Journal of Internet Services and Applications, 2024, 15:1, doi: 10.5753/jisa.2024.3835

Design and implementation of intelligent packet filtering in IoT microcontroller-based devices

Gustavo de Carvalho Bertoli [**Instituto Tecnológico de Aeronática** | ***bertoli@ita.br***]
Gabriel Victor C. Fernandes [**Universidade de São Paulo** | ***gabriel_victor@usp.br***]
Pedro H. Borges Monici [**Universidade de São Paulo** | ***pedroh.monici@usp.br***]
César H. de Araujo Guibo [**Universidade de São Paulo** | ***cesarguibo@usp.br***]
Aldri Luiz dos Santos [**Universidade Federal de Minas Gerais** | ***aldri@dcc.ufmg.br***]
Lourenço Alves Pereira Junior [**Instituto Tecnológico de Aeronática** | ***ljr@ita.br***]

Divisão de Ciência da Computação, Instituto Tecnológico de Aeronáutica (ITA), São Jose dos Campos, SP, Brazil.

Received: 9 November 2023 • **Accepted:** 24 April 2024 • **Published:** 27 August 2024

Abstract Internet of Things (IoT) devices are increasingly pervasive and essential in enabling new applications and services. However, their widespread use also exposes them to exploitable vulnerabilities and flaws that can lead to significant losses. In this context, ensuring robust cybersecurity measures is essential to protect IoT devices from malicious attacks. However, the current solutions that provide flexible policy specifications and higher security levels for IoT devices are scarce. To address this gap, we introduce T800, a low-resource packet filter that utilizes machine learning (ML) algorithms to classify packets in IoT devices. We present a detailed performance benchmarking framework and demonstrate T800's effectiveness on the ESP32 system-on-chip microcontroller and ESP-IDF framework. Our evaluation shows that T800 is an efficient solution that increases device computational capacity by excluding unsolicited malicious traffic from the processing pipeline. Additionally, T800 is adaptable to different systems and provides a well-documented performance evaluation strategy for security ML-based mechanisms on ESP32-based IoT systems. Our research contributes to improving the cybersecurity of resource-constrained IoT devices and provides a scalable, efficient solution that can be used to enhance the security of IoT systems.

Keywords: Packet filtering, intrusion detection, internet of things, constrained devices.

1 Introduction

Cybersecurity is increasingly becoming a vital strategic aspect of business continuity [Klint, 2021]. According to the World Economic Forum's 2021 Global Risk Report [McLennan, 2021], incidents of this nature represent one of the most significant post-pandemic challenges. They potentially cause economic disruption, financial losses, geopolitical tensions, and social unrest. Thus, it is important to highlight that cybersecurity should be essential to the product and service development lifecycle. Due to the digital transformation that the world is going through, incidents have appeared in media due to the possible disruption of services by cyber-attacks that affect the population directly, like urban computing systems, impacting intelligent city services and directly affecting urban populations [Kovacs, 2020; Smith, 2022]. The vulnerabilities in Internet of Things (IoT) devices, integral to urban infrastructures, exemplify this challenge, underscoring the need for robust cybersecurity mechanisms in the urban computing landscape [Almeida, 2023].

High critical attacks can be achieved in this constantly changing environment by orchestrating large-scale compromised devices for malicious purposes. In this sense, a typical approach consists of a set of computational resources composing a command and control network (i.e., *botnets*) [Bertino and Islam, 2017; Lakshmanan, 2022]. Then, the attacker's objective is to compromise computers, smartphones, Wi-Fi routers, IP cameras, and others to compose this botnet. In this context, Internet of Things (IoT) devices are a common target because they usually have a precarious update, improper configuration, and maintenance procedures, as seen in attack campaigns like Mirai and Mozi [Antonakakis *et al*., 2017; McMillen, 2021]. Moreover, this botnet risk tends to be present if the development practices do not ensure robust updating of security policies, intelligent security mechanisms, and secure-by-design. Additionally, it is common sense that IoT enables new technological, efficient, and profitable solutions. However, subverting IoT systems is profitable also to malicious actors [Al-Sarawi *et al*., 2020; Georgoulias *et al*., 2023].

Nowadays, cybersecurity mechanisms focusing on resource-constrained devices are scarce. Specifically, there is a lack of work evaluating security mechanisms using machine learning for microcontroller-based IoT systems. Our work contributes to this research gap by proposing T800: a packet filter for Internet of Things (IoT) devices. T800 is a combination of mechanism and policy to implement a more secure operating environment for IoT systems, as it can work as an enabler to implement zero-trust architectures. The mechanism consists of instrumentation of the ESP-IDF framework TPC/IP stack, the lightweight IP (lwIP). It allows intercepting the network ingress traffic and deciding whether to drop the current packet. The policies are the translation of machine-learning algorithms trained to identify malicious packets in the incoming network traffic. Hence, the mechanism allows the introduction of an

ML-based filter that returns boolean values to allow packets to proceed in the network stack. As contributions of this paper, we highlight:

- Design and implementation of T800 allowing the deployment of rules for packet filtering with machine learning algorithms such as decision trees and neural networks in the ESP32 platform (FreeRTOS, TCP/IP protocol stack lwIP, and ESP-IDF SDK) with TensorFlow.
- Considering the port scanning use-case in this paper, a solution to allow IoT devices to go unnoticed during lateral movement or internet-wide scanning campaigns.
- Performance evaluation of T800, indicating low overhead and energy efficiency for low-end edge devices. The results demonstrate a reduction in resource consumption by removing unsolicited traffic from the protocol stack and avoiding processing them.

To our knowledge, this work is the first to evaluate the technical feasibility of machine learning-based stateless intrusion detection on microcontroller-based systems. Furthermore, the work is reproducible and opens research for further developments in network security mechanisms for microcontroller-based systems. Our results advance knowledge in packet filtering for security purposes and have important implications for microcontroller-based systems and the IoT domain.

The remainder of the paper is organized as follows: Section 2 discusses related works. Section 3 describes the architecture of T800. Section 4 describes the experimental design. Section 5 reports the obtained results and the influence of the factors. Finally, Section 6 presents the conclusion of the work and future directions.

2 Related Works

As observed in a broader application domain of computer networks and security, the characterization, identification, and creation of rules for packet filtering are well established [Roesch, 1999; Khraisat and Alazab, 2021; Alieyan *et al*., 2021]. However, viewing them as componentized network functions to move them from the perimeter towards the endpoint of low computational power presents difficulties and restrictions [Qin *et al*., 2019]. Saha *et al*. [2022] performs a survey on the deployment of ML on low-resource devices. They highlight the importance of moving the intelligence to the edge to achieve independence from the network infrastructure, and low deployment costs.

The TinyML is a broader concept focusing on advancements to complex deep learning layers deployment to constrained devices and the on-device training approach [Ren *et al*., 2021; Schizas *et al*., 2022]. TinyML improves energy efficiency, low cost, data integrity, privacy, security, and latency. Abadade *et al*. [2023] presents a taxonomy of TinyML applications, but security-related applications, as proposed in this work for packet filtering, is not identified.

Among the works exploring the deployment of machine learning in constrained environments, Murshed *et al*. [2019] provides a survey on applying machine learning at the network edge. It addresses operational challenges and the benefits of such deployments. The survey underscores the need for edge-computing solutions catering to IoT environments' unique demands. This includes managing the challenge of network security within these constrained devices. Our work directly contributes to this by proposing a novel approach for deploying dynamic, updateable ML-based intrusion detection systems designed explicitly for microcontrollers. By focusing on the network security aspect within the IoT edge computing context, we provide a solution that enhances security while maintaining the operational benefits of edge deployments highlighted by Murshed *et al*. [2019].

Furthermore, Viegas *et al*. [2021] showed that ML-based solutions for network security have a limited lifetime, meaning that there is evidence that attacks have time-varying behavior. The findings indicate a framed life of 6 (six) weeks, with an acceptable accuracy of 2 (two) to 8 (eight) weeks. Therefore, solutions that allow updating security policies in response to incidents are part of the requirements for sound performance.

Low-power devices constitute an essential part of the industrial scenario [Niedermaier *et al*., 2019]. The reference shows an architecture to carry out the attack detection process in a distributed way. The focus is on the context of SCADA and PLC systems. As described, there is a precise characterization of the periodic communication behavior between the devices. Therefore, it allows the identification of anomalies. However, the study needs more generality, requiring an effort to implement new rules and apply them in other contexts.

