

Physical Layer Anomaly Detection Mechanisms in IoT Networks

Pedro Martins^{*†}, André B. Reis^{*}, Paulo Salvador^{*†}, Susana Sargento^{*†}

^{*}Instituto de Telecomunicações - Aveiro, Portugal

[†]DETI / University of Aveiro, Portugal

Abstract—With the advent of IoT and wireless mesh networks, security risks inherent to these types of networks have grown in number, severity, and potential for harm. Most of the approaches currently available for anomaly detection in IoT networks perform frame and packet inspection, which may inadvertently reveal private behavioural patterns of its users.

This paper proposes a privacy-focused framework for anomaly detection which analyses radio activity at the physical layer, measuring silence and activity periods, and extracts relevant features for training One-Class Classification (OCC) models.

We train our models with data captured from interactions with an Amazon Echo with multiple devices generating background noise, thus simulating a business-like environment, and test them against a similar scenario with a tampered network node periodically uploading data to a local machine. Our data show that the best performing model is able to detect anomalies with a 99% precision rate.

Index Terms—Anomaly Detection, IoT Networks, Network Monitoring, Machine Learning.

I. INTRODUCTION

Currently embedded sensors have become the trend, being used in both home automation and in critical and sensible infrastructures. Internet of Things (IoT) devices are always a security liability due to their heterogeneity, lack of constant monitoring, their placement in open wireless mediums, their limited resources and, more recently, the access to locally-stored sensitive data. Manufacturers do not implement strong authentication and encryption algorithms to be used in the communications between devices, forcing administrators to use monitoring strategies to keep the network secure.

The goal of most monitoring systems is to detect anomalies, i.e., sudden and often short-term deviations of the normal behavior of a given network, which may be originated by intruders with malicious intent, trying to steal information or hardware resources, or just performing Denial of Service (DoS) attacks to disrupt the entire network. Additionally, many accidents are also considered outliers, such as a sudden router overloading due to another router malfunctioning. The spike in total network attacks, their severity and complexity have forced administrators to rely on anomaly analysis tools to detect new and unforeseen phenomena, rather than solutions that look for traditional and well-known attacks. However, those are usually solely based on frame or packet inspection, allowing easy discrimination of users. This means that the communication behaviors of each device could be inferred, ultimately raising privacy concerns.

This paper proposes a privacy-focused framework for anomaly detection in IoT networks, which works by identifying and modeling subtle patterns on activity and silence periods in the OSI layer 1 power signal using One-Class Classification (OCC) models. The main goal is to identify novel or unseen phenomena, and therefore only anomaly-free data is used to train the classifiers. The proposed framework contains a full pipeline, demonstrating how raw data is captured and processed to obtain Received Signal Strength Indicator (RSSI) data throughout the time, how we transformed it into relevant features that are used by the machine learning algorithms.

Real tests consisted of monitoring the behavior of a Wi-Fi channel where multiple devices are simultaneously connected to the same Access Point (AP) to generate background activities, with particular focus on the interaction with personal assistant devices, namely the Amazon Echo. The ultimate goal is to identify a tampered node which is periodically uploading data to another node on the network. In sum, density-based models, such as Kernel Density Estimation (KDE) and Gaussian Mixture Model (GMM), proved to be the best all-around performers, mainly due to their lower false positive rates, while boundary and distance-based methods such as the One-Class Support Vector Machine (OC-SVM), Isolation Forest (IF) and Local Outlier Factor (LOF) fell short in comparison. The main drawback of solely using layer 1 data is its susceptibility to background noise, which can pose a real challenge by modifying usual activity patterns. However, the results have shown that we could accurately detect anomalies.

The remainder of the paper is organized as follows. Section II addresses related work and section III the machine learning background relevant to the present context. Section IV showcases the proposed framework, followed by the testing scenarios in section V, and results in section VI. Finally, we address our conclusions and future work in section VII.

II. RELATED WORK

Nowadays, it remains a challenge to develop a proper Intrusion Detection System (IDS) for the IoT, since embedded systems often rely on low-power CPUs. This has forced developers to ditch advanced encryption protocols, trusting the surveillance of the environment to the IDS.

Recent studies show that there has been an effort in developing custom IDS for IoT networks. Stephen and Arockiam [1] presented an IDS to detect Sinkhole attacks in Routing over Low Power and Lossy Networks (RPL). Another IDS,

Pulse [2], uses supervised learning techniques to detect DoS attacks. Both propose signature-based solutions [3], which have limited detection range, addressing very specific attacks, and also allow for easy discrimination of users' behaviors.

Regarding electromagnetic spectrum monitoring, Moss *et al.* [4] proposed a Spectral Anomaly Detection (SAD) system optimized for FPGAs by combining a low-latency Discrete Fourier Transform (DFT) to obtain the power spectrum, and an efficient anomaly detection algorithm based on the construction of bitmaps derived from the time-series signal. However, their algorithm is limited, as it is only able to detect anomalies in regular time-series. Another SAD system, SAIFE [5], resorts to an Adversarial Autoencoder (AAE) for semi-supervised anomaly detection on power spectrum density (PSD) vectors. Besides the authors not extracting features from the PSD vectors - using them as inputs of the AAE -, they only tested fully synthetic anomalies, and their classification rates were highly dependent on the signal-to-noise ratio (SNR) between anomalous and "clean" datasets.

The Received Signal Strength Indicator (RSSI), i.e., the power perceived by a given node on the network, has been used in a variety of solutions that try to mitigate physical layer attacks in wireless IoT networks. It can reveal a correlation between the node distance to the AP, or whether there is activity on the channel. The majority of current studies use the RSSI as a distance-based indicator, either finding anomalies by looking into changes in the distance between network nodes [6]–[8], or by only inspecting signal strength variations [9]. None of these proposals look into the channel activity, and a tampered node's behavior can change while staying in the same physical position.

