Grant Agreement No.: 101.015857
Research and Innovation action
Call Topic: ICT-52-2020: 5G PPP - Smart Connectivity beyond 5G

Secured autonomic traffic management for a Tera of SDN flows

D4.2: Final Evaluation of TeraFlow Security and B5G Network Integration

Deliverable type	R (Report)
Dissemination level	PU (Public)
Due date	31.12.2022
Submission date	31.12.2022
Lead editor	Alberto Mozo (UPM)
Authors	Pol Alemany (CTTC), Ricard Vilalta (CTTC), Ricardo Martínez (CTTC), Lluis Gifre (CTTC), Javier Vilchez (CTTC), Laia Nadal (CTTC), Michela Svaluto-Moreolo (CTTC), Ramon Casellas (CTTC), Sebastien Andreina (NEC), Alberto Mozo (UPM), Amit Karamchandani (UPM), Luis de la Cal (UPM), Stanislav Lange (NTNU), Min Xie (TNOR), Carlos Natalino (CHAL), Marija Furdek (CHAL), Paolo Monti (CHAL), Antonio Pastor (TID)
Reviewers	Sergio González (ATOS), Ricard Vilalta (CTTC)
Quality check team	Adrian Farrel, Daniel King (ODC)
Work package	WP4

ABSTRACT

This deliverable summarises the progress in the design and development of the TeraFlowSDN (TFS) Security and Integration components that are needed for securing and protecting the network. Specifically, this deliverable describes the progress made with the TFS Security and Integration components with respect to the previous work reported in D4.1. Furthermore, this report targets the final design and development of the TFS Security and Integration components jointly with evaluation of the components. These components are built upon and enhance base technologies. For instance, the Cybersecurity components use advanced machine-learning (ML) techniques for analysing network traffic to detect intrusions and malicious network traffic and protect the network against sophisticated attacks on the TFS network infrastructure. The Distributed Ledger Technology (DLT) Component provides novel security solutions based on Blockchain technologies that decentralise critical and sensitive data, and services of TFS. Furthermore, the NBI (formerly Compute), Integration, and Inter-Domain Components provide mechanisms and interfaces for the integration of network resources, possibly from other domains, and the connection to other networks, respectively.

Disclaimer

Acknowledgment

The research conducted by TeraFlow receives funding from the European Commission H2020 programme under Grant Agreement No 101015857. The European Commission has no responsibility for the content of this document.

Version History

Version	Date	Notes
0.1	01.Oct.2022	Table of contents created (CTTC)
0.2	02.Dec.2022	Inputs compiled and edited (UPM)
0.3.1	04.Dec.2022	Final version ready to be reviewed (UPM)
1.0	23.Dec.2022	Final version after review (UPM)
1.2	26.Dec.2022	Final version ready to be sent to QA (UPM)
1.2.1	30 Dec 2022	Revision with QA mark-ups (ODC)
1.2.2	31 Dec 2022	Final version (UPM)

EXECUTIVE SUMMARY

The TeraFlow project aims to create a novel cloud native Software Defined Networking (SDN) controller for beyond 5G networks. This deliverable summarises the progress in the design and development of the TeraFlowSDN (TFS) Security and Integration components that are needed for securing and protecting the network. Specifically, this deliverable describes the progress made with the TFS Security and Integration components with respect to the previous work reported in D4.1. In order to tackle the security and integration aspects, work package 4 (WP4) has been structured into three tasks namely T4.1, T4.2, and T4.3. The preliminary results achieved for each task are described in detail in this report. The results include implementation details as well as the publication of papers addressing the research challenges and implementation approaches.

The first task (i.e., T4.1) is concerned with cyberthreat analysis and protection, and is targeted towards designing and implementing an advanced cybersecurity solution. This solution is crucial for protecting TFS's network infrastructure in the Software-Defined Networking (SDN) domain against sophisticated attacks at the optical, network, and transport layers. Given the complexity of this task, it is performed over multiple services and specifically designed components address the individual requirements. Broadly speaking, the work is split across four different components: The Centralised Attack Detector, the Distributed Attack Detector, the Attack Mitigator, and the Attack Inference. The Centralised Attack Detector is a component that receives traffic data from the optical layer or from the Distributed Attack Detector and detects a variety of threats using a machine-learning (ML) based engine. When attacks are detected, the corresponding information is sent to the Attack Mitigator to enforce some mitigation strategy (e.g., to drop a malign connection). In T4.1, three research activities were conducted in the context of the Centralised Attack Detector. First, we developed a methodological framework for transforming neural-network-based detectors into more efficient models with respect to energy consumption. Second, we designed a new method for generating high-quality adversarial examples to transform ML-based detectors into equivalent models that are resilient to such adversarial attacks. In addition, we developed a Generative Adversarial Network (GAN) architecture to generate synthetic data that can fully substitute for real data in the training of ML-based detectors, and therefore avoid the generation of privacy breaches that might arise if real data were used to train the detectors.

The second task (i.e., T4.2) describes the design and development of a Distributed Ledger Technology (DLT) Component focusing on distributed ledgers and smart contracts to secure 5G networks. The key features of DLT, and Blockchains in particular, namely, decentralisation, immutability, and transparency, make their use appealing for managing resources and services in multi-tenant networks. Blockchains replace centralised network management with replicated databases, which lead to a resilient and trustworthy platform for storing and processing sensitive data. In the TeraFlow project, DLT is used in the multi-domain scenario to record actions by the internal components and manage TFS configurations. Additionally, the DLT serves as the data backbone for collaboration among multiple TFS nodes by sharing the SDN resources available in their transport network infrastructures. Section 4 describes the implementation of the DLT module based on the modular architecture of Hyperledger Fabric. We present the technical improvement made to the DLT Component that were published at IEEE Blockchain and ACSAC. In addition, we describe in detail the role of the DLT Component in TFS, the general architecture of the DLT Component, and how it is used to share, in a trustworthy manner, updates about the infrastructure of the different TFS domains. Another result achieved was the publication of a paper presenting a Blockchain-based architecture to provide SDN actions to configure connectivity services in transport domains.

The third set of activities covered in WP4 (i.e., task T4.3) tackles macroscopically the interworking of the TFS controller with external systems, tools, and the peering of TFS controllers that govern separate network domains. To this end, three dedicated and specialised controller/micro-service components have been developed, namely: i) the NorthBound Interface (NBI) Component (formerly referred to as the Compute Component); ii) the Web User Interface (WebUI) Component; and iii) the Inter-Domain Component. The NBI Component integrates a Network Functions Virtualization Orchestrator (NFV-O) with the TFS controller to support rolling out B5G network services (requiring both compute and networking resources). Such services are deployed over a common and shared infrastructure formed by distributed cloud sites being connected by a transport network. Thus, when the NFV-O selects the cloud sites to deploy the Virtual Network Functions (VNFs) constituting the B5G services, it queries to the TFS controller (via the NBI Component) to handle the establishment/updating/release of the network connectivity services between the selected cloud sites. In this regard, we show the interactions to deploy layer 2 virtual private network (L2VPN) services with special focus on considering energy-efficient route selection and resource allocation to reduce the overall network power consumption. The WebUI Component is a new element recently added to the TFS controller architecture to offer friendly access (via HTTP) to external users/systems. so using the WebUI, the external user/system can retrieve context information (i.e., topology, devices, links, etc.) and can trigger the setting up of network slices and connectivity services. The WebUI Component's key operations are presented and discussed. Finally, the Inter-Domain Component is responsible for interacting with peering TFS controllers governing different domains. Thus, multi-domain service/slice deployment can be achieved. This document reports the key functions and workflows between peer domain controllers to handle the required exchange of abstracted domain information (leveraging Blockchain technology) the and establishment of multi-domain slices/services.

This deliverable concludes by outlining the goals achieved in the second cycle of WP4 and the next steps to be considered after the end of WP4.

Content

List of Figures

List of Tables

Abbreviations

Abbreviation	Meaning
AE	Adversarial Example
AM	Attack Mitigator
API	Application Programming Interface
B5G	Beyond 5G
CAD	Centralised Attack Detector
CAP	Cyberthreat Analysis and Protection
DAD	Distributed Attack Detector
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DL	Deep Learning
DLT	Distributed Ledger Technology
DNN	Deep Neural Network
E2E	End-to-End
FCNN	Fully Connected Neural Network
GAN	Generative Adversarial Network
GBR	Guaranteed Bitrate
gRPC	Google RPC
hQoS	Hierarchical QoS
HTB	Hierarchical Token Bucket
IDC	Inter-Domain Component
IDS	Intrusion Detection System
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
LSTM	Long Short-Term Memory
L2VPN	Layer 2 VPN
MBR	Maximum Bitrate
MDL	Machine and Deep Learning
ML	Machine Learning
MOS	Mean Opinion Score

MS	Milestone
NFV	Network Functions Virtualization
NFV-O	Network Function Virtualization Orchestrator
ONNX	Open Neural Network Exchange
OPM	Optical Performance Monitoring
OSM	Open Source Management and Orchestration
PE	Provider Edge
QoE	Quality of Experience
QoS	Quality of Service
REST	Representational state transfer
RPC	Remote Procedure Call
SDN	Software-Defined Networking
SLA	Service Level Agreement
SSL	Sockets Security Layer
TFS	TeraFlow SDN
TLS	Transport Layer Security
UUID	Universally Unique Identifier
VID	Video on Demand
VLAN	Virtual Local Area Network
VNF	Virtual Network Function
VoIP	Voice over IP
VPN	Virtual Private Network
WAN	Wide Area Network
WGAN	Wasserstein GAN
WIM	WAN Infrastructure Manager
WP	Work Package

1 Introduction

The TeraFlow project delivers TeraFlowSDN (TFS), a next generation open source cloud native SDN controller providing smart connectivity services to B5G networks. Ensuring the security of TFS and its underlying components is paramount, and therefore this deliverable describes the progress in the design and development of the TFS Security and Integration components that are needed for securing and protecting the network. Specifically, this deliverable describes the progress made with the TFS Security and Integration components with respect to the previous work reported in D4.1 [68]. Work package (WP4) is structured into three tasks and the work in each task is subdivided into one or more components to achieve the stated goals. For each component in a task, this deliverable describes the new features, extensions, and interfaces, and the evaluation conducted on the components. The exact implementation and integration details are provided in the Milestone 4.2 (MS 4.2) report. Finally, this deliverable provides conclusions and next steps for the forthcoming evolutions of the components of the three WP4 tasks.

1.1 Purpose

The purpose of this deliverable (D4.2) is to provide a final description of TFS security and B5G network components describing new features, extensions, and interfaces and providing a final evaluation of the components.

1.2 Relation to Other Deliverables

This deliverable takes inputs from MS2.2 which includes initial use case definitions for the proposed scenarios (B5G, automotive, cybersecurity), requirement elicitation, and the draft architecture. D4.1 provides the foundation and inputs (design architecture and interfaces) to this deliverable which describes new features, extensions, and interfaces and provides a final evaluation of the TFS security and B5G network components. This deliverable also relates to D3.1 and D3.2 which provide design and implementation aspects of the core TFS components and how they interface with the Security and Integration TFS components. Furthermore, this deliverable relates to D5.2 [96], which describes the validation and evaluation of the proposed scenarios (B5G, automotive, cybersecurity).

1.3 Structure

This deliverable is structured as follows: Section 2 presents an overview of the TFS Security and Integration components that were developed in WP4. Sections 3 through 5 summarise the new features, extensions, and interfaces of the TFS Security and Integration components developed across the three tasks in WP4, namely T4.1, T4.2, and T4.3. In addition, the research contributions produced in these tasks during the last period of WP4 are detailed at the end of each section. Section 6 provides the conclusions and next steps.

2 Overview of TFS Security and Integration Components

This section provides an overview of the TFS Security and Integration components in the context of WP4. Table 1 shows how these components are mapped to the various WP4 tasks and indicates the partners that have been carrying out their design and implementation.

Table 1. Mapping of Security and Integration Components to WP4 Tasks and Contributing Partners.

WP4 Task	Security and Integration Component Name	Involved Partners
T4.1	Centralised Attack Detector	CHA, UPM, TID
	Distributed Attack Detector	TID, UPM
	Attack Mitigator	CHA, TID, UPM
	Attack Inference Component	CHA
T4.2	Permissioned Distributed Ledger and Smart Contracts	NEC, CTTC, TNOR
T4.3	NFV Orchestrator	CTTC,
	Inter-Domain Component	TNOR, NTNU, CTTC

In Section 3 we describe the four cybersecurity components: Centralised Attack Detector, Distributed Attack Detector, Attack Mitigator, and the Attack Inference Component. The Centralised Attack Detector focuses on detecting the security threats that target the physical and packetlayers in optical and packet networks respectively. The Distributed Attack Detector provides a mechanism to scale attack detection and mitigation in TFS, by grouping packets in the edge of the network and sending aggregated statistics per connection to the Centralised Attack Detector. The Attack Mitigator is in charge of enforcing mitigation action when an attack is detected in the Centralised Attack Detector. The Attack Inference Component works jointly with the Centralised Attack Detector in the optical layer to provide a scalability mechanism during inference processes.

Section 4 presents Distributed Ledger Technologies and smart contracts, and discusses how the DLT Component integrates and interfaces with other TFS components, enabling all TFS components to record, read, and register information on the DLT, as well as to use smart contracts for enhancing device and component security.

Section 5 introduces the NFV Orchestrator and the Inter-Domain Component. The front end of the NFV Orchestrator is the NBI Component (formerly the Compute Component). The Inter-Domain Component provides dedicated Quality of Service (QoS) aware inter-domain connectivity services and enables interaction among peer TFS instances.

Section 6 presents the main conclusions of each task and the future steps that can be addressed at or after the end of WP4.

3 Cyberthreat Analysis and Protection

Cyberthreat Analysis and Protection (CAP) is a crucial function that SDN controllers need to provide. TFS has one of its apps devoted to performing this task. However, two aspects make such a task very complex. Firstly, different layers of the network stack (e.g., network, physical) require different approaches due to their fundamental differences. Secondly, we may have a number of different services in the network. Each of these services might have specific cybersecurity needs, and therefore the CAP needs to be adjusted to the service-specific needs.

TFS addresses these issues in a comprehensive way, focusing on two layers of the network stack: the network and the optical physical layer. In the optical layer, the Centralised Attack Detector (CAD) is responsible for continuously assessing the security status of optical channels, and in the packet layer it detects attacks based on the connection statistics sent by the Distributed Attack Detector (DAD). The DAD is responsible for monitoring services at the packet layer, i.e., by analysing the exchanged packets and aggregating them into a set of statistics per connection to be sent later to the CAD to be analysed by an ML-based component. Due to the computational complexity of analysing packets, and the expected high volume of packets to be analysed, the DAD can be co-located with the monitored devices, reducing the network overhead, and therefore, enabling prompt responses to threats detected by the CAD. This architectural decision enables more scalable deployments of the DAD, in addition to quicker responses to threats.

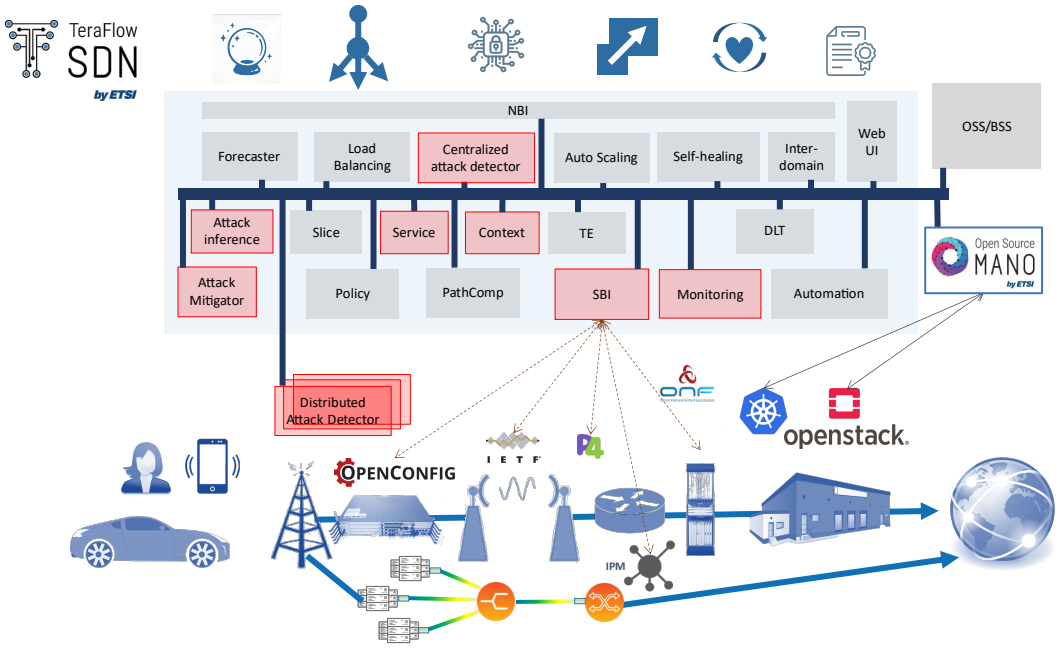

Figure 1. TFS Architecture Highlighting the Components Involved in Cyberthreat Analysis and Protection.

The second version of the CAP integrated into TFS (release 2.0 code freeze in MS4.2) decomposed the original components (Centralised and Distributed Cyberthreat Components) in to four components oriented to specific functions in the CAP (CAD, DAD, Attack Mitigator, and Attack Inference Component) and focused on evolving and integrating the workflow developed in the previous version (D4.1) with the other TFS components. Figure 1 shows the evolved TFS architecture and highlights the four new components involved in the CAP. Besides the CAP components, the integration focused on directly communicating with the Service, Context, and Monitoring Components to implement a fully

integrated workflow. Moreover, the SBI is also indirectly involved in the mitigation of attacks due to its functionality being used by the Service Component. In the following subsections, we detail each one of the four components involved in the CAP.

Figure 2 shows the integration of CAD, DAD, and Attack Mitigator with the rest of the TFS components in the packet layer cybersecurity scenario. This scenario demonstrates that novel approaches enabled by ML techniques will allow TFS to cope with new cyber threats, such as the detection of malicious encrypted traffic (e.g., cryptomining malware). Since the detection and identification of malware network flows traversing the data plane cannot be performed on a central ML-based component due to scalability issues and slow response times, we designed a distributed solution where a feature extractor is deployed at the network edge to collect and summarise the packets into connection statistics. The flow statistics aggregated by the feature extractor are sent to an ML classifier running in the CAD. Based on the real-time identification of malicious flows, the ML model will communicate with the Attack Mitigator to trigger some mitigation action (e.g., drop the malign connection) that will be enforced by the core components of the TFS Controller at scale to perform security assessments.

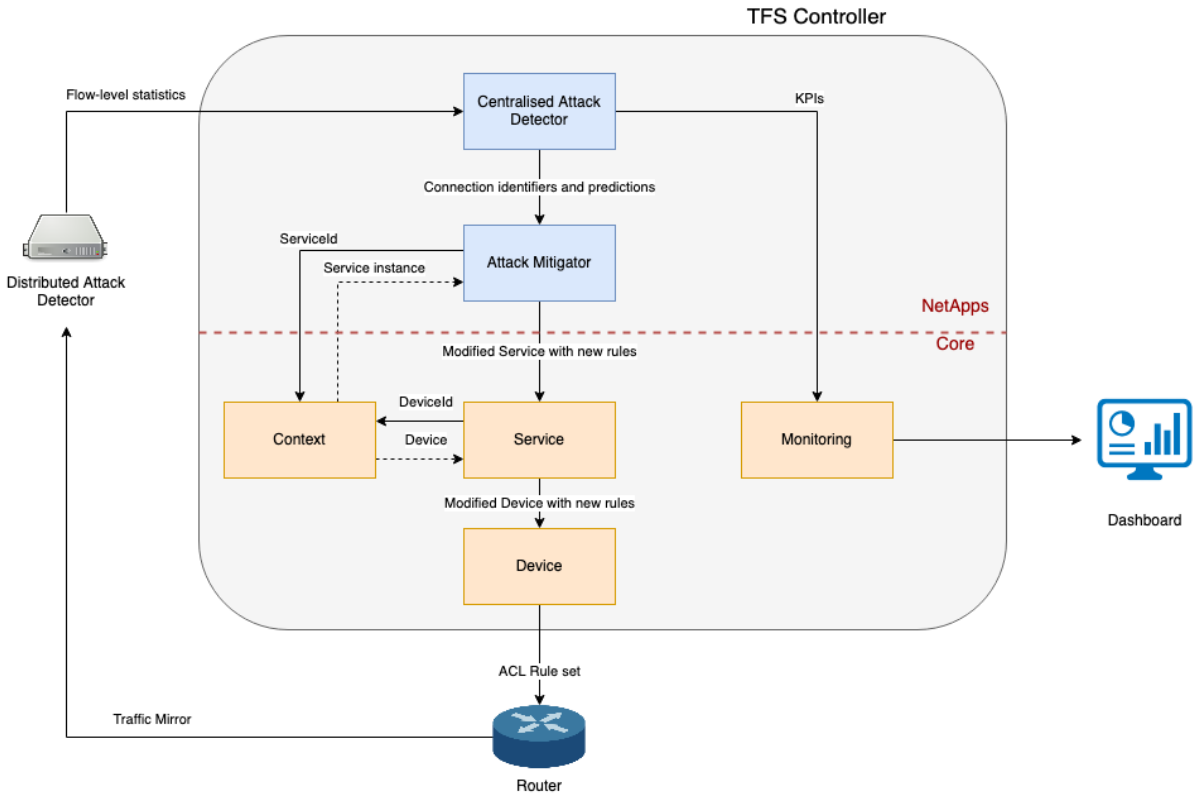

Figure 2. Final Design of Packet Layer Cybersecurity components.

Figure 3 illustrates the integration of the components related to the optical layer of the Cybersecurity scenario. This scenario focuses on demonstrating how Optical Performance Monitoring (OPM) in addition to machine learning techniques will allow TeraFlowSDN to detect, identify, and mitigate threats targeting the physical layer of optical networks. In this case, periodical data from optical devices are sent through an Optical Line System (OLS) to the monitoring component. The optical physical layer monitoring loop is triggered periodically, and retrieves the list of optical services currently running in the network. Then, for each service, the attack detector retrieves OPM data from the monitoring device and uses the Attack Inference to perform attack detection (and attack identification depending on the machine learning technique adopted). Then, if some attack is detected, the syndrome and service identifier are sent to the Attack Mitigator. The Attack Mitigator

may obtain additional context from the Context component, before performing a mitigation strategy (e.g., moving the optical service to a different spectrum) with the help of the Service component.

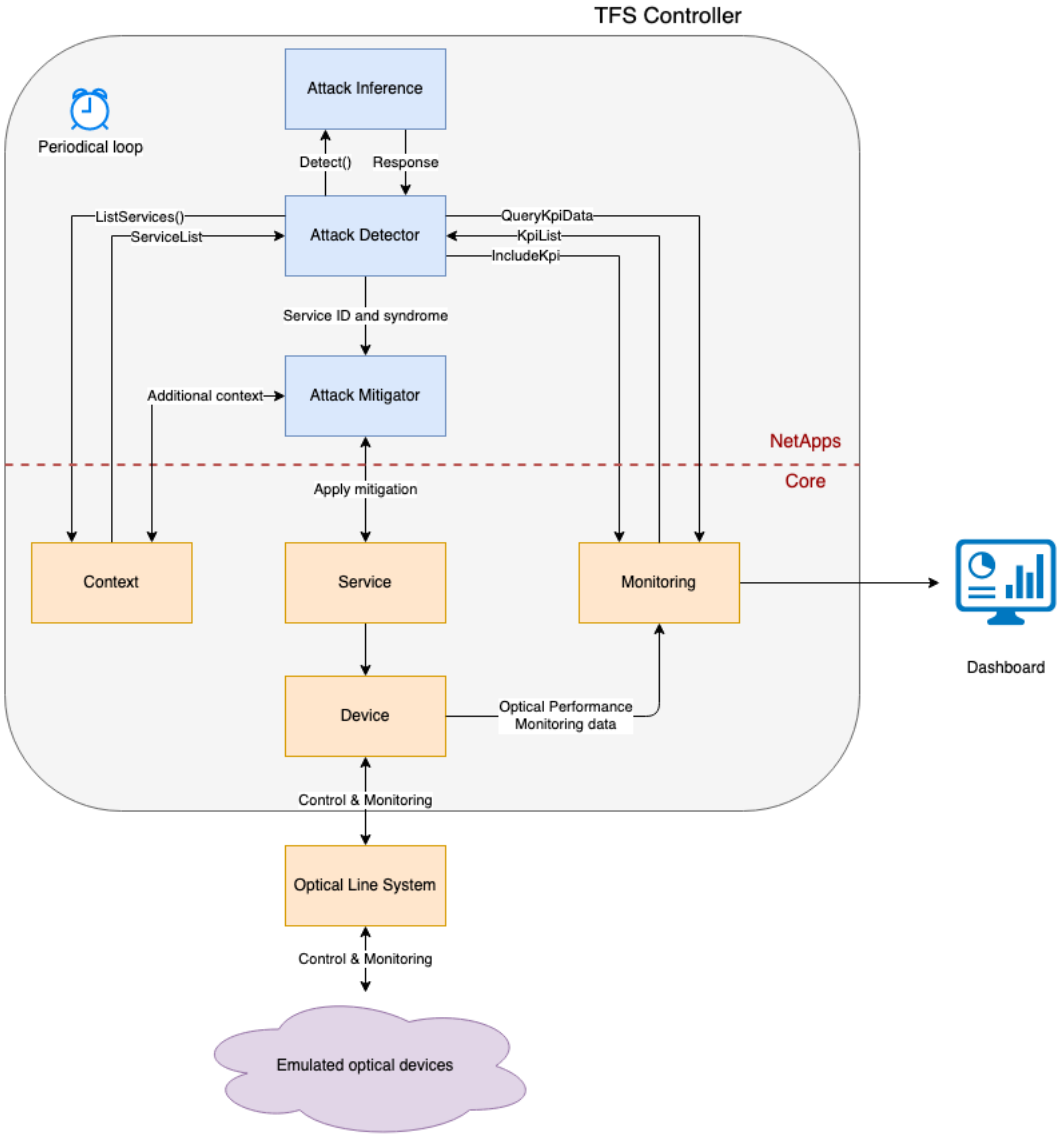

Figure 3. Final Design of Optical Layer Cybersecurity components.

At the end of this section, we summarise three research contributions conducted in T4.1 in the context of the CAD:

- Methodological framework for optimising the energy consumption of a CAD based on deep neural networks.
- Generation of black-box based adversarial examples to produce resilient CADs based on deep neural networks.
- Generation of synthetic data using Generative Adversarial Networks (GANs) to substitute and augment real data in training and testing ML-based components.

3.1 Centralised Attack Detector

The Centralised Attack Detector (CAD) is a component that works in two layers: the optical and the packet layers. In the optical layer, it receives monitoring data from the devices and performs ML-based

attack detection and identification. In the packet layer, it receives traffic data from the DAD and classifies it using an ML-based engine (e.g., as a cryptomining attack or normal traffic). This information is then sent to the Attack Mitigator and registered in the Monitoring Component.

3.1.1 New Features and Extensions

3.1.1.1 Optical Layer

The CAD was updated to take advantage of the interfaces to the Context Component that allowed the CAD to receive notifications whenever optical services are created or deleted. With this change, we expect a lower overhead caused by the operation of the CAD, both in the network and computing requirements of the Context Component. Therefore, several modifications of the CAD were conducted in order to realise a reliable and scalable loop for optical physical layer attack detection.

3.1.1.2 Packet Layer

From a design perspective, a significant part of the workload in T4.1 regarding the CAD was been centred around deploying it to a server hosted in the Telefónica premises and ensuring a lack of dependency of this component to the DAD and Attack Mitigator so that they could operate on different machines. Communication between the different machines has also been the focus of this task, especially ensuring the messages are transmitted correctly from one component to another. Another relevant change is that, as of this last milestone, the CAD only sent traffic and inference information to the Attack Mitigator if the traffic is classified as a cryptomining attack within a certain threshold, which can be adjusted using a specific parameter during the TFS deployment phase. If the traffic is classified as normal, it was deemed unnecessary to send it to the Attack Mitigator for processing, as no further action is required from the Attack Mitigator. A new ONNX (Open Neural Network Exchange) model has been trained with a new set of features to increase inference accuracy. ONNX is an open format for representing deep learning models in a standard, portable format that can be used to share models between different frameworks and tools, such as PyTorch, TensorFlow, and Caffe2. ONNX models are composed of a graph of operators (which are nodes in the graph that perform mathematical operations) and tensors (which are the inputs and outputs of these operators). The network defines how the tensors are transformed as they flow through the network, and can be serialised to a file or stored in a database, and can be deserialised and loaded using the ONNX Runtime, which is an open source project that provides a high-performance inference engine for ONNX models. ONNX Runtime can be used to accelerate the execution of ONNX models on various hardware platforms, such as CPUs, GPUs, and edge devices.

3.1.2 Final Design

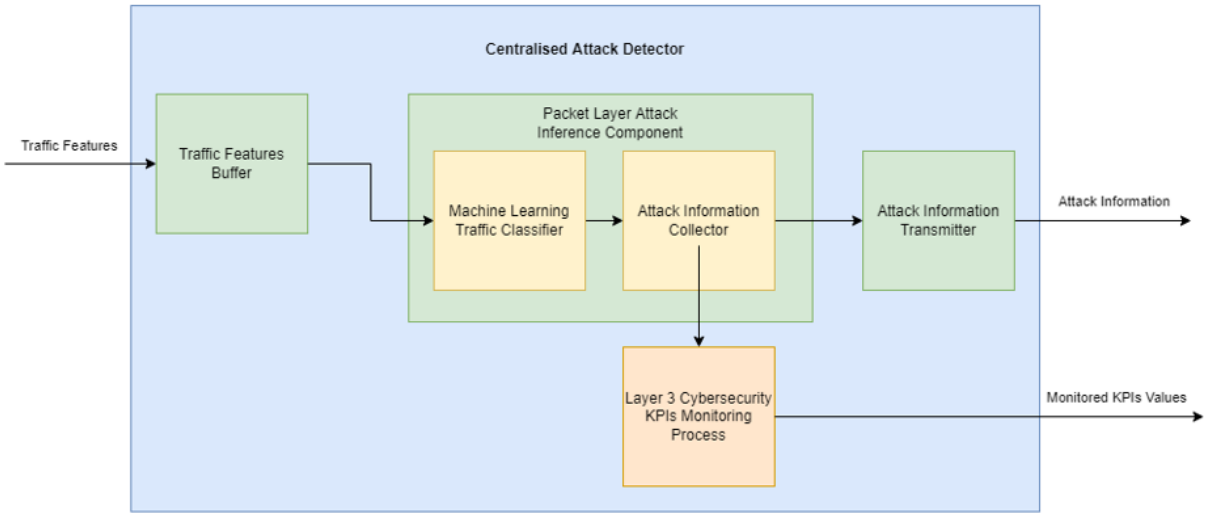

Figure 4. Final Design of the CAD.

In **Figure 4** we can observe the different internal components that are part of the CAD. The CAD takes as input a gRPC message containing traffic features that is sent from the DAD.

- **Traffic Features Buffer:** This component stores the received gRPC messages containing traffic features and sends them in batches (of a given size) to the Attack Inference Component.
- **Packet Layer Attack Inference Component:**
  - **Machine Learning Traffic Classifier:** This component receives groups of traffic features and classifies them as belonging to a cryptomining attack or not with a specified confidence and classification threshold.
  - **Attack Information Collector:** This component takes the classification results from the Machine Learning Traffic Classifier Component and transmits the information to the Layer 3 Cybersecurity KPIs Monitoring Process Component. It then crafts a gRPC message with the necessary information about traffic that is classified as an attack. This information is then sent to the Attack Information Transmitter Component.
- **Layer 3 Cybersecurity KPIs Monitoring Process:** This component receives information about the classification of the traffic and generates KPI values to monitor the presence of attacks, their rate, and other valuable information that can be easily accessed from a visual interface provided in a Grafana dashboard included in the WebUI Component. The specific KPIs that are generated by this component can be found in the deliverable D5.2 [96].
- **Attack Information Transmitter:** This component establishes a connection with the Attack Mitigator to transmit the gRPC message containing the attack information to it.

A complete description of the workflows related to the CAD can be found in deliverable D5.2.

3.1.3 Final Interfaces

The CAD exposes a gRPC interface to receive flow statistics from the DAD. Specifically, the SendFlowStatistics RPC method is used to allow the DAD to send the collected flow statistics to the CAD. In addition, a REST API is used to allow other components to query the confidence levels in the decisions made by the ML model. Finally, a configuration file is used as an input interface to allow the user to configure the component parameters, such as the size of the buffer used to store the flow

statistics before they are analysed by the ML model, and the confidence level threshold and the monitoring time interval of the cybersecurity KPIs.

3.1.4 Evaluation

The detail of the evaluations conducted on the CAD for the three topics under consideration (energy efficiency improvement, resilience against adversarial attacks, and generation of synthetic traffic for ML training) can be found in Sections 3.5.1, 3.5.2, and 3.5.3. In addition to these, we performed an evaluation of the model performance during inference. To that end, the trained model was converted to an ONNX format. ONNX is an open format well-suited for deploying deep learning models in production, since it enables faster performance and a smaller file size. In addition, this conversion process allows the model to be deployed and used in various environments, including web services and mobile devices, and on a variety of hardware platforms. The conversion process of the Deep Neural Networks (DNN) model from the TensorFlow/Keras format to ONNX is done using the tf2onnx library. Once the conversion is complete, the model can be deployed using the ONNX Runtime library, which provides an execution engine for the ONNX models.

Once the model was successfully converted to ONNX, it was evaluated in an offline fashion. This was done by first selecting a test set of data and then running inference on it using the ONNX Runtime library. The model's performance was then evaluated by comparing the predicted results to the actual labels of the data. Once the model was successfully converted to ONNX, it was evaluated offline. To do this, a test data set representing 20% of a reserved portion of the total data set that was never used for model training was first selected, and then inference was run on it using the ONNX Runtime library. The performance of the model was then evaluated by comparing the predicted results with the actual labels on the data. The metrics used to measure model performance include four well-known metrics: precision, balanced accuracy, F1 score, and confusion matrix. We incorporated balanced precision among the evaluation metrics to account for imbalances that exist in the data set. A brief explanation of the full list of metrics used to measure model performance is provided below.

- **True Negative (TN)**: This metric measures the number of cases in which the model correctly predicted a negative outcome.
- **False Positive (FP)**: This metric measures the number of cases in which the model incorrectly predicted a positive outcome.
- **False Negative (FN)**: This metric measures the number of cases in which the model incorrectly predicted a negative outcome.
- **True Positive (TP)**: This metric measures the number of cases in which the model correctly predicted a positive outcome.
- **Accuracy**: This metric measures the rate of correct predictions made by the model. It is calculated by taking the ratio of true positives and true negatives to the total number of predictions. The formula is given by: Accuracy = (TP + TN) / (TP + TN + FP + FN)
- **Balanced Accuracy**: This metric takes into account the imbalances in the dataset and measures the accuracy of the model in predicting both positive and negative classes. The formula is given by: Balanced Accuracy = (TP / (TP + FN) + TN / (TN + FP)) / 2.
- **Precision**: This metric measures the true positive rate of all positive predictions made by the model. The formula is as follows Precision = TP / (TP + FP).
- **Recall**: This metric measures the true positive rate of all true positive examples in the data set. The formula is as follows Recall = TP / (TP + FN).

- **F1 Score**: This metric is a combination of precision and recall. It is calculated by taking the harmonic mean of precision and recall. The formula is given by: F1 Score = 2 * (precision * recall) / (precision + recall).
- **Confusion Matrix**: The confusion matrix is a visual representation of the model's performance and is used to analyse the model's ability to correctly classify the data into different classes.

The results of the evaluation are shown below.

- Accuracy: 0.99996
- Balanced Accuracy: 0.99543
- Precision: 1.0
- Recall: 0.99087
- F1 score: 0.99541
- Confusion matrix:

	Predicted Negative	**Predicted Positive**
Actual Negative	97120	0
Actual Positive	4	434

From the evaluation results, it can be seen that the ONNX DNN model achieved excellent performance, with an accuracy of 0.99996, a balanced accuracy of 0.99543, a precision of 1.0, a recall of 0.99087, and an F1 score of 0.99541. This shows that the model is capable of accurately predicting the labels of the data set with a high degree of accuracy.

In addition, a temporal analysis of the ML model deployed on the CAD was performed by measuring the inference time using different batch sizes (from 1 to 1024), in an attempt to evaluate the performance of the model in real-time applications. This analysis was performed by running the inference model 1000 times, with each batch size, and measuring the time taken for each run. Mean, minimum, maximum, standard deviation, and median values were calculated for each of the batch sizes in order to evaluate the temporal behaviour of the model. The results are presented in Table 2.

Note that this evaluation was performed using a non-optimised DNN model. In Section 3.5.1, we devote considerable effort to describing how we optimised the energy consumption of the model and consequently reduced its inference time significantly. In Section 3.5.1.5.3, we perform a detailed evaluation to show how different model optimisation techniques affect the prediction performance of the ML model using the same set of batch sizes as the ones used in this evaluation.

Table 2. Mean, Standard Deviation, Maximum, Minimum, and Median Inference Time of the Machine Learning Model Deployed in the CAD Using Different Batch Sizes.

Batch Size	**Batch Inference Time**				
	Mean	**Std. Deviation**	**Maximum**	**Minimum**	**Median**
1	6.717E-04	9.275E-04	8.024E-03	1.578E-04	3.595E-04
32	8.274E-04	8.298E-04	5.373E-03	1.956E-04	4.662E-04
256	1.699E-03	3.316E-03	5.814E-02	3.635E-04	8.828E-04
1024	7.183E-03	7.404E-03	1.495E-01	1.216E-03	4.888E-03

The results presented in Table 2 show that the inference time decreases significantly with increasing batch size, as expected. Although the inference time for analysing 1024 at a time is very low, the suitability of the model for real-time application is subject to the number of connections occurring simultaneously in the network.

On the other hand, as can be seen in the table, the standard deviation is larger than the mean in the results obtained, for all batch sizes. This indicates that the data present a very skewed distribution, with most of the results clustered around the mean, but a long tail extending to much higher values. This causes the standard deviation to be larger than the mean value. Further analysis has determined that the results are accurate, and that the skewness is due to the nature of the data. Since the median also exceeds the mean, this confirms that there is a long tail of high value results. This indicates that the batch inference time is highly variable, with a few cases taking significantly longer than the mean. This observation is illustrated in Figure 5.

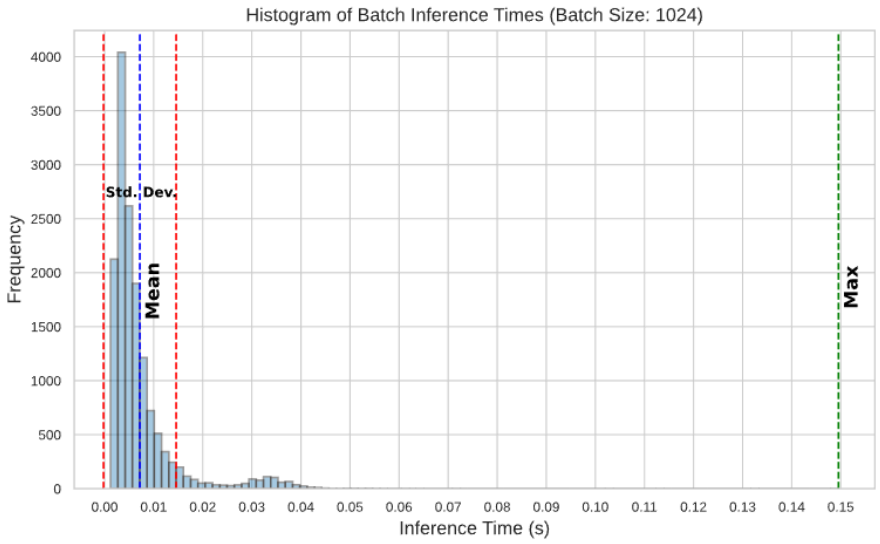

Figure 5. Histogram of the Batch Inference Times using a Batch Size of 1024.

As part of deliverable D5.3 a more thorough evaluation of the real-time performance of the model under a variety of stress conditions will be carried out and different scalability mechanisms will be studied to ensure proper operation of the cybersecurity components in the scenario 3 under high load.

3.2 Distributed Attack Detector

The Distributed Attack Detector (DAD) monitors the network data plane for the presence of malicious network flows and is deployed at the network edge to improve scalability and response time in the attack detection process and enable real-time detection of malicious traffic. To this end, a feature extractor is deployed at the network edge to collect and generate statistical summaries of network flows. To do this, packets are aggregated into flow-level statistics, where each flow is an aggregate of packets belonging to the same packet flow (same source IP address, source port, destination IP address, and destination port). This aggregation is performed for all new packets that arrive within a specific time window which can be configured. In this way, each aggregation of flow statistics is sent to the CAD to detect malicious traffic.

