**Pedro
de Bastos Martins**

**Mecanismos para Detecção de Anomalias na Camada
Física em Redes IoT**

**Physical Layer Anomaly Detection Mechanisms in
IoT networks**

**Pedro
de Bastos Martins**

**Mecanismos para Detecção de Anomalias na Camada
Física em Redes IoT**

**Physical Layer Anomaly Detection Mechanisms in
IoT networks**

“Don’t worry about it if you don’t understand.”

— Andrew Ng

**Pedro
de Bastos Martins**

**Mecanismos para Detecção de Anomalias na Camada
Física em Redes IoT**

**Physical Layer Anomaly Detection Mechanisms in
IoT networks**

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica da Doutora Susana Isabel Barreto de Miranda Sargento, Professora Catedrática do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor Paulo Jorge Salvador Serra Ferreira, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

Dedico este trabalho ao meu avô Armindo, por todo o apoio que meu deu nestes cinco anos, e que infelizmente não me pôde acompanhar neste último ano.

o júri / the jury

presidente / president

Professor Doutor Arnaldo Silva Rodrigues de Oliveira

professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

vogais / examiners committee

Professor Doutor Rodolfo Alexandre Duarte Oliveira

professor auxiliar c/ agregação do Departamento de Engenharia Eletrotécnica da Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

Professora Doutora Susana Isabel Barreto de Miranda Sargento

professora catedrática do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro (orientadora)

agradecimentos / acknowledgements

Em primeiro lugar, quero agradecer todo o apoio que a minha família me deu durante todo o meu percurso académico, em especial aos meus pais, que fizeram de tudo para que eu pudesse ter aqui chegado, e aos meus avós, que foram sempre a uma fonte de inspiração e suporte incrível.

Gostava também de agradecer aos meus colegas, que prefiro tratar por amigos, por todos os momentos incríveis que vivemos neste nosso percurso, porque tenho a certeza de que, sem ajuda e companhia deles, não chegaria aqui. Em especial ao grupo QL[©], uma família que sem dúvida nunca será esquecida!

Quero também deixar uma palavra de agradecimento a todos os meus colegas do Networks Architectures and Protocols (NAP) por me receberem de braços abertos, por toda a ajuda providenciada e todo o conhecimento passado, pelos momentos descontraídos e pelas brincadeiras. É um grupo de excelentes pessoas e profissionais, e como tal, não poderia deixar de dar um especial agradecimento à Professora Susana Sargento por me ter acolhido tão bem neste grupo, ter tirado o melhor de mim, como pessoa e estudante, e me dar sempre a liberdade para trabalhar em algo em que me sentisse recompensado. Também quero deixar um especial obrigado ao Professor Paulo Salvador por tudo o que me transmitiu, todas as trocas de ideias e conversas extremamente interessantes, e pela constante disponibilidade em ajudar. Por isso, um grande obrigado a todos.

Agradeço à Fundação Portuguesa para a Ciência e Tecnologia pelo suporte financeiro com fundos nacionais e europeus através Fundo Europeu de Desenvolvimento Regional (FEDER), no âmbito do Programa Operacional Regional de Lisboa (POR LISBOA 2020) e do Programa Operacional de Competitividade e Internacionalização (COMPETE 2020) do Portugal 2020 Mobilizador 5G (POCI-01-0247-FEDER-024539).

Palavras Chave

Redes IoT, Monitorização de Rede, Monitorização do Sinal Rádio, Detecção de Anomalias, Classificação Mono-Classe, Aprendizagem Automática.

Resumo

Com o aparecimento das redes em malha sem-fios e da Internet of Things (IoT) aumentam também os riscos associados à segurança das mesmas, seja pelo uso indevido da rede ou exfiltração de informação. A maioria das soluções atuais para deteção de anomalias em redes IoT baseiam-se em analisar tramas ou pacotes, o que, inadvertidamente, pode revelar padrões de comportamento dos utilizadores, que estes considerem privados. Além disso, as soluções que se focam em inspecionar dados da camada física normalmente usam a potência de sinal recebido (RSSI) como uma métrica de distância e detetam anomalias baseadas na posição relativa dos nós da rede, ou usam os valores do espectro diretamente em modelos de classificação sem prévio tratamento de dados.

Esta Dissertação propõe mecanismos para deteção de anomalias, assegurando simultaneamente a privacidade dos seus nós, que se baseiam na análise de atividade rádio na camada física, medindo a duração de períodos de silêncio e atividade. Depois da extração de propriedades que caracterizam estes períodos, é realizada uma exploração dos dados e um estudo das mesmas, sendo depois usadas para treinar modelos de classificação mono-classe, tanto usando algoritmos clássicos como redes neurais.

Os modelos são treinados com dados retirados de uma série de interações com um Amazon Echo, primeiramente num ambiente sem ruído, numa tentativa de simular um cenário de automação doméstica simplificado, e seguidamente, num laboratório onde existia bastante atividade gerada por uma série de dispositivos, assim como interferências. De seguida, os modelos foram testados com dados semelhantes mas contendo um nó comprometido, que periodicamente enviava um ficheiro para uma máquina local. Os dados mostram que, em ambas as situações, foi possível atingir taxas de precisão de deteção na ordem dos 99%.

Este trabalho também propõe uma arquitetura para integrar os modelos previamente validados em ambientes de produção. Nesta arquitetura é definido todo o percurso dos dados, que são capturados e processados pelos *sniffers*, enviados para um *broker*, e lidos pela instância de classificação correspondente no servidor central. Este “servidor” é responsável por gerir as instâncias de consumo de dados/classificação, armazenar as janelas de *features* e a respetiva etiqueta, e por retreinar os modelos periodicamente para que estes acompanhem as alterações dos padrões da rede. Foi realizada uma série de testes para verificar se a plataforma é capaz de escalar com um aumento do número de *probes*, mostrando que, devido a limitações de memória, é recomendável dividir os classificadores por diversas máquinas.

Keywords

IoT Networks, Network Monitoring, Radio Signal Monitoring, Anomaly Detection, One-Class Classification, Machine Learning.

Abstract

With the advent of wireless mesh networks and the Internet of Things (IoT), security risks inherent to these types of networks, either non-authorized use of the network or data exfiltration, have grown in number. Most of the approaches currently available for anomaly detection in IoT networks perform frame and packet inspection, which may inadvertently reveal the private behavioral patterns of its users. Additionally, those whose focus falls on the physical layer data often use Received Signal Strength Indicator (RSSI) as a distance metric and perform anomaly detection according to the nodes' relative distance, or use spectrum values directly as inputs of classification models without any data exploration.

This Dissertation proposes privacy-focused mechanisms for anomaly detection, which analyses radio activity at the physical layer, measuring silence and activity periods. We then extract features from the duration of these periods, perform data exploration and feature engineering, and use them for training both classical and neural network approaches of One-Class Classification (OCC) models.

We train our models with data captured from interactions with an Amazon Echo, first on a noise-free environment, simulating a home-automation scenario, and second with multiple devices generating background data exchanges on a lab full of devices and interference. We then test them against similar scenarios with a tampered network node, periodically uploading data to a local machine. Our data show that, in both situations, the best performing model is able to detect anomalies with a 99% precision rate.

This work also proposes a framework for deploying the validated models into a production environment. This proposal defines the entire data pipeline, which is recorded and processed at the sniffers, sent to a message broker, and consumed by the corresponding probe's classifier instance at a central server. This "server" is responsible for managing the consumer/classifier instances, storing the windows of features and respective labels, and periodically re-train the models so that they can adapt to the behavioral changes on the network. We performed series of tests to assert if this architecture is able to scale with a higher number of probes; these tests showed that, due to memory constraints, it is advisable to split the data consumers and classifiers across different physical hosts.

