 17 th , 2024	K. Vlachos	Template creation
0.2	Feb 19 th , 2025	F. Cugini	Partner contributions
0.3	Apr 20, 2025	K. Vlachos	Quality check
1.0	May 20, 2025	F. Cugini	Final

LIST OF AUTHORS

<i>Partner</i>	<i>Name Surname</i>
CNIT	F. Cugini, A. Ibrahim, F. Paolucci
UPM	P. Reviriego, J. Conde, J. Salvachua
SSSA	L. Valcarenghi, A. Yeganehfallah, E. Paolini
CMPG	K. Rustagi
ISRD	A. Flizikowski, M. M. Mowla
TKI	I. Pelle, E. R. Szabo
SCO	C. Fetzer
IQU	J. Vardakas
TUM	S. Reimers
NVIL	J. J. Vegas Olmos
MLNX	A. Pakouline-Navarro
ITL	F. Bianchi, A. Albanese
ISRD	A. Flizikowski
COGN	K. Obregon
TUB	D. Le Phouc

EXECUTIVE SUMMARY

This deliverable reports on developing software solutions that increment the securitization level of infrastructure used to run AI solutions. In particular, it investigates confidential computing solutions, software defined perimeters designs and architectures, deep data plane security, including decentralized features extraction and defensive wire speed AI, declarative cooperation in cloud computing environments, and dynamic swam formation. This first release includes both studies and implementation of the proposed solutions. All tools and solutions can be categorized as follows:

- Solutions to ensure secure networking
- Tools and algorithms to protect from cyberattacks and malicious attempts to falsify data
- Schemes to secure workflows across multiple stakeholders in both SW and HW level
- Protection of dynamically formatting network swarms.

TABLE OF CONTENTS

Table of Contents	6
Table of Figures	8
Table of Abbreviations and Acronyms.....	10
1. Introduction.....	11
2. Software Defined Perimeters in Secured Networking.....	14
2.1 Software defined perimeters	14
2.2 Capability-based access control.....	15
2.3 Post-Quantum Secure Optical IPsec Tunnel for Confidential AI Training.....	16
2.4 Transparently securing communications	19
3. Deep Data Plane Solutions: Decentralized Feature Extraction (DFE) and Defensive Wirespeed AI	21
3.1 Real-Time Graph Neural Network for Malicious Traffic Detection.....	21
3.2 Effectiveness of Confidentiality-Preserving Clustering Algorithms for Soft Failure Detection in Optical Networks.....	28
3.3 Defensive wirespeed AI	37
3.4 Integrated Data Sketches for Traffic Monitoring and Threat Detection.....	39
3.5 Telco Cyber-Threat Mitigation through WAI.....	41
3.6 Interfaces for the virtualized RAN and WAI.....	42
3.6.1 Options for WAI in 5G/6G.....	43
4. Declarative cooperation in cloud computing environments	47
4.1 Confidential workflows.....	47
4.1.1 Example: AI Workflow	47
4.1.2 Technical Problem Description.....	48
4.1.3 Example: A Simple Declarative Workflow.....	49
4.2 Secure accelerator interface	51
4.3 advanced testing and monitoring tools.....	51
5. Dynamic Network Swarm	56
5.1 Hierarchical-Hybrid Synchronization of Swarn Networks.....	56
5.1.1 Proposed Approach and Implementation	57
5.1.2 Results	57
5.2 Semantic-Aware Serverless Deployment with P4 Load Balancing.....	58
5.2.1 Programmable network-based components.....	59
5.2.2 Components related to the serverless engine.....	61
5.3 Dynamic 5G/B5G network SWARM.....	66

5.4	Network Swarm Coordinators at Edge Gateways Equipped with DPUs	68
5.5	Device-flow-Graph-based policy language	69
5.6	Derived Information Framework for Improving Decision making process	70
6.	Conclusions.....	72
7.	Bibliography.....	73

TABLE OF FIGURES

Figure 1-1: Bleck level design of SCONE Network Shield	11
Figure 1-2: Multi-stakeholder access to registered devices through SCONE network shield	12
Figure 1-3: Dynamic Network Swarm concept with its main components and features	13
Figure 2-1: Perimeter Establishment workflow	15
Figure 2-2: W/SW co-designed OS serving as the mediator between applications the hardware resources.....	16
Figure 2-3: Experimental Setup: (a) A mobile client communicates with the cloud using a PQC-encrypted north-south IPsec tunnel at 0.486 Gbit/s for confidential AI training. (b) The servers in the data center for confidential AI training use an PQC-based east-west IPsec tunnel, offloading AES-256 to the DPUs achieving (c), encryption at 100 Gbit/s traffic.	17
Figure 2-4: Average latency results in CPU clock cycles introduced by the execution of PQC cryptography. The first row shows the latency the client machine is penalized with. The second row represents the tax in CPU cycles introduced by PQC operations when performed on the server machine.....	19
Figure 2-5: SCONE Network Shield between an application (service) and the operating system.	20
Figure 3-1: Constructing the traffic graph	22
Figure 3-2: F1 Score vs Maximum Number of Edges	24
Figure 3-3: Prediction Time vs Maximum Number of Edges	24
Figure 3-4: Prediction Time vs F1 Score	24
Figure 3-5: F1-Prediction Time Analysis with respect to Granularity	25
Figure 3-6: Attack Analysis	26
Figure 3-7: FTG-Net vs Our Model (F1 Score)	27
Figure 3-8: FTG-Net vs Our Model (Prediction Time)	27
Figure 3-9: Proposed approach.....	29
Figure 3-15: Serial implementation of flow identification and frequency estimation	40
Figure 3-16: Proposed integrated implementation of flow identification and frequency estimation	40
Figure 3-17: Results for traffic with Pareto distribution: (left) Number of memory access in relation the number of packets, (middle) FPP is displayed based on the number of negative elements tested and (right) Average Absolute Error and Average Relative Error.	41
Figure 3-18 High level view on the points of WAI interconnection in 5G/6G	45
Figure 4-1: Multi-stakeholder computations as workflows.....	48
Figure 4-2: Provision of governance for the declarations of workflows to address potential Time-Of-Check-To-Time-Of-Use (TOCTTOU) attacks.	49
Figure 4-3: A Simple Declarative Workflow for two native containers.	49
Figure 4-4: API responses in Integration testing.....	52
Figure 4-5: Latency over time in performance analysis.....	53
Figure 4-6: Data volume for each processing flow.	54
Figure 4-7: Number and type of vulnerabilities detected.	55
Figure 5-1: synchronization model built on a combination of hierarchical and cluster-based architectures and implemented using ONOS SDN controllers	57
Figure 5-2: Automatic recovery from instance failures	58
Figure 5-3: Lab environment used for testing the serverless extensions.....	65
Figure 5-4: Sample application used for testing the serverless extensions.	65

Figure 5-5: Sample visualization of the function execution and invocation latencies in the serverless test scenario.....	66
Figure 5-6: Conceptual view of Swarmification of 5G RAN	67
Figure 5-7: Distributed SDN Controllers for Network Swarm Coordinators at Edge Gateways..	68
Figure 5-8: Conceptual view of Derived Information Framework to extract/apply insights for improved decision-making.....	70
Table 0-1: Abbreviations and Acronyms	11
Table 3-1: Summary of Experimental Configuration	27

TABLE OF ABBREVIATIONS AND ACRONYMS

Table 0-1: Abbreviations and Acronyms

ADAS	Advanced Driver-Assistance System
AGV	Automated Guided Vehicles
AI	Artificial Intelligence
AIA	AI Act
AIoT	Artificial Intelligence of Things
AIU	Attested Interface Unit
ALTAI	Assessment List for Trustworthy AI
APM	Application Performance Monitoring
AR	Augmented Reality
CPU	Central Processing Unit
CRD	Custom Resource Definition
DFG	Device Flow Graph
DGA	Data Governance Act
DPU	Data Processing Unit
EGTAI	Ethics Guidelines for Trustworthy AI
ERM	Edge Resource Monitoring
EU	European Union
FaaS	Function as a Service
FRIA	Fundamental Rights Impact Assessment
GDPR	General Data Protection Regulation
GNN	Graph Neural Network
GPU	Graphics Processing Unit
gRPC	gRPC Remote Procedure Calls
HIC	Human-in-command
HITL	Human-in-the-loop
HLEG	High-Level Expert Group
HOTL	Human-on-the-loop
HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
KPI	Key Performance Indicators
KPIs	Key Performance Indicators
MDR	Medical Device Regulation
ML	Machine Learning
NGO	Non-Governmental Organization
OECD	Organization for Economic Co-operation and Development
OT	OpenTelemetry
P4	Programming Protocol-independent Packet Processors
P4LB	P4 Load Balancer
SDN	Software-Defined Networking
TCB	Trusted Computing Base
XAI	Explainable AI

1. INTRODUCTION

This deliverable reports on the design and preliminary implementation of the (i) security solutions for trustworthy edge computing and networking, (ii) software defined perimeters and orchestration through AI-techniques; (iii) metadata representation and smart allocation solutions, and (iv) monitoring and telemetry infrastructure. In addition, it provides the indicators that show how much progress has been achieved regarding the completion of the activities, the limitations, as well as the potential changes in the plans and the corrective actions taken (if needed)) to compensate for the deviations. It is organized in 5 Sections.

Section 2 summarizes the progress of Task 4.1, on defining SW/HW tools and architectures to ensure secure networking of all data in use, in transit, and at rest. To this end, the architecture and the establishment workflow of a Software Defined Perimeter -SDP- has been proposed and detailed. In addition, access control is proposed that is based on i) a hardware-software co-design that introduces a trusted bus system for secure policy-compliant hardware acceleration and (ii) a network shield that operates as an intermediary layer between an application (service) and the operating system to ensure transparent encryption for network communications. In this way, TCP connections are automatically encrypted, and only direct communication partners can read the network traffic in clear text. Next Figure shows the *SCONE Network Shield* under development. It operates as an intermediary layer between an application (service) and the operating system. The shield transparently encrypt connections so that a) users can only access services and data when running on sufficiently protected devices and b) only direct communication partners can read the network traffic.

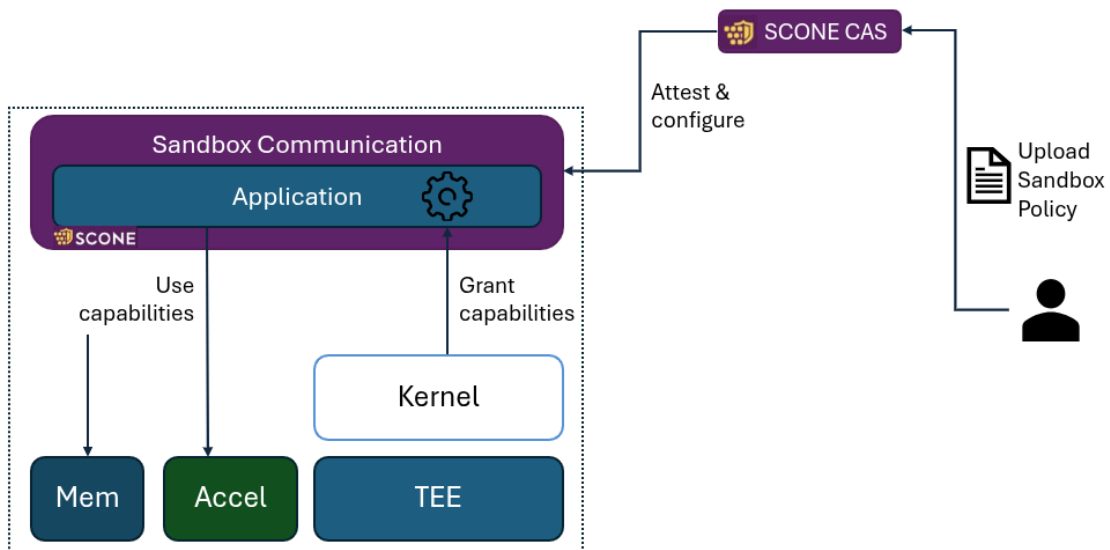


Figure 1-1: Block level design of SCONE Network Shield

Section 3 refers to tools and algorithms designed and developed to protect from cyberattacks and malicious attempts to falsify data (Task 4.2). The proposed methods must operate at wirespeeds bearing in mind the huge number of data exchanges and the increased bandwidth of networks. These tools include a) novel real-time Graph Neural Networks –GNN– algorithms for malicious traffic detection, b) fault management data protection to secure networks running automated network management systems, c) computer vision techniques for wirespeed AI enabled network intrusion detection, d) probabilistic data structures for wirespeed traffic monitoring against malicious actions.

With respect to security attacks in telco network and their mitigation, Section 3.5 and 3.6 details how and in which critical network segments/domain, WAI methodologies can be embedded to protect network infrastructure. It is important to recognize that WAI solutions can in 5G (or 6G) be implemented in RAN directly or at the level of compute infrastructure (bare-metal or virtual) that is hosting it.

Section 4 investigates a framework (Task 4.3) to set up confidential workflows across multiple stakeholders. Next Figure 1-2 displays such a scenario with stakeholders accessing application services through SCONE shield. Multi-stakeholder computations are considered different workflows, and each component of a workflow belongs to one stakeholder. Each component is a confidential service, i.e., a service running in an enclave, or an encrypted volume, i.e., a set of files that can only be read by authorized, confidential services. SCONE enables the connection of confidential services and the encrypted volumes into one workflow.

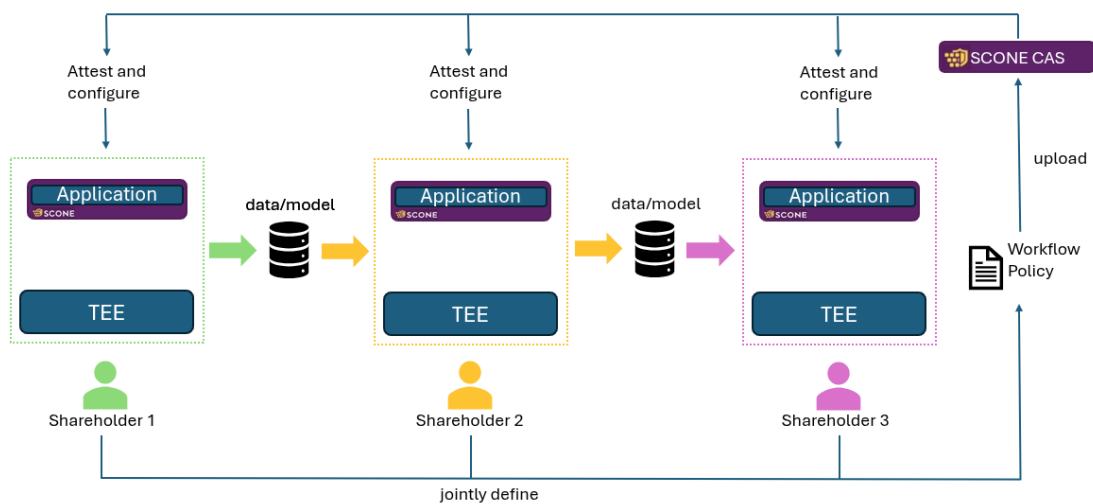


Figure 1-2: Multi-stakeholder access to registered devices through SCONE network shield

Simple declarative policies will enable the protection of the IP of the individual stakeholders. To enable a declarative, confidential approach SMARTY needs to be able to support different hardware and different security needs without the need to change the program code of existing applications, and hence a framework to use confidential CPU. In the proposed approach multi-stakeholder computations are viewed as workflows and each component of a workflow is a confidential service that can only be read by authorized, confidential services.

On the hardware level, the use of an Attested Interface Unit (AIU) is proposed, that is a hardware extension attached to all computation units of a system. AI services will be used to monitor security performance. SDP principles will also be applied.

Section 5 refers to the dynamic formation of network swarms (Task 4.4). This entails the creation of a highly programmable edge network forwarding plane using programmable hardware. Important development features include the synchronization of swarm networks and the serverless computing, particularly in the form of Function-as-a-Service (FaaS). The synchronization model proposed is built on a combination of hierarchical and cluster-based architectures, specifically implemented using ONOS SDN controllers. With respect to serverless computing in SMARTY, a novel approach of semantic-aware serverless deployment has been proposed and the key enabling components have been designed (P4 load balancer, semantic-aware traffic steering etc). The next Figure 1-3 displays the complete concept of the networking

swarms along with the semantic interfaces, the telemetry mechanism, the Hybrid-hierarchical Synchronization concept as well as the different attributes exposed to the application developer.

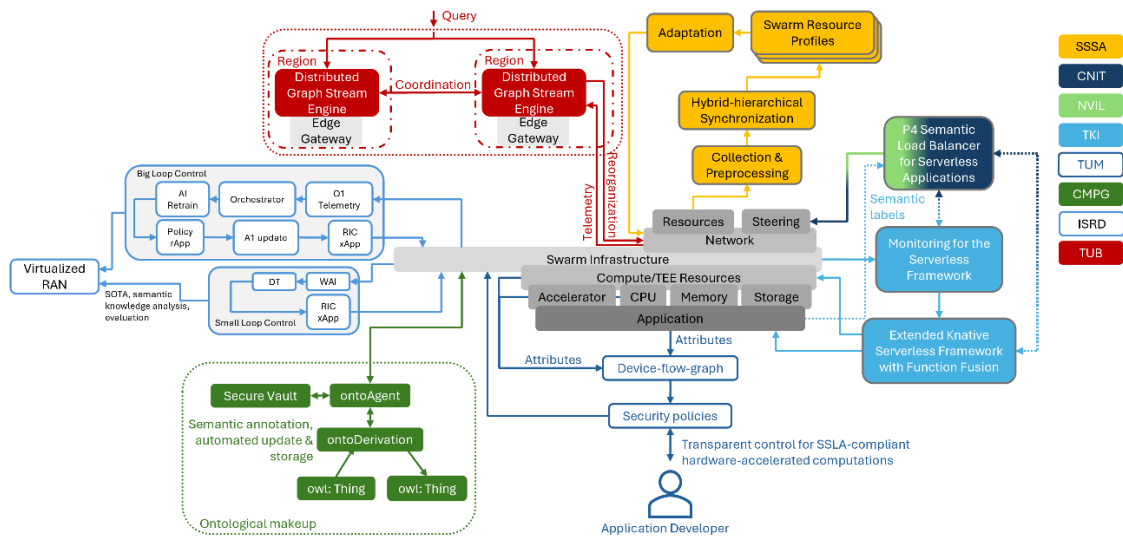


Figure 1-3: Dynamic Network Swarm concept with its main components and features

FaaS is desirable in many application areas, but FaaS platforms are not suitable for running all types of applications like SMARTY Use Case 5. In such applications, latency-sensitive applications, where response time must meet strict requirements, are not among those supported out-of-the-box. To this end, within SMARTY, the open-source FaaS framework will be extended with capabilities for running such applications using only the FaaS platform capabilities and integrating it with network telemetry to provide even lower latency. In this deliverable, the key enablers for these extensions are explored.

Finally, other studies carried out include a) the *swarmification* of RAN that can be perceived as the extended capabilities for the way network operates on a daily basis, b) a programming model around the concept of Device Flow Graphs (DFGs) enabling explicit and transparent control of the application's data flow between CPU and accelerators leveraging various types of attribute, c) the design of a swarm intelligence framework for SMARTY, emphasizing the abstract-level formation and coordination of swarm nodes and d) distributed graph stream engines for adaptively collecting telemetry and answering queries.

2. SOFTWARE DEFINED PERIMETERS IN SECURED NETWORKING

Work described under this section concerns the first definition and the architectural design of SMARTY Software Define Perimeter (SDP) along with its capability-based access control and a network shield to protect access to authorized/registered only devices. The whole design will be deployed in SMARTY use cases to demonstrate:

- Innovative security framework based on secure user Authorization and Authentication via the User management module of the Experimentation Tools, software defined perimeter protection and isolation of vertical applications and experimenters via dedicated network slices.
- Readiness of secure and trusted access AAA mechanisms at the User Management module and Software Defined Perimeter (SDP) mechanisms.
- Readiness of the implemented Security Framework for providing a secure perimeter around VNFs, NetApps and API levels.

2.1 SOFTWARE DEFINED PERIMETERS

Software defined perimeters (SDP) provide dynamic perimeters that are virtually ‘invisible’, i.e. stealthy, to outsiders and considers zero-trust while allowing entry into the perimeter of only authorised users and end points. Unique cryptographic keys are provided to enable connections to authorised users and end points, and only to those, allowing more granular access to services protected by SDP. SDP provides the ability to deploy partitioned network perimeters that are invisible to outsiders, in any location where internal services and Network Applications can be protected. SDP helps mitigate a broad set of security risks and vulnerabilities by ensuring secure network connections and adopting the assumption that there is no trust between potential participants. A key component of SDP entails rendering an organisation’s ICT infrastructure “invisible” or “dark”, meaning no DNS information or IP address information is visible and protected ICT resources cannot be detected from the Internet.

The SDP architecture entails three main entities, the SDP Controller, the client Initiating SDP Host (IH) and the endpoint Accepting SDP Host (AH). SDP Hosts can either initiate connections or accept connections. These actions are managed by interactions with the SDP Controllers via a secure control channel. Thus, in SDPs, the control plane is separated from the data plane to enable a completely scalable system. In addition, all of the components can be redundant for scale or uptime purposes. As for the SDP Controller, its basic function is to determine which SDP Hosts can communicate with each other. The Controller may relay information to external authentication services such as attestation, geo-location, and/or identity servers. The initiation of the SDP Hosts can communicate with the SDP Controller to request a list of accepting hosts that they can connect and then the SDP Controller can request information such as hardware or software inventory from the initiating hosts before providing any information. The AH is also termed as an SDP Gateway, as it marks the entry point (or gateway to) the perimeter.