Unlike the mentioned studies, this paper proposes a framework that uses the RSSI as an activity indicator, identifying the presence of outliers based on a statistical analysis of activity and silence periods.

III. BACKGROUND

The mechanisms for anomaly detection proposed in section IV resort to artificial intelligence methods to achieve their goal. This section focuses on showcasing the techniques used for anomaly detection, namely tools to reduce the dimensionality of the input feature-space, and one-class classifiers.

A. Feature Dimensionality

In many situations, improving the classification or prediction performance is not as simple as tweaking the parameters of the machine learning model or even changing it at all. In reality, the correlations that a given model is trying to decode are hidden in the features of the input data, i.e., the input vectors of the models.

Selecting the features that will feed an algorithm hugely impacts the results, as using irrelevant or partially-only relevant features may deteriorate the overall performance. Feature selection methods are a great tool to filter the relevant features, and they are often split into filter, wrapper, and embedded categories [10]. Filter methods compute features importance

based on criteria independent from the classification method, and they are usually based on correlation coefficients and statistical tests, such as information gain and Euclidean distances. Embedded models, namely tree-based classifiers such as Decision Trees, Extra-Trees, and Random Forests, often provide a feature importance property: the higher it is, the more relevant it is towards the output value. Wrapper methods use models as a black box to compute a score on a subset of features based on the classifier prediction. Unlike embedded methods, wrapper methods often use backward feature elimination strategies to infer feature relevance.

B. One-Class Classification

One-Class Classification (OCC) models check for novel or outlier samples in the test set concerning the distribution of the training data. One usually classifies them as density, distance, reconstruction, and domain-based methods [11].

1) *Density-based methods*: When a large amount of data is available, the most simple approach is often to estimate the density of the training data and set a threshold regarding that density to detect outliers. Several probabilistic distribution models are applicable, such as Gaussian distributions, or even a mixture of them to create a more flexible model, such as a Gaussian Mixture Model (GMM) [11]. GMM is given by the sum of M probability density functions p_N :

$$p_{\text{GMM}}(x) = \sum_{i=1}^M \alpha_i p_N(x, \mu_i, \sigma_i^2) \quad (1)$$

where α_i denotes the mixing weights, and μ_i, σ_i^2 the mean and standard deviation of the i -th normal distribution.

A non-parametric approach, such as the Kernel Density Estimation (KDE) [12] (or Parzen windows), is also popular. Considering $D = x_1, \dots, x_N$, a d dimensional set of N examples, and the kernel function ϕ , whose window size is given as h , the Parzen window density estimation is given by:

$$p_{\text{Parzen}}(x) = \frac{1}{N} \sum_{i=1}^N \phi(x - x_i, h) \quad (2)$$

2) *Distance-based methods*: Distance-based methods usually include clustering or nearest-neighbor approaches, which essentially compute distances between data points, i.e., their similarity. Local Outlier Factor (LOF) [13] computes the local density deviation of a given sample with respect to its neighbors, and it assumes that an outlier has a substantially lower density than clean data. It is classified as a distance-based approach, since density values are computed using distances between neighbor samples. Considering the k -distance of an object o , denoted as $\text{dist}_k(o)$, as the distance to its k -th nearest neighbor, and its k -distance neighborhood, denoted as $N_k(o) = \{o' | o' \text{ in } D, \text{dist}(o, o') \leq \text{dist}_k(o)\}$, the local reachability density of o is given by:

$$\text{lrd}_k(o) = \frac{||N_k(o)||}{\sum_{o' \in N_k(o)} \text{reachdist}_k(o' \leftarrow o)} \quad (3)$$

$$\text{reachdist}_k(o' \leftarrow o) = \max\{\text{dist}_k(o), \text{dist}(o, o')\}$$

The LOF can then be computed as the average ratio of local reachability of o and its k -nearest neighbors:

$$\text{LOF}_k(o) = \frac{\sum_{o' \in N_k(o)} \frac{\text{lrd}_k(o')}{\text{lrd}_k(o)}}{||N_k(o)||} \quad (4)$$

In sum, the lower the local reachability density of o and the higher of its k -nearest neighbors, the higher the LOF; therefore, a higher probability of o being an outlier. Since this algorithm ranks points by its neighbors, it is possible to miss outliers whose local reachability densities are similar to the ones of other neighbor outliers.

3) *Reconstruction/model-based methods*: Reconstruction-based models use prior knowledge regarding the data to build a model to profile non-outlier instances of input data, which can compute a novelty score of a test sample. Although fitting in this category, an Isolation Forest (IF) isolates anomalous instances rather than building a model solely based on normal data [14]. In order to detect outliers, the IF assumes that a tree structure can isolate every instance. Since anomalies are susceptible to isolation, they are usually in the root of the tree. If fewer steps are needed to isolate a sample, the chances of it being an anomaly increase. The anomaly score of an IF is:

$$s(x, n) = 2^{-\frac{E(h(x))}{c(n)}} \quad (5)$$

where $h(x)$ is the path length of observation x , $c(n)$ is the average path length of unsuccessful search in a Binary Search Tree (BST), and n is the number of external nodes. Scores s closer to 1 indicate possible anomalies, but one must define a threshold to separate them from clean data.

4) *Domain-based methods*: These methods try to create a boundary surrounding the training data, and they are usually agnostic to its density. The most popular approach is the One-Class Support Vector Machine (OC-SVM) [15], which uses a feature map $\phi \Rightarrow X \rightarrow F$ on a N -sample set X to map it into a dot space F , by evaluating a given kernel function $K(x, y) = \phi(x) \cdot \phi(y)$, and maximizes the distance of the obtained hyperplane characterized by w and ρ to the origin. The minimization function is then defined as:

$$\min_{w, \xi_i, \rho} \frac{||w||^2}{2} + \frac{1}{vN} \sum_{i=1}^N \xi_i - \rho \quad (6)$$

subject to:

$$(w \cdot \phi(x_i)) \geq \rho - \xi_i \quad i = 1, \dots, N$$

$$\xi_i \geq 0 \quad i = 1, \dots, N$$

where ξ_i are slack variables and the coefficients α_i are found as the solution of the equation 7.

