applied sciences

MDPI

*Article*

# A Machine-Learning-Based Cyberattack Detector for a Cloud-Based SDN Controller

**Alberto Mozo [1,*], Amit Karamchandani [1], Luis de la Cal [1], Sandra Gómez-Canaval [1], Antonio Pastor [2] and Lluis Gifre [3]**

[1] ETSI Sistemas Informáticos, Departamento Sistemas Informáticos, Universidad Politécnica de Madrid, 28031 Madrid, Spain
[2] Telefónica I+D, 28050 Madrid, Spain
[3] Centre Tecnològic de Telecomunicacions de Catalunya (CTTC/CERCA), 08860 Castelldefels, Spain
* Correspondence: a.mozo@upm.es

**Abstract:** The rapid evolution of network infrastructure through the softwarization of network elements has led to an exponential increase in the attack surface, thereby increasing the complexity of threat protection. In light of this pressing concern, European Telecommunications Standards Institute (ETSI) TeraFlowSDN (TFS), an open-source microservice-based cloud-native Software-Defined Networking (SDN) controller, integrates robust Machine-Learning components to safeguard its network and infrastructure against potential malicious actors. This work presents a comprehensive study of the integration of these Machine-Learning components in a distributed scenario to provide secure end-to-end protection against cyber threats occurring at the packet level of the telecom operator's Virtual Private Network (VPN) services configured with that feature. To illustrate the effectiveness of this integration, a real-world emerging attack vector (the cryptomining malware attack) is used as a demonstration. Furthermore, to address the pressing challenge of energy consumption in the telecom industry, we harness the full potential of state-of-the-art Green Artificial Intelligence techniques to optimize the size and complexity of Machine-Learning models in order to reduce their energy usage while maintaining their ability to accurately detect potential cyber threats. Additionally, to enhance the integrity and security of TeraFlowSDN's cybersecurity components, Machine-Learning models are safeguarded from sophisticated adversarial attacks that attempt to deceive them by subtly perturbing input data. To accomplish this goal, Machine-Learning models are retrained with high-quality adversarial examples generated using a Generative Adversarial Network.

**Keywords:** software-defined networking; machine learning; energy efficiency; green AI; adversarial attack; cryptomining attack; cybersecurity



**Citation:** Mozo, A.; Karamchandani, A.; de la Cal, L.; Gómez-Canaval, S.; Pastor, A.; Gifre, L. A Machine-Learning-Based Cyberattack Detector for a Cloud-Based SDN Controller. *Appl. Sci.* **2023**, *13*, 4914. https://doi.org/10.3390/app13084914

Academic Editor: Christos Bouras

Received: 10 February 2023
Revised: 29 March 2023
Accepted: 5 April 2023
Published: 13 April 2023



## 1. Introduction

As technology advances, the importance of security in network operations increases as software components take on a larger role and human intervention becomes less necessary. As a result, a top priority for next-generation SDN controllers is to ensure a secure environment [1]. To achieve this goal, the TeraFlow project [2] developed a novel SDN controller (TeraFlowSDN) for the 5G and beyond network era that integrates technologies, such as today's Network Function Virtualization (NFV) and supports new capabilities for flow management and device integration. Importantly, TeraFlowSDN was established by the European Telecommunications Standards Institute (ETSI) as a reference implementation for SDN controllers, and the Open Source Group TeraFlowSDN (OSG TFS) Working Group was established within ETSI specifically to focus on the development and advancement of this open-source controller [3].

The evolution of security threats, many of which are due to technological advances, creates new attack vectors (as in the case of NFV [4]) that require the protection of both network services and the network controller. To this effect, multiple Network Intrusion

Detection Systems have been proposed in previous works. However, most of them rely on outdated methods, such as Snort [5,6], that are unable to handle the encrypted nature of modern traffic [7]. It is also common for previous studies to rely on the NSL-KDD Dataset as the training data for the systems [8].

This dataset has been heavily criticized and shown to not be a realistic reflection of modern traffic [9], which makes many of the aforementioned studies not directly applicable to real-life scenarios. Furthermore, when studies do take into account novel methods for attack detection and mitigation, such as Machine Learning [10], they fail to consider both the intrinsic vulnerabilities of these methods and the energy needs for such systems. On the one hand, it has been shown that ML models are susceptible to adversarial attacks, which can compromise the system by slightly altering the input and, thus, producing false negatives [11,12].

On the other hand, as telecommunication traffic scales up, it becomes critical to implement optimization strategies aimed at reducing energy consumption, which poses a significant challenge in this industry. In this regard, this paper presents work performed using the TeraFlowSDN controller to address some of these needs from several perspectives, incorporating a distributed and scalable cybersecurity solution to the TeraFlowSDN architecture that (i) relies on standard interfaces to facilitate future extensions and modifications, (ii) defends not only the network but itself against attacks and (iii) is optimized to reduce its energy consumption.

To detect malicious flows in the data and control planes, our work proposes to integrate, in TeraFlowSDN, a distributed Intrusion Detection System (IDS) based on Machine-Learning (ML) and Deep-Learning (DL) components placed at the network edge and in the SDN controller. A distributed IDS is expected to improve the scalability and decrease the bandwidth and response time for detecting malicious flows while reporting back to the controller for an assessment of network security. Many IDSs have been proposed to identify different types of attacks [13], but proposals that address scalability, bandwidth and latency issues are lacking.

In this context, our work was conceived in three complementary dimensions: (i) a distributed ML-based IDS solution (Cybersecurity NetApp) that is integrated within the TeraFlowSDN architecture, (ii) a Green AI solution to reduce the energy consumption of the ML cybersecurity components of the solution and (iii) a fortification mechanism of the ML components to defend themselves against adversarial examples, a new type of attack that attempts to fool ML models introducing small perturbations in their inputs. To exemplify these goals, a recently appeared cryptomining attack is proposed as a showcase scenario.

The Cybersecurity NetApp is designed to address the challenge of network threat capture, identification and mitigation. To achieve this task, the Cybersecurity NetApp includes two centralized components in the TeraFlowSDN controller, the Centralized Attack Detector (CAD) and the Attack Mitigator (AM), as well as a distributed component, the Distributed Attack Detector (DAD), which is placed at a remote site.

The DAD component receives network traffic at the network edge and generates statistical summaries of the network flows by aggregating packets from the same connection at regular intervals of time. The DAD component then transmits these statistics to the CAD component, which uses them as input to an ML-based network traffic classifier running within it. After a network attack has been detected in the ML classifier, the confidence of the prediction as well as the flow connection identifier are sent to the AM component. The AM component then communicates with several core TeraflowSDN components to enforce and apply the corresponding mitigation strategy in the network elements. For example, based on a predefined policy, it can be decided that flows with the same connection identifier as the attack are blocked on the access router.

It is worth noting that the proposed design is distributed, modular and heavily relies on de facto industry standards based on open-source high-performance Remote Procedure Call (RPC) frameworks, such as gRPC and Protocol Buffer, making it highly adaptable

to integrate the detection and mitigation of other attack scenarios into the TeraFlowSDN Cybersecurity solution.

To address the pressing challenge of energy consumption in the telecommunications industry, this study proposes a solution that, taking advantage of the latest Green AI techniques [14], optimizes the energy consumption of the ML models deployed on the TeraFlowSDN controller, ensuring significant energy savings without compromising performance. To this end, a set of optimization experiments were designed to reduce the energy consumption of the DNN model that is deployed in the CAD component.

First, a deep exploration of the state of the art of energy efficiency was conducted, identifying a collection of different techniques that were combined into 11 optimization strategies. An experimental evaluation was performed on all the different optimization strategies using the original DNN model designed and trained for the CAD. The resultant models that achieved a significant reduction in energy consumption while maintaining a good level of performance in attack detection were selected.

To obtain a reliable metric regarding the power consumption during the training, model optimization, inference and model-loading phases, the experiments were performed using the Running Average Power Limit (RAPL) interface. It is worth noting that our study focused on the inference phase, since it is the most energy-consuming in ML applications that are going to be deployed in a real-time environment, such as the TeraFlowSDN controller. At the end of the experiments, the total average energy consumption was reduced by up to 83.30% with a minimal performance degradation of only 0.08% in the Balanced Accuracy score.

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 by adding carefully crafted perturbations to them, which are often imperceptible to humans.

The malicious samples obtained, more often referred to as Adversarial Examples (AEs), can be used to cause an ML 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. To minimize the impact of this type of attack in the ML models deployed in the TeraFlowSDN controller, this work proposes a strategy to fortify ML models against AEs that consists of crafting high-quality AEs that will be used later in the retraining of an ML model to fortify it against AEs.

To produce AEs, the experiments rely on a realistic "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 used for model learning. A very promising approach to elaborate AEs in a black-box environment is the use of Generative Adversarial Networks (GANs). GANs have been shown to be capable of generating data samples that are indistinguishable from real data samples when used for training ML models [15].

In addition, GAN-based attacks (e.g., MalGAN [16] and AdvGAN [17]) have been successfully applied to a variety of ML models. In order to generate high-quality AEs, this work proposes, as a novelty, to significantly enhance the performance of the MalGAN architecture using an activation function based on the Smirnov transformation that is added in the output of the generator [18]. In sharp contrast with other solutions, our enhanced MalGAN equipped with the new activation function allows us to generate high-quality AEs, whose statistical distribution is very close to the real data but maintains good evasion rates when input into the attacked ML models.

### 1.1. Contributions

- This work proposes a novel and scalable architecture of distributed cybersecurity components integrated within TeraFlowSDN, the open-source ETSI reference implementation for SDN controllers.
  - The proposed design is integrated as a series of ML-based security components into a microservice-based cloud-native SDN architecture. By using open and standardized interfaces (e.g., Protocol Buffers and gRPC), this solution allows an easy interconnection, exchange and substitution of security components in a seamless and modular way that does not impact the remainder of the components in the controller. Furthermore, the standardized design of the proposed components is general enough to be used by other ML-based components that the TeraFlowSDN controller could require in the future.
  - Contrary to other solutions that only offer either network attack detection or mitigation solutions, our components offer a fully integrated pipeline, including both of these actions.
- Existing energy optimization techniques were examined and compiled into a novel set of 11 optimization strategies to reduce the energy consumption of deep neural networks. These novel strategies were applied to the TeraFlowSDN ML-based attack detectors without a noticeable negative impact on the models' performance. This allowed for a reduction in the average energy consumption by up to 83.30% with minimum performance degradation.
- In order to protect TeraFlowSDN ML-based components against adversarial attacks, this study proposes a new mechanism to generate high-quality adversarial examples that can be used to retrain and fortify ML models. To this end, we present, as a novelty, an extension of the MalGAN generative network equipped with a Smirnov Transform in the generator network to produce adversarial examples that can fool ML models but are still very close to real attack examples.
- Previous works have typically relied on outdated network traffic datasets, such as the NSL-KDD dataset for training IDS. Our work makes use of 5G network traffic generated in a new environment based on a fully virtualized 5G network that generates realistic traffic reflecting current standards.

### 1.2. Paper Structure

The remainder of the manuscript is organized as follows. Section 2 provides an overview of previous work related to the integration of ML-based IDS into SDN solutions. Section 3 describes the setup of the cryptomining attack scenario selected as the use case and provides a detailed description of how cryptomining attacks are performed and detected as well as the mitigation strategy adopted to counter them. Section 4 shows the integration of the Cybersecurity ML-based components into the TeraFlowSDN architecture describing the components and their interfaces as well as the workflows implemented to integrate the end-to-end attack detection and mitigation process into the TeraFlowSDN controller.

Section 5 describes the ML model used to integrate the cryptomining attack-detection capability into TeraFlowSDN. In particular, it details the setup in Telefonica premises to obtain the dataset used to train the model, the model architecture and the training and evaluation procedures. Section 6 presents the results of the energy efficiency optimization of the ML models and highlights the main trade-offs between performance and energy efficiency. Section 7 shows how adversarial attack resilience is added to the ML-based TeraFlowSDN cybersecurity components. Finally, in Section 8, our main conclusions and a summary of the main findings of this work are presented along with a proposal for future research and development in this field, highlighting the areas that require more attention and exploration.

## 2. Related Work

Since the introduction of the OpenFlow protocol [19] in 2008, many aspects of the field of softwarization of telecommunication networks have evolved. SDNs decouple the control and data plane so that they are managed by a centralized controller [20]. Since the SDN controller has a global view of the network, it can access a variety of information from both the network and the data plane. This centralization of information facilitates the creation of Machine-Learning solutions in SDNs to make knowledge-based decisions in diverse areas of networks.

In recent years, great efforts have been made in the industry to develop and integrate automation and decision making in the network field. This has led to a variety of applications of Machine Learning to solve different problems related to Software Defined Networks (SDNs). For instance, some of these works [21] integrated Machine-Learning techniques in SDNs to automate the traffic classification of slices. Others [22] used Deep-Learning techniques, such as Long Short-Term Memory and Gated Recurrent Units to predict network traffic, thus, allowing the SDN controller to predict and manage traffic congestion by rerouting the flow to a path with more available bandwidth.

In [23], the authors showed how security vulnerabilities have been found in multiple widely used SDN architectures, such as OpenDaylight (ODL) [24] and the Open Network Operating System (ONOS) [25]. Among other security issues, tampering with network information, service interruption and unauthorized access to system information have been reported on SDN networks.

Previous works [5,6] have presented Network Intrusion Detection Systems (NIDS) that use Snort to detect attacks in traffic and later apply specific countermeasures, such as network reconfiguration. Although these tools may work in specific scenarios, Snort is a signature-based detection system and is, therefore, not equipped to detect unknown attacks or attack patterns outside of its ruleset. Furthermore, it has been shown that rule-based heuristic methods, such as Snort, are not suitable for NIDS when dealing with encrypted traffic [7].