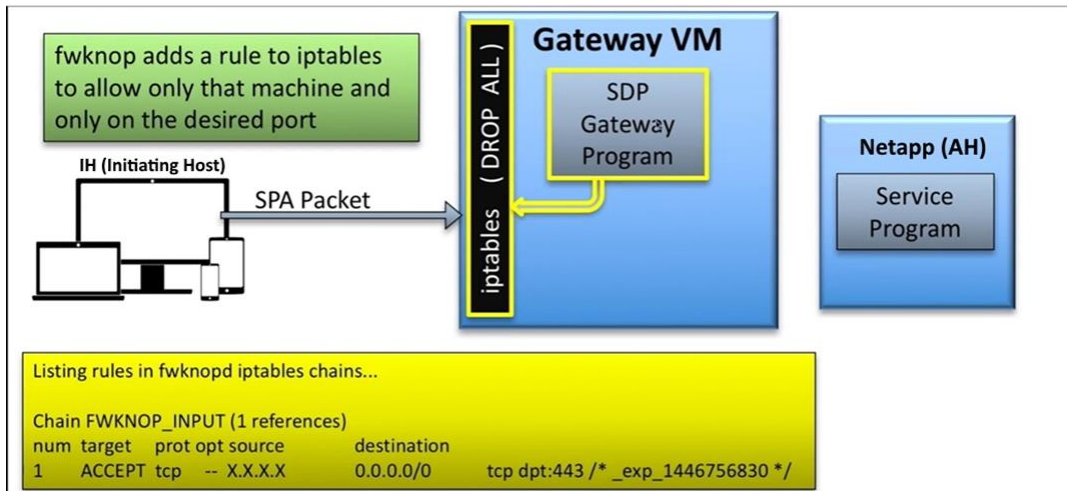


Figure 2-1: Perimeter Establishment workflow

The used facility in SMARTY utilizes an OpenStack NFVI; however, while OpenStack already supports Firewalls for VM instances via the Security Groups function, these are lacking advanced functionalities such as zone-based firewall and software-defined perimeters. To deliver a flexible and secure solution, we have started from an NFV Layer 3 Router and Firewall platform, where the Software-Defined Perimeter solution can be seamlessly deployed and configured.

As mentioned above, the NFV Infrastructure is based on the OpenStack ecosystem, i.e., OpenStack compute nodes and an OpenStack VIM. A centralized ETSI OSM instance is leveraged as the NFV Orchestrator and VNF Manager. In SMARTY, VyOS security function is used in the form of a VNF, via the ETSI OSM orchestrator. VyOS offers a software implementation of a Layer 3 Router and Firewall functions and is being adopted as the baseline platform for the SDP Gateway. VyOS is deployed as a Virtual Security Function, i.e., packaged within a VNF which simplifies onboarding, lifecycle management and configuration management. VyOS is completely free and open source, with documented internal APIs and build procedures, while offering a scriptable CLI for configuration purposes.

2.2 CAPABILITY-BASED ACCESS CONTROL

Hardware accelerators are transforming data processing on cloud platforms, with GPUs, TPUs, and FPGAs delivering exceptional compute density for emerging applications. Many of these applications handle sensitive data subject to stringent regulations like GDPR and CCPA, making strong security and regulatory compliance guarantees indispensable. However, while emerging trusted computing approaches aim to secure communication between CPUs and accelerators, they require application developers to rely on large Trusted Computing Bases (TCBs) and provide limited fine-grained control over data processing and inter-device data flows. These shortcomings hinder the enforcement of compliance and the ability to provide meaningful guarantees, increasing the risks of security breaches, misconfigurations, and regulatory non-compliance.

We propose a hardware-software co-design introducing a trusted bus system for secure policy-compliant hardware acceleration. It enables developers to define data processing tasks for accelerators as Device Flow Graphs (DFGs), associate them with fine-grained security policies (SSLA), and enforce these policies by the foundational layers of the system stack at runtime.

In this context, the co-designed OS kernel provides all primitives to securely bootstrap and manage a single network node. After a secure boot, the kernel will establish mutual trust with all local compute units, i.e. “classical” CPUs and accelerators. Similar to a traditional operating system, the co-designed OS serves as the mediator between applications and the hardware resources they intend to use. Access to hardware is granted to an application through a secure communication channel with an OS service (i.e. network stack).

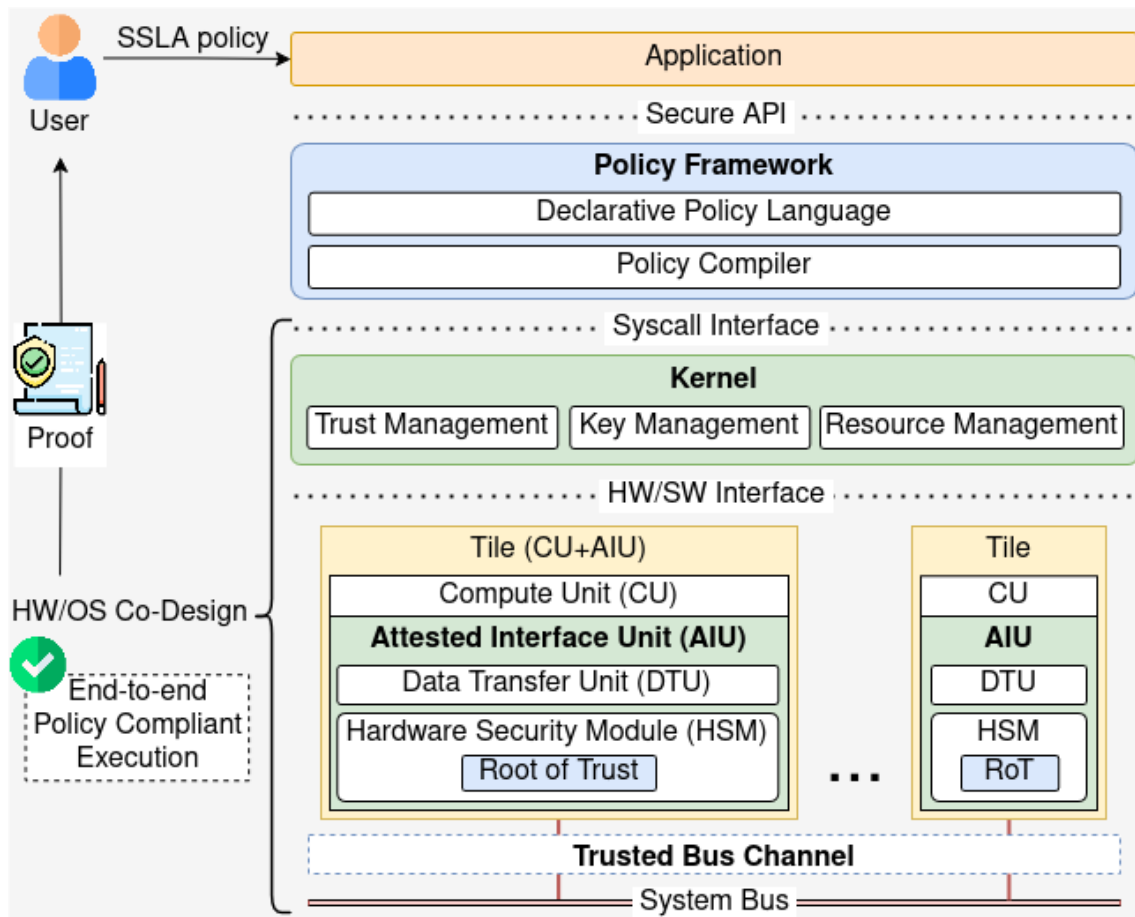


Figure 2-2: W/SW co-designed OS serving as the mediator between applications the hardware resources.

The co-design enables confidential computing through limited access to attested hardware resources.

2.3 POST-QUANTUM SECURE OPTICAL IPSEC TUNNEL FOR CONFIDENTIAL AI TRAINING

Quantum computers are expected to become commercially available in the coming years. These machines will be able to break our currently used public key cryptography methods. To withstand the quantum thread, the National Institute of Standards and Technology (NIST) chose to standardize different post-quantum cryptography (PQC) algorithms. Porting those protocols for secure transmission ie in data centers while still satisfying the ever-increasing computational demand of modern AI models is a significant technological challenge. During a confidential training job for AI, millions of IPsec connections are established per second. Accessing the service of an already trained model is usually done from outside of the data center, potentially by a mobile client in a wireless network. In modern AI models, inference is desired, putting

additional load on the cloud’s network after a client request has been submitted. This scenario can be obtained by combining both scenarios that are presented in this section: first a client connects to the cloud that runs the AI model via north-south traffic; then, within the cloud, the request is handled using out east-west IPsec scenario.

To establish a PQC-secured IPsec channel, an OpenSSL session is first initiated, to create an authenticated connection. For research purposes, self-signed certificates are used for authentication, in a real-life deployment, certificates from a trusted certificate authority would be required. The authentication process itself remains the same. Once the authenticated channel is in place, digital signatures are exchanged using the PQC algorithms Falcon and Dilithium. After that, an encryption key is exchanged via Kyber. The PQC-generated keys are then mixed with the OpenSSL key using an XOR operation, ensuring the resulting key remains secure if at least one of the original keys is secure. This resulting shared key is used to establish an IPsec tunnel that uses 256-bit Advanced Encryption Standard (AES) encryption operating in the Galois/Counter mode (GCM). AES-256 is recognized as resistant to quantum computing attacks. Depending on the re-keying interval, the key exchange procedure is repeated every ten minutes or every 2 30 cipher blocks. Setting up the IPsec tunnel requires superuser privileges. For programming the data processing unit (DPU), we use the APIs provided by NVIDIA’s DOCA (Data-Center-on-a-Chip) SDK1 to take advantage of the smartNIC’s flow steering. This process was performed on various devices to simulate different scenarios, as depicted in next Figure 2-3. To achieve statistically relevant results, each algorithmic procedure was tested 10.000 times. In the first setup (see Figure 2-3(a), a NVIDIA Jetson Nano was used as a mobile client with a 1 Gbit/s-capable Wi-Fi antenna to connect to the wireless network, which then connects to a 25G DPU in the cloud. The PQC-secured IPsec tunnel achieved an encrypted wireless throughput of 0.468 Gbit/s. This setup represents north-south traffic in a data center. The second scenario, illustrated in next Figure 2-3(b), involves an east-west channel within a data center, where multiple servers on the same network communicate by establishing PQC-secured channels. During the training phase of an AI model, this happens millions of times a second. In this setup, a PQC IPsec tunnel was established between two 100 Gbit/s DPUs connected via standard optical SMF. The throughput of the mobile client PQC-IPsec tunnel was tested using Iperf2. The east-west traffic throughput between the two 100G DPU was tested using the VIAVI traffic generator. While the VIAVI generator is capable of achieving high data rates of up to 400 Gbit/s per port, tests were conducted using 100 Gbit/s DPUs. The VIAVI traffic generator was connected between two DPUs via a QSFP cable for the testing.

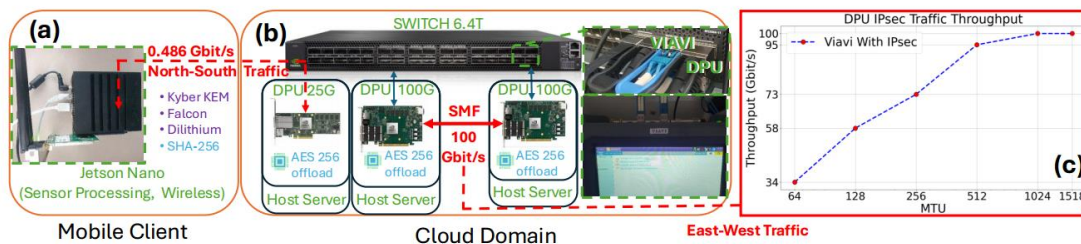


Figure 2-3: Experimental Setup: (a) A mobile client communicates with the cloud using a PQC-encrypted north-south IPsec tunnel at 0.486 Gbit/s for confidential AI training. (b) The servers in the data center for confidential AI training

use an PQC-based east-west IPsec tunnel, offloading AES-256 to the DPUs achieving (c), encryption at 100 Gbit/s traffic.

Using the Iperf traffic generator, the performance of the north-south IPsec tunnel established between the Jetson, as a mobile client and the 25G DPU in the cloud was analyzed. A throughput of 0.486 Gbit/s was achieved with AES-256 GCM-encrypted wireless. Since this scenario simulates a mobile client communicating with the cloud, the signal traverses multiple hops along the way. Consequently, we did not set the maximum transmission unit (MTU), as any device in the chain between the mobile device and the 25G DPU in the cloud can modify the MTU size.

In an intra-data center east-west traffic scenario, MTU size could be controlled. After setting up the east-west IPsec tunnel between the DPUs, the tunnel's throughput with various MTU sizes were measured using the VIAVI traffic generator. The results are shown in the above Figure 2-3(c). With 64 B MTU sized packets a throughput of 34 Gbit/s was achieved. Doubling the MTU to 128 B increased the throughput to 58 Gbit/s. Setting the MTU to 256 B resulted in a throughput of 73 Gbit/s. At 512 B MTU, the throughput reached 95 Gbit/s. Finally, from 1024 B MTU the throughput converges at a 100 Gbit/s line rate. This holds true for all MTU sizes greater than or equal to 1024 B, including jumbo-sized packets

Figure 2-4 shows the latency in CPU clock cycles introduced by cryptography operations executed on different devices and processors. The different variants of the algorithms account for different NIST security levels. The first row represents the operations that are performed by the client machine. During the execution of a signature algorithm (Falcon and Dilithium), the client must perform one step only, that is called verification. For the key exchange (Kyber), the client has to perform the key encapsulation. The second row represents the operations that are performed by the server machine. To successfully execute a signature algorithm, the server needs to do two steps: the key generation and the sign process. For the key exchange, the server must perform the key generation and the key decapsulation. The results shown in the second row of the Figure are a sum of the CPU cycles required for the execution of each single step (keygen+sign, keygen+key decapsulation). Each column represents one class of devices: The first column shows the operations performed on the 25G DPU. The second column shows the same for the Jetson, our mobile client device. The third column presents the results for an Intel Xeon CPU. The devices shown in column one and row three represent two scenarios within the data center. The first setting is that the machine does not offload any operation to the network interface card (NIC) and uses the NIC for outgoing and incoming communication only. Therefore, every time a new connection needs to be established, the host has to perform all cryptography operations itself which leads to a penalty. This can be seen in the third column of the Figure.



Figure 2-4: Average latency results in CPU clock cycles introduced by the execution of PQC cryptography. The first row shows the latency the client machine is penalized with. The second row represents the tax in CPU cycles introduced by PQC operations when performed on the server machine

In the second setting, depicted in the first column of above Figure 2-4, the host offloads the cryptography functions to its DPU. The IPsec tunnel is established between the two DPUs. The host sends unencrypted information as plaintext to the DPU via the PCI interface. The DPU handles encryption and decryption. Kyber is yet the only key encapsulation mechanism (KEM) chosen to be standardized by the NIST and is therefore expected to play a fundamental role in future network communication stacks. The algorithm performs similarly well in terms of execution speed compared to classical key exchange mechanisms (KEMs). Regardless, Kyber needs to transfer more data over the network. Comparing Falcon and Dilithium yields the following results: Falcon’s key generation is three orders of magnitude slower than Dilithium’s. Falcon’s sign process, as well as its verification, is slightly slower but within the same order of magnitude compared to Dilithium. However, Falcon’s signature is smaller than Dilithium’s. The choice which security level to use has to be determined by the developer ultimately. Which signature algorithm to choose in addition to Kyber depends on the use-case. In a data center, Dilithium proves advantageous due to its higher performance. With limited network capabilities, Falcon is more advantageous than Dilithium for its signature size is smaller and thus, fewer bytes need to be sent over the network.

2.4 TRANSPARENTLY SECURING COMMUNICATIONS

The foundation of confidential computing is that data can be protected while in use, in transit, and at rest. The main idea is that this protection is policy-controlled, i.e., one can enable the protection by declaring that, e.g., network communication is transparently encrypted such that only direct communication partners can read the network traffic in clear text. Our primary objective is to investigate to what extent this can be used to define a Software Defined Perimeter in a declarative manner.

As proof of concept, we consider the scenario where authorized users can only connect via devices that have been registered beforehand. These devices are *transparently* attested during registration as well as during each connection request. Attestation ensures that users can only access services and data when running on sufficiently protected devices that run the correct software.

The foundation of this transparent authentication and authorization is the SCONE network shield. This is a security component that provides encryption for network communications within the SCONE platform. It automatically encrypts TCP connections without requiring modifications to applications, making it particularly valuable for legacy services that do not support TLS encryption. The network shield ensures that all network traffic of an application is encrypted – without needing the cooperation of the application.

The diagram below shows that the SCONE Network Shield operates as an intermediary layer between an application (service) and the operating system. The "enclave" refers to a secure, isolated environment created by SGX, TDX, or SEV-SNP technology, ensuring that even privileged system administrators cannot intercept or read the encrypted data.

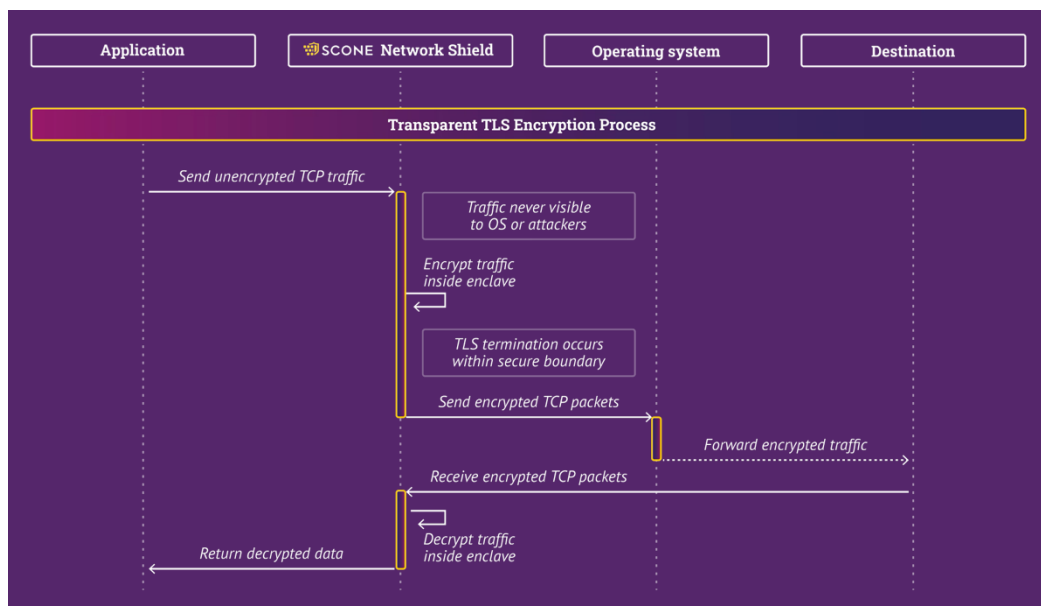


Figure 2-5: SCONE Network Shield between an application (service) and the operating system.

The Network Shield integrates with SCONE's attestation service securing communications. In particular, the attestation service:

- verifies that communicating parties run inside enclaves;
- uses TLS authentication between endpoints; and
- provides certificates proving enclave residency.

3. DEEP DATA PLANE SOLUTIONS: DECENTRALIZED FEATURE EXTRACTION (DFE) AND DEFENSIVE WIRESPEED AI

Work in this section investigates a framework to set up confidential workflows across multiple stakeholders. Simple declarative policies will enable the protection of the IP of individual stakeholders. To enable a declarative, confidential approach we need to be able to support different hardware and different security needs without the need to change the program code of existing applications, and hence a framework to use confidential CPUs, GPUs and accelerators will be investigated. The proposed methods must operate at wirespeeds bearing in mind the huge number of data exchanges and the increased bandwidth of networks. These tools include

- A novel real-time Graph Neural Networks –GNN– algorithms for malicious traffic detection,
- fault management data protection to secure networks running automated network management systems,
- computer vision techniques for wirespeed AI enabled network intrusion detection,
- probabilistic data structures for wirespeed traffic monitoring against malicious actions.

3.1 REAL-TIME GRAPH NEURAL NETWORK FOR MALICIOUS TRAFFIC DETECTION

Traditional approaches to detect cyberattacks, such as rule-based methods and statistical models, while effective in the past, have gradually been discontinued due to their limitations in adapting to sophisticated and evolving attack patterns. Indeed, these methods often rely on predefined signatures or static heuristics, limiting their effectiveness in complex network environments.

In recent years, Neural Networks (NNs) have been proposed as a potential alternative, leveraging their ability to learn complex patterns and adapt to dynamic environments. Among these, Graph Neural Networks (GNNs) show promising results when it comes to malicious traffic detection [WEI]. Unlike conventional NNs, GNNs are designed to operate on graph-structured data, enabling them to capture the intricate relationships and interactions inherent to network traffic. By leveraging graph structures, GNNs can model the connections between different hosts and flows, offering a deeper contextual understanding that traditional models cannot achieve. Several GNN models have been proposed for detecting various types of cyberattacks. However, most of these recent works have primarily focused on the development and training of the models rather than addressing their scalability, real-time applicability, or deployment in practical scenarios. A key requirement for such models when deployed in real-time network traffic analysis is the detection inference responsiveness, ideally achieved by analyzing only a few packets per flow. This enables the timely mitigation of attacks, preventing them from impacting network resources or causing denial-of-service disruptions. In addition, due to the exploding usage of edge devices, typically characterized by limited computational power and energy resources, detection models must be agile and possibly lightweight to account for these limitations.

To meet the aforementioned requirements, this work in the SMARTY project introduces a novel method for constructing traffic graphs, addressing the challenges of deploying GNNs for real-time malicious traffic detection. Specifically, our graph construction approach transforms raw

network traffic into a graph structure that effectively captures the temporal dynamics and flow relationships between packets.

Graph Construction

The graph is constructed from raw traffic packets by representing each host as a node. Directed edges are added between nodes for every packet exchanged, denoted as $e_{i,j}$ where $\{i,j\}$, where i and j are the source and destination nodes, respectively. This setup frames the problem as an edge classification task. Since node features are not utilized and edge classification models are not standard in practice, the graph is transformed such that each edge in the original graph becomes a node in a new graph G' . Formally, let $G = (V, E)$ be the original graph with nodes V and edges E . In the transformed graph $G' = (V', E')$, the nodes V' correspond to the edges E in G , while the edges E' in G' encode relationships between these new nodes.