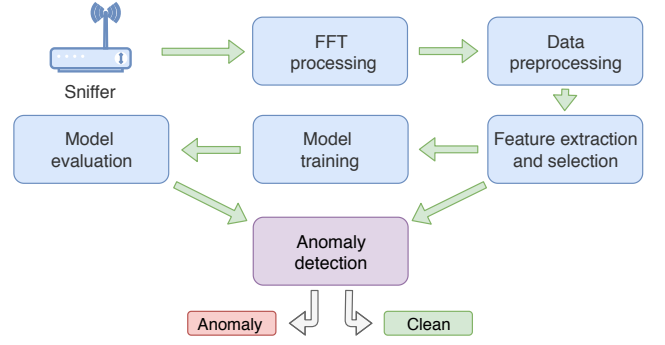


Fig. 1: High-level view of the anomaly detection architecture. The arrows indicate the data flow.

$$\min_{\alpha} \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j K(x_i, x_j) \quad (7)$$

subject to:

$$0 \leq \alpha_i \leq \frac{1}{vN} \quad \sum_i \alpha_i = 1$$

The parameter $v \in [0, 1]$ sets an upper bound on the fraction of outliers and it is used as a regularization parameter. The decision function is then defined as:

$$f(x) = \text{sgn}\left(\sum_{i=1}^N \alpha_i K(x, x_i) - \rho\right) \quad (8)$$

IV. FRAMEWORK ARCHITECTURE OVERVIEW

This section presents the proposed framework architecture. Opposite to other anomaly detection tools, which resort to deep packet inspection, or frame analysis, we are proposing privacy-focused mechanisms that only use physical layer data to model the behavior of the network. Therefore, it is harder to differentiate traffic patterns of individual users/nodes, as there is no way to identify them as one would do at layers 2 and 3, by their Media Access Control (MAC) or Internet Protocol (IP) addresses.

In order to properly present it, we split the overall system into a series of steps. All the showcased pipeline in figure 1 is further detailed in the following sub-sections.

A. RSSI data collection

The essence of this work is to identify anomalous behaviors in the OSI layer 1, thus only resorting to the RSSI. Dealing exclusively with layer 1 network data is essential, owing to the ability to monitor an environment without the need to authenticate a sniffer client in the network, assuring at the same time that communications between clients stay private as one cannot individually identify them. The RSSI is the amount of power in a received radio signal; therefore, it is not bound to any specific Radio Frequency (RF) technology, and one can use it with others such as Bluetooth or Zigbee.

To capture RF data, one needs a probe with a radio capable of recording its environment, i.e., a sniffer. That radio must

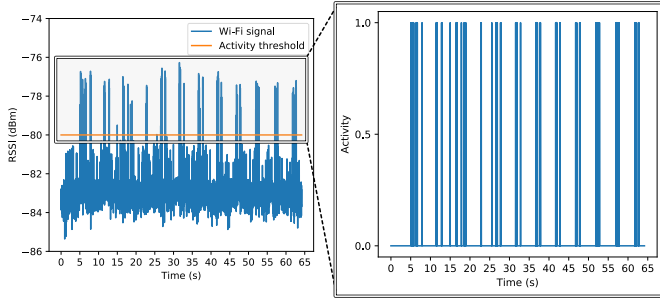


Fig. 2: Extraction of activity periods (on the right) from the original RSSI values (on the left) with a threshold of -80.0dBm .

be able to sweep an array of frequencies $F = f_1, \dots, f_n$ in the same instant t , whose components may correspond to a Wi-Fi channel as example, and return the corresponding amplitude values $A(F, t)$ for those very same frequencies. Traditional transceivers are not capable of such a task. Therefore, Software Defined Radios (SDRs) are ideal for these situations.

One must configure the SDR with an adequate sample rate f_s that covers the entire spectrum of frequencies of interest F , thus outputting samples every $t_s = 1/f_s$. To accurately check the RSSI on a specific frequency, the time domain values must be converted to the frequency domain using the Discrete Fourier Transform (DFT). It would be excessively taxing on the CPU to perform Fast Fourier Transform (FFT) computations over the full signal, so one has to define an FFT window size (size_{FFT}), indicating how many consecutive time samples the DFT will use.

Moreover, the majority of SDRs are powered by an amplifier, whose configuration must be carefully done. It is essential to assure that it is easy to distinguish ground noise from network activity. Determining the optimal parameters is usually done by inspecting the outputted signal, with multiple combinations of configuration parameters. It is nearly impossible to define a set that works across a variety of different radios, respective amplifiers, and that perfectly captures signals in both noisy and clean environments. Nonetheless, an overall well-configured amplifier facilitates the choice of the silence/activity threshold, hence overall better performance.

B. Data preprocessing

In the machine learning algorithm standpoint, the data outputted by the sniffer or after the FFT processing does not have any symbolic meaning. Therefore, one needs to transform it into relevant features. Given that layer 1 data suffer from variations owing to real-world distance changes, the addition of physical obstacles between nodes, and overall background noise, performing statistical analysis over absolute power values is not a plausible strategy.