Contents

Contents	i
List of Figures	v
List of Tables	ix
Glossary	xi
1 Introduction	1
1.1 Motivation	1
1.2 Objectives	2
1.3 Contributions	3
1.4 Document Structure	4
2 State of the Art	5
2.1 Internet of Things (IoT)	5
2.1.1 IoT Information Gathering	7
2.1.2 IoT Security Issues	8
2.2 Monitoring	11
2.2.1 Intrusion Detection Systems (IDS)	12
2.2.2 Intrusion Detection Systems for IoT networks	13
2.2.3 Network Anomaly Detection	14
2.3 Spectral Analysis	15
2.3.1 Spectral Anomaly Detection	16
2.3.2 Wireless Anomaly Detection	17
2.3.3 Software Defined Radio (SDR)	19
2.4 Time-series Analysis	21
2.4.1 Discrete Fourier Transform (DFT)	21
2.4.2 Continuous Wavelet Transform (CWT)	22
2.5 Machine Learning	24

2.5.1	Machine Learning Use Cases	25
2.5.2	Feature dimensionality	26
2.5.3	One-class Classification	28
2.5.4	Ground-truth Evaluation	39
2.6	Summary	40
3	Physical Layer Anomaly Detection Mechanisms	43
3.1	System Overview	43
3.2	RSSI Data Collection	44
3.3	Data Preprocessing	46
3.4	Feature Extraction	47
3.5	Feature Dimensionality	50
3.6	Model Training and Evaluation	52
3.7	Summary	54
4	Anomaly Detection in IoT Networks	57
4.1	IoT Home Scenario	57
4.1.1	Dataset Exploration	59
4.1.2	Feature Dimensionality	64
4.1.3	Classification Results	66
4.2	IoT Work/Lab Environment Scenario	68
4.2.1	Dataset Exploration	69
4.2.2	Feature Dimensionality	73
4.2.3	Classical Models Classification Results	74
4.2.4	Neural Networks Classification Results	76
4.3	Conclusion	85
5	Framework for Live Anomaly Detection in IoT Networks	87
5.1	Architecture Overview	87
5.1.1	Probe Data Extractor (PDE)	88
5.1.2	Network Windows Message Broker (NWMB)	91
5.1.3	Anomaly Detection Server (ADS)	91
5.1.4	Feature Verification API	93
5.1.5	Resource Monitor	93
5.2	Scalability Tests	94
5.3	Conclusion	97
6	Conclusions and Future Work	99
6.1	Future Work	101

List of Figures

1.1	Traditional network monitoring per Open Systems Interconnection (OSI) layer.	2
2.1	Prediction of the number of Internet of Things (IoT) devices installed (2014-2020), and market value of IoT in North America (2017-2022).	6
2.2	Global IoT market forecast from 2018 to 2023.	7
2.3	The top 10 IoT segments in 2018, based on 1600 real projects.	7
2.4	IoT security architecture [4], [5].	8
2.5	Top 10 IoT vulnerabilities in 2018, according to OWASP.	10
2.6	Typical SDR architecture [37].	20
2.7	Periodogram of a 100 Hz sine wave in additive $\mathcal{N}(0, 1)$ noise.	22
2.8	Examples of mother wavelets.	23
2.9	Computation of scalogram by averaging the result of wavelet transform over time ($\tau' \in T$).	24
2.10	Feature importance using tree based classifiers.	27
2.11	Dimensionality reduction from the original 3D to a 2D space, using PCA.	28
2.12	Point and context anomalies.	29
2.13	Threshold on a 1-dimensional Gaussian distribution [55].	30
2.14	Edge definition of a GMM [60].	31
2.15	Classification of a new example using kNN on a 2-class problem.	32
2.16	Basic AE architecture.	34
2.17	Simple VAE architecture.	35
2.18	Simplified version of a GAN architecture.	36
2.19	Structure steps to isolate anomaly and non-anomaly points.	36
2.20	Boundary representation of an OC-SVM and a SVDD.	39
2.21	Timeline example of an anomaly showing up every 4 windows.	40
3.1	Overall system overview.	44
3.2	RSSI spectrum obtained by computing the FFT over a IQ signal.	45
3.3	Extraction of multiple time-series data from the FFT output.	45
3.4	Extraction of activity periods from the RSSI signal.	46

3.5	Post processing of activity periods.	47
3.6	“Hard” vs. “sliding” windows approach.	48
3.7	Diagram of features extracted for every window size.	49
3.8	Scalogram performed over 64 scales distributed across 1 and 600 seconds.	50
3.9	Classification system used across different anomaly datasets.	51
3.10	Illustration of mini-batch approach cross-validation splitting.	52
3.11	Splitting of the training datasets into training and cross-validation sets.	53
4.1	Home scenario network diagram.	58
4.2	RSSI power values over time in three different datasets.	60
4.3	RSSI power values over time in two anomalous datasets.	60
4.4	RSSI power values over time in two anomalous datasets, masked by Alexa and YouTube traffic.	61
4.5	Activity timeline with and without <i>non-relevant activity filtering</i>	61
4.6	Comparison of mean silence and activity periods, without <i>non-relevant activity filtering</i>	62
4.7	Comparison of mean silence and activity periods, with <i>non-relevant activity filtering</i>	62
4.8	Comparison of the 99th percentile silence and activity periods, without <i>non-relevant activity filtering</i>	63
4.9	Comparison of the 99th percentile silence and activity periods, with <i>non-relevant activity filtering</i>	63
4.10	Scalogram of the activity timeline on clean and multiple outlier behaviors.	64
4.11	<i>F1-Score</i> for multiple statistical feature subsets, using feature selection and PCA.	65
4.12	<i>F1-Score</i> for multiple wavelet local maxima feature subsets, using feature selection and PCA.	65
4.13	<i>F1-Score</i> for multiple scalogram feature subsets, using feature selection and PCA.	65
4.14	Work/lab network diagram.	68
4.15	RSSI power values over time in three different datasets.	70
4.16	Activity timeline with and without <i>non-relevant activity filtering</i>	70
4.17	Comparison of mean silence and activity periods, without <i>non-relevant activity filtering</i>	71
4.18	Comparison of mean silence and activity periods, with <i>non-relevant activity filtering</i>	71
4.19	Comparison of maximum silence and activity periods, without <i>non-relevant activity filtering</i>	71
4.20	Comparison of maximum silence and activity periods, with <i>non-relevant activity filtering</i>	72
4.21	Scalogram of the activity timeline on clean and multiple outlier behaviors.	72
4.22	<i>F1-Score</i> for multiple statistical feature subsets, using feature selection and PCA.	73
4.23	<i>F1-Score</i> for multiple wavelet local maxima feature subsets, using feature selection and PCA.	74
4.24	<i>F1-Score</i> for multiple scalogram feature subsets, using feature selection and PCA.	74
4.25	Final AE architecture.	77
4.26	AE MSE values of clean and anomalous samples.	78
4.27	Variational Autoencoder (VAE) architecture.	79

4.28	Generative Adversarial Network (GAN) architecture.	80
5.1	Overall live anomaly detection framework overview.	88
5.2	Framework messaging times comparison (1 to 20 nodes).	95
5.3	Framework CM processing times comparison (1 to 20 nodes).	95

List of Tables

2.1	Summary of IEEE 802.11 variants [28], [29].	18
4.1	Ranges of parameters used on classical OCC models tuning.	66
4.2	Home automation scenario classification results, when not using <i>non-relevant activity filtering</i>	67
4.3	Home automation scenario classification results, when using <i>non-relevant activity filtering</i>	67
4.4	Average training and testing durations in the home automation scenario, with a $\tau = 1$ and $v = 5$	68
4.5	Work/lab scenario classical models' classification results, when not using <i>non-relevant activity filtering</i>	75
4.6	Work/lab automation scenario classical models' classification results, when using <i>non-relevant activity filtering</i>	75
4.7	Work/lab automation scenario AE classification results.	78
4.8	Work/lab automation scenario VAE classification results.	79
4.9	Work/lab automation scenario GAN classification results.	81
4.10	Work/lab automation scenario DualAE classification results.	82
4.11	Work/lab automation scenario AE-GAN classification results.	83
4.12	Work/lab scenario neural networks' classification results, when not using <i>non-relevant activity filtering</i>	84
4.13	Work/lab scenario neural networks' classification results, when using <i>non-relevant activity filtering</i>	84
4.14	Average training and testing durations in the work/lab scenario, with a $\tau = 1$ and $v = 5$	85
5.1	Comparison of average, minimum and maximum messaging and CM processing times for multiple number of nodes.	95
5.2	Comparison of Docker host resource usage, for multiple number of nodes.	96