Edges in G' are added as follows:

- **Temporal Sequence:** To capture the chronological order of packets within the same flow, edges are added between consecutive packets based on their arrival times. An undirected edge is created between two nodes (i.e., representing packets) v'_i and v'_{i+1} if they belong to the same flow and $t(v'_i) < t(v'_{i+1})$. Formally:

$$e' = (v'_i, v'_{i+1}) \text{ if } Flow(v'_i) = Flow(v'_{i+1}) \text{ and } t(v'_i) < t(v'_{i+1}).$$

This ensures that the temporal dynamics of packet exchanges within each flow are preserved in the graph structure.

- **Flow Relationships:** Edges are added between nodes representing packets from different flows if their endpoints share a connection. Specifically:

$$e' = (v'_i, v'_j) \text{ if } Dst(v'_i) = Src(v'_j) \text{ or } Src(v'_i) = Dst(v'_j).$$

This captures interactions between related flows, ensuring the graph encodes dependencies between shared endpoints.

Next Figure 3-1 illustrates the graph construction process.

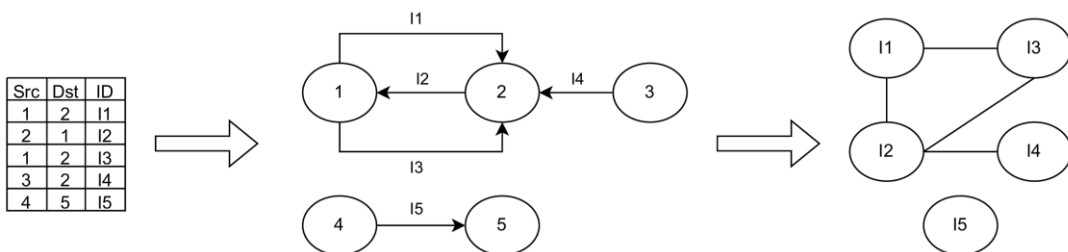


Figure 3-1: Constructing the traffic graph

To ensure computational efficiency, two constraints are introduced:

- **Degree Limitation:** The degree of each node is limited to a maximum value, d_{max} , determined experimentally.
- **Flow Compression:** An experimentally-determined number of packets from the same flow is aggregated into a single node, reducing the graph size while preserving critical information.

The Dataset

The selected dataset is the 5G-NIDD [Sam22], a network intrusion detection dataset created from a real-world 5G test network, collected using the 5G Test Network in Oulu, Finland. This dataset combines both attack and benign traffic across various attack scenarios. Traffic generation involved real mobile devices, ensuring authenticity and relevance.

The raw packets are used to build a dataset containing 11 packet features (i.e., *Length, TTL, Protocol, Layer, Flags, Transport Length, TCP Ack, TCP Flags, TCP Window Size, ICMP Type, Transport Protocol*) in addition to the arrival time, source and destination addresses and the classification label.

The dataset is then split into a training set and an evaluation set containing 70% and 30% of the original one respectively. In order to ensure that packets from the entire lifespan of the traffic are represented in both sets, the dataset is split into timesteps of 1 second each. Afterwards, the split percentage of the training and evaluation sets is applied to each of these partial datasets. It is worth noting that the distribution of malicious and benign packets is preserved when performing the split. The evaluation set contains about 185k packets belonging to benign flows and 116k packets belonging to malicious ones. Graphs are then constructed following the aforementioned methodology.

The Model

The exploited GNN to process this graph is a 6-layer GCN followed by a fully connected layer with dropout at the end for classification. For the first three convolutional layers, 32 units are exploited; while for the last three layers 8 units are used. The number of layers has been determined through a process of trial and error where it was determined that the minimum number of layers achieving high accuracy is 6. Similarly, different configurations of the convolutional layers' sizes have been tested to determine the best configuration. The goal of this model is to perform node classification where nodes are either singular packets or compressed packets belonging to the same flow. In order to train the GNN, an Adam optimizer is used with a learning rate of *0.005* and a weight decay of *0.0005*. The model is trained for a total of *30* epochs where it gets evaluated at the end of each epoch using the evaluation set. The version of the model that attains the highest F1 score throughout the training process is saved as the best-performing model.

Experiments and Results

The experiments are designed to determine the optimal values for critical hyperparameters, analyze trade-offs between F1 score and prediction time, and compare the model's effectiveness against a state-of-the-art alternative. All experiments were conducted on an Intel(R) Xeon(R) Gold 6238R CPU of 2.20GHz.

Maximum Node Degree

The node degree is defined as the number of edges a node has in a graph. As mentioned before, the maximum node degree in the input graphs to our model is to be determined experimentally. Therefore, the first experiment aims at measuring the F1 score and the prediction time for different graph sizes, with the graph size determined by (i) the number of nodes and (ii) the maximum node degree. We, therefore, train our model considering different configurations for the input graph where the number of nodes ranges between 10 and 100000 packets and the maximum degree ranges between 3 and 500. Each configuration defined by the number of nodes and their maximum degree is trained in order to record its best F1 score and the corresponding prediction time (i.e., the sum of the graph construction time and inference time).

Then, these results are plotted in order to visually decide the optimal value for the maximum node degree as shown in Figure 3-2 and Figure 3-3. It can be observed that the F1 score starts to converge around the value of 50 edges per node. As for the prediction time, it does not undergo significant changes by varying the number of edges; however, it depends mainly on the number of nodes in the graph.

The tradeoff plot between the F1 score and the prediction time is considered.

From Figure 3-4, it can be noted that the desired F1 score and prediction time can be achieved using a maximum number of edges of 10 or more. However, upon closer examination of the plot, it is evident that the desired trade-off between the F1 score and the prediction time happens when the maximum number of edges is 50 which is represented by the dash-dotted gray line. As the plot shows, going beyond 50 edges per node does not lead to significant changes in the F1 score. On the other hand, it affects the prediction time negatively as it results in higher prediction times for graphs of the same number of nodes.

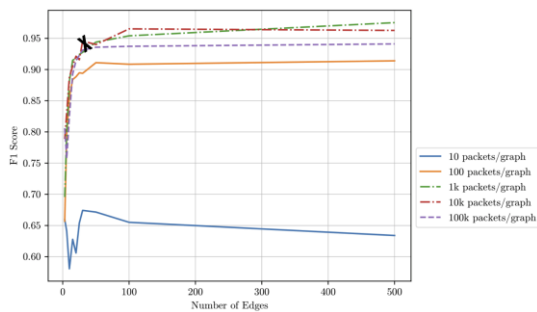


Figure 3-2: F1 Score vs Maximum Number of Edges

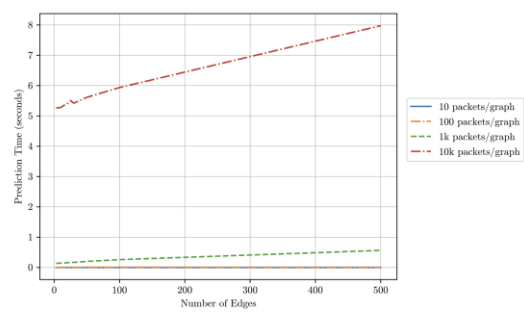


Figure 3-3: Prediction Time vs Maximum Number of Edges

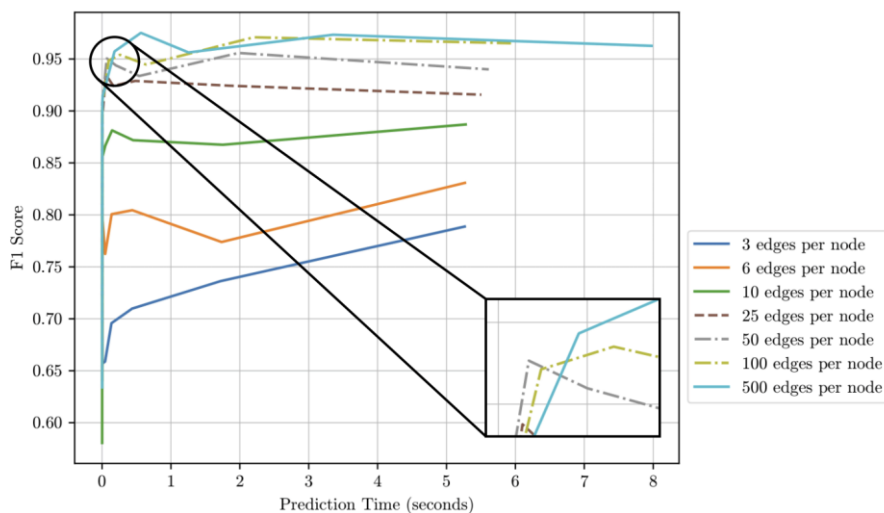


Figure 3-4: Prediction Time vs F1 Score

Optimal Node Granularity

The other parameter that is to be determined experimentally is the node granularity; i.e., how many packets belonging to the same flow can be compressed into a single node. Using the optimal maximum node degree, the graphs are constructed considering different granularity values. The best F1 score is recorded for each graph considering different classification windows;

i.e., total number of packets, and different node granularities. The goal is to find a trade-off between the F1 score and the prediction time in order to choose the best node granularity.

Upon collecting the results, the trade-off plot between the F1 score and the prediction time is visualized in Figure 3-5a. It shows that an acceptable F1 score can be achieved quickly with a granularity of 8. Upon deeper inspection of the plot (**Error! Reference source not found.b**), it can be observed that the F1 score does not improve significantly if the granularity is 200. Even though the prediction time is not significantly affected by increasing the granularity, it is worth noting that it depends on the chosen aggregation method. For simplicity, we chose the mean when aggregating the features which is relatively fast to compute. If more complex aggregation methods are to be tested, then it is important to keep the number of aggregated nodes as low as possible. Therefore, the chosen optimal granularity is 8 instead of 200.

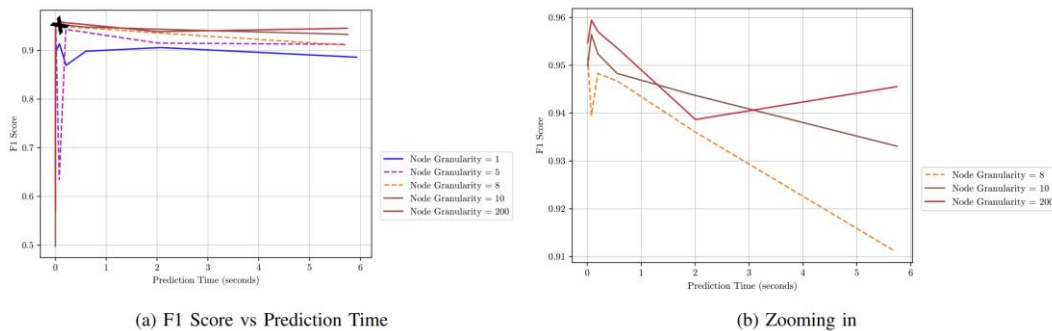


Figure 3-5: F1-Prediction Time Analysis with respect to Granularity

Attack Analysis

This experiment aims to test the number of packets our model needs to accurately detect different attacks. Different models were trained considering different classification windows for each. On average, they had a similar performance; therefore, we report here the results of the model trained with a classification window of 2000 packets. The test set used to evaluate the models during training was divided into a number of datasets such that each one would contain all the benign traffic in the original dataset in addition to the traffic related to only one attack type. Then, these datasets were transformed into graphs with different classification windows ranging between 3 and the classification window the model was trained on. So, for the 2000-packets model, the classification window size ranges between 3 and 2000.

The performance of the model with respect to each attack can be visualized in next Figure 3-6a. The model performs best at detecting Golden Eye attacks and Syn Flood attacks while struggling with Slow Loris attacks. Upon closer inspection in next Figure 3-6b, it can be shown that the maximum F1 score for each attack can be achieved using a classification window size in the interval of 50 to 75 packets.

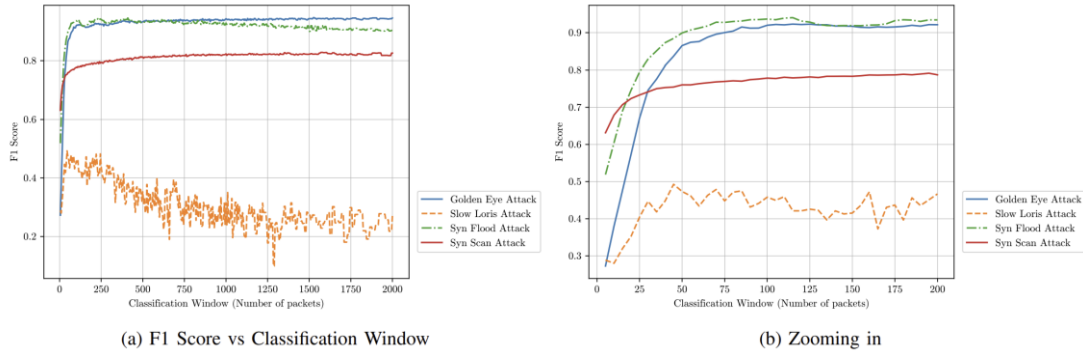


Figure 3-6: Attack Analysis

Comparison with FTG-Net

After choosing the best configuration for the graph, summarized in Table 3-1, a GNN was trained and evaluated with the purpose of comparing it to the model proposed in [Bar23].

Table 3-1: Summary of Experimental Configuration

Parameter	Optimal Value
Maximum Node Degree	50
Node Granularity (Packets per Node)	8
GNN Layers	6
Classification Window (Packets)	50-75

Our model generally achieves a higher F1 score while having a lower or similar prediction time for the small-time windows (see next figures). This means that our model outperforms FTG-Net in a real-time attack detection scenario. It is also worth noting that, while FTG-Net has a relatively higher weighted F1 score of about 0.97, it has a generally unstable lower binary F1 score. In any case, our model still has a similar weighted F1 score of 0.93 which is more coherent with the binary F1 score.

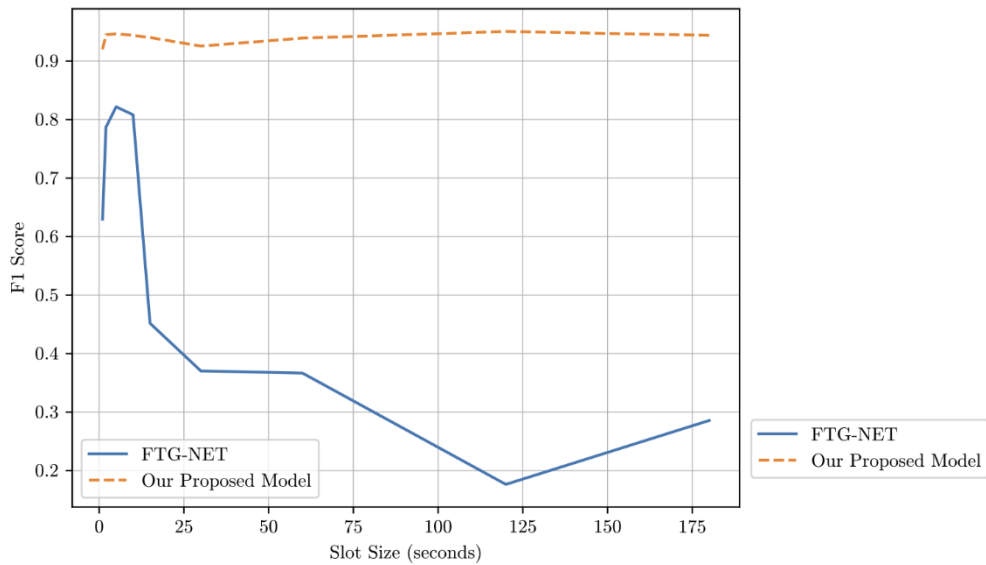


Figure 3-7: FTG-Net vs Our Model (F1 Score)

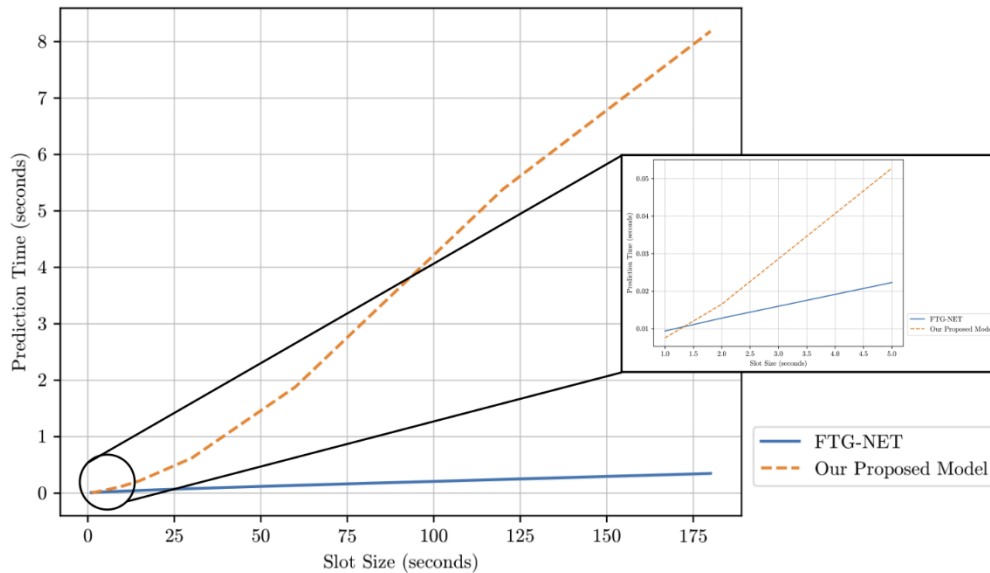


Figure 3-8: FTG-Net vs Our Model (Prediction Time)

Discussion

FTG-Net [Bar23] is a good model as it is generally fast and achieves good results if the classification window is large enough. However, one of its drawbacks is that it builds a number of flow graphs for each classification window, each containing all the packets exchanged between two endpoints within the classification window. In doing so, it assumes that the classification occurs in a centralized way. Nonetheless, with high packet rates and the need to perform the classification quickly, distributing the classification becomes inevitable. This means splitting the graph among different workers. This does not guarantee that all packets between the same endpoints are received by the same worker. Therefore, it becomes very challenging to distribute the process due to having a large number of graphs. That is why designing a new GNN that operates on one graph per classification window is needed in order to allow for future

distribution of the classification. As shown in the previous subsection, our model achieves satisfactory results for small-sized graphs. However, when the graph size becomes larger, it maintains the high prediction accuracy, but it suffers from large latency. This could be seen as a downside to our model because there are other models like FTG-Net that maintain a low prediction time regardless of the classification window. Upon further inspection of the prediction time, it was observed that the main reason behind the growing latency is the time needed to construct the input graphs. While FTG-Net constructs a large number of smaller graphs, our model constructs one graph for the entire classification window. In fact, if the graph construction time is ignored, our model is faster than FTG-Net by an order of magnitude. This is because it is smaller in size compared to FTG-Net as it is made up of just one GNN, unlike FTG-Net which consists of a hierarchy of GNNs. In order to mitigate this, we define the classification window as a constant number of packets, instead of a fixed time window which may contain a varying number of packets. Another important observation is related to how the model performs with respect to different attacks. The model is capable of accurately detecting SynFlood and GoldenEye attacks while struggling with SlowLoris and SynScan attacks due to the low presence of these attacks in the dataset. In fact, they make up only 9% of the attacks in the dataset. A possible solution to this is to retrain the model on a dataset containing more examples of these attacks.

To summarize, GNNs, designed to work on graph-structured data, have the potential to model complex data patterns, offering a more comprehensive understanding of network traffic. In this work, we proposed a novel GNN-based system for real-time malicious traffic detection, addressing critical challenges such as scalability, inference speed, and practical deployment. Through experimental analysis, we demonstrated that our model achieves an F1 score of 0.9 or more with a graph configuration that includes a maximum node degree of 50 edges and a node granularity of 8 packets. Additionally, we determined that accurate detection can be achieved with as few as 50-75 packets for certain attack types. Additionally, we also compared our model to FTG-Net, highlighting the strengths and weaknesses of each one over the other.

As next step, we plan to deploy the proposed GNN on Data Processing Unit (DPU) for real-time validation. This joint work CNIT-SSSA has been accepted for publication in:

[Ibr25] Ahmed Ibrahim, Emilio Paolini, Filippo Cugini, Francesco Paolucci, “Real-Time Graph Neural Network for Malicious Traffic Detection”, IEEE International Conference on Machine Learning for Communication and Networking (ICMLCN), 26–29 May 2025, Barcelona, Spain

3.2 EFFECTIVENESS OF CONFIDENTIALITY-PRESERVING CLUSTERING ALGORITHMS FOR SOFT FAILURE DETECTION IN OPTICAL NETWORKS

Optical networks are crucial for high-capacity and high-speed transmissions and demand exceptional performance and service quality. Their complexity makes manual monitoring impractical and calls for automated network management. Data sharing with third parties for fault management introduces data confidentiality issues, given the varied data protection policies among operators. This emphasizes the need for automated, secure network management systems that utilize network data without compromising confidentiality.

The analysis of optical network data by a third party shares several commonalities with existing literature on Machine Learning as a Service (MLaaS) or cloud-based machine learning services. However, it also presents unique characteristics and challenges. The nature of the data -whether

it be images, text, or numerical; real-time, streaming, or batch-processed- along with the criticality of response time, requires diverse solutions to ensure secure data transfer and analysis, preventing exposure during the process. We introduce a method for soft failure detection through the use of an unsupervised machine learning algorithm applied to scrambled data. Data scrambling serves as our chosen technique for safeguarding sensitive information. It is a straightforward yet effective method to disguise the data's original sequence and structure, maintaining its statistical properties. This approach not only thwarts unauthorized entities from disclosing the data but also enables legitimate users to employ the data for machine learning applications.

To detect the soft failures in the data, we use unsupervised machine learning algorithms which do not need human guide or beforehand training for data clustering. Data clustering can differentiate between the normal and different abnormal states of the network with grouping similar or proximate data points. We apply and compare six unsupervised machine learning algorithms for data clustering. We evaluate the performance of these algorithms in failure detection on both scrambled and unscrambled data using some evaluation methods. These evaluation methods measure the quality and validity of the clustering results, by comparing the clusters with the true labels or by assessing the internal structure of the clusters. They can also help us to choose the best clustering algorithm for our data, and to analyze the effects of data scrambling on the clustering performance.