Most anomalies have an underlying periodicity; hence, using statistics of activity and silence momentums is preferable, as they reveal the behavioral patterns of normal and anomalous traffic. In order to take those periods apart, one must define an RSSI threshold, as it will separate the ground noise from the network activity (figure 2). For instance, the data

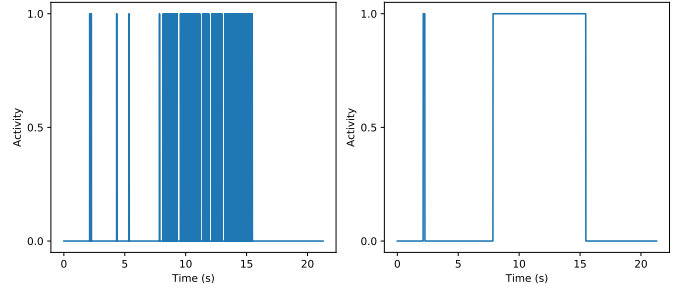


Fig. 3: Post-processing of activity periods, joining activity peaks and removing short non-relevant activities (on the right). The original activity signal is represented on the left.

extracted over an IoT network with sensors synchronously uploading data every minute may reveal that 90% of silent periods last just below 60 seconds. If a node that got tampered transmitted data every 30 seconds, this metric would be nearly 30 seconds, which one can see as an anomaly. Given a value $0.0 < \alpha < 1.0$, denoting how much smaller the threshold will be in comparison to the mean RSSI timeline value (recall that RSSI values are negative), the threshold that separates activity data from silence on a timeline with m samples is given by:

$$\text{thr} = \left\lfloor \frac{\alpha}{m} \sum_{i=0}^{m-1} \text{RSSI}_i \right\rfloor \quad (9)$$

For reference, a silent period is a block of consecutive samples whose RSSI values are below the defined threshold. On the other hand, an activity period is a block of consecutive samples whose RSSI values are over it. During a transmission, small silence gaps between activity periods can show up, due to latency from drivers and applications, waiting times while accessing the network, or the proper nature of wireless communications. Thus, we can group them into one longer activity period. Likewise, there is activity in the network that one can consider unaccounted, such as non-requested Address Resolution Protocol (ARP) and Multicast DNS (MDNS) packets, or discovery protocols such as Simple Service Discovery Protocol (SSDP). Hence, one can discard activity periods whose duration fits under a given threshold. This *non-relevant activity filter* may also delete Transmission Control Protocol (TCP) Keep-Alive and other session packets, which essentially describe the flow of data in the network. Considering this, we propose that, before settling on any filtering strategy, one tests both methods - with or without the removal of small activity periods (figure 3).

C. Feature extraction

Although the activity timeline shows the network behavior, it does not highlight changes and variations in such a way that machine learning algorithms can model those patterns.

With this in mind, the first step is to split the timeline samples into blocks, or windows, and compute relevant features over them. The most straightforward strategy would be to split the timeline into consecutive windows of the same size. However, this “hard” separation has some setbacks, as

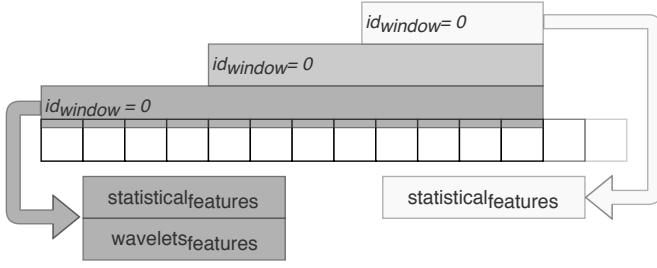


Fig. 4: Example of the features extracted for every window size. Even though statistical features are extracted in all of them, the local maxima energies and scales of the CWT are only computed for the largest defined window size.

different samples may sometimes show very different patterns, making it harder for the algorithm to create an overall model. Resorting to a “sliding” strategy can be beneficial to increase the throughput and the similarity between consecutive windows. In this case, consecutive windows will slightly overlap, and a significant advantage of this strategy over a “hard” split is that it usually ends up generating more data windows, which may increase the classification performance.

The proposed method for feature extraction is based on performing a statistical analysis over periods of silence and activity, as they will allow differentiating communications between entities with distinct periodicities.

To identify a variety of periodic events, extracting features from multiple window sizes can be a simple approach, as they provide different insights in relatively different periods. Nonetheless, the dataset must be split according to the largest window size and the defined window slide. Then, for every sub-window, which end on the same data sample, the statistical measurements are computed. For training purposes, we group all these features into a single window, even though every sub-window provides insights over different periods.

As showcased in figure 4, mean, median, standard deviation, 75th, 90th, 95th, and 99th percentiles, minimum and maximum values regarding activity and silence periods are computed for every sub-window. Additionally, we obtain a scalogram over the timeline on the longest sub-window; thus, pseudo-periodic components are also used as features. The scalogram is computed as the normalized squared modulus of the discretized Continuous Wavelet Transform (CWT), using a Morlet mother wavelet [16], on a set of scales relative to the sub-window size. It then outputs energy values for each of the scales, which reflect the periodicity of events on a given scale. The number of extracted scales is a compromise between feature extraction latency and accuracy of the representation.

The most relevant information from the scalogram concerns the local maxima scales and their energy values. The optimal number of extracted peaks is another compromise, now between the amount of information and the performance of the learning algorithm. Furthermore, using a large number of peaks may result in unwanted zeros, since the scalogram may not have many local maxima, which ultimately can make it harder for the model to differentiate patterns.

D. Feature dimensionality

It is often a good practice to perform dimensionality reduction in the feature set used to train the classifiers, especially if the number of parameters is too high when compared to the number of samples, or when using classifiers that determine non-linear boundaries, resulting in overfitted models. Additionally, the learning process will be significantly faster with a smaller number of features.

Taking it into account, one uses feature selection embedded models [10], [17] to extract feature relevance regarding each group of features (statistical and pseudo-periodic) and a ranking system to sort them. The goal is to minimize a $k < num_{features}$ ($num_{features}$ denotes the total number of features of a given group of features), by training different models with $j \in [1, num_{features}]$ features, and verify which j maximizes the classifier *F1-Score* (equation 12).