Glossary

6LoWPAN	IPv6 over Low-Power Wireless Personal Area Networks	HIDS	Host-based Intrusion Detection System
AAE	Adversarial Autoencoder	HTTP	Hypertext Transfer Protocol
ADC	Analogue-to-Digital Conversion	IDFT	Inverse Discrete Fourier Transform
ADS	Anomaly Detection Server	IDS	Intrusion Detection System
AE	Autoencoder	IEEE	Institute of Electrical and Electronics Engineers
AE-GAN	Autoencoder-Generative Adversarial Network	IF	Isolation Forest
AI	Artificial Intelligence	I/O	Input/Output
AP	Access Point	IoT	Internet of Things
API	Application Programming Interface	IP	Internet Protocol
ARP	Address Resolution Protocol	IPS	Intrusion Prevention System
ASIC	Application Specific Integrated Circuit	IPSec	IP Security Protocol
CM	Classification Manager	IQ	in-phase and quadrature
CNN	Convolutional Neural Network	JSON	JavaScript Object Notation
CPU	Central Processing Unit	KDE	Kernel Density Estimation
CUDA	Compute Unified Device Architecture	KL	Kullback-Leibler
CV	Cross-validation	kNN	k-Nearest Neighbours
CWT	Continuous Wavelet Transform	LAN	Local Area Network
DAC	Digital-to-Analogue Conversion	LNA	Low-Noise Amplifier
DDoS	Distributed Denial-of-Service	LOF	Local Outlier Factor
DFT	Discrete Fourier Transform	LSTM	Long Short Term Memory
DL	Deep Learning	LTE	Long Term Evolution
DNS	Domain Name Service	MAC	Medium Access Control
DoS	Denial-of-Service	mDNS	Multicast DNS
DSP	Digital Signal Processing	ML	Machine Learning
DualAE	Dual Autoencoder	MLP	Multi Layer Perceptron
FFT	Fast Fourier Transform	MSE	Mean Squared Error
FPGA	Field-programmable Gate Array	NIDS	Network-based Intrusion Detection System
GAN	Generative Adversarial Network	NWMB	Network Windows Message Broker
GMM	Gaussian Mixture Model	OCC	One-Class Classification
GPU	Graphic Processing Unit	OC-SVM	One-Class Support Vector Machine
GRBF	Gaussian Radial Base Function	OFDM	Orthogonal Frequency-Division Multiplexing
GPP	General Purpose Processor	OSI	Open Systems Interconnection
		OWASP	Open Web Application Security Project

PAN	Personal Area Network	SNMP	Simple Network Management Protocol
PCA	Principal Component Analysis	SOM	Self-Organizing Map
PDE	Probe Data Extractor	SQL	Structured Query Language
RAM	Random Access Memory	SSDP	Simple Service Discovery Protocol
ReLU	Rectified Linear Unit	SVM	Support Vector Machine
RF	Radio Frequency	TCP	Transport Control Protocol
RFID	Radio-Frequency Identification	TCP/IP	Transport Control Protocol/Internet Protocol
RM	Resource Monitor	UDP	User Datagram Protocol
RPL	Routing over Low Power and Lossy Networks	VAE	Variational Autoencoder
RSSI	Received Signal Strength Indicator	VGA	Variable-Gain Amplifier
SAD	Spectral Anomaly Detection	VXLAN	Virtual Extensible LAN
SDR	Software Defined Radio	WLAN	Wireless Local Area Network
SGD	Stochastic Gradient Descent	WSGI	Web Server Gateway Interface
SVDD	Support Vector Data Descriptor	WSN	Wireless Sensor Network
SNARC	Stochastic Neural Analog Reinforcement Computer		

Introduction

This chapter describes the overall motivation for addressing the topics of this Dissertation, the background of the Internet of Things (IoT) and machine learning areas, and other contributions that focused on similar issues. In this chapter, we also address the main contributions of this work and a summary of the structure of the document.

1.1 Motivation

With the advent of IoT and wireless mesh networks, security risks inherent to these types of networks, namely unauthorized use of resources and data exfiltration, have grown in number, and severity. Currently, embedded sensors have become the trend, being used in both home automation and critical and sensible infrastructures. IoT devices are always a security liability due to their heterogeneity, lack of constant monitoring, their placement in open wireless mediums, their limited computational resources, and, more recently, the possible 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 and hardware resources, or just performing Denial-of-Service (DoS) attacks to disrupt the entire network. Nevertheless, many accidents are also seen as outliers, such as sudden router overloading due to another router malfunctioning. The spike in total of network attacks, their severity and complexity has forced administrators to use tools that rely on anomaly analysis to detect new and unforeseen phenomena, rather than solutions that look for traditional and well-known attacks.

However, the majority of current anomaly detection solutions are solely based on frame or packet inspection, allowing easy discrimination of users by their Medium Access Control (MAC)

or Internet Protocol (IP) addresses. This means that the communication behaviors of each device could be inferred, ultimately raising privacy concerns, as illustrated in figure 1.1. On the other hand, inferring data exchange patterns at the physical layer is a more complex task, as a single signal power indicator may correspond to a mixture of simultaneous data transmissions. Besides masking individual behavior, Open Systems Interconnection (OSI) layer 1 data also makes it harder to monitor the overall behavior of the network.

Figure 1.1: Traditional network monitoring per OSI layer.

1.2 Objectives

This dissertation aims at detecting anomalous behavior in IoT networks without compromising user or node privacy. Even if the majority of current solutions do not study individual users' behavior on the network, they do not guarantee their privacy since they have access to data that may disclose it, such as MAC or IP addresses. Moreover, there are many scenarios where data link layer traffic is encrypted (e.g., Virtual Extensible LAN (VXLAN) over IP Security Protocol (IPSec) tunnel), which forces administrators to use physical layer data to perform network monitoring. However, modeling physical layer behavior is not as easy as on upper layers. First, one cannot discriminate individual flows of data like when inspecting the network layers. Furthermore, we are not provided with as much data as on upper layers since, for instance, IP packets and Transport Control Protocol (TCP) segments allow one to perform statistical analysis over payload sizes and periodicity of determined TCP flags. In layer 1, the only available data are power indicators throughout the time, which makes it much more challenging to differentiate network behavior. Additionally, there are a number of approaches that only resort to spectral data to perform anomaly detection. However, the majority use them “blindly” as inputs of reconstruction models (i.e., models that try to minimize the reconstruction error of “clean” data), and do not perform any network behavior analysis.

Moreover, many studies rely on modelling well-known attacks to detect outlier behavior, which makes them vulnerable to unseen phenomena. These works often resort to supervised learning techniques and feed them with labeled features characterizing certain attack-vectors and “clean” traffic. Two things can happen when those models are facing unseen patterns: either it classifies it as one of the known out-of-normal classes, or it classifies it as non-anomaly. The latter will happen much more frequently as the majority of attacks have a singular behavior. Additionally, unsupervised learning approaches also require both “clean” and anomalous samples so that they can separate both, even if the data is not labeled. We plan on using One-Class Classification (OCC) to address these issues, as the latter only use anomaly-free data as baseline for outlier detection. These techniques, also known as novelty detection, usually perform better at outlining data not previously seen than classical supervised learning.

Different scenarios must be designed to validate the detection accuracy of the proposed mechanisms. First, it is required to ensure the validity of the recorded data; thus, it is essential to start in a scenario where background noise and other interferences are not an issue. Then, one can move on to more complex scenarios, and try to detect anomalies in environments with regular background data exchanges to mask the outlier behavior.

Last but not least, deploying the machine learning models into production environments is still a big challenge, as many get stuck in research and development. It is essential to have a data pipeline that captures real-time data and perform accurate classification, so that administrators are aware of the current state of their networks.

1.3 Contributions

Taking into account the current security challenges IoT networks face, the developed work can be summarized into four points:

- The proposal of privacy-focused anomaly detection mechanisms for IoT networks, by relying exclusively on physical layer data as input data for the system.
- Accurately model anomaly-free network behavior, and detect a broad range of outliers, by using both classical and deep learning approaches of One-Class Classification (OCC) models.
- Validate the proposed mechanisms by designing two use-cases where the goal was to identify periodic outlier behavior on a Wi-Fi channel. In both, the behavior of the tampered node is masked by data exchanges of other devices in the network.
- Design a scalable architecture to deploy validated machine learning models that can accurately detect the presence of outlier behavior in the network in real-time.

The anomaly detection mechanisms and one of the scenarios has already been depicted in a conference paper submitted to 17th IEEE/IFIP Network Operations and Management Symposium (NOMS 2020). Additionally, a journal paper depicting both classical and neural network approaches, as well as the framework for deploying those models, is being written.

Finally, these mechanisms and some of the results were already presented at DS@Academy, during the DSPT Day 2019, organized by the Data Science Portugal ¹ team.