The primary contribution of this paper lies in giving a set of cluster-based algorithms working in a confidentiality preserving scheme, supported by comprehensive evaluation methods to evaluate their performance in failure detection.

Proposed Approach and implementation

Figure 3-9 below depicts the proposed approach. Following the chain from the left side of the figure, initially, the network state information dataset is collected by means of a Kafka-based monitoring framework.

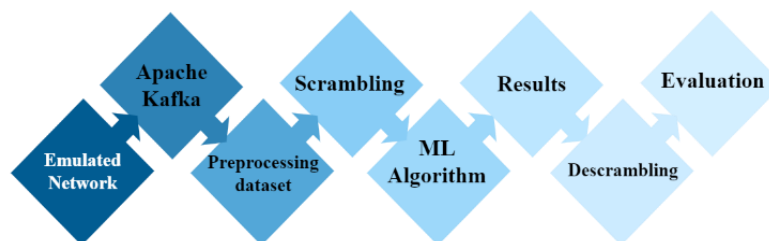


Figure 3-9: Proposed approach

The preprocessing phase involves reshaping the dataset into a format suitable for clustering algorithms, which requires time-series observations of all network elements. Each observation is documented as a single row in the dataset. Subsequently, scrambling is performed in the dataset to maintain confidentiality. All the aforementioned steps are conducted at the network provider's premises. The scrambled dataset is then dispatched to a third party for failure detection analysis. The pivotal fifth block entails applying various clustering algorithms to the scrambled dataset. The third party executes this analysis and communicates the results back to the network provider. The sixth block involves the transmission of the clusters of permuted results from the third party to the provider, who then retrieves the data in the seventh block to

pinpoint failures. The provider, possessing knowledge of the network's states and failure specifics, decodes the results to identify the failures' locations.

A. Data Collection and Preprocessing

Analyzing the data produced from emulated system shows that each equipment might have different response time when queried about its state. In the proposed approach the telemetry metrics are requested every 4 s, but the responses to the requests might be received with slightly different delays, causing the measured metrics not to be synchronized. This inevitable phenomenon can reflect the unpredictable behavior of actual optical networks. To avoid that data are stored out of order, we implemented an algorithm to recognize the contemporary queries and store them together in the dataset. Table I. shows the shape of data received from Kafka producer, related to a simple network scenario. We developed a code to change this shape (each measurement in a row) into Table O2. (all the contemporary measurements in a row), while intelligently integrating the right group of measurements for each observation.

Table O3 is a specific cut of the dataset showing inconsistency between groups of measurements. The first three rows are the measurements from normal condition of the network, the last one belongs to failure condition while we assigned the two highlighted rows into failure condition for evaluation part, however they can't be absolutely recognized as failure or normal condition. In more detail, despite the input power of amplifier 1 indicating a failure condition, the OSNR and BER measurements do not immediately reflect this, exhibiting a time lag. This delay is attributed to the time required for devices to process and estimate these parameters. Such discrepancies, stemming from different equipment response times, are likely to occur in real-world scenarios, highlighting the importance of incorporating real data into algorithm evaluation. In the following section, we will demonstrate how this type of inconsistency can also lead to confusion in machine learning algorithms when attempting to detect failures.

Table. O2. Dataset format received from Kafka producer

Time stamp	Type	ID	BER	OSNR	Input Power	Output Power
1665672632	Devices	spo1/18/11	5.08E-08	34.1		
1665672632	Devices	spo2/18/11	2.98E-08	29.5		
1665672633	Infrastructure	Ampli3			-12.7	2.6
1665672633	Infrastructure	Ampli1			-11.2	3.9
1665672633	Infrastructure	Ampli2			-11.1	4
1665672633	Infrastructure	Ampli4			-12.3	3

Table. O3.Preprocessed and cleaned dataset

BER spo1	BER spo2	OSNR spo1	OSNR spo2	Amp1 Input Power	Amp1 Output Power	Amp2 Input Power	Amp2 Output Power	Amp3 Input Power	Amp3 Output Power	Amp4 Input Power	Amp4 Output Power
-16.8	-17.5	34.3	29.8	-11.3	3.9	-11.1	4	-12.7	2.6	-12.3	3
-16.8	-17.5	33.7	29.8	-11.2	3.9	-11.1	4	-12.7	2.6	-12.3	3
-16.8	-17.5	34.3	29.7	-11.3	3.9	-11.1	4	-12.7	2.6	-12.3	3
-16.8	-17.5	32.3	29.1	-13.8	1.5	-13.5	1.6	-15.1	0.7	-14.2	1.2
-10.8	-17.5	33.3	29.2	-13.8	1.4	-13.5	1.6	-15	0.7	-14.2	1.2

-9.1	-17.5	19	28.9	-13.8	1.4	-13.5	1.6	-15.1	0.7	-14,2	1.2
------	-------	----	------	-------	-----	-------	-----	-------	-----	-------	-----

B. Confidentiality preservation

To safeguard sensitive data, we propose data scrambling before transmission to a third party. This technique effectively masks the data's original sequence and structure while preserving key statistical properties, such as the mean, variance, and covariance. Data scrambling is executed by randomly permuting the rows and columns of the data matrix using a permutation key. This key, akin to an encryption key, is retained by the data owner. Permuting the matrix's rows reorders them without altering their content, as it changes the sequence of the rows but not the individual elements. Similarly, permuting the columns alters their sequence without changing the content within each column. The simultaneous application of both row and column permutations maintains the integrity of the matrix's structure, which is crucial for ensuring that clustering algorithms remain unaffected by the scrambling process.

C. Unsupervised machine learning algorithms setup

For clustering network states, we leverage the benefits of unsupervised machine learning algorithms, which do not require training with labeled data. This ensures that the third party analyzing the network never accesses the original, sensitive data, thereby enhancing confidentiality. We utilize Python scripts for the scrambling process and the Scikit Learn library for implementing the clustering algorithms with initial setups being consistent for each algorithm on each dataset. The initial parameter for K-means is the number of clusters. DBSCAN needs two input parameters: `eps` which is the radius of the neighborhood around each data point, and `min_sample`, the minimum number of points required to form a cluster. Similarly, HDBSCAN needs two input parameters; `min-cluster_size` which is the minimum number of points required to form a cluster and `cluster-selection-epsilon` that is a distance threshold, clusters below this value will be merged. OPTICS needs one input parameter which is the minimum number of points to form a cluster as `min_samples`. Affinity propagation needs Pseudo-random number generator to control the starting state. Finally Spectral Clustering needs the dimension of the projection subspace as `n-clusters`.

Evaluations and results

In this section, we detail the outcomes of applying various clustering algorithms across two scenarios, each with distinct network configurations and failure conditions. Initial setup for different clustering algorithms is determined through numerous iterations of algorithm execution, fine-tuning them for optimal performance as in following Table 04.

Table. 04. Initial parameters setup for each algorithm

	Simple-network's dataset	Complex-network's dataset
K-means	n_clusters=2	n_clusters=13

DBSCAN	min_sample=70, eps=2	min_sample=70, eps=2
HDBSCAN	min_cluster_size=70, cluster_selection_epsilon=2	min_cluster_size=70, cluster_selection_epsilon=2
OPTICS	min_samples=70	min_samples=70
Affinity propagation	random_state=2	random_state=2
Spectral clustering	n_clusters=2	n_clusters=13

The network scenarios considered to develop two types of datasets are the following. A simple network scenario, illustrated in the next Figure 40- (top), features two transponders, one acting as TX and the other acting as RX, connected to an optical link consisting of four optical spans (SMF fiber of 70km each), after each span an EDFA amplifier is utilized to compensate the loss experienced along the line, entering the next span with 0dBm. We utilized a Variable Optical Attenuator (VOA) to realize the soft failure, reducing the optical power by 5dB in the first span. In the system a single WDM DP-QPSK channel is transmitted at 193.1 THz. Conducted for over an hour, this procedure induces five distinct failures within the chosen span, with each failure lasting five minutes and occurring amidst random periods of normal functioning. This configuration is designed to generate two clusters representing the normal state and the failure state for analysis by the clustering algorithms.

A complex network scenario, illustrated in the next Figure (bottom), presents a different network configuration. This setup involves two TX and two RX transponders, operating with different wavelengths (channel 1 at 193.1 THz and channel 2 at 193.2THz respectively) transmitting over the same four optical spans. To simulate various failure scenarios, Failure Actuators (FAs) are deployed across all spans. Each FA includes Wavelength Selective Switch (WSS) and can produce three distinct failure types on each span: a soft failure on channel 1, a soft failure on channel 2, and a soft failure on both channel 1 and channel 2. Each failure is programmed to manifest separately within its respective span, enabling the delineation of 12 unique failure states alongside a single state of normal operation. Failures are transient like the simple network. These conditions are expected to yield 13 discrete clusters, facilitating the subsequent application of clustering algorithms for analysis. In our evaluation, the performance of the proposed workflow is compared with a benchmark workflow that does not implement scrambling. This comparison is crucial to determine how data scrambling impacts the clustering algorithms' effectiveness.

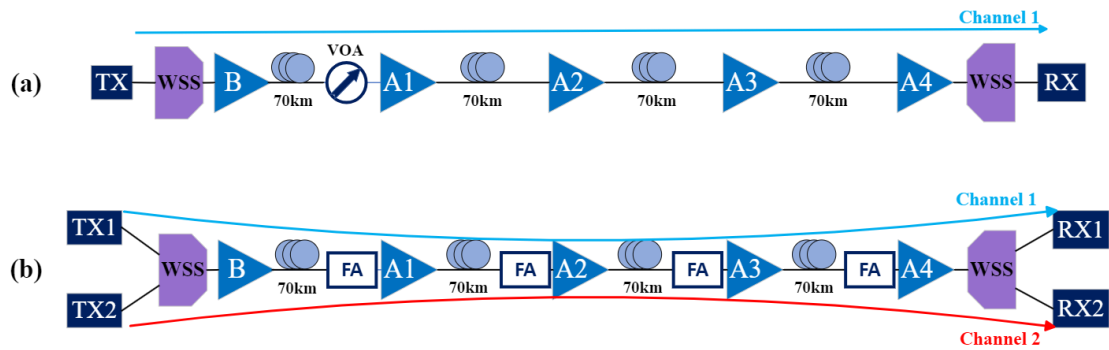


Figure 3-0: (a) simple network design with one span failure and one channel frequency; (b) complex network design with four span failures and two channel frequencies, either individually or simultaneously

Evaluating machine learning algorithms is crucial to identify the most effective one. However, due to the vast size of datasets and the multitude of clusters, manually verifying clustering results against true labels is impractical, often assessed by confusion matrix. To gauge the quality and validity of clustering outcomes, we employ both external and internal evaluation methods. External evaluations compare clusters to true labels, while internal evaluations examine the clusters' intrinsic structure. Given that different methods have varying assumptions and limitations, and may not concur on the optimal clustering solution, it is vital to apply multiple evaluation metrics and interpret the results with discernment.

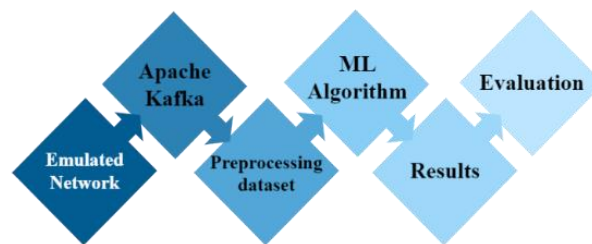


Figure 3-: Benchmark scenario

For external evaluations, we use Homogeneity, Completeness, V-measure, Adjusted Rand Index (ARI), and Adjusted Mutual Information (AMI). Homogeneity assesses if each cluster exclusively comprises data points from a single class, aiming for a perfectly homogeneous result where each cluster corresponds to one class label. Completeness evaluates whether all members of a class are grouped into a single cluster, with perfect completeness achieved when this criterion is met. V-measure is a harmonic mean of homogeneity and completeness, providing a singular metric. ARI measures the agreement between cluster assignments and class labels, based on the Rand Index, which considers the proportion of data point pairs correctly clustered. AMI quantifies the shared information between cluster assignments and class labels, reflecting the mutual dependence of the variables. The absolute value of these metrics ultimately range from 0 to 1, where higher values signify superior clustering quality. For internal evaluation, we utilize the Silhouette coefficient, which quantifies how well data points fit within their assigned clusters. Given that evaluation methods generally agree that values closer to 1 indicate better performance, we opted to use the average of these evaluations as a composite metric. This simplifies the comparison of algorithms in a visual format.

The first evaluation considered the simple network scenario, depicted in Figure 41. The different algorithms applied to the unscrambled dataset are detailed in Table 05 while Table 06

contains the results for the scrambled dataset. The evaluation results, as presented in Table 05, indicate that the top three clustering algorithms in terms of performance are DBSCAN, K-means, and HDBSCAN. The ranking in Table 06 mirrors those in Table 05, suggesting that data scrambling does not adversely affect the performance of DBSCAN, HDBSCAN, K-means, or OPTICS. Interestingly, data scrambling appears to improve the performance of Spectral Clustering. However, Affinity Propagation and OPTICS do not perform as well, primarily due to their inability to accurately determine the number of network states. This outcome demonstrates that, within a simple dataset, various algorithms can effectively cluster data, maintaining their performance even when the data is scrambled.

Table 05. The results of the evaluation methods on the original dataset from simple network scenario

Machine Learning Algorithm	DBSCAN	HDBSCAN	OPTICS	Kmeans	Affinity Propagation	Spectral Clustering
Homogeneity	0.945	0.763	0.827	0.763	1	0.735
Completeness	0.753	0.746	0.241	0.773	0.144	0.73
V-measure	0.838	0.755	0.373	0.768	0.251	0.732
Adjusted Rand Index	0.926	0.885	0.216	0.887	0.089	0.864
Adjusted Mutual Information	0.837	0.754	0.371	0.767	0.222	0.732
Silhouette Coefficient	0.942	0.974	0.63	0.985	0.489	0.971
Average of Evaluations	0.874	0.813	0.443	0.824	0.366	0.794
Standard Deviation	0.070	0.087	0.218	0.084	0.310	0.093
Number of clusters	2	2	4	2	86	2
Number of noises	22	2	48	0	0	0
Initial parameters	eps=2, min_samples=40	min_cluster_size=70, cluster_selection_epsilon=2	min_samples=70	n_clusters=2	random_state=5	n_clusters=2, assign_labels='discretize'

Upon comparison, we observe notable differences from the simpler dataset scenario. Overall, except for DBSCAN, all algorithm evaluations yield slightly lower scores in this more complex setting. The performance drop is particularly pronounced for K-means, suggesting it may not be suitable for complex datasets. OPTICS and Affinity Propagation continue to underperform, failing to accurately identify clusters. Among the remaining algorithms, Spectral Clustering shows improved results with scrambled data but is hindered by its time-intensive nature and inability to autonomously determine cluster numbers. Despite the close resemblance between DBSCAN and HDBSCAN, DBSCAN consistently outperforms, affirming its robustness in complex network scenarios.

Table 06. The results of the evaluation methods on the scrambled dataset from simple network scenario

Machine Learning Algorithm	DBSCAN	HDBSCAN	OPTICS	Kmeans	Affinity Propagation	Spectral Clustering
Homogeneity	0.945	0.763	0.826	0.763	1	0.783
Completeness	0.753	0.746	0.24	0.773	0.077	0.763

V-measure	0.838	0.755	0.372	0.768	0.143	0.773
Adjusted Rand Index	0.926	0.885	0.216	0.887	0.005	0.889
Adjusted Mutual Information	0.837	0.754	0.37	0.767	0.061	0.773
Silhouette Coefficient	0.942	0.974	0.629	0.985	0.202	0.968
Average of Evaluations	0.874	0.813	0.442	0.824	0.248	0.825
Standard Deviation	0.070	0.087	0.218	0.084	0.342	0.077
Number of clusters	2	2	4	2	131	2
Number of noises	22	2	49	0	0	0
Initial parameters	eps=2, min_samples=40	min_cluster_size=70, cluster_selection_epsilon=2	min_samples=70	n_clusters=2	random_state=5	n_clusters=2, assign_labels='discretize'

Table 07. The results of the evaluation methods on the original dataset from complex network scenario

Machine Learning Algorithm	DBSCAN	HDBSCAN	OPTICS	Kmeans	Affinity Propagation	Spectral Clustering
Homogeneity	0.926	0.901	0.873	0.821	0.992	0.739
Completeness	0.857	0.844	0.63	0.5	0.272	0.815
V-measure	0.89	0.872	0.732	0.622	0.427	0.775
Adjusted Rand Index	0.926	0.909	0.609	0.292	0.029	0.841
Adjusted Mutual Information	0.888	0.87	0.727	0.618	0.396	0.772
Silhouette Coefficient	0.705	0.715	0.115	0.465	0.311	0.535
Average of Evaluations	0.865	0.852	0.614	0.553	0.405	0.746
Standard Deviation	0.076	0.065	0.239	0.163	0.292	0.100
Number of clusters	13	13	16	13	185	13
Number of noises	207	152	2999	0	0	0
Initial parameters	eps=2, min_samples=70	min_cluster_size=70, cluster_selection_epsilon=2	min_samples=70	n_clusters=13	random_state=2 0	n_clusters=13, assign_labels='discretize'

From the standpoint of preserving confidentiality, the consistency of evaluation metrics between the original and scrambled datasets is a key indicator of effectiveness. As evidenced by Figure 42 and the accompanying tables, DBSCAN and HDBSCAN and Spectral clustering maintain high performance, suggesting they are better at finding clusters while even preserving data confidentiality during evaluation. This can be attributed to their density-based clustering approach for DBSCAN and HDBSCAN, which identifies clusters as regions of high density separated by low-density areas. Unlike K-means, which presumes convex-shaped clusters, DBSCAN's flexibility allows for clusters of any shape, making it more suitable for our datasets. Importantly, data scrambling does not compromise the performance of these algorithms since it does not alter the underlying density and distribution of data points. Consequently, DBSCAN and HDBSCAN yield consistent results with both scrambled and unscrambled data, ensuring the confidentiality of the data and its owners.

Table 08. The results of the evaluation methods on the scrambled dataset from complex network scenario

Machine Learning Algorithm	DBSCAN	HDBSCAN	OPTICS	Kmeans	Affinity Propagation	Spectral Clustering
Homogeneity	0.926	0.902	0.865	0.828	0.991	0.737
Completeness	0.857	0.845	0.616	0.505	0.269	0.811
V-measure	0.89	0.872	0.72	0.627	0.423	0.772
Adjusted Rand Index	0.926	0.91	0.595	0.295	0.033	0.841
Adjusted Mutual Information	0.888	0.87	0.715	0.623	0.388	0.769
Silhouette Coefficient	0.705	0.716	0.105	0.471	0.273	0.588
Average of Evaluations	0.865	0.853	0.603	0.558	0.396	0.753
Standard Deviation	0.076	0.065	0.239	0.164	0.294	0.081
Number of clusters	13	13	16	13	218	13
Number of noises	207	149	2969	0	0	0
Initial parameters	eps=2, min_samples=70	min_cluster_size=70, cluster_selection_epsilon=2	min_samples=70	n_clusters=13	random_state=20	n_clusters=13, assign_labels='discretize'

Identifying the correct number of clusters is crucial for detecting network failures, as it reflects the number of distinct failure regions. DBSCAN and HDBSCAN excel in this aspect, automatically determining the number of clusters with minimal input parameters, such as the neighborhood radius and the minimum number of points to form a cluster. Conversely, OPTICS and Affinity Propagation struggle to ascertain the correct cluster count, even with simpler datasets. K-means and Spectral Clustering, which require predefined cluster numbers, may not be viable when such information is unknown or variable.

Noise detection also plays a significant role in clustering performance. Algorithms like DBSCAN, HDBSCAN, and OPTICS can discern noise points - data points that do not fit into any cluster. These data points can be real noise, not completed records or discrepancy among features in on record, which can be different based on different datasets in various domains. This capability might be interpreted in two different ways: while it is beneficial for identifying outliers, it can complicate result evaluation. For instance, border records that are not clearly part of any cluster, such as those highlighted in Table 03, are classified as noise by these algorithms, despite being labeled otherwise for evaluation purposes. Defining accurate ground-labels based on any discrepancies in the dataset can potentially enhance the accuracy of the results. Thus, careful interpretation of evaluation metrics, considering noise detection, is essential.



Figure 3-: Evaluations for Different Algorithms with different scenarios

In conclusion, DBSCAN emerges as the most adept clustering algorithm for detecting failures in optical networks, capable of accommodating various network scenarios and failure scenarios, and demonstrating resilience to data scrambling. HDBSCAN also shows promise but requires further refinement in performance and noise detection. Algorithms like OPTICS, K-means, Affinity Propagation, and Spectral Clustering are less suitable due to their lower performance, sensitivity to data scrambling, complexity, and reliance on predetermined cluster numbers. A preliminary version of this work was presented at 5th International Conference on High Performance Switching and Routing [YEG23].

3.3 DEFENSIVE WIRESPEED AI

Traditional threat detection methods struggle to cope with the increased complexity, speed, and volume of traffic, highlighting the urgency of automated, intelligent systems able to process network traffic at wire speed without compromising performance.

In this context, we introduce a defensive wirespeed AI methodology that leverages the transformative potential of computer vision techniques for real-time network intrusion detection at the 5G base station (gNB) level. The proposed approach converts raw network packet streams into structured visual matrices, enabling the application of state-of-the-art computer vision architectures for rapid, accurate classification of network behaviors. The SMARTY methodology involves real-time preprocessing of packet features, transforming network flows into image-like representations amenable to CNN processing, thereby ensuring compatibility with stringent latency requirements intrinsic to wirespeed operations. By exploiting publicly available data from the recent 5G-NIDD dataset, which incorporates realistic traffic patterns and diverse attack scenarios (e.g., SYN flood, UDP flood, Slowrate DoS), we benchmark our approach against conventional machine learning classifiers. In the following Figure, the proposed defensive wirespeed AI architecture is shown.