3.2.1 New Features/Extensions

In this new version the DAD was deployed in a realistic scenario to a different server hosted in Telefónica premises and decoupled from the CAD and Attack Mitigator so they could be deployed in separate machines. A significant enhancement in this respect was the design of new communication channels between these three components so that they could be located on different machines. From now on, the DAD can capture and process traffic transmitted from any machine across the network using a typical port mirroring technique to forward the required network traffic to it. In this way, the DAD now effectively functions as a distributed feature extractor and aggregator that is deployed at the edge of the network to collect and generate statistical summaries of network flows. These flows are packets that are aggregated into flow-level statistics and grouped within a specified time frame and are subsequently sent to the CAD to be processed by the ML engine (e.g., classifying it as cryptomining or normal traffic).

3.2.2 Final Design

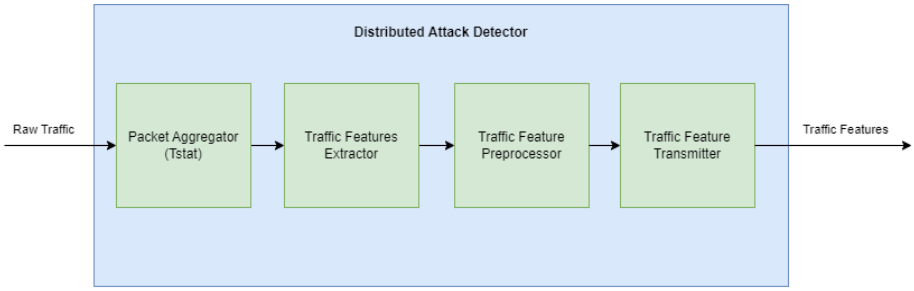

Figure 6. Final Design of the DAD.

In **Figure 6** we can observe the different internal components that are part of the DAD. The DAD takes raw traffic as an input from an attached network interface and processes it through its different internal components, which are described below.

- **Packet Aggregator (Tstat):** This component aggregates the traffic according to the flow of the packets. Therefore, packets with the same source IP address, source port, destination IP address, and destination port are grouped together.
- **Traffic Features Extractor:** This component calculates flow-level statistics from the groups formed in the previous component.
- **Traffic Feature Preprocessor:** This component extracts the specific features used by the ML model deployed in the CAD and performs the necessary preprocessing steps to adapt these features to the format expected by this ML model (scaling, imputation, etc.). In addition, this component is also responsible for creating the gRPC message that will be transmitted to the CAD and incorporates into those messages the preprocessed features that were generated by the previous component, as well as other relevant information that can be used later by other components.
- **Traffic Feature Transmitter:** This component establishes a connection with the CAD over which to transmit the gRPC messages containing the traffic features. Optionally, this component can group a batch of traffic features to be sent in a single gRPC message according to a specific parameter that defines the size of the buffer that stores these values.

A complete description of the workflows related to the DAD can be found in deliverable D5.2.

3.2.3 Final Interfaces

The DAD is an external component of TFS and, for this reason, it does not expose any gRPC interface or any REST API that allows other components to interact with it. That is, there are no inbound gRPC interfaces in the DAD component that allow other components to send requests to it. There is only one outbound gRPC interface in the DAD that allows sending information to other components, namely the CAD component. However, it should be noted that an input interface is provided in the form of the packet flows received from the network through a network interface configured to monitor the network data plane. Although this interface is not explicitly exposed and is not configurable, it is necessary for the component to perform its task. Additionally, another input interface is provided in the form of a configuration file that allows the user to specify network parameters such as the time window for aggregation of new packets received, the specific characteristics to be extracted from packet flows, the time interval during which each connection should be monitored, and the number of flow statistics sampled at each interval.

3.2.4 Evaluation

Some validation experiments were performed to ensure the correct transmission of information between the DAD and the CAD.

The evaluation scenario implemented consists of three standard Linux-based computers connected in Telefónica premises. The first computer (A) was in charge of network traffic generation using the tcpreplay tool and a set of pcap files that contain captures of real network traffic including a cryptomining attack. All the traffic stored in the pcap files can be reinjected into the network by Computer A, and the second computer (B) running the DAD receives it using the Tstat tool to capture and group into connections all the received packets. It then extracts network statistics for each connection that will be used as input features by the ML component running in the CAD in the TFS Controller of the third computer (C). To send these features to the CAD, the DAD generates gRPC messages with corresponding protobuf format. Each time the CAD receives a gRPC message from the DAD, it extracts the connection statistics and uses them as input to the ML classifier that generates a prediction based on the type of connection (i.e., whether the connection is a cryptomining attack or not). Then, a new gRPC protobuf message is sent to the Attack Mitigator with the connection identifiers and the ML inference containing the type of connection that was detected and the confidence level of the prediction.

The validation of the DAD was done using the tcpreplay tool in Computer A to reinject a pcap file containing 347k packets containing normal traffic and cryptomining attack connections that were received and processed by the DAD. Note that 346,947 packets were from normal traffic connections and only 51 from cryptomining attack connections, trying to reflect the imbalance of these two types of connections that can be found in a real scenario. The way in which Tstat was configured in this validation was to generate a snapshot (set of statistics) of a connection each time a packet was received. To maintain the efficiency of this process, when a burst of packets from the same connection were received in a short period of time, Tstat only generated one snapshot of the connection to be sent to the CAD. These results show that the DAD can capture packets in real time, group them into statistic connection snapshots, and send the corresponding connection snapshots to the CAD.

In addition to the above, a temporal analysis of the interaction between the DAD and CAD components was performed by measuring the number of snapshots (the connection statistics produced by the packet aggregator at each time step) per second sent to the CAD. To perform this measurement, the number of snapshots sent during a one-minute period was counted and then divided by 60. This

measurement was repeated 10 times and the mean, minimum, maximum, standard deviation, and median of the total number of snapshots per second obtained in each repetition were calculated. The results are presented below.

- Average number of snapshots per second: 1.66416
- Standard deviation of the number of snapshots per second: 0.23421
- Maximum number of snapshots per second: 2.19038
- Minimum number of snapshots per second: 1.3212
- Median number of snapshots per second: 1.63969

The evaluation of the DAD showed that it can capture and group packets received from the network into connections, extract in real time the required connection statistics to be used as input characteristics by the CAD, and send them in a timely manner. However, it should be noted that this evaluation was performed using Tstat as a packet capture and connection analysis tool, but since this tool was not designed with performance in mind, it is not suitable for production deployments due to its limitations in terms of efficiency in capturing time-based connection statistics. In a production system, more efficient packet aggregators, such as those based on NetFlow or IPFIX, should be used instead. Note that, to date, the available tools present a trade-off between feature extraction capabilities and performance. For developing more complex ML models, Tstat is the most suitable tool currently available, despite its lower efficiency compared to production-ready tools such as those based on NetFlow or IPFIX.

Overall, these validation experiments demonstrate that the DAD is capable of capturing network traffic and forwarding it to the CAD in a timely manner. However, these experiments constitute only a basic evaluation of the Distributed Attack Detector, as the current implementation is only a wrapper over the open source Tstat tool along with a number of feature extraction and preprocessing modules that are used to extract features from captured connections and adapt them to the input format of the ML model deployed in the CAD. Therefore, in deliverable D5.3, a more thorough evaluation of this component will be performed to address scalability issues. More specifically, the evaluation will focus on the impact of various scalability mechanisms, such as the use of different time intervals for the generation of per-connection statistics, or the use of a sampling technique to reduce the amount of data exchanged between the DAD and CAD.

A final evaluation of this component integrated with the rest of the cybersecurity components in scenario 3 (CAD and Attack Mitigator) will address scalability issues and be reported in deliverable D5.3.

3.3 Attack Inference

The ML-based attack detection and identification leveraged by our CAP function may need to take advantage of several different ML models and techniques. For this reason, in the second version of TFS we deployed the Attack Inference Component as a standalone component for our architecture. This component is responsible for hosting ML models and providing gRPC interfaces that allow any other component (especially the ones part of the CAP) to perform inferences.

The decoupling of the ML model from the other components brings several benefits:

- It allows for more general models (e.g., unsupervised learning models) to be used by any component of TFS, not only the ones pertaining to CAP.

- It allows the components (e.g., the CAD for optical services) to use multiple ML models (e.g., combine supervised and unsupervised learning) during the assessment of the services.
- It allows this component to scale independent of the other models, which may result in better resource efficiency.
- It allows ML models to be updated seamlessly without re-deploying any other component of TFS.

3.3.1 New Features/Extensions

In the current version, the Attack Inference Component is used by the CAD to perform security assessment over optical services. The component can be used in two different versions. The first version is used when we adopt supervised learning for attack detection and identification. In this case, Artificial Neural Networks (ANNs) are trained using a previously captured dataset. The resulting model is deployed using TensorFlow Serving[1] as the basis for the component. In this case, the inference request identifies which ANN model and version should be used, as well as the instances over which the inference is to be performed. Instances are similar to samples, that are composed of an array of feature values.

The second version is adopted when we use unsupervised learning, resulting in attack detection (but not identification). In this case, the algorithm of choice is DBSCAN. The deployed component uses DBSCAN Serving[2] as the basis for the component. The detection request includes the samples and features over which the anomaly detection algorithm needs to run, in addition to three parameters of the model:

- *Distance function*: The distance function to be used to calculate the distance among samples. The Euclidean distance is commonly used.
- *Epsilon*: The distance under which two samples are considered neighbours.
- *MinSamples*: The minimum number of neighbours that a sample needs to have in order to be considered a normal sample. A lower number of neighbours will result in the sample being flagged as an anomaly, which means an attack in our case.

3.3.2 Final Design

The final design of the Attack Inference Component is illustrated in Figure 7. The CAD acts as the client to the Attack Inference Component. The CAD is responsible for obtaining the OPM samples from the Monitoring Component. Once the OPM samples are available, the CAD invokes the Attack Inference Component to perform attack detection (and identification, depending on the ML model used).

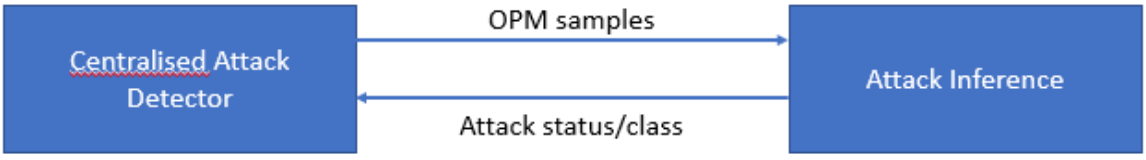

Figure 7. The Attack Inference Component Final Design.

[1] TensorFlow Serving: https://www.tensorflow.org/tfx/guide/serving
[2] DBSCAN Serving: https://github.com/carlosnatalino/dbscan-serving-python

The decision of which ML model to use impacts not only on the type of assessment that is possible, but also on the number of samples to be sent to the Attack Inference Component. If an unsupervised learning algorithm is used, the Attack Inference Component requires a number of OPM samples that is sufficient to characterise the normal operating conditions of the optical channel under analysis. If we assume an OPM cycle of 1 sample per minute from the optical devices, previous analysis showed that 300 samples are an appropriate number. On the other hand, if a supervised learning algorithm is used, only the newest samples are required to be analysed, since the normal operating conditions were already learned during training.

3.3.3 Final Interfaces

The interfaces of the Attack Inference Component are defined using protobuffers. In the case of DBSCAN Serving, there is one service containing a single RPC, three messages, and one enumeration. The enumeration defines which distance metrics are available for computing the distance between samples. The main message is called *DetectionRequest*, which contains the parameters of the DBSCAN algorithm in addition to an array of *Sample* instances. Each sample instance contains an array of feature values, where each feature value is the value for that feature for a single sample. In the case of supervised learning, the message is called *PredictRequest*, which defines the model identifier and version, in addition to the samples over which the inference is performed.

3.3.4 Evaluation

For the evaluation of the Attack Inference Component, we adopted the dataset captured in [33]. The dataset is collected from a real-world testbed with the following characteristics. Two optical services under test have 200 Gbps using 16QAM generated by commercial transponders. 10 additional channels are used to load the network and emulate realistic working conditions. The channels are loaded into a ROADM and amplified in 6 sections.

Three types of physical layer attacks are investigated: in-band jamming, out-of-band jamming, and polarisation attacks. Each attack type is launched with two intensities, denoted as light and strong. We collect the dataset by sampling the optical receiver once per minute. The normal operating conditions, and the 6 attack conditions, are each run for 24 hours, resulting in 1440 samples per condition.

Once the dataset is collected, we scale it by the unit of variance. Four dataset splits are investigated:

- OH: One-Hot encoding for the optical channel. In this case, both channels (denoted as P1 and P2) are included in the dataset, and their identification is provided as a one-hot encoded feature.
- FD: Full Dataset, denotes the case where all samples are presented to the model, but no identification of the channel is provided.
- P1: Dataset includes only one of the channels, i.e., P1.
- P2: Dataset includes only the P2 channel.

Then, we perform two tests: one with supervised learning and another with unsupervised learning. The tests with supervised learning use ANNs. We investigate the impact of the number of hidden layers and the number of neurons in each hidden layer to the final performance of the model.

We measure the cross-validation accuracy, which is the accuracy obtained after dividing the dataset into k-folds, training the dataset with k-1 of the folds, and testing the model with the last fold. The final accuracy is the average accuracy over all the rounds.

Considering the case where both channels are included, providing the identification of the channel increases the accuracy. This indicates that different channels may present different syndromes when under attack. This also indicates the need for channel-specific ML models for identifying optical physical layer attacks.

Figure ***8*** shows the average cross-validation accuracy for different number of layers and neurons at each layer. The thinner bars represent the standard deviation. In general, we can see that the ANN is more successful in classifying attacks in P2, also achieving much lower standard deviation across the k-folds. P1, on the other hand, has a much lower overall accuracy, with increased standard deviation. This shows the need for channel-specific ML models for optical cybersecurity assessment.

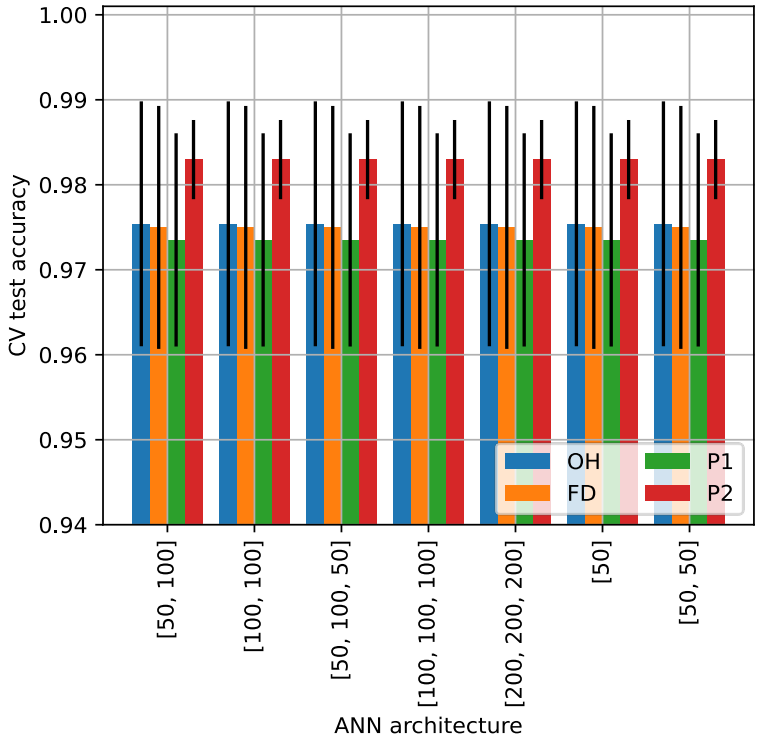

Figure 8. Average Cross-Validation (CV) Accuracy Results of the ANN.

Now we move our attention to unsupervised learning, specifically the DBSCAN algorithm. We also perform hyperparameter sensitivity analysis, varying the parameters in the following range:

- Epsilon: We tested the values in the set [0.1, 0.5, 1, 2, 3, 4, 5, 10].
- MinSamples: We tested values in the set [3, 5, 8, 10, 12, 15, 20, 50, 80, 100].

All the results are obtained using the Euclidean distance as the distance metric. We perform 50 tests for each pair of values in the hyperparameters studied. Each test consists of the following steps:

1. We randomly select 100 samples from the normal operating conditions.
2. For each attack condition, we randomly select 15 samples and run the DBSCAN algorithm over the resulting 115 samples, using the current hyperparameter configuration under test.

3. At the end of the tests with the 6 attack conditions, we average the false positive and false negative rates over all the tests.

Figure ***9*** shows the results of the hyperparameter analysis performed. Each data point represents one set of values. We highlight the curve showing the best trade-off between false positive and false negative rates. Hyperparameter values that did not yield good results (i.e., are not in the curve) are also shown, but with lighter colours. We can see that, similar to the case of ANNs, in DBSCAN the channel denoted as P2 yields better results that the one denoted as P1. On the other hand, unlike the case of ANNs, including the identification of the channel (OH) does not improve the performance of the algorithm. This can be explained by the fact that, during the inference, we only included data from one of the channels, as it is the most realistic case. When using the Attack Inference Component, it is expected that each request will contain data from a single optical channel.

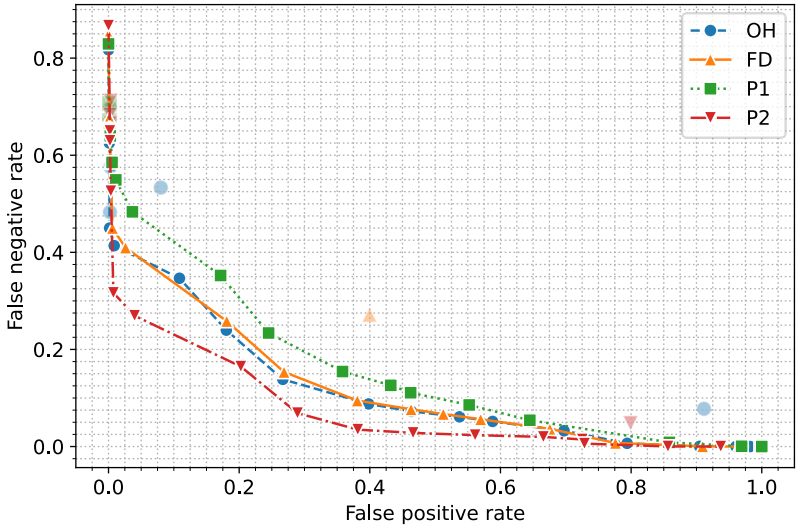

Figure 9. Performance of the Hyperparameter Analysis Using DBSCAN.

We can see that, in general, unsupervised learning does not perform nearly as well as supervised learning. On the other hand, unsupervised learning has the potential to detect previously unseen attacks, which is not the case with supervised learning. This illustrates the need for more encompassing cybersecurity analysis in optical channels, calling for ways to leverage multiple ML models and techniques in order to enable the classification of known attacks, and the detection of previously unseen attacks.

3.4 Attack Mitigator

In the previous version, the Attack Mitigator was merely a stub that received messages from the CAD regarding the identifying properties of flows such as source and destination IP addresses and ports, as well as the inference information classifying the flow as cryptomining or standard traffic, but did not process anything. However, in the current version the Attack Mitigator has been developed to process this information in the expected way.

3.4.1 New Features

3.4.1.1 Optical Layer

In the current version, the Attack Mitigator works together with the Service to compute and implement mitigation strategies. An initial strategy is to create a new optical channel with a compatible capacity to the compromised one. Then, higher-layer traffic will be migrated to the newly established optical channel. Finally, the old (compromised) optical channel will be terminated.

3.4.1.2 Packet Layer

The current version of the Attack Mitigator is deployed independently from the DAD and CAD so that they can be executed in separated machines.

3.4.2 Final Design

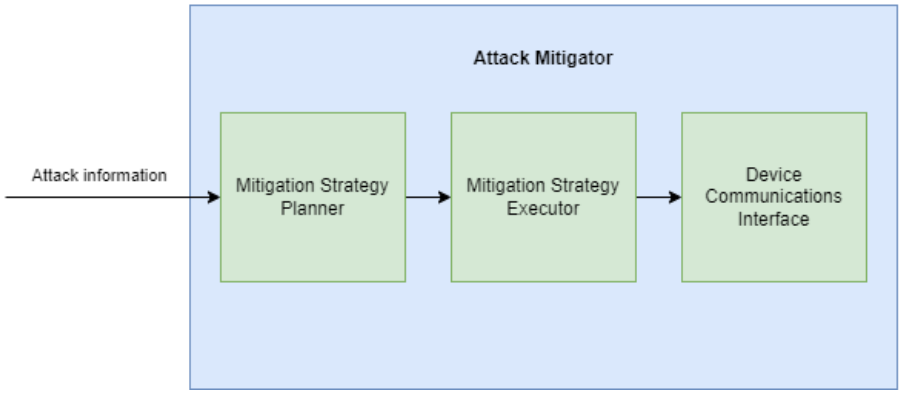

Figure 10. Final Design of the Attack Mitigator.

In **Figure 10** we can observe the different components that are part of the Attack Mitigator. The Attack Mitigator takes as input the information related to the attacks detected by the CAD and processes it through its different components to apply a mitigation strategy to block the detected attacks. These internal components are described below.

- **Mitigation Strategy Planner:** This component takes the attack information as input and develops a mitigation strategy according to its attributes and the specific conditions under which the attack was detected. The computed mitigation strategy is sent to the next component for execution.
- **Mitigation Strategy Executor:** This component takes the generated attack mitigation strategy and puts it into effect by updating the configuration of the service in which the attack was detected. These changes are then propagated to the Device Communications Interface Component.
- **Device Communications Interface:** This component translates the enforcement of a mitigation strategy to a specific action to be applied to the underlaying devices that are part of the service in which the attack was detected (for example, the creation of an ACL to be configured in a physical or emulated router to block the detected attack connections by denying incoming packets of these connections).

A complete description of the workflows related to the Attack Mitigator can be found in deliverable D5.2.

3.4.3 Final Interfaces

The Attack Mitigator exposes a gRPC interface to receive attack notifications from the CAD. In particular, the SendAttackNotification RPC method is used to allow the CAD to send attack information to the Attack Mitigator. In addition, the Attack Mitigator exposes a REST API to allow other components to query the attack mitigation strategies that are available and currently being executed by the component. Finally, a configuration file is used as an input interface to allow the user to configure the attack mitigation strategies to be used, the parameters of each mitigation strategy (if any), and the conditions under which each strategy should be used.

3.4.4 Evaluation

Some partial validation experiments have been performed to ensure the correct transmission of information between the CAD and the Attack Mitigator. In addition, the Attack Mitigator has been fully integrated with the central Context and Service Components to ensure that when the CAD detects a malicious connection (e.g., a cryptomining connection), an ACL is created and applied to the corresponding devices to apply a mitigation action (e.g., terminate the malicious connection by dropping the traffic packets related with this connection).

In addition to the above, a temporal analysis of the interaction between the CAD and Attack Mitigator was performed by measuring the time elapsed between the reception of the CAD message notifying the detection of a malicious connection and the configuration of the ACL rule in the service where the connection is occurring. The measurement of the total elapsed time between these two events was collected in 100 different trials for statistical analysis purposes, and the mean, minimum, maximum, standard deviation, and median of the total time obtained in each repetition were calculated. The results are presented below.

These results demonstrate that the transmission of information between the CAD and Attack Mitigator work effectively in collaboration with the Context and Service Components to apply a mitigation action.

- Average mitigation time (s): 15.0214
- Maximum mitigation time (s): 21.70699
- Minimum mitigation time (s): 8.86667
- Median mitigation time (s): 14.60913
- Standard deviation of the mitigation time (s): 3.67113

Currently, a single policy rule is implemented in this component that allows blocking cryptomining connections detected by configuring an ACL rule to discard traffic packets from these connections. The current implementation of the Attack Mitigator is only the basis of a more complete system. It is known that the task of mitigating malicious connections is a complex process due to the great diversity of attack techniques that can appear in the network, the different conditions under which they can occur, and the number of components involved. This implies the need to continually update system policies and improve mitigation action planning on an ongoing basis to ensure that the system is able to respond correctly in any situation. As new standards and policies are added, more comprehensive evaluation experiments will be necessary to ensure that the system is functioning correctly and effectively.

A final evaluation of this component integrated with the rest of the cybersecurity components in scenario 3 (CAD and DAD) will address scalability issues and be reported in deliverable D5.3.

3.5 Research Contributions

Several research contributions were conducted in T4.1 in the context of the CAD:

- Methodological framework for optimising the energy consumption of a CAD based on DNNs.
- Generation of black-box based adversarial examples to produce resilient CADs based on DNNs.
- Generation of synthetic data using Generative Adversarial Networks (GANs) to substitute and augment real data in training and testing ML-based components.

3.5.1 A Methodological Framework for Optimising the Energy Consumption of a CAD based on DNNs

In this section we summarise the motivation and design of the solution emphasising its novelties and the results obtained in the evaluation. The detailed results of this research can be found in "A Methodological Framework for Optimizing the Energy Consumption of Deep Neural Networks: A Case Study of a Cyber Threat Detector" [1].

3.5.1.1 Motivation

Training DNN models can be computationally expensive, requiring large amounts of memory and specialised hardware, such as GPUs or TPUs, which must run for several hours for typical applications, consuming a significant amount of power each time a new model needs to be trained. In addition, when deployed on a large scale, DNN model inference can also consume a large amount of energy over the model lifetime, especially in real-time applications where buffering the data is not possible or it is limited by the constraints of the environment, and each new input data received has to be copied to the accelerator's memory and processed through the entire model in order to obtain a single output. In addition, new instances of the model usually have to be instantiated on demand frequently, depending on the system load, which also adds to the overall high-power consumption. For this reason, as DNN-based solutions gain traction, the energy consumption associated with training and deploying DNN models has become a major concern, especially in the context of large-scale deployments. Therefore, evaluating the energy cost of ML/DL solutions is essential for understanding the sustainability of these systems and providing insights for the development of more efficient algorithms, architectures, and systems.

In addition to the energy cost, DNNs can require large amounts of resources when deployed in production environments, in terms of CPU, memory, and storage usage. Minimising the amount of storage and the amount of resources required at run-time is an important factor that can further reduce the cost of deploying and maintaining a DNN-based system, as well as reduce the carbon footprint of these systems [2] The resource utilisation of a DNN-based system limits the scalability of the system and the flexibility of the deployment. In particular, the amount of resources required by a DNN model can translate directly into the number of models that can be simultaneously trained and deployed on a single server. Although storage and resources can be added to a system at any time to meet those requirements when they change, this can further increase system power consumption, thus creating a trade-off between scalability, power, and performance.

To date, research in the area of energy-efficient DNN-based systems has focused mainly on energy efficient hardware, such as custom DNN accelerators [3][4], which aim to minimise the energy consumption of specific DNN architectures by reducing the number of operations performed and the amount of data transferred. However, these approaches are often not effective in reducing the energy consumption of DNN-based systems because they are limited to a handful of specific architectures,

primarily convolutional neural networks, which are often not the architecture of choice for specific applications outside the visual domain.

In task T4.1, we developed an integrated methodological framework to reliably analyse the energy consumption and resource utilisation of DNN-based systems when a combination of several state-of-the-art architecture-agnostic techniques is applied and the resulting model is deployed in a production environment, allowing us to understand the trade-offs between energy efficiency and performance. In our framework, two types of resources are considered: energy, which is the main resource of interest, and computational resources, such as CPU, memory, and storage usage. We validate our framework by successfully applying it to optimise a DNN classifier deployed in the CAD achieving an 82.304% reduction in total energy consumption while maintaining a 0.008% loss in the target performance metric (balanced accuracy). We provide an in-depth analysis of the energy consumption, resource utilisation, and performance of the resulting models and how they are affected by the choice of optimisation techniques.

The main novelty of the work performed in task T4.1 is the research and development of an innovative methodology to reliably analyse energy efficiency, resource utilisation, and performance by applying a combination of state-of-the-art model optimisation techniques to a target DNN model deployed in production, and automatically and reliably select the appropriate combination of techniques according to the energy efficiency, performance, and scalability requirements of the application at hand.

3.5.1.2 Energy Efficiency Optimisation Techniques for Deep Neural Networks

Deep Neural Network (DNN) training is often performed on expensive hardware accelerators that lead to high energy consumption and carbon emissions [2] This issue has been a growing concern in the AI community in recent years. It has motivated the development of more efficient algorithms to reduce the environmental impact of ML/DL models. This set of techniques, commonly referred to as Green AI, aims to fulfil this promise by reducing the energy required in the training and inference of ML/DL models.

A common approach to achieve Green AI is to use more efficient hardware. A few works have focused on developing application-specific accelerators based on FPGA or ASIC technology to efficiently train arbitrarily sized DNNs to optimise a given energy estimation model [3][4]. Although custom hardware optimised for DNN can offer significant improvements in efficiency over general-purpose hardware, it is often expensive and challenging to develop. In addition to hardware optimisation, another practical approach to realising Green AI is to use algorithms that are more efficient in their use of resources. This can be done by reducing the number of computations required for training, using more efficient data structures, or finding ways to reuse computations.

One way to reduce the energy consumption of DNN is to use Neural Architecture Search (NAS). NAS is a method that allows to automatically search for the best DNN architecture for a given task. There are many different ways to perform NAS, such as NASNet [6], DARTS [7], AmoebaNet [8], AutoKeras [9], ENAS [10], or ProxylessNAS [11], and the best method for a given application will depend on the specific constraints and objectives. Another approach is to split and distribute the computation of DNN training across multiple devices. Distributed training techniques, such as federated learning, can offer significant efficiency improvements, as they allow for harnessing the computational power of multiple devices to perform the computation. Model Compression (MC) is a related approach to achieve more efficient DNN training. This technique aims to reduce the size of the DNN model, which can lead to improved efficiency. MC can be done by pruning unused weights, using lower precision weights, or using more efficient data structures. Quantization is a technique that allows for DNN models to be

trained with lower precision, leading to improved efficiency [14]. [15] successfully applied stochastic Quantization to convert gradients to lower bit-width representations during the backward pass; this enables the use of bit convolution kernels during the backward and forward passes, which further accelerates training times and speeds up inference times. Another technique is Low-rank Factorisation (LF), which is a technique for representing a matrix with a smaller number of parameters. This can be done by decomposing the matrix into a product of two lower-rank matrices. On the other hand, Knowledge Distillation (KD) is a technique for transferring the knowledge learned by a large DNN into a smaller one. This can be done by training the smaller DNN to mimic the predictions of the larger one [16].

3.5.1.3 Analysis of the Baseline Model

In this section we describe the model we used as the basis for our energy efficiency analysis. First, we describe the setup we used to collect the data used to train the model. Next, we present the structure of the model and the procedure that was followed to train it. Finally, we evaluate the model using several standard performance metrics.

The dataset that was used to train the baseline DNN model that will serve as a target in the demonstration of the methodology has been developed for the precise task of detecting cryptomining mining attacks [17]. This dataset is provided by Telefónica I+D as part of ML research for defences against network traffic attacks generated in their Mouseworld lab [18].

The data collected in the Mouseworld lab contains traffic samples represented by a set of flow statistics derived from network packets using the Tstat tool. This traffic data was labelled to create the dataset that was used to train the cryptomining detector. In particular, two types of traffic can be found in the dataset, samples (rows) corresponding to normal traffic, and samples corresponding to cryptomining attacks. In this case, each row of the dataset was tagged as either 0 (normal traffic) or 1 (cryptomining attack traffic) using the IP addresses and ports of the known attack connections.

We used the TensorFlow library to train a Fully Connected Neural Network (FCNN) classifier to predict whether or not a connection corresponds to cryptomining activity according to all features derived from Tstat statistics except IP addresses and ports, as they are used to label the dataset (class labels) and therefore cannot be used to train the model. The structure of each of the FCNN models that was used as a baseline model is specified below. In particular, the baseline model consists of a stack of three fully connected layers with 20, 30, and 10 neurons with ReLU activation, followed by a fully connected layer with two neurons and SoftMax activation as the output layer. The model was trained using the Adam optimizer with a learning rate of 0.001 and a batch size of 4096. The model was trained for 50 epochs with a validation set consisting of 20% of the training data. The loss function used was the categorical cross-entropy, which is a standard loss function for multiclass classification problems, although in this case we are using it to classify only two classes (normal traffic and cryptomining traffic), which is equivalent to performing a binary classification with a sigmoid activation for the output layer instead of a SoftMax activation. An explanation of the ReLU and SoftMax activations is provided below.

The ReLU (Rectified Linear Unit) activation function is a simple mathematical function of the following form:

$$ReLU(x) = max(0, x)$$

The SoftMax function is a mathematical function that takes a vector of arbitrary real values and returns a vector of values that can be interpreted as probabilities. The function has the following form

$$\text{SoftMax}(x) = \left[\frac{e^{x_1}}{\sum_{i=1}^{n} e^{x_i}}, \frac{e^{x_2}}{\sum_{i=1}^{n} e^{x_i}}, \cdots, \frac{e^{x_n}}{\sum_{i=1}^{n} e^{x_i}}\right]$$

In the above equation, x is the input vector, x_i is the i-th element of the vector and n is the number of elements of the vector.

We used three well-known metrics (accuracy, balanced accuracy, and F1 score) to validate the performance of the non-optimised DNN model in a reserved portion of the data that was not used during training or validation and that represents 20% of the total dataset. We incorporated balanced accuracy among the evaluation metrics to account for the imbalances that exist in the dataset.

- Accuracy: 100%
- Balanced Accuracy: 99.5%
- F1 score: 1

We will use these results as a basis for comparison when evaluating the performance of the models obtained with the different optimization techniques that will be tested in the following sections.

3.5.1.4 Experimental Framework of the Machine Learning Energy Efficiency Evaluation

In this section, we describe the experimental framework of our methodology for analysing the energy efficiency and resource utilisation of DNN-based systems deployed in production environments. First, we explain the energy measurement process that our framework uses to collect energy consumption data and discuss the main statistics it collects to analyse the resource utilisation of the resulting models. Next, we describe how the most appropriate optimisation strategy for the problem at hand is selected in our framework. Next, we describe the energy optimisation techniques supported by our framework. Next, we provide a visual representation that illustrates the main steps of our methodology. Finally, we describe the software stack we use to implement the methodology and the requirements that the hardware platform must meet to properly execute the framework.

3.5.1.4.1 Measuring Energy and Resource Consumption

Measuring the energy consumption of a computer program is a complex task due to the multiple factors involved in this process, such as the computer architecture, the execution environment, the different power modes of the computer or battery usage, among others.

There is no procedure to precisely measure the energy consumed by a program in a deterministic way, since different computers can behave very differently depending on their components, the program to be executed, the operating system, and even the environmental conditions. The same program may produce different results in energy consumption on different computers. Therefore, power consumption can only be estimated by measuring the resource consumption and power usage of a group of representative executions and applying a statistical average to aggregate them to produce an estimate of the real power consumption. There are several ways to estimate the energy consumption of a program [19]. Traditional approaches rely on empirical evaluation of power consumption using dedicated power meters placed at different locations on the computer (power supply, GPU, VRM, etc.). More fine-grained approaches consist of estimating power consumption using simulators that model the behaviour of the main components of a computer to obtain an estimate of the power consumption of each component.

Performance monitoring counters (PMCs) are hardware-based monitors that are integrated into the processor or a specific processor chip to collect data on specific microarchitecture-related events that occur during computer execution [20]. In this regard, Intel has developed a state-of-the-art energy

model called RAPL, which estimates power consumption based on the PMC values that can be collected from Intel family processors. The framework in this study uses the RAPL interface through the *powerstat* command line profiling tool to collect the CPU power consumption of the ML models during the training, model optimisation, inference, and model loading phases.

In addition to energy measurement, information on resource consumption is collected during the three phases of the ML process (training, model optimisation, and model loading). In the following, we will refer to each of these three phases as *test*. To collect these measurements the *psutil* (python system and process utilities) library is used, which is a cross-platform library for obtaining information about running processes and system resource usage (CPU, memory, disks, network, sensors) in Python.

For each combination of techniques to be applied, the baseline model is trained and then the optimisation techniques are applied sequentially depending on the order specified in the optimisation strategy defined by the particular combination of techniques to be applied.

Once the model has been trained and optimised, its inference performance is evaluated. To evaluate the inference performance of the model obtained with each combination of techniques, the model is converted from TensorFlow/Keras to TensorFlow Lite format. TensorFlow Lite is one of the most common platforms for deploying ML models in production; being fully open source, cross-platform, and offering an industrial grade solution, it is an optimal choice for most ML applications. Note that other alternatives, such as ONNX, were discarded because of the immaturity and high complexity of this tool when working with advanced optimisation techniques, such as the one applied in this work.

In the inference stage, various batch sizes are tested to evaluate the variation in power consumption and resource utilisation. By default, the small (32), medium (256), and large (1024) batch sizes are tested, although they can be configured by the user depending on the application requirements. Additionally, the predictive performance of the optimised ML model obtained with each combination of techniques is also measured. Depending on the particular task that needs to be solved, our framework will select the most appropriate metrics to perform the evaluation, being easily extensible to incorporate other metrics as required by the problem at hand. The user must determine the type of task to be solved and select a metric as the objective to be optimised during the selection of the most appropriate combination of techniques to be applied: this is discussed in the next section.

For each of the optimisation strategies that can be applied, a configurable number of repetitions can be defined for each test (model training, inference, and load) can be defined (by default it is set to 5) together with an intermediate waiting time to restore the initial conditions of the machine where the framework is being executed. By setting a high number of repetitions, more reliable measurements can be obtained. During each repetition, the metrics listed below are collected at a configurable time interval (by default it is set to 1 second).

- Unique Set Size (USS) of the process.
- Percentage of CPU used by the process.
- System-wide CPU frequency.
- System-wide CPU power draw.

Once all repetitions have been performed, the metrics obtained at each step in each of the repetitions are aggregated using the mean, standard deviation, and maximum value.

In addition, a second aggregation is also performed, but on this occasion on the time axis to show the mean value, standard deviation, and maximum of each statistic measured throughout the test as a

summary of the results. In this way, a percentage is obtained that can be used as a metric of the improvement in energy efficiency achieved by the optimisations. The obtained value can be positive or negative depending on the change in the energy consumption of the optimisations with respect to the baseline. A positive value shows that the optimisations perform better than the baseline, while a negative value shows that the optimisations consume more energy than the baseline.

On the other hand, for each test, the size of the compressed model disk is also reported using the Deflate format. Compression is especially important for pruned models because, in the serialised weight matrices, the pruned connections are represented as zero and therefore have the same size as the weight matrices of the non-pruned model. Therefore, a compression algorithm has to be applied to remove this redundancy and obtain a much smaller model. For this reason, this step is crucial to obtain a representative file size of the ML that is deployed in production.

3.5.1.4.2 Selection of the Best Optimisation Strategy

Before starting the model optimisation process, the user must select the optimisation profile that will be used. Three different optimisation profiles are considered:

A. **Energy efficiency profile**: This profile is designed to minimise the energy consumption of the optimised model while providing acceptable performance for the user. For this purpose, the optimisation strategies are ranked in descending order according to the reduction in energy consumption they provide with respect to the non-optimised model, and then the optimisation strategy (or strategies) that provides the largest reduction is selected.

B. **Performance profile**: This profile is designed to maximise the performance of the optimised model while providing an acceptable energy efficiency gain with respect to the non-optimised model. To do this, the optimisation strategies are ranked in descending order of the performance they provide based on a user-defined target performance metric, and then the optimisation strategy (or strategies) that provides the highest performance is selected.

C. **Balanced profile**: This profile is designed to balance performance and energy efficiency by applying the optimisation strategy that provides the best trade-off between both metrics. For this purpose, two parameters are provided: the weight of the performance metric, and the weight of the energy efficiency metric.

Regarding the selection of the most appropriate combination of techniques to apply, an exhaustive search of all combinations of optimisation strategies is performed. More specifically, the framework builds a set of all possible combinations of techniques that can be applied to the DNN model. Once the set of all combinations has been constructed, the framework will evaluate each combination in the set to select the most appropriate combination to apply according to the optimisation profile selected by the user. In this way, the framework will optimise the DNN model using the most appropriate optimisation strategies among all those available according to the optimisation profile defined by the user.

3.5.1.4.3 Framework Requirements

The framework was implemented using Python (version 3.10.6) as the main programming language and using the following dependencies: TensorFlow (version 2.9.2), TensorFlow Model Optimization (version 0.7.3), psutil (version 5.9.4), and powerstat (version 0.02.27).

To run the framework correctly, it is necessary to have a Linux operating system with a CPU that supports the Intel RAPL interface. If the CPU does not support this interface, the powerstat library will

not work properly and the total energy consumed by the DNN during training and inference will not be obtained. As a result, only the resource utilisation of the DNN during the training, inference, and load stages would be obtained.

3.5.1.4.4 Supported Model Optimisation Strategies

Three different sets of optimisation strategies are supported in our framework, each containing several different combinations of the most promising state-of-the-art optimisation techniques to evaluate the most effective approach to minimise the total energy consumption of an ML model during the training, inference, and loading stages by performing a quantitative analysis of the energy and resource consumption metrics specified in Section 3.5.1.4.1.