1.4 Document Structure

This document is structured as follows:

- **Chapter 2 - State of the Art:** it presents the background of current issues and solutions to secure IoT networks, as well as current works regarding physical layer monitoring. Furthermore, it addresses time-series analysis and machine learning techniques currently used for OCC.
- **Chapter 3 - Physical Layer Anomaly Detection Mechanisms:** this chapter showcases the proposed mechanisms for detecting anomaly in layer 1. It depicts the entire data pipeline, by converting the in-phase and quadrature (IQ) components data into the frequency domain, finding activity and silence periods, and extracting relevant features for classification. It also demonstrates how those features should be treated, and how to properly train the OCC models that will be in charge of labeling each sample.
- **Chapter 4 - Anomaly Detection in IoT Networks:** it depicts the two scenarios, one with few devices and the other with higher number of connected nodes, that were designed to validate the mechanisms presented in the previous chapter. It showcases a careful Received Signal Strength Indicator (RSSI) and feature analysis, and the results that OCC models obtained.
- **Chapter 5 - Framework for Live Anomaly Detection in IoT Networks:** this chapter presents a scalable framework for real-time network behavior classification, by gathering RSSI data from different network probes and labelling it in a separate server.
- **Chapter 6 - Conclusions and Future Work:** this chapter summarizes the work presented through chapters two to five, and depicts the final conclusions of this dissertation. It finalizes this dissertation by proposing future developments in the issued area.

¹<https://www.datascienceportugal.com/>

State of the Art

This chapter presents the state of the art and background of the five main topics addressed by this work. The first regards to the current state of IoT networks, more explicitly detailing its use cases and security issues, many due to its devices being heterogeneous, low-power, and subject to trivial security policies.

The second part focuses on monitoring strategies, how they are implemented in general Intrusion Detection Systems (IDSs), and those designed specifically for the IoT, as well as an overview of anomaly detection methods used in network monitoring. This led us to believe that there is still no valid solution for physical layer anomaly detection.

Third, this chapter presents a series of spectral analysis techniques used in active research on anomaly detection. Furthermore, it showcases a series of works on detecting outlier behavior on common wireless technologies such as Wi-Fi or Zigbee.

The fourth section presents some methodologies used in time-series analysis: both the Discrete Fourier Transform (DFT), used to transform data from the time to the frequency domain, and the Continuous Wavelet Transform (CWT), that acts like a hybrid approach, providing insights in both the frequency and time domains simultaneously.

The final section illustrates a series of machine learning techniques for anomaly detection, particularly One-Class Classification (OCC) models, and feature engineering techniques, such as feature selection and feature dimensionality reduction. Besides classical OCC models, such as One-Class Support Vector Machines (OC-SVMs), neural network approaches are also described. Many of these techniques will be used for anomaly detection of physical layer behavior, as described in the following chapters.

2.1 Internet of Things (IoT)

We live in a data-driven age. In the past few years, a vast technological revolution took place, leading up to a trend of embedded sensors. They are being used in fridges, light bulbs,

and door locks, and they are collecting data every single second, monitoring the life and the surroundings of those who interact with them daily.

The term IoT, first introduced by Kevin Ashton in 1999 [1], usually points to a wireless-connected mesh of smart sensors with constant data flow that require minimal human interaction, typically associated to the execution of specific commands (e.g., using a mobile application to turn on smart lights). Their ultimate goal is to automate, facilitate, and improve our daily activities and needs [2].

Figure 2.1: On the left, prediction of the number of IoT devices installed from 2014 up to 2020. On the right, the prediction of the market value of IoT in North America from 2017 to 2022. Source: Statista ¹.

The Internet of Things is one of the most significant innovations of the twenty-first century, and it is taking over of our day-to-day. According to a collection of studies gathered by Louis Columbus [3], the number of installed IoT devices in 2014, both in consumer and business markets, grasped 5 billion units, as illustrated in figure 2.1, whereas, in 2018, it had already hit the 10 billion mark. The forecast shows no signs that this impressive growth is slowing down any time soon, as the number of IoT devices is expected to almost double by 2020. On the right, the same figure shows that the current \$280 billion dollar market in North America is expected to grow over the next years, hitting \$500 billion by 2022. Globally, the scenario is identically positive, according to a study conducted by IoT Analytics ² in 2018 (figure 2.2).

The sensors use cases range from home automation to critical and sensible infrastructures like connected industry machinery or health/medical-related applications. Many other fields are reinventing themselves due to significant investment in these technologies. Over the last few years, there has also been a great effort in the development and introduction of smart city infrastructures. According to a study conducted by IoT Analytics ² in 2018, also mentioned in [3], smart cities projects rank at the top in the total number of proposed IoT projects. Figure 2.3 shows the top ten of the proposed IoT projects in 2018.

¹<https://www.statista.com/>

²<https://iot-analytics.com/>

Figure 2.2: Global IoT market forecast from 2018 to 2023. Source: IoT Analytics ².

Figure 2.3: The top 10 IoT segments in 2018, based on 1600 real projects. Source: IoT Analytics ².

2.1.1 IoT Information Gathering

The IoT is often considered a group of devices and services capable of communicating and interacting between them with minimal human intervention. Most of the time, they work independently of their surroundings, but they can also be tightly coupled with what happens in their environment. Regarding their interaction with the environment, IoT devices can be classified as:

- *Non-environment dependent*, which, as the name indicates, work independently of what happens in their surroundings, not waiting for any instruction or external occurrence to transmit or process data. They often gather data and communicate with other nodes or external servers periodically. A temperature sensor uploading data every second to a central server is an example of these devices.

- *Environment dependent*, when part or the totality of their operation depends on an external interaction or occurrence to trigger some action. Personal assistants hubs are an example of these devices and, just like the example provided in the last bullet point, a temperature sensor that only triggers an alarm to a given server when exceeds a certain threshold.

2.1.2 IoT Security Issues

IoT devices are always a security liability due to their heterogeneity, lack of constant monitoring, their placement in open wireless mediums, their limited computational resources and, more recently, the possible access to locally-stored sensitive data. Moreover, they are now an integral part of the management of sensible infrastructures, such as energy and water suppliers.

Although the Transport Control Protocol/Internet Protocol (TCP/IP) or the OSI models are commonly used to define the communication between computer systems, the IoT communication stack is usually split into three layers: the *perception* (equivalent to the physical layer on the OSI model and the link layer on the TCP/IP model), the *network* (equivalent to the network and transport on the OSI model and the internet and transport layers on the TCP/IP model) and the *application* layers [4], [5].

Figure 2.4: IoT security architecture [4], [5].

There is a number of vulnerabilities across all these layers, either being insufficient authentication mechanisms between devices - leading to personification attacks -, insecure network services, lack of data encryption and integrity verification - allowing the interception and/or modification of exchanged messages -, the imminent risk of DoS, or the lack of prevention regarding flooding and *black hole* attacks. Furthermore, we cannot disregard the careless maintenance of IoT devices, as many run insecure, and outdated software or firmware

with public exposed vulnerabilities. Their wireless nature makes them also prone to easy physical access, a problem like any other already mentioned [6].

In 2015, U. Farooq *et al.* [7] surveyed a series of IoT security challenges over the different layers of the communication stack. The *perception layer* is tightly coupled with the physical media of data exchanges, encompassing many technologies, such as Radio-Frequency Identification (RFID) or Zigbee. In IoT, the one thing they usually all have in common is the use of the wireless medium for communication, which means that the signal is publicly available for anyone to monitor, intercept and even purposely jam it [4]. For instance, the lack of authentication mechanisms may lead to unauthorized access to RFID tags or even end up cloned. Besides all the technological shortcomings, since many sensors are located in public areas, taking over a given node can be as simple as having physical access to it, which may open doors to radio credentials and other sensitive information, compromising not only a single node, but the entire network.

The *network layer* also falls under a variety of attacks regularly. Nonetheless, one has to take into account that many of these intrusions target Wireless Sensor Networks (WSNs), a subset of IoT networks. Some of the mentioned strikes are the following [5], [7]:

- **Sybil** is an attack where a node presents itself as multiple identities, giving the system a false sense of redundancy, thus compromising communications and degrading network performance;
- **Sinkhole** makes a given node attract all the traffic towards it and then dropping all or the majority of it, therefore, its *sinkhole* analogy;
- **Sleep deprivation** consists of keeping the nodes awake when, most of the time, they should be on low power mode to save battery life. Devices would start shutting down, and ultimately, all the network nodes would get turned off, culminating in an unavailable system;
- A **DoS** occurs when the attacker floods the network with a massive amount of useless traffic, degrading its performance to the point that makes it unavailable;
- **Malicious code injection** occurs when the attacker manages to inject code on the system to shut down or control over a node on the network;
- A **Man-in-the-Middle** attack happens when the attacker targets a connection between nodes and manages to monitor and even control it.

