Detection and Mitigation of Attacks at the Edge of IoT Networks Using Deep Learning and P4

Antônia Mayara de A. da Silva
Postgraduate Program in Computing
Federal University of Ceará (UFC)
Quixadá, CE, Brazil
mayaraalmeida@alu.ufc.br

Michel Sales Bonfim
Postgraduate Program in Computing
Federal University of Ceará (UFC)
Quixadá, CE, Brazil
michelsb@ufc.br

Arthur de Castro Callado
Postgraduate Program in Computing
Federal University of Ceará (UFC)
Quixadá, CE, Brazil
arthurcallado@gmail.com

Enyo José T. Gonçalves
Postgraduate Program in Computing
Federal University of Ceará (UFC)
Quixadá, CE, Brazil
enyo@ufc.br

Abstract

Context: The concern for security in IoT networks is becoming increasingly common, considering the accelerated use of these devices in recent years.
Problem: Traditional methods for detecting attacks in IoT networks are ineffective due to device heterogeneity. While learning approaches show promise in threat detection, the computational limitations of IoT devices make it challenging to implement these solutions.
Solution: This work proposes a deep learning solution for IoT attack detection, focusing on efficient threat mitigation using the Programming Protocol-independent Packet Processors (P4) language, adaptable feature extraction for algorithm inference, and execution on SBC devices.
SI theory: This work is based on General Systems Theory and Machine Learning Theory, emphasizing integrating components in IoT networks and learning for attack detection.
Methods: This study uses a case study to evaluate the effectiveness of deep learning algorithms in detecting attacks in IoT networks, adopting a descriptive and quantitative approach with metrics like accuracy and F1 score.
Results: In our work, the Long Short Term Memory (LSTM), Convolutional Neural Network (CNN), and Gated Recurrent Unit (GRU) algorithms achieved accuracy rates of 96.9%, 95.6%, and 93.9%, respectively. However, the CNN model was selected for the final implementation due to its superior inference time and resource consumption performance.
Contributions and impact in the IS area: This work contributes to Information Systems by integrating deep learning and P4 programming for attack detection and mitigation in IoT networks, enhancing security, and promoting further research in the field.

CCS Concepts

• **Security and privacy → Intrusion detection systems**.

Keywords

Attack detection, P4 language, IoT, Deep learning, Network security

1 INTRODUCTION

The Internet of Things (IoT) comprises billions of devices connected to the Internet. Currently, these devices can be found everywhere and in different areas of application: they can be found in homes, offices, transportation, healthcare, urban mobility, and more . The number of IoT networks has been growing quickly, but with this growth comes a concern: the technology still needs advancements to ensure security in device communication [22].

The vast amount of data and the popularization of IoT networks make them attractive targets for malicious attacks. [25] mentions that IoT systems (IoTS) are complex, and devices are generally highly heterogeneous. Due to limited computing and energy resources, IoT devices cannot support robust security structures. Therefore, ensuring security across a broad attack surface in IoT is a challenge [5]. Due to software vulnerabilities that compromise the confidentiality, integrity, and availability of information, it is currently considered essential to perform static and dynamic analyses of the entire information system. Previously, the analysis focused on individual files or software components [15].

According to [2], finding a technique capable of monitoring IoT devices and providing an intelligent solution for attacks is an ongoing necessity. Machine learning (ML) and deep learning (DL) are potent data analysis methods for analyzing data, learning IoT network traffic patterns, and understanding how components and devices interact and the data transferred between them. Traditional ML algorithms have been used in security solutions; however, these algorithms rely heavily on attack signatures to achieve good results. On the other hand, DL models require less of this dependency, as they work by discovering the most important features of the data and recognizing patterns [21].

With the increasing use of these approaches, it becomes necessary to assess the performance of different algorithms accurately and compare the results to find the best technique for the problem. Moreover, research using datasets that provide vast network flows tends to yield more consistent results with a real IoT network. Besides, the algorithm's test performance should be as close as possible to when it is implemented. Finally, using P4 programming is a relatively new approach that has been the subject of research

in recent years. Although it has been explored in few studies to mitigate attacks, such as in studies like [20] and [12], where the work focused only on Denial of Service (DoS) and Distributed Denial of Service (DDoS) attacks, its adaptability and achieved results make it an excellent alternative for IoT networks.

In this work, we conduct a comparative analysis of different Deep Learning algorithms — CNN, LSTM and GRU — with a focus on attack detection. The initial analysis was carried out on Google Colaboratory, where the algorithms were trained and tested. Among the models evaluated, the CNN algorithm, in general, outperformed the others in metrics such as accuracy, precision, recall, and F1 score. The selection of DL algorithms aligns with the need for anomaly-based systems, allowing the detection of various types of attacks. Comparing the three algorithms contributes to the field of IS by providing insights into their effectiveness in various contexts. In addition, this work contributes to privacy, ethics and information security, as well as innovation and digital transformation, by presenting a solution that automates security processes in IoT networks.

The proposed solution includes, in addition to the DL algorithm for detection, a P4 application responsible for mitigating the packets by blocking those identified as attacks. P4 also serves a fundamental role in providing the features needed for the DL model's classification. To enable the execution of the solution at any point in the network, it was necessary to convert the DL algorithms into a lighter and more compact format using OpenVINO[1]. This new format makes the algorithms compatible, including with Single Board Computer (SBC) operating at the edge of IoT networks, considering that these devices have computational limitations that prevent the execution of the algorithms in their original format (TensorFlow SavedModel). The three models were reevaluated, now converted and implemented on a Raspberry Pi 4. To allow the execution of the solution, a controller was developed that receives the features extracted with P4 and enables packet classification, as well as adding rules to the P4 tables to block packets classified as attacks. In the new tests, accuracy, inference time for the classification of each model, and CPU and RAM consumption of the device were evaluated. In the accuracy evaluation, the algorithms showed similar results, ranging from 93% to 96%, with LSTM achieving the best result. Regarding execution time and resource consumption, the CNN algorithm was superior to the others.

This work is organized as follows: Related work is discussed in Section 2. The methodology is described in Section 3. The proposed architecture is presented in Section 4. Results, discussions, and the performance of the proposed solution are described in Section 5. Finally, Section 6 discusses the conclusion and future work.

2 REFERENTIAL THEORY

This section will present the main concepts addressed in this work.