In Manocchio *et al*. [2022], the authors implement a microcontroller-based network intrusion detection system. They evaluate three different microcontroller architectures for their proposition. Their focus, however, consisted of loading the trained model on the resource-constrained device and sending network flow-based samples through a serial interface for only inference on the devices. Thus, this still needs to be a real security mechanism on a microcontroller-based system once an actual system is required to process the packet since its reception in the network interface.

Targeting a single-board computer platform, Soe *et al*. [2020] presents a solution for detecting malicious activities and provides results that indicate low computational demand for execution. Even though it is a flexible solution, the generalization of the filtering method and an update mechanism was outside the scope of the work. However, means for adapting to new attacks, revoking access, and incorporating new devices are out of their scope. In Jan *et al*. [2019], the authors implement a system focused on IoT devices using SVM (*Support Vector Machine*) as an identification model. Even though the results indicate reproducibility, the data source consists of a conventional dataset without considering the implementation in an IoT device, with all conclusions based on MATLAB analysis. Filus *et al*. [2021] presents neural networks as a potential approach. However, it does not discuss the need for a feasible update mechanism to tackle the evolutionary nature of network attacks. It neither implements nor evaluates its proposition on resource-constrained devices.

In Mudgerikar *et al*. [2019], IoT devices are considered

specific-purpose devices, where deviations from their primary periodic function, called profiles, can indicate an intrusion or malicious activity. The authors propose profiling IoT devices by analyzing running processes and system calls. Their approach aims to apply to various architectures and Linux distributions, but it is unsuitable for microcontroller-based systems. Furthermore, their method relies on an external server to process all device logs, whereas our approach suggests processing and decision-making at the edge. Moreover, their work primarily focuses on IoT devices and does not address the scope of microcontroller-based systems, which typically lack system logs and SysMon for log generation. Regarding attack detection, their system-level approach identifies attacks only after the device has been compromised, while our network traffic-based approach allows for early attack prevention. The authors provide a qualitative analysis of CPU and memory usage for device and server applications. Still, it lacks precise details about the specific hardware used, as recommended by Apruzzese *et al*. [2023], making it difficult for more comparison on runtime performance evaluations.

Mirsky *et al*. [2018]; Eskandari *et al*. [2020]; Hafeez *et al*. [2020] presents an anomaly-based intrusion detection system. They use real traffic generated by testbeds composed of IoT devices/sensors to evaluate their proposition. However, in contrast to our approach, these propositions consider the intrusion detection system part of a gateway instead of deploying the solution in the node device. In all cases, no update mechanism is discussed, and their runtime performance evaluation on a single-board computer considers only network throughput for Mirsky *et al*. [2018]. Eskandari *et al*. [2020]; Hafeez *et al*. [2020] evaluates metrics such as CPU, network bandwidth, and memory consumption but no power consumption.

In this work, we design, implement, and evaluate machine learning algorithms in an ESP32 (320 kB of RAM) for protecting IoT devices (i.e., edge nodes), demonstrating adequate performance. Another differentiation from our work compared to the knowledge on intrusion detection for IoT is the lack of low-level implementation (i.e., kernel mode) for the security mechanism. Usually, they work with user-space implementations and commonly consider the use-case of deployment in an IoT gateway. In our case, we use edge deployment in the device itself. Additionally, works consider more powerful devices than microcontroller-based systems, such as single-board computers like Raspberry Pi [Mirsky *et al*., 2018; Eskandari *et al*., 2020; Hafeez *et al*., 2020; Utomo and Hsiung, 2019; Bertoli *et al*., 2021]. We evaluate and propose a low-level implementation tied to the network stack on microcontrollers (Lightweight IP – lwIP).

We summarize all the presented related works in Table 1. Our findings show IoT, IT, and OT domain applications. Next, we evaluate the presence of an update mechanism for the security mechanism, for this attribute only our work presents an specific solution. We are also assessing the deployment of T800 on resource-constrained devices, the microcontroller-based system. Regarding runtime performance evaluation, our scope is CPU, memory, network, and, in addition to previous works, the power consumption of the proposed mechanisms.

Our contribution advances the state-of-the-art in deploying machine learning for network security on constrained devices. Unlike previous frameworks that primarily focused on the characterization and prevention of known attacks using machine learning models in high-resource environments [Bertoli *et al*., 2021], our work specifically addresses the challenges of implementing such models directly on microcontrollers. We propose an approach that facilitates the deployment of machine learning algorithms on devices with limited computational capacity and incorporates a dynamic update mechanism. This mechanism enables our system to adapt to evolving network threats, a feature absent in most existing solutions. By embedding the intelligence and decision-making capabilities directly on the edge device, our method achieves autonomy from network infrastructure (i.e., zero-trust), enhances security, and lowers deployment costs.

Furthermore, we systematically address the issue of the limited lifespan of ML-based security solutions by proposing a modular framework that allows for easy updates and modifications in response to changing attack patterns. This approach ensures that our system remains effective over time, addressing the framed life limitation of 6 to 8 weeks identified in previous studies [Viegas *et al*., 2021]. The direct deployment on microcontrollers, combined with an updating protocol, sets our work apart by providing a scalable, efficient, and cost-effective solution for enhancing the security of IoT devices at the network's edge.

Most related works focus on point methods with restricted generality and often lack on-device deployment (edge nodes). T800 differs from them because it has a design adaptable to other platforms with low computational performance and capable of coupling to different TCP/IP protocols and operating systems stacks. Our solution also allows IoT devices to drop malicious traffic, increasing constrained devices' complete scope and considering CPU, memory, network, and power for runtime performance evaluation policies. T800 enables the execution of decision trees, logistic regression, SVM, and multilayer perceptron algorithms. Hence, the present work advances state-of-the-art by making security policies directed to packet filtering viable in resource-constrained IoT devices.

3 T800 — Packet filtering

In this section, we detail the design and execution plan of the T800 packet filtering mechanism, engineered for ESP32 microcontroller-based IoT devices, which are constrained by a limited memory capacity of 320 kB RAM. The primary goal of this design is to facilitate the deployment of efficient packet filtering mechanisms on such devices, ensuring they consume minimal computational resources while providing an additional security layer.

Addressing the unique challenges of edge computing devices, our design strategy emphasizes minimal overhead and seamless integration with the device's existing software infrastructure. To this end, we adopt machine learning-driven security policies with a zero-trust framework. This approach shifts the defense mechanisms onto the device, moving away from traditional network perimeter-based security models.

Table 1. Related works for IoT attack detection / security-mechanisms. The ✓ means the presence of the attribute under evaluation, in the resource-constrained device the evaluated microcontroller is presented. For runtime performance evaluation, the complete scope considers CPU, Memory, Network, and Power.

	Domain	**Update Mechanism**	**Resource-constrained Device**	**Runtime Performance Evaluation**
Niedermaier *et al*. [2019]	OT		✓ (ARM Cortex-M7)	only network
Manocchio *et al*. [2022]	IoT		✓ (ESP32, ESP8266, ATMega328p)	
Soe *et al*. [2020]	IoT			
Jan *et al*. [2019]	IoT			
Filus *et al*. [2021]	IoT			
Bertoli *et al*. [2021]	IT			✓(no power)
Mudgerikar *et al*. [2019]	IoT			✓(no power, qualitative)
Mirsky *et al*. [2018]	IoT			only network
Eskandari *et al*. [2020]	IoT			✓(no power)
Hafeez *et al*. [2020]	IoT			✓(no power)
Utomo and Hsiung [2019]	IoT			
Our work	IoT	✓	✓ (ESP32)	✓

This is particularly relevant in IoT, where devices face diverse and continuously changing threats.

The T800 initiative responds to these challenges by offering a framework for implementing adaptable security policies tailored to resource-constrained environments. While T800 currently focuses on *stateless* packet filtering, its architecture allows for integrating more sophisticated models, including *stateful* processing and anomaly detection mechanisms. T800 leverages the ESP-IDF[1] — an open-source IoT development framework by Espressif designed for creating applications on the ESP32 platform. This ensures that the T800 packet filter minimizes computational demand and offers a flexible structure that can be extended to accommodate additional functionalities and adapt to various system requirements.

3.1 Architecture

The T800 component, as depicted in Figure 1, is designed to analyze network traffic on ESP32-based devices, focusing on evaluating incoming packets.

Figure 1 provides a comprehensive view of the IoT device architecture, detailing the integration of the T800 component within the FreeRTOS environment supported by the ESP-IDF. This includes interactions with the scheduler, memory management, process and task management, and system monitoring. It emphasizes that T800 intercepts incoming network packets at the kernel space level before any processing by the device, ensuring a preliminary security assessment of all ingress traffic.