Literature often defines an intermediate *middleware* layer between *network* and *application* zones [7], [8], whose responsibility is to enable effortless communication between heterogeneous IoT devices, providing scalability, security, and privacy when dealing with personal and/or industry-related data. However, it often sees itself under the scope of several attacks. If attackers can bypass security, they can prevent access or even delete sensitive information, especially the one available in cloud and data storage IoT environments. DoS and malicious insider attacks - someone from the inside initiates an attack to benefit a third party - are also regularly exploited.

The *application layer* is the one that is more prone to security breaches, due to its inherent diversity and variety of use cases, which range from smart homes to farms and even supply

chains. Malicious code injections, DoS attacks, spear-phishing attacks - emails sent to high-ranked personnel on a company - and sniffing attacks (after forcing a sniffer into the system, the attacker can monitor the network and the data that flows through it) are examples of application-level strikes.

Open Web Application Security Project (OWASP) ³ keeps a list of the top 10 security issues regarding IoT networks, which was refreshed back in 2018, as illustrated in figure 2.5.

Figure 2.5: Top 10 IoT vulnerabilities in 2018, according to OWASP.

The vulnerabilities mentioned above have already been in the news spotlight. A massive IoT botnet attack Mirai took place in October 2016 [9], infecting numerous IoT devices to later use them in order to perform a massive Distributed Denial-of-Service (DDoS) flood attack at Domain Name Service (DNS) provider Dyn. The raid resulted in several major websites, such as GitHub, Netflix, Shopify, SoundCloud, Spotify, and Twitter, having their services unavailable. Mirai took advantage of devices running an older version of the Linux kernel when configured with the default access credentials.

In November of the same year, another DDoS attack was conducted to shut down the heating of two buildings in Lappeenranta, Finland, by continuously rebooting the system.

³<https://www.owasp.org/>

Consequently, the heating system would never start properly.

In 2017, a similar attack to Mirai was rolled out, also relying on outdated devices and poor management, once more taking advantage of default credentials. It was named Brickerbot since it just bricked the device, instead of taking over it to later use it for an attack propagation. In companies relying on IoT devices to handle critical operations, this assault could be particularly dangerous.

2.2 Monitoring

As mentioned in section 2.1.2, a high number of vulnerabilities are still present in IoT networks. For manufacturers, balancing both production costs and implementation of advanced security policies is a hard goal when there is such heterogeneity of device architectures and a wide range of applications, especially when the end product has limited processing power and the majority run on battery-only power. This forces administrators to use monitoring strategies to keep the network secure.

Rather than implementing strong authentication and encryption algorithms to comprise proper security mechanisms, one can keep track of what is happening in the network at any given moment by registering a series of events throughout the time. Monitoring is not exclusive to the security surveillance scenario; yet, network administrators usually implement mechanisms to inspect the overall network and inside systems, looking for problems caused by crashed or overloaded servers, or power outages.

One can split network monitoring between two branches [10]–[12]:

- **Active monitoring** works by pushing new traffic to see how the network behaves, or by actively gathering data from nodes, such as live application log collection. However, injecting traffic into the network adds overhead both on the network and on the hardware, which can ultimately “kill” them if overused. On the other side, it can be particularly useful since it can register real-time data on the network performance and can be used to evaluate how a specific application behaves, even though it is synthetic traffic and it may not be completely representative of a real-life scenario. Depending on the use case, having a narrower view of the network when compared with passive monitoring may be a disadvantage. There are a variety of tools used in active monitoring, such as the injection of Simple Network Management Protocol (SNMP) queries to obtain or even change settings on devices on the network environment, probing tools such as `ping` and `traceroute` to check for connectivity, and scanners, such as `nmap`, to scan a given network (e.g., check the IP addresses of active machines on a subnet).
- **Passive monitoring** works by directly scanning and collecting the traffic flowing through the network, usually to obtain statistics of the network behavior. Large volumes of data are ideal for performing predictive analysis on intrusions and bandwidth usage baselines, commonly through machine learning strategies. It is easier on the network and the monitoring hardware, although it mainly relies on historical data to classify the

traffic, as opposing to active monitoring, which is able to pull real-time data from the network.

In an ideal scenario, to perform optimal network monitoring, both active and passive techniques must be combined to attain a far more effective analysis of the environment.

On the other hand, taking into account the characteristics of nodes and the network being monitored, it may be crucial to store the least amount of data as possible. On a high-traffic network, saving all this information would consume resources far beyond their capacity, and it would be almost impossible to perform a decent analysis without a warehouse-scale computer cluster.

As previously mentioned, applying security mechanisms in every node of the network may be unpractical due to its power and processing limitations. Moreover, if one assumes that continuously extracting information from a single node associated with a given user can result in inferring the user's typical behavior and traffic characteristics, then user privacy could be compromised and possibly violated by the monitoring system.

2.2.1 Intrusion Detection Systems (IDS)

Intrusion Detection System (IDS) are one of the tools that network administrators rely on to perform monitoring. They are often considered event-action sensors (device or software application) that rely on the data gathered from network packet inspection or hosts' logs and resource usage information to look for any activity or policy violation that may be harmful to the network and its nodes [13]. Intrusion Detection Systems are often built as binary classifiers; thus they either classify the extracted data as clean traffic, and no further action needs to be taken, or as an outlier, triggering an alert due to abnormal behavior. IDSs work as event sensors, since they generate an event (alert) upon a violation, but do not take any action to counter it. On the other hand, Intrusion Prevention Systems (IPSs), very similar in operation, take action upon an event that triggers the alert, possibly blocking the traffic that made that warning go off [14]. Usually, IDSs are designed to detect both internal and external attacks, i.e., generated by an internal node (or a group of them) that has already been compromised spreading the attack across the local network, or originated by an entity outside of the local network, respectively.

The output of IDSs is considerably straightforward, i.e., it either outputs a "normal" traffic tag or an alert tag. Several problems are likely to be pointed out when taking further consideration upon this operation mode. One is when traffic behavior becomes indistinguishable from harmless activity. Depending on the situation, this can be considered a common scenario. For example, in a shopping center, where multiple Access Points (APs) provide internet access to clients, a Distributed Denial-of-Service (DDoS) attack can look similar to a sudden increase in the number of connected clients and respective devices. Another critical issue is that, even a system with a real small false positive rate, can trigger hundreds or thousands of false alerts in small amounts of time, especially if the monitored network has a considerable large amount of nodes. Lastly, considering that attackers are often skilled

enough to perform an intrusion going unnoticed, it is almost impossible to track down those types of behaviors [14].

Over the years, a series of studies have proposed a taxonomy to classify IDSs [15], [16], but there is still no universal consensus. Sabahi and Movaghar [17] synthesized several works into creating a taxonomy that classifies IDS systems based on five criteria: information source, analysis strategy, time aspects, architecture, and response.

Nonetheless, a considerably simpler taxonomy is followed by most of the literature [14] as IDSs are often classified by their action domain (information source) and their decision-making process (analysis strategy). The action domain regards to where the IDS is operating; a **Network-based Intrusion Detection System (NIDS)** operates at the network domain (physical layer and packet capture data), while a **Host-based Intrusion Detection System (HIDS)** works at a host/node level (log and resource usage inspection, resorting to tools such as SNMP). The second criterium splits them into **signature-based** systems (most of the commercial anti-virus solutions), **anomaly-based** systems (which try to detect atypical phenomena) and **specification-based** systems (a hybrid between the previous two) [13].

Signature-based IDS is a simple approach to known security attacks. It matches the current behavior of the network against the signature, i.e., the behavior and pattern, of a well-known attack/intrusion stored in a database. Although easily implemented, it has its limitations, since it is not prepared to detect attacks whose signature is not present in the database. In order to make the IDS detect new patterns, one must manually insert them into the database.

An **Anomaly-based IDS** is the opposite of its signature-based counterpart. It works by learning how the network behaves in a typical way and detects malicious activity when it behaves differently. Network behavior is usually defined by an automated training, while statistical modeling detects outliers with minimal false-positive rates. It is way more precise, efficient, and reliable than a signature-based system, but they are sometimes used together to complement each other.