The layer of encryption makes it difficult for Snort to inspect the contents of the packet, and therefore, attackers can use this to bypass this detection method. This is a significant problem as increasingly traffic is being encrypted, making it increasingly difficult for NIDS to detect attacks. For that reason, this study focuses on deep-learning solutions as they do not rely on packet inspection, and our training dataset includes both encrypted and non-encrypted traffic to ensure that the extra layer of encryption will not prevent our system from detecting malicious traffic.

The literature includes works where a deep analysis of the creation of a NIDS in relation to an SDN can be observed [8]. The authors of this study performed a series of tests on different Machine-Learning classifiers, such as Random Forest, Decision Trees and XG-Boost, to find the best-performing algorithm to detect attacks. The experiments were performed using the NSL-KDD dataset, which is a data-mining dataset containing different traffic features with their corresponding tag that determines whether the traffic is normal or part of an attack. This dataset also contains a variety of attacks, containing Denial-of-Service, User-to-Root, Remote-to-Local and Probe attacks.

This paper achieved good results in the detection of attacks and their classification but does not go into detail on how to proceed with these attacks. In contrast, our solution offers a full pipeline of components that interact with each other within the SDN to stop the attack connections from their source. Furthermore, even though the NSL-KDD dataset is one of the most popular and complete IDS datasets, it still suffers from some of the problems [9] that prevent it from being a perfect representative of existing real networks. These excerpts from the KDD Cup 1999 dataset do not represent realistic traffic and do not contain traffic belonging to 5G networks, such as in our study. In contrast, the traffic that was used to train and test our models was generated in a fully virtualized 5G network and represents normal and attack traffic faithfully according to the latest network traffic patterns and encryption methods.

Another work in the literature implements an NIDS to detect Distributed Denial-of-Service (DDoS) attacks using an Openflow SDN controller [26]. Their method uses Self-Organizing Maps and an unsupervised artificial neural network, which was trained with standard and DDoS traffic generated by them to classify traffic. In their study, they also focused on the lightweight nature of their method. Since they only used a small number of relevant features, they achieved a lower overhead than traditional approaches based on the KDD-99 dataset.

This study offers novelty in the selection of a small number of features and the creation of a custom dataset to detect DDoS attacks. However, it does not include any type of countermeasure once the attack has been detected. The purpose of NIDS in this study is merely informative, since it alerts the system administrator of the predicted threat. This research served as a seminal contribution to the field of Network Intrusion Detection (NID) in Software-Defined Networking (SDN); however, despite its initial significance, its efficacy as a detection system remains untested in light of recent developments, such as the advent of 5G network traffic, which may have altered network traffic patterns.

Some works have focused on addressing attack mitigation in IoT networks by using an SDN controller and previously assuming that the attacks have already been detected by a NIDS present in an IoT system [27]. This study proposes the use of an in-system SDN controller as a honeypot to isolate the attacker's traffic. The purpose of this controller is two-fold: to isolate the attacker and maintain a connection with them through network spoofing and the use of phantom nodes. Network spoofing is a technique in which the controller creates false network elements, such as fake IP addresses, to mislead the attacker.

Phantom nodes are network elements that do not actually exist but appear to be present to the attacker. Using these techniques, the controller can maintain a connection with the attacker while isolating their traffic in a quarantined environment. This allows the controller to detect the attacker's malicious activities without placing the actual system at risk. This study presents an innovative approach to attack mitigation and the tracking of malicious activity utilizing an SDN controller. However, it should be noted that the study does not provide information on the initial step of attack detection and, therefore, represents only part of the entire threat detection and prevention pipeline.

Other studies offer a complete pipeline of intrusion detection and attack mitigation in different fields, for example, in the context of Industrial Healthcare Systems using an SDN controller and Reinforcement Learning [10]. This paper follows the detection of attacks on telecontrol equipment and systems in the medical field and tests a series of Machine-Learning methods, including Logistic Regression, Random Forest, SVM and CART Decision Tree Classifiers, which offer the best results against the TCP/IP network and device payload flow statistics.

Once the attacks are classified, the cost of the attack is computed, and this information is used to select an appropriate mitigation strategy. This work offers promising results in the context of cyberattack detection and mitigation using Machine Learning in SDNs but does not consider the possible vulnerabilities of the models to adversarial attacks. Our work takes this concept one step further by evaluating the models against sophisticated adversarial attacks techniques that could potentially mislead the Machine-Learning models by introducing slight disturbances in the input features [11,12] making them resilient against them.

Many recent works have highlighted the good results obtained by applying ML techniques to NIDS, which achieved results of over 99.46% F1-score in DoS attack detection [28], 97.29% F1-score in DDoS attack detection [29] and over 99.95% F1-score detecting different types of connection flooding [30]. All of these show the good applicability of DL techniques for cyberattack detection. However, all of them fail to consider the energy needs of such systems when scaled to the dimensions of telecommunication systems. Even though some of them mention feature reduction to reduce model complexity [30], relating this to a faster processing time, the studies, overall, do not offer other possible optimization strategies that could make the models more energy-efficient and applicable to real-life traffic dimensions.

In conclusion, after performing a thorough inspection of the literature, it is evident that most works lack a complete pipeline of network attack detection and mitigation. Many of the solutions focus only on one of the two components of a complete system and do so in a case-specific way that does not make them applicable to other ML-based scenarios. Furthermore, none of the previous SDN controller proposals (e.g., ODL, NOX and ONOS) contemplated that IDS and its ML components that use industry standard protocols and interfaces or that are integrated within their architecture by design as TeraFlowSDN does with its components.

This lack can limit the scalability and flexibility of previous solutions, which is not the case in TeraflowSDN, where the ML components can access the rest of the internal components in a flexible and efficient way using the interfaces and protocols established within the microservice-based architecture. In addition, to the best of our knowledge, previous works lack information regarding two vital points.

On the one hand, they failed to analyse how the energy consumption impacts their ML-based cybersecurity solutions, and they did not attempt to take any steps towards making their ML models more energy efficient by applying optimization strategies. This is a critical point, especially considering the scale of real-world telecommunication applications. On the other hand, previous solutions that have applied ML towards the creation of NIDS in SDN solutions failed to test the resilience of their models against novel adversarial attacks, which could leave their systems vulnerable to false negatives by introducing small perturbations in the input data.

Table 1 presents a summary of the primary findings and contributions of previous studies in the research field that this article addresses. As demonstrated in the table, the fortification of ML models against adversarial attacks has been largely overlooked in the current literature. On the other hand, although there are some works that address the issue of optimizing the energy efficiency of ML models in this context, these works are largely speculative and lack experimental validation. Our proposal departs from related works by addressing these two crucial issues by providing empirical evidence and experimental validation of the proposed techniques.

The table also shows how, even though most of the proposed solutions base their experiments on open-source SDN controllers, the availability of their code in most cases is non-existent or limited to pseudo-code. This can hinder their applicability to other problems and is a main distinguishing point in our proposal. In sharp contrast, the solution proposed in this work is integrated in TeraFlowSDN (TFS) (an open-source controller with publicly available code) and utilizes standardized interfaces to make it easily adaptable to other ML problems in an SDN controller. The open nature and modularity of our approach are key characteristics that distinguish it from previous proposals.

**Table 1.** Comparative analysis of the contributions of the proposals related to our work.

| Proposals | Public Code | SDN Controller Used | Dataset | Attack Types | NIDPS Implementation | Detection Method | Mitigation Strategy | Energy Efficiency Optimization | Fortification against Adversarial Attacks |
|---|---|---|---|---|---|---|---|---|---|
| Our work | Yes | TeraFlowSDN (TFS) | Synthetic Dataset | Cryptomining Attacks | Complete | Deep Neural Network | Drop packets of detected attack connections | Yes | Yes |
| Xing et al. [5] | No | SDNIPS controller | Synthetic Dataset | ICMP | Complete | Snort | Network reconfiguration | No | No |
| Chung et al. [6] | No (Pseudo-code available) | non-specified controller | Synthetic Dataset | DDoS | Complete | Snort | Countermeasure pool | No | No |
| Alzahrani et al. [8] | No | POX SDN controller | NSL-KDD | DDoS, PROBE, R2L and U2R | Only detection | Decision Tree, Random Forest and XGBoost | No | No | No |
| Radoglou–Grammatikis et al. [10] | No (Pseudo-code available) | Ryu controller | Custom IEC 60 870-5-104 dataset | IEC 60 870-5-104 cyberattacks | Complete | CART classifier | Security strategies, including asset isolation | No | No |
| Zhou et al. [11] | No (Pseudo-code available) | N/A | UNSW-SOSR2019 | Hierarchical Adversarial Attack | NA | N/A | NA | No | No |
| Aiken et al. [12] | Yes | Faucet SDN controller | CICIDS dataset | Adversarial Attacks | NA | N/A | NA | No | No |
| Perera et al. [21] | No | RYU Controller | P Network Traffic Flows (Kaggle) | NA | None | K-Means, SVM, Decision Trees and Random Forest | NA | Limited | No |
| Prabhavat et al. [22] | No (Pseudo-code available) | RYU Controller | Synthetic Dataset | NA | None | LSTM and Gated Recurrent Units | NA | Limited | No |
| Braga et al. [26] | No | NOX Controller | Synthetic Dataset | DDoS | Only detection | Self-Organizing Maps | No | No | No |

**Table 1.** *Cont.*

| Proposals | Public Code | SDN Controller Used | Dataset | Attack Types | NIDPS Implementation | Detection Method | Mitigation Strategy | Energy Efficiency Optimization | Fortification against Adversarial Attacks |
|---|---|---|---|---|---|---|---|---|---|
| Lin et al. [27] | No | SDN-enabled hardware switches and ONOS SDN Controller | Synthetic Dataset | IoT cyber attacks | Only mitigation | N/A | Honeypot traffic rerouting | No | No |
| Kamel et al. [28] | No | No implementation | DDOS-attack SDN dataset | DDoS | Only detection | Decision Trees | No | No | No |
| Makuvaza et al. [29] | No (Pseudo-code available) | No implementation | CICIDS 2017 | DDoS | Only detection | Deep Neural Network | No | No | No |
| Alzahrani et al. [30] | No | RYU Controller | Synthetic Dataset | Fin flood, UDP flood, ICMP flood, OS probe scan, port probe scan, TCP bandwidth flood and TCPsynflood | Only detection | KNN, AdaBoost, Decision Trees, Random Forest, Naïve Bayes, Multilayer Perceptron, SVM and XGBoost | No | No | No |

## 3. Cyberthreat Analysis and Protection Use Case

The scenario presented in this work introduces scalable and reliable security assessment of the services established using the TeraFlowSDN controller. TeraFlowSDN is an innovative open-source, cloud-native Software-Defined Networking (SDN) controller that integrates with existing Network Function Virtualization (NFV) and Multi-Access Edge Computing (MEC) frameworks and offers revolutionary capabilities for both service-level flow management and the integration and management of the underlying network infrastructure, including transport network elements (optical and microwave links) and Internet Protocol (IP) routers, while incorporating cybersecurity capabilities through ML and forensics for multi-tenancy based on Distributed Ledgers.

### *3.1. Cyberthreat Analysis and Protection Scenario*

The use case presented in this work demonstrates that novel approaches enabled by Machine-Learning techniques allow TeraFlowSDN 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, this work proposes the implementation of a distributed solution where ML components are deployed on Point of Presence (PoP) nodes. To this end, a feature extractor is deployed at the network edge to collect and summarize the packets. The flow statistics aggregated by the feature extractor are sent to an ML classifier. Based on the real-time identification of malicious flows, the ML model is able to report to the TeraFlowSDN controller at scale to perform a security assessment.

The setup considered for this demonstration is illustrated in Figure 1. Assuming a typical telecommunication MPLS-based network, a Level 3 Virtual Private Network (VPN) service (L3VPN) is deployed using the TeraFlowSDN controller. The controller activates this service using provisioned templates over the standardized Internet Engineering Task Force (IETF) Network Configuration Protocol (NETCONF) South-Bound Interface against the different Provider Edge (PE) routers from the ADVA manufacturer. In this demonstration, a traffic generator, emulating a branch office, is connected to one PE to replay a mix of normal traffic with cryptomining malware activity. The second PE, the central office, provides internet access offered by the L3VPN service and leverage by the malware. This specific malicious traffic is represented as a red dashed line in Figure 1.

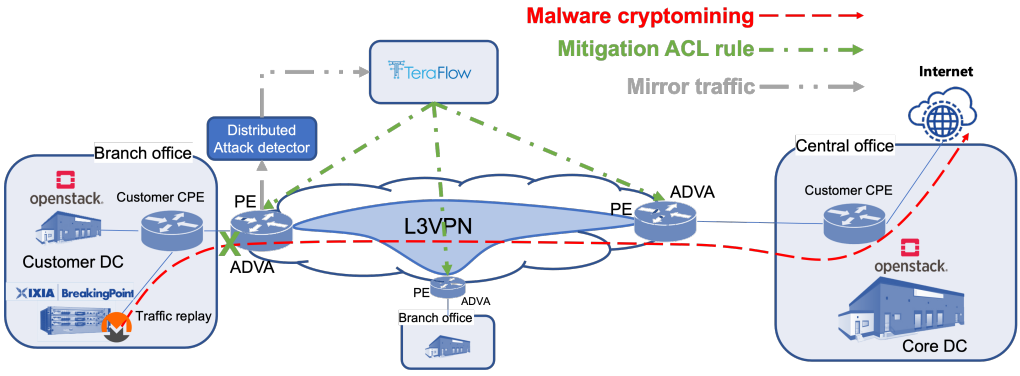



**Figure 1.** Global overview of the cyber threat analysis and protection process [31].

As part of the VPN provisioning process, a request for mirroring only the traffic in the logical interfaces that conform to the L3VPN is also included to copy the traffic towards a logical component co-located to the ADVA (grey dash-dot-dot line in Figure 1). This distributed component (detailed in Section 4.2.1 as the Distributed Attack Detector) will extract and calculate statistical features from network flows to be delivered to the TeraFlowSDN controller for further processing.