The first set contains combinations of quantization techniques that can be applied as a post-processing step after training the ML model. The second set contains various methods of compressing the physical representation of an ML model (number of total model parameters or the in-memory size of each parameter), in order to reduce the amount of energy consumed by an ML model during the inference stage and in subsequent retraining that might be necessary due to a change in the underlying data distribution during the operational stage. Finally, the third set contains combinations of the individual techniques included in the other two sets.

The specific combinations of techniques in each set are listed in Table 3. It should be noted that, in order to reduce the computational cost and time spent on the third test set, only the optimal post-training quantization technique according to the results of the first test set and in terms of the selected optimisation profile is applied to the models of the third set. The reason for this choice is that all the post-training quantization techniques have a negligible computational cost compared to that of the techniques found in the second set. Therefore, by evaluating them beforehand and selecting the optimal one as the one used for the combinations present in the third test set, the total evaluation time is considerably reduced. After preliminary validation, we conclude that this approach does not affect the results obtained with the third set in any meaningful way.

An important detail to note is that the baseline model is trained using a fixed number of epochs rather than using the early stopping technique. The decision to do so is primarily because the early stopping technique does not always guarantee that the model will complete training at an equal number of epochs each time it is performed. This, in turn, affects the energy consumption metric obtained on each occasion and would not provide a fair metric for comparisons between different optimisation techniques.

Table 3. Supported Energy Efficiency Optimisation Strategies.

Set	Optimisation Strategy Id.	Optimisation Strategy
N/A	0	No optimisations (baseline)
Post-Training Optimisation Techniques	1	Full 8-bit Integer (INT8) Weight Quantization
	2	Half-precision Floating-point (FP16) Weight Quantization
	3	Full Integer Weight Quantization with 16-bit Integer (INT16) Activations and 8-bit Integer (INT8) Weights
	4	Pruning-aware Model Fine-tuning
	5	Quantization-aware Model Fine-tuning

Training-aware Optimisation Techniques	6	1) Neural Architecture Search 2) Knowledge Distillation
Combined Optimisation Techniques	7	1) Pruning-aware Model Fine-tuning 2) Quantization-aware Model Fine-tuning
	8	1) Neural Architecture Search 2) Knowledge Distillation 3) Pruning-aware Model Fine-tuning
	9	1) Neural Architecture Search 2) Knowledge Distillation 3) Quantization-aware Model Fine-tuning
	10	1) Neural Architecture Search 2) Knowledge Distillation 3) Pruning-aware Model Fine-tuning 4) Quantization-aware Model Fine-tuning
	11	1) Pruning-aware Model Fine-tuning 2) Optimal post-training Quantization
	12	1) Neural Architecture Search 2) Knowledge Distillation 3) Optimal post-training Quantization
	13	1) Neural Architecture Search 2) Knowledge Distillation 3) Pruning-aware Model Fine-tuning 4) Optimal post-training Quantization

3.5.1.4.5 Summary of the Methodology

In Figure 11 we represent the complete workflow of the methodology that we have described in this section. As can be seen, the methodology is divided into four main steps. The first step is to perform the energy optimisations of the DNNs. For this, the TensorFlow Model Optimization (TFMO) library is used, which allows us to apply different pruning and quantization techniques to our target DNN. In the second step, the model is converted to TensorFlow Lite format, a production-ready format that accelerates inference speed and reduces storage requirements. In the third step, the powerstat and psutil libraries are used to profile the energy consumption and resource utilisation of the DNNs during the training, inference, and loading stages. In the fourth step of the methodology, the energy-accuracy trade-off that exists with the application of different energy optimisations is analysed to provide a detailed report of the impacts of different energy optimisation techniques on the prediction accuracy of the target DNN. In the fifth and final step, the automatic selection of the best optimisation strategy is carried out according to a user-defined optimisation profile that represents the user's preferences in terms of the relationship between energy efficiency and accuracy shown by the final optimised model.

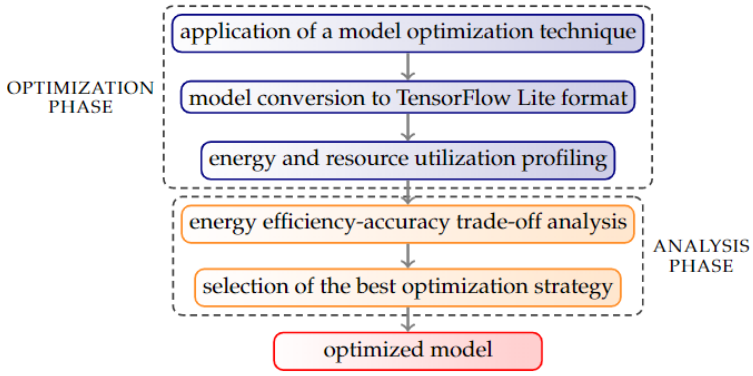

Figure 11. Summary of the Main Steps of the Workflow in our Energy Efficiency Methodology Framework for DNNs.

3.5.1.5 Experimental Evaluation

In this section, we present an experimental evaluation of the framework. First, we describe the experimental setup that we have defined to test the framework, including the main parameters that have been set in the framework to configure the optimisation process to be applied to our target model, as well as the hardware platform that we have used to perform the experiments. In addition, we specify the selection of hyperparameters that have been used for the application of the optimisation techniques that were tested. Next, we analyse the experimental results obtained in the model inference state for each of the optimisation strategies that were applied. Moreover, we also provide a detailed timing analysis of the different stages of the model inference and load stages and an evaluation of the energy cost of the training and application of the different optimisation strategies applied. Finally, we provide a summary of the main conclusions of our experimental evaluation.

3.5.1.5.1 Experimental Setup

To demonstrate the effectiveness of the methodology, we performed an experimental evaluation in which we applied the methodology with all combinations of supported techniques to the cryptomining detector described in Section 3.5.1.3 using the default parameters and settings of our framework (number of repetitions set to 5 and sample measurement time interval of 1 second). In this way, we were able to obtain a detailed analysis of the trade-off between energy and accuracy that exists when applying energy optimisation techniques to an FCNN.

To carry out the optimisation process with the framework, we use the three available profiles (energy efficiency profile, performance profile, and balanced profile) to select the most appropriate optimisation strategy. We set, as performance threshold, a minimum acceptable reduction in energy consumption with respect to the non-optimised model of 25% and a minimum balanced accuracy of 0.9. Furthermore, to apply the balanced profile, we set the ratio of these two factors as 0.5 for both in order to obtain the optimisation strategy that leads to the most balanced results between the two objectives and analyse its comparison with those obtained with the other two profiles.

The hardware platform used to validate the methodology is a system with an Intel(R) Core(TM) i7-2600 CPU (Sandy Bridge microarchitecture; base clock 3.40 GHz; turbo boost 3.80GHz; 8 MB cache). The system has 32 GB of RAM and Ubuntu 20.04.5 (with 5.15.0-48-generic Linux kernel) was used as the operating system. As the run-time environment, we used Python (version 3.10.6) and the following packages: TensorFlow (version 2.9.2), TensorFlow Model Optimization (version 0.7.3), psutil (version 5.9.4), and powerstat (version 0.02.27).

3.5.1.5.2 Results Obtained in the Inference Stage

In this section, we present the results obtained in the inference stage of the optimised models. These results determine the optimisation strategy that provides the best compromise between energy efficiency and performance according to the three optimisation profiles defined in 3.5.1.4.2. Three different batch sizes (small: 32, medium: 256, and large: 1024) were tested to analyse the influence of the batch size used to perform the prediction on the results obtained. The results obtained for the optimisation strategies that have been applied to the objective model are shown in **Table 4**, **Table 5**, and **Table 6** for the three batch sizes considered, respectively. The results shown in these three tables correspond to the mean, standard deviation, and maximum energy consumption and resource utilisation metrics derived from the aggregation of the measured values collected over the duration of the model inference at 1-second intervals and over 5 iterations for each optimisation strategy.

We observe from the results presented in **Table 4**, **Table 5**, and **Table 6** is that almost all optimisation strategies lead to a significant reduction in energy consumption, exceeding in most cases the target threshold of reduction in energy consumption with respect to the non-optimised model that was set at the beginning of the experimental evaluation.

Table 4. Energy Consumption and Resource Utilisation Metrics for the Optimisation Strategies Using a Batch Size of 32.

Opt. Strategy Id.	0	1	2	3	4	5	6[a,c]	7	8[b]	9	10	11	12	13
Total Avg. CPU Energy Consumption (J)	4.127	3.691	1.847	8.418	1.879	3.696	0.828	3.921	1.84	3.882	3.872	1.905	**0.795**	1.858
Percentage of Total Avg. CPU Energy Consumption Reduction (%)	N/A	10.576	55.243	-103.962	54.482	10.456	79.927	4.999	55.427	5.944	6.187	53.832	**80.741**	54.974
Avg. CPU Power Draw (W)	**15.504**	15.77	15.757	15.668	15.807	15.67	15.731	16.018	15.862	16.101	16.064	16.093	15.902	15.883
Std. Dev. CPU Power Draw (W)	0.083	0.349	0.279	0.147	0.281	0.259	0.12	0.375	0.34	0.302	0.29	0.319	**0.076**	0.216
Max. CPU Power Draw (W)	**25.69**	26.82	26.82	26	26.96	26.66	26.03	27.23	27.46	27.18	27.3	27.44	26.38	26.55
Avg. CPU Usage (%)	6.244	6.269	6.244	6.244	6.269	6.244	6.244	6.269	6.244	**6.219**	6.269	6.244	6.244	6.269
Std. Dev. CPU Usage (%)	**0**	0.05	0.112	0.079	0.05	**0**	**0**	0.05	**0**	0.05	0.094	**0**	0.079	0.094
Max. CPU Usage (%)	**12.488**	12.738	12.738	12.738	12.738	**12.488**	**12.488**	12.738	**12.488**	**12.488**	12.738	**12.488**	12.738	12.738
Avg. CPU Frequency (MHz)	1.913	2.002	1.936	2.062	1.899	1.985	1.948	2.002	**1.869**	1.952	2.106	1.966	1.99	1.959
Std. Dev. CPU Frequency (MHz)	0.072	0.07	0.087	0.107	0.039	0.08	0.088	0.095	**0.008**	0.052	0.195	0.078	0.143	0.109
Max. CPU Frequency (MHz)	2.138	2.149	2.16	2.406	2.137	2.27	2.318	2.163	**2.136**	2.145	2.456	2.14	2.408	2.177
Avg. RAM Memory USS (B)	10.105	**6.902**	9.97	9.85	9.985	9.876	9.869	9.89	10.02	9.881	9.871	10.023	9.894	10.029
Std. Dev. RAM Memory USS (B)	0.01	0.021	0.012	0.025	0.012	0.014	0.014	0.013	0.022	0.011	0.016	0.023	0.022	**0.009**
Max. RAM Memory USS (B)	16.75	**10.445**	16.473	16.344	16.539	16.34	16.289	16.352	16.598	16.352	16.367	16.617	16.367	16.605
Accuracy	1	0.996	1	1	1	1	0.984	1	1	1	1	1	0.997	1
Balanced Accuracy	0.995	0.5	0.995	0.995	0.995	0.995	0.987	0.995	0.995	0.995	0.995	0.995	0.708	0.995
F1 Score	1	0.993	1	1	1	1	0.989	1	1	1	1	1	0.997	1

[a] Best optimization strategy according to the energy efficiency profile.
[b] Best optimization strategy according to the performance profile.
[c] Best optimization strategy according to the balanced profile.
Underlined values: Top three optimization strategy that resulted in the best improvement in the measured statistic.
Bold and underlined values: Optimization strategy that resulted in the best improvement in the measured statistic.
Double underlined values: Optimization strategy that results in a model with performance above the minimum acceptable performance threshold.
J: Joules. W: Watts. B: Bytes. MHz: Megahertz

Table 5. Energy Consumption and Resource Utilisation Metrics for the Optimisation Strategies Using a Batch Size of 256.

Opt. Strategy Id.	0	1	2[b]	3	4	5	6[a,c]	7	8	9	10	11	12	13
Total Avg. CPU Energy Consumption (J)	4.309	3.651	1.854	8.495	1.887	3.732	**0.763**	3.682	1.867	3.748	3.827	1.931	0.817	1.873
Percentage of Total Avg. CPU Energy Consumption Reduction (%)	N/A	15.28	56.975	-97.138	56.199	13.398	**82.304**	14.551	56.666	13.025	11.195	55.188	81.046	56.532
Avg. CPU Power Draw (W)	16.138	15.794	**15.691**	15.773	15.782	15.758	15.806	15.88	15.898	16.088	15.961	16.11	15.842	15.98
Std. Dev. CPU Power Draw (W)	0.851	0.267	0.119	0.055	0.289	0.273	0.263	0.232	0.28	0.371	0.383	0.278	0.074	**0.054**
Max. CPU Power Draw (W)	25.86	26.94	**25.81**	26.05	26.89	26.9	27.03	26.69	27.21	27.75	27.23	27.26	26.22	26.49
Avg. CPU Usage (%)	6.219	**6.169**	6.194	6.269	6.194	6.244	6.244	6.219	6.269	6.244	6.219	6.244	6.294	6.219
Std. Dev. CPU Usage (%)	0.094	0.061	0.061	0.05	0.061	**0**	**0**	0.05	0.094	**0**	0.094	0.079	0.1	0.094
Max. CPU Usage (%)	12.738	**12.488**	**12.488**	12.738	**12.488**	**12.488**	**12.488**	**12.488**	12.738	**12.488**	12.738	12.738	12.988	12.738
Avg. CPU Frequency (MHz)	1.975	1.976	2.012	2.195	**1.89**	1.991	1.935	1.977	1.952	2.021	1.942	1.969	1.92	1.918
Std. Dev. CPU Frequency (MHz)	0.089	0.101	0.13	0.236	**0.047**	0.135	0.091	0.153	0.101	0.165	0.075	0.086	0.084	0.071
Max. CPU Frequency (MHz)	2.156	2.404	2.416	3.168	2.145	2.252	2.414	2.413	2.164	2.412	2.146	2.17	**2.141**	2.15
Avg. RAM Memory USS (B)	10.559	**7.415**	10.509	10.393	10.496	10.423	10.356	10.427	10.545	10.422	10.39	10.552	10.386	10.543
Std. Dev. RAM Memory USS (B)	0.017	0.031	0.018	0.021	0.025	0.024	0.014	0.016	**0.013**	0.024	0.016	0.029	0.021	0.027
Max. RAM Memory USS (B)	17.387	**11.215**	17.289	17.16	17.332	17.168	17.051	17.16	17.324	17.16	17.078	17.375	17.105	17.336
Accuracy	1	0.996	1	1	1	1	0.984	1	1	1	1	1	0.997	1
Balanced Accuracy	0.995	0.5	0.995	0.995	0.995	0.995	0.987	0.995	0.995	0.995	0.995	0.995	0.708	0.995
F1 Score	1	0.993	1	1	1	1	0.989	1	1	1	1	1	0.997	1

[a] Best optimization strategy according to the energy efficiency profile.
[b] Best optimization strategy according to the performance profile.
[c] Best optimization strategy according to the balanced profile.
Underlined values: Top three optimization strategy that resulted in the best improvement in the measured statistic.
Bold and underlined values: Optimization strategy that resulted in the best improvement in the measured statistic.
Double underlined values: Optimization strategy that results in a model with performance above the minimum acceptable performance threshold.
J: Joules. W: Watts. B: Bytes. MHz: Megahertz

Table 6. Energy Consumption and Resource Utilisation Metrics for the Optimisation Strategies Using a Batch size of 1024.

Opt. Strategy Id.	0	1	2[b]	3	4	5	6[a,c]	7	8	9	10	11	12	13
Total Avg. CPU Energy Consumption (J)	4.463	3.604	1.865	8.498	1.984	3.791	0.798	3.766	1.871	3.783	3.868	1.898	**0.795**	1.9
Percentage of Total Avg. CPU Energy Consumption Reduction (%)	N/A	19.249	58.208	-90.411	55.555	15.05	82.124	15.609	58.074	15.242	13.322	57.47	**82.183**	57.437
Avg. CPU Power Draw (W)	16.764	**15.613**	15.689	15.763	16.876	15.767	15.753	15.967	15.848	15.958	16.323	16.521	16.839	16.029
Std. Dev. CPU Power Draw (W)	2.774	0.283	0.337	**0.213**	1.774	0.421	0.448	0.818	0.312	0.82	1.003	2.381	2.789	0.729
Max. CPU Power Draw (W)	38.96	26.71	26.8	25.92	25.84	**25.66**	26.46	26.85	27.19	26.09	27.07	36.97	26.22	26.48
Avg. CPU Usage (%)	6.244	6.219	6.269	6.244	6.244	6.244	6.244	6.294	6.219	6.219	6.219	6.219	6.244	**6.194**
Std. Dev. CPU Usage (%)	**0**	0.094	0.05	0.079	0.079	**0**	0.079	0.061	0.05	0.05	0.05	0.05	0.079	0.061
Max. CPU Usage (%)	**12.488**	12.738	12.738	12.738	12.738	**12.488**	12.738	12.738	**12.488**	**12.488**	**12.488**	**12.488**	12.738	**12.488**
Avg. CPU Frequency (MHz)	2.298	1.947	1.922	2.051	1.938	1.951	**1.891**	1.961	1.914	1.906	1.939	1.922	1.974	1.925
Std. Dev. CPU Frequency (MHz)	0.357	0.095	0.074	0.053	0.086	0.075	**0.035**	0.113	0.068	0.069	0.133	0.089	0.143	0.089
Max. CPU Frequency (MHz)	3.54	2.14	2.14	**2.139**	2.151	2.151	2.152	2.344	2.145	2.294	2.411	2.342	2.324	2.159
Avg. RAM Memory USS (B)	10.195	**6.822**	9.973	9.854	9.953	9.846	9.875	9.875	10.021	9.9	9.91	10.07	9.911	10.075
Std. Dev. RAM Memory USS (B)	0.02	0.018	0.02	0.027	0.013	0.029	0.013	0.014	0.02	0.021	0.013	0.012	**0.011**	0.018
Max. RAM Memory USS (B)	16.824	**10.348**	16.469	16.332	16.492	16.348	16.285	16.328	16.586	16.375	16.387	16.609	16.344	16.617
Accuracy	1	0.996	1	1	1	1	0.984	1	1	1	1	1	0.997	1
Balanced Accuracy	0.995	0.5	0.995	0.995	0.995	0.995	0.987	0.995	0.995	0.995	0.995	0.995	0.708	0.995
F1 Score	1	0.993	1	1	1	1	0.989	1	1	1	1	1	0.997	1

[a] Best optimization strategy according to the energy efficiency profile.
[b] Best optimization strategy according to the performance profile.
[c] Best optimization strategy according to the balanced profile.
Underlined values: Top three optimization strategy that resulted in the best improvement in the measured statistic.
Bold and underlined values: Optimization strategy that resulted in the best improvement in the measured statistic.
Double underlined values: Optimization strategy that results in a model with performance above the minimum acceptable performance threshold.
J: Joules. W: Watts. B: Bytes. MHz: Megahertz

Table 7. Size of the Compressed (Deflate) Model File in Persistent Storage Obtained with Each Optimisation Strategy.

Opt. Strategy Id.	0	1	2	3	4	5	6+⋆	7	8	9	10	11	12	13
Model Size (B)	13194	12415	45977	12722	27929	12028	**1660**	10992	27917	12326	11345	27909	1662	27886
Accuracy	1	0.996	1	1	1	1	0.984	1	1	1	1	1	0.997	1
Balanced Accuracy	0.995	0.5	0.995	0.995	0.995	0.995	0.987	0.995	0.995	0.995	0.995	0.995	0.708	0.995
F1 Score	1	0.993	1	1	1	1	0.989	1	1	1	1	1	0.997	1

+ Optimization strategy that minimizes model size.
⋆ Optimization strategy that minimizes model size and satisfies the minimum acceptable threshold as measured by the user-defined performance target metric.
Underlined values: Top three optimization strategy that resulted in the best improvement in the measured statistic.
Bold and underlined values: Optimization strategy that resulted in the best improvement in the measured statistic.
Double underlined values: Optimization strategy that results in a model with performance above the minimum acceptable performance threshold.
B: bytes.

3.5.1.5.3 Analysis of the Training, Inference, and Load Times

The results obtained in the training, loading, and inference stages of the optimised models are summarised in **Table 8**. The results shown in this table correspond to the mean, standard deviation, and maximum time (in seconds) obtained at each stage over five iterations for each optimisation strategy. The most important result that can be observed in this table is the reduction of the inference time of the optimised models with respect to the baseline model. Achieving an improved time for inference is essential from a scalability point of view when the system must process large amounts of data in real time. In this context, increasing the number of predictions the system can perform per second can have a significant impact on system performance, reducing the need to add more resources to the system to meet the real-time deadline required by the application, and thus reducing infrastructure costs and power consumption.

Table 8. Average, Standard Deviation and Maximum Times for Training, Inference, and Load Obtained for Each Optimisation Strategy.

Opt. Strategy Id.	0	1	2	3	4	5	6⋆	7	8	9	10	11	12+	13
Avg. Inference Time (s)	0.318	0.273	0.157	0.584	0.159	0.28	0.088	0.278	0.159	0.286	0.287	0.161	**0.084**	0.164
Std. Dev. Inference Time (s)	0.007	0.005	**0.002**	0.012	0.007	0.006	0.004	0.008	0.01	0.018	0.017	0.006	0.006	0.009
Max. Inference Time (s)	0.329	0.281	0.158	0.603	0.166	0.292	0.092	0.288	0.177	0.322	0.321	0.171	**0.09**	0.179
Avg. Load Time (s)	0.086	0.062	0.065	0.063	0.065	0.071	0.064	0.069	0.061	0.064	**0.06**	0.062	0.062	0.063
Std. Dev. Load Time (s)	0.005	**0.002**	**0.002**	**0.002**	0.003	0.015	**0.002**	0.015	0.006	0.004	0.003	0.003	0.006	0.003
Max. Load Time (s)	0.092	0.066	0.068	0.066	0.071	0.101	0.067	0.098	0.072	0.069	**0.065**	0.067	0.073	0.068
Accuracy	1	0.996	1	1	1	1	0.984	1	1	1	1	1	0.997	1
Balanced Accuracy	0.995	0.5	0.995	0.995	0.995	0.995	0.987	0.995	0.995	0.995	0.995	0.995	0.708	0.995
F1 Score	1	0.993	1	1	1	1	0.989	1	1	1	1	1	0.997	1

+ Optimization strategy that minimizes average inference time.
⋆ Optimization strategy that minimizes average inference time and satisfies the minimum acceptable threshold as measured by the user-defined performance target metric.
Underlined values: Top three optimization strategy that resulted in the best improvement in the measured statistic.
Bold and underlined values: Optimization strategy that resulted in the best improvement in the measured statistic.
Double underlined values: Optimization strategy that results in a model with performance above the minimum acceptable performance threshold.
s: seconds.

3.5.1.5.4 Analysis of the Energy Cost of the Optimisation Strategies

An interesting point to analyse is the energy consumption required to train the model that is the object of the optimisation process and the time required to apply the optimisation strategy on it. In the results shown in **Table 9**, it is observed that the optimisation strategies that present a higher energy cost are those that involve a training procedure. That is, quantization-aware model fine-tuning, knowledge distillation and pruning-aware model fine-tuning, in that order of increasing energy cost. This is because optimisation strategies involving a training procedure require more energy to process the data, since they involve a more complex set of operations on the data set. Interestingly, knowledge distillation is not the most expensive optimisation strategy among these techniques, since training the student model requires less energy than training the full model, as the number of parameters is smaller. On the other hand, strategies that do not require a training procedure (such as post-training weight quantization) require less energy because the data are already processed, and the only task required is to modify the model to achieve the desired optimisation objective. This is an important consideration when designing systems that must be periodically retrained to incorporate new knowledge or learn from new data when the statistical properties of the data or the relationship between input and output variables change over time, since the cost of applying the optimisation strategy will occur each time retraining is necessary.

Table 9. Total Average Energy Consumption for Multiple Executions Over the Duration of the Experiment for Baseline Model Training with the Application of Optimisation.

Opt. Strategy Id.	0	1	2+*	3	4	5	6	7	8	9	10	11	12	13
Total Avg. CPU Energy Consumption (J)	**275.193**	418.206	300.446	535.77	353.696	498.46	654.814	830.898	1035.579	1008.426	1362.88	372.319	766.375	873.113
Accuracy	1	0.996	1	1	1	1	0.984	1	1	1	1	1	0.997	1
Balanced Accuracy	0.995	0.5	0.995	0.995	0.995	0.995	0.987	0.995	0.995	0.995	0.995	0.995	0.708	0.995
F1 Score	1	0.993	1	1	1	1	0.989	1	1	1	1	1	0.997	1

+ Optimization strategy that minimizes total average energy consumption.
* Optimization strategy that minimizes total average energy consumption and satisfies the minimum acceptable threshold as measured by the user-defined performance target metric.
Underlined values: Top three optimization strategy that resulted in the best improvement in the measured statistic.
Bold and underlined values: Optimization strategy that resulted in the best improvement in the measured statistic.
Double underlined values: Optimization strategy that results in a model with performance above the minimum acceptable performance threshold.
J: Joules.

3.5.1.5.5 Summary of the Results Obtained

The main conclusions and observations of the results obtained in the experimental evaluation are summarised below.

- The knowledge distillation technique provides the largest reduction in energy consumption in most cases studied, reducing the total average energy consumption by up to 82.304% with a minimal performance degradation of just 0.08% in the balanced accuracy, 0.016% in the accuracy, and 0.11 in the F1 score.
- The optimisation strategy based on the application of the half-precision floating-point weight quantization provides a reduction in energy consumption of up to 58.208%, with no performance degradation compared to the baseline model.
- Some of the techniques studied are not mutually exclusive and can be applied in conjunction with each other to further reduce the energy consumption of the model. In this regard, applying a weight quantization as final post-processing can potentially reduce the energy

consumption of the model in the inference stage. In particular, we observed that the optimisation strategy based on the application of the knowledge distillation technique followed by a half-precision floating-point weight quantization provides a reduction in energy consumption of up to 80.741%, with a negligible performance degradation of 0.08% in the balanced accuracy, 0.016% in the accuracy, and 0.11 in the F1 score. However, we noticed that this technique provided an improvement in energy efficiency only when knowledge distillation was applied first, but this was not always the case, since in some cases the reduction in energy consumption is not appreciable. For this reason, the cost-effectiveness of applying a post-training weight quantization technique should be evaluated on a case-by-case basis.

- Related to the above, since pruning makes a significant portion of the model weights zero, if a post-training quantization technique is subsequently applied, it will only reduce the model size by a small factor and, in general, will do little to increase the model efficiency already achieved by applying the pruning technique.

- Another technique that can be applied in conjunction with the knowledge distillation technique to further reduce the energy consumption of the model is the pruning-aware model fine-tuning. The optimisation strategy based on the application of the knowledge distillation technique followed by a pruning-aware model fine-tuning provides a reduction in energy consumption of up to 81.046% but with a significantly higher performance degradation of 0.287% in the balanced accuracy. In addition, in some preliminary tests, we observed that performance degradation was unpredictable, as the same optimisation strategy was applied on different occasions and provided different performance results. For this reason, we recommend caution when applying this optimisation strategy.

- Regarding quantization-aware model fine-tuning, when this technique is applied as a single optimisation step, it provides a reduction in energy consumption of up to 15.05%, with no performance degradation. Although our results confirm the effectiveness of this technique in improving model energy efficiency when performance degradation is not permissible, the improvement in energy efficiency obtained is comparatively lower than the one obtained with other techniques, such as knowledge distillation or pruning-aware model fine-tuning. In addition, when this technique is applied as part of a two- or three-step optimisation strategy in which knowledge distillation or the pruning-aware model fine-tuning technique are applied first, we observed that it did not result in a reduction in energy consumption.

- The best performing post-training quantization is half-precision floating-point weight quantization, followed by 8-bit integer weight quantization. Quantization of integer weights with 16-bit integer activations and 8-bit integer weights significantly increases power consumption due to the unavailability of kernels optimised for the type of operations needed to perform inference using the model produced with this quantization scheme in TFLite, being much slower than with the original model (1.84x slower) or with other quantization schemes, such as 8-bit integer weight quantization (2.1x slower) and half-precision floating point (3.72x slower). However, this problem could be solved in the future when support for cores optimised for this quantization scheme is added in TFLite.

- The combination of knowledge distillation and pruning-aware model fine-tuning yields a more energy-efficient model with respect to the baseline but does not always improve the boost in energy efficiency that is obtained from the application of knowledge distillation.

- The results show that, regardless of the batch size used for inference, an optimal balance between energy and accuracy can be obtained with the application of the knowledge distillation technique, as it provides a very high energy savings with minimal degradation of performance. The results also demonstrate that, when performance degradation is not allowed, the optimisation strategy based on the application of the half-precision floating-point weight quantization provides the best energy-consumption results.

- The difference in energy consumption reduction provided by optimisation strategies using a large batch size and a small batch size does not result in a variation in the relative ranking of the optimisation strategies. In most cases, optimisation strategies that provide the greatest reduction in energy consumption for small batch sizes also provide the greatest reduction for large batch sizes, and when this is not the case, the difference in energy savings is small. However, it is clear from the results that the reduction in energy consumption obtained with the application of optimisation strategies is greater for larger batch sizes, regardless of the optimisation strategy applied. From this observation, we can conclude that while it is true that larger batch sizes provide higher energy efficiency compared to the non-optimized model using the same batch size, there may be cases where smaller batch sizes actually consume less energy. For example, if the non-optimized model using a large batch size requires substantially more energy than the non-optimized model using a small batch size, then the reduction obtained with an optimization strategy on the large batch size may lead to a net energy consumption that is actually greater than the net energy consumption obtained with the same optimization strategy on the small batch size. That is, although larger batch sizes typically provide greater energy efficiency, there may be cases where smaller batch sizes are preferable for optimal energy savings, and therefore the choice of batch size should be carefully considered to ensure that net energy consumption is minimized.

- Knowledge distillation allows a great reduction in the model size and almost completely avoids performance degradation. In our case, it even allowed further reduction of the model size, although the performance becomes very inconsistent across different trials, failing to correctly predict the minority class on some occasions. However, this might not be a problem for model-specific optimisation, which will certainly compensate for the cost of the retraining needed to obtain a good model in the medium term. On the other hand, we have found that when training a model of the same size as the one used to perform the knowledge distillation technique, the model performs poorly. This allows us to conclude that knowledge distillation allows the use of very small models that would otherwise not be usable due to lower than acceptable classification accuracy and therefore could not be implemented in production. When the temperature factor is increased until the optimum point is reached progressively, the performance of this model is observed to increase, nearly approximating the performance obtained with the baseline (teacher) model.

- An important detail is that in the case of using an unbalanced dataset, it is necessary to pay special attention to the choice of hyperparameters used to apply these techniques. In our case, we have found that pruning-aware model fine-tuning and quantization techniques are particularly sensitive to the choice of these values when dealing with unbalanced datasets. In that case, if the hyperparameter values are not chosen carefully, performance degradation after optimisation can be severe. We have found that when the classes in the dataset are balanced, these techniques are more robust against the choice of these hyperparameters than in the case of using an unbalanced dataset, since the performance is generally maintained above the minimum acceptable through different retrainings. Moreover, the case of

knowledge distillation is similar. The model can be reduced by a larger factor if the number of samples in each class is balanced, but if it is not, the model can be reduced less with respect to the original model. In the latter case, if the model is reduced excessively, the accuracy in predicting the negative class will drop to a random estimate or, in the worst case, to zero (i.e., the model overfits to predict the majority class), although, in some cases, it is still possible to find a local minimum in some training attempt to obtain acceptable accuracy for both classes. However, the likelihood of this occurring decreases as the model is reduced further.

3.5.2 Generation of Black Box-Based Adversarial Examples to Produce Resilient CADs Based on DNNs

In this section we summarise the motivation and design of the solution emphasising its novelties and the results obtained in the evaluation. The detailed results of this research can be found in “Crafting Black-Box Adversarials to Attack Machine Learning-Based Systems with GANs” [21] and “Improving the quality of generative models through Smirnov transformation” [22].

3.5.2.1 Motivation

The rapid adoption of ML models in many different tasks in the industry has made them an attractive target for malicious individuals. In this regard, in recent years, adversarial attacks have highlighted the weakness of state-of-the-art ML techniques in terms of robustness and generalisation, inspiring malicious adversaries to exploit this weakness to attack systems that integrate ML models at the core of their decision-making process to achieve their purposes. More specifically, adversarial attacks are referred to as a type of attack in which an attacker deliberately manipulates the inputs to an ML model to generate a particular output or cause it to misbehave. This can be done by adding carefully crafted perturbations to the inputs, which are often imperceptible to humans. The malicious samples obtained, more often referred to as Adversarial Examples (AEs), can be used to cause a model to misclassify an input or to cause the model to classify the provided samples with an arbitrary label according to the purpose of the attacker. In addition, AEs can also be crafted to leak sensitive information that is encoded in the model weights. These concerns have been shown to be a general problem for many ML models, including DNNs [21].

Adversarial attacks have become a serious threat to the security of ML models in recent years; the lack of robustness against AEs has raised serious concerns in security-critical applications, such as facial recognition and fraud detection, due to the possibility of these attacks being used for malicious purposes. As a result, the situation has become increasingly more demanding for system designers, who are very often faced with the challenge of defending these critical systems against these types of attack.

In order to design robust ML models, in recent years, researchers and practitioners have started to investigate methodologies and techniques to elaborate effective adversarial attacks and develop defences against them. This never-ending "arms race" has resulted in a large and growing body of literature on adversarial attacks and defences. In general, the proposed approaches can be divided into two groups: 1) those that aim to find adversarial examples, and 2) those that focus on characterising the behaviour of an adversarial attack in order to develop techniques for detecting or mitigating them.

Moreover, hand-crafted attacks based on manually defined heuristics are becoming less and less effective as ML models become more sophisticated. In order to keep up with the state-of-the-art in ML, it is necessary to develop more advanced automated methods for crafting AEs. In this regard,

methods that relay on gradient-information are not practical as they require access to the training data and the model parameters used to train the model (i.e., white-box access). This assumption is not realistic in practice; as in real-world settings, data may be proprietary, and model parameters may be hidden, often only accessible via an API. This situation is typically termed a "black-box" setting where the attacker only has access to the model output and has to design an attack without knowing the model architecture or parameters or the training dataset that was used for model learning. Several methods have been proposed to craft AEs in black-box settings. A first direction is the transferability of AEs: that is, the ability of an AE crafted to fool a given model to deceive another model that was trained on a different data set or is based on a different architecture. With this approach, key details such as the choice of loss function or the presence of specific architectural elements in the black-box model, which are often unknown to the attacker, are not needed for the attack to succeed [23]. Taking advantage of this property, surrogate models have been proposed to fit a model so that it behaves as closely as possible to the target black-box model. With this approach and given that the transferability property can be satisfied, AEs that fool the target model can be found using the surrogate model instead of the target model itself [Pap2016b]. A second direction is the use of Gradient-Free Optimization [23] or Zeroth-order Optimization [24], which uses query-based optimisation algorithms to optimise a loss function without requiring the model to be differentiable as gradient-based methods do [25], thus being also applicable to non-differentiable models, such as decision trees or random forests.

A more promising approach is the use of Generative Adversarial Networks (GANs) to generate AEs in a black-box environment. GANs are a generative model consisting of two neural networks, a generator and a discriminator. The generator is trained to generate data samples that can fool the discriminator into believing that they are real data samples. On the other hand, the discriminator is trained adversarially to distinguish between real and generated data samples. GANs have been shown to be capable of generating data samples that are indistinguishable from real data samples. This technology was initially applied to craft AEs to maximise the probability of misclassification of a DNN-based image classifier. Since then, GAN-based attacks have been successfully applied to a variety of ML models, including but not limited to DNNs, Support Vector Machines (SVM), and Logistic Regression (LR) models. Several GAN-based solutions have been proposed recently, and among them MalGAN [26], AdvGAN [27], and AdvGAN++ [28] can be considered among the best performing representatives.

MalGAN is a GAN-based framework for generating AEs in a black-box environment to fool malware detection models that can be integrated into antivirus software products or provided as an API for security-related services. In MalGAN, the discriminator acts as a surrogate model, trained to distinguish between real and malicious samples, while the generator is trained to generate AEs from real malicious samples that can fool the discriminator. At the time of publication, MalGAN was shown to be able to generate targeted and untargeted AEs to compromise state-of-the-art malware detection models. However, one of the main drawbacks of MalGAN is that it does not generate AEs that are statistically indistinguishable from real malicious samples, implying that MalGAN-generated AEs can still be easily detected by human validators and statistical filtering methods.

On the other hand, AdvGAN++ is a white-box GAN-based framework for generating targeted and untargeted EAs. AdvGAN++ was proposed as an improvement of AdvGAN which proved to be able to provide a higher attack success rate than other relevant attack strategies. Furthermore, AdvGAN was shown to be applicable without knowledge of any defences that may have been applied to the target model (e.g., adversarial training or model distillation), being able to preserve high perceptual quality and maintaining a higher attack success rate than other attack models when applied to attack models hardened with state-of-the-art defences. To enable this attack in black-box scenarios, AdvGAN++

proposes to train distilled models that approximate the decision boundary of the target model using randomly sampled data pairs drawn from a dataset that is disjoint from the one used to train the target black-box model. This distilled model is then used to train the GAN model to perform a white-box attack. Specifically, the GAN generator is trained to generate AEs from real malicious samples and a random noise vector that can fool the distilled model. This is achieved by using a penalty-based objective function to maximise the probability that the generated malicious samples are misclassified by the distilled model. The discriminator is trained to distinguish between real and generated malicious samples so that it can provide useful information to the generator. In addition, the L2 loss between the generated AEs and the original malicious samples is added to the GAN objective function to minimise the number of perturbations of the generated AEs with respect to the real malicious data. Similar to MalGAN, AdvGAN++ is able to generate AEs that can successfully deceive the black-box objective model with a high success rate. However, it fails to generate AEs that are statistically indistinguishable from real malicious samples. For this reason, the AEs generated by AdvGAN++ can be trivially identified by human experts or filtered by other statistically based approaches.

Unfortunately, all the solutions analysed, although providing very high avoidance rates (in some cases, higher than 99%), generate very poor-quality adversarial data, both if we analyse the statistical distribution of the values of each feature and the joint distribution of them all. The proposed solution that we applied in the ML engine of the CAD component tries to alleviate this shortcoming by obtaining adversarial data generators that achieve similar evasion values to the current solutions, but which generate very high-quality data whether we analyse the features separately or their joint distribution.

3.5.2.2 Solution

In task T4.1 we have researched the use of GANs to generate AEs that can fool a target black-box model. To this end, we developed a model-agnostic solution that is able to greatly improve the quality of results obtained with MALGAN one of the state-of-the-art models for generating AEs. First, we describe the architecture and next, we detail the two novelties that allow the solution to significantly increase the quality of the adversarial examples generated by the GAN. Using these high-quality AEs we retrained the ML-based engine of the CAD component to be resilient against these sophisticated attacks.

3.5.2.2.1 System Architecture

The black-box model we use is an external system and can be installed with different ML-based traffic detection algorithms. In our experiments we used a model based on a feed-forward DNN model composed of 5 layers, but other models (e.g., Random Forest) can also be adopted. Since we are operating in a black-box environment in supervised learning mode, we will assume that the black-box model can be queried in an unbounded way. That is, for each query, a feature vector representing the queried sample can be provided as input to the model, and the model will return as output the predicted class corresponding to that sample. Therefore, we assume that we have no access to the parameters of the model, the type of algorithm on which it is based, and the probability of the class predictions produced.

Our GAN model is inspired by the standard GAN architecture proposed by Goodfellow et al. [Goodfellow20]. This GAN architecture consists of two main components: the generator and the discriminator. In our case, the discriminator is used to model a third component, the unknown black-model target. We refer to the discriminator in this specific setting as a substitute model, as it will try to learn the black-box behaviour. As a consequence, this configuration implies a higher complexity in the training process than the standard GAN as we must also track the behaviour of a given classifier that will act as a black-box model during the training phase.

Figure 12 shows an overview of the GAN system based on MalGAN, with three defined square boxes. Each box contains an ML model that is producing predictions, and each circle contains input or output data tensors. These boxes are from left to right: (i) the generator, which is the DNN to be trained for AE generation, (ii) the black-box detector, which is the model that is the target of the attack, and (iii) the substitute model, which will try to learn the behaviour of the target model and will also serve as a trainer for the generator to learn how to produce effective AEs.

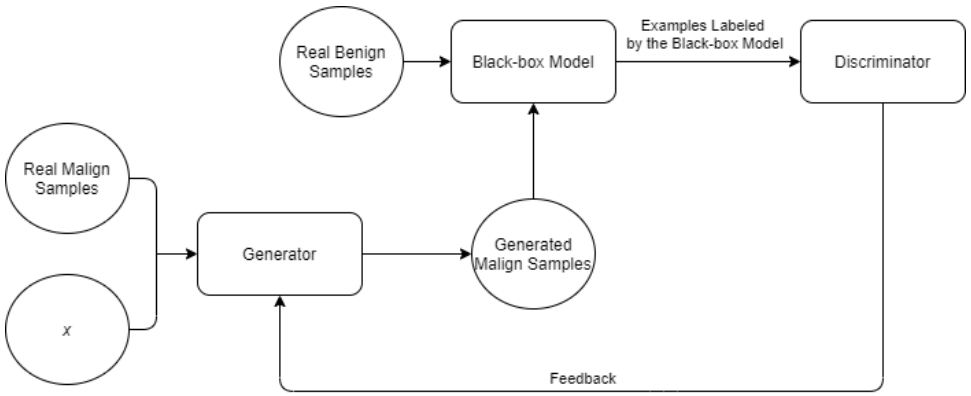

Figure 12. Overview of the GAN Solution Based on MalGAN.