E. Model training and evaluation

Once the data is processed, one may use it to train machine learning algorithms that fit the problem. The first step is to split the data into training and testing sets. Even though we only consider OCC models, anomaly-labeled data is also split into training and testing sets as we will use it for Cross-Validation (CV) purposes. Each model is then trained num_{trains} times, so that using different shuffled training and testing sets may reduce the variance of the performance metrics and return a confidence interval over the classification results.

Choosing an optimized model is not a straightforward procedure. In each training iteration, after splitting the datasets (one only containing “clean” data and the other having an anomaly), classifiers are trained with multiple combinations of hyperparameters. The objective is to find the set $\lambda_{opt} \in \Lambda$, where Λ corresponds to a group of sets of hyperparameters, defining a model ψ that maximizes the average *F1-Score* (F_1) on multiple sets of cross-validation (CV):

$$\lambda_{opt} = \arg \max_{\lambda \in \Lambda} \frac{1}{num_{splits}} \sum_{X, y \in CV} F_1(\psi, \lambda, X, y) \quad (10)$$

For each combination, the overall “clean” training set is divided into training and CV sets using the Monte Carlo cross-validation splitting [18], which randomly selects a given percentage ($batch_{split}$) of the overall set to serve as CV data, while the remaining is used for training the classifier. For every combination of hyperparameters, the splitting exemplified in equation 11 is repeated num_{splits} times.

$$\begin{aligned} train_{size} &= (1 - batch_{split}) \times trainSet_{size} \\ outlier_{size} &= \frac{trainSet_{size} \times batch_{split} \times outliers_{fraction}}{1 - outliers_{fraction}} \\ cv_{size} &= trainSet_{size} - train_{size} + outlier_{size} \end{aligned} \quad (11)$$

Besides feature selection, an important technique to improve the performance of some models is feature scaling. This

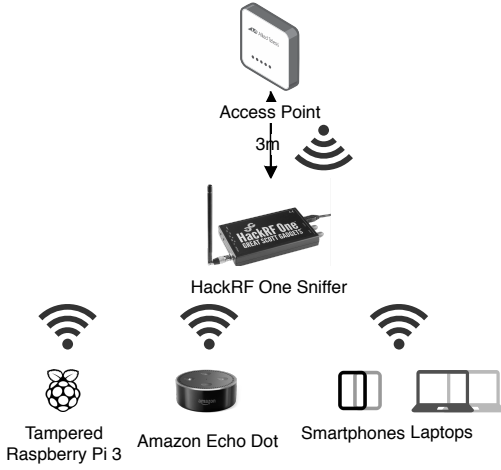


Fig. 5: Testing scenario network diagram.

method essentially rescales the features such that they behave like a normal distribution with zero mean and a standard deviation of one. This is a critical step since some models, such as SVMs and k -Nearest Neighbours (k -NN), do not deal well with features whose values are on very distinct scales.

The goal is to correctly model the “normal” behavior of the network; therefore, the models should ideally not be learning with any specific anomaly, making one-class classifiers ideal for this use case [19].

The built models should be as broad as possible; in an optimal scenario, they would be able to detect any presented anomalies and avoid sending alarms when real “normal” traffic is flowing through the network, i.e., they would ideally have few false positives. Each model will be evaluated and compared according to its *F1-Score*. Consequently, both precision and recall must be computed according to their number of outliers true positives (TP), false positives (FP), and false negatives (FN), as showcased in equation 12.

$$\text{precision} = \frac{TP}{TP + FP} \quad \text{recall} = \frac{TP}{TP + FN} \quad (12)$$

$$F_1 = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

V. EXPERIMENTAL SETUP

This section presents the experimental scenario and setup, whose goal is to detect the presence of periodic outlier behavior on a Wi-Fi channel.

Initially, we designed a scenario aimed to work as a proof-of-concept. It mainly reflected a home automation scenario, containing a tampered Raspberry Pi 3, which is periodically uploading data to a local server, and an Amazon Echo. The “normal” behavior of this IoT network consisted of the user’s usual interaction with the Amazon Alexa, consisting of casual requests, such as weather and sports updates, and Spotify streaming. Then, we added a second node to generate more traffic and make this scenario relatively more realistic.

Nonetheless, these were basic examples, and we just used them as trial runs. A more real-life scenario, whose results we showcase in this paper, was used to test this framework, which corresponds to a work environment with the same IoT devices previously mentioned. As the amount of non-tampered connected devices is superior - always more than 5, in comparison to the maximum 2 in the previous case -, the Wi-Fi channel will have far more usage, making it harder to detect the presence of periodic outlier behavior (figure 5).

The outlier consisted of a “tampered” Raspberry Pi 3, which is running a script that uploads to a local machine a 1MB file with the following statistical distributions:

- A file transfer which occurred every 10 or 60 seconds;
- A file transfer which occurred with Gaussian distribution with 10 mean, and 5 seconds standard deviation;
- A file transfer which occurred with exponential distribution with 10 or 60 seconds mean.