The Cybersecurity TeraFlow NetApp component identifies the attack as a cryptomining activity and proposes a mitigation solution to the TeraFlow Core components, which triggers the mitigation. This mitigation is instantiated (the green dash-dot line in Figure 1) as a new customized Access Control List (ACL) rule in the ADVA router with specific parameters (the protocol identifier (Transmission Control Protocol), destination IP address and destination port). Figure 1 shows an additional branch office to represent multisite L3VPN functionality where the same rule can be enforced in additional PE routers, thus, providing protection to all offices of the L3VPN client.

### *3.2. Cryptomining Attack Detection*

One of the most prevalent contemporary networking threats is the misuse of computing resources for cryptomining attacks. As described in [7], cryptomining entails the validation of transactions on a decentralized cryptocurrency blockchain. A cryptomining attack involves the creation of a botnet, which consists of compromised devices that act as miners to validate transactions and earn digital currency rewards, such as Ethereum (ETH) and Monero (XMR), for the attacker.

The attacker may exploit devices already infected with malware or infect new devices to enlist their resources to create the botnet for cryptomining. The attacker may use various methods, such as spreading malicious links on social networks, phishing attacks, and disseminating malicious applications, to infect devices.

Additionally, the attacker must choose the cryptocurrency to mine and the cryptomining pool to join to validate transactions. A mining pool is a service that enables miners to pool their resources to validate transaction blocks and receive rewards. After obtaining all the necessary components, the attacker can establish the botnet to mine cryptocurrencies for their gain.

According to [32], the network is the most effective place to detect cryptomining traffic promptly and accurately. However, detecting such activity on the network can be challenging due to encryption methods that protect the payload and obscure its contents. For instance, attackers can leverage the Secure Sockets Layer/Transport Layer Security (SSL/TLS) encryption protocol to conceal the cryptomining protocol within the encrypted communication's payload. This renders traditional techniques, such as Deep Packet Inspection (DPI) or Cryptomining Pool Domain Name Identification (in the case of using Server Name Indication (SNI) or web proxies) ineffective in detecting cryptomining activity in current networks. Consequently, more advanced techniques are required to equip cybersecurity professionals to deal with these real-world situations.

To address these challenges, ML techniques can be employed to train models that accurately identify the presence of cryptomining traffic in real time, even when encrypted, by leveraging network and transport-level data-flow characteristics [32]. Accordingly, in this study, an ML model was trained using a substantial set of network features derived from network flow statistics to detect cryptomining activity for both encrypted and unencrypted links with a high degree of accuracy. This ML model forms part of the Cybersecurity TeraFlowSDN NetApp, which enables the detection of cryptomining traffic in the network's data plane in real time, thereby facilitating appropriate remediation measures to safeguard the network.

## 4. Integration of an ML-Based Cybersecurity Detector and Mitigator in the TeraflowSDN Controller

In this section, first, we provide a brief overview of the TeraFlowSDN architecture. Then, we discuss the integration of the ML-based cybersecurity detector and mitigator in the TeraFlowSDN architecture. Finally, we present the workflows of the proposed integrated system for detecting and mitigating cyber-attacks in TeraFlowSDN networks.

### 4.1. TeraFlowSDN Architecture

TeraFlowSDN is an open-source, cloud-native reference implementation for Software-Defined Networking (SDN) controllers that has been endorsed and sponsored by the European Telecommunications Standards Institute (ETSI) to support high-capacity IP and optical networks and to provide a toolbox for experimentation with innovative network technologies and use cases beyond 5G. ETSI is a pre-eminent non-profit organization that specializes in the development and publication of global standards for Information and Communication Technologies (ICT).

TeraFlowSDN is a collaborative effort of the Open Source Group TeraFlowSDN (OSG TFS), a dedicated working group within ETSI focused on the development of TeraFlowSDN to provide a comprehensive set of tools and platforms for the rapid prototyping and experimentation of novel network technologies and use cases. OSG TFS builds upon the results of the EU-funded TeraFlow 5G PPP research project and has developed a microservices architecture that is designed to facilitate network transformation. The software platform provides support for features, such as flow aggregation, management, network equipment integration, AI/ML-based security and forensic evidence for multi-tenancy. The software will also be a valuable tool for research projects and ETSI groups working on network transformation.

The software platform will be instrumental in addressing the challenges of autonomous networks and cybersecurity, which are prevalent in the telecommunications industry. Additionally, the software will be beneficial to several ETSI industry specification groups that are focused on network transformation and will facilitate the integration of existing Network Functions Virtualization (NFV) and Multi-Access Edge Computing (MEC) frameworks. Furthermore, the software is designed to interoperate with the ETSI Open Source MANO (OSM) platform.

TeraFlowSDN also strives to gain support and foster collaboration with existing and future research projects in the 5G PPP and the Smart Grid and Services Joint Undertaking (SNS JU) domains. The TeraFlowSDN source code is publicly accessible in the repository (https://labs.etsi.org/rep/tfs/controller, accessed on 5 April 2023) under the Apache 2.0 license, making it accessible and available to a wide range of stakeholders in the ICT industry.

The TeraFlowSDN controller architecture consists of stateless microservices interacting with each other to fulfil network management tasks in addition to a few stateful microservices responsible for keeping the state of the network. TeraFlowSDN relies on Kubernetes to handle the containers supporting the microservices. Kubernetes is a state-of-the-art container orchestrator that provides a broad set of management capabilities and can operate geographically distributed infrastructures.

Figure 2 shows the proposed microservice-based architecture. Following the design principles from cloud-native applications, each component is implemented as a microservice that is able to export a set of Remote Procedure Call (RPC) services to other components. Each microservice can be instantiated once or with multiple replicas, which allows the application of load-balancing techniques. By adopting stateless microservices, requests can be handled by any replica of the microservice.

Load balancing works by establishing an endpoint that will receive all the requests for a service. The endpoint acts as a load balancer by delegating each request to one of the replicas of the service. The load balancer is also responsible for keeping track of the replicas, i.e., tracking the addition and deletion of replicas and updating its internal list of replicas. Depending on the RPC implementation adopted, the built-in Kubernetes load balancer may be used, or an external one may be adopted. Each replica is composed of a pod, i.e., a collection of containers that are managed by Kubernetes as a single entity.

The Context component stores the network configuration (e.g., the topologies, devices, links and services) and its status as managed by the TeraFlowSDN components in a No-SQL database to optimize concurrent access. Internally, it implements a database API enabling switching between different backends. The TeraFlowSDN controller uses its North-Bound

Interface (NBI) component (previously known as Compute) to receive Layer 2 Virtual Private Network (L2VPN) requests and convert them to necessary connectivity services or Transport Network Slices via the Slice and Service components.

The Service component is responsible for selecting, configuring and deploying the requested connectivity service through the South-Bound Interface (SBI). To this end, the SBI component interacts with the network equipment through pluggable drivers. In addition, a Driver Application Programming Interface (API) has been defined to facilitate the addition of new network protocols and data models to the SBI component. The Automation component implements several Event Condition Action (ECA) loops to define the automation procedures in the network.

Monitoring manages the different metrics configured for the network equipment and services, stores monitoring data related to selected Key Performance Indicators (KPI) and provides the means for other components to access the collected data. Internally, the Monitoring component relies on a database to store the monitoring data as a time series, exploiting its powerful querying and aggregation mechanisms for retrieving the collected data.

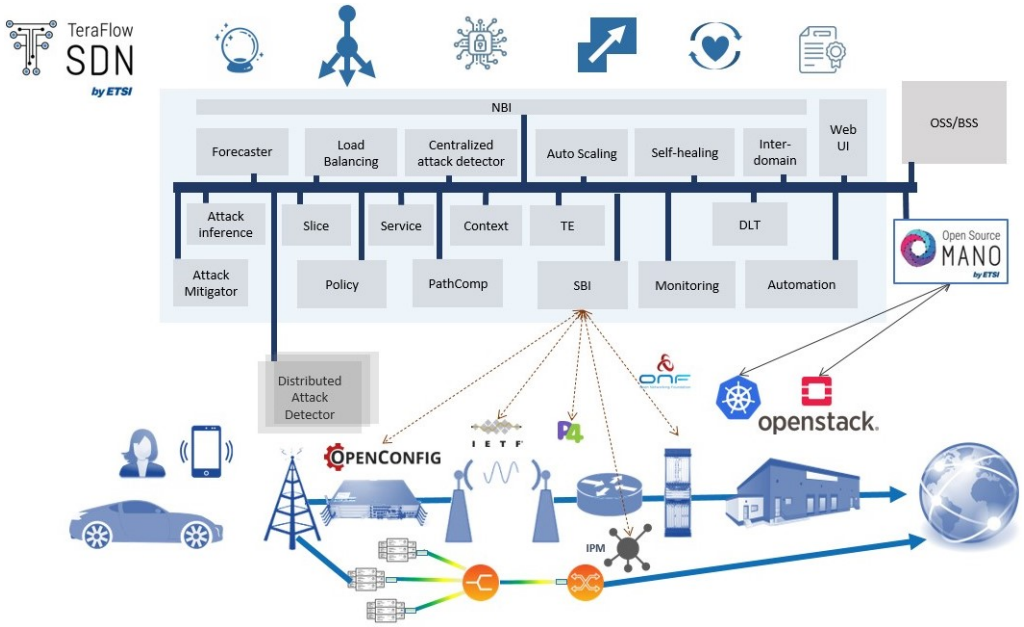



**Figure 2.** TeraFlowSDN architecture [33].

The North-Bound Interface (NBI) component serves as the interface for internal gRPC (gRPC Remote Procedure Call) and protocol buffers towards external Representational State Transfer (REST)-like requests. It provides a Representational State Transfer API (REST-API) that is based on NBI external systems, such as Network Function Virtualization (NFV) and Multi-Access Edge Computing (MEC) frameworks. Another component included is a Web-Based User Interface (WebUI) that uses the gRPC-based interfaces made available by the TeraFlowSDN components to inspect the network state and to issue operational requests to the TeraFlowSDN components.

TeraFlowSDN provides extended and validated support for OpenConfig-based routers and interaction with optical SDN controllers through the Open Networking Foundation (ONF) Transport API (TAPI). Moreover, TeraFlowSDN release 2 includes complete integration for microwave network elements (through the Internet Engineering Task Force (IETF) network topology YANG model) and Point-to-Multipoint integration of XR optical transceivers and P4 routers. New features for P4 routers include loading a P4 pipeline on a given P4 switch, obtaining runtime information (i.e., flow tables) from the P4 switch and pushing runtime entries into the P4 switch pipeline, thus, allowing total usage of the P4 switches.

Cybersecurity components are integrated in the architecture for attack detection (either distributed or centralized) and mitigation in order to protect the network from known and unknown cyberthreats at the IP and optical levels. The integration of the IP layer cybersecurity components is described in detail in the next section.

### *4.2. Integration of the Cybersecurity Components in the TeraFlowSDN Architecture*

The TeraFlowSDN controller utilizes a robust cybersecurity framework to protect the network from potential cyberthreats that could compromise the integrity or performance of the network. The Cybersecurity NetApp, consisting of three main modules, is focused on detecting and mitigating network attacks to ensure seamless and secure functioning.

This section delves into the details of the different components that compose the cybersecurity framework and how they cooperate to provide a continuous assessment of the security status of IP services on the network. The Cybersecurity NetApp focuses on the capture, identification and mitigation of network threats, implementing a protection layer that is crucial for the correct functionality that SDN controllers need to provide.

The Cybersecurity NetApp includes two core centralized components, the Centralized Attack Detector and the Attack Mitigator, along with a distributed component, the Distributed Attack Detector, to be placed at a remote site (e.g., a Point of Presence (PoP) node). The distributed attack detection and mitigation workflow provides TeraFlowSDN with a continuous assessment of the security status of IP services. Figure 3 depicts the three cybersecurity and other core components that live in the TeraFlowSDN controller and how they are connected together to implement the end-to-end cyber threat analysis and mitigation process in the network.