First, all the evaluation is based on packet header features to differentiate the traffic between malicious and benign. For this purpose, T800 captures each network packet entering the TCP/IP stack. The implementation follows the function of processing the TCP/IP protocol stack packets within ESP-IDF's lightweight IP (lwIP) component.

T800 has been integrated into the ESP-IDF as a novel component, extending the framework's capabilities. Once T800 evaluates trained machine learning algorithms for the packet filtering task, it introduces dependencies on two critical libraries for its operation: *esp-nn*[2], which is Espressif's optimized library for neural network operations, and *tflite-lib*[3], provided by Google for deploying TensorFlow Lite models on resource-constrained devices like the ESP32. These libraries enable machine learning techniques, such as model quantization, to process network traffic data efficiently.

While running on the ESP32, T800 undergoes an initialization step. This initialization step occurs by providing an initial configuration structure, including a function that classifies packets and information about the function execution context (static or dynamic). This type of context is necessary because T800 can operate in two different ways, whether storing the TCP flow state (*stateful*) or not (*stateless*). During the T800 initialization, it runs in the foreground of the process in a dedicated thread (i.e., FreeRTOS task). As a result, FreeRTOS preempts the ready processes (tasks) among the available CPU cores, and T800 is one of them. After the initialization step, T800 acts directly on the Network and Transport layers, capturing the packets provided by one of the network interfaces. The function that uses a machine learning model to classify packets is chosen and defined in advance in the initial configuration structure.

The analysis begins with the examination of TCP and IP header data. These are encapsulated within packets that the ESP32 device captures, utilizing a structure known as `pbuf`, a fundamental component of the lwIP stack [Dunkels, 2001]. The `pbuf` structure encompasses the essential elements of network communication, including TCP/IP headers, data link layer information, payload content, and, optionally, links to subsequent packets in a sequence when applicable.

The `pbuf` structure in the lwIP stack is a sophisticated mechanism designed to efficiently manage network packets within constrained environments, such as embedded systems. At its core, `pbuf` facilitates the encapsulation and sequencing of network data, allowing for the modular handling of diverse communication protocols. This structure is particularly adept at conserving memory resources, a critical consid-

[1] esp-idf: `https://github.com/espressif/esp-idf`

[2] esp-nn: `https://github.com/espressif/esp-nn`

[3] Tensorflow Lite: `https://github.com/tensorflow/tflite-micro`

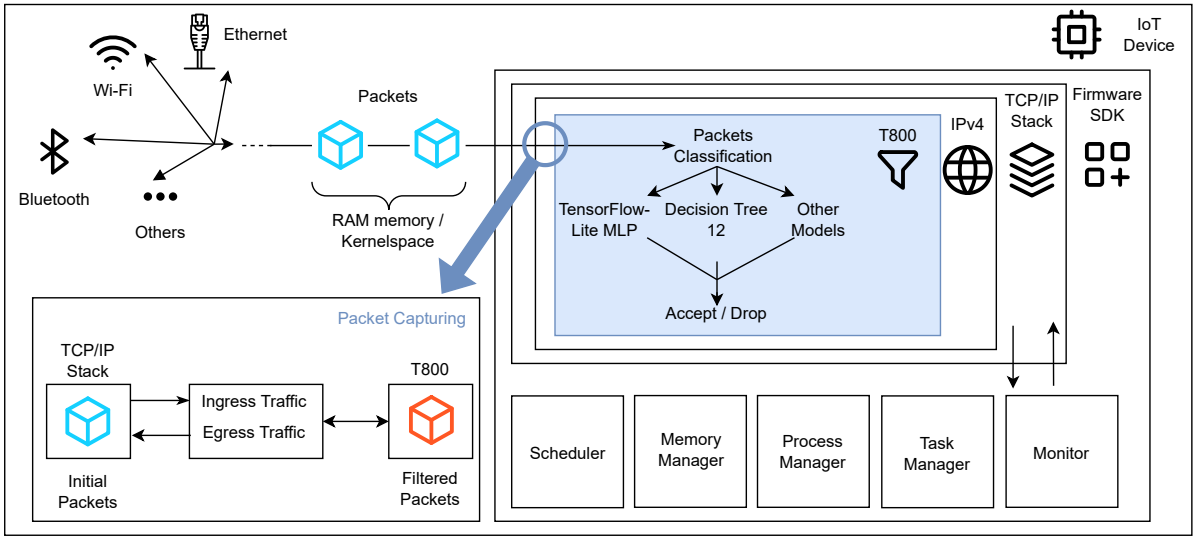

Figure 1. T800 packet filtering architecture. T800's mechanism consists of intercepting the network packets after they are available in RAM. The interception point corresponds to the first function inside the TCP/IP stack to ensure filtering before processing by the adjacent layers. The decision to accept or drop packets is pluggable, leaving the security policy specification flexible and dynamic (Adapted from Fernandes *et al*. [2022]).

eration for devices like the ESP32. It achieves this through a chained list architecture, enabling the assembly and disassembly of packet sequences without necessitating contiguous memory blocks.

T800's evaluation process decisively impacts how the lwIP stack processes packets. Should the integrated machine learning model classify a packet as malicious, T800 promptly disposes of the corresponding `pbuf`, effectively neutralizing the threat. In contrast, packets classified as safe are permitted to proceed along the ESP-IDF's processing pipeline, ensuring uninterrupted legitimate network traffic. Through this discerning approach, T800 serves as a critical line of defense against malicious incursions, enhancing the ESP32-based IoT device's security posture at the very edge of the network.

4 Methodology

This section initially provides an overview of the machine learning approach employed in this study. We then detail the T800 modular implementation and its update mechanism. Following that, we delve into creating packet filtering rules using machine learning and describe the specific use case of port scanning that we have considered in this work. We also discuss the dataset for training the models and provide details about the machine learning parameters. Finally, we present a comprehensive methodology for rigorously evaluating computational metrics.

4.1 Machine Learning Approach

In traditional software development, a specific set of rules is defined to process inputs and produce desired outputs. For instance, consider a software program designed to control a thermal system. This program would read data from a temperature sensor and compare it against a predetermined threshold. If the temperature is below the threshold, the program will activate a heater; otherwise, no action will be taken.

In contrast, machine learning takes a different approach. Rather than relying on predefined rules, machine learning models are trained using data and corresponding labels (the supervised learning approach). Taking our previous example, a machine learning approach would learn from historical temperature data and the related decision to use the heater. While the previous example is relatively straightforward and can be effectively addressed using traditional software implementations, more complex scenarios, such as intrusion detection, pose challenges that cannot be easily solved with fixed rules or accommodate the ever-evolving nature of network traffic.

Our study employs a machine learning approach to develop models that effectively distinguish between benign and malicious packets a microcontroller-based system receives. In this application, the complexity and dynamic nature of attack behaviors make it infeasible to code rules to cover all possible scenarios manually. Furthermore, understanding and interpreting these behaviors for traditional software implementation would also be challenging. Therefore, leveraging machine learning enables us to tackle the intricacies of the problem and effectively detect and classify attacks in a more adaptable and comprehensive manner.

4.2 T800 Implementation

Implementing T800 has the following main objectives: low computational cost and the ability to update filtering policies (update mechanism). In this context, the filter implementation establishes a standardized interface that, with few resources, facilitates the development and implementation of new filtering rules. Figure 2 depicts this structure. It has two essential entities: a value responsible for encoding how it works (working mode) and a classification function that enforces a packet filtering policy, a classification function represented by a trained machine learning model.

The working mode defines how the system performs the packet capture and when it executes the classification func-

tion, including its arguments. The classification function receives the required context for performing the classification and returns the packet's classification. Therefore, given requirements for execution time and memory footprint, the possible choices could be made to achieve the most suitable working mode and the best policy available. Such as selected working mode (thus capturing and categorizing the incoming packets) but using a lower footprint machine learning algorithm, like a decision tree.

An important attribute is that this approach allows the alteration of both entities at runtime, like turning it off or changing the trained machine learning model. This design decision provides greater flexibility to the filter and achieves the update mechanism requirement.