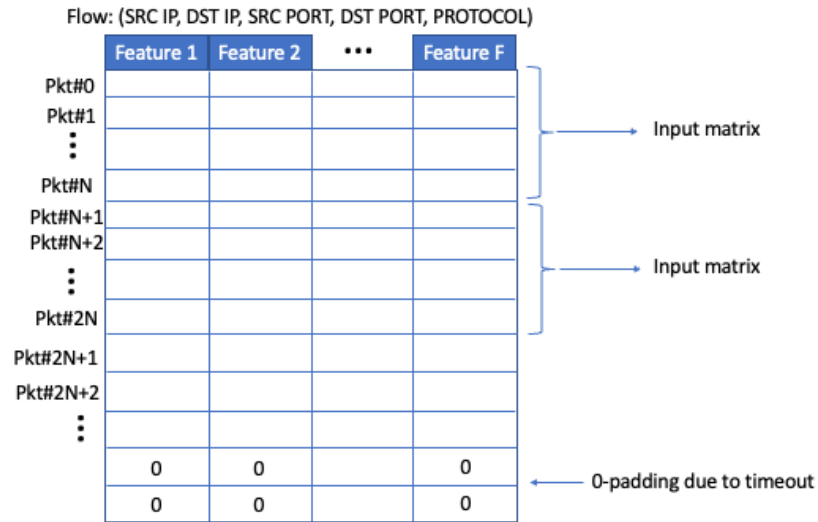


Figure 3:- Proposed methodology to process packets in real-time.

Initially, raw network packets collected from User Plane Functions (UPF) at base stations are transformed in real-time into structured images by extracting relevant packet features and applying normalization and zero-padding.

To evaluate the proposed approach, extensive benchmarking was conducted using the publicly available 5G-NIDD dataset, featuring realistic network conditions and multiple contemporary threat scenarios, including SYN Flood, UDP Flood, HTTP Flood, and Slowrate DoS. The performance of several state-of-the-art computer vision architectures, including ResNet, MobileNet, EfficientNet, DenseNet, Inception, and Xception, were tested against a custom-designed lightweight CNN. The experimental results, reported in the following Table, demonstrate the clear superiority of the customized CNN, achieving exceptional classification accuracy measured through F1-score values (0.99593, 0.99860, and 0.99895 for packet window sizes $N = 10, 50,$ and $100,$ respectively), outperforming all state-of-the-art CNN benchmarks.

Model	N=10	N=50	N=100
Customized CNN	0.99593	0.99860	0.99895
ResNet	0.92266	0.92167	0.79025
MobileNet	0.78308	0.56924	0.64268
EfficientNet	0.81165	0.66327	0.79159
DenseNet	0.88585	0.94067	0.76900
Inception	0.99447	0.87925	0.69674
Xception	0.81352	0.79042	0.76556

Furthermore, prediction latency tests highlight a critical trade-off between model complexity and real-time inference capabilities. As reported in the following Figure, although state-of-the-art models scale well computationally, the custom CNN maintains a favorable balance, significantly reducing latency compared to more complex architectures.

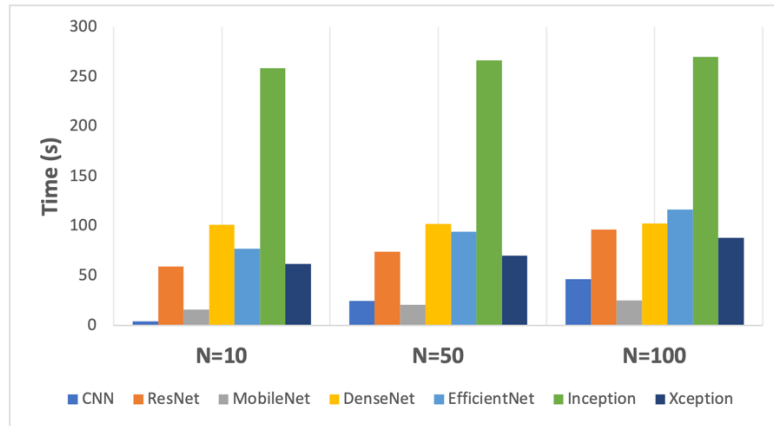


Figure 3-: Serial implementation of flow identification and frequency estimation

Part of this work has been published in [Pao24].

3.4 INTEGRATED DATA SKETCHES FOR TRAFFIC MONITORING AND THREAT DETECTION

Efficient traffic monitoring is crucial to extract features of the traffic that enable anomaly and attack detection. However, processing traffic at wire speed is challenging. Within this context probabilistic data structures that have a much lower cost than traditional exact implementations play an important role. These data structures, also known as data sketches, can provide information with sufficient accuracy at a fraction of the cost of traditional solutions and are widely used in high-speed traffic monitoring. In our case, we are interested in the accurate identification of specific packet flows and the estimation of their number of packets. To identify and track a specific subset of the flows, approximate membership query filters such as the Bloom filter can be used and to count the number of packets, frequency estimation sketches like the Count Min Sketch (CMS) are a good option. However, having to use two independent data structures to identify the flows and count the number of packets is not efficient.

The novel solution proposed and implemented in SMARTY is a new integrated probabilistic data structure that can both identify the specific flows and count their packets. The innovative aspects include:

- Efficiency in Search: Leveraging the inherent speed and agility to facilitate searches for specific elements of the network traffic.
- Precise Estimation: Employing the Count Min Sketch to estimate the frequencies of packets associated with identified flows, ensuring an accurate representation of network dynamics.
- Integration of filtering and estimation: eliminating the need for multiple processing steps, the proposed mechanism allows for the simultaneous execution of search and estimation operations, significantly optimizing computational resources.

The filter used in our proposed integrated data structure is the XOR filter which is generally more memory efficient than a Bloom filter while for frequency estimation, the classical Count Min Sketch is used. The traditional solution is illustrated in the figure below where it can be seen that independent operations and memories are needed.

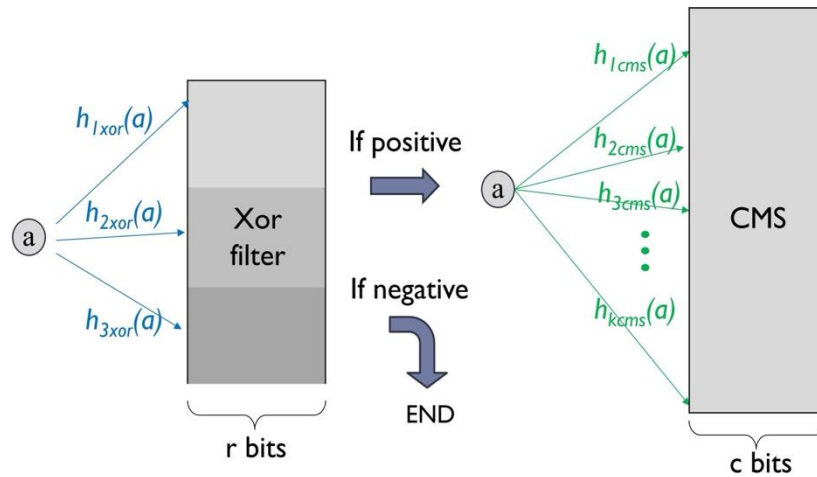


Figure 3-10: Serial implementation of flow identification and frequency estimation

Instead, in the proposed solution illustrated in the figure below, only one memory and a reduced number of memory accesses is needed to perform both operations on the integrated data structure. The integration also brings additional benefits as the CMS can be used to reduce the false positive rate of the filter in some cases.

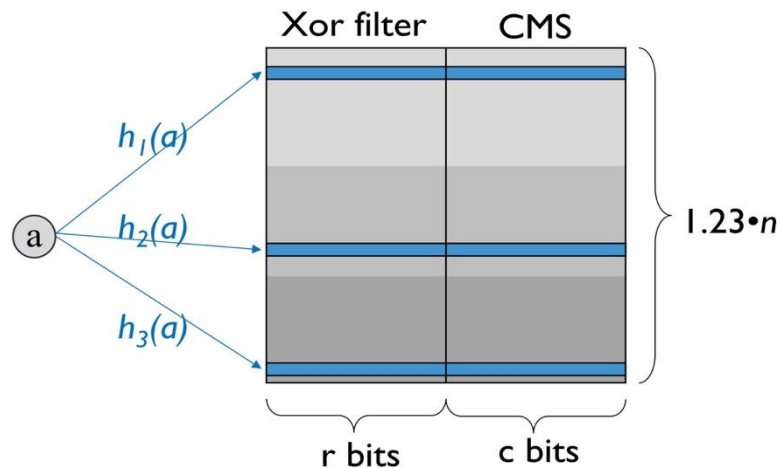


Figure 3-11: Proposed integrated implementation of flow identification and frequency estimation

The proposed scheme has been implemented and compared with the serial implementation in terms of the number of memory accesses, false positive probability and estimation error in the packet counts. It can be observed how the integrated data structure reduces the number of memory accesses for positive queries, the false positive probability and the estimation error for the packet counts of the flows. Similar results were obtained for other traffic distributions.

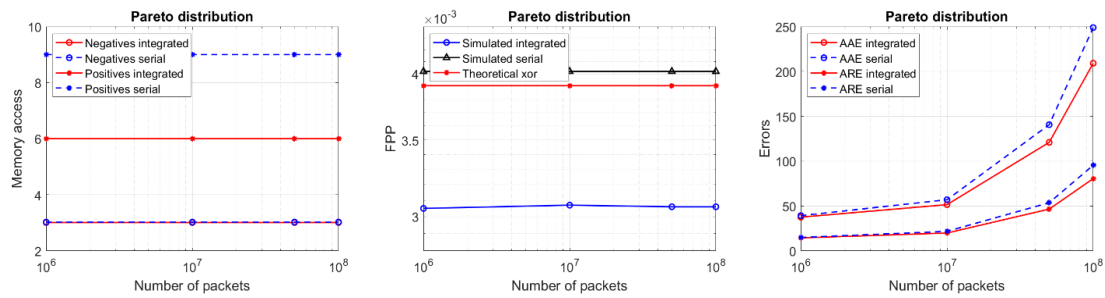


Figure 3-12: Results for traffic with Pareto distribution: (left) Number of memory access in relation the number of packets, (middle) FPP is displayed based on the number of negative elements tested and (right) Average Absolute Error and Average Relative Error.

These results show how integrated data structures can be an efficient solution to achieve wire speed traffic monitoring and feature extraction by processing several features at the same time thus reducing cost and increasing speed.

A preliminary version of this work as presented at 25th International Conference on High Performance Switching and Routing:

R. M. Aguilar, P. Reviriego and D. Larrabeiti, "Combined Filtering and Frequency Estimation with the Integrated Xor Filter and Count-Min Sketch," 2024 IEEE 25th International Conference on High Performance Switching and Routing (HPSR), Pisa, Italy, 2024, pp. 155-160, doi: 10.1109/HPSR62440.2024.10635957.

3.5 TELCO CYBER-THREAT MITIGATION THROUGH WAI

The involvement in Telco Cyber-Threat Mitigation through WAI Methodology actively contributes to the development of a telco-specific solution addressing cyber-threats within deployed telecom systems. Leveraging the Wide Artificial Intelligence (WAI) methodology, it enhances network security, particularly in high-data-rate environments, ensuring robust and adaptive protection against evolving threats. The solution addresses the following key activities:

- **Requirements and Interfaces Definition**

The effort focuses on defining precise requirements, methodologies, and interfaces tailored for telecom networks within the WAI framework. This ensures that security mechanisms align with telco-specific operational constraints and performance expectations.

- **Integration of WAI Methodology in Telecom Networks**

A novel solution addressing cyber-threats in telecom infrastructures is proposed by embedding WAI methodologies into critical network segments/domains. This implementation enhances the detection and mitigation of cyber-attacks while maintaining seamless service quality.

- **Enhancing High-Data-Rate Video Transmissions:** An advanced threat detection mechanism leveraging WAI safeguards high-data-rate video transmissions from cyber-attacks such as Distributed Denial-of-Service (DDoS). This system dynamically identifies, neutralizes, and prevents disruptions in real time, ensuring uninterrupted service quality.
- **AI-Driven Threat Detection and Response:** A cutting-edge AI-powered packet analysis framework detects and blocks malicious traffic with minimal latency. This solution

intelligently distinguishes legitimate from malicious traffic using comprehensive techniques, enhancing network resilience.

- **AI-Based QoS Optimization:** The AI-driven approach dynamically optimizes Quality of Service (QoS) by prioritizing network packets based on real-time analysis. This ensures seamless voice, video, and data transmission while proactively counteracting cyber threats without impacting network performance.

- **Technological Validation and Laboratory Support**

State-of-the-art laboratory infrastructure is suitable for the technological validation of WAI-based security solutions within the telco use case. This controlled environment enables rigorous testing, validation, and optimization of AI-driven cyber-threat mitigation techniques, ensuring readiness for deployment in live networks.

Through this initiative, an innovative AI-based security framework is being considered to enhance telecom network resilience against emerging cyber threats while maintaining optimal service performance. The validation of this approach in the test-bed lab setting further solidifies its applicability and effectiveness in modern telecom infrastructures.

3.6 INTERFACES FOR THE VIRTUALIZED RAN AND WAI

With respect to security attacks and their mitigations, the SOTA (Software Over The Air) of the 5G (and beyond) network architectures identifies the levels of dealing with the situation. For the ORAN based architectures the options are more open to place and execute such actions (whether they are handled in reactive or proactive fashion). It means that in ORAN there are additional interfaces (e.g. A1, E2), which are potentially of use when it comes to deliver “security treatment”. There are various options for the mitigations to be placed, which will eventually determine the speed of reaction:

- Low at the PHY layer (L1) - e.g. dealing with NOMA (non-orthogonal multiple access) and handling suspicious users by adjusting power scaling factors of the selected connections, changing used resources to avoid serving certain UEs, applying precoding to avoid allocating radio resources to certain UEs etc. Here, we are dealing with the cyber-physical systems and such level of protection. It deals with low details of signal handling thus also can be very efficient and fast to assure reaction. Here the signal handling methods are usually engaged.
- At MAC/PDCP layers (L2) - here the mitigations can either be at data plane (IPS, IDS) or in the control plane. In the control plane, what can be monitored and analysed is the signaling messages related to RACH, admission, or any similar “session oriented” indicators that might suggest that there is an ongoing attack. The speed of reaction in non-proprietary systems (i.e. ORAN compliant) will depend on the used level of control plane applications hierarchy used. So basically, the reaction speed will expand as the security mitigations get implemented “higher” into the dApp, xApp, rApp structure (i.e. from RAN stack, towards the management plane). With the latter requiring the longest processing time (>1sec) but also having access to richest set of indicators (i.e. at the SMO level). For the xApps, the reactions times can be at the level of 10-1000ms based on the typical length of the feedback loop. Obviously the dApps are the most capable as their speed of reaction goes well below 10ms, so towards real-time.

- At higher L3 layers (SDAP for the user plane, and RRC for the control plane) the reactions will only be slower and longer as the communication already spans towards the core network. Although in special cases there can be local breakout (LBO) that can be utilized to redirect some traffic to put it into the sanitization stage for load balancing.

In addition, it is worth indicating auxiliary options that recently gain interest in the research and commercial domains:

- Utilizing accelerators to offload and expedite processing of certain stack parts (e.g. L1, L2 and others). Particularly the data plane accelerator solutions (e.g. P4, DPU) can be utilized to introduce fast-track processing, regarding the location of the potential security mitigation "agent" closer to the radio part (e.g. closer to the fronthaul where the baseband signal is processed)
- Utilizing predictive measures (e.g. by means of local or federated, distributed federated learning) which can be acting as IPS solution which augments selected layer from the enumeration provided above. A simple way of dealing with prediction is to locate it at the preferred level of RAN stack (depending on required reaction speed) and introduce additional boost for the decision processes working for the security subsystem.
- Applying AI/ML models (e.g. based on LLMs, GenAI, DRL) which can be utilized, also combined with other models, in a systematic way by introducing a digital twin architecture [1][2][3]. The benefit of digital twin is the fact that based on the models and assuring real-time feedback from the network, these solutions can potentially bring the ultimate speed and capacity for preventive reactions. Such models not only are continuously retrained upon a trigger but can be networked with other models representing various layers of the RAN stack (including both user and control planes). Obviously, the main assumption above is the models themselves to be approached as trustworthy.

In parallel to employing various levels of security mitigation algorithms (dApp, xApp, rApp) a complementing feature is assuring the trust for the execution environment (runtime) which is the prerequisite to assure guarantees of non-intrusion. A combined assurance of a data-in-transit and data-at-rest protection will be enabled by various PQC/QKD solutions that rump-up the level of protection. However the preliminary step in the direction towards systematic inclusion of such solution would start with application level.

3.6.1 Options for WAI in 5G/6G

With respect to the action summarized in the previous section, it is important to recognize that WAI solutions can in 5G (or 6G) be implemented in RAN directly or at the level of compute infrastructure (bare-metal or virtual) that is hosting it.

Component /Layer	Option1 (real-WAI)	Option2 (quasi-WAI)	Option3 (non-WAI)
Algorithm example	ML-DDoS, ML Anti-Jamming ML-Precoder Proactive CU-UP placement with AI/ML in-network	ML-Anti-Jamming	DDoS legacy

Control Algorithm location	dApps / In-network (or hybrid)	dApp / xApp	rApp (SMO)
RAN stack deployment	WAI: COTS + DPU + In-Network P4/DOCA + Cognitive Plane	Accelerated: COTS + DPU + Cognitive Plane	Legacy: COTS HW + Cognitive Plane
Hypervisor role	ProActive (WAI functions)	Active (w/AI models)	Passive (Runtime only)
ORAN Interfaces	E3, FH	E2	A1, E2, O1
Transport network	Transport domain's SDN + P4/DOCA	P4/DOCA + LBO	Legacy
Interacting SWARM entities	In-Network (DFE, AI/ML) and 5G/6G RIC	5G/6G RIC as SWARM entity	RIC <> 3 rd party SDN
Mitigation speed	<10ms	10-1000ms	>>1s
Model location	RT RIC + In-Network	Hybrid (near/RT RIC)	Hybrid (near/non-RT RIC)
Model Framework	Digital Twin (e.g. AI/ML)	ORAN	SON

The above table indicates the multiple layers of a 5G/B5G network infrastructure and proposes three (potential) options, highlighting the “level of compliance” with the SWARM/WAI requirements. The table should be read across the whole layers and the feasible solutions picked from the three columns. The closer the solution for a given component is to Option1 the higher capability of WAI/DFE can be provided for such deployment. This way, the optimal solution should be utilized.

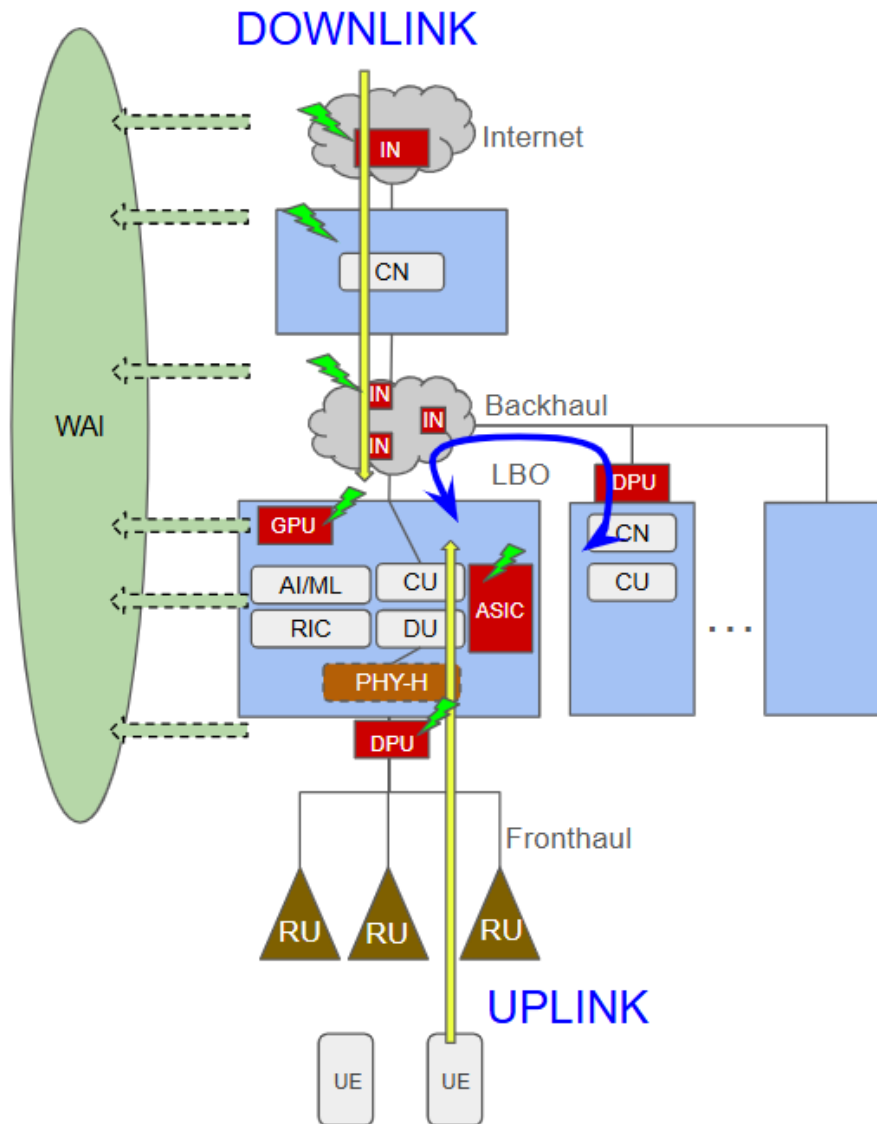


Figure 3-13 High level view on the points of WAI interconnection in 5G/6G

Regarding the interfaces towards WAI for the mobile networks (5G,6G) the figure presents the high level view on the main capabilities. Two main data flows are downlink and uplink, they identify capabilities for the security detection points at various layers and interfaces of the network. Considering the donwlink, the locations (besides the WAN and LAN networks) that interconnect the mobile backhaul with the internet related with the core network (e.g. UPF) are suitable to perform “wire-speed” reaction in the data plane. Here the integration with WAI can easily utilize the data-plane soultion for DFE that are compliant with wired networking architecture for WAI. Preventing the side-effects of attacks on user-traffic here is relatively easy. After crossing the core-network (CN) line, we start dealing with GTP tunnels and slices. Here slices can also be utilized to manage data-plane traffic in order to assure some isolation in case of attacks – especially when slices are combined with security features. The good reference for secure slices design can be found in the research projects NETWORK (Horizon EU), TSORAN (Eureka Celtic) and FNS (national Dutch project).