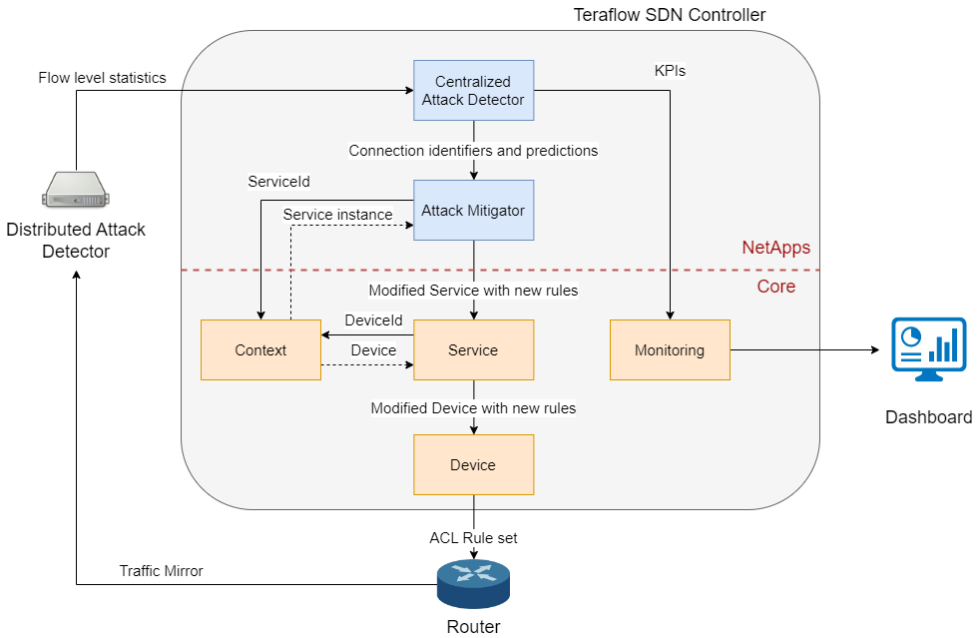



**Figure 3.** Cybersecurity component architecture.

#### 4.2.1. Distributed Attack Detector

The Distributed Attack Detector (DAD) component monitors the network data plane for the presence of malicious network flows by receiving IP traffic from co-located packet processors. The DAD is deployed at the edge of the network (e.g., at the central office or edge data centre) to improve the scalability and response time in the attack-detection process and enable the real-time detection of malicious traffic. For this purpose, a packet processor is used to generate statistical summaries of the network flows by aggregating packets 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 approach of sending summary statistical characteristics of monitored traffic to the CAD also favours scalability, as it eliminates the need to send full traffic information to the centralized controller. Monitoring traffic at the IP layer is expected to use a considerable amount of bandwidth between the packet processors and the DAD, but avoiding the transmission of huge amounts of telemetry to the TeraFlowSDN controller is a major improvement in terms of preserving the network bandwidth to the controller and, therefore, improving the scalability. In addition, processing traffic at the edge of the network allows for reduced delays in the attack-detection process.

Unlike the other components, the DAD does not expose a gRPC server but rather runs a script to obtain and process network traffic. In the current deployment, the packet processor is emulated by re-injecting network packets previously stored in a PCAP file using tcpreply, which is a standard tool available on Linux systems. The injected packets are processed with the Tstat [34] tool to generate statistical summaries of the network flows by aggregating all new packets arriving within a specific time window that can be configured.

In addition, Tstat obtains additional information about the status of the connection and stores this information in the log file. The DAD continuously reads the information generated from this log file, processes it to extract the information in a structured way and adjusts it to apply the gRPC message format expected by the CAD. Once this is completed, the DAD communicates with the CAD to send this information.

In the current implementation, it uses unary gRPC messages to report the traffic monitoring summary information collected by Tstat to the Centralized Attack Detector component, which makes inferences with the ML model to detect IP-level attacks. In later versions, flow messages will be implemented to avoid the delay limitations that can occur with current unary messages.

#### 4.2.2. Centralized Attack Detector

The Centralized Attack Detector (CAD) component provides IP-layer attack-detection capabilities and a consolidated attack-detection mechanism based on DAD reports. The CAD consolidates information collected from multiple instances of the DAD. This allows for the monitoring of malicious network traffic while forming a view of the security status of IP traffic. The CAD stores the information it receives from multiple instances of the DAD within a certain time interval, and this can be parameterized in a buffer. From the summarized traffic statistics received from the different instances of the DAD, this component performs attack detection using an embedded ML model.

After the configured time interval elapses, the ML model classifies each connection in the buffer as normal traffic or as part of an attack, and a confidence level decision is derived. From this inference, the CAD produces a description of the connection, including a confidence value indicating the probability that the connection is an attack or normal traffic. If a connection is detected to be part of an attack with a confidence level at or above a configurable threshold, the CAD notifies the Attack Mitigator component with the attack description, providing a full characterization of the attack properties and other relevant information to perform an attack-mitigation strategy.

In the current implementation, the ML model used is the Random Forest model that was previously converted and stored in the ONNX (Open Neural Network Exchange) format. This format allows the embedding of a compiled model with an optimized graph that can reduce the overall size of a model, speed up the prediction inference time and reduce the use of computing resources.

In addition, security monitoring cycles are run periodically with a configurable time interval to collect Key Performance Indicators (KPIs) that provide an overview of the security status of the network. This information is stored in a database for further analysis and security audits. In addition, these KPIs are displayed on a Grafana dashboard

to provide the network administrators with a real-time view of the current state of the network cybersecurity.

#### 4.2.3. Attack Mitigator

The Attack Mitigator (AM) component is responsible for computing viable attack-remediation solutions to prevent the execution of attacks identified by the CAD component. Upon receiving an attack notification, the role of the AM component is to instruct certain core components of TeraFlowSDN to enforce appropriate actions that can mitigate the attacks detected on the network.

For example, in the case of detecting a cryptomining attack, the AM component communicates with the Service component, which is responsible for managing the services that are running on the TeraFlowSDN controller, to update the configuration of the service on which the attack has been detected in order to implement a new ACL rule to block the malign connection. This ACL rule is then configured by the Device component in the network devices through the standardized OpenConfig protocol over the South-Bound Interface. In particular, the ACL rule is configured in the ingress interface of the device located at the edge of the network (see Figure 1).

In its current state, the AM component possesses only one mitigation strategy, which consists of adding an ACL rule to the corresponding router that blocks traffic from a specific source that has been classified as an attack. The plan is to evolve this component in the future to add more complex functionalities and different network-aware mitigation strategies. For example, before deciding to permanently block a source, a module could be added to assess the severity of the threat or the confidence that the traffic is an attack. If the potential attack is classified as having low confidence or a small impact, the AM component could wait to receive more alerts corresponding to that source to activate any countermeasures or could activate less severe strategies. The AM could, for instance, consider the risk level to add a timer to the instruction to block traffic and allow it to expire in a certain amount of time.

In addition to this, several interesting mitigation strategies that could be integrated into the AM have been studied in previous works [35]. One of the strategies is threat-based routing in which traffic that is classified as an attack can be redirected through the path with the least-utilized links in terms of bandwidth consumption. This would allow traffic to still reach its destination through a longer route but would minimize the impact on standard traffic.

This solution is very forgiving of false positives and could be a good alternative for attacks classified as low impact or with low confidence in classification. Other solutions proposed by the study are to assign long timeouts to the flows detected as malicious and to aggregate them to occupy the least amount of space in the TCAM (Ternary Content Addressable Memory) tables, thus, reducing the communication from component to component. Another possible mitigation would be to redirect potentially harmful traffic to a separate component that monitors the traffic more closely to ensure that it is not harmful.

The isolation of traffic from the central system was also proposed as an effective mitigation strategy in another study [27], where an in-system SDN controller acted as a honeypot to isolate the attacker's traffic. That study used a separate controller to protect the system while maintaining the connection with the attacker and to mislead them through network spoofing to gather more data about their intentions. Obtaining more information about possible attacks could help develop better security measures in the future.

In summary, although the current implementation of the AM would protect the system against detected attacks by blocking the traffic from the source on the router, the component is still a work in progress, and it is expected to evolve with new functionalities that help it adapt to the context and information of the attack with an array of complex mitigation strategies.

### *4.3. Attack Detection and Mitigation Workflows*

This section presents the main workflows that illustrate how the Cybersecurity NetApp interacts with other TeraFlowSDN core components to implement the detection and mitigation of network attacks at the IP layer.

When a new service is created on TeraFlowSDN, the Cybersecurity NetApp communicates with the core components during the different stages of the process. The Cybersecurity NetApp starts by subscribing to service events from the Context component during start-up. When a service request is received, the service setup stage is triggered, which involves changes to several components of TeraFlowSDN. The service identifier is then returned to the customer who requested the service. The KPI setup stage then starts with the Context component notifying the Cybersecurity NetApp about the new service.

The Cybersecurity NetApp will then begin performing the attack detection and mitigation process on the service and tracking relevant KPIs through the Monitoring component. To implement the detection and mitigation of network IP-level attacks, four workflows were created with each workflow focusing on a specific part of the system.

The first workflow focuses on the DAD component and covers the process of collecting traffic statistics from IP-level traffic and reporting to the CAD component. The second workflow focuses on the CAD component and covers the process of processing the traffic statistics reported by the DAD component and making inferences with the ML model to detect IP-level attacks. The third workflow focuses on the AM component and covers the process of computing attack-remediation solutions in response to attack notifications from the CAD component. Finally, the fourth workflow focuses on the monitoring of cybersecurity KPIs and covers the process of collecting and storing KPIs related to the security status in the network.

#### 4.3.1. Workflow 1—Capture and Label Traffic at the Edge of the Network

The DAD component workflow is specified in Figure 4. First, the DAD component requests the features that serve as input to the ML model in the CAD component via an RCP method to this same component. The DAD stores the list of these features in a local variable. The DAD then communicates via RCP methods with the Context component in order to obtain the *service_id* and *endpoint_id* attributes so that the connection is traceable in the TeraFlowSDN System and the mitigation strategies can later be implemented on the correct devices.

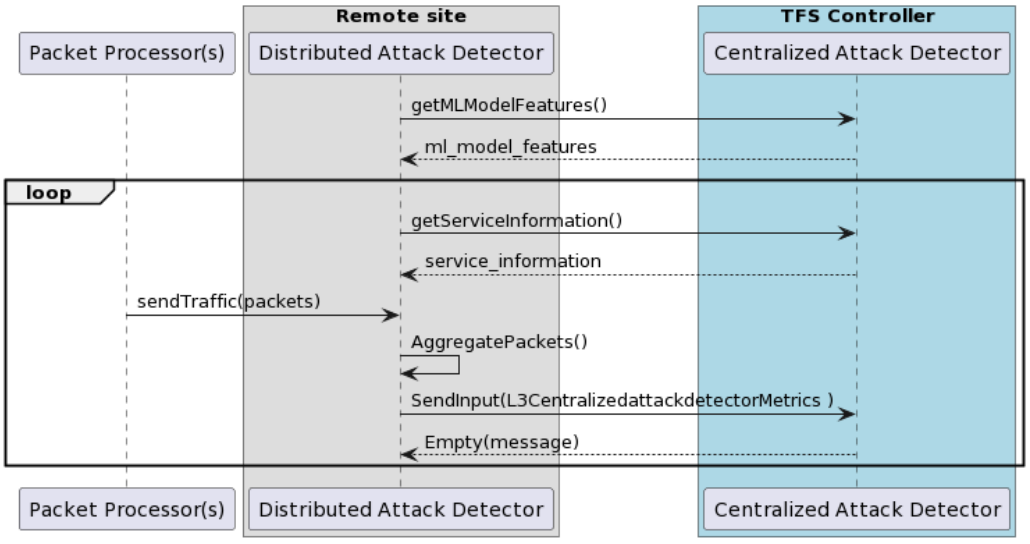



**Figure 4.** Distributed Attack Detector component workflow.

After the traffic is received in the machine where a DAD instance is deployed, it is grouped into flow-level statistics using the Tstat tool. The DAD component selects the

appropriate features that will later serve as input for the ML model using the local list of these features as a guide. Once all of these features and the connection data are grouped into a *L3CentralizedattackdetectorMetrics* object, this object is sent via the RCP method *SendInput* to the CAD.

#### 4.3.2. Workflow 2—Detect Cryptomining Malware Connections Using Supervised ML

As described in Figure 5, the CAD component receives and stores flow statistics in the form of *L3CentralizedattackdetectorMetrics* objects. It then calls a function with these objects as the parameters to perform inference with the ML model using these objects as input features. As a result, the ML model classifies the statistics contained in each of these objects as either belonging or not belonging to a crypto-mining attack. If the statistics belonging to a particular flow are classified as a cryptomining attack, the *SendOutput* RCP method is called and sends the confidence score and the result of the attack classification to the AM, together with the corresponding flow information (source and destination IP address and port, protocol, *service_id*, etc.) in an *L3AttackmitigatorOutput* object.

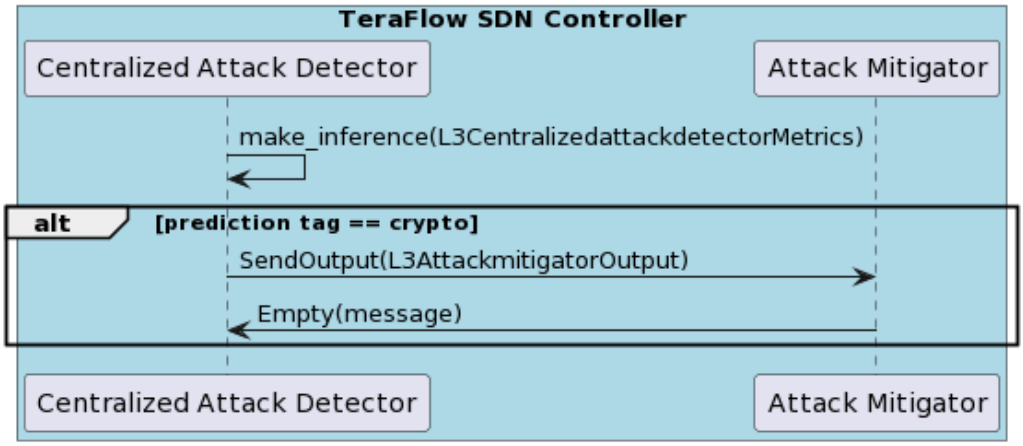



**Figure 5.** Centralized Attack Detector component workflow.

#### 4.3.3. Workflow 3—Mitigate Detected Cryptomining Attacks

The workflow of the AM component is given in Figure 6. In this diagram, it can be seen that, after the component receives the connection data belonging to a cryptomining attack, it creates a mitigation strategy. At the moment, this strategy consists of creating an ACL rule to drop the traffic belonging to that particular connection. AM then communicates with the Context component to receive the instance representing the service where the attack was detected (i.e., the service specified by the *service_id* contained in the flow information). After receiving the service object, the *ComposeMitigation* method is responsible for adding the new ACL rule to drop the traffic belonging to the flagged connection.