2.1 Internet of Things

According to [8], the concept of a network of smart devices was first introduced and discussed in 1982, with a modified Coca-Cola machine at Carnegie Mellon University becoming the first internet-connected device capable of reporting its stock and whether newly loaded beverages were cold. Kevin Ashton is known as the inventor of the term "Internet of Things" to describe a system where the Internet is connected to the physical world through sensors.

With the increasing use of IoT, concerns about the security of these networks are now well known. The main pillars of Information Security primarily aim to ensure the confidentiality, integrity, and accountability of systems and data. In [24], it is stated that, however, traditional solutions have significant limitations when applied to IoT devices. This is because the computational power of these devices is, in most cases, insufficient for long-duration operations. Furthermore, scalability concerns arise due to the many connections established by IoT devices.

2.2 Attacks

IoT systems have limited computational resources and storage, which renders traditional intrusion detection systems inefficient due to the heterogeneity of networks and the increasing vulnerabilities arising from the growth of IoT devices. The two main intrusion detection techniques in IoT systems are signature-based and anomaly-based methods. According to [1], the signature-based technique works well for known attacks. However, a drawback of this approach is its inability to detect zero-day or unknown attacks, as it relies on signatures of previously identified attacks. The anomaly-based technique is preferred as it analyzes normal traffic patterns and triggers an alert or blocks traffic when an abnormal pattern is detected.

Attacks on IoT devices generally occur at the physical, application, or network layer. At the physical layer, attacks can control the device by modifying its hardware. At the application layer, attacks are executed using software applications and include phishing, trojans, ransomware, worms, etc. The network layer is where most attacks occur, including eavesdropping, man-in-the-middle attacks, DoS or DDoS attacks, exploitation attacks, spoofing attacks, among others [14].

2.3 Deep Learning

An alternative for attack detection in IoT networks is the use of learning algorithms. Deep learning (DL) is a powerful form of machine learning that enables computers to solve perception problems. Deep learning methods, such as deep artificial neural networks, make use of multiple processing layers to observe patterns and structures in large datasets. Each layer learns a concept from the data, forming a sequence that repeats for the subsequent layers; the higher the level, the more abstract the discovered concepts are [17].

The use of ML and DL in various IDS (Intrusion Detection Systems) models is a powerful technique for detecting attacks in Internet of Things networks and recognizing new types of intrusions to secure access to these networks. The need for creating an intrusion detection system to identify attacks quickly and automatically increases as IoT usage becomes more widespread and the large volume of data becomes a frequent target for attackers [13].

2.4 P4 Language

In the context of mitigating identified attacks, we can observe the use of data plane programming. Currently, the most widely

[1]https://docs.openvino.ai/2025/index.html

used programming language and concept for the data plane are Protocol-Independent Packet Processors (P4). Initially introduced as research in 2014, P4 is now developed and standardized by the P4 Language Consortium. It is supported by various software and hardware platforms and has been widely adopted both in research and industry [11].

According to [4], P4 is a high-level language for programming the data plane. P4 can work in conjunction with SDN control protocols such as OpenFlow. Among the functions that can be configured with P4, programmers have the ability to change how switches process packets after deployment. Switches are not tied to any specific network protocol, and programmers can detail packet processing functions independently of the hardware used.

3 RELATED WORK

Several works in the literature aim to detect attacks by monitoring network traffic. In [18], an Intrusion Detection and Prevention System (IDPS) was developed for an IoT network, employing Complex Event Processing (CEP) technology to analyze traffic and identify potential attacks on devices using MQTT and CoAP protocols. However, traditional IDPS methods have limitations in both detection accuracy and adaptability to different devices. In light of this, recent research has proposed the use of Machine Learning for traffic classification, considering that ML can automate this verification, delivering good results in the face of device diversity and increasing detection speed [2].

[19] presented an intelligent detection system for detecting DDoS attacks of different magnitudes. The proposed method functions as a sensor that can be installed at any point in the network and can classify traffic using a strategy with the Random Forest algorithm, which performs classifications by analyzing traffic samples collected from network devices.

In addition to traditional ML techniques, recent studies have introduced deep learning to detect and mitigate attacks. [6] developed a DDoS attack detection system in SDN environments, known as DDoSNet. This method employs Deep Learning techniques, combining Recurrent Neural Network (RNN) with the autoencoder algorithm. The model was trained using the CICDDoS2019 dataset[2], which contains a diverse range of DDoS attacks. The experiments demonstrated accuracy of 99.81%, precision of 99.77%, recall of 99.89%, and F1 score of 99.83%, results that surpass conventional machine learning solutions for security in cloud services.

In the context of IoT, [10] compares the performance of various machine learning approaches for accurately detecting attacks and anomalies in IoT systems. The machine learning algorithms used included Logistic Regression (LR), Support Vector Machine (SVM), Decision Tree (DT), Random Forest (RF), and Artificial Neural Network (ANN). The conducted experiments indicated RF as the best-performing algorithm with an accuracy of 99.4%. This study utilized a dataset with approximately 450,000 samples and encompassed seven different types of attacks.

In [23], a new security and attack detection system is presented, using a DL model to detect malicious devices efficiently. The proposed mechanism employs a Convolutional Neural Network (CNN) for precise data extraction and utilizes the Long Short-Term Memory (LSTM) model for classification. Data were collected from a set of IoT devices, twenty of which were Raspberry Pi infected with various malicious codes, and only three devices with network traffic were considered normal. Eight different types of attacks were used in the experiment, and the method achieved an accuracy of 96%.

[20] presented a set of Machine Learning-based DDoS Attack Detection (DAD) strategies and the integration of SDN state data planes, with a particular focus on SYN flood attacks in the Transmission Control Protocol (TCP). The authors introduce stateful data planes to reduce latency in data forwarding and quicker access to critical network resources for ML algorithms, resulting in a faster response and reduced attack-induced damage. Four ML algorithms, RF, KNN, SVM, and ANN, were evaluated in this research for DDoS attack detection, and the test results showed that attack detection can achieve accuracy, precision, recall, and F1 score above 98% in most cases, with a time reduction to less than 200 μs when P4 is used for feature extraction.

Combining different algorithms to achieve improved results is also addressed in [12]. This article utilized the Long Short-Term Memory and Naive Bayes algorithms to handle Denial of Service (DoS) attacks in P4-based Software-Defined Networks. The developed model achieved an accuracy of 88% on the SDN-DL dataset, 98% on NSL-KDD, and 96% on CICIDS2017. Finally, the authors compared the developed technique with machine learning and deep learning methods.