As mentioned in the previous section, using an SDR as a sniffer is ideal for these scenarios. We conducted all tests using a HackRF One [20], an open-source hardware SDR, adequately configured according to the environment, and connected to a laptop performing live FFT computations over the outputted data. To provide compatibility with older devices, the tests were performed over the 2.4GHz band and on a “free” channel, i.e., where, before these experiments, no Access Points or devices were working on that specific frequency. Another aspect to consider is the data bandwidth: we set it to 20MHz, the approximate size of a Wi-Fi channel, and the maximum allowed by the HackRF. Then, every $1/20e6 \approx 50ns$, it would generate a new sample, which is an 8-bit quadrature (8-bit Q and 8-bit I). The HackRF fills a buffer of 262,144 bytes about 152 times every second, i.e., it outputs 131,072 samples per buffer. Storing all these data is not feasible as an hour-long dataset could potentially reach hundreds of gigabytes. In order to reduce the CPU load over the intensive FFT computations and disk usage, only 1024 samples of each buffer are converted to the frequency domain and stored, which means that 1024 power samples are going to be stored approximately every 6.55ms. Each FFT bin will correspond to about 19.5kHz, a reasonable precision given that we are working with 2.4GHz Wi-Fi and only interested in whether there is activity on the channel by averaging those 1024 power values and obtain the mean RSSI.

When calibrating the HackRF, the goal is to separate the ground noise from activity in the spectrum. In the conducted tests, we set its amplifier LNA gain to 8dB, and the VGA gain to 0dB. The first corresponds to the main gain, while the latter amplifies ground noise, thus setting it up to 0dB. For proper configuration, it is recommended to test multiple combinations of these parameters - we tested all the available combinations from the HackRF One - and analyze graphical representations of the outputted signal. Nonetheless, these proved to correctly operate on both the proof-of-concept scenarios, the one with a much less noisy environment, and the scenario here presented. After testing several values in the range [0.90; 1.0], the activity

TABLE I: Overview of the used parameters.

Parameter	Values
SDR center frequency	2.4GHz
SDR sample rate	20MHz
SDR LNA gain	8dB
SDR VGA gain	0dB
FFT size	1024
Activity threshold α	0.98
Activity period threshold	75 samples
Silence period threshold	0 or 5 samples
Feature window size	30, 60, 300, 900 seconds
Sliding window size	2 seconds
num _{trains}	5
num _{splits}	5
Training split	0.8 training / 0.2 testing
batch _{split}	0.7 training / 0.3 CV
outliers _{fraction}	0.5

threshold α was set to 0.98, setting the threshold to -81dBm .

The placement of the sniffer is an essential aspect, as it should not be very far from the AP; otherwise, activity and ground noise will be almost indistinguishable. We placed them about 3 meters from each other. The majority of the tests were performed on the same periods of weekdays, early afternoons, where the number of connected devices is often higher.

The activity and silence thresholds were yet other parameters that one had to set. We set the first to an interval of approximately $6.55\text{ms} \times 75 \approx 500\text{ms}$. This value is relatively high to assure that it accounts for latency from driver and applications or waiting times while accessing the network. The second was set to a relatively smaller time interval of $6.55\text{ms} \times 5 \approx 33\text{ms}$; therefore, capturing non-requested ARP exchanges and other unaccounted activity. However, we also extracted features using a zero-sample silence threshold, i.e., not filtering out non-relevant network activity.

The data are split into sliding windows of 15 minutes with 2 seconds sliding distance, with a set of sub-windows of 30, 60, and 300 seconds. Multiple sizes allow having different perspectives over distinct periodicities, and the small sliding distance captures even small intervals of activities, besides generating more training data.

The majority of datasets had around 2 to 3 hours worth of data. The dataset with a “clean” network was a mixture of multiple datasets to capture multiple non-anomalous scenarios, resulting in a much bigger set when compared to datasets with contamination. We duplicated the samples of the smallest sets until their size did not exceed the size of the largest set to balance their sizes. For instance, considering that the “clean” set has 5000 samples and an anomaly set has 2000, the latter will be doubled to 4000 samples.

Next, one has to perform a study about the relevance of the features and select those that are considered most important and most likely to help the OCC models to differentiate outliers from “clean” behavior. The final step is to train each model with the selected features and to extract the results. The tested models were OC-SVM [21], IF [14], LOF [13], KDE [12], GMM [11], using Scikit-learn [22]

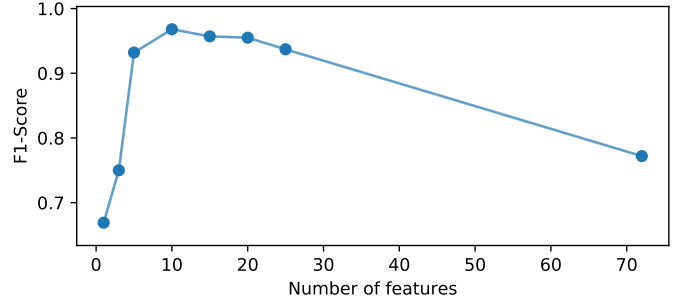


Fig. 6: *F1-Score* for multiple statistical feature subsets.

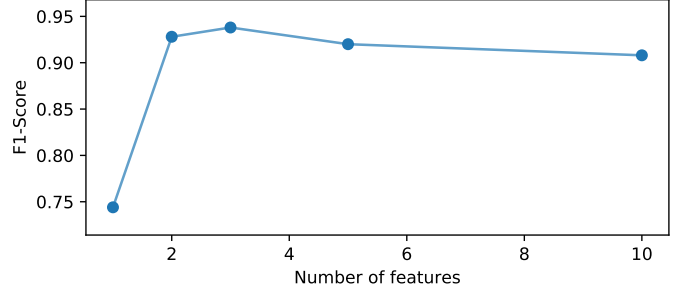


Fig. 7: *F1-Score* for multiple wavelet feature subsets.

implementations. For each dataset and each model, the data were randomly split into 80% training and 20% testing 5 times and split according to the Monte Carlo cross-validation method 5 times for every tested set of hyperparameters. We used a batch size for the CV sets equal to 30% of the training set, and so, the remaining 70% of “clean” data were used for training, while the first 30%, plus an equal amount of outlier samples (outliers_{fraction} = 0.5), were used as CV data.

Table I provides an overview of the overall parameters used in the testing scenario.