After calling the RCP method *UpdateService* with the modified service instance, the Service component propagates the newly created configuration rule to the Device component, which is responsible for incorporating the ACL rule in the corresponding router using the OpenConfig protocol to drop the malign connection, thus, completing the implementation of the current mitigation strategy.

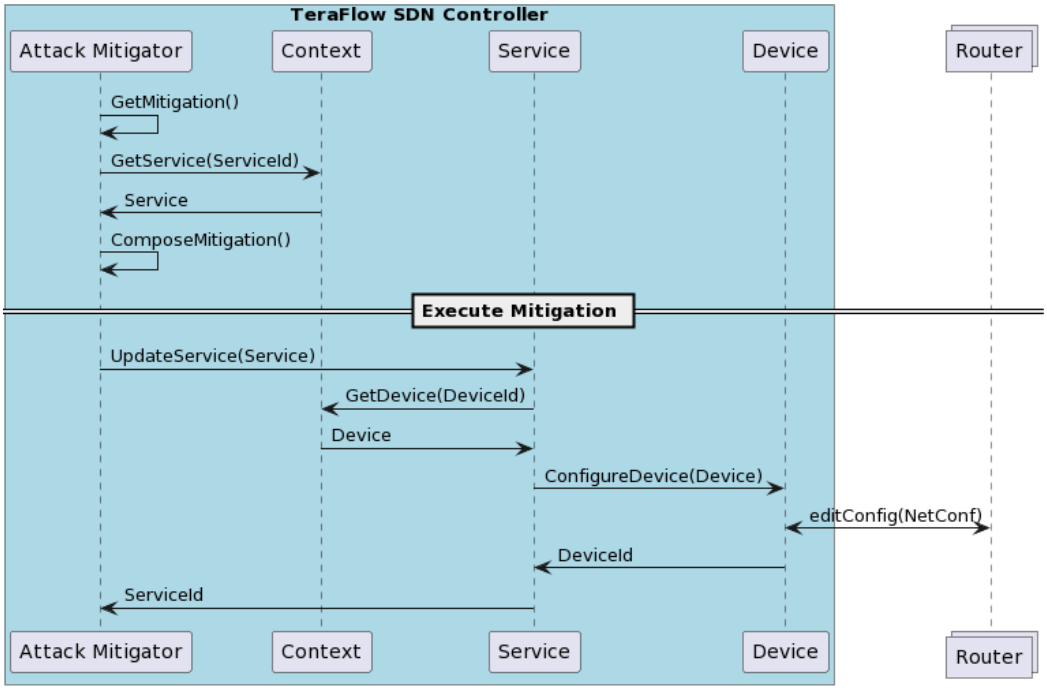



**Figure 6.** Attack Mitigator component workflow.

#### 4.3.4. Workflow 4—Monitor Relevant Cybersecurity-Related Key Performance Indicators

CAD monitors five relevant KPIs for each active service. Below, the cybersecurity KPIs that are observed and recorded and their associated KPI sample type are listed:

- Cryptomining detector confidence in security status over the last time interval (*KPI ML CONF*).
- Security status against cryptomining attacks of the service in a time interval (*KPI L3 CRYPTO SECURITY STATUS*).
- Number of attack connections detected in a time interval (*KPI UNIQUE ATTACK CONNS*).
- Number of unique compromised clients of the service in a time interval (*KPI UNIQUE COMPROMISED CLIENTS*).
- Number of unique attackers of the service in a time interval (*KPI UNIQUE ATTACKERS*).

The values of *KPI L3 ML CONFIDENCE* are collected for predictions that take place during a specific time interval (e.g., 5 s). This is performed separately for predictions that correspond to an attack and predictions that correspond to normal traffic. At the end of each time interval, the values of both lists are aggregated independently to calculate the average. If an attack connection occurred during that time interval, the average confidence of the predictions corresponding to an attack is sent to the Monitoring component as *KPI L3 ML CONFIDENCE* with "1" as the *KPI L3 SECURITY STATUS SERVICE*.

Otherwise, the average confidence of the predictions corresponding to normal traffic is sent to the Monitoring component as *KPI L3 ML CONFIDENCE with "0" as the KPI L3 SECURITY STATUS SERVICE*. The *KPI L3 UNIQUE ATTACK CONNS* counts the number of unique attack connections that were detected in each time interval. As with the previous KPIs, these values are collected during each time interval. Once the interval is over, these values are aggregated and sent to the monitoring component. Note that the packet aggregator running in the DAD component aggregates the new packets from the same connections as soon as they are received, and the characteristics are sent to the ML model.

For this reason, if subsequent packets are received from the same connections, the DAD will produce new statistics that the ML model will also ingest. For this reason, connections may be detected as an attack more than once. However, in *KPI L3 UNIQUE ATTACK CONNS*, these repeated connections will only be counted as one. Similar to *KPI L3 UNIQUE ATTACK CONNS*, *KPI UNIQUE COMPROMISED CLIENTS* measures the number of com-

promised cryptocurrency clients in each time interval by counting the number of flows that correspond to the same source IP.

On the other hand, *KPI UNIQUE ATTACKERS* measures the number of unique attackers in each time interval by counting the number of flows that correspond to the same destination IP. *KPI L3 UNIQUE ATTACK CONNS* provides a measure of the intensity with which compromised clients attack the network. *KPI UNIQUE COMPROMISED CLIENTS* and *KPI UNIQUE ATTACKERS* extend this information by revealing the scale of the compromised network and quantifying how many attackers are involved in attacking the network.

CAD creates these KPIs at launch time by registering *KpiRequest* for each KPI through the Monitoring client and thereby requesting the Monitoring service process to create and add them to the Management DB as depicted in Figure 7. For each *KpiRequest*, a *KpiDescriptor* is provided that includes the service information, device and endpoint identifiers as well as the description and KPI sample type of each KPI. Once successfully created, the KPIs can be effectively monitored by sending samples to the Monitoring service via the RPC method *IncludeKpi*. As each sample is received by the Monitoring service, they are inserted into the QuestDB database, which collects the TeraFlowSDN metrics to be accessible through the Grafana dashboard, where they are displayed in a linear time-series representation.

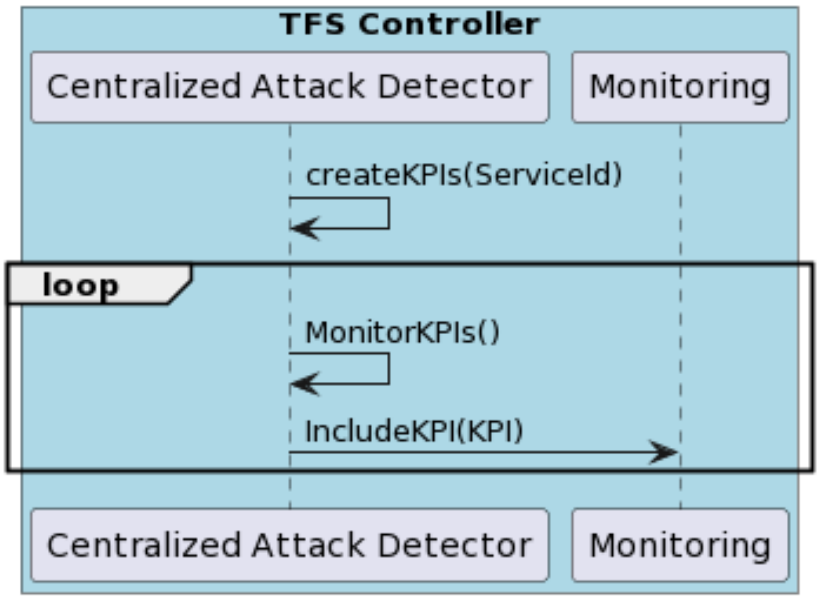



**Figure 7.** Monitoring component workflow.

## 5. Analysis of the Attack Detector

In this section, we describe, in detail, the ML model that we trained for the task of detecting cryptomining attacks in the CAD component. First, we describe the setup that 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.

### *5.1. Cryptomining Dataset Creation and Traffic Labelling*

It is common for other works to make use of the NSL-KDD dataset, which is a data-mining dataset containing different traffic features with their corresponding tag that determines whether the traffic is normal or part of an attack. The dataset contains a variety of attacks, containing Denial-of-Service, User-to-Root, Remote-to-Local and Probe attacks. Even though the NSL-KDD dataset is one of the most popular and complete IDS datasets, it still suffers from some problems [9] and is, therefore, not a perfect representative of existing real networks.

In contrast to these excerpts from the KDD Cup 1999 dataset that do not represent realistic traffic, all of our experiments and the training of our models were performed using 5G network traffic generated in a real environment based on a fully virtualized 5G network [7]. This environment allowed us to emulate real 5G traffic in a controlled way that was on demand and to take into account this new standard that had not been considered in most of the solutions proposed in the past.

The dataset that was used to train the model that will serve as a target in the demonstration of the proposed methodology was developed for the precise task of detecting cryptomining attacks [32]. This dataset was generated in the Mouseworld lab [36], an open lab for 5G experimentation that provides Network Digital Twin emulation capabilities [37] and is located at the Telefonica I+D premises.

This emulation environment allowed us to configure and execute specific attacks mixed with normal traffic (e.g., web, file hosting and streaming) by instantiating virtual machines that deployed normal traffic and specific attack clients connected to real servers located at different points on the internet. In this way, the Mouseworld lab can be used to set up and emulate attack scenarios in a controlled way and to generate and collect, in a PCAP file, all packets of the attack and normal traffic to be used later for the training and testing of ML algorithms. One key feature of the Mouseworld Lab is the repeatability capacity, which allows us to evaluate different mitigation tools or versions under the same conditions and using similar statistical patterns.

The data that were collected for our study in the Mouseworld lab contain traffic samples represented by a set of flow (TCP connection) statistics derived from network packets using the Tstat tool. The statistics of a TCP connection were calculated periodically (at fixed intervals or when a new burst of packets was received), resulting in many different examples in the dataset for the same flow representing the state of the connection over its lifetime. This traffic data was labelled to create the dataset that was used to train and test the cryptomining detector.

In particular, two types of traffic can be found in the dataset, samples corresponding to normal traffic and samples corresponding to cryptomining attacks. In this case, each sample of the dataset was tagged as either 0 (normal traffic) or 1 (cryptomining attack traffic) using the IPs and ports of the known attack connections.

### *5.2. Training of the Machine-Learning Model for Cryptomining Detection*

The architecture employed to implement the model for the task of cryptomining detection utilized a Fully Connected Neural Network (FCNN) architecture. This approach accurately predicts the labels of the dataset with a high degree of accuracy. This architecture was chosen based on its superior performance in the context of the cryptomining attack as demonstrated in a previous study that performed an exhaustive comparison with other similar techniques [32].

The TensorFlow library was used to train a FCNN-based classifier to predict whether a connection corresponded to cryptomining activity or not according to all features derived from Tstat statistics except for the IPs and ports, which were used only to label the dataset (class labels). Note that source and destination IPs and ports can be easily changed by the attacker; therefore, they do not provide significant information to the ML-based detector.

The structure of the FCNN model is specified below. In particular, the 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 output layer. The training hyperparameters are as follows: a batch size of 4096 and the Adam optimizer with a learning rate of 0.001. Furthermore, the early stopping technique is used to automatically terminate the training process if the validation loss does not improve for 20 epochs, restoring the model weights to those obtained in the epoch with the lowest validation loss after training is complete. For the validation procedure, 20% of the training data are reserved for the validation split. Finally, as a loss function, the categorical cross-entropy function is used.

During the experimentation, it was observed that, although the model's accuracy using all available features was high, a considerable number of these features did not significantly contribute to the predictive performance. Therefore, in order to enhance the training efficiency and model inference, a random selection of the most commonly utilized features was made. This resulted in a reduction of the required input to ten features while maintaining a high F1 Score (>95%). The selected features are enumerated in Table 2, where it is worth noting that, if a feature has a CS (Client–Server) and SC (Server–Client) identifier, it is because it has been measured in both directions. On the other hand, if a feature has only one identifier, it is because it has been measured in the direction indicated by the identifier type (CS or SC).

**Table 2.** Selected features of the Crypto dataset to train the baseline model.

| CS ID | SC ID | Name | Type | Description |
|---|---|---|---|---|
| 13 | 27 | SYN count | Numeric | Number of SYN segments observed (including rtx). |
| - | 90 | window scale | - | Scaling values negotiated [scale factor] |
| 70 | - | MSS | Bytes | MSS declared |
| 71 | 94 | max seg size | Bytes | Maximum segment size observed |
| 72 | 95 | min seg size | Bytes | Minimum segment size observed |
| 73 | 96 | win max | Bytes | Maximum receiver window announced (already scaled by the window scale factor) |
| 74 | 97 | win min | Bytes | Minimum receiver window announced (already scaled by the window scale factor) |
| 76 | 99 | cwin max | Bytes | Maximum in-flight-size computed as the difference between the largest sequence number so far, and the corresponding last ACK message on the reverse path. It is an estimate of the congestion window |
| 77 | 100 | cwin min | Bytes | Minimum in-flight-size |
| 78 | - | initial cwin | Bytes | First in-flight size, or total number of unack-ed bytes sent before receiving the first ACK segment |

CS: Client to server traffic. SC: Server to client traffic.