In this regard, several studies have focused on machine learning algorithms for attack detection, while research combining deep learning algorithms with the P4 language remains more scarce. Most of the works presented collect data and perform attack detection outside the IoT environment. Typically, the data is forwarded to the cloud or a desktop where the learning model, whether machine learning or deep learning, is located, and from there, incoming traffic is classified. The few identified studies that combine Deep Learning and P4 have only been applied to DDoS attacks, excluding other types of attacks from their experiments. With the growing adoption of these core concepts, research involving deep learning models and using the P4 language in attack mitigation is essential to enhance IoT network security.

In this work, both detection and mitigation are performed directly on the IoT device, eliminating the need to send packets for external classification, which could result in higher latency. Coverage of various types of attacks is also an important feature of this solution, as the algorithms were trained on a dataset containing 14 types of attacks. Besides, the proposed solution demonstrates adaptability to different devices, with an integrated development approach that allows it to operate at any point in the IoT network.

4 MATERIALS AND METHODS

This work presents a comparative study between three deep learning algorithms, evaluating their results using cross-validation. The selected metrics were accuracy, precision, recall, and F1 score. Additionally, we present an architecture that combines a deep learning algorithm with the P4 language for detecting and mitigating attacks in IoT networks. We also assess CPU and memory consumption

[2]https://www.unb.ca/cic/datasets/ddos-2019.html

while running the solution on an SBC device. The following subsections provide further details on the methods and tools used.

Figure 1 illustrates the stages necessary for developing the methodology. The first stage involved defining the models and hardware used. In the second stage, the selected models were tested and evaluated. The third stage focused on architecture planning, defining both functionality and employed technologies. In the fourth and final stage, the proposed solution was developed and tested.

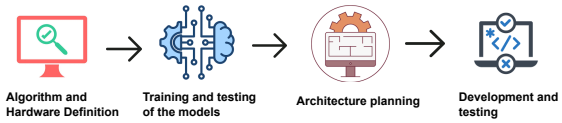

Figure 1: Employed methodology.

4.1 Algorithm and Dataset Selection

The selection of the algorithms was based on a literature review to identify which deep learning algorithms achieved the best results in traffic classification. The CNN was chosen for its high capability to identify specific patterns in network packets, extracting important features without the need to model long-term temporal dependencies.

The choice of the LSTM algorithm was due to its ability to analyze data patterns over time. Since network traffic occurs sequentially, LSTM can capture relationships between packets and detect anomalies. Additionally, this algorithm handles complex data well without losing important information.

GRU is a simplified alternative to LSTM, with fewer parameters and lower computational consumption. Including this algorithm allows for an evaluation of whether a lighter architecture can achieve performance similar to LSTM, providing a comparison between computational cost and effectiveness in detecting malicious traffic.

Although LSTM and GRU are both recurrent neural networks, they handle memory differently, which can impact their final performance. In our study, LSTM and GRU are used to assess how these differences affect traffic detection in IoT networks. Despite their similarities, variations in memory management may lead to different results in certain applications.

After defining the algorithms, the research was focused on identifying data sets with a focus on attacks on IoT networks. Among the datasets found, Edge-IIoTset [7] was chosen. The choice for Edge-IIoTset was due to the scope of dataset. This dataset was developed to validate proposed security solutions for IoT networks and employs more than ten types of IoT devices to collect traffic, resulting in 14 types of attacks, including DoS/DDoS, man in the middle and malware attacks. Edge-IIoTset uses the Zeek tool to extract features and makes the .pcap files of each capture available. An important characteristic of this dataset is its coverage of the seven layers in defining the scenarios. The main technologies used in IoT were implemented at each layer, seeking to develop accurate and complete traffic.

4.2 P4Pi

Currently, various devices support the P4 language, including programmable network interface cards (NICs), bare-metal switches, and FPGAs. Additionally, some platforms facilitate the development and testing of P4 programs, such as P4Pi [16].

P4Pi is a platform developed by the P4 Education Working Group with the goal of popularizing the teaching and use of the P4 language on low-cost hardware, such as the Raspberry Pi. Although the initial objective was to facilitate learning P4, its development on an SBC device emerges as an alternative for implementing programmable switches on devices with computational limitations. Furthermore, P4Pi supports various networking technologies, such as stateful forwarding, packet filtering, NAT (Network Address Translation), and load balancing. This platform leverages the DPDK library, which assists in packet processing and optimization in the data plane, as well as the T4P4S[3] compiler and the P4Runtime[4] API.

4.2.1 T4P4S. We chose the T4P4S compiler for this experiment, a framework that functions as a target-independent compiler for protocol-independent packet processors. In its initial version, P4Pi includes T4P4S as the default compiler, and it is compatible with various versions (including $P4^{14}$ and $P4^{16}$. It supports a DPDK-based software switch. There is also the P4C compiler, but for P4Pi devices, it exhibited a lower performance when compared to T4P4S [16].

4.2.2 P4Runtime. The P4Runtime API is an interface developed for the control plane to control the behavior and elements of the data plane, acting as a link between the two layers. It is used for state manipulation and adding rules to tables. Its creation aimed to make communication between the controller and P4 switches easier and more abstract, enabling the execution of more complex monitoring operations.

5 P4-DL Architecture

The proposed architecture was developed to detect and mitigate malicious traffic, as illustrated in Figure 2. Data traffic can originate from the broader internet via the Ethernet interface or from devices connected to the Wi-Fi interface. Regardless of the source, the traffic must pass through the P4 switch on the Raspberry Pi 4. The necessary features are extracted by switch and sent to the decision module, which is composed of the DL model that achieved the best performance in the tests and the controller responsible for adding the rules. If the traffic is classified as benign, it continues normally, but if it is classified as an attack, a new rule is added to ensure the discarding of new malicious packets.

Deep Learning algorithms are complex and consume a lot of computational resources. To execute these algorithms in limited environments, such as IoT, the OpenVINO platform is a commonly used. In this work, OpenVINO will be utilized to optimize Deep Learning models and run them on Raspberry Pi. It is worth mentioning that Raspberry PI 4 supports OpenVINO Runtime[5].