Lastly, there are **Specification-based IDS**, which are similar to anomaly-based systems. However, instead of using machine learning algorithms to learn the normal behavior of the network, typical network patterns are manually specified, which is precisely the main downside of this approach.

2.2.2 Intrusion Detection Systems for IoT networks

Currently, the majority of available IDSs are either designed for WSNs or the conventional IT infrastructure [2]. It is essential to mention that a WSN is different from an IoT network. Initially, WSNs were designed for local networks, although many of its applications also take advantage of the Internet. On the other hand, the Internet is a nuclear part of the IoT architecture. Right now, WSNs are considered a subset of IoT networks; therefore, IDSs targeted at WSNs cannot be used in IoT. IDSs used on typical IT networks also cannot be used in IoT networks since they were not designed considering the heterogeneity, scale, and use cases of an IoT ecosystem.

It is still a challenge to develop a proper IDS for the IoT since embedded systems often rely on lower-power Central Processing Units (CPUs). This forces developers to ditch advanced communication encryption protocols, trusting the security of the environment to the IDS.

Recent studies show an effort in developing custom IDSs for IoT networks. R. Stephen and L. Arockiam [18] attempted to develop an IDS to detect Sinkhole attacks over the Routing over Low Power and Lossy Networks (RPL) protocol (mainly used in WSNs). Razza et al. created SVELTE [19], a real-time IDS directed at IoT networks, whose main goal was to fight the existing vulnerabilities in meshes running IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN) protocol. Pulse [2] is yet another IDS for the IoT, which detected DoS attacks by using supervised learning techniques, more specifically, a Naive Bayes classifier with clean and malicious datasets.

The mentioned studies target some of the current issues mitigating IoT networks. However, they have a limited range of action/detection, as they only address specific attacks; thus, one can classify them as signature/rule/model-based solutions.

One example of anomaly-based approaches is N-BaIoT [20], a network-based anomaly detection system that uses autoencoders reconstruction errors to detect anomalous behavior in the network, namely botnet attacks. The authors were able to obtain zero-valued false-positive rates on some devices, while in others, the algorithm struggled to model its usual behavior on the network.

2.2.3 Network Anomaly Detection

A network anomaly can be defined as a sudden and often short-duration deviation of the normal behavior of a given network. Some are originated from intruders with malicious intent, trying to steal information or hardware resources, or just performing DoS attacks to disrupt the entire network. On the other hand, other outliers are mere accidents, such as a sudden malfunction in a router that forces all the traffic to deviate through another output router that eventually ends up overloading. In any current scenario, quick detection and intervention are necessary for any network to make the side effects as meaningless as possible.

The spike in the total of network attacks, their severity, and complexity has forced administrators to use tools that rely on anomaly analysis to detect new and unforeseen phenomena, rather than solutions that look for traditional and well-known attacks.

As already specified in section 2.2.1, anomaly-based NIDSs collect data in real-time to look for intrusions and other kinds of anomalies, many of them revealed in statistics extracted from network traffic. Nonetheless, these anomalies can reflect different behaviors, thus shaping a set of rules that can identify every single case is not always a mundane task. Sabahi and Movaghar [17] divide anomaly-based IDSs into the following categories:

- **Statistical methods**, which store variables associated with a user, device, or network behavior looking for deviations from the considered usual presence.
- **Distance-based methods**, which attempt to overcome some of the limitations of the statistical methods by calculating the distance between points. However, it cannot

handle data on substantial dimensional spaces.

- **Rule-based methods** are similar to signature-based IDS, even though Sabahi and Movaghar [17] classify them as anomaly-based since they characterize the usual behavior by a set of pre-defined rules, not the abnormal behavior.
- **Profiling methods**, which look for deviations on different profiles of the network, device, or user behavior built using data mining or heuristic techniques. They can be seen as a mix of rule and model-based approaches.
- **Model-based approaches**, which check for divergences on a model built using predictive data mining techniques such as Deep Neural Networks.

Any of the before mentioned solutions, except model-based approaches, are not easily portable to different contexts and applications since it is not feasible to regularly change a set of rules, profiles, or statistics. Model-based approaches based on learning algorithms are often better performers across different scenarios, since they may be able to continuously learn and adapt to what is considered a “normal” behavior and what it is not.

2.3 Spectral Analysis

Network diagnosis is often solely based on frame or packet statistics at the MAC layer and above. However, it makes it possible to discriminate devices. With proper statistical analysis, one can infer the communication behaviors of each device, which can ultimately raise privacy concerns. Furthermore, with the increased use of encrypted tunnels, even at the data link layer, administrators are forced to resort to physical layer data to perform network monitoring.

Spectral analysis is the process of separating a physical signal into its multiple components at various frequencies [21]. A simple example would be to capture the Wi-Fi signal on a given channel and breaking it down to the multiple frequencies that compose that channel interval (e.g., channel 1 goes from 2401MHz up to 2423MHz; thus the signal power would have to be discriminated in a bin of frequencies in that range).

Nonetheless, in the physical analog world, captured signals are continuous, thus storing such data on a computer is not feasible. In order to properly make them available in a digital context, these signals are sampled, i.e., sliced into discrete time intervals, therefore fitting inside the device limited memory. If the samples are equally spaced, then they are also known as Nyquist samples [22]. The sampling frequency controls the amount of data stored and how well it represents the sampled signal in comparison with the original input signal. Nevertheless, choosing an excessively high frequency could overtax both the memory and the CPU during the sampling operation.

Furthermore, as Shannon’s sampling theorem states, a sinusoid signal can be accurately represented as long as two or more samples are registered under the course of a period [22]. Considering that sampling points are equally spaced (Nyquist samples), the sampling frequency f_s must be at least two times greater than the sinusoid frequency $f_{sinusoid}$. Thus, taking into account that a waveform can be a mixture of multiple sinusoids, in order to accurately

reconstruct the original signal, f_s must be at least two times greater than its maximum frequency f_{max} .

On the other hand, even when selecting an appropriate sampling frequency, thus solving problems associated with time slicing, one must also take into account the input data slicing in amplitude [22]. The amplitude resolution equals the number of bits in the binary output used to represent the original signal. Therefore, increasing the number of bits returns a better representation of the original signal and a smaller quantization error, i.e., the difference between the original continuous value and the obtained sampled representation in every point of the digital data. However, blindly using a higher resolution does not merely correspond to a better representation of the input signal. Increasing the amplitude resolution over the optimal compromise between representation and resource usage only generates a more de-noised version and does not increase the quality of the slicing.

2.3.1 Spectral Anomaly Detection

When it comes to inspecting the electromagnetic spectrum, the detection of anomalies is often a very complex operation. The captured signals can behave so differently that modeling the correct or usual behavior of the spectrum and the underlying network is not always possible.

On the spectrum, there is no clear definition of what an anomaly is. One can consider the presence of activity out of its reserved band an anomalous behavior (e.g., the establishment of a Long Term Evolution (LTE) communication close to the 2.4GHz frequency, which is widely agreed as one of the reserved Wi-Fi bands). One can also consider an anomaly the surfacing of a periodic activity on a channel often characterized by not so periodic behaviors. Another unusual behavior is the sudden increase or decrease of activity on a given band.

There have been a series of works that make use of the electromagnetic spectrum to perform outlier detection, either with privacy and security purposes in mind or just as a simple way to remove noise. Moss *et al.* [23] proposed a Spectral Anomaly Detection (SAD) system optimized for Field-programmable Gate Arrays (FPGAs), by combining a low-latency DFT to obtain the power spectrum, and an efficient detection algorithm based on the construction of bitmaps derived from the time-series signal. Their DFT algorithm aims at obtaining time-series samples from the frequency spectrum at an improved computation complexity of $\mathcal{O}(1)$, when compared to the original $\mathcal{O}(N \log_2 N)$. However, their anomaly detection algorithm was limited since it was only capable of detecting anomalies in regular time-series data.

Sehatbakhsh *et al.* created Syndrome [24], a tool for external monitoring and detection of anomalous behavior in IoT medical devices due to malware injection, through spectral analysis of the electromagnetic energy released by the processor. Their tests revealed a 100% true-positive rate of attacks and zero false positives, with a testing latency under two milliseconds.