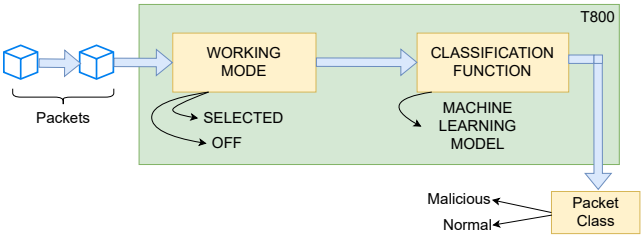

Figure 2. Implementation structure of T800. The current packet workflow is dynamically activated or deactivated depending on the execution context. In addition, the pluggable classification function implements the heuristics to identify malicious or normal packets (Adapted from Monici *et al*. [2022]).

4.3 Packet filtering rules

A packet filtering policy specifies whether a packet continues its natural processing path within a protocol layer. Thus, considering how general this definition may be, there are numerous ways to construct such a policy. Its construction can range from rules selected by a domain expert to unsupervised machine learning models. In this work, we aim to assess if machine learning-based models apply to microcontroller-based IoT systems for packet filtering.

T800 follows the AB-TRAP framework [Bertoli *et al*., 2021] that seeks to design protection mechanisms based on machine learning models. AB-TRAP covers the whole development chain, from the study of the normal behavior characterization to the implementation and performance evaluation of the proposed solution to the operation. Thus, the packet filtering solutions tested on the T800 followed this same framework.

First, we consider generating data from the attacks and training the applicable machine-learning models with this data. Next, we transpose those trained models to a software component. Then, we implement these components on the ESP32 device for performance evaluation. In other words, the filtering policies used by T800 correspond to machine learning models trained offline, which may require periodic updates according to the needs of each application and environment.

For the T800 evaluation, we employ five different filtering policies – machine learning supervised learning algorithms – to test the operational viability on a low computational cost platform represented by the ESP32 target. Another point under consideration is to verify the applicability of the T800 in preserving the ability to generalize the rules.

The algorithms under consideration for effectiveness include a Decision Tree (DT) with a depth of 12 (DT-12). Additionally, a Multi-layer Perceptron (MLP) featuring two hidden layers, each containing 16 neurons, alongside an output layer equipped with two neurons. Furthermore, the study also encompasses Logistic Regression (LR) and a Support Vector Machine (SVM) fitted with a linear kernel, which are also being assessed.

For a baseline reference, we consider the standard system provided by the ESP-IDF framework that comprises the FreeRTOS with lwIP but without the T800 component; this approach allows us to verify how the introduction of our solution impacts the system resources.

Decision Trees require less computing power when put into operation, whereas Multilayer Perceptrons are more intensive than the former, as reported on Bertoli *et al*. [2021]. Thus, choosing them for comparison allows us to explore a wide range of classical machine learning algorithms in the computing exigency spectrum, a microcontroller-based device. A key point is the algorithm implementation feasibility in our testbed, the Espressif ESP32. Therefore, we intend to verify the on-device cost of models with both high and low expected computing costs.

These traditional machine learning algorithms (i.e., policies) fit nicely in implementing the T800 component since minimizing the computational cost in a microcontroller-based device is essential due to the scarcity of resources. Considering the system's limited resources, the Logistic Regression and the SVM models are expected to exhibit these same qualities. Furthermore, the models are implemented with the Tensorflow framework and then converted to the final model for ESP32 through Tensorflow-Lite.

To better understand, the Decision Trees (DT) correspond to a chain of conditional structures based on attribute tests derived from the original models. Interpretability and portability (from training to implementation) are key features that make DT feasible for low-power computing platforms, requiring low memory footprint and computing power. Thus, these models are a good fit to implement as policies for the T800 component. Similarly, the Logistic Regression and the SVM models exhibit the same qualities concerning the system's limited resources. Furthermore, these models were implemented through the Tensorflow framework and then converted to the final model of ESP32 through Tensorflow-Lite.

The MLP is specified with a sigmoid activation function for the hidden layers and a softmax activation function for the output layer. Both its implementation and training are done with Tensorflow and converted to the final model on ESP32 with Tensorflow-Lite. This conversion makes this model adopt space and processing optimizations that offer satisfactory performance in implementing inference mechanisms on embedded and resource-constrained devices.

4.4 Use-case, dataset and model training

In this work, the use case under analysis is detecting port scanning attacks. Port scanning is part of the reconnaissance step of an attack. The reconnaissance is the first step according to the cyber kill chain framework [Yadav and Rao, 2015].

Thus, avoiding an attack in its initial phase saves resources and reduces its possible impact.

Considering the IoT networking characteristics, most attacks rely on scanning the networks to identify visible and vulnerable IoT devices. Also, scanning the local infrastructure (e.g., lateral movement) or throughout the Internet. Thus, port scanning represents a critical attack vector to IoT systems [Durumeric *et al*., 2014, 2015; DeMarinis *et al*., 2019; Dahlmanns *et al*., 2022].

Regarding the model training, the decision tree uses the entropy metric as a division criterion for the training stage. The MLP was trained for $2,000$ *epochs* with the Adam optimizer, a learning rate of 1.10^{-5}, and a batch size 260.

The dataset for all training steps is the AB-TRAP [Bertoli *et al*., 2021] with a slight change that consists of removing the `tcp.window_size` attribute. This dataset generates attack packets through a testbed and a collection of exclusively port-scanning malicious traffic. On the one hand, we use TCP scanners such as `Zmap`, `masscan`, `Hping3`, `Unicorn Scan`, and `NMap`, resulting in 86.480 malicious packets.

On the other hand, MAWILab [Fontugne *et al*., 2010]) to represent the benign samples. The dataset consists of $103,094$ packets sampled on November 21, 2019, from an Internet link that connects the United States of America to Japan. After that, the malicious and benign traffic is merged into a single dataset through a *salting* approach. From this process, the results of *F1-score* 0.79, 0.34, 0.93, and 0.91 were obtained for the Logistic Regression, SVM, Decision Tree, and MLP in an anticipated evaluation before T800 deployment and assessment.

4.5 Computational Metrics Evaluation

The T800's performance is measured using four metrics summarized in Table 2. The first is the CPU utilization rate of the two cores present in ESP32. The second is the amount of memory allocated by the T800 (only static memory is measured as the component does not perform dynamic memory allocation). The third is the rate at which the Wi-Fi interface receives packets. The last one is the power consumption of the device. These metrics are similar to the performance, memory, and energy metrics proposed by Almakhdhub *et al*. [2019].

Table 2. Measurements collected and evaluated for T800.

Metric	Description
CPU	CPU usage (all cores)
MEM	Memory usage (stack only)
NET	Network usage (Wi-Fi only)
POWER	Energy consumption in milliwatts (mW)

Software developed for edge IoT devices usually works in the context of scarce computing resources. Thus, these metrics are relevant to the context of embedded systems, which is why we consider them in our analysis. For example, the ESP32 has only 320 kB of RAM and a dual-core processor with a clock cycle of 240 MHz. Thus, verifying eventual increases in the lwIP processing rate or allocated memory is mandatory to ensure the system's operation. Furthermore, the system may become overloaded depending on the packet rate experienced by the network interface due to such computing constraints. Thus, monitoring the network utilization is crucial to understanding how T800 impacts the TCP/IP protocol stack implementation of lwIP. Finally, as some IoT devices could be battery-powered, it is also necessary to understand the impact of the T800 on energy consumption.

Based on these metrics, we consider different execution environments to contrast the ESP32 operation with and without the T800 in different situations. Table 3 describes the design of experiments we adopt in our study. Two factors are under consideration for simulating different execution scenarios. One of them is the intensity of benign network traffic received on the Wi-Fi interface (I), which can be 8 Mbps (I0) or 16 Mbps (I1). The other is the presence of malicious packets in the network traffic destined for ESP32 that varies between the absent (M0) and present (M1) values. We refer to the experiments as IxMy codes in all possible combinations, {M0I0, M0I1, M1I0, M1I1}.

Table 3. Properties of the traffic during the experiments.

Property	Level	Code
Traffic intensity	8 Mbps or 16 Mbps	I0, I1
Malicious traffic	Absent or Present	M0, M1

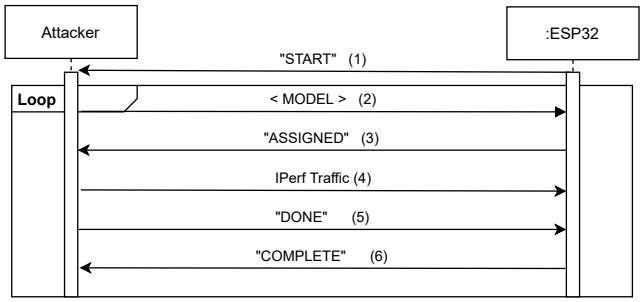

Figure 3. Sequence diagram for ESP32's evaluation metrics capturing. Describe the protocol to execute an experiment replica. It dynamically loads the corresponding classification function, and the workload excites during a period after the system signaling is ready (Adapted from Monici *et al*. [2022]).