[3] https://github.com/P4ELTE/t4p4s
[4] https://p4.org/p4-spec/p4runtime/main/P4Runtime-Spec.html
[5] https://docs.openvino.ai/2022.3/home.html

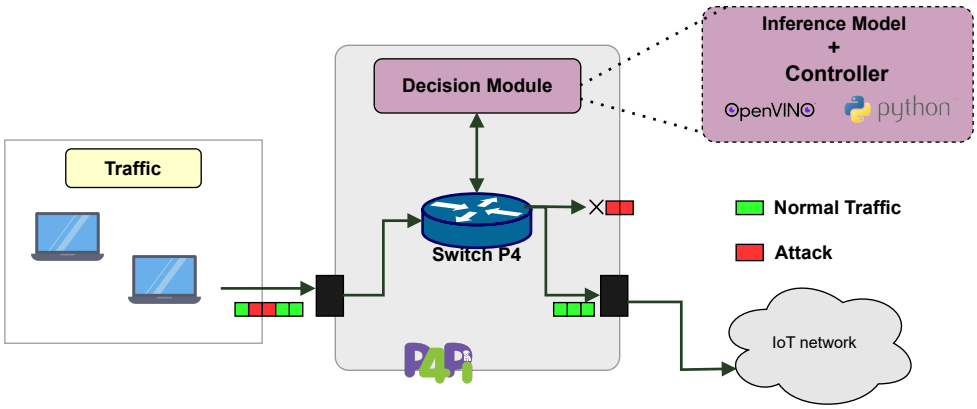

Figure 2: Architecture model.

5.1 Hyperparameter definition

For training and testing the three algorithms, the hyperparameters were defined in a similar way. To perform cross-validation, the dataset was divided into 5 folds, with parameters set to shuffle the data before splitting and the random_state set to 42.

In all three models, an initial layer was created to filter the data input and map the main characteristics, followed by a layer for dimensionality reduction. The CNN also has a convolutional layer, a pooling layer for more complex data extraction, a Flatten layer to convert the features into a unidirectional vector and a dense layer with 15 output units. The LSTM model, in addition to the two initial layers, has an LSTM layer with 50 units for processing the data sequence, a dropout layer with a value set at 0.5 to prevent overfitting during the training, a Flatten layer, a dense layer with 64 input units for assigning weights and a dense layer with 15 output units. The GRU, like the others, still contains two GRU layers with 50 units each, interspersed with two dropout layers with values fixed at 0.5, a Flatten layer and a dense layer with 15 dropout units. exit.

To compile the three models during the training process, the ADAM optimizer was chosen. As for the loss function, the function *sparse_categorical_crossentropy* was defined, this option is widely implemented to deal with data that contains many classes.

5.2 P4 implementation

The use of the P4 programming in this solution aims to ensure processing speed and network flexibility. Our P4 program has been implemented to extract the features required by the DL algorithm and to support rules for blocking malicious traffic.

We extended a traffic filter code available in the official P4Pi repository [9] to create our mitigation mechanism. Then, the code was compiled and validated using the IPerf tool, where a server was created on the Ethernet interface and a client connected to the Wi-Fi interface. In a P4 implementation, it is common to define headers only up to layer four at most. In our implementation, we defined the headers related to the application algorithms by accessing the payloads of the transport layer protocols.

For feature extraction, the data was sent to the classifier via digests. The extraction is performed with the help of in-band network telemetry (INT), using the eXport Data (XD) mode of operation. INT-XD works by sending telemetry data directly to the controller without altering the original packet [3]. In this mode, each node is responsible for creating the instructions and collecting the necessary data. To avoid overhead, a digest was defined for each protocol, extracting the corresponding fields required for the inference model. In the switch, the structures for each digest are defined with the protocol header information, and an action is created to collect and prepare the data for sending. The sending is done conditionally: for example, if the packet is TCP, the switch checks the destination port. If the port matches the default port for MQTT, HTTP, or MBTCP, the switch will send, in addition to the TCP digest, the corresponding protocol digest. If the destination port does not match any of these defined ports, the switch will send only the TCP protocol digest. The same type of check occurs with UDP. When the packet is ICMP, only the digest for this protocol is sent, with no additional verification required.

Figure 3 presents the structure of each table in the P4 switch. The three tables receive exact-type data and have two actions associated with them. During the rule creation process, the tables are populated according to the protocol of the verified packet.

One of the fundamental parts of the P4 code is Apply. When starting, it initializes a variable with a value of 0 and a check is made to see if the IPv4 header is valid and if the TTL is not zero. Once both conditions are met, it checks whether the ICMP packet is valid and whether the variable is still zero. After that, it searches the rules table for any rule that matches the analyzed header; if positive, updates the variable to 1. The exact process is repeated for TCP and UDP headers in their respective tables. Finally, check the value of the variable; if it is 1, the packet is discarded. Additionally, a Python controller was developed to receive the inference results and add rules to the tables. In Figure 4, we present an excerpt from the Apply block of code.

ICMP table

```
table idps_icmp_exact {
    key = {
        hdr.ipv4.srcAddr: exact;
        hdr.ipv4.dstAddr: exact;
    }
    actions = {
        drop;
        NoAction;
    }
    size = 1024;
    default_action = NoAction();
}
```

UDP table

```
table idps_udp_exact {
    key = {
        hdr.ipv4.srcAddr: exact;
        hdr.ipv4.dstAddr: exact;
        hdr.udp.srcPort:  exact;
        hdr.udp.dstPort:  exact;
    }
    actions = {
        drop;
        NoAction;
    }
    size = 1024;
    default_action = NoAction();
}
```

TCP table

```
table idps_tcp_exact {
    key = {
        hdr.ipv4.srcAddr: exact;
        hdr.ipv4.dstAddr: exact;
        hdr.tcp.srcPort:  exact;
        hdr.tcp.dstPort:  exact;
    }
    actions = {
        drop;
        NoAction;
    }
    size = 1024;
    default_action = NoAction();
}
```

Figure 3: Table structure.

```
apply {
    bit<8> dropped = 0;
    if (hdr.ethernet.isValid() && hdr.ethernet.etherType ==
    0x0806) {
        if (idps_arp_exact.apply().hit) {
            dropped = 1;
        }
    } else if (hdr.ipv4.isValid()) {
        if (hdr.tcp.isValid() && dropped == 0) {
            if (idps_tcp_exact.apply().hit) {
                dropped = 1;
            }
        } else if (hdr.udp.isValid() && dropped == 0) {
            if (idps_udp_exact.apply().hit) {
                dropped = 1;
            }
        } else if (hdr.icmp.isValid() && dropped == 0) {
                if (idps_icmp_exact.apply().hit) {
                dropped = 1;
            }
        }
    }
```

```
if (dropped != 1) {
        if (hdr.ipv4.protocol == 1) {
            send_digest_icmp();
        }else if (hdr.ipv4.protocol == 17 && hdr.udp.dstPort
            == 53){
            send_digest_udp();
            send_digest_dns();
        }else if (hdr.ipv4.protocol == 6 && hdr.tcp.dstPort ==
            80){
            send_digest_tcp();
            send_digest_http();
        }else if (hdr.ipv4.protocol == 6 && hdr.tcp.dstPort ==
            1883 && hdr.mqtt.msgType == MQTT_PUBLISH){
            send_digest_tcp();
            send_digest_mqtt_publish();
        }...
        }

    standard_metadata.egress_port =
    (standard_metadata.ingress_port + 1) % 2;
```

Figure 4: Apply block.