Additionally, to ensure that one variable did not dominate the results due to its scale, all data underwent standardization to achieve a mean of 0 and a standard deviation of 1. This standardization greatly improved the results obtained from the FCNN model.

### *5.3. Performance Evaluation of the Machine-Learning Model for Cryptomining Detection*

Once trained, the model was evaluated in an offline fashion. For this purpose, a test dataset representing 20% of a reserved portion of the total dataset that was never used for model training was first selected, and then inference was run on it. The performance of the model was then evaluated by comparing the predicted results with the actual labels in the data. The metrics used to measure the model performance include the well-known metrics of the Precision, Balanced Accuracy, F1 Score and Confusion Matrix. Balanced Precision was incorporated among the evaluation metrics to account for the imbalances that exist in the dataset. A brief explanation of the metrics used and other relevant definitions is provided in the following.

- **True Negative (TN)**: number of cases in which the model correctly predicted a negative outcome. The True Negative Rate (TNR) measures the rate of negative outcomes that were correctly predicted as negative.
- **False Positive (FP)**: the number of cases in which the model incorrectly predicted a positive outcome. The False Positive Rate (FPR) measures the rate of negative samples that were mislabelled as positives.

- **False Negative (FN)**: the number of cases in which the model incorrectly predicted a negative outcome. The False Negative Rate (FNR) measures the rate of positive samples that were mislabelled as negative.
- **True Positive (TP)**: the number of cases in which the model correctly predicted a positive outcome. The True Positive Rate (TPR) measures the rate of positive samples that were correctly labelled as positive.
- **Accuracy**: 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**: the Accuracy of the model in predicting both positive and negative classes. The formula is given by: Balanced Accuracy = (TP/P + TN/N)/2, where P is the total number of positive examples and N is the total number of negative examples.
- **Precision**: the True Positive Rate of all positive predictions made by the model. The formula is as follows: Precision = (TP)/(TP + FP).
- **Recall**: the true positive rate of all True Positive examples in the dataset. The formula is as follows: Recall = (TP)/(TP + FN).
- **F1 Score**: it is calculated by taking the harmonic mean of the Precision and Recall. The formula is given by: F1 Score = 2 × (Precision × Recall)/(Precision + Recall).
- **Confusion Matrix**: The Confusion Matrix (Figure 8) 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: 0.99998.
- Recall: 0.99543.
- F1 Score: 0.99541.

From the evaluation results, it can be seen that the FCNN model achieved excellent performance with an Accuracy of 0.99996, a Balanced Accuracy of 0.99543, a Precision of 0.99998, a Recall of 0.99543 and an F1 Score of 0.99541. This shows that the model is capable of accurately predicting the labels of the dataset with a high degree of accuracy.

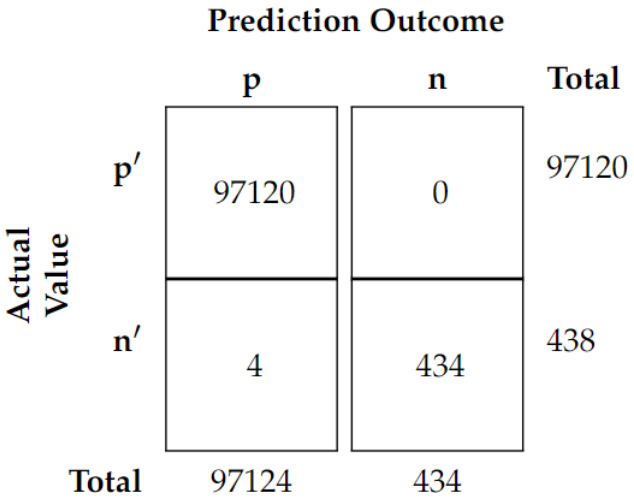



**Figure 8.** Confusion matrix showing the results of a classification model and comparing the actual values to the predicted outcomes. The table displays the number of True Positives, True Negatives, False Positives and False Negatives.

## 6. Energy Efficiency

In this section, an evaluation of the energy efficiency optimization of the DNN model deployed in the CAD component responsible for the cryptomining detection task is provided. We present a comparison of the energy efficiency achieved with a set of 11 optimization strategies that we designed combining different state-of-the-art techniques. Then, we discuss the energy efficiency trade-offs arising from the model optimization and identify the best performing approaches for the task at hand according to a variety of criteria considering different energy efficiency and accuracy requirements.

First, in Section 6.1 an analysis of the ways in which energy consumption is measured is performed. After that, Section 6.2 presents all of the different state-of-the-art optimization strategies that have been tested. Once the strategies have been specified, Section 6.3 provides some final details and parameters that were taken into account for the experimental setup. Finally, Section 6.4 offers a comprehensive analysis of the results and conclusions derived from the conducted experiments.

### *6.1. Measuring Energy Consumption*

After evaluating the different state-of-the-art optimization techniques that were available, optimization strategies that combine the different techniques to measure the energy consumption of the cryptomining detector deployed in the CAD component were designed. To measure the energy consumption of the DL-based cryptomining detector, this study leveraged the Performance Monitoring Counters (PMCs) due to their precise and granular measurements and their ability to measure energy consumption in real time and with low overhead [38].

For each combination of techniques to be applied, the baseline model that was analysed in Section 5.3 was trained, and then the optimization techniques were applied sequentially according to the order specified in the optimization strategy defined by the particular combination to be applied. In the inference stage, the prediction performance was evaluated using a batch size of 256 samples, which was determined to be the optimal batch size for the application in terms of latency and energy efficiency. In each inference test, we calculated the Accuracy and F1 Score as well as the Balanced Accuracy to account for the class imbalance that exists in our data. Once all the repetitions were performed, the metrics obtained at each time step among all the repetitions were aggregated using the mean, standard deviation and maximum value.

### *6.2. Selected Model-Optimization Strategies*

Several different optimization strategies were selected to offer combinations of the most promising state-of-the-art techniques [39,40]. Table 3 displays the different sets of strategies divided by post-processing quantization, training-aware model compression techniques and a combination of both of them.

**Table 3.** Energy efficiency optimization strategies.

| Set | Opt. Strategy Id. | Opt. Strategy |
|---|---|---|
| N/A | 0 | No optimizations (baseline) |
| Post-Training Optimization Techniques | 1 | Full 8-bit Integer (INT8) Weight Quantization [41–43] |
| | 2 | Half-Precision Floating-Point (FP16) Weight Quantization [42,44] |
| | 3 | Full Integer Weight Quantization with 16-bit Integer (INT16) Activations and 8-bit Integer (INT8) Weights [45] |
| Training-Aware Optimization Techniques | 4 | Pruning-Aware Model Fine-Tuning [46] |
| | 5 | Quantization-Aware Model Fine-Tuning [42,47] |
| | 6 | (1) Neural Architecture Search [48]<br>(2) Knowledge Distillation [49] |

**Table 3.** *Cont.*

| Set | Opt. Strategy Id. | Opt. Strategy |
|---|---|---|
| Combined Optimization Techniques | 7 | (1) Pruning-Aware Model Fine-Tuning [50]<br>(2) Quantization-Aware Model Fine-Tuning [50] |
| | 8 | (1) Neural Architecture Search<br>(2) Knowledge Distillation<br>(3) Pruning-Aware Model Fine-Tuning |
| | 9 | (1) Neural Architecture Search<br>(2) Knowledge Distillation<br>(3) Quantization-Aware Model Fine-Tuning |
| | 10 | (1) Neural Architecture Search<br>(2) Knowledge Distillation<br>(3) Pruning-Aware Model Fine-Tuning<br>(4) Quantization-Aware Model Fine-Tuning |
| | 11 | (1) Pruning-Aware Model Fine-Tuning<br>(2) Optimal Post-Training Quantization |
| | 12 | (1) Neural Architecture Search<br>(2) Knowledge Distillation<br>(3) Optimal Post-Training Quantization |
| | 13 | (1) Neural Architecture Search<br>(2) Knowledge Distillation<br>(3) Pruning-Aware Model Fine-Tuning<br>(4) Optimal Post-Training Quantization |

### *6.3. Experimental Setup*

An experimental evaluation in which all combinations of optimization techniques defined in Section 6.2 were applied to the cryptomining detector described in Section 5 was performed. In addition, the experiments were repeated five times with a 1 s time interval for sample measurements to collect energy efficiency metrics. In the experiments, a balanced profile was established, defining, as a performance threshold, a minimum acceptable reduction in energy consumption concerning the non-optimized model of 25% and a minimum Balanced Accuracy of 0.9. Furthermore, to apply the balanced profile, the ratio of these two factors was set to 0.5 for both to obtain the optimization strategy that leads to the most balanced results between the two objectives.

### *6.4. Analysis of the Results Obtained*

Figure 9 shows the percentage of the total average CPU power consumption obtained for each optimization strategy during the inference phase. The values represented were obtained from the aggregation of measured values collected during the duration of model inference at 1 s intervals and over five iterations for each optimization strategy using a batch size of 256 (medium size) to perform the prediction.

In summary, it can be observed that almost all optimization strategies lead to a significant reduction in energy consumption, exceeding, in most cases, the threshold of reduction in energy consumption with respect to the non-optimized model that was set at the beginning of the experimental evaluation. The knowledge distillation technique provided 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 only 0.08% in the Balanced Accuracy, 0.016% in the Accuracy and 0.11 in the F1 Score. Conversely, the optimization strategy that achieved the worst results for the percentage of total average CPU energy consumption reduction, was the full integer quantization.

Although it did not have a significant negative impact on the Balanced Accuracy, it caused a 97.14% increase in energy consumption, making it the only tested optimization strategy that increased the baseline inefficiency. Therefore, in our case, the use of the

knowledge distillation technique to optimize the DNN model implemented in the CAD component is the most recommended strategy among the ones evaluated, as it provides the highest energy savings and minimal performance degradation.

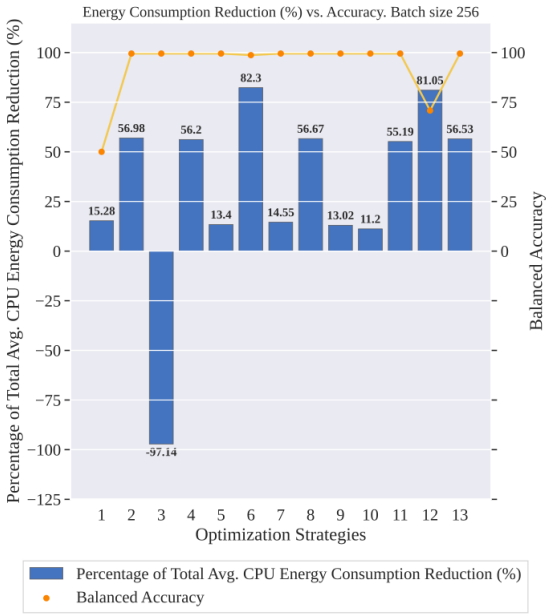



**Figure 9.** Energy consumption reduction obtained with each optimization strategy in the inference phase with respect to the non-optimized model using a batch size of 256.

## 7. Resilient Cyberthreat Detector against Adversarial Attacks

The technique of adversarial training was chosen to secure the Machine-Learning model against the recently appeared adversarial attacks that generate inputs know as adversarial examples (AEs). These inputs are specifically designed to deceive the Machine-Learning model and trigger incorrect predictions that benefit the attacker. The adversarial training approach is a technique employed in the field of Machine Learning, which entails retraining a model with adversarial examples.

The goal of this technique is to enhance the model's robustness and ability to defend against potential attacks by exposing it to various adversarial scenarios. By undergoing this process, the model can better adapt to the challenges presented by such attacks, thereby increasing its overall efficacy in combating them. Specifically, this work proposes to retrain the Machine-Learning model with high-quality adversarial examples to create a resilient classifier that can defend itself against adversarial attacks.

Therefore, to strengthen the TeraFlowSDN ML-based attack detectors against adversarial attacks, we designed a GAN-based solution to generate high-quality adversarial examples, which are very similar to real attack data but are able to fool the ML-based attack detector by misclassifying them. These high-quality AEs can be used later to retrain the TeraFlowSDN ML models and fortify the attack detectors against this type of sophisticated attack.

Our solution is inspired by the standard GAN architecture proposed in [51], which consists of two main components: the generator and the discriminator. In [15], it is shown that this architecture can be used to generate synthetic network traffic data that can fully replace real data in the training of ML models without significant performance loss. The generative model developed in this work is an extension to the MalGAN architecture [16]. In this design, the discriminator is used to model a third component, the unknown black-model target (e.g., the attacked TeraFlowSDN ML model). The discriminator in this specific setting is referred to as a substitute model, as it will attempt to learn the black-box behaviour.

As a consequence, this configuration implies a higher complexity in the training process compared with the standard GAN [52] as the behaviour of a given classifier that

will act as a black-box model during the training phase must also be tracked. Figure 10 shows an overview of the MalGAN architecture, in which each box contains an ML model that is producing predictions, and each circle contains input or output data.

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 attempt 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. Benign data represent the normal traffic transmitted on the network and malign data model the attack that will be manipulated to fool the ML (black-box) model into misclassifying it.

Unfortunately, experimental observations showed that, although the AEs generated by MalGAN achieved a very good ratio of misclassification when input to the black-box model (very close to a 100% evasion ratio), they were very different from real both malign and benign examples, which can favour their detection by using simple statistical filters (e.g., based on the mean of the real benign and malign data distributions).