The messages between the test devices occur between an ESP32 and an attacking computer on the same wireless network. The attacking computer generates both benign and attack traffic (over TCP) and collects all the experiment data (over UDP). To do this, they communicate through the UDP protocol to manage the settings of a TCP connection that will be active for 360 seconds. As depicted in Figure 3, in the first execution step (1), ESP32 sends a message to the attacking machine, signaling the beginning of the experiment. In the second step (2), the attacker sends a code specifying the T800's filtering policy or disabling T800. Then, in the third step (3), the ESP32 responds by communicating that it has received the necessary information and has already completed its initial configuration. In this process, two servers start on ESP32. The first gets all TCP traffic from the simulation.

In contrast, the second collects performance metrics and

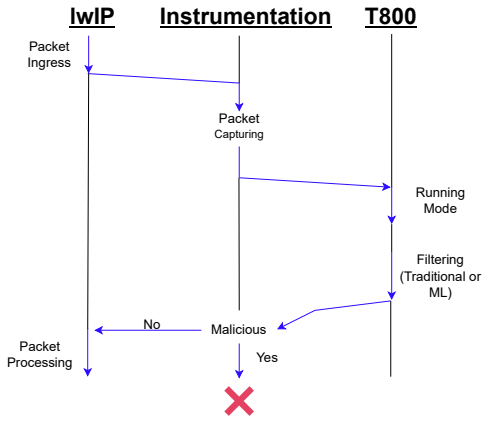

Figure 4. Interaction of T800 with ESP-IDF stack lwIP. T800's design assumes coupling to the lwIP, influencing the minimum with the original system. However, it is flexible to adapt to other stacks (Adapted from Monici *et al*. [2022]).

Table 4. Description of the hardware used in the experiments.

Description	CPU (GHz)	Memory	Operating System
Test computer	Intel i7-8565U-(4.6) x8	16 GiB	Manjaro Linux x86_64
ESP32 DevKit V1	Xtensa-(0.240) x2	320 kB	FreeRTOS, ESP-IDF
Test computer	Intel i5-3337U-(2.7) x4	6 GiB	Ubuntu 20.04.2 LTS

sends them to the attacking machine via UDP at a 1-second rate. After that, in the fourth step (4), the attacking machine generates the network traffic. Finally, after collecting all the metrics, in the fifth step (5), the attacking machine sends a message to ESP32 representing the end of the experiment. Finally, in the sixth step (6), the ESP32 sends a response confirming the end of the experiment. It is worth noting that this approach allows the packet filtering policy to be changed at runtime (step 2), enabling the solution's adaptability.

As illustrated in Figure 4, the packet filter integrates with the system's base TCP/IP stack (lwIP). First, packets ingress through the stack and are intercepted by the instrumentation part before reaching T800. Then, T800 is configured with its working mode to perform the filtering. Such filtering can be traditional or advanced. The first performs static packet filtering rules without requiring complex algorithms. The advanced one follows the approach taken in our experiments with the help of machine learning models. Finally, the packet can be classified as malicious or not and can be processed in lwIP or discarded. This integration allows T800 to have an in-depth view of the system, which allows the measurement of computational cost.

All the performance metrics except the energy consumption are available from the FreeRTOS API functions implemented in ESP32. For power consumption, the ESP32 power pins with a current/power sensor[4] allows to monitor the energy consumption by hardware. The sensor transmits the readings of these measurements by I2C communication to a reading device (Arduino-based) that uses serial communication with the test computer, in our case the same as the attacker, to record these measurements (serial over a USB connection). In conjunction with the energy measurements, the reading device also monitors a discrete signal from ESP32 that is responsible for indicating the start and conclusion of an experiment, thus facilitating the post-processing of the data. In addition, the normal traffic with intensities of 8 Mbps (I0) and 16 Mbps (I1) are generated through *IPerf v2.0* and malicious traffic is generated with *Nmap* to perform scanning attacks through wireless communication. Figure 5 depicts the setup.

[4]Texas Instrument INA219: `https://www.ti.com/lit/ds/symlink/ina219.pdf`

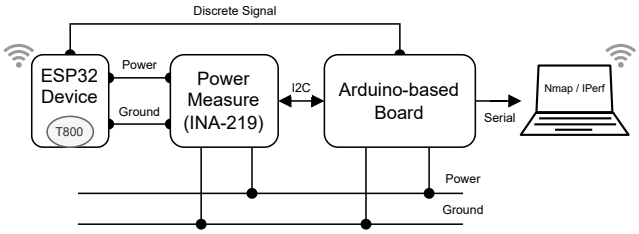

Figure 5. Setup for measuring the T800 energy consumption in an ESP32-based system. The monitor is implemented in hardware and attached to the physical system. All data gathered is stored separately on the monitor.

This work adopts a complete factorial design of experiments. Experiments I0M0, I0M1, I1M0, and I1M1 run with all filtering policies and without T800. Further, we collect all measurements in a thirty (30) replicas experiment execution. Finally, the hardware specifications of both the test/attacker computer and the ESP32 are present in Table 4, along with those of the device used to measure energy consumption during collection (later test computer).

5 Results and Discussions

All processes described in Section 4.5 result in the values presented in Figures 6 to 9. These figures show the consumption of CPU, Network Bandwidth, Stack Memory usage, and Energy, respectively. We grouped metrics under interest in each of the corresponding experiment graphs. For instance, Figure 6 presents CPU consumption for the I0M0 experiment with all models under consideration: decision tree (DT-12), logistic regression (LR), multilayer perceptron (MLP), SVM and without T800. We presented all the metrics as violin plots, meaning the samples represent a single aggregate value. However, the energy metric was continuously collected throughout the experiment, whereas we employed $3,600$ samples for the remaining metrics. Then, it was possible to analyze the median, maximum, and minimum. Finally, we compare the data from the executions of each model and those that do not use T800. The bars of each violin plot represent the extremities (maximum and minimum values) and the median (central bar) with the probability density function.

The CPU usage is present in the graphs of Figure 6. For the experiment, I0M0, the medians of this metric are in the interval $[0.10, 0.12]$. For the experiment I0M1, the medians are in the interval $[0.10, 0.15]$. Finally, for I1M0, the medians are in the interval $[0.15, 0.19]$. Finally, for the experiment I1M1, the medians are in the interval $[0.16, 0.20]$. This data indicates a significant difference between the versions of T800 (all models) and those without T800. Furthermore, Figure 7 shows the network bandwidth. The I0 experimental levels represent a traffic intensity of 8 Mbps (median), and the I1 level exposes a median of 16 Mbps. Therefore, this metric's value remains close to the benign traffic intensity specified for the experiment.

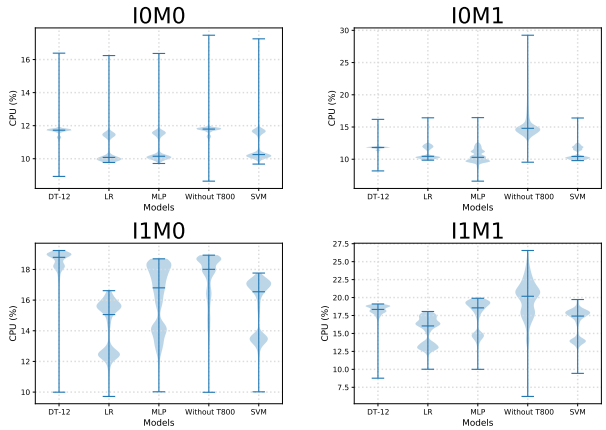

Figure 6. Consumption values obtained from experiments: CPU Usage. The row I0Mx corresponds to low-intensity traffic (8 Mbps) and I1Mx to the high intensity (16 Mbps). The column IxM0 indicates the absence of malicious traffic, and IxM1 the presence. T800 enables the system to experience lower CPU usage, as it drops the unsolicited packets before processing them.