VI. CLASSIFICATION RESULTS

Given the number of features, we analyzed the importance of both statistical and wavelets local maxima data, using Extra Trees, Decision Tree, and Random Forest classifiers to extract feature relevance. The k most important were then used to train an OCC model, in this particular case, an OC-SVM. As figure 6 shows, the optimal number of features ranges between 10 and 20 - there is a minimal difference between its *F1-Scores*. Using a higher number of features often increases training times; hence, we used 10 of the 72 original features.

We applied the same methodology to the local maxima scales and energies of the CWT. As observed in figure 7, the optimal number of scales sits between 2 and 3, which gives a total of 4 and 6 features, respectively. The difference is minimal, but we chose 3 scales to train the models to provide some more intuition regarding the traffic periodicity.

The left side of table II, which corresponds to the final classification results over datasets where *non-relevant activity filtering* was applied, shows that KDE outperforms every other tested model, scoring above an admirable 99.4% *F1-Score* on all tests, with the GMM model closely behind. The OC-SVM

TABLE II: Classification Results (in percentage)

Type of anomaly		WITH non-relevant activity filter					WITHOUT non-relevant activity filter				
		OC-SVM	IF	LOF	KDE	GMM	OC-SVM	IF	LOF	KDE	GMM
Every 10s	<i>Precision</i>	90.2±0.6	79.3±1.1	83.4±0.7	99.4±0.2	92.1±1.5	94.3±1.2	77.1±1.9	83.8±0.8	99.1±0.3	92.3±1.1
	<i>F1-Score</i>	94.9±0.4	86.6±1.3	90.7±0.4	99.7±0.1	95.9±0.8	96.4±0.6	85.3±1.8	91.1±0.4	99.5±0.1	96.0±0.6
10s mean, 5s std Gaussian	<i>Precision</i>	87.8±0.7	87.6±1.1	85.1±1.2	99.0±0.1	93.6±0.9	87.4±0.6	87.4±0.6	85.7±0.8	99.6±0.2	93.8±1.1
	<i>F1-Score</i>	93.5±0.4	93.4±0.6	91.9±0.7	99.5±0.0	96.7±0.5	93.3±0.3	93.3±0.3	92.1±0.5	99.8±0.0	96.8±0.6
10s mean exponential	<i>Precision</i>	82.1±0.4	73.8±2.6	84.7±0.9	99.4±0.4	94.1±0.3	92.2±0.7	77.4±2.6	84.8±1.0	99.6±0.4	94.0±0.6
	<i>F1-Score</i>	90.1±0.3	65.1±3.3	89.6±1.0	99.7±0.2	97.0±0.1	95.7±0.5	79.2±5.1	90.2±1.1	99.8±0.2	96.9±0.3
Every 60s	<i>Precision</i>	88.8±0.2	87.1±1.4	89.0±1.0	99.5±0.2	95.6±0.6	94.3±0.6	83.7±2.1	89.0±1.4	99.6±0.1	96.0±0.5
	<i>F1-Score</i>	94.0±0.1	89.2±1.0	94.2±0.6	99.7±0.1	97.7±0.3	97.1±0.3	84.4±3.2	94.2±0.8	99.8±0.1	98.0±0.3
60s mean exponential	<i>Precision</i>	82.3±0.5	75.8±2.4	85.3±0.7	99.6±0.1	93.7±0.5	92.0±0.6	80.7±2.6	83.9±3.3	99.4±0.5	93.3±1.0
	<i>F1-Score</i>	90.3±0.3	68.1±2.6	90.0±0.9	99.6±0.1	96.7±0.3	95.5±0.3	85.5±2.5	90.4±2.3	99.7±0.2	96.5±0.5
Mixture of samples	<i>Precision</i>	89.3±2.2	87.1±2.0	88.9±2.0	99.8±0.1	96.2±0.6	95.6±0.5	86.3±1.9	89.9±2.6	99.6±0.1	96.0±0.3
	<i>F1-Score</i>	93.6±0.7	80.9±4.0	93.5±1.1	99.9±0.0	98.1±0.3	97.4±0.3	88.1±3.7	94.2±1.6	99.8±0.1	98.0±0.2

generally performed well, although it was not as good as the KDE or GMM models. Its results show that the lower *F1-Score* comes from lower precision rates, i.e., a higher number of FP. The Isolation Forest models show a significantly worse performance when it comes to identifying outlier data with an exponential distribution. Similarly to the OC-SVM and IF, LOF also recorded some sub-optimal precision numbers.

The right portion of table II also shows that removing unaccounted activity periods deteriorates the performance of the classifiers slightly. KDE and GMM did not show any significant improvements, but they still had the best overall performance. This proves that density methods were not affected by removing non-relevant activity, and they were still able to model clean data correctly. The remaining improved their *F1-Scores*, owing to better FP rates. The most notorious improvements occur while detecting anomalies with an exponential distribution, which, theoretically, are harder to detect.

Overall, density methods revealed themselves as the optimal models to be used for these types of anomalies. On the other hand, despite not being represented in table II, the majority of the models had a higher recall than precision. This means that the majority of outliers were being detected, but “clean” samples were being wrongly marked as outliers.

VII. CONCLUSION AND FUTURE WORK

This paper proposed a framework for the detection of anomalous behaviors in wireless networks, which resorted to a set of machine learning algorithms to determine the accuracy of the proposed framework in detecting the outliers resorting to layer 1 power data. The obtained results in a real scenario, assessed through a set of One-Class classification models, have shown that it is possible to detect anomalies at layer 1 with high accuracy, even without using real and diverse anomalous data during training. Related work has shown that it is possible to obtain detection rates grasping 100% accuracy, but the majority added anomalous behavior synthetically to the “clean” datasets. Differently, power indicators are often susceptible to background noise and interferences, which may lead to unexpected classification results.