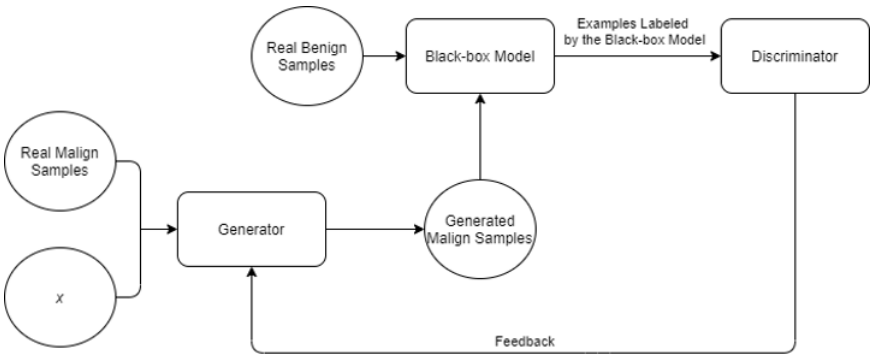



**Figure 10.** Overview of the enhanced GAN solution based on MalGAN.

As a novelty, our enhanced version of the MalGAN architecture uses a custom activation function based on the Smirnov Transform (ST) [18] as the last layer of the generator to help to generate AEs that mimic the statistical behaviour of real malign examples, thus, transforming the generator output variables into variables that, from a statistical perspective, are distributed exactly the same as the input variables. Our proposal is related to a key problem with GANs: typically, without further tuning, the output distribution of each of the random variables obtained in the generator output is approximately normal.

This is related to the mode-collapse problem, a well-reported behaviour of the GANs. To address this problem, the job of the generator is facilitated by using, as an activation function of each output variable, a customized function that is able to capture the statistical subtleties of each variable of the malign data. Each customized function implements the inverse of the Smirnov Transform of each malign data variable. This transformation converts random vectors with normal marginal distributions (the output of a normal GAN) into random vectors with approximately the marginal distribution of the malign data variable.

In addition, the ST activation function is fully deterministic and differentiable, which allows it to be seamlessly integrated into the backpropagation step during the GAN training processes. In [18], the experimental results demonstrated the significant improvement provided by this custom activation function when applied in GAN architectures in terms of the quality of the generated samples.

The GAN was trained with the same datasets used in Section 5.1 and previously described in [32]. The generator and discriminator networks were defined as three-layer FCNNs (Fully Connected Neural Networks) assuming a moderate complexity in the black-box model. In the case that the black-box model is supposed to be more complex, more layers and neurons can be added to the generator and discriminator. The details of the training process and hyperparameters used are similar to those described in [18].

In Figure 11, the results obtained for the Vainilla MalGAN are compared to our proposal (MalGAN equipped with ST activation functions). The top row shows the Vainilla MalGAN distances (Figure 11a) and evasion ratios (Figure 11b) at each epoch, and the bottom row plots the distances (Figure 11c) and evasion ratios (Figure 11d) for a MalGAN equipped with ST activation functions.

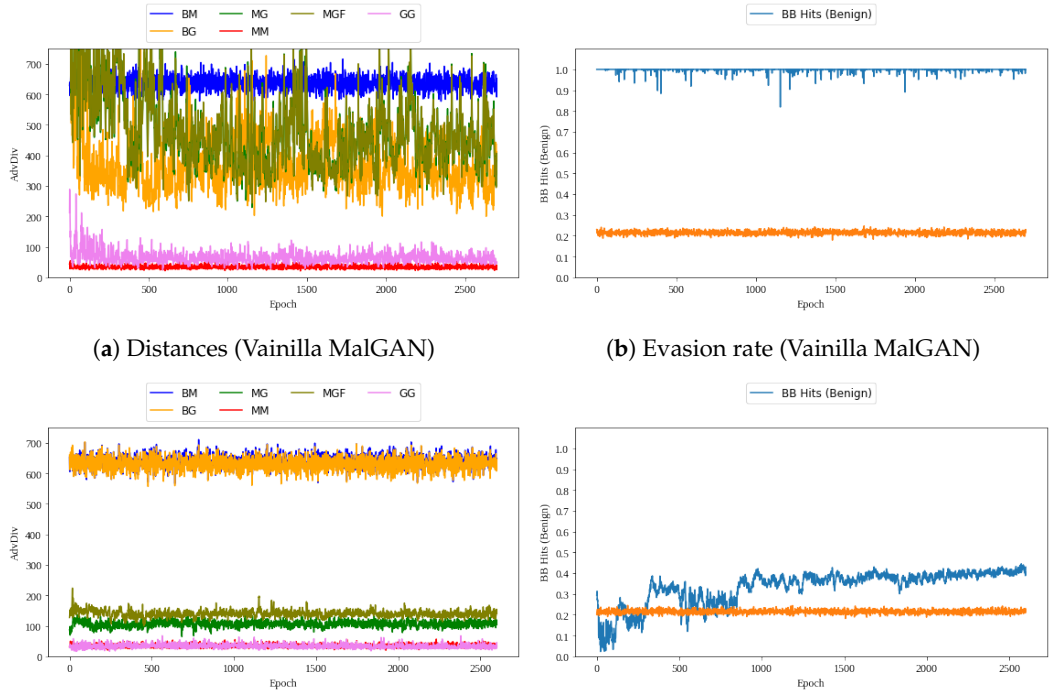
It can be seen that, although the evasion ratios of the Vainilla MalGAN are roughly 1.0, this architecture completely fails to generate synthetic adversarial examples that are close to the malign data and far from the benign data since its distances between (i) malign and generated malign data (MG), (ii) malign and generated malign data that fool the black-box model (MGF) and (iii) benign and generated malign data (BG) are very far from the expected: BG should be similar to the distance between benign and malign data (BM) and MG and MGF should be small and close to the distance between two samples of malign data (MM).

This is a clear symptom that the generator is producing adversarial (synthetic) examples that are very different from the real malign examples, and therefore, they could be identified in a real environment using a simple statistical filter. Note that, to effectively fortify an ML model against these types of attack, AEs should be virtually indistinguishable from the real malign data. It is worth noting that the Vainilla MalGAN training process was slightly modified to avoid generating synthetic data that were very far from the real data by substituting the black-box labels that were added to the synthetic adversarial examples by their real labels.

However, as can be seen in Figure 11a, the distances of the generated synthetic data with respect to benign and malign data still did not achieve the expected good behaviour as they are far from both the benign and malign data. In sharp contrast, our proposal (MalGAN equipped with ST activation functions) generated adversarial examples that are close to the real malign data (Figure 11c), as (i) the MG and MGF distances are small and close to MM, and (ii) the BG distances are very similar to BM. The trade-off of this solution is that the evasion ratios (the blue line in Figure 11d) are not as good as those obtained with the Vainilla MalGAN but are at least greater than the ratio of misclassified malign data (the orange line in Figure 11d).

Finally, after high-quality adversarial examples were 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. In order to test the degree of resilience of the retrained ML model, we reserved a dataset of malign data that was not used for training the GAN. The reserved data had not been seen by the GAN during its training and can, therefore, be considered as data similar to what could appear in a real scenario.

By using examples from this dataset along with Gaussian noise vectors, we generated synthetic samples that were statistically very similar to an attack generated by a malicious attacker. To measure the degree of resilience that the ML model offers in real time, we counted the synthetic samples that managed to deceive the new version of the ML model that was strengthened with our adversarial examples. This approach enabled us to assess the robustness of the model to adversarial attacks in a real-world scenario. The end result in our scenario was a retrained ML model in which the Accuracy in detecting new AEs generated with different MalGANs increased to 99% (i.e., the evasion ratio decreased from the original 48% to 1%).




(**a**) Distances (Vainilla MalGAN) (**b**) Evasion rate (Vainilla MalGAN)

(**c**) Distances (MalGAN with ST activation) (**d**) Evasion rate (MalGAN with ST activation)

**Figure 11.** (**Left** column) Distances between samples of real and synthetic data distributions: BM (benign and malign data), MG (malign and generated malign data), MGF (malign and generated malign data that fool the black-box model), BG (benign and generated malign data), GG (two samples of generated malign data) and MM (two samples of malign data). (**Right** column) Evasion ratios: (blue) generated malign examples (AEs) and (orange) real malign examples that are classified as benign by the black-box model. In all figures, the x-axis represents the GAN training epochs.

## 8. Conclusions

In this work, a proposal for a standardized and distributed approach to cyber-attack detection and mitigation in the context of the TeraFlowSDN controller was presented. TeraFlowSDN is an open-source, next-generation, cloud-native Software-Defined Networking (SDN) controller that has been specifically designed to support the evolving requirements of 5G and beyond networks. As TeraFlowSDN is the European Telecommunications Standards Institute reference implementation for SDN controllers, the solution proposed in this article can serve as a reference framework for future developments of cybersecurity solutions within commercial SDN controllers.

First, this study proposed a novel distributed component architecture based on Machine Learning (ML) and Deep Learning (DL) within the TeraFlowSDN controller to implement scalable attack detectors in the data and control plane. To this end, a set of ML-based cybersecurity components were integrated into the microservice-based TeraFlowSDN architecture. The integration of these cybersecurity components was exemplified with an end-to-end security analysis and mitigation process based on the detection of cryptomining activity, an emerging attack vector that is becoming increasingly common in today's telecommunication networks.

The proposed solution consists of two centralized components, the Centralized Attack Detector and the Attack Mitigator, as well as a distributed component, the Distributed Attack Detector, placed at the edge of the network. Although a specific attack model (cryptomining) was used to demonstrate the integration of the cybersecurity components in TeraFlowSDN, the proposed architecture based on microservices and the adoption of standard interfaces and protocols (Protocol Buffers and gRPC) commonly used in the telco industry will allow for the seamless integration of new types of attack detectors into the TeraFlowSDN controller.

Second, the energy efficiency of the proposed architecture is deemed crucial in our design considerations as it is expected to become a major limiting factor for the deployment of new services. To address this challenge, this work proposed a systematic process based on state-of-the-art optimization techniques that were combined in a set of 11 optimization strategies that can be applied to reduce the energy consumption of the DL-based attack detection models used in the TeraFlowSDN Centralized Attack Detector component. Various optimization strategies were evaluated, achieving energy consumption reduction of up to 83.30% with a minimal performance degradation of 0.08% in Balanced Accuracy. It is worth noting that the proposed method is general enough to be applied in the same way to any DL-based model before being deployed in the TeraFlowSDN controller.

Last but not least, to strengthen TeraflowSDN ML models against sophisticated adversarial attacks that can mislead them into making the wrong decisions and allow malicious traffic to bypass the security system, this study proposed a technique to add resilience to ML models based on retraining the models with high-quality adversarial examples. To produce these high-quality adversarial examples, MalGAN (an adversarial example generator based on Generative Adversarial Networks for highly restrictive black-box attack scenarios) was extended by adding, to the generator network, a recently proposed activation function based on the Smirnov transformation. This improvement allowed us to generate high-quality adversarial examples that were used to retrain and fortify the ML model deployed on the TeraFlowSDN controller. After retraining, the resilient ML model reduced its original evasion ratio from 0.50 to 0.01.

## 9. Future Work

Given the growing demand for more efficient and sustainable computing systems, future work should focus on addressing the concern of energy consumption in deep neural networks by exploring the impact of different model architectures and varying numbers of parameters in the existing trade-off between the performance and energy consumption of the model as well as the energy efficiency that can be achieved with the different optimization techniques presented in this article.

Furthermore, in upcoming research, it could be interesting to investigate the interplay between energy efficiency and model resiliency in the development of a comprehensive model that exhibits both energy efficiency and resiliency towards adversarial attacks. Further studies will be necessary to understand the relationship between these two important factors, such as whether optimizing the energy consumption negatively impacts the model's ability to withstand attacks and whether enhancing the resiliency requires increased energy consumption. Additionally, alternative approaches should be investigated for combining these properties for optimal performance.

The proposed Attack Mitigator component is slated for further development and enhancement with the integration of advanced functions to enable autonomous network-based decision making and the deployment of tailored mitigation strategies based on the confidence level of the attack and the perceived threat level. These complex functions can be considered as Zero Touch Provisioning actions that can be triggered in response to a specific detected cyberthreat.

**Author Contributions:** Conceptualization, A.M., A.K., L.d.l.C., S.G.-C., A.P. and L.G.; methodology, A.M., A.K., L.d.l.C., S.G.-C., A.P. and L.G.; software, A.M., A.K., L.d.l.C., S.G.-C., A.P. and L.G.; validation, A.M., A.K., L.d.l.C., S.G.-C., A.P. and L.G.; formal analysis, A.M., A.K., L.d.l.C., S.G.-C., A.P. and L.G.; investigation, A.M., A.K., L.d.l.C., S.G.-C., A.P. and L.G.; resources, A.M., A.K., L.d.l.C., S.G.-C., A.P. and L.G.; data curation, A.M., A.K., L.d.l.C., S.G.-C., A.P. and L.G.; writing—original draft preparation, A.M., A.K., L.d.l.C., S.G.-C., A.P. and L.G.; writing—review and editing, A.M., A.K., L.d.l.C., S.G.-C., A.P. and L.G.; visualization, A.M., A.K., L.d.l.C., S.G.-C., A.P. and L.G.; supervision, A.M., A.K., L.d.l.C., S.G.-C., A.P. and L.G.; project administration, A.M., A.K., L.d.l.C., S.G.-C., A.P. and L.G.; funding acquisition, A.M., A.K., L.d.l.C., S.G.-C., A.P. and L.G. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work was partially supported by the European Union's Horizon 2020 Research and Innovation Programme under Grant 101015857 (TeraFlow) and Horizon Europe SNS R&I Work Programme under Grant 101097122 (ACROSS).