From the perspective of a link protection (and attacks that can be mitigated with it) it is worth mentioning that especially links between core networks (e.g. roaming case), core and CU (local deployment), CU-DU (midhaul) – they should be considered when guaranteeing protection with

the post-quantum solutions (PQC, QKD). Although PQC can also be deployed in the UE terminals, which can be attractive solutions for mobile operators at early stage of introducing quantum protection. However DDoS attacks for example may make the situation even worse with such additional protections, as they may incur additional processing costs (e.g. key exchange, generation).

On the opposite side, when attacks are generated from the radio interface (Uu) the proper configuration of the 5G/6G RAN will determine the “wire-speed” capabilities. In case of deployments where high-phy or elements of DU are processed on the DPU within edge server, this is the main point for traffic offloading as at this stage the load of connections e.g. under DDoS attacks may be smoothly offloaded from the main CPU of the higher protocol stacks. In similar fashion, “already created” slices and user data-plane connections can be “wire-speed” protected by alternative offloading strategies. But in the DDoS case the main challenge comes from the level of user session creation (admission control requests, RRC requests). Malicious traffic requests at this stage can be blocked with smart algorithms (like xApp based threshold based or data-driven learning algorithms). Alternatively depending on the smart algorithms implemented it is also feasible to assume the traffic will be steered/offloaded to other servers in order to better identify characteristics of the attack (e.g. by intelligent CU-UP scaling).

In case of attacks directed to the “local connections” e.g. in case of intranet/extranet traffic for the employees under LBO, secure slices can provide necessary protection. Here the traffic will be by default offloaded to the “local core” and may largely be mitigated without involving the main core (CN), due to offloading at the CU level (routing of PDUs to the local data network with mini-core).

In order to expose identified attack vectors and traces algorithms at RAN can export attack signatures (and related data or model parameters) based on the legacy telemetry exporters or dedicated (customized) protocols. The further details will be specified in other deliverables of the SMARTY.

4. DECLARATIVE COOPERATION IN CLOUD COMPUTING ENVIRONMENTS

Cooperation is a key factor in tackling complex problems that frequently arise in the real world. A single entity may not possess the necessary knowledge or tools to solve these challenges efficiently. In such cases, cooperation naturally emerges among different parties that share a common goal. However, even when partners agree to collaborate, they may be reluctant to share their intellectual property (IP) with others. To address this challenge, we explore, develop, and evaluate a framework for establishing confidential workflows involving multiple stakeholders. This approach allows stakeholders to protect their intellectual property while still working together to achieve a common objective.

Given this challenge, in the sections below we detail each partner's contribution to tackle this problem. SCO focuses on its SCONE solution which provides a declarative framework for setting up confidential workflows across multiple stakeholders. TUM provides confidential computing compliant environments for various processors and accelerators. IQU offers guidance on the use of Software Defined Perimeters (SDP) in the context of Confidential Computing. COGN evaluates a framework within its AI services to support cooperation among multiple stakeholders.

4.1 CONFIDENTIAL WORKFLOWS

The declarative cooperation focuses on enabling multiple partners to work together on a common task despite potential competitors in different projects. The focus is on ensuring that each partner can protect their own intellectual property (IP) on a technical instead of a legal or organizational solution. That is, each partner can declare what they want to protect and verify that in the context of a given computation, the partner's IP is always secured.

In this context, we view multi-stakeholder computations as workflows. Each component of a workflow belongs to one stakeholder. Each component is a confidential service, i.e., a service running in an enclave, or an encrypted volume, i.e., a set of files that can only be read by authorized, confidential services. SCONE, our confidential computing platform, enables the connection of confidential services and encrypted volumes into one workflow.

Take as an example the confidential workflow depicted in the figure below. All stakeholders work toward a joint goal and yet, ensure that their IPs remain confidential. To achieve that, in confidential workflows, stakeholders protect their services with the help of SCONE policies. When operating services of other partners as part of a workflow, the SCONE policy can prevent the cloud/service/application provider from inspecting a service. On the other hand, a policy can prevent a service instance from communicating with other services outside a given workflow. In this way, one can exclude that a service leaks the data that it processes. Moreover, a policy defines which secrets a service can access and allow the import and export of secrets from and to other SCONE policies. This sharing of secrets enables the secure communication of services.

4.1.1 Example: AI Workflow

Let us consider the example depicted in the figure below. In the context of machine learning, a data provider might store data in a confidential database. The data provider joins a confidential workflow but decides to grant access to its data to the workflow only after verifying the workflow policy. During a machine learning (ML) algorithm training or an inference phase, the

confidential workflow policy grants its access only to the data provider database; no other input is allowed. To ensure that the service running the ML algorithm cannot forward the data to other entities, the workflow firewalls it. To ensure the data provider stays in the loop, any changes in the workflow policy would require an opt-in by the database owner. In this way, the data owner can protect its data.

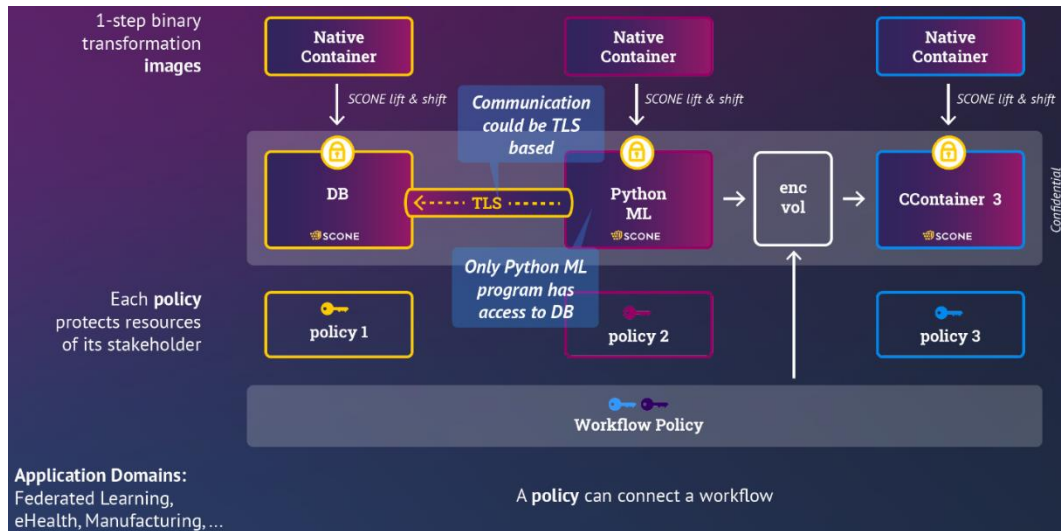


Figure 4-1: Multi-stakeholder computations as workflows.

The owner of the machine learning algorithm can protect the IP of its Python code by encrypting the Python image. Only the machine learning algorithm itself can see the unencrypted Python code. The generated machine learning model might be written to a Kubernetes volume encrypted by SCONE.

4.1.2 Technical Problem Description

When defining a workflow, we need to ensure that all stakeholders can inspect the declaration of the workflow. The workflow running in a system must match the workflow declaration that was shared with the stakeholders. To address potential Time-Of-Check-To-Time-Of-Use (TOCTTOU) attacks, we provide governance for the declarations (see below). Another issue is that the code of the confidential services might need to be protected. Hence, we have an issue that the code of services cannot be inspected to ensure that it does not share data with outside components. To address this, all components are running in a sandbox that is controlled by the workflow declaration.

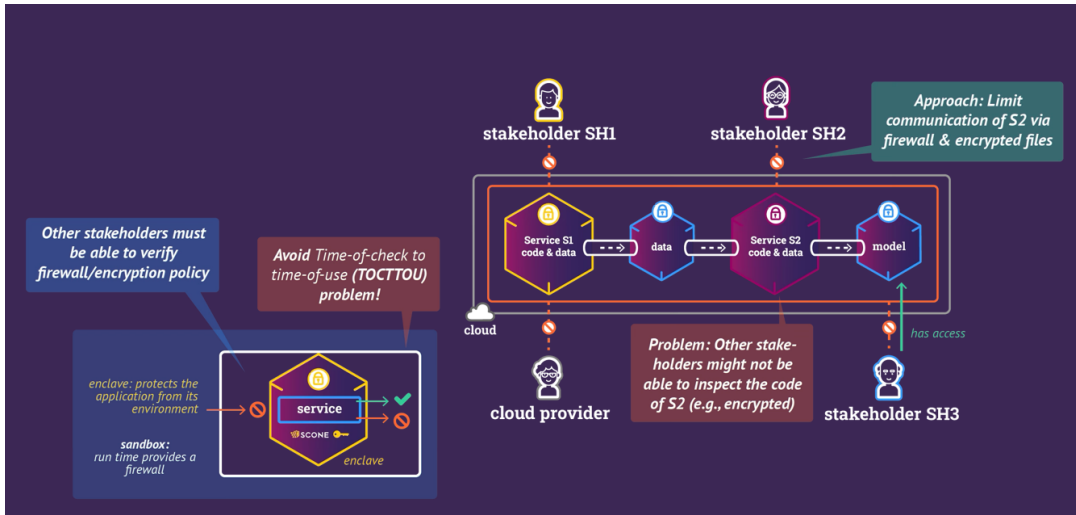


Figure 4-2: Provision of governance for the declarations of workflows to address potential Time-Of-Check-To-Time-Of-Use (TOCTTOU) attacks.

4.1.3 Example: A Simple Declarative Workflow

Let us now look at a simplified example that shows the two problems stated before. In this example, we use two containers and two volumes. The service deployed by *container 1* writes a file in *volume 1*. It then terminates and *container 2* is started. The service deployed in *container 2* will read the data from *volume 1*, perform some computation in the input and then write the data to *volume 2*.

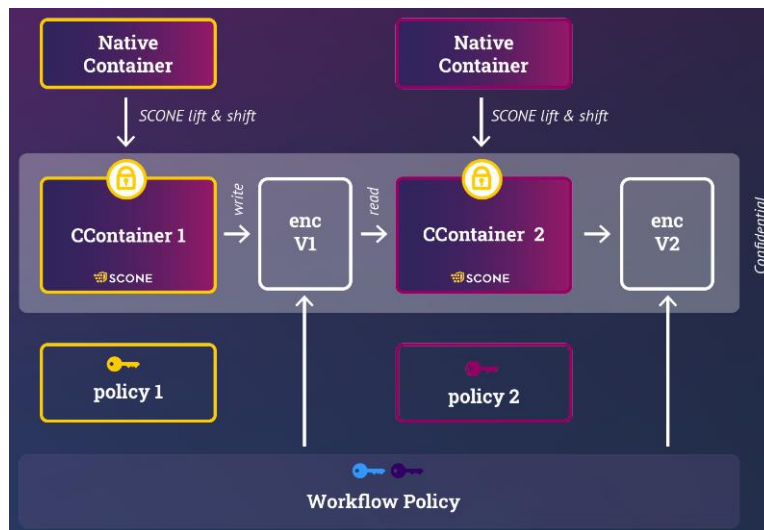


Figure 4-3: A Simple Declarative Workflow for two native containers.

The volumes are created as ordinary Kubernetes volumes which are persistent meaning that the files will live until the volume is explicitly deleted. With the SCONE policy we can automatically encrypt these volumes. Technically, the SCONE runtime being part of the services running inside of enclave encrypts all data written to these volumes. Of course, when the service reads a file, the SCONE runtime automatically decrypts the data.

Note that an operating system or Kubernetes can also encrypt the data at rest. However, in this case, the operating system or Kubernetes could see the data in plain text. In our trust model, we

need to ensure that only authorized services running inside of enclaves can access these volumes as plain text files.

Let us have a look at how one can define such an encrypted volume and how to share it with other policies. Consider the snippet below. In the sketched example, we define two encrypted volumes *V1* and *V2*. These encrypted volumes are generated by the workflow policy - which is a standard SCONE security policy. Each policy has a unique name. Similar to Kubernetes, SCONE supports namespaces for policies and each policy is a namespace.

```
1. name: $workflownamespace
2. version: "0.3"
3.
4. access_policy:
5.   read:
6.     - ANY
7.   update:
8.     - CREATOR
9.   create_sessions:
10.    - ANY
11.
12. volumes:
13.   - name: V1
14.     export:
15.       - session: $workflownamespace/policy2
16.       - session: $workflownamespace/policy1
17.   - name: V2
18.     export:
19.       - session: $workflownamespace/policy2
```

A policy name can be viewed like a path of a file and the directories in this path are namespaces. In this example, we use a variable called *workflownamespace* to denote this path (see Line 1). This variable ensures that multiple people can run this example in different namespaces. In practice, one might want to use the same namespaces in Kubernetes and in SCONE.

Each policy has an access control section (Line 4). This section defines who can read the policy, who can change the policy, and who can create sessions in this namespace. In this example, we do not define any explicit secrets as part of the workflow policy. Hence, we grant read access to everybody (Lines 5 and 6). This is denoted by the term *ANY*.

This workflow policy can only be updated by the creator of this policy (Lines 7 and 8). The creator is identified by a public key. To update the policy, one needs to know the private key belonging to this public key. To simplify matters, SCONE CAS determines the public key of the creator and stores it as part of the policy. In this way, one can just store the term creator as part of the access policy.

Moreover, one can define other policies and hence, namespaces in the context of a policy (as they can be nested). This is allowed on Lines 9 and 10. In this example, we create *policy1* and *policy2*, which we use on Lines 15, 16 and 19, but omit for the sake of explainability, in the namespace of the workflow policy. Each policy must specify a version number. This is the version of the SCONE policy language.

The last section of the workflow policy defines volumes *V1* and *V2* (Lines 12 through 19). By default, such a volume can only be accessed by service specified in the same policy. In this workflow policy, we do not specify any service. Hence, no service could actually access these two volumes. However, we export the volumes to other policies. *V1* is exported to *policy1* and *policy2*. These two policies can then grant access to *V1* to services running in the context of *policy1* and *policy2*. On the other hand, access to *V2* is only granted to *policy2*. This means that the services running in the context of *policy1* can access *V1* but **not** *V2*.

4.2 SECURE ACCELERATOR INTERFACE

Physical attacks are those launched by untrusted hardware administrators with physical access to the platform, typically cloud provider servers. We consider two subcategories: (1) insertion of untrusted devices, which may offer weaker security guarantees and allow attackers to manipulate the victim's program state, and (2) interconnect eavesdropping and tampering, compromising the confidentiality and integrity of data exchanged between accelerators and system components. To mitigate these threats as well as non-physical attacks, we attach an Attested Interface Unit (AIU), introduced in Subsection 2.2, to all compute units in a system.

The AIU is a minimal hardware extension, built on the concept of DTUs. It is accelerator agnostic, thus provides a uniform abstraction over a diverse set of accelerators. AIUs extend the typical message passing capabilities of a DTU, with secure capabilities:

- Silicon Root-of-Trust: Built-in cryptographic material, that allows verification of the AIU
- Attestation module: Allows the OS to bootstrap trust into the AIU;
- Authenticated access control: Only the trusted kernel can establish secure channels between AIUs, and by extension the applications using them.

4.3 ADVANCED TESTING AND MONITORING TOOLS

Artificial intelligence (AI) has become a critical component in healthcare and financial services, revolutionizing how data is processed, analyzed, and utilized for decision-making. To ensure that AI-driven solutions operate efficiently, securely, and in compliance with industry standards, it is essential to evaluate the underlying framework that supports these technologies.

COGN is committed to assessing the performance, interoperability, and security of its AI framework when applied to complex environments involving multiple stakeholders. This evaluation is particularly crucial in sectors such as healthcare and finance, where reliability, confidentiality, and adaptability are key to ensuring operational success.

The assessment process employs a structured approach, utilizing industry-leading tools to validate the framework across multiple dimensions:

1. Integration Testing – Ensures seamless interoperability and efficiency in real AI environments through API testing, load testing, and stability checks.
2. Performance Analysis – Measures system responsiveness, scalability, and security in handling confidential data and cooperative workflows.
3. Simulations and Use Cases – Tests framework adaptability and robustness in controlled environments with representative data sets.

4. Security Assessments – Validates compliance with privacy and confidentiality standards through penetration testing and vulnerability assessments.

By leveraging advanced testing and monitoring tools such as Postman, Apache JMeter, Grafana, Prometheus, OWASP ZAP, and others, COGN ensures that its AI framework meets the highest standards of performance, security, and interoperability.

This structured evaluation allows for comprehensive validation in AI-driven healthcare and financial applications, ultimately contributing to safer, more efficient, and reliable technological solutions in these critical industries.

The framework evaluation can be conducted using specific tools for each use case. This approach ensures a comprehensive validation in terms of interoperability, performance, adaptability, and security in AI-driven healthcare and financial environments.

Integration Testing

Evaluate the interoperability and efficiency of the framework in real AI environments.

Tools:

- Postman → To test API integrations with healthcare and financial systems.
- Apache JMeter → To assess load capacity and framework behavior under real-world traffic.
- K6 → To simulate multiple users and identify stability issues in framework integration.

Integration Testing: API Response Time Distribution

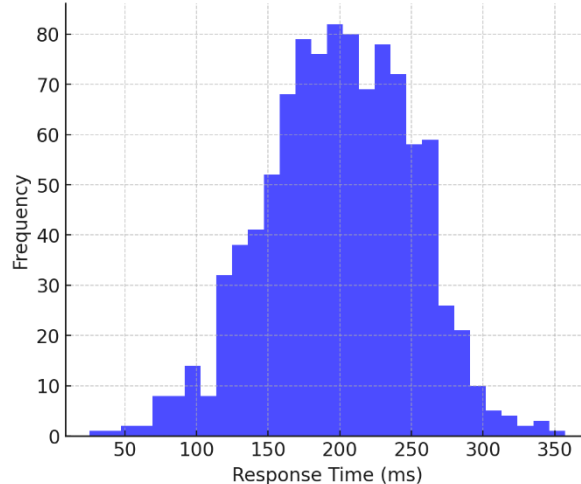


Figure 4-4: API responses in Integration testing

Use Case Example:

- Implement the framework in an AI-based hospital management system.
- Use Postman to test API interoperability with other systems.
- Run JMeter to simulate multiple requests and measure performance impact.

Performance Analysis

Measure latency, scalability, and security in confidential data management and declarative cooperation.

Tools:

- Grafana + Prometheus → For real-time monitoring of latency and resource consumption.
- New Relic → For performance analysis and optimization in cloud environments.
- Datadog → For event observability, traceability, and real-time response.

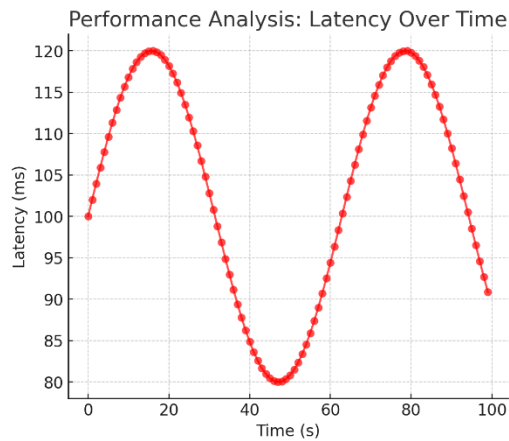


Figure 4-5: Latency over time in performance analysis

Use Case Example:

- Deploy the framework in a financial data management service.
- Use Prometheus to capture latency metrics in service communication.
- Configure Grafana to visualize performance in dynamic dashboards.
- Utilize New Relic to identify bottlenecks and enhance framework efficiency.

Simulations and Use Cases

Assess the adaptability and robustness of the framework in controlled environments with representative data.

Tools:

- Apache NiFi → To manage and process data flows in controlled tests.
- Splunk → To analyze framework logs and events in simulated scenarios.
- ELK Stack (Elasticsearch, Logstash, Kibana) → To visualize framework behavior across different test environments.

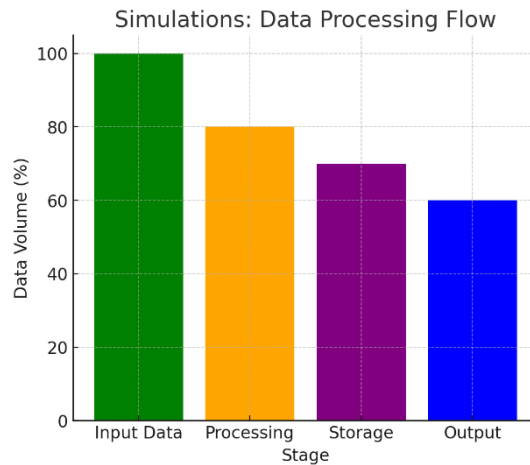


Figure 4-6: Data volume for each processing flow.

Use Case Example:

- Simulate electronic health record management with multiple data sources in a healthcare setting.
- Use Apache NiFi to model data flows from various medical entities.
- Analyze logs with Splunk to detect potential failures or improvements in data structuring.
- Visualize usage metrics and efficiency with Kibana.

Security Assessments

Ensure the framework meets privacy and confidentiality requirements in cloud computing environments.

Tools:

- OWASP ZAP → For penetration testing and vulnerability detection in the framework.
- SonarQube → For static code analysis and identification of potential security flaws.
- Kali Linux (Metasploit, Nmap) → For security audits and intrusion testing.