Through previously trained models, it is possible to adapt this approach to a single-channel live-detection system and to use it with a variety of radio technologies. One could resort to multiple radios and various models to monitor a broader range of frequencies. Additionally, we recommend using a sniffer per AP, as it would be responsible for monitoring any interactions with that wireless node.

As future work, we plan to train neural network approaches such as Autoencoders, and Generative Adversarial Networks (GAN), since their testing times, and, in some occasions, training times, are usually shorter than some classical models here presented, especially density and distance-based methods.

Additionally, we only tested low-volume and considerably high-frequency behavior, as it seemed more difficult to detect smaller data transfers as the remaining traffic masked them. However, we plan on testing different anomalous behaviors, particularly long-duration and low-frequency ones (e.g., uploading a 1GB video every ten minutes), as it helps to model anomaly-free data and to tune the models.

ACKNOWLEDGMENT

This work is supported by the European Regional Development Fund (FEDER), through the Regional Operational Programme of Lisbon (POR LISBOA 2020) and the Competitiveness and Internationalization Operational Programme (COMPETE 2020) of the Portugal 2020 framework [Project 5G with Nr. 024539 (POCI-01-0247-FEDER-024539)], and by FCT/MEC through national funds and when applicable co-funded by FEDER PT2020 partnership agreement under the project UID/EEA/50008/2019.

REFERENCES

- [1] R. Stephen and L. Arockiam, “Intrusion detection system to detect sinkhole attack on rpl protocol in internet of things,” *International Journal of Electrical Electronics and Computer Science*, vol. 4, no. 4, pp. 16–20, 2017.
- [2] E. Anthi, L. Williams, and P. Burnap, “Pulse: An adaptive intrusion detection for the internet of things,” in *Living in the Internet of Things: Cybersecurity of the IoT-2018*, pp. 1–4, IET, 2018.
- [3] T. Sherasiya, H. Upadhyay, and H. B. Patel, “A survey: Intrusion detection system for internet of things,” *International Journal of Computer Science and Engineering (IJCSE)*, vol. 5, no. 2, 2016.

- [4] D. J. Moss, Z. Zhang, N. J. Fräser, and P. H. Leong, "An fpga-based spectral anomaly detection system," in *2014 International Conference on Field-Programmable Technology (FPT)*, pp. 175–182, IEEE, 2014.
- [5] S. Rajendran, W. Meert, V. Lenders, and S. Pollin, "Saife: Unsupervised wireless spectrum anomaly detection with interpretable features," in *2018 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, pp. 1–9, IEEE, 2018.
- [6] J. Wang, G. Yang, Y. Sun, and S. Chen, "Sybil attack detection based on rssi for wireless sensor network," in *2007 International Conference on Wireless Communications, Networking and Mobile Computing*, pp. 2684–2687, IEEE, 2007.
- [7] H. Chen, P. Han, X. Zhou, and C. Gao, "Lightweight anomaly intrusion detection in wireless sensor networks," in *Pacific-Asia Workshop on Intelligence and Security Informatics*, pp. 105–116, 2007.
- [8] J. Tang, P. Fan, and X. Tang, "A rssi-based cooperative anomaly detection scheme for wireless sensor networks," in *2007 International Conference on Wireless Communications, Networking and Mobile Computing*, pp. 2783–2786, IEEE, 2007.
- [9] A. Sheth, C. Doerr, D. Grunwald, R. Han, and D. Sicker, "Mojo: A distributed physical layer anomaly detection system for 802.11 w lans," in *Proceedings of the 4th international conference on Mobile systems, applications and services*, pp. 191–204, ACM, 2006.
- [10] L. Ladha and T. Deepa, "Feature selection methods and algorithms," *International journal on computer science and engineering*, vol. 3, no. 5, pp. 1787–1797, 2011.
- [11] M. Markou and S. Singh, "Novelty detection: a review—part 1: statistical approaches," *Signal processing*, vol. 83, no. 12, pp. 2481–2497, 2003.
- [12] D. Miljković, "Review of novelty detection methods," in *The 33rd International Convention MIPRO*, pp. 593–598, IEEE, 2010.
- [13] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: identifying density-based local outliers," vol. 29, no. 2, pp. 93–104, 2000.
- [14] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 Eighth IEEE International Conference on Data Mining*, pp. 413–422, IEEE, 2008.
- [15] B. Schölkopf, R. C. Williamson, A. J. Smola, J. Shawe-Taylor, and J. C. Platt, "Support vector method for novelty detection," in *Advances in neural information processing systems*, pp. 582–588, 2000.
- [16] N. Delprat, B. Escudié, P. Guillemain, R. Kronland-Martinet, P. Tchamitchian, and B. Torresani, "Asymptotic wavelet and gabor analysis: Extraction of instantaneous frequencies," *IEEE transactions on Information Theory*, vol. 38, no. 2, pp. 644–664, 1992.
- [17] G. Chandrashekar and F. Sahin, "A survey on feature selection methods," *Computers & Electrical Engineering*, vol. 40, no. 1, pp. 16–28, 2014.
- [18] Q.-S. Xu and Y.-Z. Liang, "Monte carlo cross validation," *Chemometrics and Intelligent Laboratory Systems*, vol. 56, no. 1, pp. 1–11, 2001.
- [19] M. A. Pimentel, D. A. Clifton, L. Clifton, and L. Tarassenko, "A review of novelty detection," *Signal Processing*, vol. 99, pp. 215–249, 2014.
- [20] "HackRF - Great Scott Gadgets." Available at <https://greatscottgadgets.com/hackrf/>.
- [21] L. M. Manevitz and M. Yousef, "One-class svms for document classification," *Journal of machine Learning research*, vol. 2, no. Dec, pp. 139–154, 2001.
- [22] "scikit-learn: machine learning in Python." Available at <https://scikit-learn.org>.