5.3 Decision Module

Figure 5 illustrates how the solution's decision module operates. This module corresponds to a Python program that communicates with the P4 switch through P4Runtime. After feature extraction by the switch, the digests are received by the feature collector and converted into the format required by the inference model. Subsequently, the packets classified as attacks are directed to the rule manager, which adds the rules to the correspondence table.

5.3.1 Feature Collection. In this submodule, the received digests are separated into variables defined for each field. At this stage, when necessary, the data is converted to decimal format. With the correct format, the information is stored in a matrix and then consumed by the inference model. Figure 6 shows an example of how these digests work.

5.3.2 Inference Module. The inference module runs on a new thread, with each row of the received matrix representing a new input to the model. Before the inference is executed, data processing occurs, including column conversion and missing data filling. The data related to the source and destination IP addresses and ports are

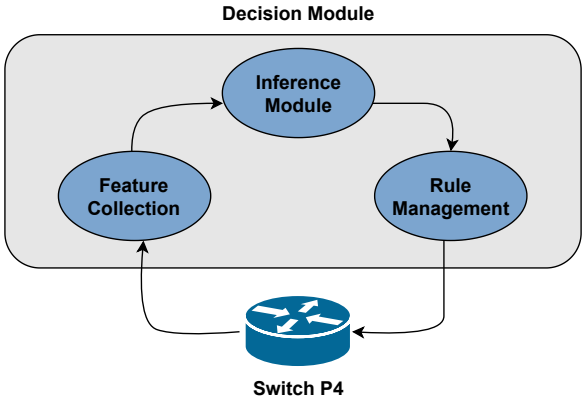

Figure 5: Decision Module

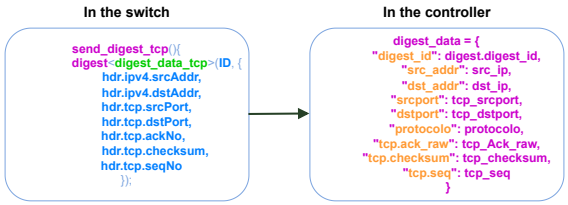

Figure 6: Example of digest operation.

stored before the inference begins to be used later if the model classifies the packet as an attack.

5.3.3 Rule Management. Figure 7 presents an example of rule creation. First, the classifier must detect the packet as an attack. Then, the driver retrieves the necessary data to create the rules, which are then inserted into the switch's rule table. For creating an ICMP rule, the source and destination IP addresses are required. For UDP and TCP rules, in addition to the IP addresses, the source and destination ports are also necessary. By default, each table supports 1024 rules. As this limit is reached, older rules are deleted to allow for the insertion of new ones.

6 EVALUATION AND DISCUSSION

Initially, we trained and tested three algorithms—CNN, LSTM, and GRU—and analyzed the results of the Accuracy, Precision, Recall, and F1 score metrics. For this algorithm selection phase, we used the Edge-IIoTset dataset. The selected algorithms were trained according to pre-established parameters and evaluated using the K-fold cross-validation technique. Next, we converted the algorithms using the OpenVINO tool to enable their execution on the Raspberry Pi. We ran the three algorithms and re-evaluated accuracy, adding assessments of CPU and RAM consumption. Finally, we selected the algorithm with the best results for the final version of the architecture.

6.1 Cross-validation

This technique randomly divides the dataset into k iterations, where each iteration tests the model with the reserved portion and uses the remainder for training. Therefore, the tests are performed with data the model has not seen before. For this stage, the number of defined iterations was 5.

Figure 8 shows the results of the three algorithms when evaluating accuracy. In the first fold, the CNN and LSTM algorithms presented similar results, above 99%, while the GRU algorithm started with results slightly lower than the others, with 95%. In the following folds, the GRU algorithm saw an increase in its results and approached CNN and LSTM with a value greater than 99%.

Figure 9 presents the results of the precision metric of the CNN, GRU and LSTM algorithms. In the first folds, the CNN and LSTM algorithms achieved similar values, while the GRU presented lower precision with results close to 98%. In the other tests, the results followed the same pattern, with CNN presenting superior results to the other two algorithms.

The results of the Recall metric are presented in Figure 10. As identified in previous tests, GRU remained the algorithm with the lowest values and the CNN and LSTM algorithms obtained very close results. The results achieved showed 94% recall in the GRU algorithm and 96% in the CNN and LSTM algorithms.

Finally, Figure 11 presents the results of the F1 score metric. The CNN algorithm started testing with values around 98% and remained stable throughout the 5 folds. The same was observed in the LSTM algorithm, where the results of all tests obtained values between 98% and 99%. The GRU algorithm presented results in the range of 96% to 97% in the 5 interactions.

Table 1 presents the average values of the results of the three algorithms.

Table 1: Average of algorithm results.

Algorithms	Acc (%)	Pre (%)	*Rec* (%)	F1 *score* (%)
CNN	99.9	99.8	97.7	98.7
GRU	99.0	98.8	94.2	96.5
LSTM	99.7	99.5	97.5	98.1

In general, the three algorithms presented similar results when executing these steps. However, it is important to highlight that the CNN algorithm presented better results than the others in three of the four metrics evaluated, while the GRU algorithm obtained the lowest values in the four metrics collected. The inferior performance of the GRU algorithm can be explained by the combination of the simplicity of its structure and the complexity of IoT traffic data, which requires analysis of local and temporal patterns, areas in which CNN and LSTM excel. Compared to LSTM, GRU is lighter and faster; however, this reduction in complexity can compromise its ability to identify patterns. Hyperparameter tuning could also have contributed to these results; however, we seek to standardize the definition of architectures and tests to allow a fair comparison.