SAIFE [25] is yet another SAD system, which resorts to an Adversarial Autoencoder (AAE) for semi-supervised anomaly detection. In sum, the authors fed their AAE model, based on

a Long Short Term Memory (LSTM) encoder and a Convolutional Neural Network (CNN) decoder, with power spectrum vectors. The encoder is trained to confuse and play a *min-max* adversarial game with two external Neural Networks to optimize its overall performance. Besides the authors not extracting features from the power spectrum density vectors, using them directly as inputs of the AAE, the model was only tested with fully synthetic anomalies, whose classification rates were highly dependent on the signal-to-noise ratio between anomalous and “clean” datasets.

An approach that only considered OCC models was proposed by Feng *et al.* [26]. The authors recorded raw IQ data using a Software Defined Radio (SDR) and used them as inputs of the Autoencoders (AEs) and Principal Component Analysis (PCA) models. They were able to detect anomalous behavior by defining a threshold that separated low (“clean”) from higher (anomaly) reconstructing errors. Even though the authors developed models averaging near 100% accuracy, the only anomalous behavior taken into account was additive white Gaussian noise. Samples with anomalies were easily detectable as it corrupted the entire frequency range, and it was purely synthetic.

2.3.2 Wireless Anomaly Detection

Many technologies share the wireless spectrum with very different purposes. IoT devices communications are performed by a panoply of well-established radio technologies, such as Wi-Fi, Bluetooth, Zigbee, and RFID, due to its wireless nature.

Wi-Fi is a radio technology applied in Wireless Local Area Networks (WLANs), which is based on the 802.11 Institute of Electrical and Electronics Engineers (IEEE) standard, whose first version was released in 1997 and it specifies the data link and physical layers protocols used for WLANs [27].

This technology is widely used by many devices, ranging from personal gadgets, such as smartphones and laptops, to IoT devices. Wi-Fi is aimed to work within an unlicensed spectrum, whose bands are widely agreed and free to use by anyone. However, the 2.4GHz band, the current go-to frequency for the majority of Wi-Fi devices, is shared with other communication protocols like Bluetooth and Zigbee, or even home appliances such as microwave ovens. Hence, 802.11 WLANs signal is prone to interference and background noise. Under the 802.11 IEEE standard, over the last 20 years, a series of iterations were proposed and summarized in table 2.1.

As already mentioned, other radio technologies such as Bluetooth and Zigbee were designed for Personal Area Networks (PANs) instead of Local Area Networks (LANs). They both share the 2.4GHz band, whereas the latter also works on the 868 and 916MHz frequencies. Bluetooth has a wide variety of applications such as audio streaming from a mobile device to headphones or speakers, connecting a wireless mouse to a computer, or even sharing data between devices. On the other hand, Zigbee, IEEE 802.15.4 standard, focuses more on low power devices and low bit-rate communications, widely used on WSNs, smart buildings, and home automation. For instance, Philips smart lights line Hue uses Zigbee to communicate between the central bridge and the light bulbs.

Table 2.1: Summary of IEEE 802.11 variants [28], [29].

IEEE 802.11 Variant	Date of approval	Frequency bands	Bandwidth	Maximum data rate	Range
802.11a	July 1999	5GHz	22MHz	54Mbps	35m
802.11b	July 1999	2.4GHz	21MHz	11Mbps	35m
802.11g	June 2003	2.4GHz	23MHz	54Mbps	70m
802.11n	July 1999	2.4 & 5GHz	24 & 40MHz	600Mbps	70m
802.11ac	December 2013	5.8GHz	160MHz	6.93Gbps	35m
802.11ad	December 2012	60GHz	2.16GHz	6.76Gbps	10m
802.11af	February 2014	470-790MHz (Europe) 54-698 MHz (USA)	6,7,8MHz	26.7Mbps	> 1km
802.11ah	May 2017	900MHz	1,2,4,8,16MHz	40Mbps	1km
802.11ax	Expected in 2019	1-7GHz	20,40,80,160MHz	11Gbps	-

RFID has been increasing in popularity for facilitating the management of objects and materials in retail and logistics, as it enables identification from a certain distance. Furthermore, it also provides enough information for detecting anomalies, like using the RSSI regarding the RFID tag as a distance metric and checking if an item is being shoplifted. This last scenario is proposed and tested by Parada *et al.* [30].

All the above radio technologies are commonly victims of similar attack vectors at the physical layer, such that there has been an effort to try to mitigate them by inspecting the power spectrum vector of wireless signals. The RSSI is the signal strength (power) received by a given node on the network, and it has been used to deal with a variety of problems. Besides having a clear correlation with the node distance to the AP or the sniffer, it also indicates whether there is an active communication on the channel. Wang *et al.* [31] proposed an algorithm to detect Sybil attacks on WSNs by computing the relative position of nodes through their RSSI and then checking if a single position, i.e., a given value of RSSI, could be mapped into multiple identities. Through a similar strategy, Yang *et al.* [32] aimed at detecting anomalies in WSN using a multi-layer framework, which collected information from the physical, data link, network, and application layers and combined them to detect intrusions and other outliers in the network. The idea is that, even if it disregards an anomaly at a given layer, the framework would eventually detect it in any of the other layers. Regarding the physical layer, they used the RSSI to perform node location; thus, intruders were detected if a never registered RSSI value has suddenly appeared on the network. However, they promptly affirm that these values are somewhat susceptible to background noise or weather conditions. Tang *et al.* [33] also used RSSI values to detect the physical displacement of nodes on a WSN by sharing that exact data, allowing them to keep track of the location of the remaining nodes. However, these solutions assume that the IDS is working in every node, hence disregarding potential CPU and memory bottlenecks.

Sheth *et al.* developed MOJO [34] to discriminate anomalies at the physical layer without inspecting other layers as they tend to aggregate the effects on anomalous behaviors on the underlying layers. Their objective was to create a flexible, easy to implement and deploy framework, positioning sniffers able to detect and implement countermeasures on near-real-

time performance anywhere on the network. The authors limited the range of anomalies detected to hidden network terminals, capturing effects and signal strength variations at the AP. The first two out of those three were cause of retransmissions at the source due to collisions, and were detected by checking if the transmissions of two nodes associated with the same AP intercepted, according to their timestamps ($T_{\text{start}1} \leq T_{\text{start}2} \leq T_{\text{end}1}$). They evaluated signal strength variations by checking for concurrent changes in the signal strength values on different network sniffers.

Sheng *et al.* [35] led a study focused on MAC spoofing detection in 802.11 networks by creating RSSI profiles in “air monitors” (sniffers). They concluded that these profiles were a mixture of Gaussian distributions, due to the use of multiple different antennas, which had different signal prints.

The majority of recent studies use the RSSI as a real power indication, either correlating it with the distance to network nodes and then finding anomalies using a distance-based metric, or either by only inspecting signal strength variations. This is one of the major shortcomings of these approaches, as one should be aware that anomalies can be masked within the recorded power signal, even if the node’s relative distance does not change. For example, a compromised node can duplicate the data sent to a safe and legit server and also push it to a server belonging to the attacker. The spectrum would most likely only show a single activity period, even though the network was under attack. When there is activity in a channel, the power increases in absolute value for a certain amount of time $t_{\text{transmission}} = t_{\text{legit}}$. If the data is sent to two servers, this time will increase, therefore, $t_{\text{transmission}} = t_{\text{legit}} + t_{\text{non_legit}}$. Nonetheless, increasing the length of activities does not imply variations in the absolute signal strength.

2.3.3 Software Defined Radio (SDR)

To be able to perform anomaly detection on the radio spectrum, one has to use the proper tools. Nowadays, an SDR-enabled device and laptop simplify this job. SDRs are everywhere, from our smartphones to our headphones, as they show to be more flexible, easily tunable, and smaller than the bulkier analog radio.

The use of reconfigurable transceivers is not as recent as one might think, as it goes back to the 1980s [36]. These transceivers were split into a radio Front-End, mainly responsible for the digital sampling of the antenna output through an Analogue-to-Digital Conversion (ADC) process, and a radio Back-End, responsible for the remaining digital signal processing, such as demodulation, filtering, and channel decoding. Later in the 80s, this architecture was famous as it used Application Specific Integrated Circuit (ASIC) chipsets, which were low priced at the time but limited in terms of flexibility. Then, there were considerable investments every time those chipsets had to be upgraded [37]. SDR architectures were then designed to mitigate this issue, as they allowed signal processing to be controlled or even performed via software on FPGAs or General Purpose Processors (GPPs) [36], [38].