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** Not applicable.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Dargahi, T.; Caponi, A.; Ambrosin, M.; Bianchi, G.; Conti, M. A survey on the security of stateful SDN data planes. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 1701–1725. [CrossRef]
2. Vilalta, R.; Munoz, R.; Casellas, R.; Martínez, R.; López, V.; de Dios, O.G.; Pastor, A.; Katsikas, G.P.; Klaedtke, F.; Monti, P.; et al. Teraflow: Secured autonomic traffic management for a tera of sdn flows. In Proceedings of the 2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit), Porto, Portugal, 8–11 June 2021; pp. 377–382.
3. Dahmen-Lhuissier, S. TFS. Available online: https://www.etsi.org/committee/2064-tfs (accessed on 5 April 2023).
4. Lal, S.; Taleb, T.; Dutta, A. NFV: Security threats and best practices. *IEEE Commun. Mag.* **2017**, *55*, 211–217. [CrossRef]
5. Xing, T.; Xiong, Z.; Huang, D.; Medhi, D. SDNIPS: Enabling Software-Defined Networking based intrusion prevention system in clouds. In Proceedings of the tenth International Conference on Network and Service Management (CNSM) and Workshop, Rio de Janeiro, Brazil, 17–21 November 2014; pp. 308–311. [CrossRef]
6. Chung, C.J.; Khatkar, P.; Xing, T.; Lee, J.; Huang, D. NICE: Network Intrusion Detection and Countermeasure Selection in Virtual Network Systems. *IEEE Trans. Dependable Secur. Comput.* **2013**, *10*, 198–211. [CrossRef]
7. Mozo, A.; Pastor, A.; Karamchandani, A.; de la Cal, L.; Rivera, D.; Moreno, J.I. Integration of Machine Learning-Based Attack Detectors into Defensive Exercises of a 5G Cyber Range. *Appl. Sci.* **2022**, *12*, 10349. [CrossRef]
8. Alzahrani, A.O.; Alenazi, M.J.F. Designing a Network Intrusion Detection System Based on Machine Learning for Software Defined Networks. *Future Internet* **2021**, *13*, 111.
9. McHugh, J. Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as Performed by Lincoln Laboratory. *ACM Trans. Inf. Syst. Secur.* **2000**, *3*, 262–294. [CrossRef]
10. Radoglou–Grammatikis, P.; Rompolos, K.; Sarigiannidis, P.; Argyriou, V.; Lagkas, T.; Sarigiannidis, A.; Goudos, S.; Wan, S. Modeling, Detecting, and Mitigating Threats Against Industrial Healthcare Systems: A Combined Software Defined Networking and Reinforcement Learning Approach. *IEEE Trans. Ind. Inform.* **2022**, *18*, 2041–2052. [CrossRef]
11. Zhou, X.; Liang, W.; Li, W.; Yan, K.; Shimizu, S.; Kevin, I.; Wang, K. Hierarchical adversarial attacks against graph-neural-network-based IoT network intrusion detection system. *IEEE Internet Things J.* **2021**, *9*, 9310–9319. [CrossRef]
12. Aiken, J.; Scott-Hayward, S. Investigating Adversarial Attacks against Network Intrusion Detection Systems in SDNs. In Proceedings of the 2019 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Dallas, TX, USA, 12–14 November 2019; pp. 1–7. [CrossRef]
13. Zhang, C.; Patras, P.; Haddadi, H. Deep learning in mobile and wireless networking: A survey. *IEEE Commun. Surv. Tutor.* **2019**, *21*, 2224–2287. [CrossRef]
14. Schwartz, R.; Dodge, J.; Smith, N.A.; Etzioni, O. Green AI. *Commun. ACM* **2020**, *63*, 54–63. [CrossRef]
15. Mozo, A.; González-Prieto, Á.; Pastor, A.; Gómez-Canaval, S.; Talavera, E. Synthetic flow-based cryptomining attack generation through Generative Adversarial Networks. *Sci. Rep.* **2022**, *12*, 2091. [CrossRef] [PubMed]
16. Hu, W.; Tan, Y. Generating adversarial malware examples for black-box attacks based on GAN. In Proceedings of the Data Mining and Big Data: Seventh International Conference, DMBD 2022, Beijing, China, 21–24 November 2022; Part II; Springer: Berlin/Heidelberg, Germany, 2023; pp. 409–423.
17. Xiao, C.; Li, B.; Zhu, J.Y.; He, W.; Liu, M.; Song, D. Generating adversarial examples with adversarial networks. *arXiv* **2018**, arXiv:1801.02610.
18. González-Prieto, Á.; Mozo, A.; Gómez-Canaval, S.; Talavera, E. Improving the quality of generative models through Smirnov transformation. *Inf. Sci.* **2022**, *609*, 1539–1566. [CrossRef]
19. McKeown, N.; Anderson, T.; Balakrishnan, H.; Parulkar, G.; Peterson, L.; Rexford, J.; Shenker, S.; Turner, J. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.* **2008**, *38*, 69–74. [CrossRef]
20. Xie, J.; Yu, F.R.; Huang, T.; Xie, R.; Liu, J.; Wang, C.; Liu, Y. A Survey of Machine Learning Techniques Applied to Software Defined Networking (SDN): Research Issues and Challenges. *IEEE Commun. Surv. Tutor.* **2019**, *21*, 393–430. [CrossRef]
21. Perera Jayasuriya Kuranage, M.; Piamrat, K.; Hamma, S. Network Traffic Classification Using Machine Learning for Software Defined Networks. In *Machine Learning for Networking, Proceedings of the International Conference on Machine Learning for Networking, MLN 2019, Paris, France, 3–5 December 2019*; Boumerdassi, S., Renault, E., Mühlethaler, P., Eds.; Lecture Notes in Computer Science; Part II; Springer: Berlin/Heidelberg, Germany, 2020; pp. 28–39. [CrossRef]

22. Prabhavat, S.; Thongthavorn, T.; Pasupa, K. Deep Learning-Based Early Detection and Avoidance of Traffic Congestion in Software-Defined Networks. In Proceedings of the 2022 14th International Conference on Information Technology and Electrical Engineering (ICITEE), Yogyakarta, Indonesia, 18–19 October 2022; pp. 1–6.
23. Secci, S.; Diamanti, A.; Vilchez, J.M.S.; Bah, M.T.; Vizzarreta, P.; Machuca, C.M.; Scott-Hayward, S.; Smith, D. Security and Performance Comparison of ONOS and ODL Controllers. Ph.D. Thesis, Open Networking Foundation Informational Report, Palo Alto, CA, USA, 2019.
24. Medved, J.; Varga, R.; Tkacik, A.; Gray, K. OpenDaylight: Towards a Model-Driven SDN Controller architecture. In Proceedings of the IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014, Sydney, Australia, 19 June 2014; pp. 1–6. [CrossRef]
25. Berde, P.; Gerola, M.; Hart, J.; Higuchi, Y.; Kobayashi, M.; Koide, T.; Lantz, B.; O'Connor, B.; Radoslavov, P.; Snow, W.; et al. ONOS: Towards an Open, Distributed SDN OS. In Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, Chicago, IL, USA, 22 August 2014; pp. 1–6. [CrossRef]
26. Braga, R.; Mota, E.; Passito, A. Lightweight DDoS flooding attack detection using NOX/OpenFlow. In Proceedings of the IEEE Local Computer Network Conference, Denver, CO, USA, 10–14 October 2010; pp. 408–415. [CrossRef]
27. Lin, H. SDN-based In-network Honeypot: Preemptively Disrupt and Mislead Attacks in IoT Networks. *arXiv* **2019**, arXiv:1905.13254.
28. Kamel, H.; Abdullah, M.Z. Distributed denial of service attacks detection for software defined networks based on evolutionary decision tree model. *Bull. Electr. Eng. Inform.* **2022**, *11*, 2322–2330. [CrossRef]
29. Makuvaza, A.; Jat, D.S.; Gamundani, A.M. Deep neural network (DNN) solution for real-time detection of distributed denial of service (DDoS) attacks in software defined networks (SDNs). *SN Comput. Sci.* **2021**, *2*, 1–10. [CrossRef]
30. Alzahrani, A.O.; Alenazi, M.J.F. ML-IDSDN: Machine learning based intrusion detection system for software-defined network. *Concurr. Comput. Pract. Exp.* **2023**, *35*, e7438. [CrossRef]
31. Secured Autonomic Traffic Management for a Tera of SDN Flows. Deliverable 5.2, Implementation of Pilots and First Evaluation. Project H2020 Teraflow. Available online : https://www.teraflow-h2020.eu/ (accessed on 5 April 2023).
32. Pastor, A.; Mozo, A.; Vakaruk, S.; Canavese, D.; López, D.R.; Regano, L.; Gómez-Canaval, S.; Lioy, A. Detection of encrypted cryptomining malware connections with machine and deep learning. *IEEE Access* **2020**, *8*, 158036–158055. [CrossRef]
33. Secured Autonomic Traffic Management for a Tera of SDN Flows. Deliverable 2.2, Final Requirements, Architecture Design, Business Models, and Data Models. Project H2020 Teraflow. Available online: https://www.teraflow-h2020.eu/ (accessed on 5 April 2023).
34. Mellia, M.; Carpani, A.; Lo Cigno, R. TStat: TCP STatistic and Analysis Tool. In *Quality of Service in Multiservice IP Networks*; Marsan, M.A., Corazza, G., Listanti, M., Roveri, A., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2003; pp. 145–157. [CrossRef]
35. Dridi, L.; Zhani, M.F. SDN-Guard: DoS Attacks Mitigation in SDN Networks. In Proceedings of the 2016 fifth IEEE International Conference on Cloud Networking (Cloudnet), Pisa, Italy, 3–5 October 2016; pp. 212–217. [CrossRef]
36. Pastor, A.; Mozo, A.; Lopez, D.R.; Folgueira, J.; Kapodistria, A. The Mouseworld, a security traffic analysis lab based on NFV/SDN. In Proceedings of the 13th International Conference on Availability, Reliability and Security, Hamburg, Germany, 27–30 August 2018; pp. 1–6.
37. Mozo, A.; Karamchandani, A.; Gómez-Canaval, S.; Sanz, M.; Moreno, J.I.; Pastor, A. B5GEMINI: AI-driven network digital twin. *Sensors* **2022**, *22*, 4106. [CrossRef]
38. García-Martín, E.; Lavesson, N.; Grahn, H.; Casalicchio, E.; Boeva, V. How to Measure Energy Consumption in Machine Learning Algorithms. In Proceedings of the ECML PKDD 2018 Workshops, Dublin, Ireland, 10–14 September 2018; Alzate, C., Monreale, A., Assem, H., Bifet, A., Buda, T.S., Caglayan, B., Drury, B., García-Martín, E., Gavaldà, R., Koprinska, I., et al., Eds.; Lecture Notes in Computer Science; Springer International Publishing: Cham, Switzerland, 2019; pp. 243–255. [CrossRef]
39. Guo, Y. A Survey on Methods and Theories of Quantized Neural Networks. *arXiv* **2018**, arXiv:1808.04752.
40. Gou, J.; Yu, B.; Maybank, S.J.; Tao, D. Knowledge Distillation: A Survey. *Int. J. Comput. Vis.* **2021**, *129*, 1789–1819.
41. Jacob, B.; Kligys, S.; Chen, B.; Zhu, M.; Tang, M.; Howard, A.; Adam, H.; Kalenichenko, D. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 2704–2713. [CrossRef]
42. Novac, P.E.; Boukli Hacene, G.; Pegatoquet, A.; Miramond, B.; Gripon, V. Quantization and Deployment of Deep Neural Networks on Microcontrollers. *Sensors* **2021**, *21*, 2984. [CrossRef] [PubMed]
43. Post-Training Integer Quantization | TensorFlow Lite. Available online: https://www.tensorflow.org/lite/performance/post_training_integer_quant (accessed on 5 April 2023).
44. Post-Training Float16 Quantization | TensorFlow Lite. Available online: https://www.tensorflow.org/lite/performance/post_training_float16_quant (accessed on 5 April 2023).
45. Post-Training Integer Quantization with Int16 Activations | TensorFlow Lite. Available online: https://www.tensorflow.org/lite/performance/post_training_integer_quant_16x8 (accessed on 5 April 2023).
46. Pruning Comprehensive Guide | TensorFlow Model Optimization. Available online: https://www.tensorflow.org/model_optimization/guide/pruning/comprehensive_guide (accessed on 5 April 2023).

47. Quantization Aware Training Comprehensive Guide | TensorFlow Model Optimization. Available online: https://www.tensorflow.org/model_optimization/guide/quantization/training_comprehensive_guide (accessed on 5 April 2023).
48. Elsken, T.; Metzen, J.H.; Hutter, F. Neural Architecture Search: A Survey. *arXiv* **2019**, arXiv:1808.05377.
49. Hinton, G.; Vinyals, O.; Dean, J. Distilling the Knowledge in a Neural Network. *arXiv* **2015**, arXiv:1503.02531.
50. Pruning Preserving Quantization Aware Training (PQAT) Keras Example | TensorFlow Model Optimization. Available online: https://www.tensorflow.org/model_optimization/guide/combine/pqat_example (accessed on 5 April 2023).
51. Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. Generative adversarial networks. *Commun. ACM* **2020**, *63*, 139–144. [CrossRef]
52. González-Prieto, Á.; Mozo, A.; Talavera, E.; Gómez-Canaval, S. Dynamics of fourier modes in torus generative adversarial networks. *Mathematics* **2021**, *9*, 325. [CrossRef]