6.2 Experiment Configuration

To evaluate the algorithms on the Raspberry Pi, the switch was configured to extract and send the features to the classifier. The data can be sent to the controller either through mirroring or digests. The choice to send summaries was made to avoid overloading the controller, which receives only the necessary data and not the entire packet.

The developed controller contains the DL algorithms already converted with the help of OpenVINO into the intermediate representation (IR) format. The implementation is divided into two threads: the first receives the digest data and organizes it into a queue, while the second executes the inference model, consumes the data, and creates the rules when necessary.

6.3 Performance Analysis

To evaluate the performance of each algorithm on the Raspberry Pi, the Tcpreplay tool was used. The tests aimed to determine which algorithm presented the highest percentage of correct classifications and the lowest CPU and RAM usage for performing inferences. To reproduce the traffic with Tcpreplay, a capture of normal Message Queuing Telemetry Transport (MQTT) traffic and a capture of Port Scanning attacks from the Edge-IIoTset dataset, containing a total of 15,278 packets, were used.

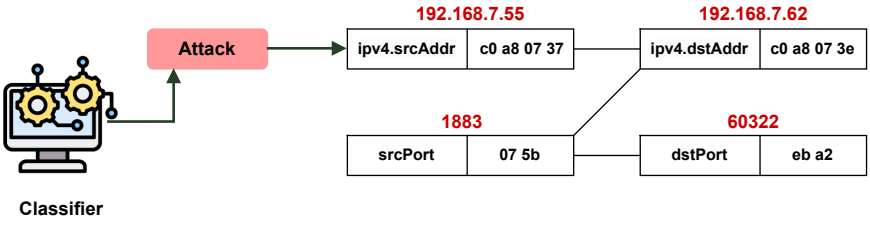

Figure 7: Example of Rule Creation

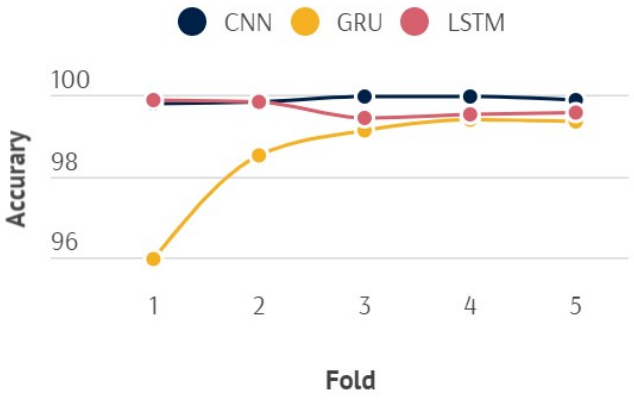

Figure 8: Accuracy of the three algorithms.

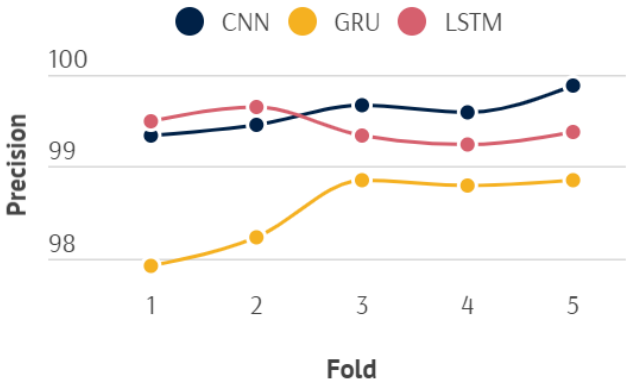

Figure 9: Precision of the three algorithms.

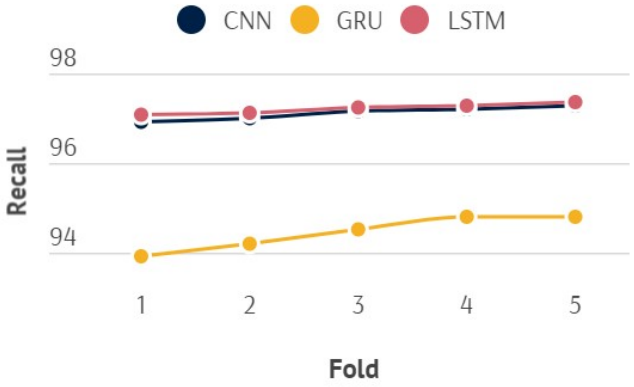

Figure 10: Recall of the three algorithms.

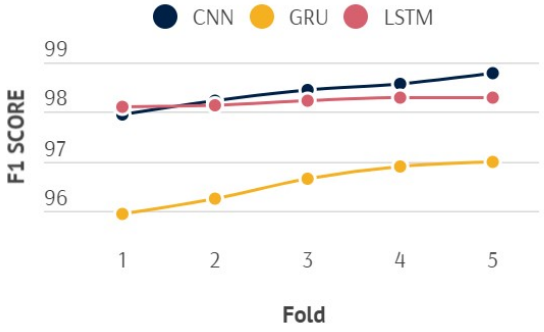

Figure 11: F1 score of the three algorithms.

Table 2: Evaluation on the Raspberry Pi.

Metrics	CNN	LSTM	GRU
Accuracy (%)	95.60	96.93	93.95
Total time (s)	7.128	12.815	12.230

After conducting the tests, the CNN, LSTM, and GRU algorithms achieved similar results in the OpenVINO implementation. Table 2 presents the results of the accuracy and total inference time metrics for the generated traffic. The LSTM algorithm reached a correct classification rate of 96.93%, while the CNN achieved 95.60% accuracy in predictions, and the GRU correctly classified 93.95% of the packets.

When analyzing the waiting time for each algorithm to finish the classification, CNN was the algorithm that finished this process the fastest. In LSTM, the package took around 12.815 s to complete the package inference. In GRU, the estimated time to complete this step was 12.230 s. The CNN algorithm completed this step in 7.128 s.

Using the Netdata[6] tool, we collected data on CPU and RAM usage during the tests. As shown in Figure 12, when only the controller is running, the average CPU usage remains around 55%. When the classifier is started, a peak in CPU usage is observed across all three algorithms: in the CNN and GRU algorithms, the peak reaches approximately 80%, while in the LSTM algorithm, the peak reaches up to 90%.