In this figure, the components are interconnected by arrows according to the scheme indicated by the MalGAN, which is as follows: The generator takes as input a latent vector of random noise and real malign samples *x* that were randomly chosen without replacement and outputs synthetic AEs of the malicious class. The generator is trained by the surrogate model, which takes as input the synthetic AEs and outputs the class predictions. These predictions provide an error signal that is used as feedback for generator learning. On the other hand, the discriminator is trained by the black-box detector, which takes as input the synthetic AEs and the real benign samples that were randomly chosen without replacement and labels them. The same input is then provided to the discriminator. The predictions generated by the discriminator are compared with those produced by the black-box detector, and the error is minimised by gradient descent. The number of real malign and benign samples that are randomly chosen from the dataset during the described procedure is the same. The total number of samples depends on the batch size to be used to train the GAN model.

3.5.2.2.2 Novelties of the Design

It is well known that measuring the performance or quality of GANs is a difficult and challenging task in general, especially when the intention is to guide the training procedure to generate AEs that are statistically indistinguishable from real malicious samples while maximising the variety of generated AEs and their evasion rate.

To address this issue, we define a new distance metric that is capable of measuring the ability of a GAN to generate AEs that are statistically indistinguishable from real malicious samples, thus preserving their malicious features while maintaining sufficient resemblance to real malicious samples to fool a target black-box model. More specifically, we define an approximation to the Wasserstein distance for N-dimensional distributions that is capable of measuring the statistical distance between two distributions based on an iterative process to compute the distance between samples from each distribution. Furthermore, we have incorporated this statistical distance measurement into the generator cost function during its training phase, allowing the gradient to be directed toward generating examples that preserve their malicious nature while maintaining a benign appearance that causes them to be misclassified by the black-box model.

In addition, we use a custom activation function based on the Smirnov-Transform (ST) as the last layer of the generator to help generating AEs that mimic the behaviour of real malicious samples, transforming the generator output variables into variables that are distributed exactly the same as the input variables. The ST activation function is fully deterministic and differentiable, which allows to be seamlessly integrated into the backpropagation step during the GAN training processes. In [31], we demonstrated the significant improvement provided by this custom activation function in terms of the quality of the generated samples.

3.5.2.3 Evaluation

In this subsection, we detail the experimental setup we followed to perform our experiments. Next, we define the methodology used to evaluate the effectiveness of each of our proposed methods. Finally, we show the results obtained in our experiments and discuss the main conclusions.

The data of our experiments was obtained from a realistic cryptomining attack scenario [Pastor-crypto] deployed in the Mouseworld lab, a controlled environment set up in Telefónica I+D premises. This scenario was composed of typical Internet applications (e.g., web browsing, multimedia, file access via shared folders) and cryptomining clients interacting with real mining server pools spread across the Internet. The TCP connections were captured and analysed by a modified version of Tstat, a traffic analysis tool, in order to extract in real-time a set of features derived from several connection statistics that serve as the input features for our machine learning models. Then, we added an ad-hoc code to automatically assign a binary label to each TCP flow instance (0: normal traffic connection, 1: cryptomining connection).

In order to prove the effectiveness of our approach, a set of four experiments was designed to empirically test the efficiency of each of the proposed methods. These experiments are defined as follows.

- Experiment 0: MalGAN: vanilla version as defined by its authors. This experiment serves as baseline for later comparison with the rest of the experiments.
- Experiment 1: MalGAN with a Smirnov Transform activation function as the output of the last layer of the generator.
- Experiment 2: MalGAN adding the Wasserstein distance to the cost function.
- Experiment 3: MalGAN with a Smirnov Transform activation function as the output of the last layer of the generator and the Wasserstein distance added to the cost function.

These experiments allow us to understand how the generation of AEs is affected when we use a Smirnov Transform activation function or when we add our proposed distance in the generator loss calculation. A rich set of hyperparameters were applied to train the GAN following a random search heuristic. Furthermore, we performed a grid search procedure to tune the ratio parameters of the loss function and the L2 regularisation factor of the generator and discriminator models that we incorporated in experiments 2 and 3.

In the evaluation phase we used two complementary metrics at the end of each epoch to analyse and compare the performance of the models. First, we used an approximation to the Wasserstein distance to calculate the distance between benign and maligns examples and synthetic adversarial examples. In addition, we plotted a histogram of each feature of samples drawn from the malign and synthetic adversarial examples to check that data distributions are similar. Furthermore, we calculated the evasion rate of the synthetic adversarial examples when input to the black-box model. The evasion rate is the ratio of adversarial examples that are incorrectly classified as benign examples by the black-box model.

The goal of our experiments was to generate adversarial examples that are as close as possible to the malign examples but obtaining a decent evasion rate. It is worth noting that we have experimentally observed that current GAN proposals (e.g., MalGAN) tend to obtain a very good evasion rate, but generate adversarial examples that are very far from real examples (benign and malign). In this way, a simple filter can be built to detect and discard such adversarial examples before they are input to the black-box model. In sharp contrast, our GAN solution generates high quality adversarial examples that are very difficult to distinguish from real malign examples and therefore, they can be used to fortify the black-box model against this kind of sophisticated attacks.

3.5.2.3.1 Summary of the Results

We present in this section the main results obtained in the four experiments. For each experiment we plot the distances between all data distributions, the evasion rate of the synthetic generated malign examples, and the histogram of each feature to observe the quality of the synthetic malign examples.

The distance plot shows how close the synthetic malign examples are from the real malign data points (green line), how far the synthetic malign examples are from the real benign examples (orange line), the distance between real benign and malign examples (blue line), and the residual distance between two malign samples (red line) as baseline metric. The goal is to generate synthetic maligns that are very close to the real maligns (green line as close as possible to the red line), and as far from the benign examples as the real maligns are from them (orange line as close as possible to the blue line).

The evasion rate curve shows the ratio of synthetic malign examples that fool the black-box model. The goal of our experiments is to obtain, at the same time, a good evasion rate and synthetic examples of high enough quality that they will be indistinguishable by tools and human experts. Note that, current proposals (e.g., MalGAN vanilla in experiment 0) tend to obtain very good evasion rates, but very poor-quality synthetic examples, which precludes them from use in realistic scenarios as they can be easily detected and filtered with simple tools.

The first 650 epochs were calculated and plotted in the distance and evasion rate figures.

In addition to these metrics, a comparative histogram of the four features of two samples drawn in the 640-th epoch from real malign examples and synthetic adversarial malign examples is presented. These histograms graphically show whether the synthetic data distribution matches the real malign data distribution. Histogram bars in blue represent the frequency of real malign values and the brown ones correspond to the frequency of synthetic adversarial examples.

In the rest of the section, we analyse the results obtained in each experiment.

Experiment 0: MALGAN (vanilla version): No Smirnov Transform-based activation or distance functions are implemented

In Figure 13 we plot the evolution of distances between the involved examples: benign, malign and adversarial examples. In this experiment MALGAN vanilla models were trained using an FCNN black-box model and did not implement the Smirnov Transform-based activation function at the last layer of the generator nor the approximation of the Wasserstein distance in the cost function. Furthermore, Figure 14 shows the evasion rate of adversarial examples (BB_hits) when input to the black-box model.

In experiment 0 (MalGAN vanilla) we can clearly observe the problem that motivated our research. Although the evasion rate is very close to 1 (100% of synthetic adversarial examples fool the black-box classifier), the synthetic data that has been generated by the MalGAN is very far from the malign data that MalGAN tried to replicate (green line). Furthermore, the distance from the generated adversarial examples to the benign data is closer than to the malign. Therefore, it seems that the MalGAN model

is generating synthetic data from a different distribution that, because it is relatively close to the benign examples, can fool the black-box model easily. In fact, the adversarial examples are not fooling the black-box model as they are very similar to benign examples and therefore the black-box classifier is doing the right thing as it is classifying as benign examples that are very close to the real benign. This lack of quality in the replication of malign data can be observed with more detail in the four histograms of
Figure **15** where the brown bars (generated adversarial examples) do not overlap in any case the shape of the malign examples' distribution (blue bars).

The rest of experiments (1, 2, and 3) show the benefits of the proposed solution providing adversarial generators that achieve evasion rates near to 1 with synthetic examples of high quality.

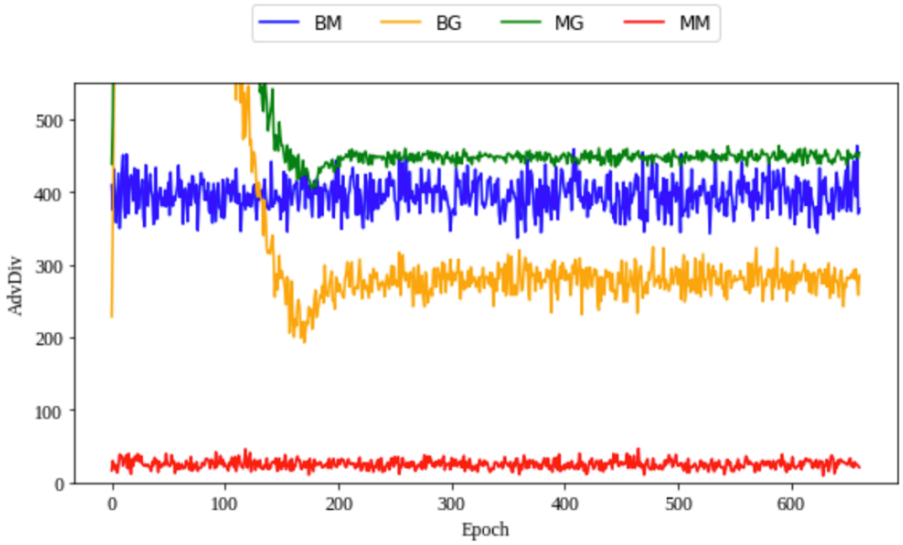

Figure 13. Experiment 0. MalGAN (vanilla version). Distances between benign and malign examples (BM, blue), malign and generated adversarial examples (MG, green), benign and generated adversarial examples (BG, orange) and two samples of malign examples (MM, red).

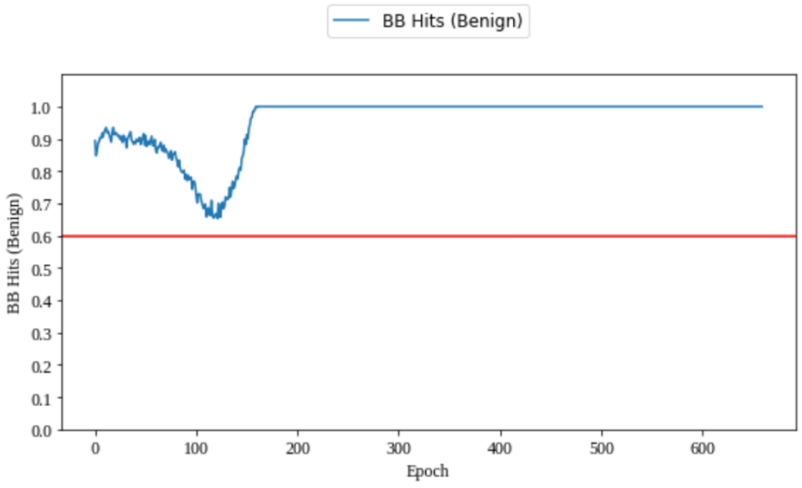

Figure 14. Experiment 0. MalGAN (vanilla version). Evasion rate (BB Hits) of generated adversarial examples when input to the black-box model.

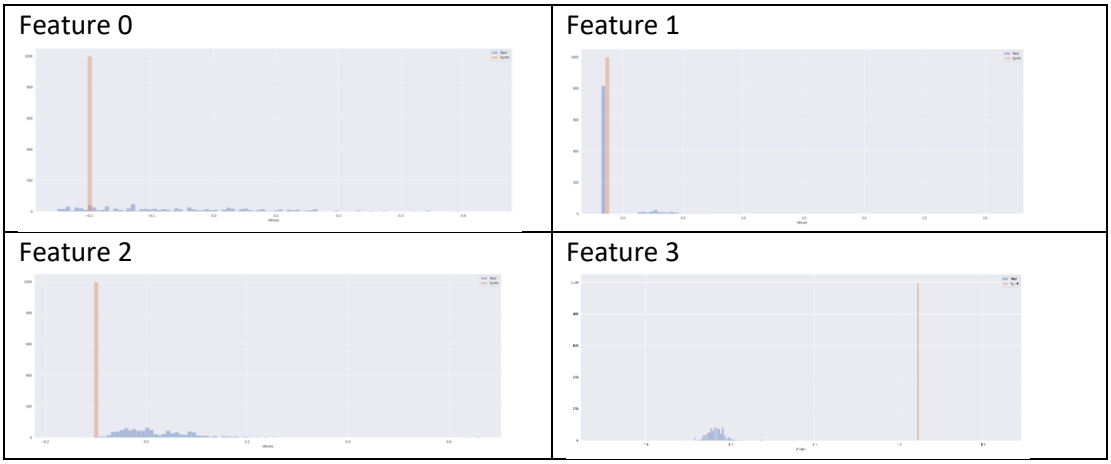

Figure 15. Experiment 0. MalGAN (vanilla version). Comparison of data distribution histograms of malign and adversarial examples for each data feature in epoch 670. Malign examples are represented in blue colour and generated adversarial examples are in orange.

Experiment 1: The Smirnov Transform-based activation function is added to the last layer of the MALGAN generator

In experiment 1, only the Smirnov Transform-based activation function was used in the last layer of the generator. The nice effect of this activation function is to force the four features of the output values to be distributed similarly to the real malign examples. This effect can be observed in Figure 16 where the synthetic and the real histograms are fully overlapped for all features. Furthermore, it can be seen in Figure 14 that this effect is also beneficial to obtain synthetic adversarial examples that replicate the real malign examples (green line is close to the red line and far from blue and orange lines that represent similar distances) and generate a decent evasion rate (with some oscillations during the training around the 200-th epoch).

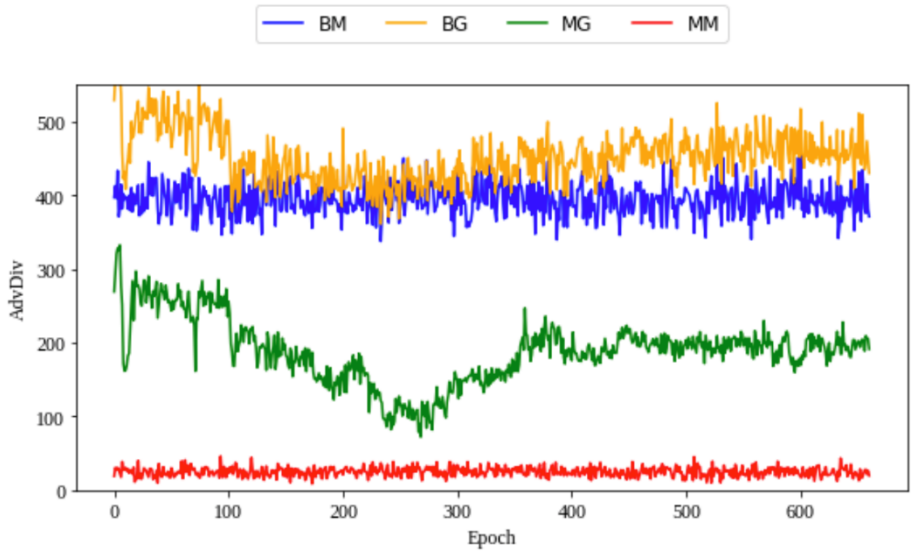

Figure 16. Experiment 1. MalGAN with Smirnov Transform-based activation function. Distances between benign and malign examples (BM, blue), malign and generated adversarial examples (MG, green), benign and generated adversarial examples (BG, orange) and two samples of malign examples (MM, red).

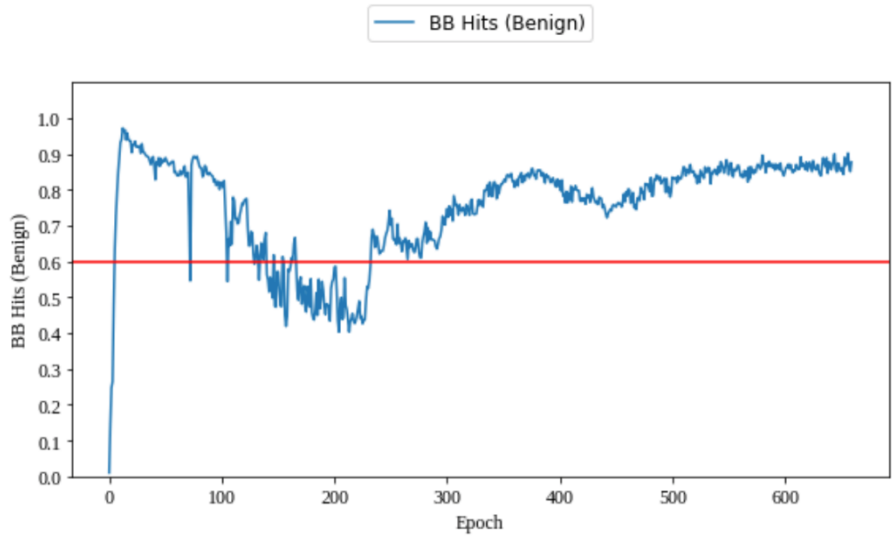

Figure 17. Experiment 1. MalGAN with Smirnov Transform-based activation function. Evasion rate (BB Hits) of generated adversarial examples when input to the Black-Box model.

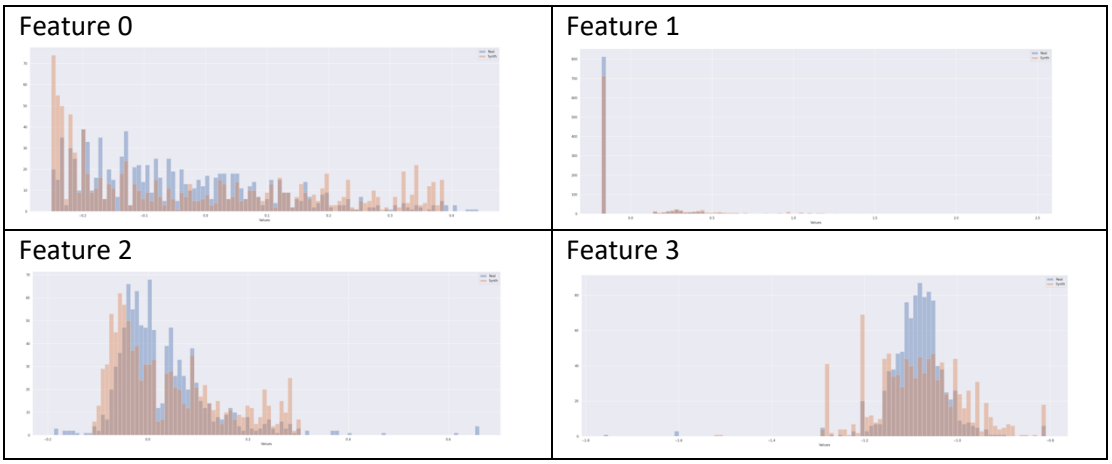

Figure 18. Experiment 1. MalGAN with Smirnov Transform-based activation function. Comparison of data distribution histograms of malign and adversarial examples for each data feature in epoch 670. Malign examples are represented in blue colour and generated adversarial examples are in orange.

Experiment 2: Wasserstein distance in the cost function of the MALGAN generator.

In experiment 2, only the approximation of the Wasserstein distance was applied to the GAN architecture and this effect produces oscillations in the distances during the training epochs, which indicates that the convergence of the training process is not as smooth as when activation functions force the adversarial values to be in the same range as the real maligns. In fact, the four histograms reflect that the synthetic and the real variables are not fully overlapped, which indicates different statistical distributions for them. However, the evasion rate stabilises at the maximum value (100%) in the very first epochs, although the distances show a changing behaviour reflecting that the first adversarial examples are very similar to benign examples, but after 150 epochs the generator starts creating adversarial examples close the malign data, but powerful enough to keep fooling the black-box model.

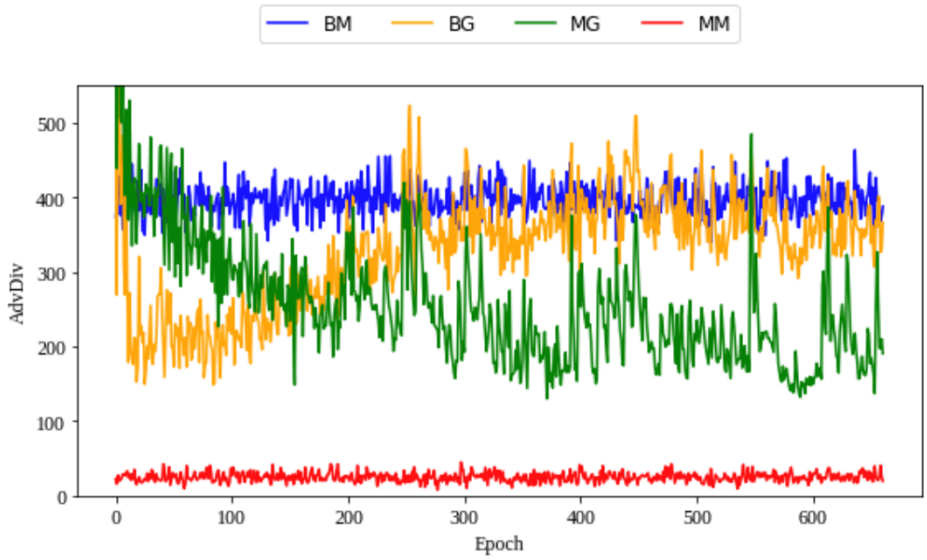

Figure 19. Experiment 2. MalGAN with Wasserstein distance in the cost function. Distances between benign and malign examples (BM, blue), malign and generated adversarial examples (MG, green), benign and generated adversarial examples (BG, orange) and two samples of malign examples (MM, red).

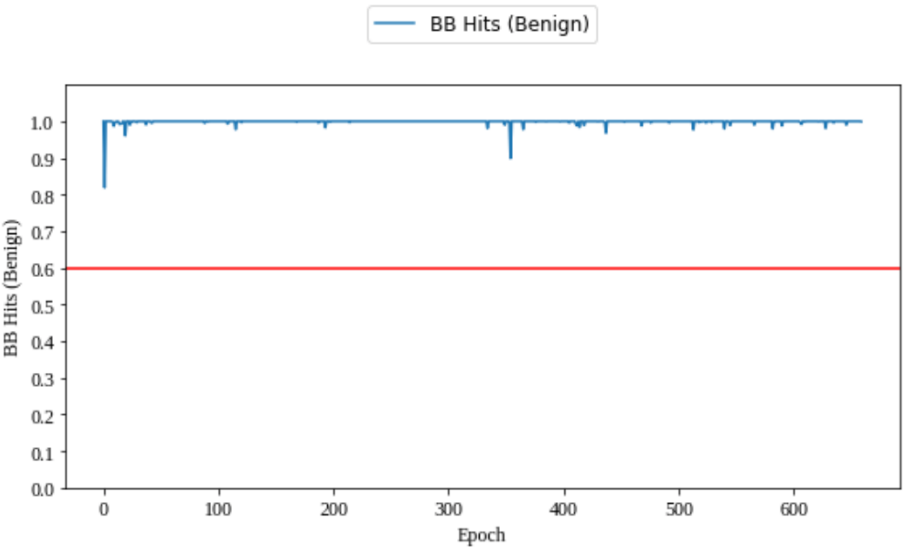

Figure 20. Experiment 2. MalGAN with Wasserstein distance in the cost function. Evasion rate (BB Hits) of generated adversarial examples when input to the Black-Box model.

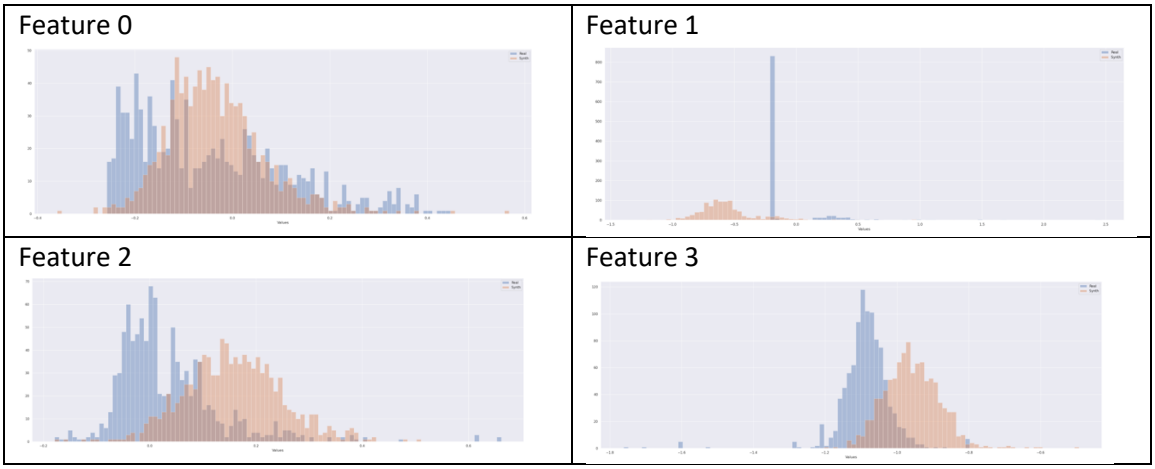

Figure 21. Experiment 1. MalGAN with Smirnov-based activation function. Comparison of data distribution histograms of malign and adversarial examples for each data feature in epoch 670. Malign examples are represented in blue colour and generated adversarial examples are in orange.

Experiment 3: The Smirnov Transform-based activation function is added to the last layer of the MALGAN generator and the approximated Wasserstein distance is used in the cost function of the MALGAN generator.

Finally, Experiment 3 combines the two techniques (Smirnov Transform-based activation and Wasserstein distance functions) producing the best results: The synthetic adversarial examples are getting closer to the malign examples as the training progresses and the evasion rate is stabilised near to 1 when the first 100 epochs are reached during the training process. Furthermore, the four histograms show a nearly perfect overlap of the data distributions of synthetic adversarial and malign examples.

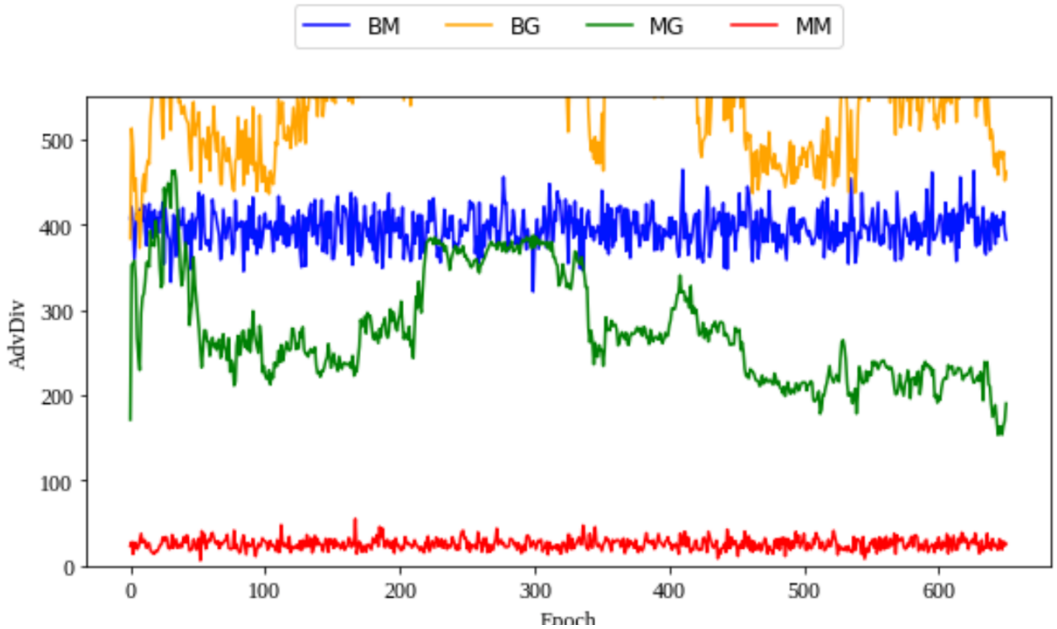

Figure 22. Experiment 3. MalGAN equipped with Smirnov Transform-based activation function and Wasserstein distance in the cost function. Distances between benign and malign examples (BM, blue), malign and generated adversarial examples (MG, green), benign and generated adversarial examples (BG, orange) and two samples of malign examples (MM, red).

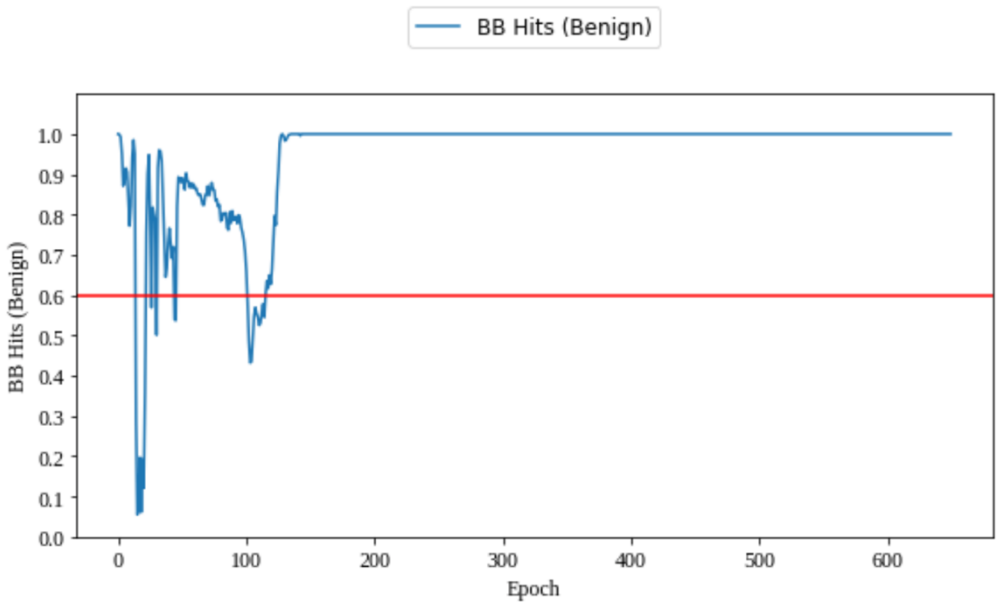

Figure 23. Experiment 3. MalGAN equipped with SmirnovTranform-based activation function and Wasserstein distance in the cost function. Evasion rate (BB Hits) of generated adversarial examples when input to the black-box model.

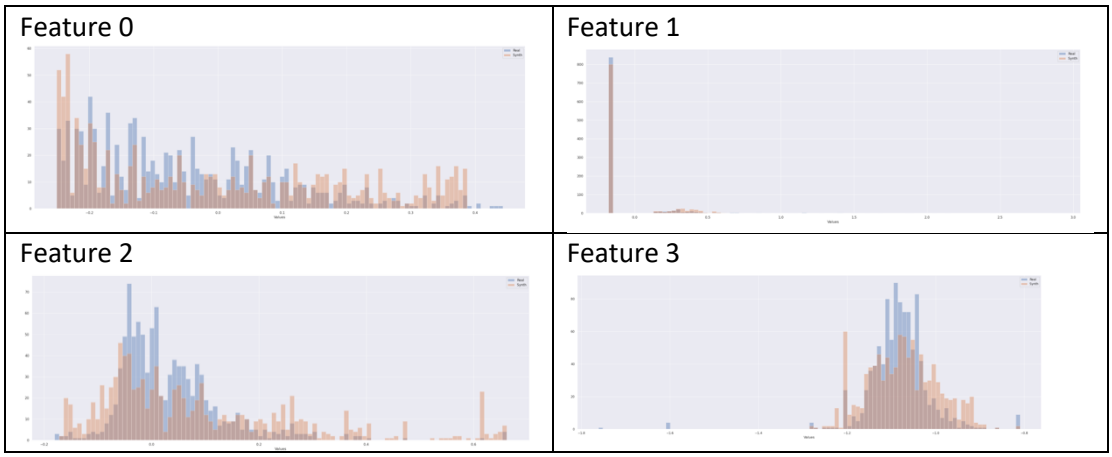

Figure 24. Experiment 3. MalGAN equipped with Smirnov Transform-based activation function and Wasserstein distance in the cost function. Comparison of data distribution histograms of malign and adversarial examples for each data feature in epoch 670. Malign examples are represented in blue colour and generated adversarial examples are in orange.

3.5.2.3.2 Defending Against Adversarial Examples

Once effective AEs have been constructed, the next step to strengthen the defences of the black-box model is to retrain it incorporating these AEs into the training dataset to increase its robustness against the attack. In this way, the decision boundary will be adjusted to correctly classify these AEs. However, it should be noted that it is still possible for an attacker to generate new AEs that can cause the model to behave improperly. Nevertheless, it should be more difficult for the attacker to produce new AEs that are effective using only small perturbations. Moreover, this situation will be more challenging as the black-box model continues to be retrained with these AEs in the future. However,

the addition of AEs to the training data set inevitably increases the variability of the data, which can hinder model learning and degrade model performance.

Another approach is based on label smoothing. This method consists of using the predictions of the black-box model to label the training data set and train a distilled model using this dataset [25]. The distilled model is trained using SoftMax activation instead of the classical sigmoid function to obtain more robust predictions that take into account the probability that a given input belongs to each class. Thus, the predictions of this distilled model are more spread out than those of the initial model, which reduces the overfitting of the model as its overconfidence is penalised. With this approach, each prediction takes into account the probability that the sample belongs to each of the classes, making it more difficult for an attacker to produce effective AEs that mislead the model. In addition, this SoftMax activation can be adjusted as a function of a factor called temperature. The temperature can be increased to obtain more uniformly distributed probabilities. Thus, increasing the temperature results in a more difficult situation for the attacker at the cost of a degradation in the performance of the model which may, itself, be considered a successful attack.

Although other approaches have been proposed in the literature to train the black-box model more robustly, such as input averaging or model ensemble, in this work we chose to focus on retraining the black-box model using high quality AEs to defend against the suggested threat model.

3.5.2.3.3 Summary of the Evaluation

- Current GAN techniques for the generation of adversarial examples for network traffic classifiers produce adversarial examples that, instead of being similar to malign examples, tend to be closer to the benign examples. This explains the very good evasion rate these techniques exhibit as we are inputting pseudo-benign examples instead of malign elements to the black-box classifier. Therefore, this approach is not appropriate in a real time deployment as simple filters can be programmed to identify these low-quality adversarial examples that are so different from the target malign data points.

- The combination of activation and distance functions in the GAN architecture produces a generator that generates synthetic adversarial examples that can fool a black-box classifier, nearly replicating the statistical distribution of the malign examples. The evasion rates achieved with this method are in the same range that the ones obtained by traditional methods (reaching 100%) and the synthetic adversarial and malign data distributions are statistically similar and both far from the benign examples, which makes their detection using filters or visual inspection by an expert more difficult. Although the Smirnov Transform-based activation function produces decent results in Experiment 1, the combination of both techniques in Experiment 3 produced the best-balanced performance.

- After high-quality adversarial examples are produced, the black-box model of the CAD component was retrained using these high-quality adversarial examples to create a resilient ML-based classifier that can defend itself against the suggested threat model (i.e., the retrained black-box model accuracy rose to 95%).

3.5.3 Generation of Synthetic Data Using GANs to Substitute or Augment Real Data for Training and Testing an ML-based CAD

In this section we summarise the motivation and design of the solution emphasising its novelties and the results obtained in the evaluation. The detailed results of this research can be found in “Synthetic flow-based cryptomining attack generation through Generative Adversarial Network” [29], “Improving the quality of generative models through Smirnov transformation” [22], and “B5GEMINI: AI-Driven Network Digital Twin” [30].

3.5.3.1 Motivation

Cybersecurity and large-scale network traffic analysis are two important areas receiving considerable attention over the last few years. Among other reasons, this is due to the necessity of empowering the telecom industry to adopt suitable mechanisms to face emerging and sophisticated cyber-attacks. Nowadays, Internet Service Providers (ISPs) and their clients are exposed to a growing rise in the number and type of threats (e.g., network attacks and data theft over the wire), some of which also attack at the application level using the network for identity theft, phishing, or malware distribution. In general terms, these threats put QoE at severe risk, undermining services, network resources, and users' confidence. In this context, one promising solution is the use of Machine and Deep Learning (MDL) techniques to address the appearance of new points of vulnerability and exposure to new attack vectors. At the same time, malicious agents are moving forward in the same direction to use MDL for their activities or to deceive MDL inference engines.

The application of MDL techniques requires the availability of considerable amounts of data to take advantage of their powerful learning processes. Telecom data management processes are not well suited to offer these required data sets as they exhibit a set of problems not only related to the gathering and sharing of data, but also to their processing in an MDL pipeline. It is worth noting that the applicability of MDL algorithms should take into account the evolution of attack patterns over time, which implies periodically producing additional volumes of relevant data for training new MDL models.

Moreover, a great percentage of MDL techniques used in Intrusion Detection Systems (IDS) are the so-called supervised techniques that require labelled data sets to train and validate MDL models. As in many other domains, telecom industry faces the impossibility of having labelled data sets or developing efficient and accurate processes to label them. Since network traffic is generated by end users and applications, it can be challenging for an ISP to identify and label the nature of network traffic at the detailed level required by MDL techniques. This difficulty is exploited by cyber criminals, who seek to mix cyber-attacks with normal traffic by encrypting it over common TCP ports (e.g., TLS using TCP port 443 used by HTTPS). Although unsupervised techniques that do not need labelled data sets can be applied in some scenarios, a significant number of sophisticated attacks require supervised MDL methods to be detected.

Even if efficient mechanisms for labelling data sets can be implemented, data are increasingly protected by the legal regulations that governments impose to guarantee the privacy of their contents (e.g., European General Data Protection Regulation (GDPR)). These restrictions may discourage the use of real data sets for MDL training and validation.

For a suitable advance on cybersecurity research, and specifically, on threat detection in network traffic, the telecom industry requires novel methods to generate labelled data sets to be used in MDL training and validation.

In the last decade, GANs [29] have gained significant attention due to their ability to generate synthetic data simulating realistic media such as images, text, audio and videos. Nowadays, GANs are broadly studied and applied through academic and industrial research in different domains beyond media (e.g., natural language processing, medicine, electronics, networking, and cybersecurity). In short, a GAN model is represented by two independent neural networks (the generator and the discriminator) that compete to learn and reproduce the distribution of a real data set. After a GAN has been trained, its generator can produce as many synthetic examples as necessary, providing an efficient mechanism for solving the lack of labelled data sets and potential privacy restrictions.

In this context, significant effort in task T4.1 has been devoted to researching the application of GANs to generate synthetic flow-based network traffic that mimics network attacks and normal traffic. In contrast to other approaches that are based on data augmentation solutions, we aim to generate synthetic data that can fully replace real data (attacks and normal traffic). Therefore, MDL models trained with synthetic data will obtain a similar performance to MDL models trained with real data when both are tested and deployed in real-time scenarios. This solution has two clear advantages: Firstly, addressing the existing shortage of publicly available network traffic datasets containing both attacks and normal traffic, and secondly, avoiding the privacy violations that could appear when real data is used in MDL training and testing processes.

In the light of these advantages, interesting applications can also be devised in relation to MDL cross-developments. Providing labelled data that does not incur privacy breaches can foster cross-development of MDL components by third parties. For example, a telecom provider developing ML-based components to be part of an IDS, receives synthetic data from a telecom operator to train and validate these ML-based components. As the synthetic data have been generated from real data using GANs, the ML component after training will reach the desired level of performance and furthermore, no breach of data privacy will be raised as the telecom operator is not sharing any real data with the telecom provider.

The GAN models developed in the context of T4.1 can be used as the basic building block of a Synthetic Network Traffic Generator to obtain synthetic network traffic data (attacks and well-behaved connections) that reproduce the statistical distribution of real traffic. High quality synthetic traffic can be used in training and testing of ML-based engines and in particular, in the training of ML-based engines that have been deployed in the CAD component. In the following subsections we introduce GANs, then a new set of metrics is proposed for measuring the quality of the synthetic data they generate, then we detail the GAN solution we have designed and finally, we present the experimental results when the ML-based cryptomining detector included in the CAD component was used.

3.5.3.2 Final Design

3.5.3.2.1 GANs for Data Generation

A Generative Adversarial Network (GAN), introduced by I. Goodfellow in [29], is a generative model in which two neural networks, G, called the generative network, and D, called the discriminant, compete to improve their performance (Figure 25).