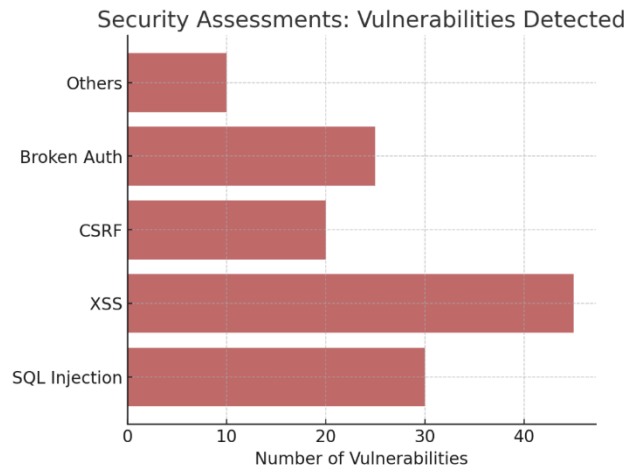


Figure 4-7: Number and type of vulnerabilities detected.

Use Case Example:

- Deploy the framework in a financial transaction processing service.
- Run OWASP ZAP to identify potential security breaches in data communication.
- Use SonarQube to review the source code and detect vulnerabilities before deployment.
- Conduct audits with Kali Linux (Metasploit and Nmap) to evaluate framework resilience against external attacks.

5. DYNAMIC NETWORK SWARM

Work under this section refers to tools and solutions for the dynamic formation of network swarms. This entails the creation of a highly programmable edge network forwarding plane using programmable hardware. In this section, we report

- a Hierarchical-Hybrid Synchronization of Swarn Networks model is proposed, built on a combination of hierarchical and cluster-based architectures, specifically implemented using ONOS SDN controllers
- semantic-aware serverless deployment of network swarms using programmable network-based components namely a P4 load balancer designed to support multiple traffic distribution methods and take autonomous decisions; a real-time telemetry data tool for edge node resource utilization (CPU, GPU, memory) to optimize workload distribution; and semantic-aware traffic steering
- Swarmification in RAN for the extension of its resilience capabilities
- Network Swarm at Edge Gateways Equipped with DPUs
- Device-flow-Graph-based policy programming model to enable SLA-compliant hardware acceleration
- An information Framework for Improving Decision making process

5.1 HIERARCHICAL-HYBRID SYNCHRONIZATION OF SWARN NETWORKS

In edge computing scenarios, maintaining an up-to-date and synchronized view of network resources is crucial for efficient operation and rapid response to changes and threats. This subsection details the implementation of a Hierarchical-Hybrid Swarm Synchronization methodology based on a resilient, hierarchical SDN architecture leveraging a hybrid synchronization approach. Inspired by recent advancements in large-scale SDN synchronization techniques, the proposed system implements a two-tier hierarchical model to gather and aggregate telemetry data from multiple distributed edge platforms. Each edge platform is locally managed by dedicated SDN controller clusters, while a parent controller coordinates cross-platform resource allocation decisions, thus enabling a dynamic swarm-based approach to resource management.

This synchronization model (reported in the following Figure) is built on a combination of hierarchical and cluster-based architectures, specifically implemented using ONOS SDN controllers. Telemetry events from edge platforms (child clusters) are proactively collected, transformed, and sent via efficient, low-latency gRPC communication channels to the centralized swarm manager (parent cluster), maintaining full consistency through robust event ordering and delivery mechanisms.

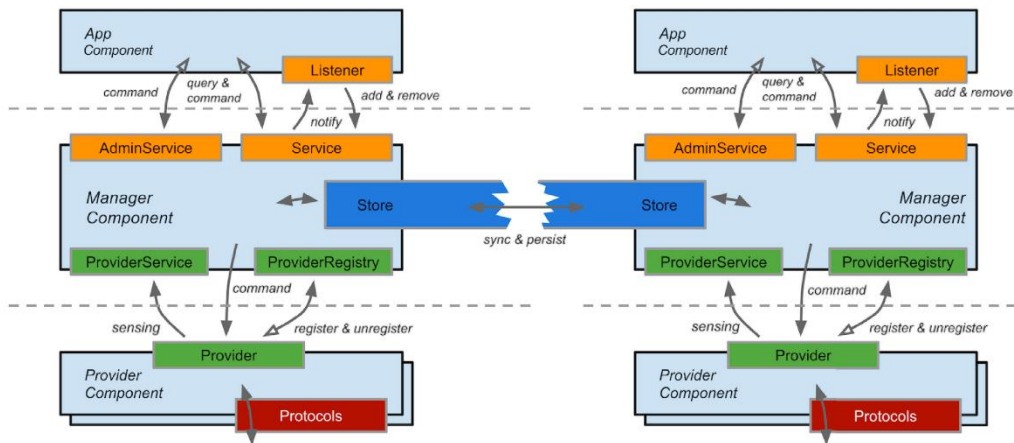


Figure 5-1: synchronization model built on a combination of hierarchical and cluster-based architectures and implemented using ONOS SDN controllers

5.1.1 Proposed Approach and Implementation

5.1.1.1 Data Collection and Preprocessing

Telemetry data is collected at ingress/egress points using Kafka-based monitoring components, structured into event-based representations compatible with ONOS cluster synchronization models. Each telemetry event (network status, load metrics, device health, etc.) is encapsulated into structured Protobuf objects for efficient transmission and low latency processing.

5.1.1.2 Hybrid-Hierarchical Synchronization

The collected telemetry data undergoes synchronization using a hybrid-hierarchical approach. Child controllers aggregate data within their cluster, maintaining consistency and resilience through Atomix-based distributed storage. The data is then securely and efficiently propagated to the parent controller via gRPC, enabling the parent controller to dynamically and accurately assess overall network status and available resources at all times.

5.1.1.3 Swarm Resource Profiling and Adaptation

The aggregated telemetry data at the parent controller is processed to generate real-time resource profiles for each edge platform. These profiles enable dynamic, automated decision-making, optimizing resource allocation, proactively mitigating threats, and dynamically reconfiguring network resources.

5.1.2 Results

Experimental evaluations demonstrate the efficiency and resiliency of the proposed synchronization model. Real-time latency measurements for event propagation consistently remain below 10 ms under normal operation and around 40-60 ms under high-load conditions.

Scenario	Average Latency	Max Latency	Failure Recovery Time
Normal Load	< 10 ms	~ 20 ms	< 1 sec
High Load (burst events)	~ 40 ms	~ 60 ms	< 2 sec

Failure tests confirm the robustness of the synchronization approach, with automatic recovery from instance failures, ensuring continuous resource profiling without loss of synchronization accuracy or consistency.

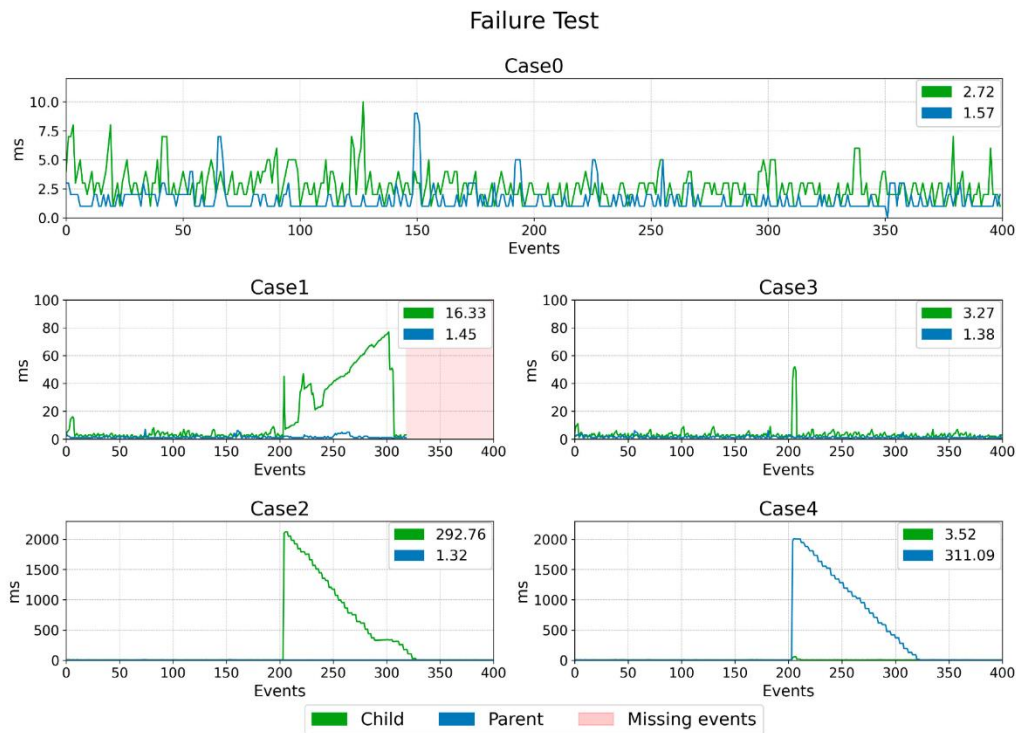


Figure 5-2: Automatic recovery from instance failures

Part of this work has been published in:

Pacini, A., Scano, D., Sgambelluri, A., Valcarenghi, L., & Giorgetti, A. (2025). Hybrid-Hierarchical Synchronization for Resilient Large-Scale SDN Architectures. IEEE Access.

5.2 SEMANTIC-AWARE SERVERLESS DEPLOYMENT WITH P4 LOAD BALANCING

Serverless computing, particularly in the form of Function-as-a-Service (FaaS) is a cloud-native technology that offers an application development model for developing and operating distributed applications. Its essence is to separate operational tasks from the development process, allowing developers to focus exclusively on writing code. Similarly to microservices, this solution is also based on the decomposition of applications, but here we usually work with even smaller units. The elementary unit is the function, which – similarly to functions found in

programming languages – has an input, an output, and is used to implement a given task. FaaS platforms handle code execution and scaling tasks and can be beneficial for fine-grained resource usage. The fact that the technology provides a flexible and scalable way to execute stateless functions on demand makes it well-suited for dynamic edge environments, and, moving out of the cloud, FaaS has gained significant traction at the edge as well. At the same time, Software-Defined Networking (SDN) and programmable networks using P4 have demonstrated their ability to enhance network flexibility by enabling real-time, fine-grained traffic steering. Recent research [Pel23] suggests that integrating FaaS and P4-based programmable networking can significantly improve service adaptability. Additionally, the emergence of semantic communication introduces a transformative approach to optimizing data transmission. Unlike traditional communication paradigms that focus on raw data transfer, semantic communication emphasizes the meaning and relevance of transmitted information, thereby reducing redundant data exchange and enhancing efficiency. When applied to serverless edge computing, semantic-aware processing can optimize function execution, minimize network congestion, and enhance decision-making for traffic steering.

5.2.1 Programmable network-based components

Building upon these technologies, in SMARTY we propose a novel approach of semantic-aware serverless deployment with P4 load balancing that integrates serverless computing, P4-driven load balancing, and semantic communication to enhance edge computing performance in resource-constrained environments. The system will prioritize the transmission of high-value semantic information while reducing unnecessary data flow between edge nodes. The P4 Load Balancer (P4LB) will dynamically distribute computation tasks across multiple edge nodes based on real-time resource constraints and semantic importance.

In this deliverable, we report on the design of the key enabling components:

- A P4-based load balancer designed to support multiple traffic distribution methods and take autonomous decisions according to pre-defined set of options.
- A real-time telemetry data on edge node resource utilization (CPU, GPU, memory) to optimize workload distribution.
- Incorporate semantic-aware traffic steering, ensuring that critical functions receive prioritized execution based on contextual relevance.

In the following we report an extract of the P4 program (preliminary version) where a register is used to store the status of the least loaded edge node, which determines where traffic should be directed. The program assumes: (i) a register array (*edge_load_register*) stores CPU load values for multiple edge nodes; (ii) The least loaded node's ID is stored in another register (*least_loaded_edge*); (iii) The P4 pipeline processes incoming traffic and directs it based on the least loaded node.

Furthermore, semantic information is included to handle object-specific routing in addition to load balancing. To this purpose, each packet carries an object type (e.g., vehicle or person). This allows the P4 switch to decide which edge node can process that specific object. The switch only considers edge nodes that can process the given object and among those, it selects the least loaded edge.

```
#define NUM_EDGE_NODES 4
#define OBJECT_TYPE PERSON 1
```

```

#define OBJECT_TYPE_VEHICLE 2

// Register to store CPU load of each edge node
register<bit<8>>(NUM_EDGE_NODES) edge_load_register;

// Registers to store least loaded edge node for each object type
register<bit<8>>(1) least_loaded_person;
register<bit<8>>(1) least_loaded_vehicle;

// Registers to store processing capabilities (1 = can process, 0 = cannot)
register<bit<1>>(NUM_EDGE_NODES) edge_person_capability;
register<bit<1>>(NUM_EDGE_NODES) edge_vehicle_capability;

// Metadata structure to carry selected edge node
struct metadata_t {
    bit<8> selected_edge_id;
    bit<8> object_type;
}

// Edge Resource Monitoring (ERM) header (CPU Load Information)
header ERM_Header_t {
    bit<8> edge_id;
    bit<8> cpu_load;
}

// Object Type Header (Indicates the object being processed)
header OBJ_Header_t {
    bit<8> object_type; // 1 = Person, 2 = Vehicle
}

// Parser to extract ERM and Object Type Headers
parser MyParser(packet_in pkt, out ERM_Header_t erm_hdr, out OBJ_Header_t obj_hdr) {
    state start {
        pkt.extract(erm_hdr);
        pkt.extract(obj_hdr);
        transition accept;
    }
}

// Action to update CPU load for an edge node
action update_edge_load() {
    edge_load_register.write(hdr.erm_hdr.edge_id, hdr.erm_hdr.cpu_load);
}

// Action to select least loaded edge node for persons
action select_least_loaded_person() {
    bit<8> load_0; bit<1> can_process_0;
    bit<8> load_1; bit<1> can_process_1;

    edge_load_register.read(load_0, 0);
    edge_person_capability.read(can_process_0, 0);
    edge_load_register.read(load_1, 1);
    edge_person_capability.read(can_process_1, 1);

    if (can_process_0 == 1 && can_process_1 == 1) {
        if (load_0 <= load_1) {
            least_loaded_person.write(0, 0);
        } else {
            least_loaded_person.write(0, 1);
        }
    } else if (can_process_0 == 1) {
        least_loaded_person.write(0, 0);
    } else if (can_process_1 == 1) {
        least_loaded_person.write(0, 1);
    }
}

// Action to select least loaded edge node for vehicles
action select_least_loaded_vehicle() {
    bit<8> load_2; bit<1> can_process_2;
    bit<8> load_3; bit<1> can_process_3;

    edge_load_register.read(load_2, 2);
    edge_vehicle_capability.read(can_process_2, 2);
    edge_load_register.read(load_3, 3);
    edge_vehicle_capability.read(can_process_3, 3);
}

```

```

    if (can_process_2 == 1 && can_process_3 == 1) {
        if (load_2 <= load_3) {
            least_loaded_person.write(0, 2);
        } else {
            least_loaded_person.write(0, 3);
        }
    }

    } else if (can_process_2 == 1) {
        least_loaded_vehicle.write(0, 2);
    } else if (can_process_3 == 1) {
        least_loaded_vehicle.write(0, 3);
    }
}

// Table to process ERM messages and update CPU load
table edge_monitoring {
    key = {
        erm_hdr.edge_id: exact;
    }
    actions = {
        update_edge_load;
        select_least_loaded_person;
        select_least_loaded_vehicle;
    }
}

// Table to forward packets based on object type and least loaded edge
table traffic_steering {
    actions = {
        send_to_best_edge;
    }
    default_action = send_to_best_edge();
}

// Action to forward traffic to the selected edge node
action send_to_best_edge() {
    bit<8> target_edge;
    bit<8> object_type;

    metadata_t.selected_edge_id = 255; // Default: No valid edge found
    metadata_t.object_type = object_type;

    if (object_type == OBJECT_TYPE_PERSON) {
        least_loaded_person.read(target_edge, 0);
    } else if (object_type == OBJECT_TYPE_VEHICLE) {
        least_loaded_vehicle.read(target_edge, 0);
    }

    // modify_field(metadata.selected_edge_id, target_edge);
    metadata.selected_edge_id = target_edge;
}

```

The system operates by using Edge Resource Monitoring (ERM) [Pel23] messages to update the CPU load for each edge node. Each node is assigned a predefined capability, meaning it can process only specific types of objects, such as vehicles or people. The system dynamically updates CPU loads for each edge node through ERM messages. When a packet arrives, its object type is identified, and the network selects the least loaded edge node capable of processing that specific object. For example, if the packet contains a person, it is directed to the least loaded edge node that supports people. Similarly, if the packet contains a vehicle, it is forwarded to the least loaded node that can process vehicles.

5.2.2 Components related to the serverless engine

While the traits of FaaS are desirable in many areas, FaaS platforms are not suitable for running all types of applications. Our focus area, the latency-sensitive applications where response time must meet strict requirements and function composition/function fusion where deployed

artifacts can contain multiple application components, are not among those supported out-of-the-box. The application described by *Use Case 5 Secure automotive application: edge-based collective perception for cooperative ADAS*, is such an application. In this case, it is also important to utilize edge nodes, where data processing is performed at locations closer to the users. This significantly reduces the delay between requests and responses compared to transmitting the data to a central cloud location.

TKI's activity in this work revolves around extending an open-source FaaS framework with capabilities for running such applications using only the FaaS platform capabilities and integrating it with network telemetry to provide even lower latency for rerouting traffic between equal instances of the same FaaS function deployed to various edge nodes depending on diverse low-level metrics or semantic information joining to CNIT's corresponding contribution described in the previous subsection.

In this deliverable, we report on the exploration, design and initial implementation of the following key enablers of the serverless component of the joint work:

- identifying a base open-source FaaS platform to extend,
- design and initial implementation of the novel function fusion and edge execution capabilities to the platform incorporating invocation support at the level of application components with invocation routing,
- integration with an open-source observability layer that is able to collect and store compute, FaaS function and application component-level metrics, and
- initial exploration and design for collecting network-level metrics and sharing semantic information with a load-balancer or FaaS function invocation router component implemented via network programming (see P4LB, P4 Load Balancer in Section 5.2.1).

After comparison with other options, due to its wide-spread use we selected Kubernetes as an underlying platform and identified Knative as a FaaS platform above it due to it being open and under active development. Knative consists of the following components utilized in our current activity:

- Knative Serving: It helps in deploying Kubernetes services through Custom Resource Definitions (CRDs). The most important of them is the Service resource, which ensures the proper configuration of our services and also manages their lifecycle.
- Knative Eventing: It contains components that support an event-driven architecture, such as event sources, channels, and allows our applications to subscribe to events.
- Knative Functions: It differs from the previous ones, as it does not contain resources to be deployed in the cluster. The Functions component is a command-line tool that provides a convenient programming model for developing FaaS applications. It also offers function templates for many popular programming languages. The templates consist of, among others, a handler file and a configuration file. The former serves as the entry point for the application and contains the business logic of the function, while the latter allows to configure the environment and deployment parameters of our application (replica number, scaling, allocated resources, etc.). The deploy command creates a container image from the template and deploys it as a Knative Service using the configurations.

As for the observability aspects, we opted to integrate our application component-level signals (required for proper monitoring of fused application components) with the OpenTelemetry (OT)

[Ope25] observability framework which is able to unify various signal types and make their collection and generation independent of technologies. In our first implementation we mainly focus on the management of metrics and traces, however, our preliminary design also considers log signals. Metrics are numerical data describing the performance of the application, e.g. CPU and memory usage or requests served per second. They are usually queried at predetermined intervals. Traces can describe the path of a request or transaction in the system, including time data. They can be used to identify the runtime of individual components, the delays between them, and any bottlenecks. A trace is made up of spans, which typically represent a call or event in the system. A trace can also group together different components of a distributed system, which requires the propagation of the trace context. In the case of the HTTP protocol, this is achieved by embedding the trace identifier in the request header, so that the called component can add its own spans to the trace. Finally, logs are text entries generated while the application is running, they record events or errors. In contrast to traces, the emphasis here is not on requesting, but on providing as much detailed information as possible. As the OT format is supported by most observability backends, we keep the possibility to replace the tools used to store and display signals without having to change the application code or configuration. The OT Collector receives signals from different sources using receivers and then converts them into the format required by the observability backend in a processing pipeline.

Here we report the collection of resource usage metrics from edge nodes, execution time of each application component and trace invocations and transactions between them for compiling an end-to-end latency metric. These are crucial fundamental metrics for determining application performance. We leverage the OT Collector's built-in functionality to query the kubelet agent for node-level metrics and provide our own implementation for the rest. To track the application component interactions, we developed a special-purpose wrapper that includes monitoring instrumentation code that does not leak into the business logic. When the wrapper starts, it creates a purpose-built "tracer" object, which can be used to generate traces. It then checks whether the header of the incoming request contains an already started trace context. If so, it will append its own spans to it, otherwise a new trace is created. This ensures that we can connect the steps of an invocation of the application. After the function has finished running, it passes the trace context along to the next function and so on until reaching the last application component. The tracer object sends the traces via the gRPC protocol to the specified endpoint of the OT Collector which converts them into a suitable format. A similar process occurs in the case of node metrics received from the Kubernetes kubelet.

We opted for this method instead of the automatic instrumentation (where an OT Kubernetes operator injects monitoring sidecars into the pods executing the Knative FaaS functions) due to the monitoring container being only able to extract interpretable data at network boundaries. Thus, the method would hinder us in collecting metrics of the processes running within a given FaaS function (i.e., fused application components).

For storage and visualizing observability data, we leverage Elasticsearch [Ela25a], Elastic Application Performance Monitoring (APM) [Ela25b] and Kibana [Ela25c] due to them being open source and horizontally scalable. The OT Collector forwards collected signals to the Elastic APM server. The Elastic APM is an application monitoring system built on top of Elasticsearch that processes metrics sent to it and automatically stores them in Elasticsearch indexes, ready for querying. The APM writes the received traces to an Elasticsearch index. Elasticsearch in turn allows for redundant storage of data on multiple nodes, and distributed processing of large amount of data that fit our requirements. The data stored in the index can then be queried from

the application code via the Elastic client or from the Kibana user interface. This latter option provides a graphical user interface for exploring data stored in ElasticSearch.