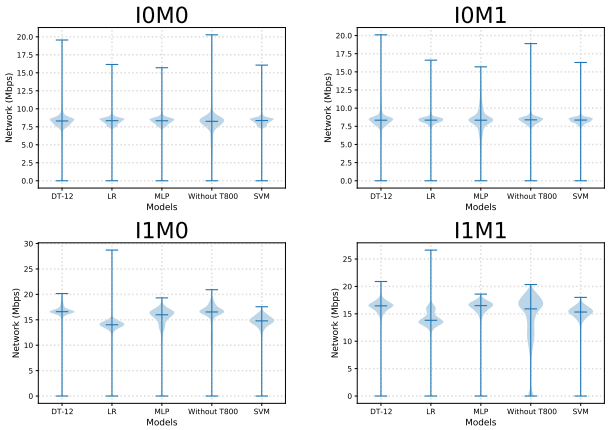

Figure 7. Consumption values obtained from experiments: Network Bandwidth. The row I0Mx corresponds to low-intensity traffic (8 Mbps) and I1Mx to the high intensity (16 Mbps). The column IxM0 indicates the absence of malicious traffic, and IxM1 the presence.

Moreover, the stack memory metric is shown in Figure 8. The resulting medians remain constant with the value of 4116 bytes, and the interquartile ranges show that the variation in this metric is negligible because of the magnitude being in tens of bytes. For last, Figure 9 displays the energy consumption. In them, most medians are in the range of $[174, 176]$.

Besides, it is relevant to analyze the time series of the CPU usage shown in Figure 10. The results of the experiments using machine learning models trained by Tensorflow presented lower computing consumption (SVM, Logistic Regression, and Multilayer Perceptron). It suggests that the framework optimizations make the models suitable for our low computing capacity systems context. Additionally, the graphs display an apparent discrepancy in performance when malicious traffic is present in the network flow. For instance, the baseline policy (AN - without T800) CPU usage is much higher than the filtering policies that use machine learning models in the I1M1 and I0M1 graphs. However, after the malicious traffic is interrupted, the CPU usage decays. Thus, T800 provides (*i*) security protection against the reconnaissance (Cyber Kill Chain [Yadav and Rao, 2015]) and (*ii*) a reduction in the IoT system resource usage.

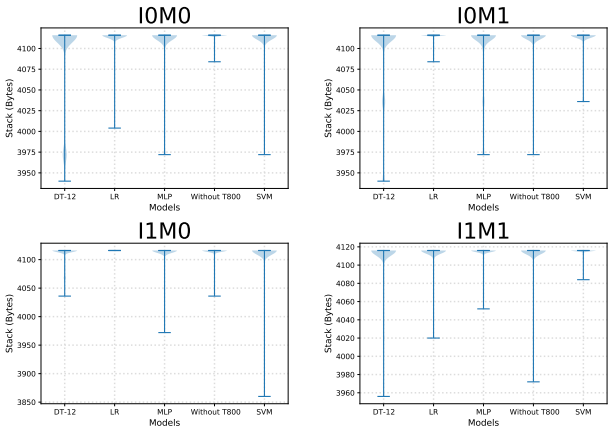

Figure 8. Consumption values obtained from experiments: Stack Memory Usage. The memory footprint remains approximately the same as the system with the T800.

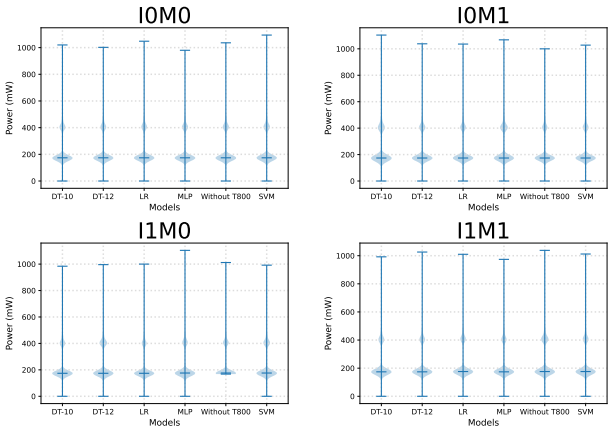

Figure 9. Consumption values obtained from experiments: Energy Usage. The energy demands remain approximately the same as the system with the T800.

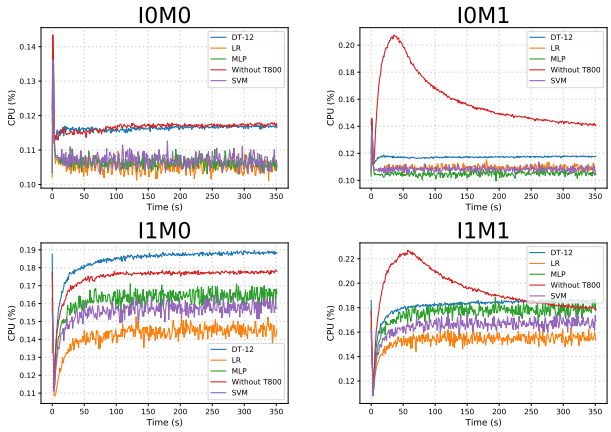

Figure 10. Time series of the CPU consumption values obtained through the experiments. We observed that the experiments executed without the T800 demand more CPU than those with a packet filter. For instance, the time series without T800 presents a high peak, and it is due to the burst of the malicious packets received by the network interface. On the other hand, each machine-learning model rejected those packets, causing a steady-state behavior.

5.1 Influence of Factors

To analyze the influence of the selected factors on the variation of CPU usage values, we consider a machine learning-based filtering policy (**A**) as a factor of the experiments. Thus, as presented in Table 5, besides the factors and levels of the experiment design already presented in Section 4.5, the

values present (A1) or absent (A0) were introduced to make up the new factor **A**. In addition, the levels were mapped for discrete values so that $T0 = -1$ and $T1 = 1$ for any factor A. In this way, the experiment and such performance metric started to be represented by Table 6, where the variables $y_{i,j}$ are the mean of the values obtained in the replication j of the experiment i. From that, it was realized that the regression $\mathbf{XQ} = \bar{\mathbf{Y}}$ that considers an additive system model under analysis, as presented by equations 1, and 2.

$$\mathbf{X} = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & T & I & M & TI & TM & IM & TIM \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \quad (1)$$

$$\mathbf{Q} = \begin{bmatrix} q_0 \\ q_T \\ q_I \\ q_M \\ q_{TI} \\ q_{TM} \\ q_{IM} \\ q_{TIM} \end{bmatrix} \text{ and } \bar{\mathbf{Y}} = \begin{bmatrix} \bar{Y_1} \\ \bar{Y_2} \\ \bar{Y_3} \\ \bar{Y_4} \\ \bar{Y_5} \\ \bar{Y_6} \\ \bar{Y_7} \\ \bar{Y_8} \end{bmatrix} \quad (2)$$

Table 5. Properties of the traffic during the experiments.

Property	Level	Code
Machine learning policy	Absent or Present	A0, A1
Traffic intensity	8 Mbps or 16 Mbps	I0, I1
Malicious traffic	Absent or Present	M0, M1

Where $\mathbf{Q}$ represents the set of estimated parameters by the least squares algorithm, $\mathbf{X}$ the set of predictors related to each factor and the interactions between them, and $\bar{\mathbf{Y}}$ the sample mean values generated by the experiments, thus, it is possible to not only obtain the variations caused by each factor and their combinations (SS_A, equation 3) but also to compute the variation attributed to experimental errors (SSE, equation 4) and a value corresponding to the total variation of the metric.

$$SS_A = 2^3 \;\; 10 \;\; q_A \quad (3)$$

$$SSE = \sum_{i=1}^{8} \sum_{j=1}^{10} (y_{ij} - \bar{y}_i)^2 \quad (4)$$

Finally, we computed the influence factor as the variation caused by a factor divided by the total variation for each filtering policy. Then, it calculated an average of all the policies as a form of aggregation (Equation 5).

$$SST = SS_T + SS_I + SS_M + SS_{TI} + SS_{TM} + SS_{IM} + SS_{TIM} + SSE \quad (5)$$

The obtained results are in Table 7. Each column in the table quantifies directly how much the factor impacts CPU utilization. As seen, the **I** factor (traffic intensity) impacts most in CPU utilization, 46%, 52%, 57%, 63%, and 72% for Linear Regression (LR), without T800 (w/o T800), Support Vector Machine (SVM), Multilayer Perceptron (MLP), and Decision Tree (DT-12), respectively. For example, the traffic intensity influences from 46% to 72% of the current observed CPU utilization. The presence of T800 (factor **A**) is the second most impacting factor of CPU, followed by the incidence of malicious traffic (factor **M**). On the other hand, the interaction of the factors (columns **AI**, **AM**, **IM**, and **AIM**) is negligible. The last column (Err) is the error observed in the least square regression due to the systems' stochastic behavior. Finally, the last row is the simple average of the values. In conclusion, the traffic intensity (**I**) corresponds to 58% on average of the currently observed CPU utilization. T800 (**A**) leads to an overhead of 12%. The malicious traffic (**M**) yields 10%. Furthermore, the interaction of the factors corresponds to 4%, and the observed error is 14%.