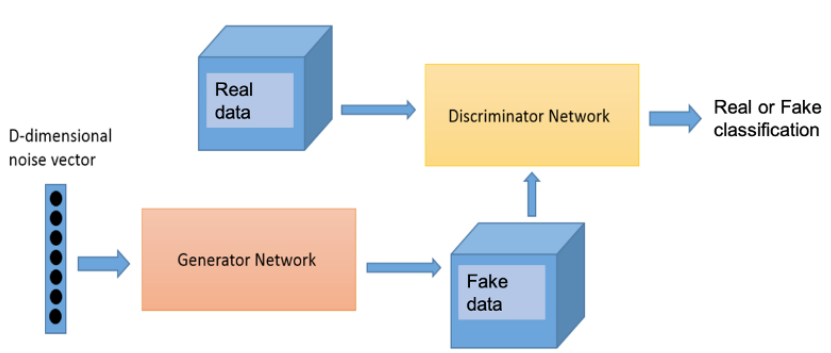

Figure 25. GAN Architecture Used as a Reference Model.

The discriminant is a network computing function *"D"* that is trained to solve a typical classification problem: given an input *X,* the discriminant is intended to predict whether *X* is compatible with being a real instance or it is a fake datum. On the other hand, the generative network computes a function *"G"* that takes as input *Z*, a latent noise vector (i.e., Z can take any random value on the dimension of the latent vector) and produces as output a vector *X'* distributed as similarly as possible to the real data. The competition appears because networks D and G try to improve non-simultaneously satisfactory objectives. On the one hand, D tries to improve its performance in the classification problem but, on the other hand, G tries to generate the best results possible to cheat D. To be precise, it is worth noting that the perfect result for the classification problem for D is *D(X)*=1 if *X* is an instance of real data and *D(X)*=0 if not. Hence, the objective of the game is:

$$\min_G \max_D \mathcal{F}(D,G) = \min_G \max_D \mathbb{E}_\Omega f[D(X)] + \mathbb{E}_\Lambda f[-D(G(Z))].$$

Despite the simplicity of the formulation of the cost function, the optimisation problem is far from trivial. The best scenario would be to obtain a so-called Nash equilibrium for the game, such that neither D or G can improve their result unilaterally. Based on this idea, the classical training method as proposed by Goodfellow is to alternate optimisation of D and G using classical gradient descent-based backpropagation. Despite that this method may provoke some convergence issues, it is a widely used learning algorithm due to its simplicity and direct implementation using standard machine learning libraries like Keras or TensorFlow.

To be precise, the algorithm proposed by Goodfellow suggests freezing the internal weights of G and to use it to generate a batch of fake examples. With this set of fake instances and another batch of real instances (e.g., sampling uniformly at random from the dataset of real instances), we train D to improve its accuracy in the classification problem with the usual backpropagation (i.e., gradient descent) method. Afterwards, we freeze the weights of D and we sample a batch of latent data (i.e., we sample random noise using the latent distribution) and we use it to train G using gradient descent for G with objective *f(-D(G(Z)))* (i.e., fake samples are labelled as real samples to allow G to learn how to cheat D). We can alternate this process as many times as needed until we reach the desired performance.

As noted in a recent work, the game to be optimised is not a convex-concave problem, so in general the convergence of the usual training methods is not guaranteed. Under some assumptions on the behaviour of the game around the Nash equilibrium points, it has been proved that the usual gradient descent optimisation is locally asymptotically stable. However, the hypotheses needed to apply this result are quite strong and seem to be infeasible in practice. For this reason, several heuristic methods

for stabilising the training of GANs have been proposed such as feature matching, minibatch discrimination, and semi-supervised training as well as approaches changing the weight function "*f*" as the Wasserstein GAN [32]. The most promising approaches are based on the modification of the usual alternating gradient descending optimisation such as the introduction of instance noise and regularisation methods based on gradient penalty.

We can summarise that two important problems are still open and under investigation in the GAN research community: (i) How to route the optimisation process to convergent behaviour and as a by-product to obtain a stopping criterion, and (ii) How to measure the quality of the synthetic data generated by a GAN. A significant percentage of our research activities in task T4.1 has been driven by these two open questions, and in this deliverable we detail several innovative solutions we developed to cope with them.

3.5.3.2.2 Quality Measures Used for Evaluating Data Generation

To date, there is no global consensus on how to measure the quality of the synthetic data generated by GANs. For that reason, we propose to evaluate GAN performance using two different types of metrics. The first set of metrics is inspired by the L1 functional distance and the Jaccard coefficient (known as the L1 metric and the Jaccard index – see below) and aim to quantify the similarity of the synthetic data with respect to the real data from a statistical perspective and considering the joint distribution of data features. On the other hand, the second set of metrics attempts to quantify the performance of synthetic data when it is used as a substitute for real data in the training of an ML classifier that is trained to distinguish between normal and cryptomining traffic. These two types of metrics will be used to compare the real and synthetic distributions, and will also be applied to implement a stopping criterion for GAN training to select generators that produce high-quality synthetic data.

L1 Metric and Jaccard Index

These two metrics try to measure the difference between the probabilistic distributions of the real data and of the synthetic data. They are based on two well-known statistical coefficients applied for hypothesis testing and probabilistic distances. These coefficients can only be applied in a scenario where the density functions are perfectly known. This obviously does not hold in a practical situation. However, the definition of these coefficients can be straightforwardly extended to the sampling scenario by replacing the probability density function with the histogram of a sample. Instead of computing independent distances in each of the data dimensions, we first map K-dimensional data to a single dimension and compute the distances on unidimensional vectors. This mapping consists in allocating D-dimensional elements in J partitions (e.g., bins or cubes) to compute later the distances on the 1-Dimensional set of partitions.

Then, *d_L1(X,Y)* is the sum of the differences of X and Y points in each bin divided by the total number of points. Therefore, *d_L1(X,Y) = 0* if and only if the number of samples of X and Y in each bin are equal, and if the bins of the partition are the same. Note that if all points of X and Y are in different bins, *d_L1(X,Y) = 2.*

In a similar vein, the Jaccard index can be estimated by dividing the number of bins in which X and Y both have at least one point by the total number of bins. This coefficient takes values in the interval [0,1] and the larger the value of this index the more similar the two data distributions are.

Nested ML Performance

The second set of metrics attempts to quantify the performance of synthetic data when it is used as a substitute for real data for training an ML classifier to distinguish between normal and attack network traffic.

The quality of the synthetic data generated by a GAN will be evaluated as follows. Suppose that, we have trained two GANs, G0 and G1, to replicate two types of data. In our experiments, G0 synthesised data with label 0 (real traffic), and G1 produced synthetic traffic with label 1 (attack traffic). Using G0 and G1, we generate samples M and N of synthetic traffic of label 0 and 1 respectively. With this new data set of M+N samples, we train a standard ML classifier C (say, a random forest classifier). Then, screening all of the precision, recall, and F1-score of C against a test split made of real data, we are able to measure the quality of the generated data: the higher these measures, the better the synthetic data. Hence, large values of these coefficients point out that the synthetic data generated by G0 and G1 can be used to faithfully substitute the real instances. Observe that no real traffic is used for such training purposes, although real traffic is always used for testing.

As previously stated, this is a differentiating characteristic of our solution with respect to existing solutions. Data augmentation solutions are usually proposed in state-of-the-art solutions, where synthetic data is mixed with real data during training, generating data privacy breaches as real data is used.

Several variants of this proposal can be considered. In the first approach, instead of creating the dataset with fully trained GANs, we compute these coefficients at each of the training epochs of the GAN. Therefore, at the end of the training, we have an ordered collection of generators for each label. In this way, we are able to screen the evolution of the training and to relate it to the quality of the generated data. In particular, this idea enables a novel stopping criterion: when the GAN training epochs do not produce any significant enhancement in the nested ML task, the training process is stopped.

A second variant consists of evaluating the marginal quality of each of the generators. In the first approach, we generated the synthetic dataset for training using synthetic samples of both types of traffic, normal (label 0) and attack (label 1). However, if we want to test the quality when generating only one of the two types of traffic, say label 0, the dataset can be also created by mixing synthetic samples of label 0 with real samples of label 1. In this way, the corresponding ML accuracy coefficients will only measure the ability of G0 to generate synthetic traffic of label 0, regardless of the fitness of G1.

3.5.3.2.3 GAN Solution for Synthetic Attacks and Normal Traffic Generation

To demonstrate the applicability of GANs to the generation of synthetic flow-based network data, we select the cryptomining attack scenario that was previously considered in the development of the CAD component. Cryptomining is a paradigmatic cryptojacking attack that is gaining momentum. Cryptomining attacks concern network traffic generated by cybercriminals that create tailored and illegal processes for capturing computational resources from users' devices without their consent and using them to the benefit of the criminal for mining cryptocurrencies. It has been shown that these malicious connections can be detected in real-time with decent accuracy even at the very beginning of the connection's lifetime by using an ML classifier [12].

Our goal was to obtain synthetic traffic of sufficient quality to allow a complete replacement of real data by synthetic samples during the training of an ML-based crytomining attack detector. This ensures that we will not violate any privacy restriction and sets our solution apart from pre-existing works that only propose data augmentation solutions based on mixing real with synthetic data.

To generate flow-based information that replicates normal traffic and cryptomining connections, we applied two Wasserstein GANs (WGANs) to generate both types of network traffic separately. Unlike current solutions, our WGANs replicate not only already completed connections, but also connections at different stages of their lifetime.

Considering that successful GAN training is still an open research problem, we designed a set of GAN metrics to measure the impact on the convergence of the WGAN training and the quality of the synthetic data generated. We proposed two new metrics based on the L1 distance and the Jaccard coefficient to measure the quality of the synthetic data generated by GANs with respect to their similarity to real data. The novelty of these metrics is that they consider the joint distribution of all the variables in the flow-based data rather than the mean over the distance of each variable as state-of-the-art solutions propose. In addition, we measured the quality of the synthetic data using nested ML classifiers to evaluate separately the marginal quality of each of the generators.

To the best of our knowledge and due to the ill-convergence of GANs, none of the existing works propose a clear stopping criterion during GAN training to obtain the best performing synthetic data. To address this problem, we designed a simple heuristic for selecting the best performing GANs when it is required to fully replace real data with synthetic data for training MDL models. This heuristic consists in evaluating separately the marginal performance of each generator obtained at the end of a mini-batch training stage to select the best one. In this way, each WGAN can be trained in parallel without requiring the other to evaluate their joint performance and therefore, each training can be stopped at different epochs when no significant enhancement is observed.

We experimentally observed that even when selecting the best performing generator for each type of traffic, the performance obtained when we mix them may not be the best, and therefore we propose a second heuristic for finding the best performing combination.

Since we will use two different WGANs, one for each type of traffic, trying to find the best performing combination would imply evaluating the ML performance of G0 at each epoch with all generators obtained during G1 training and conversely, at each G1 epoch we should combine its generator with all G0 generators to measure ML performance. Hence, assuming we trained a pair of GANs for M and N epochs respectively, we would require M x N evaluations of the ML task to obtain the best performing combination. In order to avoid generating the Cartesian product (M * N) of the two sets of generators and running the corresponding ML evaluations, we observed experimentally that drawing approximately several dozen samples by choosing uniformly at random one generator of each type of traffic tended to produce results equivalent to the brute force approach of trying all possible combinations. Moreover, we observed that when the F1-score on testing was used as the performance metric, reducing the sample GAN universe to the top-10 best models in the F1-score elitism for each type of traffic produced similar results, but required significantly fewer samples.

To demonstrate the proposed solution, we ran an extensive set of experiments. The data sets used in our experiments were previously generated in a realistic network digital twin called the Mouseworld lab (Figure 26). In this lab, we launched real clients and servers that interacted with other hosts located in different places in the Internet and collected the generated traffic composed of encrypted and non-encrypted flows (normal traffic and cryptomining connections).

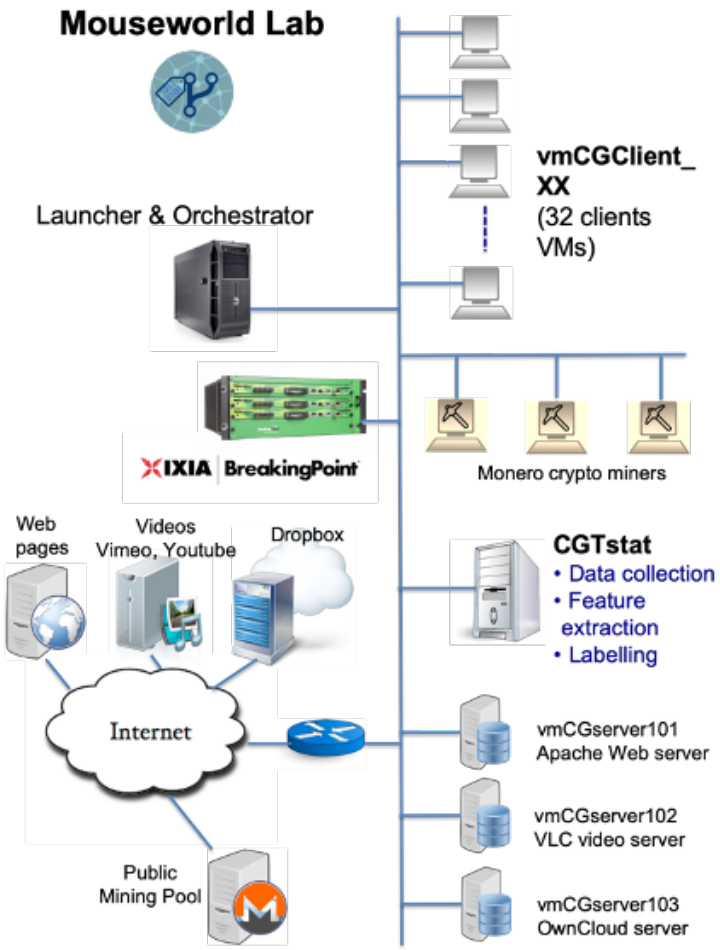

Figure 26. The Mouseworld Lab in the Telefónica I+D Premises.

A set of 59 statistical features were extracted from each TCP connection in real time each time a new packet was received. We carefully selected a reduced set of 4 features for our GAN experiments (Figure 27). We trained two WGANs independently, one for each type of traffic, configuring them with a rich set of hyperparameters (Table 10). Then, a blind random search in the hyperparameter space of each WGAN was applied, and in order to select the best configuration of hyperparameters, we used the F1-score obtained in a nested ML-classifier that was executed after each training epoch.

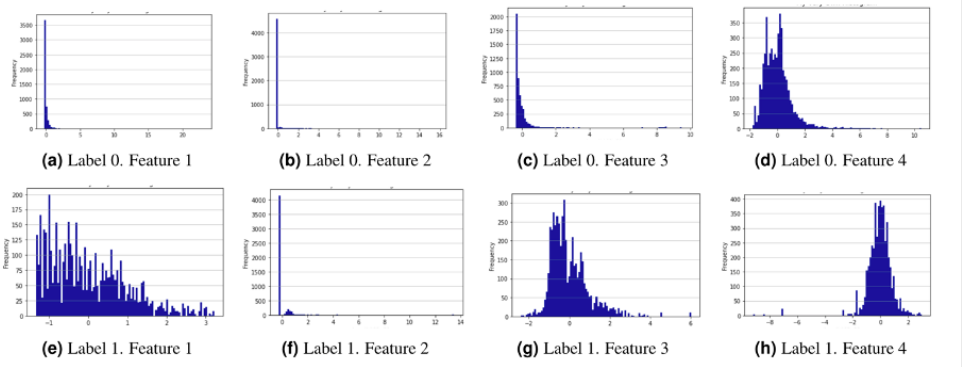

Figure 27. Frequency Histogram of the 4 Features Extracted from Normal (label 0) and Cryptomining (label1) Traffic.

Table 10. WGAN Hyperparameters.

		Range of values
Genera-tor/discriminator FCNN architectures	*# layers*	[2..6]
	# units per layer	[100..10000]
Generator	*Output Activation*	(linear,custom)
	Output filtering with discriminator (>0, percentile)	(True,False) [0..100]
	Latent vector with Embedding (categories)	(True,False) [1..20]
	latent vector	Fixed 123
	noise for latent vector (distr,std)	(normal, uniform) std=[0.1..100]
	batch normalization	[True..False]
	regularization: L2, dropout	Fixed values (0,0)
	learning rate	Default value (0.001)
	LeakyRelu alpha	Fixed value (0.15)
	Percentage of tanh/LeakyRelu in internal units	[0..100]
Discriminator	*Noise in fakes (distribution,std)*	(normal,uniform) [0..20]
	Noise in all examples (distribution,std)	(normal,uniform) [0..20]
	Ratio Label change	[0..20]
	batch normalization	[True..False]
	regularization: L2, dropout	[0..2], [0..30]
	LeakyRelu alpha	Fixed value (0.2)
	learning rate	[0.0001..0.001]
Adaptive mini-batch	*generator. ratio fake pass*	Fixed (0.3)
	discriminator. ratio TP	Fixed (0.01)
	discriminator. ratio TN	Fixed (0.01)

For each type of traffic, we selected the WGAN that obtained the best F1-score for the nested classifier in any of its epochs. In addition, we computed the two proposed metrics (L1 distance and Jaccard index) on each WGAN to analyse the similarity of the synthetic traffic they are generating with respect to the real data. Given that, for each winning WGAN, we have as many intermediate generators as the training steps we run, the previously proposed heuristic is run to choose pairs of generators that produce good results in a global nested evaluation of both WGANs. Thus, we avoid testing all possible combinations of generators from the two WGANs.

To measure the quality of the synthetic data generated by our GANs, we train an ML classifier for detecting cryptomining connections in real-time using only a combination of the synthetic traffic generated by the two WGANs (normal traffic and cryptomining attacks). Another ML classifier configured with the same set of hyperparameters is trained using real data. Both models are tested against a second set of real data to measure whether the ML model trained exclusively with synthetic data performs at the same level as the model trained with real data. As a baseline for the quality of synthetic traffic, we use a naive approach based on a noise generator added to the averages of the variables. As an upper bound, we considered the results obtained with real traffic.

GAN Architecture

Aiming to mimic two types of behaviour (cryptomining attacks and well-behaved connections), in preliminary experiments, we adopted a well-known conditional GAN model, the so-called Auxiliary Classifier GANs (AC-GAN), as the architecture to generate at the same time the two types of traffic variables. As this strategy did not produce an adequate performance when replicating the two types

of traffic and generated significant oscillations in the convergence process, we opted to use a different approach.

Assuming that the two types of traffic are independent of each other and therefore it is not necessary to use one for the synthetic generation of the other, we finally proposed to train two standard GANs independently, one for normal traffic (i.e., well-behaved connections) and the other for the cryptomining connections. The reference architecture for these GANs is shown in Figure 25. As previously explained, this model is composed of two networks, a generator and a discriminator, that compete. The generator is input a random noise vector and produces a synthetic sample. The discriminator receives real and fake samples as input and tries to classify them appropriately in their correct category. During training, the goal of the generator is to learn how to produce fake samples that can be classified as real by the discriminator. In contrast, the goal of the discriminator during training is to learn how to differentiate real from fake examples.

To get rid of the mode-collapse problems that frequently appear during GAN training, we adopted as a reference model the WGAN architecture, in which a Wasserstein loss function is used as loss function instead of a standard cross-entropy function. In addition to replacing the loss function, we tested two different strategies to enforce the required Lipschitz constraint in this function. Initially, a radical weight clipping strategy was applied. Later, we replaced weight clipping with a more elaborated gradient penalty approach. It is worth noting that in our experiments, none of them produced a significant enhancement in the convergence of the GAN training and, in many occasions, we observed that the gradient penalty heuristic even produced significant oscillations. Therefore, we finally chose a WGAN architecture with no additional strategy to enforce the Lipschitz constraint and the discriminator was optimised using only small learning rates and a new adaptive mini-batch procedure as heuristics to avoid reaching mode collapse situations.

We selected fully connected neural networks (FCNNs) as the architectural model for both the discriminant and the generative networks. This decision was based on the observation that the statistical nature of the four features to be synthetically replicated did not exhibit any topological structure or time relationship among them and therefore, convolutional (CNNs) or recurrent networks (e.g., LSTMs) respectively would not provide any advantage compared to FCNNs.

We observed, in preliminary experiments, that very deep networks with a large number of hidden layers or units did not generate significant improvements in performance and on the contrary, they enlarged convergence times and produced non-negligible oscillations in the convergence during the training process. Therefore, we selected a moderate number of hidden layers (between 3 and 5) for generator and discriminator networks. To provide the generator with more complex nonlinear capabilities to learn how to fool the discriminator, we used hyperbolic tangent and Leaky-ReLU functions as activation functions in the neurons of its hidden layers. In the case of the discriminator, only LeakyReLUs were used.

An important issue in the generation of synthetic replicas of network traffic variables is that real data are sometimes not normally distributed. In general, telecom domain variables used in ML are usually statistical data representing the evolution of flow variables such as counters, accumulators, and ratios that always take positive values. In our real-time experiments, flows are monitored periodically from start to finish and, therefore, the collected values for some of these variables are not normally distributed and tend to follow an exponential distribution with many occurrences of values near to 0 and a long tail of large values appearing very rarely.

Generator networks usually have a linear activation in the neurons of their output layer, which produces output variables following a normal distribution with mean value equal to that of the

distribution of the real data. When the real data follow a different distribution (e.g., exponential), using linear activation functions in the output layer of the generator can produce synthetic data outside the domain of the real data. For example, if we consider a variable representing an accumulator, only positive values are possible in the domain of real data. However, the generator will produce synthetic data with the same mean value as the real data distribution (e.g., exponential) but following a normal distribution that contains negative data not existing in the real domain.

To mitigate this problem, we propose to use specific activation functions in the output neurons of the generator to adjust as much as possible the data distributions of the generator outputs to the statistical distribution of the real variables and thus, avoid the generation of negative values outside the domain of such variables. To try to replicate variables that follow an exponential distribution, we propose to use ReLU functions in the generator as activation functions of the neurons at the output layer. The ReLU function only generates positive values due to its non-linear behaviour (the output is the input value for positive values and 0 for negative values). In order to provide a smoother transition at values close to 0, we experimentally observed that a Leaky-ReLU function with a very small slope for negative values performed better than a pure ReLU function.

Furthermore, we designed three novel mechanisms and applied a well-known heuristic based on adding noise to the discriminator and tested each of them to see if they could impact on the training convergence or the quality of the generated synthetic data. These heuristics are the following: (i) an adaptive number of mini-batch cycles are applied to the training of discriminator and generator networks to avoid the occurrence of the so-called collapse mode and balance the learning speed of both networks (Figure 26); (ii) different types of noise are added to the discriminator inputs to slow down its learning speed; (iii) a multi-point embedding of a single class is added to the input layer of the generator for augmenting the variety of the latent noise vector; and (iv) real traffic of the class not modelled in the GAN is added jointly with the set of fake examples to slow down the learning process of the discriminator.

```
1:  procedure GAN_TRAIN_BATCH
2:      repeat
3:          train_batch(discriminator)
4:          preds ← discriminator.predict(sample(real, synthetic))
5:      until TP(preds) and TN(preds) are in range
6:      repeat
7:          train_batch(generator)
8:          preds ← discriminator.predict(generator.predict(noise))
9:          ratio_fake_pass ← len(preds = real_label)/len(preds)
10:     until ratio_fake_pass > min_ratio_fake_pass
```

Figure 28. Enhanced GAN Training Procedure with Adaptive Mini-Batches.

3.5.3.3 Evaluation

In this section, we summarise the results obtained in the experiments we conducted when real data sets are replaced by fully synthetic datasets. For this purpose, we compared the performance obtained by an ML classifier when trained with: (i) real data; (ii) data generated through a simple mean-based generator (working as baseline); and (iii) a synthetic dataset generated with a standard WGAN (with no variants or improvements implemented). In addition, we present the effects of using an improved WGAN. Recall that we denote normal traffic as Label 0 and cryptomining attacks as Label 1.

Table 12 compiles the results of all experimental variations. For each variation we show the F1-score on testing and the confusion matrix for the best and the default (0.5) decision thresholds.

First, a Random Forest was trained to classify the original real dataset into normal and cryptomining-based traffic. We chose Random Forest with 300 estimators due to its well-known good performance

in classification tasks and in particular, when cryptomining and normal traffic has to be classified. This experiment provides a superior benchmark to compare against the synthetic data generated by the WGANs. Results are shown in row 1, column 1.

Then, the training dataset was fully substituted with a synthetic dataset generated through a simple mean-based generator. This served as a baseline for all experiments. The mean-based generator was created by computing the mean and variance of each of the four features of the dataset per class. With these data, a completely new dataset was generated by drawing samples from a multivariate normal centred at the means with diagonal covariance matrix (i.e., each feature is drawn independently) for each class. The obtained results (row 1, column 2) evince that the naive mean-based generator is not a good approach for generating a synthetic dataset when the data distributions are not easily separable as happened in our experiments. Hence, these results point out that a much subtler method of generation is required to obtain compelling performance.

WGANs

The initial WGAN experiments consisted of using standard WGANs with linear activation functions in the output layer.

For normal traffic, the set of hyperparameters that produced the best performing WGAN was as follows:

- Generator. Architecture: [123,200,500,3000,500,4]; output activation: linear; latent vector size: 123; multipoint single-class embedding: False; noise for latent vector: Normal (0,5); batch normalisation: True; Percentage of tanh: 15%.
- Discriminator. Architecture: [4,380,800,600,177,23,1]; output filtering: False; noise in input (real and fakes): N(0,0.02); noise in fakes: N(0,0); ratio label change: 0; batch normalisation: True; regularisation: 0.02; dropout: 0.1; learning rate RMS-Prop: 0.001.

For cryptomining connections, the set of hyperparameters that produced the best performing WGAN was as follows:

- Generator. Architecture: [123,600,3000,1000,4]; output activation: linear; latent vector size: 123; multipoint single-class embedding: False; noise for latent vector: uniform (0,3.5); batch normalisation: True; Percentage of tanh: 5%.
- Discriminator. Architecture: [4,280,903,500,23,1]; output filtering: False; noise in input (real and fakes): N(0,0.01); noise in fakes: N(0,0); ratio label change: 0; batch normalisation: True; regularisation: 0.05; dropout: 0.15; learning rate RMS-Prop: 0.001.

The selection of a pair of generators and the number of samples produced by each of them was done using the following three policies:

- An unbalanced dataset (with 400K/4K instances) is generated by picking at random one model for each label among the partially trained models.
- An unbalanced dataset (with 400K/4K instances) is generated by picking at random two models for each label. The instances of the synthetic dataset were obtained by mixing the outputs of the two chosen generative models, in the expectancy of increasing the variety and diversity of the synthetic dataset.
- A balanced dataset (with 4K/4K instances) is generated by picking at random one model for each label.

Table 11. Summary of WGAN Experiment Results.

Baseline results using real data for training

Dataset	Quality Measure	Best		Default	
Training 400K/4K Real dataset	*Threshold*	0.4		0.5	
	F_1-*score*	0.962		0.928	
	Confusion matrix	399817	183	399877	123
		459	3929	1008	3380
Training 4K/4K Real dataset	*Threshold*	0.8		0.5	
	F_1-*score*	0.919		0.793	
	Confusion matrix	398602	1398	394172	5828
		197	4191	62	4326

Baseline results using a naïve mean-based generator for training

Dataset	Quality Measure	Best		Default	
Training 400K/4K Mean-based generation	*Threshold*	0.8		0.5	
	F_1-*score*	0.732		0.664	
	Confusion matrix	396318	3682	390060	9940
		1894	2493	1416	2971
Training 4K/4K Mean-based generation	*Threshold*	0.8		0.5	
	F_1-*score*	0.601		0.583	
	Confusion matrix	377352	22648	370528	29472
		839	3548	537	3850

Performance of synthetic traffic generated by standard WGANs. with linear activation in the output.

Results on testing using the best models on training.

Dataset	Quality Measure	Best		Default	
Training 400K/4K Policy 1) dataset	*Threshold*	0.4		0.5	
	F_1-*score*	0.936		0.933	
	Confusion matrix	399926	74	399962	38
		927	3461	998	3390
Training 400K/4K Policy 2) dataset	*Threshold*	0.8		0.5	
	F_1-*score*	0.927		0.915	
	Confusion matrix	399449	551	399601	399
		701	3687	983	3405
Training 4K/4K Policy 3) dataset	*Threshold*	0.8		0.5	
	F_1-*score*	0.878		0.835	
	Confusion matrix	399030	970	396381	3619
		1108	3280	315	4073

Performance of synthetic traffic when generators use custom activation functions in the output.

Results on testing using the best models on training

Dataset	Quality Measure	Best		Default	
Training 400K/4K Policy 1) dataset	*Threshold*	0.8		0.5	
	F_1-*score*	0.649		0.555	
	Confusion matrix	382755	17245	357385	42615
		241	4146	96	4291
Training 400K/4K Policy 2) dataset	*Threshold*	0.8		0.5	
	F_1-*score*	0.622		0.559	
	Confusion matrix	377693	22307	358897	41103
		163	4224	82	4305
Training 4K/4K Policy 3) dataset	*Threshold*	0.8		0.5	
	F_1-*score*	0.532		0.467	
	Confusion matrix	345452	54548]	298568	101432
		21	4366	0	4387

Performance of synthetic traffic generated by standard WGANs. after filtering fake samples by discriminator.

Results on testing using the best models on training.

Dataset	Quality Measure	Best		Default	
Training 400K/4K Filtering out fakes	*Threshold*	0.6		0.5	
	F_1-*score*	0.925		0.914	
	Confusion matrix	399511	489	399221	779
		767	3621	722	3666

Performance of synthetic traffic generated by standard WGANs by sampling generators with elitism among the top 10 models in training sorted by L1-distance and Jaccard index. Results on testing.

Dataset	Quality Measure	Best		Default	
Training 400K/4K Top 10 L^1-distance	*Threshold*	0.8		0.5	
	F_1-*score*	0.869		0.858	
	Confusion matrix	399491	509	398536	1464
		1506	2882	1094	3294
Training 400K/4K Top 10 Jaccard index	*Threshold*	0.8		0.5	
	F_1-*score*	0.884		0.826	
	Confusion matrix	399033	967	396881	3119
		1031	3357	720	3668

Performance of synthetic traffic generated by standard WGANs by sampling generators with elitism among the top 10 models in training sorted by F1-score. Results on testing.

Dataset	Quality Measure	Best		Default	
Training 400K/4K Top 10 in F_1-score	*Threshold*	0.4		0.5	
	F_1-*score*	0.951		0.950	
	Confusion matrix	399800	200	399832	168
		608	3780	658	3730

The results of the best standard WGAN are summarised in row 2, column 1 of Table 11. In addition, this table includes the results after applying the following improvements to the WGAN standard configuration:

1. (Row 2, column 2) Custom Activation function.

   Two linear functions at the output of the generator are substituted by custom LeakyReLU units to fit exponential distributions of these variables (e.g., counters or accumulators).

2. (Row 3, column 1) Discriminator as quality assurance.

   Use the discriminator network of the GAN as a quality assessor. To be precise, after training the GAN, instead of using all samples synthesised by the generator network, we add to the generated dataset only those that were classified as real samples by the discriminator agent ($D(x)>0$), while the samples judged as fake ($D(x)\leq 0$) are ruled out.

3. (Row 4, column 1) Elitism by F1-score.

   This is a different strategy for more efficiently sampling the pair of models for data generation. Instead of picking a random model among all epochs, we only draw samples from the top 10 models obtained throughout all the training epochs, in the sense that they achieved the best F1-scores. With this strategy, we aim to apply some type of elitism that prevents a drastic fall of the performance due to a random choice of a bad generator. In addition, this strategy significantly reduces the number of combinations that we must evaluate to find which pair of generators produces the best performing synthetic data.

4. (Row 3, column 2) Elitism by statistical measures.

   This strategy is also for more efficiently sampling the pair of models for data generation. We sample the model used for generating the dataset from the top 10 models in training. However, instead of using F1-score as quality measure, we use the statistical measures of L1 distance and Jaccard index to sort the best performing models. This analysis is useful to determine whether the statistical measures are a faithful quality control of the performance expected in a nested ML model.

3.5.3.3.1 Summary of the Evaluation

We conclude this section summarising below the main experimental results and observations.

- A WGAN is able to generate synthetic samples that can fully substitute the real samples needed to train a nested ML classifier. On this synthetic dataset, the performance of the nested ML is, in some cases, slightly better than the results obtained with the original data, and widely outperforms the results with a mean-based generator. Combining the synthetic data generated by two WGANs per label produces more consistent results between executions, even though the best results are slightly worse than that of a single WGAN per label due to the added spurious noise of the mix.
- As a consequence of the above result, the ML-based engine of the CAD can be trained using only synthetic data, which allows for no privacy violation, as no real data is used during the process. In addition, synthetic data can be shared with third parties that can develop ML-based engines for the CAD component without incurring privacy breaches.
- Measuring the quality of the synthetic data at the end of each epoch, it is observed that although statistical measures (L1 distance and Jaccard index) show an asymptotic decrease during GAN training, there is no direct correlation with a better performance of the ML nested task. Therefore, these classical measures of the goodness of fit are not good estimators of the information contained in the generated data with respect to the performance of a nested ML model.
- When a custom activation function at the output of the generator network is applied, the performance of the nested ML model slightly decreases. However, the new samples can preserve some important domain constrains such as non-negative values. This property is mandatory when we want to obtain data for use in applications in which it is crucial that the

data do not contain any non-existent value. Additionally, with this approach the number of false negatives suffered by the nested ML model drastically decreases.

- No significant improvement is detected when the generated samples are filtered out according to the prediction of the discriminant agent.
- When the GAN models are selected according to F-1-score elitism, the results improve slightly with respect to a standard GAN, and moreover, only a small number of samples are required to obtain a pair of WGANs that perform similarly to real data in a nested ML task. However, when the GAN models are selected according to their statistical quality metrics (L1 distance and Jaccard index), the performance of the models decreases. This result shows that these probability-based coefficients are not suitable for assessing the quality of the synthetic samples.

4 Distributed Ledger and Smart Contracts

This section introduces all aspects related to the Distributed Ledger Technology (DLT) Component used within the TeraFlow project, more specifically in T4.2.

DLT allows a set of nodes to become a distributed database by sharing information in a verifiable and transparent manner and recording the data to be stored using a consensus mechanism. Blockchain [34] is the best-known example of a DLT. Blockchain is not only a distributed data base, but it also allows the execution of code (i.e., smart contracts) allowing the peers to work together without the need of a central element commanding any action. Blockchain is being used in multiple research SDN aspects, such as the security of flow management [35] and the recovery of SDN nodes after a failure [36].

The use of Blockchains replaces centralised network management consisting of conventional database management systems. Major advantages are the elimination of trusted third parties that maintain the databases with single points of failure, and data provenance including data immutability and traceability.

First, in Section 4.1 we detail the new features, the final design and an evaluation of the Permissioned Distributed Ledger and Smart Contracts. Then, several blockchain improvements are presented in Section 4.2

4.1 Permissioned Distributed Ledger and Smart Contracts

4.1.1 New Features/Extensions

- Record management: The DLT Component enables any TFS instance to add, update, or delete records on the DLT in order to share real time information on the infrastructure's status. The underlying DLT then automatically propagates the transaction through all the DLT nodes. Records can be of any of the following type: context, topology, device, link, service, or slice.
- Subscription to events: TFS components can subscribe to record updates in order to be notified of real-time changes happening to the network architecture.

4.1.2 Final Design

The goal of the DLT Component is to facilitate the information sharing between the different TFS domains through the means of a trusted database. Furthermore, through efficient smart contracts, the DLT Component enables other components to subscribe to notifications of all or a filtered list of events. The DLT Component is split between the DLT Gateway and the DLT Connector. The DLT Gateway is used to connect to the blockchain environment as well as generating credentials to access and communicate with the deployed blockchain. The blockchain deployment is based on the open source Hyperledger Fabric with technological enhancement to improve its scalability, security, and throughput. All communication between the DLT Gateway and the blockchain deployment use a gRPC API and are secured using TLS. The DLT Gateway is implemented in Kotlin and provides a simple gRPC API for the connection with the DLT Connector, itself implemented in Python.

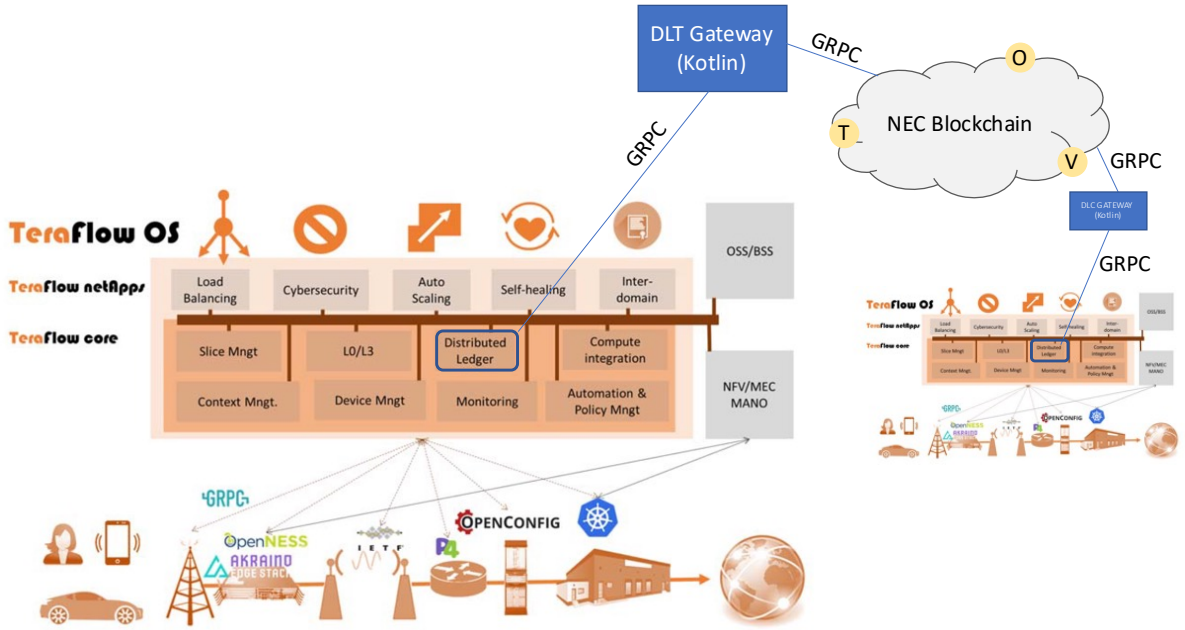

Figure 29. Overview Architecture of TFS with the DLT Component.

4.1.2.1 NEC Blockchain Architecture

Analogous to the Hyperledger fabric, NEC blockchain is a permissioned private blockchain. The governance is managed by a defined set of organisations, and each node or client needs to be part of one of the organisations. Access control is based on digital certificates akin to the existing web certificates used in today's web security for SSL/TLS.

Comparable to public blockchain, NEC blockchain defines two main types of nodes: the Orderers, that handle the consensus algorithm of the blockchain to ensure an identical view for all the participants, comparable to Ethereum and Bitcoin's miners, and the Peers, that store the blockchain, execute the smart contract, and service the client requests. Since NEC Blockchain is permissioned and private, signifying the identity (certificate) of all the participants is known, the Orderers can use a Byzantine Fault Algorithm (BFT) to run consensus. BFT protocols are well studied and are able to achieve much higher throughput and lower latency compared to the Proof of Work used by public blockchain. Furthermore, BFT algorithms achieve *finality*, meaning the protocol cannot create forks, hence blocks generated correctly according to the BFT rules will never be reverted. The smart contracts in NEC blockchain are written in Golang, Java, or NodeJS. Akin to most blockchain, the persistent storage is a simple key-value store, where the key is a string, and the value is any arbitrary data encoded as a byte array.

Different to pre-existing blockchain, in NEC blockchain the transaction workflow is split into two steps, analogous to Hyperledger fabric: 1) the endorsement; and 2) the ordering. In the first step, the transaction is simulated by the peers in the network and the outcome of the simulation (the read and write set) is signed by the peers. After collecting sufficiently many signatures (defined by an endorsement policy), the transaction is sent to the ordering service to be included in a block. Once the transaction is included in a block and distributed, the peers update their local database by applying the write set. Due to this two-phases process, if the read set or write set of the transaction is modified after step 1 and before step 2, the peers will "reject" the transaction and consider it as not valid. This is done to prevent shadow reads (i.e., the transaction is evaluated on an old, outdated state) and shadow writes (i.e., the transaction overwrites an update from another transaction without knowing it). Therefore, when designing the smart contract, great care should be employed to minimise the risk

of conflicting read-write sets. For example, if the smart contract creates a hashmap, stores all the information inside, and inserts only one entry in the database (the actual hashmap), any update to the hashmap will be in conflict with any other update. Instead, by storing the actual key-values directly in the database, one can reduce the risk of conflicting updates. Similarly, one should aim at minimising the number of globally accessed variable that are updated frequently (e.g., a counter to count the total number of transactions would result in many transaction updates conflicting).