In Figure 12, we also identified an increase in CPU usage when Tcpreplay was started. As expected, there was a significant rise in all three algorithms. The average CPU usage while the CNN algorithm was running stabilized at 85%. For the LSTM, the average CPU usage was 91%. Finally, the GRU algorithm had an average consumption value of 89% during traffic execution.

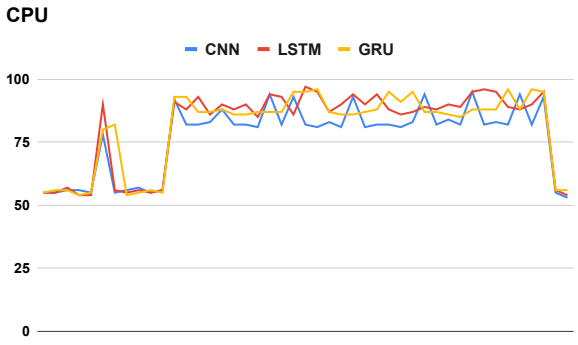

Figure 12: CPU usage.

In summary, the CPU usage peaks and averages reflect the characteristics of each model and the demands of each stage (control, classification, and traffic). CNN, being less complex, consumes less CPU, while LSTM, requiring more sequential processing, is more resource-intensive. GRU, despite being less complex than LSTM, still demands a high level of CPU due to its temporal dependencies.

The execution of the controller also caused an increase in RAM usage on the Raspberry Pi. Figure 13 illustrates how this increase occurred for each algorithm. The CNN and GRU algorithms showed similar growth, with averages of 3.32 and 3.31, respectively, while the LSTM algorithm reached an average of 3.51.

Among the reasons justifying the higher RAM usage by the LSTM algorithm is the model's greater complexity. Unlike CNN and GRU, its internal structure requires more robust processing of inputs to handle both long-term and short-term memory.

Thus, the results demonstrated that, after implementing the algorithms on the Raspberry Pi, the values obtained by the three algorithms did not show significant variation in classification accuracy, although the LSTM was slightly superior to the others. However, the CNN was faster than the other algorithms in critical metrics for many environments: inference execution time and resource consumption (in this work, we evaluated CPU and RAM usage), making it the chosen algorithm for the final implementation of the solution.

[6]https://www.netdata.cloud/

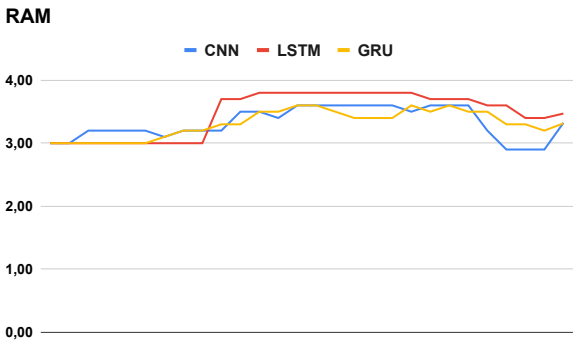

Figure 13: RAM Usage.

The final version with the CNN algorithm showed promising results. Considering that the tests were conducted in a real environment, the time spent on packet classification is minimal, making it suitable for implementation in real-time attack detection. The algorithm's accuracy shows a high percentage of correct predictions. The dataset used contains a considerable variety of attacks, and still, after conversion, the model achieved a result above 95%. During the tests, resource consumption showed some spikes that deserve attention. It is important to emphasize that the implementation of this solution should be done on a device dedicated exclusively for this purpose. The average values are acceptable and did not impact the solution's performance, but they should be monitored in the long term to prevent an increase in latency.

6.4 Threats to validity

The validity of experimental results is a key factor in research involving experimentation. As cited by [26], it is essential to consider and mitigate various threats to validity. Among these, four key aspects should be addressed: internal validity, external validity, construct validity, and conclusion validity. Regarding internal validity, an important consideration is the configuration of hyperparameters. Inadequate configurations can compromise results, leading to unfair comparisons between algorithms. To minimize bias, cross-validation was used for model partitioning and evaluation. Additionally, the reliance on tools like OpenVINO was recognized as a potential influence on the results, especially if its limitations or specific optimizations are not applicable to other scenarios. To mitigate potential threats to external validity, a dataset was selected that includes all traffic generated by IoT network devices, along with 15 distinct types of attacks. The dataset contains both .pcap and CSV files, and its documentation specifies how data classification and labeling were performed. Another important aspect is that the solution allows deployment at different points within an IoT network, increasing its applicability to various scenarios. It can be implemented either in a central location or at the network edge, providing flexibility and adaptability. In our approach, the solution operates between the Broker and IoT devices. Networks such as smart homes and industrial IoT networks typically follow this configuration and could serve as potential deployment scenarios. One limitation is that the algorithms were trained with MQTT

packet data. If the network uses a different protocol, such as CoAP, retraining will be necessary, along with the inclusion of mandatory fields in the P4 switch. For construct validity, the experiment considered multiple evaluation metrics, including standard performance metrics such as accuracy, precision, recall, and F1-score, as well as resource consumption metrics (CPU, RAM, and processing time) to provide a more comprehensive assessment. Finally, regarding conclusion validity, we considered the selection of a limited fraction of the dataset for evaluation on the Raspberry Pi, which may not fully encompass the diversity of attacks available.

7 CONCLUSION

This work addresses one of the main challenges faced by IoT networks: security. Our work is based on General Systems Theory by treating IoT networks. In practice, the solution combines attack detection and mitigation, increasing the security of these systems. We compared different DL algorithms, considering the growing application of these techniques and the need for studies in this area. Additionally, we introduced an architecture for detecting and mitigating attacks by combining the DL algorithm with the P4 language, an emerging language for programming data planes. Three DL algorithms were selected, and their accuracy, precision, recall, F1 score, inference time, and CPU and RAM usage were evaluated. In the first tests, the results showed a balance between the algorithms, with very small differences in the metric values, with CNN being slightly superior. With the implementation of the algorithms on the Raspberry Pi and the conversion of the models using OpenVINO, the results showed slight variations, ranging from 93% to 96%. However, considering the impact of implementing this architecture on an SBC device, the evaluation of inference time and resource consumption were the determining factors in the selection of the CNN algorithm for the final version of the architecture.

It is important to emphasize that the developed architecture covers a wide range of attacks, not limiting itself only to denial-of-service attacks, which are mostly the target of research. Moreover, the implementation of deep learning algorithms on SBC devices for the purposes of this research represents a promising result for the area of IoT network security, which faces challenges due to the computational limitations of these devices.