Table 6. Representation of the complete factorial planning with adding the factor T and discrete levels.

i	1	T	I	M	TI	TM	IM	TIM	$\mathbf{Y}_{i,1}$	...	$\mathbf{Y}_{i,175}$	$\bar{\mathbf{Y}}_i$
1	1	−1	−1	−1	1	1	1	−1	$y_{1,1}$	...	$y_{1,10}$	$\hat{y_1}$
2	1	−1	−1	1	1	−1	−1	1	$y_{2,1}$	...	$y_{2,10}$	$\hat{y_2}$
3	1	−1	1	−1	−1	1	−1	1	$y_{3,1}$	...	$y_{3,10}$	$\hat{y_3}$
4	1	−1	1	1	−1	−1	1	−1	$y_{4,1}$	...	$y_{4,10}$	$\hat{y_4}$
5	1	1	−1	−1	−1	−1	1	1	$y_{5,1}$	...	$y_{5,10}$	$\hat{y_5}$
6	1	1	−1	1	−1	1	−1	−1	$y_{6,1}$	...	$y_{6,10}$	$\hat{y_6}$
7	1	1	1	−1	1	−1	−1	−1	$y_{7,1}$	...	$y_{7,10}$	$\hat{y_7}$
8	1	1	1	1	1	1	1	1	$y_{8,1}$	...	$y_{8,10}$	$\hat{y_8}$

The influence of factors indicates that the predominant factor in the CPU usage value is the benign traffic produced by the *IPerf* traffic generator (I). Also, the results obtained for the presence or absence of the machine learning models showed that this factor has little influence on the variation of CPU usage. Therefore, our results suggest that T800 in lightweight IP incurs low computing overhead regardless of the network workload (high security and low traffic or one with low security and high traffic), a feasible solution for an intelligent packet filtering mechanism on microcontroller-based devices.

Table 7. Results of the influence of factors analysis for the CPU usage metric. Here, **A** represents the presence of a filtering policy in the T800, **I** represents the *IPerf* network traffic intensity, and **M** represents the presence of malicious traffic in the network.

Model	A	I	M	AI	AM	IM	AIM	Err
w/o T800	0.00	0.52	0.23	0.00	0.00	0.03	0.00	0.22
MLP	0.12	0.63	0.07	0.01	0.03	0.00	0.02	0.12
DT-12	0.03	0.72	0.05	0.02	0.06	0.01	0.01	0.11
LR	0.27	0.46	0.09	0.00	0.03	0.00	0.01	0.12
SVM	0.17	0.57	0.08	0.00	0.04	0.00	0.01	0.12
Average	0.12	0.58	0.10	0.00	0.03	0.00	0.01	0.14

6 Conclusion

This paper presented the T800 packet filter for Internet of Things (IoT) devices with low-computing power. The proposed architecture is adaptable to other platforms due to the instrumentation of the protocol stack used, simply identifying the packet interception point. It allows different filtering

policies deployment through implementation with the operating system. We show the solution's effectiveness by describing the implementation with the ESP32 development platform (FreeRTOS, lwIP TCP/IP stack, and ESP-IDF framework).

The experiments showed high efficiency in the chosen approach since the presence of the T800 reduced the consumption of computing resources in the presence of malicious traffic. Moreover, the action of the packet filter allowed malicious traffic disposal and positively impacted the system. The results suggest that our design adequately filters network packets with low impact in IoT platforms. Furthermore, our findings showed no statistically significant difference among the baseline, Decision Tree, and Multilayer Perception; therefore, the model with the highest accuracy is suitable for T800's deployment.

In future works, the objective is to implement *stateful* packet filtering policies. Another point of experimentation concerns an anomaly detection approach to detect never seen attacks [de Carvalho Bertoli *et al*., 2023]. Finally, to integrate a *zero-trust* based architecture aimed at IoT devices where the T800 is an enabler for such a solution.

Finally, this paper represents an effort to bring more realism while approaching machine learning to devise security mechanisms in IoT with a microcontroller-based implementation. Understanding how to operate the algorithms (MLOps) in kernel mode is challenging. Moreover, evaluating how to translate data science notebook analysis into an ESP32 implementation represents a turning point toward realistic testbeds. All reproducible code used is in the repository `https://github.com/c2dc/T800`.

Declarations

Acknowledgements

This paper is an extended version of a previously published paper [Fernandes *et al*., 2022; Monici *et al*., 2022], and we have enhanced the work's scope, including new data and analysis to evaluate packet filtering models. We improved the review and positioning of the contribution regarding the related works. Detailed explanation for the proposed methodology. Addition of SVM and Logistic Regression algorithm analysis on the proposed T800 and inclusion of additional data from both approaches. Introduced CPU consumption time series analysis. Re-run of the experiments, increasing the sample size from 1800 to 3600 samples.

This research had the support from the Programa de Pós-graduação em Aplicações Operacionais—PPGAO/ITA, from Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) granting #2020/09850-0 and #2021/09416-1, and from Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) granting #157021/2021-1.

Authors' Contributions

All authors equally contributed to the writing of this article, read and approved the final manuscript.

Competing interests

The authors declare that they have no competing interests.

Availability of data and materials

All reproducible code used is in the repository `https://github.com/c2dc/t800`.

References

Abadade, Y., Temouden, A., Bamoumen, H., Benamar, N., Chtouki, Y., and Hafid, A. S. (2023). A comprehensive survey on tinyml. *IEEE Access*, 11:96892–96922. DOI: 10.1109/ACCESS.2023.3294111.

Al-Sarawi, S., Anbar, M., Abdullah, R., and Al Hawari, A. B. (2020). Internet of things market analysis forecasts, 2020–2030. In *2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)*, pages 449–453. DOI: 10.1109/WorldS450073.2020.9210375.

Alieyan, K., Almomani, A., Anbar, M., Alauthman, M., Abdullah, R., and Gupta, B. B. (2021). Dns rule-based schema to botnet detection. *Enterprise Information Systems*, 15(4):545–564. DOI: 10.1080/17517575.2019.1644673.

Almakhdhub, N. S., Clements, A. A., Payer, M., and Bagchi, S. (2019). Benchiot: A security benchmark for the internet of things. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 234–246. DOI: 10.1109/DSN.2019.00035.

Almeida, F. (2023). Prospects of cybersecurity in smart cities. *Future Internet*. DOI: 10.3390/fi15090285.

Antonakakis, M., April, T., Bailey, M., Bernhard, M., Bursztein, E., Cochran, J., Durumeric, Z., Halderman, J. A., Invernizzi, L., Kallitsis, M., *et al*. (2017). Understanding the mirai botnet. In *26th USENIX security symposium (USENIX Security 17)*, pages 1093–1110. Available at:`https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis`.

Apruzzese, G., Laskov, P., and Schneider, J. (2023). Sok: Pragmatic assessment of machine learning for network intrusion detection.

Bertino, E. and Islam, N. (2017). Botnets and internet of things security. *Computer*, 50(2):76–79. DOI: 10.1109/MC.2017.62.

Bertoli, G. D. C., Júnior, L. A. P., Saotome, O., Dos Santos, A. L., Verri, F. A. N., Marcondes, C. A. C., Barbieri, S., Rodrigues, M. S., and De Oliveira, J. M. P. (2021). An end-to-end framework for machine learning-based network intrusion detection system. *IEEE Access*, 9:106790–106805. DOI: 10.1109/ACCESS.2021.3101188.

Dahlmanns, M., Lohmöller, J., Pennekamp, J., Bodenhausen, J., Wehrle, K., and Henze, M. (2022). Missed opportunities: Measuring the untapped tls support in the industrial internet of things. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '22, page 252–266, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/3488932.3497762.

de Carvalho Bertoli, G., Alves Pereira Junior, L., Saotome, O., and dos Santos, A. L. (2023). Generalizing intrusion detection for heterogeneous networks: A stacked-

unsupervised federated learning approach. *Computers & Security*, 127:103106. DOI: 10.1016/j.cose.2023.103106.

DeMarinis, N., Tellex, S., Kemerlis, V. P., Konidaris, G., and Fonseca, R. (2019). Scanning the internet for ros: A view of security in robotics research. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8514–8521. DOI: 10.1109/ICRA.2019.8794451.