We, therefore, differentiate the 4 main parts of the DLT platform:

1 **Chaincode**: A chaincode, more commonly known as a smart contract, handles the business logic of a use case. It defines the API, the access pattern to the storage, and can run simple calculations on data. The chaincodes of NEC Blockchain are Turing complete and can technically perform any computation, however, since the same computation has to be made by many different nodes, it is usually preferred that it remains lightweight.
2 **Peer**: The peers are the main interface between clients and the blockchain. Peers run the chaincodes and keep the state of the blockchain up to date. They serve all the requests of the clients.
3 **Orderer**: The role of the Orderers is only to generate block in a decentralised manner. In a general deployment, each organisation possess one or more ordering nodes, all of which will run the consensus algorithm. After each round of consensus, the Orderers output a new block that is then broadcast to the nodes.
4 **Certificate Authority**: Lastly, since NEC Blockchain is permissioned and private, the organisation relies on certificates managed by a certificate authority. Each organisation can deploy its own certificate authority, with the root certificate safely stored in the blockchain. The certificate authority then signs the certificate of all the other members, such as the peers, the Orderers, and the clients. Access to the certificate authority is managed by each organisation individually.

All communications in NEC Blockchain are authenticated using the certificates and secured through TLS. The communications between the different nodes and clients are performed through gRPC.

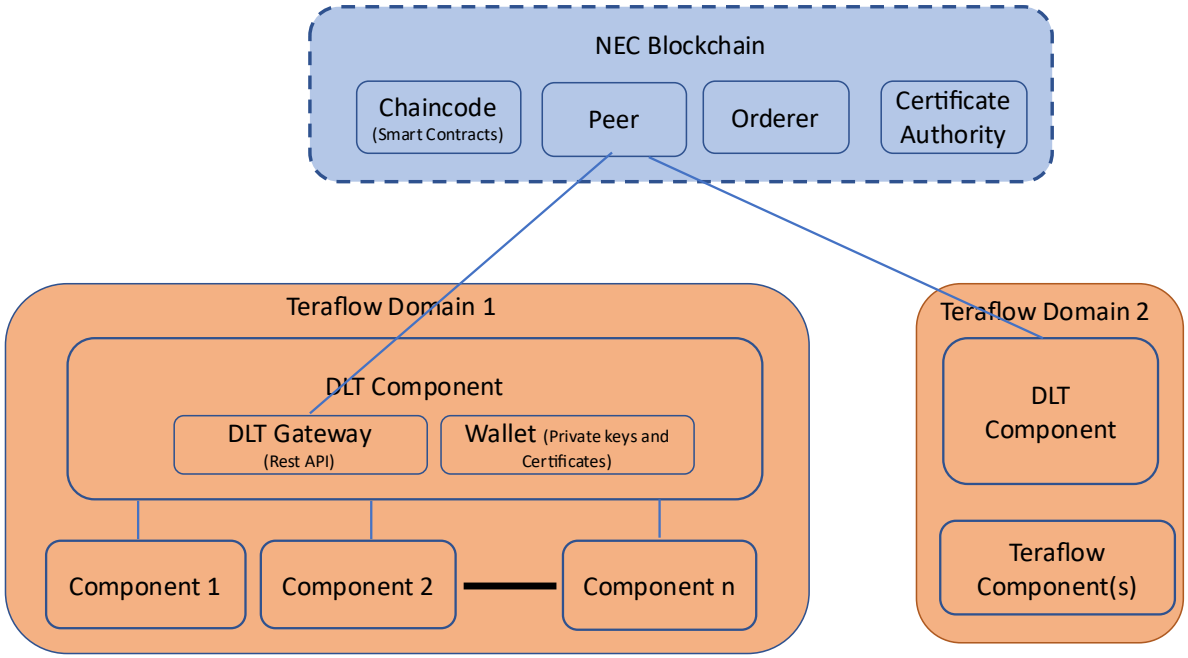

Figure 30. High-Level Overview on NEC Blockchain.

While general access to the blockchain is secured through an identity management system based on certificate chains, the chaincodes have to be secured as well against improper access of data. For example, it should not be possible for a node in domain 2 to modify or update records added by a node or client from domain 1. Access control is therefore an important part of the chaincode security and is generally enforced through the certificate identity. Identity-based access control in the Hyperledger Fabric and NEC Blockchain can be done either at the user level, based on the certificate public key, or at the organisation level, based on the top-level certificate in the verification chain. In this case, we estimate that since there is no individual data to a single DLT client, organisation-based access control is sufficient.

4.1.3 Final Interfaces

In order to share and propagate infrastructure updates to all the domains, two APIs are defined for the DLT Component. The DLT Connector and DLT Gateway APIs. The DLT Connector API defines the API exposed to the rest of the components, while the DLT Gateway API exposes a generic API to enable different DLT Gateways, each for a particular DLT technology.

The API for the DLT Connector contains the following gRPC functions:

- “RecordAll”: records into the DLT all the devices, links, services, and slices belonging to the specified topology/context.
- “RecordAllDevices”: records into the DLT all the devices belonging to the specified topology.
- “RecordDevice”: records into the DLT the specified device.
- “RecordAllLinks”: records into the DLT all the links belonging to the specified topology.
- “RecordLink”: records into the DLT the specified link.
- “RecordAllServices”: records into the DLT all the services belonging to the specified context.
- “RecordService”: records into the DLT the specified service.
- “RecordAllSlices”: records into the DLT all the slices belonging to the specified context.
- “RecordSlice”: records into the DLT the specified slice.

The DLT Gateway API exposes the following 3 gRPC functions:

- “RecordToDLT” which takes as input a record that can contain information on the context, topology, devices, links, services, or slices. The operation can be either to add, update, or delete information from the blockchain.
- “GetFromDLT” which returns a record from the blockchain deployment given a record ID.
- “SubscribeToDLT“ which takes as input a list of types of events to subscribe to. This last gRPC function returns a stream of events. Events added through the RecordToDLT function that match the type of the subscription would result in an event being sent through the stream containing the record ID.

The DLT Gateway assumes each record has a unique identifier (e.g. a unique key). In the backend, blockchain peers use a CouchDB database to store the entries in a key-value store fashion. Through the provided APIs, TFS clients can efficiently share, retrieve, and subscribe to updates of the infrastructure.

4.1.4 Evaluation

Experimental Setup

The setup is deployed on a server equipped with a 24-core AMD Ryzen Threadripper 3960X processor clocked at 2.2 GHz, with 32 GB of DDR4 RAM, and 4TB HDD storage. TFS is deployed on this server on top of MicroK8s v1.24.8, and no resource limits are set in order to allow stress testing.

The microservices deployed for the benchmark include: i) 1 instance of the Slice Component; ii) 5 instances of the Service Component; iii) 5 instances of the PathComp Component pod (e.g., Front-End and Back-End containers); iv) 1 instance of the Device Component; v) 1 instance of the Context Component; and vi) 1 instance of the DLT Component. All pods are deployed behind a Kubernetes service for load balancing purposes (when applicable).

The transport network topology used for this assessment is that shown in **Figure *31***. The topology is formed of 7 packet switches/routers which are controlled/programmed by a TFS Controller instance. It is worth mentioning that the data plane is emulated and only the controller building blocks are considered to collect the numerical results.

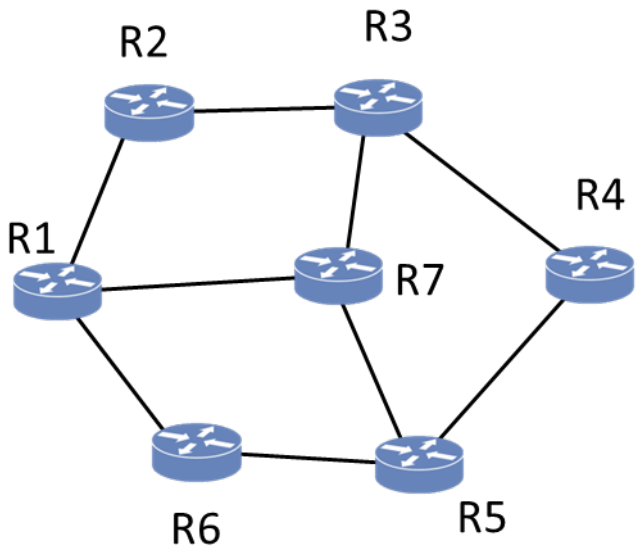

Figure 31. Transport Network Topology for DLT Evaluation.

Functional Evaluation

The functional evaluation consisted of creating a device directly in the Context Component and then requesting the DLT Component to upload it to the blockchain. The methods used for links, services, and slices, are the same, just replacing the entity names and objects. For this reason, we only include the details of the device entities.

Figure *32* shows a Wireshark capture of the externally-visible messages involved in this test. It is worth noting that the DLT Connector and DLT Gateway run within the same pod and Kubernetes does not expose the packets they exchange, so Wireshark is unable to capture them. In that figure, an arbitrary TFS component issues a request to add a device into the Context Component (messages 2009 and 2015). Then, that component triggers the recording of that device into the blockchain (message 2030). To do that, the arbitrary component issues a “RecordDevice” request to the DLT Component, that is received by the DLT Connector. The DLT Connector then retrieves the device details from the Context Component (not shown since it is an internal Kubernetes communication) and forwards the request to the DLT Gateway that triggers the upload into the blockchain hosted by NEC in Germany (messages

2056-2885) When the operation is done, the DLT Gateway replies to the DLT Connector and the later replies to the requesting component (message 2888).

No.	Time	Source	Destination	Protocol	Length	Info
2009	4.749	10.0.2.10	10.1.105.117	GRPC	388	SETTINGS[0], HEADERS[1]: POST /context.ContextService/SetDevice, WINDOW_UPDATE[1], DATA[1] (GR
2015	4.750	10.1.105.117	10.0.2.10	GRPC	234	HEADERS[1]: 200 OK, WINDOW_UPDATE[1], DATA[1] (GRPC) (PROTOBUF), HEADERS[1], WINDOW_UPDATE[0]
2030	4.751	10.0.2.10	10.1.105.77	GRPC	392	SETTINGS[0], HEADERS[1]: POST /dlt.DltConnectorService/RecordDevice, WINDOW_UPDATE[1], DATA[1]
2056	4.763	10.1.105.77	teraflow.nlehd.de	TLSv1.2	1337	Application Data
2063	4.815	teraflow.nlehd.de	10.1.105.77	TLSv1.2	108	Application Data
2069	4.820	teraflow.nlehd.de	10.1.105.77	TCP	1516	7051 → 47786 [ACK] Seq=53 Ack=1321 Win=32729 Len=1460 [TCP segment of a reassembled PDU]
2073	4.826	teraflow.nlehd.de	10.1.105.77	TLSv1.2	637	Application Data
2078	4.841	10.1.105.77	teraflow.nlehd.de	TLSv1.2	1720	Application Data
2084	4.842	10.1.105.77	teraflow.nlehd.de	TLSv1.2	1720	Application Data
2104	4.916	teraflow.nlehd.de	10.1.105.77	TLSv1.2	108	Application Data
2105	4.916	teraflow.nlehd.de	10.1.105.77	TLSv1.2	108	Application Data
2112	4.916	teraflow.nlehd.de	10.1.105.77	TCP	1516	7051 → 39986 [PSH, ACK] Seq=53 Ack=1696 Win=32737 Len=1460 [TCP segment of a reassembled PDU]
2113	4.916	teraflow.nlehd.de	10.1.105.77	TCP	1516	9051 → 50938 [ACK] Seq=53 Ack=1696 Win=32737 Len=1460 [TCP segment of a reassembled PDU]
2120	4.917	teraflow.nlehd.de	10.1.105.77	TLSv1.2	737	Application Data
2121	4.917	teraflow.nlehd.de	10.1.105.77	TLSv1.2	730	Application Data
2128	4.918	10.1.105.77	teraflow.nlehd.de	TLSv1.2	112	Application Data
2848	7.071	teraflow.nlehd.de	10.1.105.77	TCP	1516	7051 → 47798 [ACK] Seq=1 Ack=1 Win=31879 Len=1460 [TCP segment of a reassembled PDU]
2849	7.071	teraflow.nlehd.de	10.1.105.77	TCP	1516	7051 → 47798 [PSH, ACK] Seq=1461 Ack=1 Win=31879 Len=1460 [TCP segment of a reassembled PDU]
2850	7.071	teraflow.nlehd.de	10.1.105.77	TCP	1516	7051 → 47792 [ACK] Seq=1 Ack=1 Win=31879 Len=1460 [TCP segment of a reassembled PDU]
2851	7.071	teraflow.nlehd.de	10.1.105.77	TCP	1516	7051 → 47792 [PSH, ACK] Seq=1461 Ack=1 Win=31879 Len=1460 [TCP segment of a reassembled PDU]
2852	7.071	teraflow.nlehd.de	10.1.105.77	TCP	1516	7051 → 47798 [ACK] Seq=2921 Ack=1 Win=31879 Len=1460 [TCP segment of a reassembled PDU]
2853	7.071	teraflow.nlehd.de	10.1.105.77	TCP	1516	7051 → 47798 [PSH, ACK] Seq=4381 Ack=1 Win=31879 Len=1460 [TCP segment of a reassembled PDU]
2854	7.071	teraflow.nlehd.de	10.1.105.77	TCP	2976	7051 → 47792 [ACK] Seq=2921 Ack=1 Win=31879 Len=2920 [TCP segment of a reassembled PDU]
2871	7.072	teraflow.nlehd.de	10.1.105.77	TLSv1.2	1282	Application Data
2872	7.072	teraflow.nlehd.de	10.1.105.77	TLSv1.2	1282	Application Data
2881	7.082	teraflow.nlehd.de	10.1.105.77	TCP	5896	9051 → 40978 [ACK] Seq=1 Ack=1 Win=31879 Len=5840 [TCP segment of a reassembled PDU]
2885	7.082	teraflow.nlehd.de	10.1.105.77	TLSv1.2	1282	Application Data
2888	7.088	10.1.105.77	10.0.2.10	GRPC	219	HEADERS[1]: 200 OK, WINDOW_UPDATE[1], DATA[1] (GRPC), HEADERS[1], WINDOW_UPDATE[0]

Figure 32. Transport Network Topology for DLT Evaluation.

Performance Evaluation

The metric considered for assessing the DLT Component is the delay incurred by both the DLT Connector and the DLT Gateway, which includes the processing time to upload the records into the DLT.

The elapsed time takes into consideration the workflow shown in Figure 30 and described above: i) an arbitrary TFS component requests to upload a record; ii) the DLT Connector receives the request and collects the required information from the Context Component; iii) the DLT Connector requests to the DLT Gateway to retrieve the record associated with the record to be updated/deleted to decide whether a create/update/delete is required; iv) the DLT Gateway contacts the configured peer to retrieve the record information and status; v) the DLT Connector prepares and issues the appropriate create/update/delete record to the DLT Gateway; and vi) when the record is uploaded into the blockchain, the DLT Gateway returns its status to the DLT Connector. Given that RecordAll* methods are just a filter-and-loop extension of single-entity record methods, RecordAll* methods are not considered in this assessment.

To produce the CDF (Cumulative Distribution Function) of the DLT delay, 100 requests are generated uniformly distributed between queries of layer 2 and layer 3 network services and slices. The endpoints of every request are chosen randomly from the transport topology shown in **Figure *31***. Each request is generated with a Poisson statistical model whose inter-arrival time is set to 200ms while the duration of the service/slice is modelled exponentially with a holding time of 10s. Each time a request is dispatched, the affected records are uploaded to the blockchain through the DLT, i.e., for slices, a slice, a service, and the traversed devices are uploaded, while for services, only the service, and the traversed devices are uploaded.

Figure *33* shows the CDF of the DLT latency for the 100 requests. We observe that the delay in general is close to 10 seconds. The main contribution of this delay is due to the cost of uploading the record into the blockchain due to the consensus and ordering constraints that need to be fulfilled.

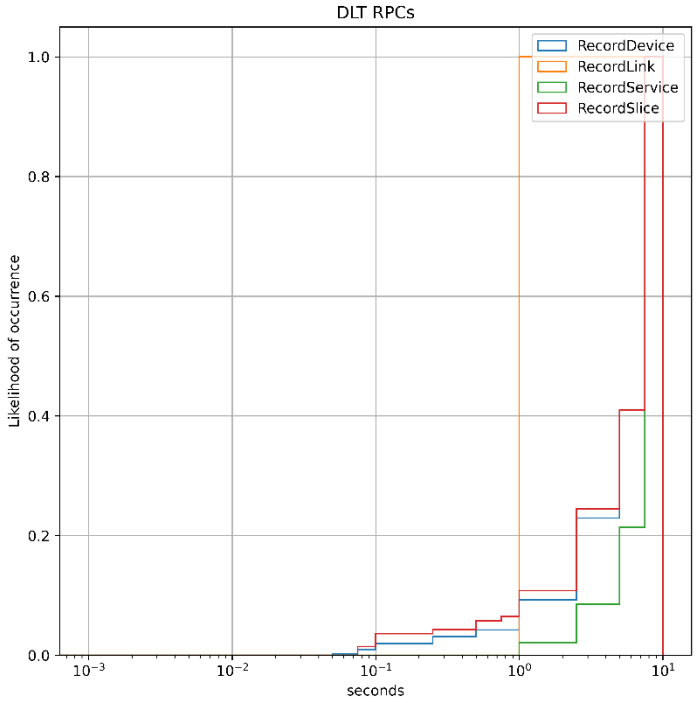

Figure 33. CDF for the DLT Delay.

4.2 Blockchain Improvements

Existing blockchain technologies suffer from multiple shortcomings, such as lack of security for smart contracts that lead to hundreds of millions of US Dollars-worth of tokens being stolen over recent years, limited scalability that hinders adoption for decentralised applications (Dapps) and the ever-growing size of the blockchain storage, due to nodes having to store all the transactions that ever happened on the blockchain.

In this section, we summarise our research contributions on blockchain technologies. More details are available in the following publications:

1. "*On the Storage Overhead of Proof-of-Work Blockchains*" [37]
2. "*Mitosis: Practically Scaling Permissioned Blockchains*" [37]
3. "*Practical Mitigation of Smart Contract Bugs*" [38]

4.2.1 Securing Smart Contracts

In spite of their popularity, developing secure smart contracts remains a challenging task. Existing approaches are either impractical as they do not support many complex real-world contracts or leave the burden to developers for fixing bugs.

Several high-profile attacks on the Ethereum blockchain, such as the *TheDAO*[3] attack, the *Parity Multisig Wallet*[4] attack, and the more recent *Lendf.me* and *Uniswap*[5] attacks have fuelled research to strengthen the security of smart contracts. However, all the proposed solutions are impractical for smart contract developers today as they are typically hard to use, suffer from false positives and negatives, or require huge manual effort by developers. A recent study reveals that 75% of the survey respondents would like to leverage public blockchains such as Ethereum, but cite security as a key concern to high adoption [55].

Recently, several automated smart contract bytecode rewriting approaches have been proposed, but these cannot handle complex attack patterns such as re-entrancy, are incompatible with the upcoming eWASM (Ethereum WebAssembly) bytecode format, their inserted security checks cannot be easily verified and inspected by developers, and are very specific to Ethereum and as such cannot be integrated within other blockchain platforms such as Hyperledger Fabric.

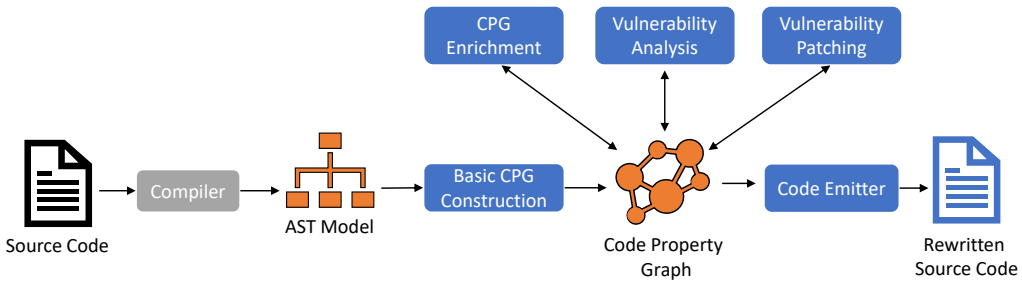

Figure 34. Overview of our Source-Code Hardening Tool.

We built the first practical smart contract compiler, which automatically inserts security hardening checks at the source-code level. Our tool builds a code property graph (CPG) to model control-flows and data-flows of a given smart contract and detect potential vulnerabilities. Our tool relies on the Abstract Syntax Tree (AST) model generated by the compiler or using AST parsing tools such as ANTLR to generate the CPG. Thanks to the CPG notation, our detection and patching mechanisms are agnostic of the smart contract platforms and programming languages. It can therefore be easily extended to any new platform by replacing the compiler with the correct compiler or using ANTLR, updating the CPG enrichment to also consider platform specifics (e.g., storage access pattern, etc.) and updating the code emitter to ensure it emits code in the correct programming language.

We demonstrated the effectiveness of this approach on Ethereum's Solidity smart contracts and show that it efficiently mitigates re-entrancy and integer bugs. We also show how to integrate our tool within enterprise blockchain platforms such as Hyperledger Fabric and NEC Blockchain. Our evaluation on 10k real-world contracts demonstrates that our tool is highly practical and effectively prevents all 122 verified attack transactions in our dataset. The paper detailing the analysis and evaluation of our tool can be found at [20] and is currently under submission at a top tier security conference.

4.2.2 Improving Blockchain Scalability through Effective and Dynamic Sharding

Scalability remains one of the biggest challenges to the adoption of blockchain technologies for large-scale deployments. On the one hand, permissionless systems such as Bitcoin provide a rather weak notion of “eventual consensus”: while blocks are generated at a regular pace, only sufficiently deep

[3] https://crypto.news/learn/the-dao-attack-understanding-what-happened/
[4] https://blog.openzeppelin.com/on-the-parity-wallet-multisig-hack-405a8c12e8f7/
[5] https://peckshield.medium.com/uniswap-lendf-me-hacks-root-cause-and-loss-analysis-50f3263dcc09

blocks become stable with high probability. This probabilistic consistency guarantee leads to high transaction confirmation times and, consequently, high latency. In contrast, permissioned blockchains exhibit low latencies at the cost of poor scalability in the number of blockchain nodes.

Various solutions have been proposed to capture “the best of both worlds”, targeting low latency and high scalability simultaneously, with blockchain sharding emerging as the most prominent scaling technique. Most existing sharding proposals are tailored to the permissionless model and specific blockchain deployments. A few sharding methods have been proposed for the permissioned model, however, they rely on strong trust assumptions that are not justified in practice. In particular, sharding solutions for permissioned systems assume a static set of blockchain nodes that are always available. However, real-world blockchain deployments require that blockchain nodes can join and leave the system dynamically.

To address this problem, we designed a novel approach to improve the scalability of permissioned blockchains while preserving the requirement of dynamic participation. Our proposal, dubbed MITOSIS, allows the dynamic creation of new blockchains, as more participants join the system, to meet practical scalability requirements. Crucially, MITOSIS enables the division of an existing blockchain (and its participants) into two — reminiscent of mitosis, the biological process of cell division. MITOSIS inherits the low latency of permissioned blockchains while preserving high throughput via parallel processing. Newly created chains are fully autonomous, can choose their own consensus protocol, and yet they can interact with each other to share information and assets, achieving high levels of interoperability. Our solution can be ported with little modification and manageable overhead to existing permissioned blockchains such as Hyperledger Fabric. Our detailed solution [19] was published at the Annual Computer Security Application Conference (ACSAC) 2021.

4.2.3 Optimising Blockchain Storage.

Blockchain systems such as Bitcoin have long been criticised for their high computational and storage overhead. Unfortunately, while a number of proposals address the energy consumption of existing Proof-of-Work deployments, little attention has been given so far to remedy the storage overhead incurred by those blockchains. In fact, it seems widely accepted that full nodes supporting the blockchains have to volunteer hundreds of GBs of their storage, to store and verify all transactions exchanged in the system. We explored the solution space to effectively reduce the storage footprint of Proof-of-Work based blockchains. To do so, we analysed, by means of thorough empirical measurements, how existing full blockchain nodes utilise data from the shared ledger to validate incoming transactions/blocks. Based on this analysis, we show that it is possible for full nodes to locally reduce their storage footprint to approximately 15 GB, a close to 96% reduction in storage requirements, without any modification to the underlying blockchain protocol. We also investigate other client-side strategies to further reduce the storage footprint while incurring negligible computational overhead on the nodes. Our detailed results [18] were published at the IEEE International Conference on Blockchain (Blockchain) 2022.

5 Interworking Across Beyond 5G Networks

This section tackles the heterogenous and external interactions supported by the TFS Controller from different perspectives: i) via a defined NorthBound Interface (NBI) which allows exposing different interfaces for diverse interaction purposes such as an NFV Orchestrator, or to allow an external entity handling the lifecycle of both connectivity services and slices; ii) via a new defined TFS component called WebUI which provides a friendly tool to retrieve context, service, and slice information along with real monitoring data; and iii) interworking between different TFS Controller instances to support multi-domain network connectivity services. These three external interactions offered by the TFS Controller are described in detail not only from an architectural and interface point of view, but also complemented with selected and dedicated studies/activities to validate their supported functionalities and operations.

5.1 NBI Component (Formerly Compute Component)

This component was formerly named the Compute Component in D4.1 [40]. That former TFS component allowed an external NFV Orchestrator (such as ETSI OSM) that requested a network connectivity service between pairs of remote data centre sites to deploy a network service (i.e., including both cloud and network resources). Thereby, the Compute Component's objective was to map the REST API contents supplied by the NFV Orchestrator when requesting a network connectivity service to the targeted gRPC API supporting communications with other TFS components such as the Service, Context, or PathComp Components. However, driven by new requirements to support more advanced functionalities and interactions with external entities (other than an NFV Orchestrator), the Compute Component has been generalised to become the new NBI Component. Thus, the idea is to centralise heterogeneous and multi-purpose interactions with external entities (e.g., NFV Orchestrators, OSS/BSS, clients, etc.) within a single component/micro-service. The goal is to end up with a simpler and more scalable TFS architecture. The following reports the architecture and interfaces supported by the NBI Component.

5.1.1 New Features/Extensions

- Service creation and termination using the ETSI OSM REST API and interactions of the NBI Component with other TFS components such as the Service, Context, and PathComp Components.
- Support the interactions (via REST API) to handle the lifecycle management (LCM) of the network connectivity services steered by the ESTI OSM.
- Support of the IETF L2VPN Service Model to query the establishment of Layer 2 VPN connectivity services.
- Support for demanding network slices with specific SLAs.
- Enable the detailed definition of the L2VPN Network Model.
- Offer exposure of topology information to an external system

5.1.2 Final Design

Figure 35 depicts the architecture of the NBI Component and its main internal functional blocks. As stated above, the NBI Component is intended to act as the frontend of the TFS to interact with external entities (e.g., ETSI OSM, OSS/BSS, …). Thus, the NBI becomes an expandable component to be

continuously enhanced to support the interactions with other upcoming external elements that will interact with the TFS.

As illustrated in the architecture, the NBI exposes two interfaces: REST API to interact with the external entities, and gRPC to communicate with other components. This provides the TFS the appealing capability to quickly adapt and offer its supported functionalities to other external entities. At the time of writing, the NBI supports the following multi-purpose interactions: L2VPN Service Model (SM), the SLICE, the Topology, and the L2VPN Network Model (NM).

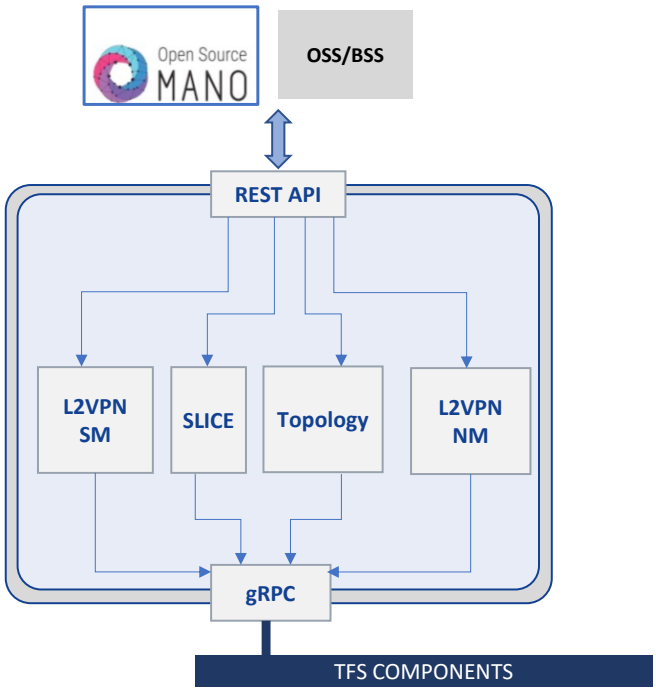

Figure 35. Architecture and Functional Blocks of the NBI Component.

5.1.3 Final Interfaces

This section tackles the description of the current set of supported interfaces within the NBI Component. These interfaces cover different/complementary functions to allow interaction with external systems. The first is the so-called "IETF topology" to disclose (abstracted) transport details upon demand to, e.g., a high-layer orchestrator entity. The "L2VPN service model" interface supports that external entity requesting the establishment of L2VPN network connectivity specifying endpoints (e.g., cloud site gateways) and the requirements (in terms of bandwidth, latency, etc.) to be fulfilled. Finally, the "IETF slice" interface aims to support the request of a specific partition of the transport infrastructure to be managed by the TFS Controller.

5.1.3.1 IETF Topology

The purpose of this interface is to expose details of the underlying transport infrastructure managed by the TFS Controller to an external system such as an upper-layer orchestrator (e.g., OSM), OSS/BSS, parent network/resource orchestrator (aligned with hierarchical network control), etc. This interface is supported over a REST API. The NBI Component realises a mapping/translation function of the topological information (i.e.., devices and links) stored in the Context Component. To this end, upon a request being sent by the external system (via a REST GET method), the NBI Component communicates with the Context Component (using gRPC API) to query the transport topology details. This information is then encoded into the REST API response delivered then to the external system. In

the current version of TFS, this information entails: devices with their IDs and endpoints along with specific characteristics (e.g., device type), and links with their IDs as well as specific attributes such as available capacity, latency, cost, etc.

5.1.3.2 L2VPN Service Model

This interface is based on the model defined in [41] which is implemented through a RESTConf API with an encoding based on JSON. This allows layer 2 connectivity services with a rich set of SLA constraint to be requested of the TFS Controller. For instance, it can specify QoS policies (e.g., traffic shaping, latency and jitter bounds, acceptable packet loss, etc.). The interface allows indicating the VPN service (VLAN ID), the bearer reference, the expected constraints (e.g., supporting primary and backup paths for a single service), etc. A demonstration of creating protected L2VPN services automatically between remote data centre sites using the L2VPN Service Model Interface is reported in [81].

5.1.3.3 L2VPN Network Model

The NBI Component will support the L2VPN Network Model according to the definition in [43] and as reported in deliverable D3.2 [44]. For the sake of completeness, this model covers required network orchestration and control mechanisms such as transport protocol selection, operational parameters, etc.

5.1.3.4 IETF Slice

The IETF Slice interface provides a specific interface to support the creation, modification, deletion, and monitoring of IETF network slices. A defined model is supported covering specific attributes such as the Service Demarcation Points that form the Slice, and covering P2P, P2MP, or Any-to-Any connections, the Service Level Expectations (SLE), and the Service Level Objectives (SLO), etc. Detailed information about this interface integrated in the NBI Component is described in deliverable D3.2 [44].

5.1.4 Evaluation

Evaluation of the NBI Component has been the subject of two different studies: i) the creation of L2VPN services demanded by an NFV Orchestrator; ii) tackling the reduction of the power consumption targeting low-energy networks. Both studies are described from a functional perspective. For the former, the experimental validation of the creation of L2VPN services was conducted in a conference demonstration. For the latter, the adopted assumptions modelling the energy consumption in a transport infrastructure formed by routers/packet switches along with the proposed traffic-aware energy-efficient algorithm are reported. Numerical results of this algorithm and its comparison with a benchmark algorithm without energy consumption considerations will be specifically tackled in in the upcoming TeraFlow D53 deliverable.

5.1.4.1 L2VPN Services

This study focuses on the validation of the creation of L2VPN services triggered by an NFV Orchestrator (OSM) when interconnecting two remote data centre sites with a protected connectivity service. Figure 36 shows the deployed setup. As mentioned, the OSM controller rolls out a network service instantiating computing functions at both data centres controlled by OpenStack. Then, it queries the creation of a protected L2VPN service between PE1-PE3 for the primary route, and PE2-PE4 for the backup route via the IETF L2VPN API exposed by the NBI Component. Once the L2VPN service is demanded and managed by the TFS, the underlying transport devices (packet and optical) need to be configured. In this study/validation the device configuration was emulated, whilst the rest of the

component interactions within the TFS were observed. Figure 37 shows the RESTConf messages exchanged between the OSM and TFS (via the NBI Component) to eventually create the L2VPPN service. In the validation, this takes around 1.12 s (without actual hardware configuration since it is emulated).

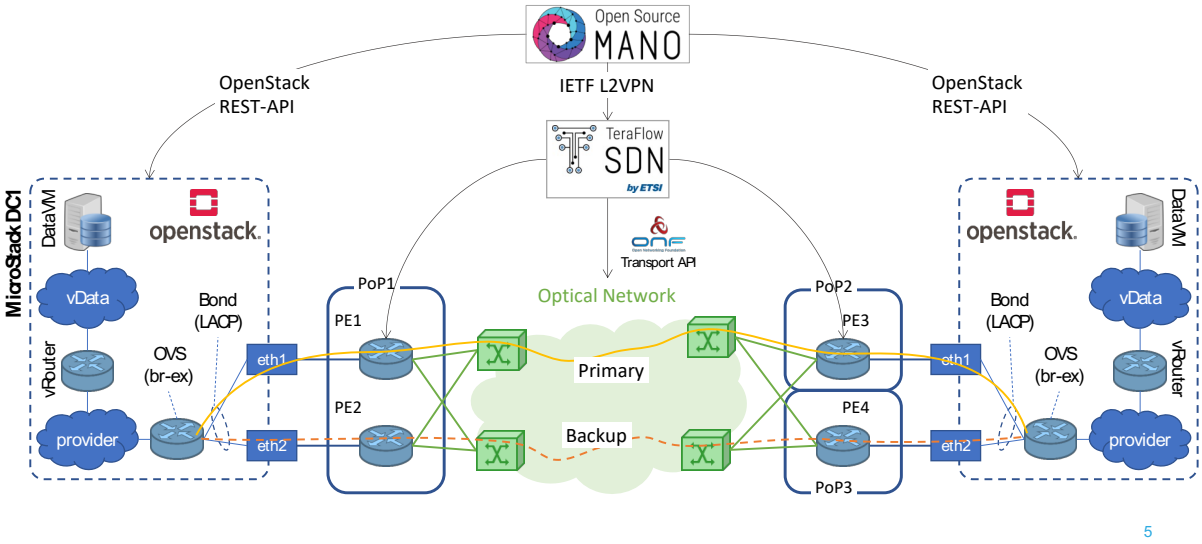

Figure 36. L2VPN Service Creation.

Source	Destin	Protocol	Length	Info
OSM	TFS	HTTP	305	GET /restconf/data/ietf-l2vpn-svc:l2vpn-svc/vpn-services
TFS	OSM	HTTP/JSON	71	HTTP/1.0 200 OK
OSM	TFS	HTTP/JSON	226	POST /restconf/data/ietf-l2vpn-svc:l2vpn-svc/vpn-services
TFS	OSM	HTTP/JSON	71	HTTP/1.0 201 CREATED
OSM	TFS	HTTP/JSON	643	POST /restconf/data/ietf-l2vpn-svc:l2vpn-svc/sites/site=DC1/site-network-accesses/
TFS	OSM	HTTP	176	HTTP/1.0 204 NO CONTENT
OSM	TFS	HTTP/JSON	643	POST /restconf/data/ietf-l2vpn-svc:l2vpn-svc/sites/site=DC1/site-network-accesses/
TFS	OSM	HTTP	176	HTTP/1.0 204 NO CONTENT
OSM	TFS	HTTP/JSON	643	POST /restconf/data/ietf-l2vpn-svc:l2vpn-svc/sites/site=DC2/site-network-accesses/
TFS	OSM	HTTP	176	HTTP/1.0 204 NO CONTENT
OSM	TFS	HTTP/JSON	643	POST /restconf/data/ietf-l2vpn-svc:l2vpn-svc/sites/site=DC2/site-network-accesses/
TFS	OSM	HTTP	176	HTTP/1.0 204 NO CONTENT
OSM	TFS	HTTP	355	GET /restconf/data/ietf-l2vpn-svc:l2vpn-svc/vpn-services/vpn-service=b93... 9f16a4/
TFS	OSM	HTTP/JSON	71	HTTP/1.0 200 OK

Figure 37. Exchanged RESTConf Messages though the NBI Component between the OSM and TFS.

5.1.4.2 Energy-Aware Routing

One of the objectives to be tackled in the TeraFlow project is to contribute to reducing the energy consumption thanks to adopted algorithms that combine centralised computing elements and low-energy networks. The scenario depicted in Figure 38 is considered. The goal is to deploy network services which entail the instantiation of Virtual Network Functions (VNFs) over distributed data centres which are interconnected by a transport network (e.g., formed by a pool of routers and packet switches).

Upon the arrival of a new network service, the NFV Orchestrator (based on ETSI OSM) selects the cloud sites to deploy the VNFs. Besides the VNFs, a network service describes the virtual links which provide the connectivity between pairs of VNFs. The mechanism/algorithm handling the placement of the VNFs over the diverse data centres is out of the scope of this study. Indeed, it is assumed that once the NFV Orchestrator decides the computing sites to host the VNFs, it queries to the TFS Controller to set up the network connectivity services to accommodate the virtual links interconnecting the VNFs

at the different data centres. To this end, the OSM relies on a WAN Infrastructure Manager (WIM) (as addressed in Section 5.1.4.1) to request those connectivity services to the TFS Controller based on a REST API. Such a request carries the pair of Provider Edges (PEs) physically connected to the data centres, the demanded bandwidth (e.g., in Mb/s), and the maximum tolerated latency (e.g., in ms). In other words, for each virtual link between a pair of VNFs hosted in different data centres, the TFS needs to seek and allocate the network resources which fulfil the virtual links requirements. This objective is pursued and complemented with attaining a reduced amount of energy consumed in the whole network. That is, when selecting the routers/packet switches to deploy the connectivity service, it is done in a way that the resulting consumed energy caused by using devices and ports is minimised. The final objective is thus to accomplish a good trade-off between transport resource utilisation and energy consumption. This objective is seen by the telecom operators as crucial, since some estimations state that the electricity used by networks entails around 3-5% of the global energy demands [45].

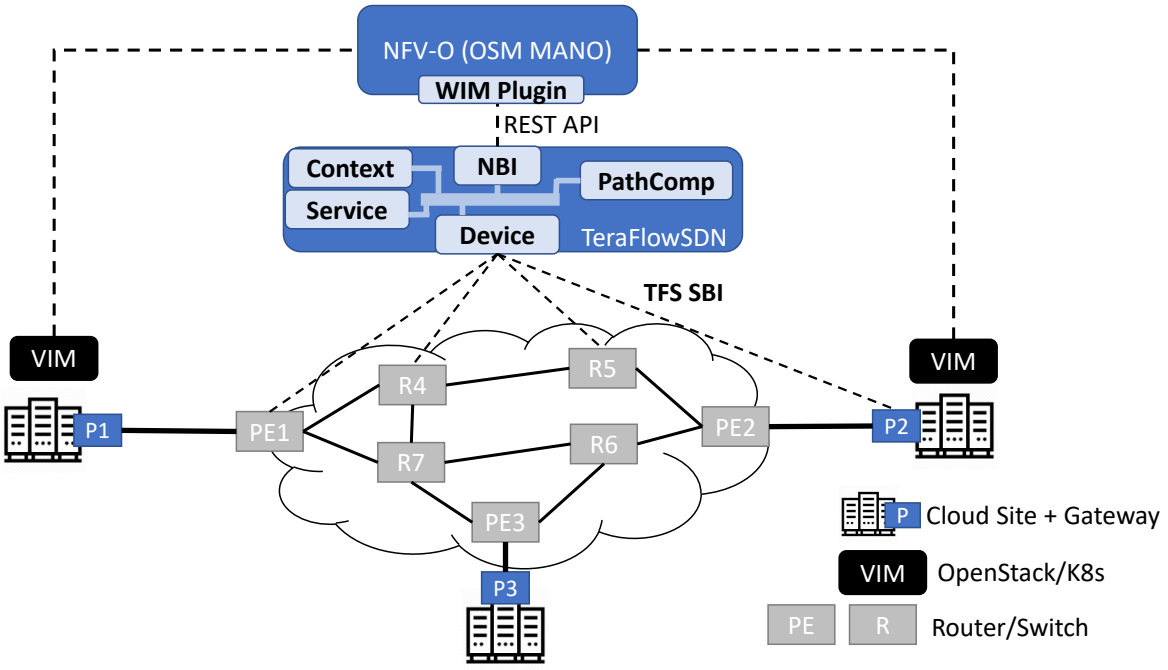

Figure 38. Compute and Network Scenario for Energy Efficient Routing.