Here we report the implementation of two more extensions to Knative to support edge deployments and function fusion as part of this activity: routing and edge node affinity. (We note that further components, including an initial control method realizing a load balancer functionality and function warm-up methods have been implemented as part of task *T5.3 AI-based self-management mechanisms built on swarm compute and network telemetry and capability exposure of WP5 Continuum Intelligence and Trustworthy AI applications*). Invocation routing is intrinsic for properly mapping application component invocations to Knative FaaS function invocations as it is no longer enough to determine which FaaS function to send the request to, but also to which application component within each of the functions the request is intended to. Since Knative uses the HTTP protocol for communication between functions, we wrapped this information in a unique HTTP header. We have also created a function entry point in our wrapper. This extracts the contents of the header from the incoming HTTP request, then calls the appropriate component based on the received inputs. After the component has completed its task, it returns with a value pair: the name of the component following it (if any) and the input data intended for it (a.k.a., the context). The wrapper writes the name of the next component into the header of the outgoing HTTP request, serializes the context object in its body, and then invokes the downstream FaaS function. It can be stated that the wrapper encapsulates the communication details, so the components in it contain only the business logic. We introduced the edge node affinity property for Knative FaaS functions to be able to control their placement. Such a feature is available in Kubernetes but it contradicts the serverless concept at the level of Knative. However, as we use Knative as a base deployment engine and offer extended deployment APIs that will interact only with a higher-level control entity (aware of the network topology and edge node placement – developed as part of T5.3 of WP5) and not exposed directly to users, the serverless concept remains unaffected from the end users' point of view.

We evaluated the current setup (together with the load balancer developed in T5.3) over a three-node Kubernetes/Knative cluster in a lab environment corresponding to the setup shown in Section 5.2.1 using an application that takes simplified components inspired by Use Case 5. The cluster structure is shown in **Error! Reference source not found.**. The nodes contain the following components:

- Knative Serving: Contains the resource definitions for running FaaS functions.
- Load balancer: Ensures communication between functions and uniform load on nodes.
- OpenTelemetry Collector: Responsible for collecting traces and metrics.
- ElasticSearch node: Serves as the observability backend, storing metrics and traces.

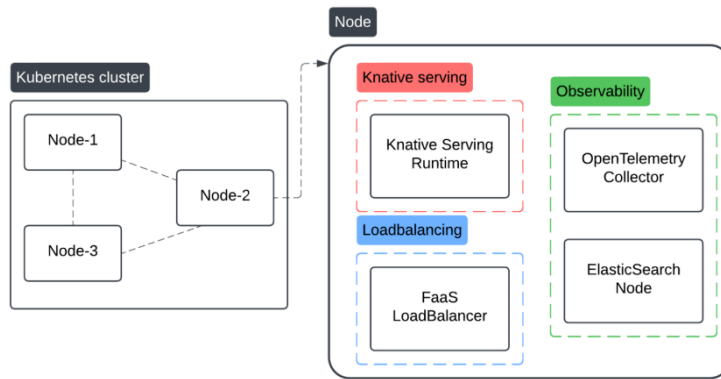


Figure 5-3: Lab environment used for testing the serverless extensions.

The load balancer and OT Collector are also deployed as daemon sets in the cluster, ensuring that one instance of each is running on each node. The purpose of this is to reduce latency, as this way each function only needs to communicate with components within its own node. The distribution of ElasticSearch nodes primarily provides redundancy. If one unexpectedly shuts down, the others can take over, ensuring uninterrupted system operation until the affected node is back up and running.

The steps of the deployed application are shown in the next figure. The Image grab component receives an image in base64 format as its input. The next processing steps resize and convert the image to grayscale, then pass it to the first object detection function. The Cut component crops the image based on the bounding boxes provided by Object detection and then performs the second phase of the object detection on the resulting pieces. Finally, the Tag component marks the found objects in the original image and saves it.

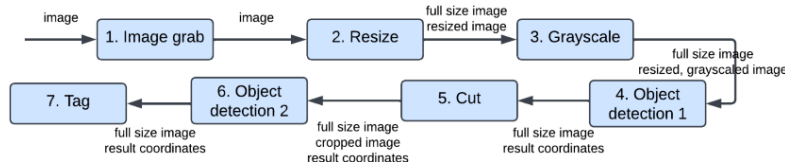


Figure 5-4: Sample application used for testing the serverless extensions.

Next figure shows a sample execution of the application through a trace tracked by our framework. Each colored line represents a span. We can see the events broken down into components, including the runtime of each step and the end-to-end latency the request is subject to while traversing the application. From the color-coded lines and the corresponding “Services” legend above, we can also track which functions the load balancer directed the request to during the run. The dashboard also contains other useful functions, such as being able to summarize the number of incoming requests for a given period, the average latency, and the error rate of the requests.



Figure 5-5: Sample visualization of the function execution and invocation latencies in the serverless test scenario.

To provide integration with the network telemetry, we designed additional components that integrate with the OT Collector to send collected metrics as ERM messages and additional entities that can receive network telemetry to be exposed at a single location for the control layer operating above the serverless runtime. We also performed initial exploration of the Knative FaaS function invocation methods and design on exposing application-level semantic information that can be processed by the network layer at the P4LB.

5.3 DYNAMIC 5G/B5G NETWORK SWARM

The approach considered for mobile network workloads (the 5G/B5G) in SMARTY is targeting trusted and resilient environments. This way the network to be deployed should allow multiple degrees of resiliency, including capability to run on a confidential computing node (e.g. x86 node with PQC and trusted execution environment). Typically, the virtualized networks are deployed utilizing popular virtualization platforms like K8 or Docker. The main architectural assumption in the SMARTY is that networks/processing nodes are self-managed and can operate differently depending if wire-speed access to AI/ML based metrics is available: a) it can offer WAI solution on its own or b) it can utilize WAI available from an underlying network, alternatively it can c) utilize cooperation between the two WAI node types (RIC and DOCA switch).

Swarmification in RAN can be provided in multiple flavors, but in general it can be perceived as the extended capabilities for the way network operates on a daily basis.

- adjusting resilience mechanisms among DU or CU level
- switching RRM between centralized vs distributed based on security status (from P4)
- enabling interaction among RIC and in-network computing (trends/profiles) in order to maximize synergetic behaviours
- in situation under attack the 5G/B5G network is expected to adjust its telemetry pipeline in order to essentially shorten it and expedite – this may be implemented as changing (redirecting) signaling traffic between E2/O1 and other interfaces (e.g. proprietary ones like Prometheus)

The common denominators in such case will be: AI/ML pipelines, policy exchange, conflict management, etc. For the speed of feedback loops two alternative loops are available based on the severity of security levels:

- Big loop: collecting telemetry via O1 -> orchestrator -> AI/ML retrain -> policy rApp -> A1 update -> RIC xApp -> dApp
- Small loop: collecting telemetry via WAI --> utilizing digital twin --> RIC xApp/dApp.

The main idea for the “small loop” is to provide faster decisions, less energy and increased resilience. Moreover, by providing additional federate knowledge databases (e.g. FL, LLM + RAG) it should be possible to ensure faster convergence of the whole network.

Based on the SOTA analysis of load balancing and security solutions for disaggregation RAN vs SMARTY infrastructure the interfaces and capabilities of utilizing WAI/In-network telemetry has been identified in section 3.6. Next important stage will be identifying complementary capabilities of the semantic knowledge that can be utilized to inform 5G/6G RAN by means of digital twinning extensions that are fueling SWARMified decision algorithms (in xApp, dApp).

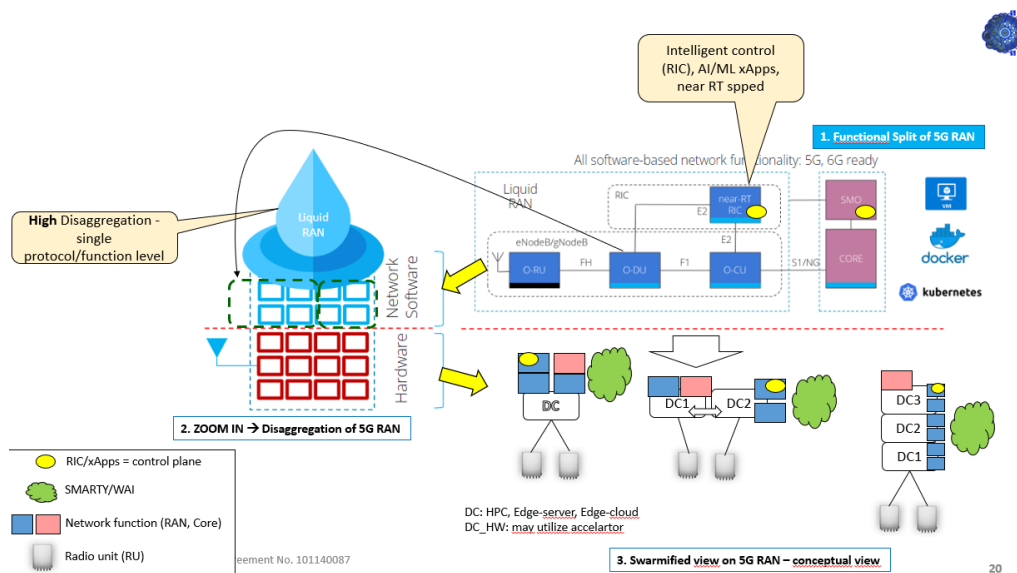


Figure 5-6: Conceptual view of Swarmification of 5G RAN

The figure presents the evolution of disaggregated RAN architecture that can adapt itself, especially when under attack by morphing its composition (service chaining) in connection with continuous adaptation with the help of digital-twinning. In the latter case the real-time (or near-RT) data from telemetry can be utilized to predict attacks and periods of instability. In the top of the figure (depicted as “1”) is the typical 3GPP/ORAN based deployment. It can be modified based on the feedback from the network (e.g. DT based) and reconfigure internal networking function modules to shift towards more disaggregated. The latter case is depicted with “2”. Eventually the custom level of disaggregation can be utilized in order to perform suitable operation techniques (e.g. scaling, migration, placement) in order to make the system more robust (depicted with “3” in the figure). This way 5G/6G systems can implement “moving target” security architecture.

5.4 NETWORK SWARM COORDINATORS AT EDGE GATEWAYS EQUIPPED WITH DPUS

This design introduces a novel architecture that moves away from traditional SDN approaches relying on cloud-centric controllers and static databases are insufficient for the dynamic, high-throughput environments emerging at the edge. In the design illustrated in the below figure, these coordinators orchestrate real-time, adaptive swarms using edge-native SDN controllers, local telemetry processing, and distributed graph stream engines. The approach replaces centralized decision-making with localized, federated intelligence, allowing networks to self-organize, rebalance, and respond to conditions with minimal latency and maximum autonomy.

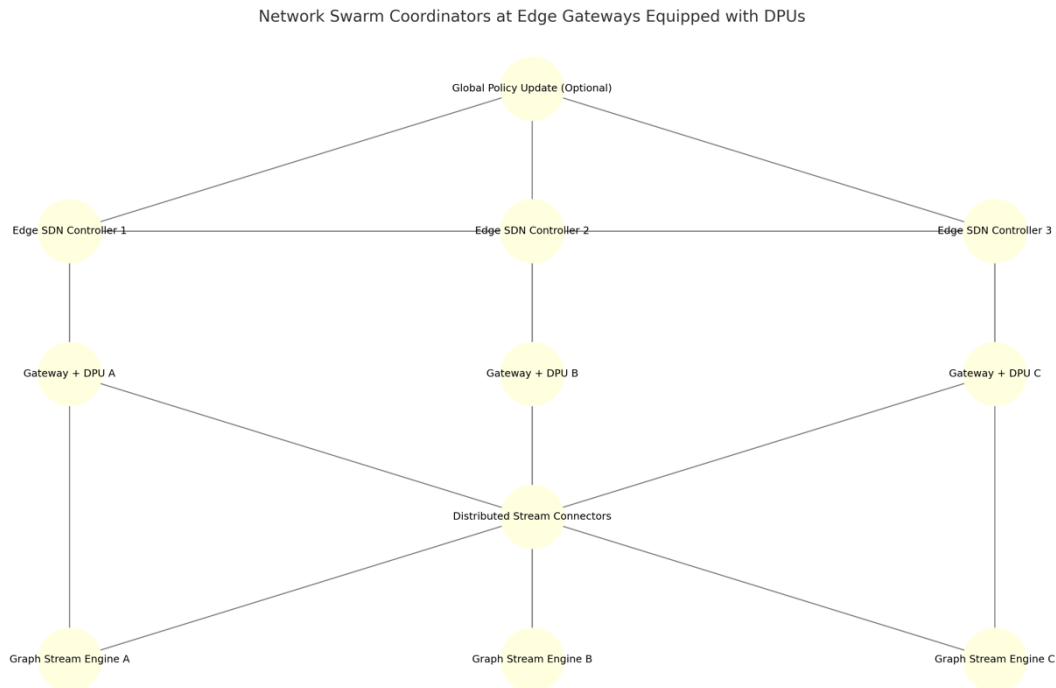


Figure 5-7: Distributed SDN Controllers for Network Swarm Coordinators at Edge Gateways

Modern DPUs offer on-chip acceleration for packet inspection, telemetry collection, and control function offloading. In this architecture, each DPU acts as an autonomous analytics unit, capturing: i) Real-time flow metrics (latency, jitter, drop rate); ii) Deep packet signatures for service classification. iii) Anomaly events (congestion, link degradation). These telemetry streams are pushed into localized processing pipelines using distributed graph stream engines extended from TUB work Fed4Edge [NDuc]. Instead of central aggregation, each region runs a dedicated graph stream engine that processes telemetry close to the data source.

To replace centralized telemetry storage and state coordination, we introduce Distributed Graph Stream Engines co-located near the edge gateways. Each engine maintains a real-time dynamic graph of: i) Flow relationships, ii) Device mobility, iii) Topological changes, and iv) Swarm clustering metrics.

These engines model the network as a live, evolving structure. Nodes represent devices or services, and edges reflect communication flows with dynamic attributes like latency and throughput. Queries and updates are executed locally and optionally shared with peer engines when inter-swarm coordination is required. This localized graph representation enables swarm-

aware routing, fast swarm reshaping, and regional anomaly detection without depending on global views.

Swarm formation is driven by both local analytics and peer-to-peer coordination among controllers. The process follows a multi-stage logic:

1. Flow Initiation or Device Join: Edge controller detects a new node or service.
2. Telemetry Activation: DPU starts capturing metrics and reporting events.
3. Graph Stream Update: The local graph engine updates topology and metrics.
4. Swarm Evaluation:
 - Grouping is adjusted based on link quality, service locality, and load.
 - Swarm transition decisions are coordinated between controllers when needed.
5. Flow Rule Programming: SDN controller issues new flow rules via P4Runtime or OpenFlow to switches or smart NICs.

Swarm decisions are adaptive and distributed, minimizing control overhead while allowing real-time responsiveness. The distributed nature means that swarms can reform or expand even during partial network outages or controller failures.

5.5 DEVICE-FLOW-GRAPH-BASED POLICY LANGUAGE

This technical contribution completes the design described in subsection 2.2 and 4.2. To enable SSLA-compliant hardware acceleration, we propose a programming model around the concept of Device Flow Graphs (DFGs). DFGs expose the underlying communication on the system interconnect to the application developer, enabling explicit and transparent control of the application's data flow between CPU and accelerators. This approach allows developers to specify fine-grained SSLA requirements—concisely through a declarative policy language—to be enforced by the secure hardware established in earlier sections. DFGs provide a solid foundation for enabling SSLA-compliant hardware-accelerated computations. This is accomplished by specifying security policies that regulate data flows through the system in a fine-grained manner, leveraging various types of attributes:

1. **Hardware attributes:** Policies can define constraints based on device characteristics, such as restricting execution to devices located in specific geographic regions or data centers, or enforcing the use of devices with specific capabilities like TEE support while excluding those without such capabilities.
2. **Software attributes:** Policies might constrain how tasks and OS services interact with data, e.g., ensuring that only authorized tasks approved by the developer can process sensitive data, or restricting OS services to meet security criteria, such as allowing only encrypted and replicated file systems to act as data sinks.
3. **Data attributes:** Policies could specify requirements for data integrity, authenticity, durability, and confidentiality. For instance, they might verify the provenance of input data before processing or prevent sensitive data from leaving specific devices (e.g., ensuring no sensitive data is transmitted off a GPU after computation).
4. **Temporal attributes:** Policies can constrain the timing of task execution or data flows to ensure compliance with regulatory requirements. For instance, policies might prohibit

using sensitive data beyond a specific expiration time, abiding by GDPR’s data minimization principle.

DFGs express an applications computation on a single node, and therefore can also be used to distribute workloads in a cluster, based on the restrictions they contain.

5.6 DERIVED INFORMATION FRAMEWORK FOR IMPROVING DECISION MAKING PROCESS

This contribution supports the development of a framework for SWARM intelligence into SMARTY, emphasizing the abstract-level formation and coordination of SWARMS. Each SWARM comprises multiple agents that represent various elements across both high and low levels. Every agent includes a secure vault to store essential information for decision-making. The framework enables agents to function autonomously as a SWARM while also allowing centralized control when necessary. Additionally, it utilizes CMPG’s Derived Information Framework to extract and apply insights for improved decision-making.



Figure 5-8: Conceptual view of Derived Information Framework to extract/apply insights for improved decision-making

The Derived Information Framework automatically updates and maintains derived data, ensuring it remains updated upon access. In principle, this functionality will be further expanded to enhance decision-making capabilities within SWARM elements. “Derivation” here is referred as the record for a singular occurrence of the fact that some pieces of information are derived or calculated from some other pieces of information.

Semantic annotation of the source and evolution of results from each experimental step:

- The process of “deriving” new information from existing inputs
- Agents as executable knowledge components
- Ontological makeup for input/output

The Depth-First Search (DFS) algorithm is utilized to traverse graphs efficiently. By employing backtracking, it updates relevant branches dynamically, ensuring optimal paths are explored. This approach supports tasks across different timescales, accommodating both asynchronous and synchronous operations seamlessly.

6. CONCLUSIONS

This deliverable reports on the design and preliminary implementation of the (i) security solutions for trustworthy edge computing and networking, (ii) software defined perimeters and orchestration through AI-techniques; (iii) metadata representation and smart allocation solutions, and (iv) monitoring and telemetry infrastructure. In essence it investigates confidential computing solutions, architectural designs of software defined perimeters, deep data plane security solutions, including decentralized features extraction and defensive wire speed AI, declarative cooperation in cloud computing environments and dynamic swarm formation. All these solutions refer to software solutions for increasing the securitization level of infrastructure used to run AI solutions. This first release includes both studies and SW implementation. These can be categorized as: a) Solutions to ensure secure networking, b) Tools and algorithms to protect from cyberattacks, c) Schemes to secure workflows across multiple stakeholders in both SW and HW level and d) schemes to protect dynamically forming of network swarms.

7. BIBLIOGRAPHY

- [Bar23] L. Barsellotti, L. De Marinis, F. Cugini, and F. Paolucci, "Ftg-net: Hierarchical flow-to-traffic graph neural network for ddos attack detection," in IEEE 24th International Conference on High Performance Switching and Routing (HPSR), 2023
- [Ela25a] Elasticsearch B.V., "ElasticSearch," 2025, [Online]. Available: <https://www.elastic.co/elasticsearch>. Accessed: 2025-03-11.
- [Ela25b] Elasticsearch B.V., "Application Performance Monitoring (APM)," 2025, [Online]. Available: <https://www.elastic.co/observability/application-performance-monitoring>. Accessed: 2025-03-11.
- [Ela25c] Elasticsearch B.V., "Kibana," 2025, [Online]. Available: <https://www.elastic.co/kibana>. Accessed: 2025-03-11.
- [Ibr25] Ahmed Ibrahim, Emilio Paolini, Filippo Cugini, Francesco Paolucci, "Real-Time Graph Neural Network for Malicious Traffic Detection", IEEE International Conference on Machine Learning for Communication and Networking (ICMLCN), 26–29 May 2025, Barcelona, Spain
- [NDUC] Nguyen-Duc, M., Le-Tuan, A., Calbimonte, J.P., Hauswirth, M., Le-Phuoc, D. (2020). Autonomous RDF stream processing for IoT edge devices. In: Wang, X., Lisi, F., Xiao, G., Botoeva, E. (eds) *Semantic Technology. JIST 2019*. Lecture Notes in Computer Science, vol 12032. Springer, Cham. [HTTPS://DOI.ORG/10.1007/978-3-030-41407-8_20](https://doi.org/10.1007/978-3-030-41407-8_20)
- [Ope25] OpenTelemetry Authors, "What is Open Telemetry?," 2025 [Online]. Available: <https://opentelemetry.io/docs/what-is-opentelemetry/>. Accessed: 2025-03-11.
- [Pao25] Paolini, E., Valcarenghi, L., Maggiani, L., & Andriolli, N. (2024). Real-time network packet classification exploiting computer vision architectures. IEEE Open Journal of the Communications Society, 5, 1155-1166.
- [Pel23] I. Pelle, F. Paolucci, B. Sonkoly, F. Cugini, "P4-assisted seamless migration of serverless applications towards the edge continuum," Future Generation Computer Systems, Vol. 146, 2023, Pp. 122-138, ISSN 0167-739X, <https://doi.org/10.1016/j.future.2023.04.010>.
- [Sam22] S. Samarakoon, Y. Siriwardhana, P. Porambage, M. Liyanage, S.-Y. Chang, J. Kim, J. Kim, and M. Ylianttila, "5g-nidd: A comprehensive network intrusion detection dataset generated over 5g wireless network," 2022
- [Yeg24] A. Yeganehfallah, A. Sgambelluri, A. Pacini, L. Valcarenghi and M. F. Silva, "Effectiveness of Confidentiality-Preserving Clustering Algorithms for Soft Failure Detection in Optical Networks," IEEE 25th International Conference on High Performance Switching and Routing (HPSR), Pisa, Italy, 2024, pp. 87-92, doi: 10.1109/HPSR62440.2024.10636007.
- [Wei21] R. Wei, L. Cai, L. Zhao, A. Yu, and D. Meng, "Deephunter: A graph neural network based approach for robust cyber threat hunting," in Security and Privacy in Communication Networks: 17th EAI International Conference, SecureComm 2021, Virtual Event, September 6–9, 2021, Proceedings, Part I 17, pp. 3–24, Springer, 2021