Dunkels, A. (2001). Design and implementation of the lwip tcp/ip stack. *Swedish Institute of Computer Science*, 2(77). Available at:`https://www.artila.com/download/RIO/RIO-2010PG/lwip.pdf`.

Durumeric, Z., Adrian, D., Mirian, A., Bailey, M., and Halderman, J. A. (2015). A search engine backed by internet-wide scanning. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 542–553. DOI: 10.1145/2810103.2813703.

Durumeric, Z., Li, F., Kasten, J., Amann, J., Beekman, J., Payer, M., Weaver, N., Adrian, D., Paxson, V., Bailey, M., *et al*. (2014). The matter of heartbleed. In *Proceedings of the 2014 conference on internet measurement conference*, pages 475–488. DOI: 10.1145/2663716.2663755.

Eskandari, M., Janjua, Z. H., Vecchio, M., and Antonelli, F. (2020). Passban ids: An intelligent anomaly-based intrusion detection system for iot edge devices. *IEEE Internet of Things Journal*, 7(8):6882–6897. DOI: 10.1109/JIOT.2020.2970501.

Fernandes, G., Monici, P., Guibo, C., de Carvalho Bertoli, G., Santos, A., and Jr., L. A. P. (2022). Implementação de um filtro de pacotes inteligente para dispositivos de internet das coisas. In *Proceedings of the 40th Brazilian Symposium on Computer Networks and Distributed Systems*, pages 238–251, Porto Alegre, RS, Brasil. SBC. DOI: 10.5753/sbrc.2022.222301.

Filus, K., Domańska, J., and Gelenbe, E. (2021). Random neural network for lightweight attack detection in the iot. In Calzarossa, M. C., Gelenbe, E., Grochla, K., Lent, R., and Czachórski, T., editors, *Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 79–91, Cham. Springer International Publishing. DOI: 10.1007/978-3-030-68110-4$_5$.

Fontugne, R., Borgnat, P., Abry, P., and Fukuda, K. (2010). Mawilab: Combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking. In *Proceedings of the 6th International Conference*, pages 1–12. DOI: 10.1145/1921168.1921179.

Georgoulias, D., Pedersen, J. M., Falch, M., and Vasilomanolakis, E. (2023). Botnet business models, takedown attempts, and the darkweb market: A survey. *ACM Comput. Surv.*, 55(11). DOI: 10.1145/3575808.

Hafeez, I., Antikainen, M., Ding, A. Y., and Tarkoma, S. (2020). Iot-keeper: Detecting malicious iot network activity using online traffic analysis at the edge. *IEEE Transactions on Network and Service Management*, 17(1):45–59. DOI: 10.1109/TNSM.2020.2966951.

Jan, S. U., Ahmed, S., Shakhov, V., and Koo, I. (2019). Toward a lightweight intrusion detection system for the internet of things. *IEEE Access*, 7:42450–42471. DOI: 10.1109/ACCESS.2019.2907965.

Khraisat, A. and Alazab, A. (2021). A critical review of intrusion detection systems in the internet of things: techniques, deployment strategy, validation strategy, attacks, public datasets and challenges. *Cybersecurity*, 4(1):18. DOI: 10.1186/s42400-021-00077-7.

Klint, C. (2021). These are the top risks for business in the post-covid world. Available at:`https://www.weforum.org/agenda/2021/01/building-resilience-in-the-face-of-dynamic-disruption/`.

Kovacs, E. (2020). Critical vulnerability could have allowed hackers to disrupt traffic lights. *SecurityWeek*. Available at:`https://www.securityweek.com/critical-vulnerability-could-have-allowed-hackers-disrupt-traffic-lights/`.

Lakshmanan, R. (2022). Mantis botnet behind the largest https ddos attack targeting cloudflare customers. Available at:`https://thehackernews.com/2022/07/mantis-botnet-behind-largest-https-ddos.html`.

Manocchio, L. D., Layeghy, S., and Portmann, M. (2022). Network intrusion detection system in a light bulb. In *2022 32nd International Telecommunication Networks and Applications Conference (ITNAC)*, pages 1–8. DOI: 10.1109/ITNAC55475.2022.9998371.

McLennan, M. (2021). The global risks report 2021 16th edition. Available at:`https://www.weforum.org/reports/the-global-risks-report-2021`.

McMillen, D. (2021). Internet of threats: Iot botnets drive surge in network attacks. Available at:`https://securityintelligence.com/posts/internet-of-threats-iot-botnets-network-attacks/`.

Mirsky, Y., Doitshman, T., Elovici, Y., and Shabtai, A. (2018). Kitsune: an ensemble of autoencoders for online network intrusion detection. *Network and Distributed Systems Security (NDSS) Symposium*. DOI: 10.48550/arXiv.1802.09089.

Monici, P., Fernandes, G., Guibo, C., Bertoli, G., Santos, A., and Jr., L. P. (2022). T800: ferramenta de firewall e benchmark para iot. In *Anais Estendidos do XL Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 33–40, Porto Alegre, RS, Brasil. SBC. DOI: 10.5753/sbrc$_e$$stendido$.2022.222753.

Mudgerikar, A., Sharma, P., and Bertino, E. (2019). E-spion: A system-level intrusion detection system for iot devices. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, Asia CCS '19, page 493–500, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/3321705.3329857.

Murshed, M. G. S., Murphy, C., Hou, D., Khan, N., Ananthanarayanan, G., and Hussain, F. (2019). Machine learning at the network edge: A survey. *ACM Computing Surveys (CSUR)*, 54:1–37. DOI: 10.1145/3469029.

Niedermaier, M., Striegel, M., Sauer, F., Merli, D., and Sigl, G. (2019). Efficient intrusion detection on low-performance industrial iot edge node devices.

Qin, T., Wang, B., Chen, R., Qin, Z., and Wang, L. (2019). Imlads: Intelligent maintenance and lightweight anomaly detection system for internet of things. *Sensors*, 19(4). DOI: 10.3390/s19040958.

Ren, H., Anicic, D., and Runkler, T. A. (2021). Tinyol:

Tinyml with online-learning on microcontrollers. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. DOI: 10.1109/IJCNN52387.2021.9533927.

Roesch, M. (1999). Snort - lightweight intrusion detection for networks. In *Proceedings of the 13th USENIX Conference on System Administration*, LISA '99, page 229–238, USA. USENIX Association. Available at:`https://www.usenix.org/legacy/publications/library/proceedings/lisa99/full_papers/roesch/roesch.pdf`.

Saha, S. S., Sandha, S. S., and Srivastava, M. (2022). Machine learning for microcontroller-class hardware: A review. *IEEE Sensors Journal*, 22(22):21362–21390. DOI: 10.1109/JSEN.2022.3210773.

Schizas, N., Karras, A., Karras, C., and Sioutas, S. (2022). Tinyml for ultra-low power ai and large scale iot deployments: A systematic review. *Future Internet*, 14(12). DOI: 10.3390/fi14120363.

Smith, S. (2022). Out of gas: A deep dive into the colonial pipeline cyberattack.

Soe, Y. N., Feng, Y., Santosa, P. I., Hartanto, R., and Sakurai, K. (2020). Implementing lightweight iot-ids on raspberry pi using correlation-based feature selection and its performance evaluation. In Barolli, L., Takizawa, M., Xhafa, F., and Enokido, T., editors, *Advanced Information Networking and Applications*, pages 458–469, Cham. Springer Intl. Publish.. DOI: 10.1007/978-3-030-15032-7$_3$9.

Utomo, D. and Hsiung, P.-A. (2019). Anomaly detection at the iot edge using deep learning. In *2019 IEEE International Conference on Consumer Electronics - Taiwan (ICCE-TW)*, pages 1–2. DOI: 10.1109/ICCE-TW46550.2019.8991929.

Viegas, E., Santin, A. O., and Abreu Jr, V. (2021). Machine learning intrusion detection in big data era: A multi-objective approach for longer model lifespans. *IEEE Transactions on Network Science and Engineering*, 8(1):366–376. DOI: 10.1109/TNSE.2020.3038618.

Yadav, T. and Rao, A. M. (2015). Technical aspects of cyber kill chain. In Abawajy, J. H., Mukherjea, S., Thampi, S. M., and Ruiz-Martínez, A., editors, *Security in Computing and Communications*, pages 438–452, Cham. Springer International Publishing. DOI: 10.1007/978-3-319-22915-7$_4$0.