Bearing in mind the above macroscopic objective, the TFS Controller is enhanced, specifically adopting a traffic aware energy efficient routing algorithm to be triggered in the PathComp Component [44]. This topic is not new and has received notable attention in the literature, e.g., [46][47][48][49]. In general, since most of the time the network resources (e.g., link ports) are under-utilised (around 30-40%), the idea is to sleep/turn off devices and network elements with low traffic volume. This is the basic principle being followed by the energy-aware routing algorithm adopted in the TeraFlow project. Prior to describing the devised algorithm and the interactions of the TFS components when setting up a new connectivity service (tied to e.g., a virtual link of a network service), it is worth presenting the considered energy model of the network devices.

- **Energy model of the considered underlying transport network devices**

  Herein, we exclusively focus on routers and packet switches as the transport devices forming the network infrastructure interconnecting distributed data centres. These routers/switches are based on the typical store-and-forward switch architecture [47], shown in Figure 39. The device is in general formed by a set of line cards, where each one has a pool of ports (*n*) interconnected by

a circuitry with (input/outputs) memories to store the packets and Ternary Content Addressable Memory (TCAM) used for realising the quick routing/forwarding of the stored packets. A switch fabric allows interconnecting the set of line cards forming the router. In this model, the power consumed by the entire router/switch depends on the traffic being transported, i.e., the number of activated ports and line cards and the utilisation of the TCAM (formed by chips and circuitry) [46][47]. Moreover, there are other elements in the routers and packet switches that increase the energy consumption called the environment, e.g., the fans to cool the device.

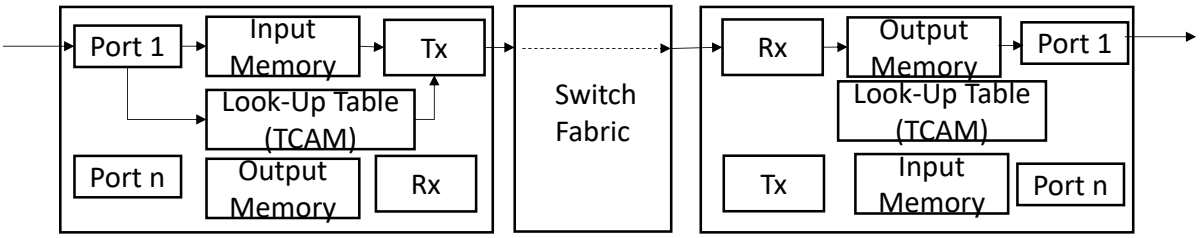

Figure 39. Adopted Store-and-Forward Switch Architecture.

With the above basic architectural description of the considered transport devices, the energy consumption model (being widely used in the literature, e.g., [49]) can be determined with analytical expressions dependent on the amount of traffic being handled/routed. The power consumption of a device/node can be macroscopically described by:

$$P_{device} = P_{ctlr} + P_{environment} + P_{data}$$

The P_{ctlr} is bound to the power needed to manage the switch and routing functions (i.e., interactions with the TFS controller through the Device Component and using the TFS SBI interface, see Figure 38). The $P_{environment}$ is the power consumption associated with the environment equipment, such as fans. The P_{data} has two contributing components: 1) a constant baseline power since the device is powered on regardless of traffic being transported; 2) the power consumed for the traffic being processed, stored, switched, and forwarded. Typically, the P_{ctlr}, $P_{environment}$, and P_{data} baseline are grouped into P_{idle}. Considering that, the power consumption (in Watts) for a device active formed by *L* links/ports is expressed as:

$$P_{device} = P_{idle} + \sum_{(i,j)\in L} P_{(i,j)} \cdot x_{i,j}$$

where $P_{i,j} = E_{bit} \cdot \tau_{i,j}$. E_{bit} describes the total aggregated energy (in Joules) to process (lookup in TCAM), store, and forward a bit (in a packet) over a link $l_{i,j}$ connecting device *i* and device *j*. $x_{i,j}$ is a binary which takes 1 if the link $l_{i,j}$ is on and 0 otherwise. $\tau_{i,j}$ accounts for the amount of data traffic (b/s) being transported over that link $l_{i,j}$.

Therefore, the total power (in Watts) being consumed by the *N* devices forming the network can be determined by:

$$P_{total} = \sum_{n\in N} P_{idle} \cdot y_n + \sum_{n\in N} \sum_{(i,j)\in L} P^n_{(i,j)} \cdot x^n_{i,j}$$

y_n is a binary value indicating 1 if the switch is active (i.e., powered on) and 0 otherwise. Thus, the objective of the traffic-aware energy-efficient algorithm is to select the nodes and links fulfilling the network connectivity demands (e.g., bandwidth, latency) while achieving the lowest P_{total}.

- **Traffic-aware energy-efficient algorithm**

As commented above, when a new connectivity service request arrives at the TFS Controller, the path and network resources in the transport infrastructure are decided by the PathComp Component (see Figure 38) triggering a dedicated algorithm. The proposed algorithm uses as input: i) the connectivity service request detailing the endpoints (PEs), the demanded bandwidth, and the maximum permitted latency; ii) the context information embracing the network topology, the link bandwidth utilisation, and the energy parameters such as the P_{idle} and the $P_{i,j}$. The algorithm output will explicitly determine the list of devices (nodes and links) where the connectivity service will be set up.

The algorithm is based on a K-Shortest Path (ksp) computation which seeks for a feasible path connecting the endpoints while meeting both bandwidth and latency requirements. First, the algorithm tries to only consider those nodes and links which are active, being used to carry existing data traffic. The resulting set of paths (i.e., ksp) are then processed and re-sorted with respect to their resulting P_{total}. The final set of ksp is finally ordered as an ascending list according to the P_{total}. If two or more paths in the ksp yield the same P_{total}, then the path with lowest traversed devices is sorted first. Otherwise, if no path is found over the active context, then the computation is re-considered using the whole network/context formed by both active and turned off/sleeping devices and links. This triggers the K-Shortest Path computation whose output ksp is also ordered according to the P_{total}. If no path can be computed (lack of available bandwidth or unable to meet the latency constraint), then the connection is blocked.

As mentioned before, we provide a high-level description of the algorithm executed in the PathComp Component to reduce the overall network energy consumption. The validation and results of adopting this algorithm will be conducted in the D5.3 deliverable.

- **Workflow for establishing a new connectivity service request**

Figure 40 shows the workflow for establishing the connectivity services required to deploy a network service coordinated by the OSM. As mentioned above, the connectivity services requested of the TFS Controller facilitate the deployment of the virtual links between VNFs placed at remote data centres. The connectivity services can be either L3VPN or L2VPN. However, here we specifically focus on the latter. Bearing this in mind, once the VNFs are placed by the OSM, it delegates management of the connectivity services to the Controller. The NBI processes the incoming connectivity services from the OSM and delivers the information to the Service Component. That component asks the PathComp Component to find a feasible path fulfilling the requirements while attaining an energy-efficient solution. Prior to triggering the algorithm, the PathComp Component retrieves an updated view of the network infrastructure interacting with the Context Component. If the algorithm succeeds in finding a feasible path, this is returned to the Service Component. Finally, the Service Component interacts with the Device Component to come up with the required transport network configuration on the selected routers.

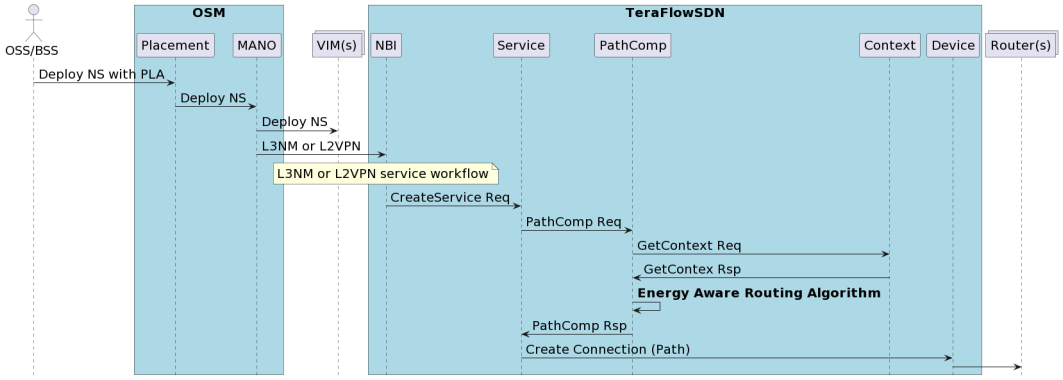

Figure 40. Workflow for setting up a connectivity service for a network service deployment.

5.2 Web User Interface

The Web-based User Interface (WebUI) is a new TFS component/microservice which acts as a user-visible service aiming at offering a friendly toolset to an external client for interacting with the pool of functionalities supported by the Controller. The WebUI enables, for instance, a network operator to manually interact with the TFS Controller to perform configuration operations and inspect the state of the network.

5.2.1 New Features/Extensions

The WebUI allows an external client, OSS/BSS, etc. to:

- Retrieve the topology and context, as well as detailed information of the active services.
- Request network connectivity services.
- Onboard descriptors (e.g., context, topologies, devices, links, services, slices, etc.).
- Plot real time monitoring data collected from different devices (using the Grafana dashboard).
- Interact with other components to handle the above operations.

5.2.2 Final Design

Figure 41 shows the architecture of the WebUI. It offers a REST API to request a defined set of functionalities/pages. At the time, there are seven defined functionalities/pages: Home, Device, Link, Slice, Service, Grafana, and About. The user, via the Web frontend with a specific IP server address and port, gets access and is then granted access to either retrieve information or command a specific operation. Depending on the action, the WebUI determines the associated TFS component involved (if any). That is, the WebUI interacts with the specific TFS component to complete the action. For instance, if details of the set of links are requested, the WebUI queries the Context Component to provide this information, which is then delivered to the external client.

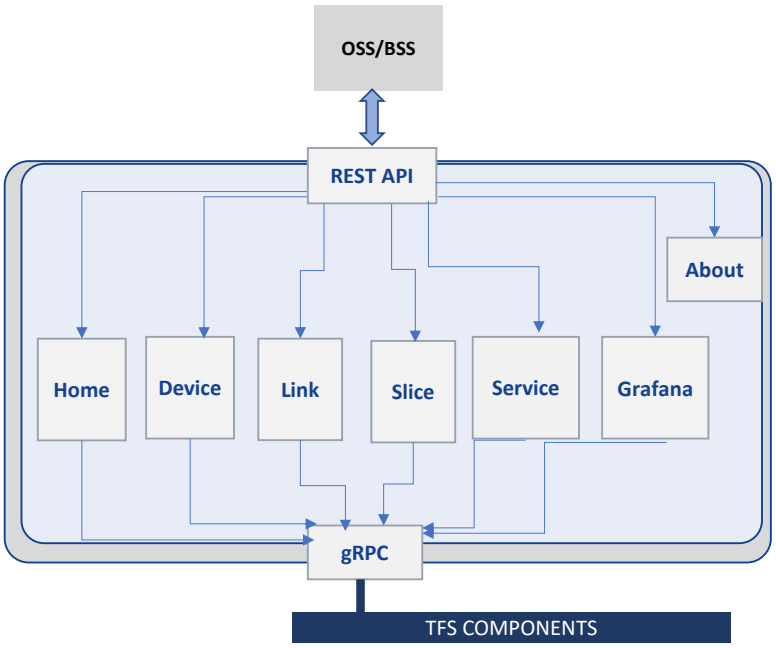

Figure 41. Architecture of the WebUI Component.

- The Home page is used to upload a JSON file containing the descriptors for the devices, links, services, etc. that need to be managed, and to choose the desired context. It also includes a view of the network topology with the devices and links it contains.
- The Device page lists of all devices' details such as the UUID, type, endpoints, drivers, and status. From this list, it grants access to add a device.
- The Link page shows the list of all links, displaying their UUID and endpoints.
- The Service page lists of all the existing connectivity services, specifying their identifiers (in UUID form), type, endpoints, and status. In addition, for a specific connectivity service, it shows the paths (traversed devices and links) and the service constraints.
- The Slice page lists of all transport network slices, and displays their UUID, endpoints, and status. For a specific slice, it can show the associated devices, the service identifier, and the constraints.
- The Grafana page provides a few dashboards that plot real time monitoring data collected from the different network devices.
- The About page contains details and links related to the ETSI TFS Open Source Group.

5.2.3 Final Interfaces

The supported interfaces by the WebUI are:

- The REST API is used to query the WebUI about specific operations and actions from an external user as detailed above.
- The gRPC API allows communication between the WebUI and the rest of the TFS component such as Context, Slice, Service, etc.

5.2.4 Evaluation

The WebUI has been implemented using the Flask Framework, and all the supported actions from the WebUI (i.e., Home, About, Device, Link, Service, Slice, and Grafana) have been validated from a functional perspective. The following figures shows some selected screenshots of the WebUI , e.g., 1)

to request the creation of a new connectivity service (see Figure 42), and 2) to retrieve specific device information (see Figure 43).

For the creation of a new connectivity service, the user determines the context identifier, the service type (e.g., L2VPN), the service endpoints, and the requirements (e.g., diversity in the example in Figure 40). Once it has successfully deployed the service (i.e., resources are allocated), the WebUI shows the related service information such as the status (e.g., ACTIVE) as well as the paths (i.e., devices and links) selected to accommodate the service.

For a specific device (with a determined UUID), the WebUI shows information such as the supported driver to enable the programmability, the type (e.g., packet router), and the endpoints (i.e., ports).

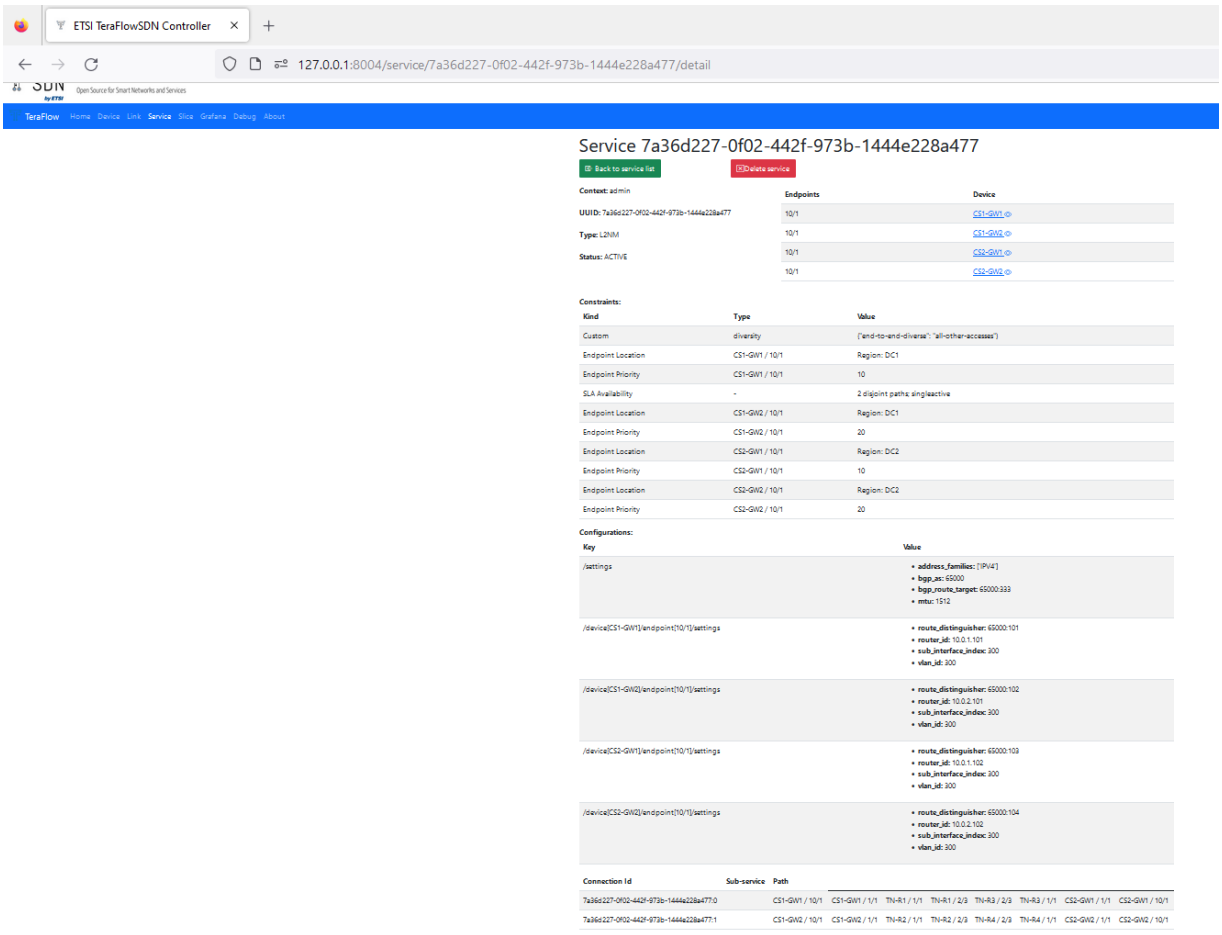

Figure 42. WebUI Component: New Network Connectivity Service Creation.

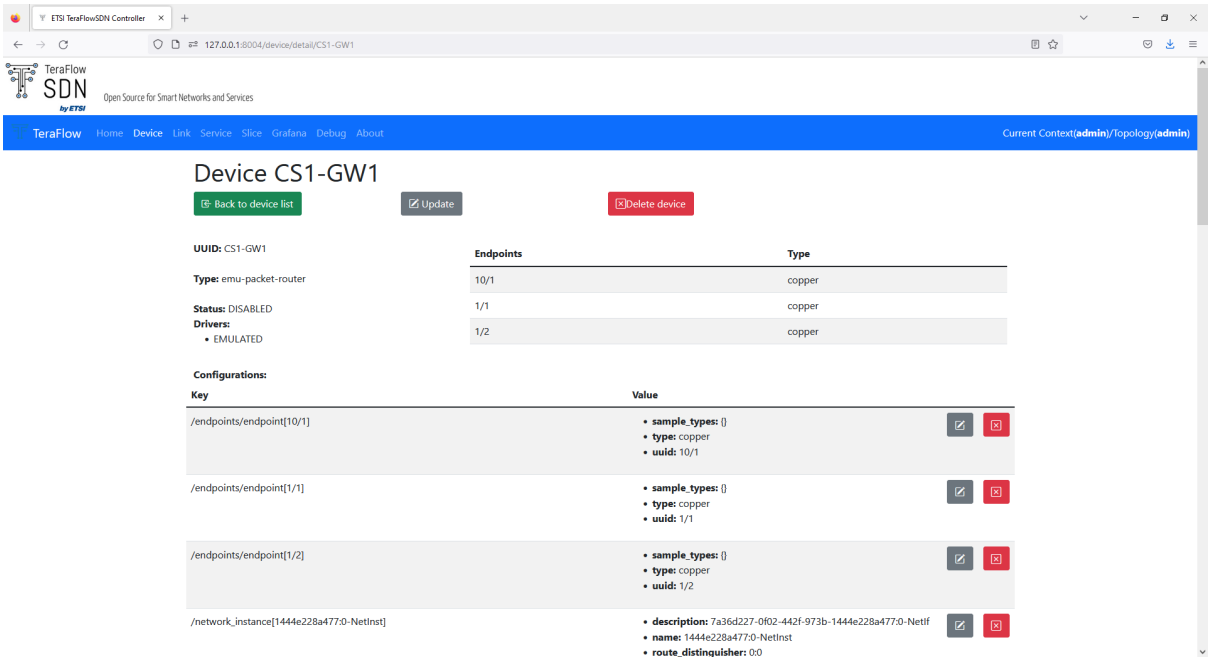

Figure 43. WebUI Component: Retrieving Device Details.

5.3 Inter-domain Component

The Inter-Domain Component of the TFS Controller handles the control interactions with peer TFS Controllers that govern separate network domains. In other words, the Inter-Domain Component is used to establish, update, and delete multi-domain network connectivity services (managed by diverse controllers). The primary functions supported by this component are: i) the exchange of abstracted domain information between peer TFS Controllers; and ii) the activation of slices and services encompassing two or more domains controlled by different TFS instances.

5.3.1 New Features/Extensions

- Computation and maintenance of abstracted topology of the local domain, delegated to the PathComp Component (see [44]).
- DLT-based sharing of local topology abstraction.
- DLT-based ordering and updating of slices in remote domains.

5.3.2 Final Design

Figure 44 illustrates the final design of the Inter-Domain Component (IDC). The main changes to the previous version are as follows. First, information exchange between domains is now performed via the DLT Component rather than a direct interaction between local and remote IDCs. Furthermore, the new PathComp Component is used to compute end-to-end SLA-enabled paths through multiple domains. For this, each local IDC additionally computes and maintains an abstracted version of its domain which can be shared with peer domains.

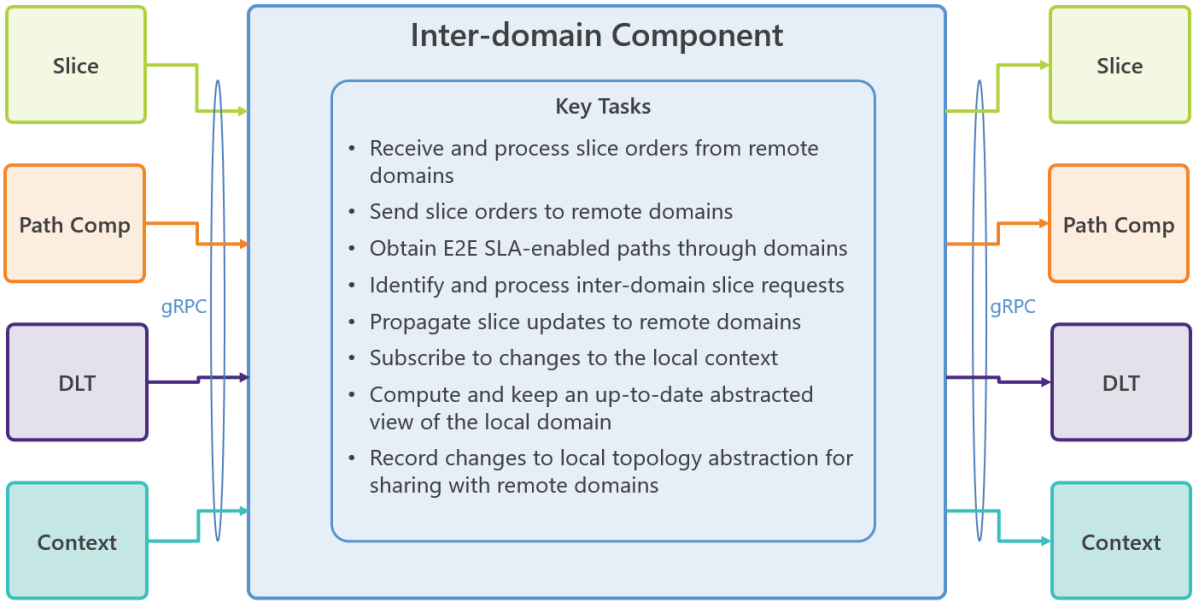

Figure 44. Final Design of the Inter-Domain Component (IDC).

5.3.3 Final Interfaces

The final interfaces of the IDC are summarised in the following table which is organised according to the involved components as well as the corresponding tasks of the IDC and workflows which the interactions are part of.

Table 12. IDC Final Interfaces.

Components	Tasks of IDC	Workflows
IDC ↔ PathComp	• Obtain E2E SLA-enabled path through domains.	• Inter-domain service preparation and activation. • Handling inter-domain SLA violation events.
IDC ↔ Slice, Slice ↔ IDC ↔ DLT	• Identify and process inter-domain slice request. • Propagate slice updates to remote domains.	• Inter-domain service preparation and activation. • Handling inter-domain SLA violation events.
IDC ↔ Context, IDC ↔ IDC (self)	• Subscribe to changes to the local Context Component, in particular events from context, topology, device, link, service, and slice entities. • Using the above information, compute and keep an up-to-date abstracted view of the local domain for sharing with remote domains.	• DLT record exchange (“sharing process for entity X”).

IDC ↔ DLT	• Record changes to local topology abstraction for sharing with remote domains. • Receive and process slice orders from remote domains. • Send slice orders to remote domains.	• DLT record exchange ("sharing process for entity X"). • Inter-domain service preparation and activation.

5.3.4 Evaluation

The evaluation of the Inter-Domain Component is tackled from two different studies, namely: 1) the functional validation of DLT-based end-to-end inter-domain transport slice management; and 2) QoS-enabled inter-domain connectivity.

5.3.4.1 Inter-Domain Component: Functional Validation

[52] reports the public demonstration of Inter-Domain Component functions that aim to automatically deploying inter-domain services in a reliable manner, fulfilling demanded SLAs. To this end, it is necessary to: i) know basic topological information of the domains; and ii) provide appropriate traceability of inter-domain services requests.

To ensure the privacy of the network operators when abstracting their topologies between IDC peers, the blockchain technology has been selected leveraging its consolidated approach to distribute information coming from different sources. On the other hand, to support the traceability of the inter-domain services, a leader domain controller is designated (e.g., the ingress domain) which takes over selecting the sequence of domains, interacts with other domain controllers, monitors the end-to-end SLAs, and handles the appropriate mitigation problems if the SLA is violated, as shown in [53].

The functional demonstration setup for the Inter-Domain Component functions shown in **Figure *45***, where the data plane of each router is emulated. Each domain is controlled by a dedicated TFS Controller instance. Each TFS Controller features a DLT Component (Gateway) which takes over interfacing with a vendor- and operator-agnostic permissioned blockchain used to share the per-domain details with neighbouring domains. The Inter-Domain Component of the Controller becomes the key enabler to support inter-domain transport slices/services.

Upon the arrival of an inter-domain transport slicing service request, the source domain (as the designated leader) decides the sequence of domains to be traversed. Then, it interacts with the respective TFS Controllers using the IDC. This communication is supported by the DLT Component, which stores and detects records in the blockchain corresponding to the inter-domain transport slices.

Figure 46 illustrates the workflow for sharing (via the DLT) the abstracted topology among remote domains (i.e., among TFS Controllers). The workflow starts with the IDC at each domain subscribing to receive notifications on device or link changes tracked by the local Context Component (step 1.1). The entities monitored in this workflow are denoted by the symbol X which can be: context, topology, device, link, service, and slice. Each DLT Component subscribes to receive notifications when a change is made on the blockchain (step 2.1). When a change is stored in the local domain's (A) Context Component, an event is broadcast (step 1.2) to all the subscribed components. The IDC receives the event, and re-computes the local abstracted topology (step 1.3). For every change, it sends a Record request to the DLT Component (steps 1.4 and 1.7), that creates, updates, or marks as deleted the record on the blockchain (steps 1.5 and 1.6), and synchronises the change with the other peers of the

Blockchain (1.8). When the remote domain (N) synchronises the changes (step 2.2), if a record associated with the subscription has been changed, it issues a notification towards the DLT Component containing the unique record identifier (step 2.3). The DLT Component interrogates its associated blockchain peer to retrieve the up-to-date record (2.4 and 2.5) and updates the appropriate record in its Context Component (2.6 and 2.7), that stores the local copy of the domain A's abstracted topology.

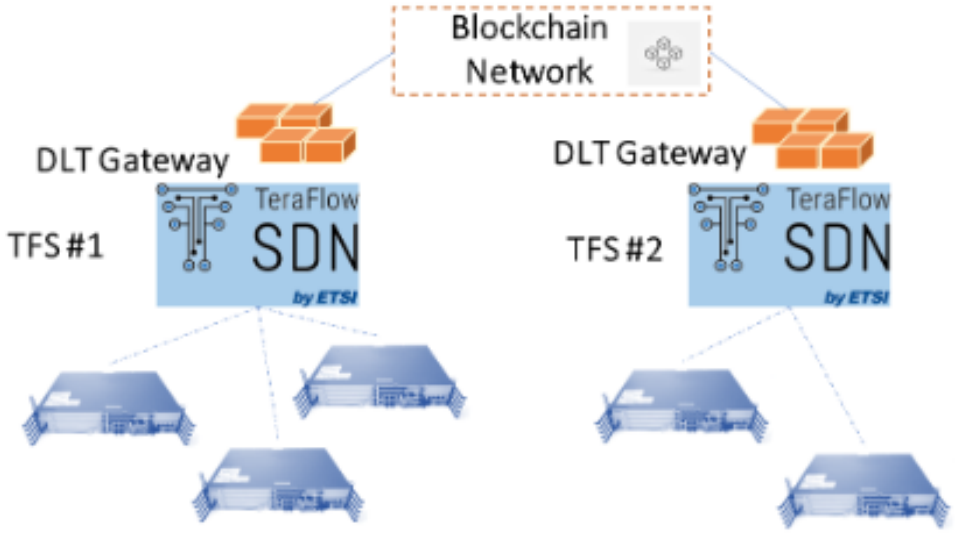

Figure 45. Demonstration Setup for Inter-Domain Component Functions.

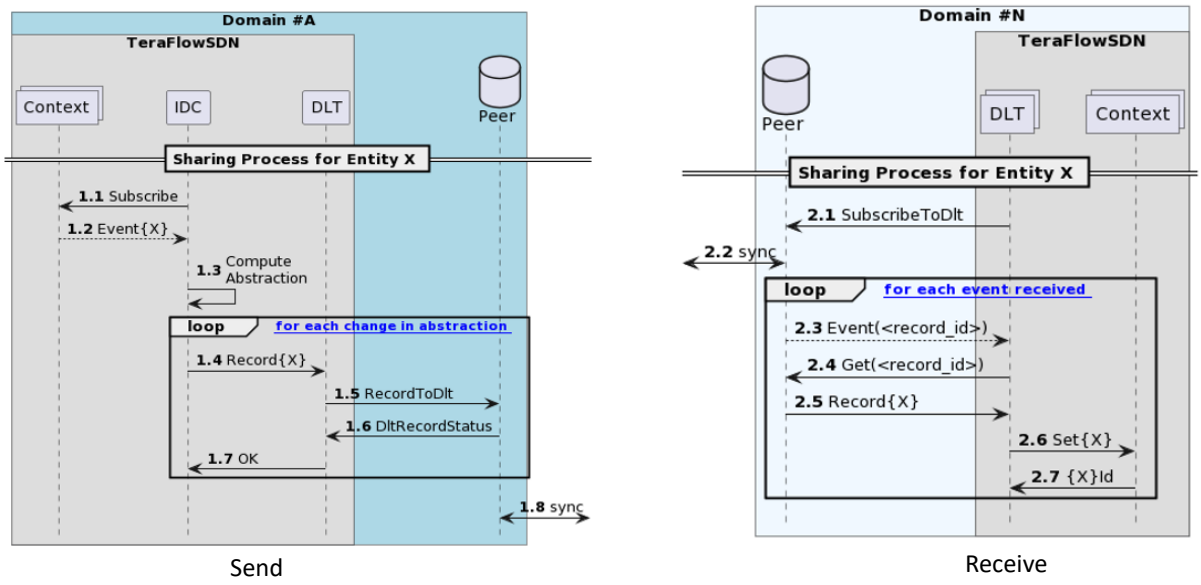

Figure 46. Send/Receive Domain Changes via DLT Component.

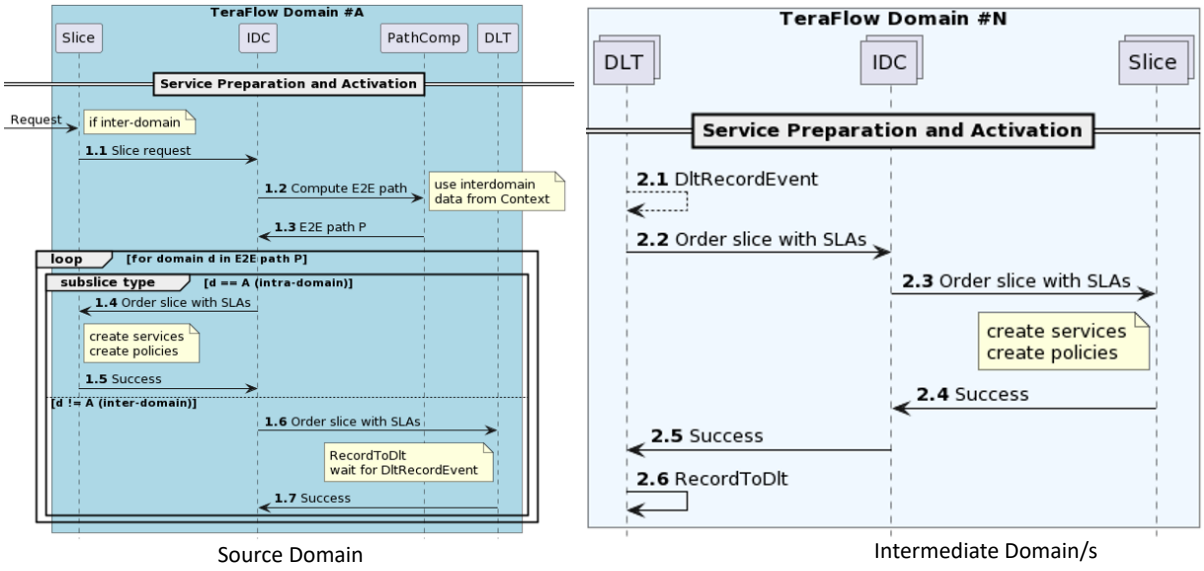

Figure 47. IDC Interactions: Activate Inter-Domain Slice with SLAs

Figure 47 shows the workflow at both the source and intermediate domains to activate a new (multi-domain) slice service. The Slice Component receives a request to create an inter-domain slice, e.g., from either a customer or an external system. This component sends an inter-domain slice creation request to the local Inter-Domain Component (step 1.1). The Inter-Domain Component relies on the PathComp Component to find an end-to-end inter-domain path fulfilling the requirements (i.e., bandwidth, latency, etc.) (step 1.2). The PathComp Component uses the diverse domain data collected by the Context Component to compose the global abstract topology. From this abstract multi-domain topology, the PathComp Component decides the domains traversed by the slice/service (step 1.3).

For each domain, the Inter-Domain Component inspects whether it is the local or a remote domain. For the local domain (steps 1.4 and 1.5), it configures the services through the Slice and Service Components that create the required SLA policies in the Policy Component (not depicted in the workflow) according to the requested slice. For the intermediate domains, the IDC stores a slice record carrying the required SLAs on the blockchain through the DLT Component (step 1.6) and waits for the receipt of an event from the remote domain informing it that the slice record was modified. Upon receiving this event, DLT Component retrieves that record and checks that the slice was properly activated (not shown in the figure), and returns control to the Inter-Domain Component. Note that requests to the other domains are issued in parallel. Thus, in the remote domain (N), when a slice record belonging to the domain (N) is created, and the event is detected (step 2.1), the DLT Component delegates the creation of the inter-domain sub-slice to the Inter-Domain Component (step 2.2). The IDC requests creation of a local slice through a slice creation request to the Slice Component (see steps 2.3 and 2.4) like that described for messages in both steps 1.4 and 1.5. When the slice is activated, the Inter-Domain Component replies to the DLT Component with the up-to-date slice record (step 2.5); the DLT Component stores it on the blockchain (step 2.6).

5.3.4.2 QoS-Enabled Inter-Domain Connectivity

In this section, we summarise our research contributions related to different aspects of QoS-enabled inter-domain connectivity which resulted in two publications [50][51]. The former, evaluates the traffic aggregation mechanisms for leveraging economies of scale while meeting heterogeneous QoS demands in the context of inter-domain connectivity. The latter tackles an analysis of trade-offs

between the degree of slice isolation and delay performance in environments with heterogeneous traffic profiles.

One key goal of the TeraFlow project consists of formalising and designing mechanisms for inter-domain interconnection possibilities between operators to support differentiation beyond pure endpoint reachability as provided by business VPNs. This includes enabling QoS profiles and/or guarantees with respect to aspects such as throughput and delay, while leveraging economies of scale by means of traffic aggregation to improve cost and resource efficiency.

The top part of Figure 48 displays an example multi-domain, multi-application setup that features spatially distributed application clients that interact with application servers located at service providers' premises. Depending on the chosen end-to-end (E2E) path, packets can traverse black-box transit domains as well as different administrative domains with potential (re-)aggregation of traffic flows at domain boundaries. Hence, when choosing an E2E path, operators not only have to consider the path itself, but also different mechanisms for aggregating flows that can have a significant impact on the QoS performance. Such aggregation options are highlighted in the bottom part of the figure and include treating all traffic in a best effort (BE) manner, using prioritisation (PRI), and network slicing (SLI) with either hard or soft isolation which can be emulated by means of a Hierarchical Token Bucket (HTB). Each of these options represents a trade-off regarding isolation, resource efficiency, deterministic QoS performance, costs, as well as (control plane) complexity and scalability.

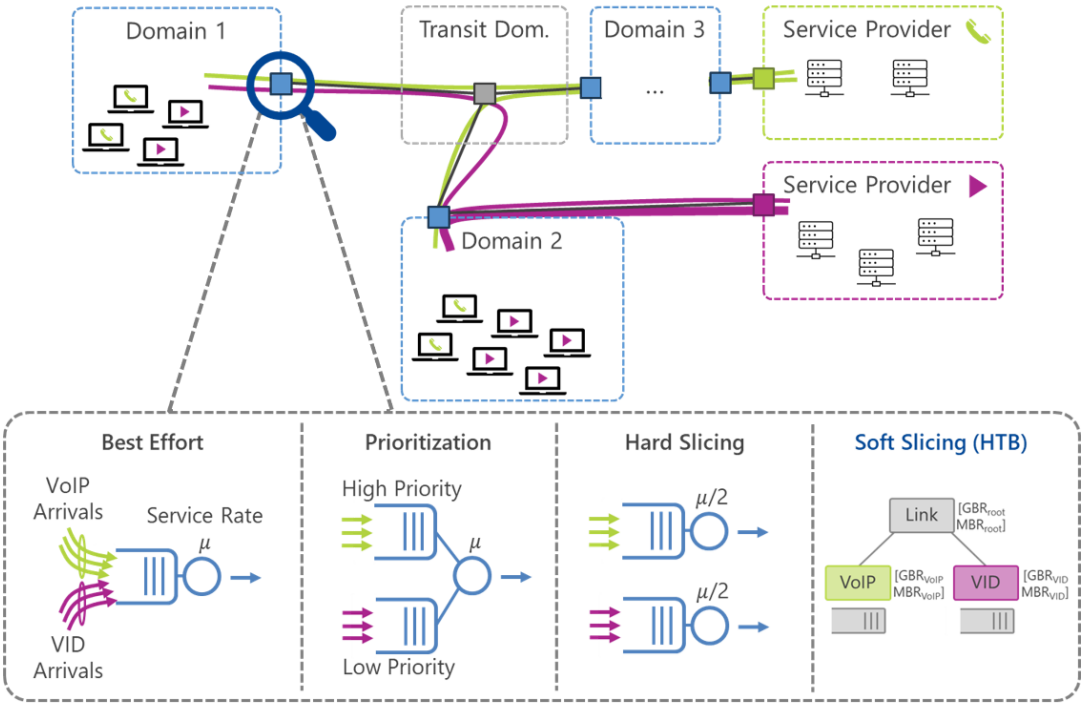

Figure 48. Overview of Inter-Domain Environment (top) and Traffic Handling Options (bottom).

In the subsections that follow, we first present results from queueing simulations to investigate the relationship between using different aggregation options, delay performance requirements, and the maximum tolerable link utilisation to support those requirements. Second, we use more detailed simulations to illustrate how parameters of soft slice isolation mechanisms can be tuned in order to benefit from multiplexing gains while meeting delay requirements at various levels of strictness.

5.3.4.2.1 Traffic Aggregation for QoS-Aware Inter-Domain Connectivity

Aggregation Mechanisms. For this part of the study, we consider the first three traffic aggregation mechanisms that are visualised in Figure 48. Given two arrival streams of packets from two traffic profiles, the aggregation mechanisms determine the allocation of the available service capacity as well as packets' trajectory through the system. In the following, we discuss how these mechanisms work and their operational implications.

The first and most simple approach is referred to as Best Effort (BE) and consists of putting all packet arrivals into a shared queue that is serviced at maximum rate in a FIFO manner. Since differentiated treatment per traffic class is not necessary when using this approach, it does not incur control plane overhead. Furthermore, it allows leveraging economies of scale since the service unit is active whenever a packet of any kind is in the system. However, the only way to improve the delay performance of the aggregate traffic stream consists of lowering the utilisation, either by reducing the number of admitted clients or overprovisioning.

In 5G and B5G networks, slicing plays a crucial role for multiplexing heterogeneous traffic. Hence, the second abstraction mimics traffic handling in the presence of slicing (SLI) with hard isolation. In this context, the available capacity is split between two systems with fully isolated queues and service units, each responsible for one of the two arrival streams. Such an approach would require corresponding hardware capabilities such as virtual routers and ports to be implemented on a shared physical infrastructure. Additionally, the granularity of the resource split, the number of virtual sub-systems, and the degree of isolation can vary between devices. As a consequence, the slicing approach involves control plane overhead and requires fine-tuning of (potentially device-specific) parameters. Although strict isolation with resource reservation does not offer multiplexing gains, elevated security and robustness requirements can be met which are crucial in contexts such as Public Protection and Disaster Relief (PPDR).