In future work, we intend to introduce the detection and mitigation of adversarial attacks, which are attacks focused on impairing the classification of machine learning and deep learning algorithms. Another important step is to compare the performance of the architecture on another IoT device to assess the viability of the solution. Finally, we plan to implement efficient rule management by deleting the least used rules and dynamically adding new ones.

References

[1] Rasheed Ahmad and Izzat Alsmadi. 2021. Machine learning approaches to IoT security: A systematic literature review. *Internet of Things* 14 (2021), 100365.

[2] Mohammed Ali Al-Garadi, Amr Mohamed, Abdulla Khalid Al-Ali, Xiaojiang Du, Ihsan Ali, and Mohsen Guizani. 2020. A survey of machine and deep learning methods for internet of things (IoT) security. *IEEE Communications Surveys & Tutorials* 22, 3 (2020), 1646–1685.

[3] Faris Alhamed, Davide Scano, Piero Castoldi, Juan Jose Vegas Olmos, Ilya Vershkov, Francesco Paolucci, and Filippo Cugini. 2023. P4 Telemetry collector. *Computer Networks* 227 (2023), 109727.

[4] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. 2014. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review* 44, 3 (2014), 87–95.

[5] Antonia Raiane Santos Araujo Cruz, Rafael Lopes Gomes, and Marcial Porto Fernandez. 2021. DIMI: Detecção Inteligente de Botnets Mirai em Redes IoT. *Anais do Computer on the Beach* 12 (2021), 362–369.

[6] Mahmoud Said Elsayed, Nhien-An Le-Khac, Soumyabrata Dev, and Anca Delia Jurcut. 2020. Ddosnet: A deep-learning model for detecting network attacks. In *2020 IEEE 21st International Symposium on "A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*. IEEE, 391–396.

[7] Mohamed Amine Ferrag, Othmane Friha, Djallel Hamouda, Leandros Maglaras, and Helge Janicke. 2022. Edge-IIoTset: A new comprehensive realistic cyber security dataset of IoT and IIoT applications for centralized and federated learning. *IEEE Access* 10 (2022), 40281–40306.

[8] Pradyumna Gokhale, Omkar Bhat, and Sagar Bhat. 2018. Introduction to IOT. *International Advanced Research Journal in Science, Engineering and Technology* 5, 1 (2018), 41–44.

[9] P4 Education Working Group. 2021. P4Pi: P4 on Raspberry Pi. https://github.com/p4lang/p4pi.

[10] Mahmudul Hasan, Md. Milon Islam, Md Ishrak Islam Zarif, and M.M.A. Hashem. 2019. Attack and anomaly detection in IoT sensors in IoT sites using machine learning approaches. *Internet of Things* 7 (2019), 100059. https://doi.org/10.1016/j.iot.2019.100059

[11] Frederik Hauser, Marco Häberle, Daniel Merling, Steffen Lindner, Vladimir Gurevich, Florian Zeiger, Reinhard Frank, and Michael Menth. 2023. A survey on data plane programming with p4: Fundamentals, advances, and applied research. *Journal of Network and Computer Applications* 212 (2023), 103561.

[12] Sya Raihan Heggi, Parman Sukarno, and Satria Akbar Mugitama. 2022. LSTM-NB: DoS Attack Detection On SDN With P4 Programmable Dataplane. In *2022 International Conference on Advanced Creative Networks and Intelligent Systems (ICACNIS)*. IEEE, 1–6.

[13] Ammar D Jasim et al. 2022. A survey of intrusion detection using deep learning in internet of things. *Iraqi Journal For Computer Science and Mathematics* 3, 1 (2022), 83–93.

[14] Ansam Khraisat and Ammar Alazab. 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 (2021), 1–27.

[15] Igor Kotenko, Konstantin Izrailov, and Mikhail Buinevich. 2022. Static analysis of information systems for IoT cyber security: a survey of machine learning approaches. *Sensors* 22, 4 (2022), 1335.

[16] Sándor Laki, Radostin Stoyanov, Dávid Kis, Robert Soulé, Péter Vörös, and Noa Zilberman. 2021. P4Pi: P4 on Raspberry Pi for networking education. *ACM SIGCOMM Computer Communication Review* 51, 3 (2021), 17–21.

[17] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436–444.

[18] Milton Lima, Ricardo Lima, Fernando Lins, and Michel Bonfim. 2022. Beholder–A CEP-based intrusion detection and prevention systems for IoT environments. *Computers & Security* 120 (2022), 102824.

[19] Francisco Sales de Lima Filho, Frederico AF Silveira, Agostinho de Medeiros Brito Junior, Genoveva Vargas-Solar, and Luiz F Silveira. 2019. Smart detection: an online approach for DoS/DDoS attack detection using machine learning. *Security and Communication Networks* 2019 (2019), 1–15.

[20] Francesco Musumeci, Ali Can Fidanci, Francesco Paolucci, Filippo Cugini, and Massimo Tornatore. 2022. Machine-learning-enabled DDoS attacks detection in P4 programmable networks. *Journal of Network and Systems Management* 30 (2022), 1–27.

[21] Dhiego Ramos Pinto, Julio Cesar Duarte, and Ricardo Sant'Ana. 2019. A deep learning approach to the Malware classification problem using autoencoders. In *Proceedings of the XV Brazilian Symposium on Information Systems*. 1–8.

[22] Qiaofeng Qin, Konstantinos Poularakis, and Leandros Tassiulas. 2020. A learning approach with programmable data plane towards IoT security. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 410–420.

[23] Amiya Kumar Sahu, Suraj Sharma, Mohammad Tanveer, and Rohit Raja. 2021. Internet of Things attack detection using hybrid Deep Learning Model. *Computer Communications* 176 (2021), 146–154.

[24] Eryk Schiller, Andy Aidoo, Jara Fuhrer, Jonathan Stahl, Michael Ziörjen, and Burkhard Stiller. 2022. Landscape of IoT security. *Computer Science Review* 44 (2022), 100467.

[25] Sabrina Rocha Souza, Bruno Pedraça Souza, Anderson Gonçalves Uchôa, and Daniella Oliveira Costa. 2022. TEl-IoT: A Template for Eliciting IoT Software System Requirements. In *Proceedings of the XVIII Brazilian Symposium on Information Systems*. 1–8.

[26] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in software engineering*. Springer Science & Business Media.