The final abstraction is based on prioritisation (PRI) with shared capacity, and represents a middle ground between the first two options. While it still requires some hardware capabilities and configuration effort, differentiated treatment is possible without losing economies of scale. However, it is worth noting that performance guarantees are less strict so that low-priority traffic might suffer unless more complex shaping is employed, e.g., by means of hierarchical QoS (hQos) using a Hierarchical Token Bucket (HTB). In our simulations, the first traffic class in a combination receives prioritised treatment whereas the second is treated with a low priority.

Traffic Profiles. In order to cover a wide range of applications, we start out by characterising three typical applications with respect to their traffic properties. In the process, we define archetypal traffic profiles whose parameters, such as rate or packet size, can be fine-tuned to fit and more closely represent other applications or variations of existing ones. For instance, parameters of a video streaming archetype could be adjusted to represent video streaming with content of different bitrates, while a voice-over-IP (VoIP) archetype could serve as starting point for real-time conversational video services. Since we are primarily interested in general relationships between traffic profiles and aggregation techniques, we omit in-depth application- and protocol-level behaviour such as dynamic bit rate adaptation or congestion control.

A total of three traffic profiles are considered: VoIP, file download (FDL), and video streaming (VID). They are listed alongside their main traffic characteristics in **Table *13***. Both VoIP and FDL exhibit constant bit rate (CBR) arrival patterns with packet interarrival times (IATs) of 20 and 6 ms using small and big packets respectively, and resulting rates of 30 kbps and 2 Mbps respectively. In contrast, the VID profile captures the typical behaviour of video streaming where a video file is split into segments

that are watched and downloaded as the session progresses. Assuming a segment duration of 2 sec and an average rate of 2 Mbps, this results in a bursty on/off pattern with a 4 Mbit burst every 2 sec with no activity in between.

Table 13. Traffic Profiles Under Study.

Traffic Profile	Packet Size	Arrival Pattern	Rate
VoIP	Small (75 B)	CBR (20 ms IAT)	30 kbps
File download (FDL)	Big (1,500 B)	CBR (6 ms IAT)	2 Mbps
Video stream (VID)	Big (1,500 B)	Bursty on / off	2 Mbps

We deliberately do not impose delay or other QoS requirements a priori, but leave them variable for the evaluation so that it is possible to check which constraints can be met with each aggregation mechanism.

Simulation Setup and Evaluation Metrics. Using our three traffic profiles, we run simulations with each possible two-profile combination in conjunction with each of the three aggregation mechanisms. We control the load that is offered to the system by varying the number of clients *n* we simulate for each profile. The traffic mix is dimensioned so that the total rate per profile is identical. Hence, FDL and VID are mixed with a 1:1 ratio, whereas the number of VoIP clients is multiplied by 66 to compensate for the rate difference. We use link capacities of 100 Mbps, 1 Gbps, and 10 Gbps, and by varying *n* appropriately, the offered load covers a range from 40 to 100% of the capacity in increments of 4%. Furthermore, we ensure independence between individual clients by randomly offsetting their starting time, i.e., the time of the first packet.

To obtain statistically significant results, each scenario is simulated ten times. The duration of individual simulation runs is chosen in a way that we record the statistics of at least two million packets per traffic profile per run during the steady phase of the simulation. We ignore events from the first 20 sec of each run to avoid transient behaviour during initialisation and compute the mean alongside 95% confidence intervals of relevant statistics across the ten repetitions. All simulation parameters are summarised in **Table *14***.

Table 14. Simulation Parameters.

Parameter	Values
Link capacity	100 Mbps, 1 Gbps, 10 Gbps
Aggregation mechanism	BE, SLI, PRI
Traffic combination	$\{VoIP, FDL, VID\}^2$
Offered load	$40{,}44, \ldots, 100\%$
Number of simulated packets	At least 2e6 per profile per run
Repetitions per scenario	10

While it is possible to extract a multitude of per-packet, per-client, and per-profile performance metrics, our main interest is on packets' *sojourn time* per traffic profile, i.e., the time difference between the moment a packet arrives at the system and the moment it departs after having been processed by the service unit. This allows us to derive insights regarding the trade-offs between different traffic aggregation mechanisms under different load conditions and traffic mixes.

Evaluation Aspects. In the following, we discuss numerical results of our queueing simulation. After analysing the sojourn times in the different simulation scenarios, we turn our focus to operational

aspects by studying the effects of traffic aggregation strategies, traffic composition, and delay requirements on the maximum tolerable load level.

Evaluation of Sojourn Time. **Figure *49*** displays the results for scenarios in which VoIP and VID traffic profiles are mixed. While the sub-plots correspond to different aggregation strategies and traffic profiles, the x-axis denotes the offered load, and the logarithmically scaled y-axis shows the mean sojourn time including 95% confidence intervals. Differently coloured curves represent different link capacities.

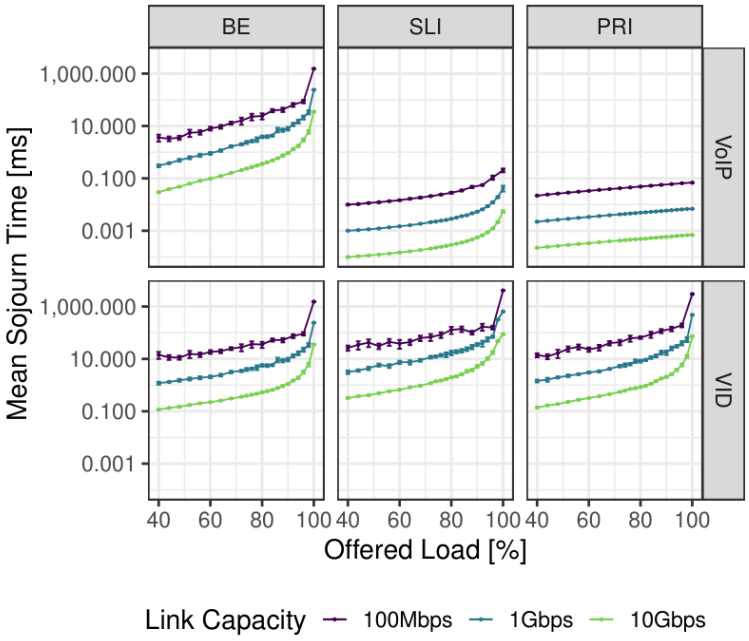

Figure 49. Impact of Offered Load on Mean Per-Application Sojourn Time When Mixing VoIP and VID Under Different Aggregation Mechanisms and Link Capacities.

A general observation can be made across all plots, namely that the link capacity is inversely proportional to the resulting mean sojourn time. This is in line with the fact that a higher link capacity leads to lower packet service times and therefore also to faster recovery times from burst arrivals.

In the case of BE, we can observe a sojourn time increase of up to three orders of magnitude for both traffic types when going from the lowest to the highest load levels. In contrast, PRI keeps the sojourn time increase of VoIP packets linear since they are prioritised and processed at full capacity while making up only half of the arriving load. With slicing-based aggregation, a sharper sojourn time increase can be observed towards the highest load levels. This can be explained by the fact that each slice operates at half the original link capacity. However, since VoIP clients exhibit a CBR traffic pattern, the increase is not as steep as for VID whose clients have bursty arrivals.

Furthermore, we can observe that while the delay performance of VID traffic does not change as much as that of VoIP, the resulting sojourn times decrease from BE to PRI to SLI. Although VID traffic is handled with low priority in the PRI case, the packets are still processed at full capacity whereas in the case of SLI, only half the link capacity is available, which especially affects bursty arrivals.

Evaluation of Tolerable Load under Delay Constraints. Since we are particularly interested in the effects of traffic aggregation strategies, traffic composition, and delay requirements on operational aspects such as the maximum tolerable load level, we perform the following preprocessing steps on the simulation outputs in order to obtain the visualisations in this subsection: given delay constraints

in the range 0,1, … ,100 ms for each application, we find the scenarios in which these delay constraints are met, and from those we extract the scenario with the highest number of clients, i.e., the highest offered load. As an example, we present results for scenarios with a link capacity of 1 Gbps and the traffic mix of VoIP+VID in **Figure *50***.

While the sub-plots in each graphic represent different aggregation strategies, the logarithmically scaled x- and y-axes denote delay constraints for the first and second traffic profile in the mix, respectively. Using the outlined preprocessing procedure, the colour of each tile corresponds to the maximum load that can be handled while meeting the respective delay constraints.

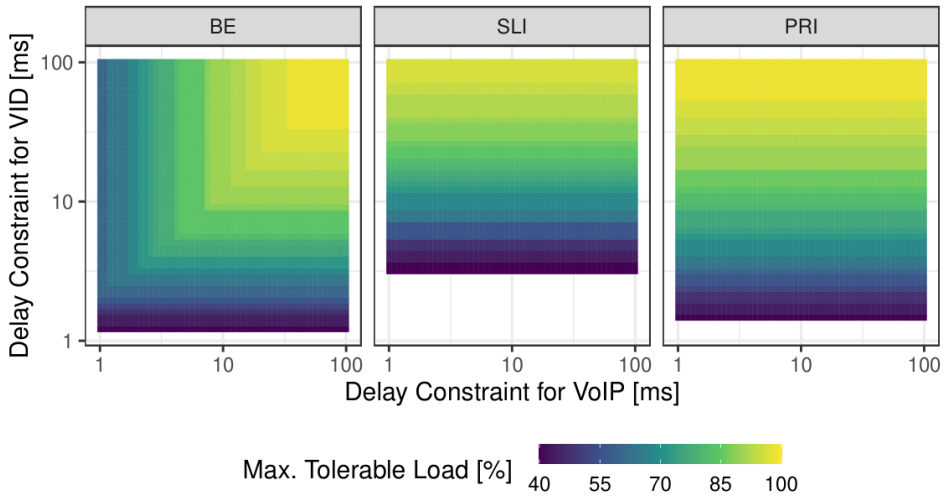

Figure 50. Impact of Delay Constraints and Aggregation Mechanisms on Maximum Tolerable Link Load when Superimposing VoIP and Video Streaming (VID) Traffic Profiles on a 1 Gbps Link.

In the case of BE-based aggregation, the maximum tolerable load level of 98% can only be reached if both traffic classes exhibit a delay tolerance of at least 35ms each. Additionally, the maximum tolerable load gradually declines when stricter delay constraints are introduced on any of the two traffic types. This is in line with the traditional strategy of overprovisioning resources to achieve targeted QoS constraints.

In contrast, when using priority-based aggregation (PRI), the 98% load level can be handled with significantly stricter delay constraints for the prioritised class, allowing 1ms delays. However, this comes at the price of higher delays for the second traffic type in the mix and therefore requires it to be able to sustain delays up to 56ms.

With slicing-based isolation (SLI), the lack of multiplexing gains manifests itself in a narrower range of delay constraint combinations that allow for its maximum load level of 96%. However, SLI provides isolation-related benefits with respect to security and robustness against interference between classes. It is also worth noting that splitting the available processing resources between slices results in longer service times, so that the lowest delay constraints cannot be met in case of the bursty VID traffic. This leads to the gap at the bottom of the corresponding sub-plot.

When considering the pairwise difference between cells of different aggregation scenarios, gains with respect to the maximum tolerable load for each combination of delay constraints can be derived. For instance, if constraints of 5 and 60ms were chosen for VoIP and VID, respectively, BE, SLI, and PRI would have maximum tolerable load levels of 84, 94, and 98%. Hence, using PRI over BE for this particular constellation would allow for a 14% higher link utilisation while meeting the delay performance requirements of both applications. Since such considerations are performed at each

aggregation step along the E2E path, they might be even stricter. We also note that while the absolute numbers in terms of tolerable load and delay constraints vary depending on the application mix and traffic parameters, the qualitative relationships between aggregation mechanisms are stable between scenarios.

In summary, our simulations allow us to quantify the impact of and trade-offs between various aggregation strategies on different heterogeneous traffic mixes and identify feasible regions for efficient operation. Future investigations will focus on how changing traffic profiles and profile parameters affect the gradients in the presented heatmaps, and whether there is a generalisable relationship between them.

5.3.4.2.2 Trade-offs Between the Degree of Slice Isolation and Delay Performance

Per-Hop Isolation and Delay. A key challenge for operators in the inter-domain setup outlined in Figure 48 is to reap multiplexing gains at each domain traversal, but without sacrificing the required QoS characteristics of affected application flows. However, before tackling the problem in an E2E fashion, a deep understanding of the effects and trade-offs between different degrees of isolation on a per-hop basis is required. Hence, we propose the single-hop setup displayed in the top part of Figure 51 which allows for a quantitative analysis of such trade-offs.

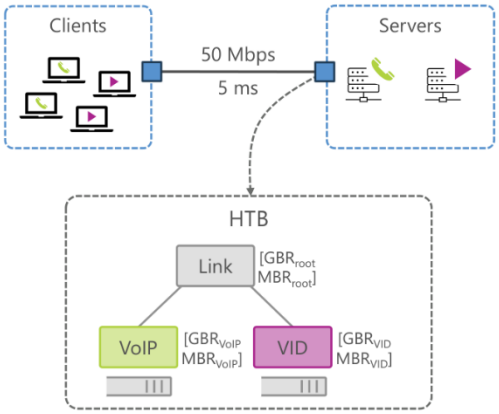

Figure 51. Top: Simplified Simulation Setup for Investigating the Relationship Between Isolation and Delay Performance in a Single-Hop Scenario. Bottom: HTB Tree and Parameters that Control Resource Sharing.

The setup consists of a single link that interconnects clients who use VoIP and VID applications with the corresponding application servers. In the following, we elaborate how HTB is used to mimic slicing with soft isolation and establish a relationship between HTB parameters and measures for quantifying the number of shared resources and therefore the degree of isolation. Furthermore, we provide details about the applications as well as the simulation environment and parameters.

Hierarchical Token Bucket (HTB). The central data structure for configuring hQoS in HTB is the HTB tree, an example of which is displayed in the bottom part of Figure 51. Starting at the root node which represents the entire link, each node in the tree has two parameters that control the Guaranteed Bitrate (GBR) and Maximum Bitrate (MBR) of the matching traffic. By arranging the nodes in this tree and setting appropriate GBR and MBR values, the degree of resource sharing (“borrowing”) and therefore isolation can be adjusted.

On arrival, packets are classified based on their header information and stored in the queues of the corresponding leaf nodes. Afterwards, packets are extracted from the queue according to a Deficient Round Robin (DRR) queuing discipline and sent out. In this work, we use our implementation of the HTBQueue [56] module for OMNeT++ / INET which follows the Linux implementation of HTB.

It is worth noting that the HTB offers additional parameters such as burst sizes and DRR quantum granularity for fine-tuning its behaviour. However, we limit our studies to the primary impact factors regarding the capacity split, and leave other parameters at their default values. Finally, although HTB allows deeper tree structures that could be used to provide even per-flow QoS, we argue that per-flow QoS would not be a scalable solution for E2E scenarios and hence focus on per-traffic-type resource allocation.

Resource Sharing Measures. Our main interest in this work lies in the relationship between the degree of isolation and various network KPIs. Hence, we introduce two measures that allow us to quantify different aspects of isolation or, equivalently, the degree of resource sharing. Figure 52 illustrates how the HTB parameters GBR and MBR relate to the amount of shared capacity and degree of contention in case of two competing applications, VoIP and VID, on a single link with a given capacity.

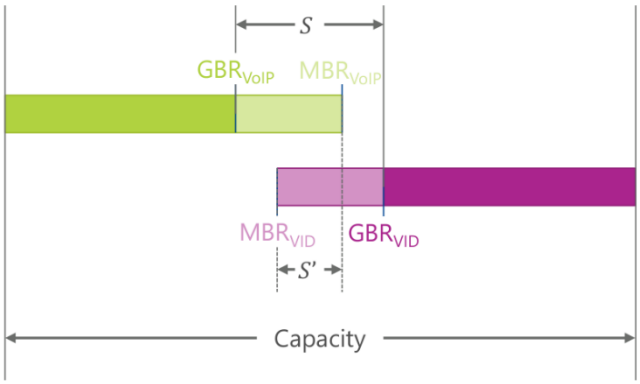

Figure 52. Bandwidth Sharing Relationships when Two Traffic Types are Configured, with Particular Focus on the Relationship Between GBR and MBR HTB Settings and Isolation/Contention Indicators S and S'.

Using the total available capacity c and the GBR/MBR parameters, we define two measures $S = 1 - \sum_i \frac{GBR_i}{c}$ and $S' = \sum_i \frac{MBR_i}{c} - 1$ which are also highlighted in the figure. While S depends on GBRs and denotes the amount of potentially contested link capacity, S' considers the MBRs and reflects the actual overlap between configured HTB capacities. Note that an overlap between the MBR of one traffic type with the GBR of the other is also possible.

Applications and Simulation Parameters. As indicated in Figure 51, we mix traffic that is generated by clients of two applications. These include a TCP-based adaptive video streaming application (VID) as well as a UDP-based VoIP application. For each application client, we can collect packet-level statistics such as E2E delays that are used to compute means, quantiles, and the variation of packet delays. Additionally, per-session aggregated metrics like the Mean Opinion Score (MOS) are obtained via well-established QoE models, namely the ITU-T P.1203 model in case of VID and the ITU-T G.107 e-model in case of VoIP.

In order to emulate emerging and more demanding applications such as the ones involving haptic feedback, we modify the VoIP application to increase its bandwidth usage and delay sensitivity by a factor of 10 each. To this end, we decrease the inter-packet delay to reach a bandwidth consumption of 300 kbps when a client is active, and feed the QoE model inflated delay values. Using the default parameters for the duration of talk spurts and silence periods (i.e., [56]), VoIP clients are active roughly 65% of the time and therefore consume an average of 195 kbps.

While each VoIP client generates UDP packets at a constant bitrate when active, adaptive video streaming clients are more dynamic and make decisions about the quality level of downloaded video segments based on their buffer fill level. Hence, their bandwidth is less predictable and can range from few hundred kbps to multiple Mbps, corresponding to resolutions between 480p and 1080p. As a reference, the average bitrate at 720p equals 1,600 kbps in our setup.

An overview of all simulation parameters is provided in Table 15. Given a fixed link capacity and baseline delay, we vary the load that is offered to the system by gradually increasing the number of clients per application. To investigate the effects of isolation on the delay performance and general system characteristics like the overall link capacity usage, we first determine a split between the two application types based on their expected bandwidth consumption. Given the previously mentioned mean bitrates, $\frac{195}{195+1600} \simeq 11\%$ of the link capacity (5.5 Mbps) is used as a starting point for VoIP traffic and the remaining 44.5 Mbps for VID. From these reference points, both GBR and MBR values are varied to cover all combinations that are listed in the table, representing a wide range of isolation and resource sharing conditions. We perform five repetitions per parameter combination to obtain statistically significant results.

Table 15. Simulation Parameters.

Parameter	Values
Link capacity	50 Mbps
Link delay	5ms
Simulation time	400s
Video duration	Uniform (280, 320) sec
Number of clients per application	$12,16,20,24$
(p, q)	$\{0, 1, 2, \dots, 10\}^2$
GBR for VoIP traffic	$(5.5 - 0.5 \cdot q)$ Mbps
MBR for VoIP traffic	$(5.5 + 0.5 \cdot p)$ Mbps
GBR for video traffic	$(44.5 - 0.5 \cdot q)$Mbps
MBR for video traffic	$(44.5 + 0.5 \cdot p)$ Mbps
Number of repetitions per configuration	5
Total number of simulation runs	2,420

Evaluation Aspects. In the context of the simulation setup, we are mainly interested in evaluating how and to what extent different levels of isolation that are controlled by the GBR and MBR parameters impact different on delay, QoE, and system-level KPIs. To this end, we first identify key factors that affect the respective KPIs by means of a correlation- and ANOVA-based analysis of main effects. Then, we study the actual KPI values in-depth in order to quantify their sensitivity to our isolation and resource sharing measures.

Analysis of Main Effects. Table 16 displays the results of a correlation-based analysis and an Analysis of Variance (ANOVA-based) of isolation-related HTB parameters which directly impact the amount of remaining and shared capacity between traffic types. The effect of each of the two parameters on a total of six KPIs is studied. These are: the mean; five-nines; the maximum delay of the VoIP traffic; the variance of its delay as a measure of jitter; the QoE-related average MOS value among VoIP clients; and the mean link capacity usage as an indicator of resource efficiency. In addition to calculating Spearman's correlation coefficient r_s between each pair of metric and parameter, we additionally perform an ANOVA and report the resulting values of the *F* statistic alongside the corresponding *p*-values.

Table 16. Correlation- and ANOVA-based Analysis of Isolation-Related Parameters and their Effect on Delay Performance and Link Capacity Usage.

Parameter	GBR_{VoIP}			MBR_{VoIP}		
Metric	r_s	*F*	*p*	r_s	*F*	*p*
Mean delay of VoIP traffic	-0.48	367.84	**< 0.001**	0.08	12.86	**< 0.001**
Five-nines delay of VoIP traffic	-0.43	316.20	**< 0.001**	-0.07	22.71	**< 0.001**
Maximum delay of VoIP traffic	-0.43	320.79	**< 0.001**	-0.07	22.38	**< 0.001**
Variance of delay of VoIP traffic	-0.43	257.54	**< 0.001**	-0.01	36.87	**< 0.001**
MOS of VoIP clients	0.42	350.94	**< 0.001**	-0.04	11.34	**< 0.001**
Mean link capacity usage	0.00	0.08	0.78	0.20	73.01	**< 0.001**

While both GBR and MBR values have a statistically significant impact on all delay and QoE-related metrics, the correlations and *F* values suggest that the GBR is the main influencing factor on these metrics. This is in line with the fact that a higher GBR corresponds to a higher degree of isolation and therefore minimises the interference that can occur. In the case of mean link capacity usage, the MBR value is identified to have the larger impact. Again, this matches the expectation that a higher degree of resource sharing leads to a more efficient use of available resources, i.e., temporarily idle resources of one application can be utilised by another.

Impact of Isolation on Determinism. Based on the results of the main effects study, we perform an in-depth analysis of the relationship between the resource sharing measures *S* and *S'* and the individual KPIs at different load levels.

The first row of Figure 53 displays the results for the three delay metrics, i.e., mean, five-nines, and maximum delay experienced by the VoIP clients. Given the fraction of shared link capacity *S* on the x-axis, the y-axis shows the corresponding metric, while differently coloured curves denote different load levels in terms of the number of clients per application. Furthermore, error bars display 95% confidence intervals. As suggested by the results of the previous analysis, a positive correlation between *S* and the delay metrics can be observed. Additionally, the strictness of the delay requirements dictates the sensitivity, e.g., the five-nines delay deteriorates significantly earlier than the mean delay. For instance, with 12 clients the mean delay of VoIP clients remains almost unchanged even up to *S=0.2*, whereas a sharp increase starts at *S=0.16* in case of the five-nines delay.

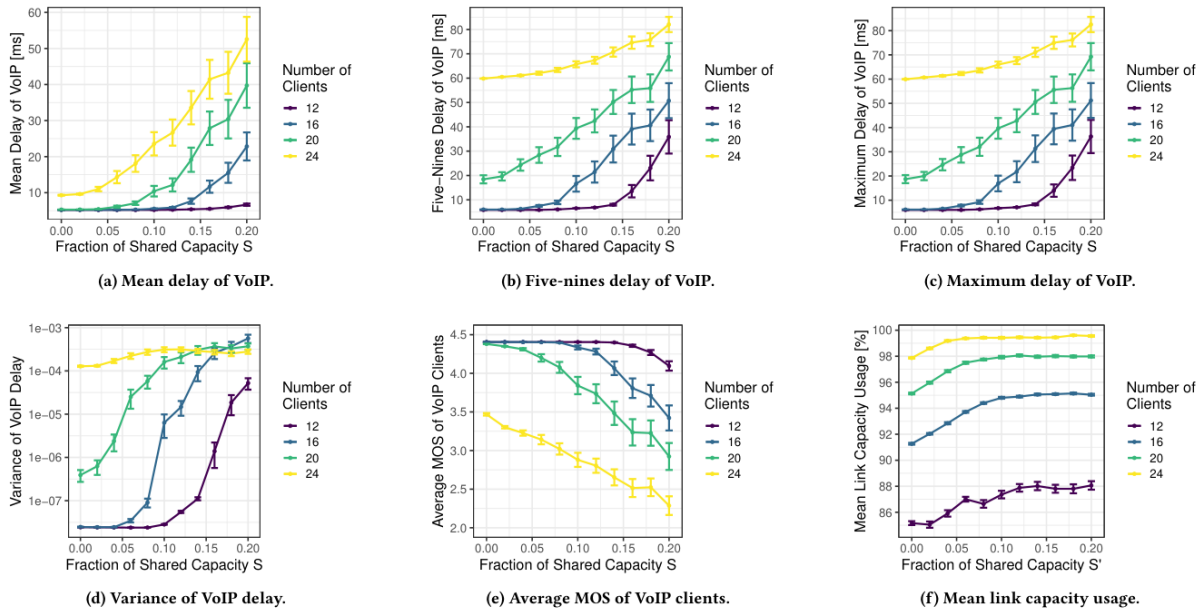

Figure 53. Impact of Isolation-Related Measures and Number of Clients on Delay and QoE Performance of Delay-Sensitive Traffic and Link Capacity Usage.

We can observe in Subfigure (d) of Figure 51 that, for the same number of clients, the variance is the delay metric that deteriorates earliest – around the *S=0.12 to 0.14* mark – and is therefore most sensitive to interference. While the KPIs discussed so far are related to pure QoS, Subfigure (e) of Figure 51 also provides a QoE perspective. Since the default QoE model is based on measured delays per talk spurt, the previously observed delay increases also lead to a degradation of MOS values.

Finally, the mean link utilisation has been identified to be mostly affected by the maximum bitrate and hence its relation to the *S'* indicator is shown in Subfigure (f). The results illustrate that depending on the baseline load level, an increase of *S'* from *0* to *0.2* leads to a steady increase of the mean utilisation. Differences between the two extremes range from 2% utilisation at the highest load level to roughly 4% with 16 clients per application. Such medium-load scenarios offer the most potential for multiplexing gains since high loads already have a high utilisation to begin with and clients tend to have enough resources without sharing at low loads.

In summary, our simulations not only allow determination of the qualitative effect of isolation-related parameters on QoS, QoE, and resource usage characteristics, but also enable us to quantitatively assess the relationships. Using such an approach in conjunction with application-specific requirements can help finding system configurations that hit the sweet spot between resource utilisation and application performance.

6 Conclusions and Next Steps

This deliverable (D4.2) concludes the evaluation of the TeraFlow security and B5G network integration. This deliverable and its accompanying Milestone (MS4.2) reflect the work done in the second phase of WP4. It describes the new features and extensions of security components of the TFS and B5G network integration providing an evaluation of each of them. In the following paragraphs we summarise some conclusions and next steps of each individual task in WP4.

As for the cybersecurity components in Task 4.1 significant efforts have been devoted to their integration in two realistic scenarios that show how to detect and mitigate cyberthreats at the optical and packet layer. Furthermore, three research works have been conducted in the CAD. First, we developed a methodological framework for transforming neural network-based detectors into more efficient models with respect to the energy they consume. Second, we designed a new method for generating high-quality adversarial examples to transform machine learning-based detectors into equivalent models that are resilient to such adversarial attacks. In addition, we developed a GAN architecture to generate synthetic data that can fully substitute real data in the training of machine-learning based detectors and therefore avoiding the generation of privacy breaches when real data is used. We also evolved the overall components by specialising the Attack Inference Component, allowing for more granular scalability features, the use of different ML models and techniques, as well as seamless updates of models. Future work will explore the generalisation of the energy efficiency framework to Recurrent and Convolutional Neural Networks and the generation of time-series of synthetic data, and mitigation strategies for attacks in the optical layer.

For the DLT Component in Task 4.2, the DLT module described is the foundation to work on core blockchain technologies and improve the scalability, privacy, and governance. In particular, we enhanced the scalability of the underlying consensus algorithms using advanced architectures that distribute the transaction capacity without losing the core blockchain properties. In the current deliverable we designed and implemented a smart contracts for sharing, retrieving updates on the infrastructure, as well as automatically notifying clients when new updates are included in the blockchain. Our smart contract aims at reducing the attack surface of TFS and its components by providing a platform to share all the different components of the infrastructure, effectively enabling any participant to do a complete check of the deployed components and ensure only trusted devices are included in the network topology. Future work may include using the DLT Component to additionally share resource utilisation in real time and enforce resource allocation and analysis of real time weaknesses of the network applications.

Section 5 reported the different activities (carried out in T4.3) to support the interworking of the TFS Controller for accommodating B5G services. This comprises three key components: NBI, WebUI, and Inter-Domain. For the former, the different interfaces are described that enable interaction with external systems including network service creation requested by an NFV Orchestrator (i.e., ETSI OSM). This entails that the NBI not only processes the incoming NFV-O connectivity requests (e.g., to interconnect remote cloud sites), but also manages the interactions with other components (i.e., Service, Context, and PathComp) to handle the lifecycle management of the resulting network services. In this regard, the support of managing IETF L2VPN services is presented. In addition, the NBI has also been enhanced to cover the processing to roll out network slices with stringent SLA needs. Last but not least, the specific objective of energy-efficiency is addressed through presenting a devised algorithm aiming at reducing the network power consumption when deploying network services (e.g., between cloud sites). The next steps for the NBI will focus on increasing the supported functionalities

and services that can be requested through this component, along with exhaustively conducting performance evaluation of selected algorithms such as for energy-efficient routing.

The overall architecture of the WebUI is presented and detailed. As stated, the idea of this component is to increase the interactions of external user/applications, via HTTP and the defined REST API, with the TFS Controller. In this regard, the basic operations that are already supported (i.e., retrieving context information, triggering slice/service, and monitoring the status of selected artifacts/resources of the infrastructure) will be investigated with a view to being enhanced. The goal is to expose further and more advanced functions to the external user/systems, such as increasing the granularity of monitored information (e.g., link occupancy, device ports, etc.).

Finally, the functional interactions carried out by the Inter-Domain Components hosted in peer TFS Controllers are exhaustively presented for: i) retrieving abstracted domain information; and ii) deploying multi-domain slice/connectivity services. To this end, the adopted solution leveraged blockchain technology using the DLT Component. The overall interactions in terms of workflows are detailed, and have been successfully showcased in specific conferences. Moreover, in the context of the evaluation of the Inter-Domain Component, the contribution bound to QoS-enabled inter-domain connectivity disseminated has been reported in two publications: i) traffic aggregation mechanisms; and ii) analysis of the trade-offs between slice isolation and delay performance. A planned next step for the Inter-Domain Component is to deploy TFS Controller instances for different domains, and assess the SLA-violation of multi-domain slices/network services.

7 References

[1] Karamchandani, A., Mozo, A., Gómez-Canaval, S., Pastor, A., 'A Methodological Framework for Optimizing the Energy Consumption of Deep Neural Networks: A Case Study of a Cyber Threat Detector', submitted to Neural Computing and Applications, Dec. 2022.

[2] D. Patterson et al., 'Carbon emissions and large neural network training', arXiv:2104.10350 [cs], Apr. 2021.

[3] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, 'Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks', IEEE Journal of Solid-State Circuits, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[4] J. Qiu et al., 'Going deeper with embedded FPGA platform for convolutional neural network', May 2016.

[5] A. Gordon et al., 'MorphNet: Fast & simple resource-constrained structure learning of deep networks', arXiv:1711.06798 [cs, stat], Apr. 2018.

[6] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, 'Learning transferable architectures for scalable image recognition', arXiv:1707.07012 [cs, stat], Apr. 2018.

[7] H. Liu, K. Simonyan, and Y. Yang, 'DARTS: Differentiable architecture search', arXiv:1806.09055 [cs, stat], Apr. 2019.

[8] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, 'Regularized evolution for image classifier architecture search', arXiv:1802.01548 [cs], Feb. 2019.

[9] H. Jin, Q. Song, and X. Hu, 'Auto-keras: An efficient neural architecture search system', arXiv:1806.10282 [cs, stat], Mar. 2019.

[10] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, 'Efficient neural architecture search via parameter sharing', arXiv:1802.03268 [cs, stat], Feb. 2018.

[11] H. Cai, L. Zhu, and S. Han, 'ProxylessNAS: Direct neural architecture search on target task and hardware', arXiv:1812.00332 [cs, stat], Feb. 2019.

[12] B. D. Rouhani, A. Mirhoseini, and F. Koushanfar, 'DeLight: Adding energy dimension to deep neural networks', in Proceedings of the 2016 international symposium on low power electronics and design, San Francisco Airport CA USA, Aug. 2016, pp. 112–117.

[13] B. D. Rouhani, A. Mirhoseini, and F. Koushanfar, 'Deep3: Leveraging three levels of parallelism for efficient deep learning', in Proceedings of the 54th annual design automation conference 2017, New York, NY, USA, Jun. 2017, pp. 1–6. doi: 10.1145/3061639.3062225.

[14] Y. Guo, 'A survey on methods and theories of quantized neural networks', arXiv:1808.04752 [cs, stat], Dec. 2018.

[15] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, 'DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients', arXiv:1606.06160 [cs], Feb. 2018.

[16] J. Gou, B. Yu, S. J. Maybank, and D. Tao, 'Knowledge distillation: A survey', International Journal of Computer Vision, vol. 129, no. 6, pp. 1789–1819, Jun. 2021.

[17] A. Pastor et al., 'Detection of encrypted cryptomining malware connections with machine and deep learning', IEEE Access, vol. 8, pp. 158036–158055, 2020.

[18] A. Pastor, A. Mozo, D. R. Lopez, J. Folgueira, and A. Kapodistria, 'The Mouseworld, a security traffic analysis lab based on NFV/SDN', in Proceedings of the 13th international conference on availability, reliability and security, 2018, pp. 1–6.

[19] E. García-Martín, N. Lavesson, H. Grahn, E. Casalicchio, and V. Boeva, 'How to measure energy consumption in machine learning algorithms', in ECML PKDD 2018 workshops, Cham, 2019, pp. 243–255.

[20] E. García-Martín, C. F. Rodrigues, G. Riley, and H. Grahn, 'Estimation of energy consumption in machine learning', Journal of Parallel and Distributed Computing, vol. 134, pp. 75–88, Dec. 2019.

[21] Mozo, A., Karamchandani, A., Pastor, A., Gómez-Canaval, S., 'Crafting Black-Box Adversarials to Attack Machine Learning-Based Systems with GANs', submitted to Applied Intelligence, Dec. 2022.

[22] González-Prieto, Á., Mozo, A., Gómez-Canaval, S., & Talavera, E., 'Improving the quality of generative models through Smirnov transformation', Information Sciences, 609, 1539-1566, Sep. 2022.

[23] Alzantot, M., Sharma, Y., Chakraborty, S., Zhang, H., Hsieh, C.J., Srivastava, M., 2019. GenAttack: Practical Black-box Attacks with Gradient Free Optimization. arXiv:1805.11090.

[24] Chen, P.Y., Zhang, H., Sharma, Y., Yi, J., Hsieh, C.J., 2017. ZOO: Zeroth Order Optimization based Black-box Attacks to Deep Neural Networks without Training Substitute Models, in: Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, pp. 15–26.

[25] Carlini, N., Wagner, D., 2016. Towards Evaluating the Robustness of Neural Networks arXiv:1608.04644.

[26] Hu, W., Tan, Y., 2017. Generating adversarial malware examples for black-box attacks based on GAN. arXiv:1702.05983.

[27] Xiao, C., Li, B., Zhu, J.Y., He, W., Liu, M., Song, D., 2018a. Generating Adversarial Examples with Adversarial Networks, arXiv: 1801.02610.

[28] Mangla, P., Jandial, S., Varshney, S., Balasubramanian, V.N., 2019. AdvGAN++ : Harnessing latent layers for adversary generation, arXiv: 1908.00706.

[29] Mozo, A., González-Prieto, Á., Pastor, A., Gómez-Canaval, S., & Talavera, E., 'Synthetic flow-based cryptomining attack generation through Generative Adversarial Networks', Scientific reports, 12(1), 1-27, Feb. 2022.

[30] Mozo, A., Karamchandani, A., Gómez-Canaval, S., Sanz, M., Moreno, J. I., & Pastor, A., 'B5GEMINI: AI-Driven Network Digital Twin', *Sensors*, *22*(11), 4106, May. 2022.

[31] González-Prieto, Á., Mozo, A., Gómez-Canaval, S., & Talavera, E. (2022). Improving the quality of generative models through Smirnov transformation. Information Sciences, 609, 1539-1566.

[32] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein gan. arXiv preprint arXiv:1701.07875, 2017.

[33] M. Furdek et al., "Machine learning for optical network security monitoring: A practical perspective", J. Lightw. Technol., vol. 38, no. 11, pp. 2860–2871, 2020. DOI: 10.1109/JLT.2020.2987032.

[34] S.S. Sachin et al., "Blockchain for Distributed Systems Security", Wiley-IEEE Computer Society, 2019.

[35] S. Boukria et al., "BCFR: Blockchain-based Controller Against False Flow Rule Injection in SDN," 2019 IEEE ISCC, 2019.

[36] S. Misra et al., "Blockchain-Based Controller Recovery in SDN," IEEE INFOCOM, 2020.

[37] A. Sforzin, M. Maso, C. Soriente, G. Karame, "*On the Storage Overhead of Proof-of-Work Blockchains*", IEEE Blockchain, 2022.

[38] G. Marson, S. Andreina, L. Alluminio, K. Munichev, G. Karame, "*Mitosis: Practically Scaling Permissioned Blockchains*", ACSAC, 2021

[39] J-R. Giesen, S. Andreina, et al. "*Practical Mitigation of Smart Contract Bugs*", Arxiv, 2022

[40] TeraFlow Deliverable D4.1: Preliminary Evaluation of TeraFlow Security and B5G Network Integration, Dec. 2021

[41]B. Wen, G. Fioccola, C. Xie, and L. Jalil, "A YANG Data model for Layer 2 Virtual Private Network L2VPN Service Delivery", IETF RFC 8466, Oct. 2018.

[42] Ll. Gifre, at. al., "Experimental Demonstration of Transport Network Slicing with SLA Using the TeraFlowSDN controller", in proc. of European Conference on Optical Communications (ECOC), Sept. 2022.

[43] M. Boucadair, O. González de Dios, S. Barguil, L. Munoz, "A YANG Network Data Model for Layer 2 VPNs", IETF RFC 9291, Sept. 2022.

[44] TeraFlow Deliverable D32, "Final evaluation of Life-cycle automation and high performance SDN components", Dec. 2022.

[45] Telecoms to Triple Electricity Consumption, Boosting Growth of Distributed Energy Generation, https://www.environmentalleader.com/2021/02/telecoms-to-triple-electricity-consumption-boosting-growth-of-distributed-energy-generation/.

[46] B. G. Assefa, et. al., "RESDN: A Novel Metric and Method for Energy Efficient Routing in Software Defined Networks", IEEE Transaction in Network and Service Management, 2019.

[47] A. Vishwanath, et. al., "Modeling Energy Consumption in High-Capacity Routers and Switches", IEEE Journal in Selected Areas of Communications, 2014.

[48] P. T. Congdon, et. al., "Simultaneously reducing latency and power consumption in openflow switches", IEEE/ACM Trans on Networking, 2013.

[49] S. S. Tadesse, et. al., "Energy-efficient traffic allocation in SDN-based backhaul networks: theory and implementation", IEEE Annual Consumer Communications & Networking Conf (CCNC), 2017.

[50] S. Lange, J. F. Pajo, T. Zinner, H. Lønsethagen, and M. Xie, "QoS-aware Inter-Domain Connectivity: Control Plane Design and Operational Considerations", in IEEE/IFIP Network Operations and Management Symposium (NOMS) - FlexNGIA Workshop, 2022.

[51] S. Lange, M. Gajić, T. Zinner, J. F. Pajo, H. Lønsethagen, M. Xie, and R. Vilalta, "Towards Assessing Effects of Isolation on Determinism in Multi-Application Scenarios", in ACM SIGCOMM Workshop on Future of Internet Routing & Addressing, 2022.

[52] LL. Gifre, R. Vilalta, S. Andreina, M. Xie, J. F. Pajo, K. Munichev, J. Lønsethagen, S. Lange, T. Zinner, G. Marson, P. Alemany, M. Gajić, R. Casellas, R. Martínez, R. Muñoz, "DLT-based End-to-End Inter-Domain Transport Network Slice with SLA Management Using Cloud-based SDN controllers", in Proc. of NFV-SDN, Nov. 2022.

[53] R. Vilalta, et. al., "End-to-end interdomain transport network slice management using cloud-based controllers", in Proc. of OECC/PSC, 2022.

[54] http://creativecommons.org/licenses/by-nc-nd/3.0/deed.en_US

[55] https://assets.ey.com/content/dam/ey-sites/ey-com/en_gl/topics/blockchain/ey-public-blockchain-opportunity-snapshot.pdf

[56] Bosk, Marcin, et al. "HTBQueue: A Hierarchical Token Bucket Implementation for the OMNeT++/INET Framework." arXiv preprint arXiv:2109.12879 (2021). https://doc.omnetpp.org/inet/api-current/neddoc/inet.applications.voip.SimpleVoipSender.html

[57] TeraFlow Project Deliverable D5.2 "Implementation of pilots and first evaluation"