Besides that, traditional transceivers usually lock into a given frequency, i.e., they return the amplitude value a of a specific frequency f at a given moment t ($a(f, t)$). Accordingly, if there were a multitude of frequencies of interest $F = f_1, f_2, \dots, f_n$, the radio would have to

be configured to sweep across all the frequencies in F , resulting in probable information loss since the obtained results would have slightly different timestamps. However, SDRs are able to register an array of amplitude values A of a bus of frequencies $F = f_1, f_2, \dots, f_n$ of interest simultaneously ($A(F, t)$).

SDRs are usually composed by the *hardware-defined subsystem* and the *software-defined subsystem*. The first encompasses a multi-frequency antenna, an ADC and a Digital-to-Analogue Conversion (DAC) mechanism with a sampling rate at least two times higher than the frequencies of the signals of interest, and a *software-defined subsystem* with enough processing power to handle the Digital Signal Processing (DSP) of the input signals [37] (figure 2.6).

Figure 2.6: Typical SDR architecture [37].

There are a variety of processing frameworks for SDR such as GNU Radio ⁴, NeXtMidas ⁵, and REDHAWK⁶. GNU Radio is likely the most popular one, and it is part of the GNU Project ⁷. Its basic processing unit is a block, which has a single working purpose and a set of getter/setter attributes. The data flows through these blocks which communicate synchronously via C++ vectors of void pointers or asynchronously via message passing. Besides the graphical tools (GNU Radio Companion), it also provides a Python interface to build SDR systems [39].

There is also a myriad of SDR devices, such as the HackRF One ⁸ and the bladeRF ⁹. The majority are compatible with the mentioned processing frameworks, but they also provide open-source tools which provide an interface to manage them.

⁴<https://www.gnuradio.org/>

⁵<https://nextmidas.techma.com/>

⁶<https://redhawksdr.github.io/>

⁷<https://www.gnu.org/>

⁸<https://greatscottgadgets.com/hackrf/>

⁹<https://www.nuand.com/bladerf-2-0-micro/>

2.4 Time-series Analysis

As mentioned in section 2.3.3, using SDRs facilitates monitoring radio signals in a given environment. However, those signals are captured as discrete samples over time. Given that a big part of the detectable anomalies at the physical layer tend to show a periodical behavior, pure time-series data are not able to accurately represent that.

Furthermore, as the majority of wireless communication protocols are able to use a variety of channels, many times using Orthogonal Frequency-Division Multiplexing (OFDM) to split data transmissions across them, it is essential to perform an analysis in the frequency domain, rather than on the time domain. The two techniques here presented, the DFT and CWT, have slightly different purposes but are both very important.

2.4.1 Discrete Fourier Transform (DFT)

The DFT is a transform extremely useful for power spectrum analysis and filter simulation of sampled time series originated from a continuous input waveform [40]. As any Fourier transform, it represents the fundamental connection between the time and frequency domains, although it does not provide any information about when these frequency components appear over time [21], [41].

On the other hand, computing the DFT of a time-series using the direct method is very taxing on the CPU. The Fast Fourier Transform (FFT) surfaced as a highly efficient computational tool to calculate the DFT of a time-series, thus solving the performance issues regarding the original method.

Equation 2.1 [21] defines the Fourier transform, and it represents the transformation of a time signal $g(t)$ into the frequency spectrum $G(f)$.

$$G(f) = \int_{-\infty}^{\infty} g(t)e^{-2\pi jft} dt \quad (2.1)$$

Inversely, the original time signal can be retrieved using the inverse Fourier transform [21]:

$$g(t) = \int_{-\infty}^{\infty} G(f)e^{2\pi jft} df \quad (2.2)$$

On the other hand, one uses the DFT when dealing with sampled, not continuous signals, which is defined as in equation 2.3 [21], [40], [42], [43]:

$$A_r = \sum_{k=0}^{N-1} (X_k)W^{rk}, \quad r = 0, \dots, N-1 \quad (2.3)$$

$$W = e^{-\frac{2\pi j}{N}}$$

A_r is the r -th coefficient of the DFT and X_k regards the k -th of the N samples of the time-series. Given that $j = \sqrt{-1}$, and X_k usually represents a complex number, A_r also corresponds to a complex number. Similarly to the continuous inverse transform, the sampled

time signal can also be retrieved using the Inverse Discrete Fourier Transform (IDFT): [21], [40], [42], [43]:

$$X_k = \frac{1}{N} \sum_{r=0}^{N-1} (A_r) W^{-rk}, \quad k = 0, \dots, N-1 \quad (2.4)$$

The power spectrum density, or periodogram (figure 2.7), i.e., the power value per frequency, is defined as the squared modulus of the DFT. For a signal discretely sampled every Δt :

$$S(f) = \frac{\Delta t}{N} \left| \sum_{k=0}^{N-1} (X_k) W^{rk} \right|^2 \quad (2.5)$$

Computing directly the DFT and IDFT requires $\mathcal{O}(N^2)$ operations, since for each one of the N outputs X_k , N terms have to be summed. This method is inefficient in performing real-time signal analysis. As already mentioned, the FFT is a highly efficient method to compute the DFT of a sampled time series, doing about $\mathcal{O}(N \log_2 N)$ operations, rather than the $\mathcal{O}(N^2)$ on the direct method. For the typical transform size value of 1024, the speed-up would be over 100 times faster. Besides reducing execution time, it also reduces the round-off error by $\mathcal{O}(\frac{\log_2 N}{N})$ [40]. The FFT follows a *divide-and-conquer* strategy, i.e., it decomposes the overall process into smaller DFT computations and later combine them. This algorithm provides two different strategies: *decimation in time* or *decimation in frequency*. The first regards to, before calculating the DFT, successively decomposing the X_K samples in the time-series into smaller sequences, while the second rearranges the A_r coefficients in the frequency domain into smaller subsets before the DFT computations [43].

Figure 2.7: Periodogram of a 100 Hz sine wave in additive $\mathcal{N}(0, 1)$ noise.

2.4.2 Continuous Wavelet Transform (CWT)

If one considers the DFT as a pure representation of a continuous waveform in the frequency domain, the CWT is an intermediate case. Each wavelet exhibits characteristics regarding both time and frequency domains, making it an excellent tool for time-series analysis.

The CWT, which returns a spectrum of wavelets, is a convolution of an input signal $x(t)$ with a scaled and translated version of a mother wavelet $\psi(\tau, s)$ (figure 2.8) [41], [44], [45]:

$$W_{x;\psi}(\tau, s) = \int_{-\infty}^{\infty} x(t) \psi_{\tau,s}^*(t) dt \quad (2.6)$$

where τ is the translation value (time shift) and s is a scale parameter (dilation), which is tightly coupled with the frequency, so that high scales correspond to low frequencies and low scales to high frequencies. The $*$ denotes the complex conjugate of the function $\psi_{\tau,s}(t)$, which is obtained by scaling the mother wavelet $\psi(t)$ at a time τ and a scale s :

$$\psi_{\tau,s}(t) = \frac{1}{\sqrt{s}} \psi\left(\frac{t - \tau}{s}\right) \quad (2.7)$$

Figure 2.8: Examples of mother wavelets.

By using the properties of the Fourier transform, the CWT can also be represented in the frequency domain:

$$W_x(\tau, s) = \frac{\sqrt{s}}{2\pi} \int_{-\infty}^{\infty} X(\omega) \Psi^*(s\omega) e^{i\omega\tau} d\omega \quad (2.8)$$

where ω corresponds to a radian angular frequency according to $f = \frac{\omega}{2\pi}$.

Using equation 2.8 and, again, the properties of the Fourier transform showcased in section 2.4.1, it is possible to obtain the CWT of a discrete sequence X_k , with k corresponding to the k -th of N samples of the time-series sampled with a period T , and $\tau = mT$ [45]–[47]:

$$W_X(mT, s) = \frac{\sqrt{s}}{N} \sum_{k=0}^{N-1} A_k \Psi^*(s\omega_k) W^{-mk} \quad (2.9)$$

$$\omega_k = \begin{cases} \frac{2\pi k}{NT}, & k = 0, 1, \dots, \frac{N}{2} \\ \frac{2\pi(k-N)}{NT}, & k = \frac{N}{2} + 1, \dots, N-1 \end{cases}$$

The DFT is usually used to obtain the periodogram, i.e., the power spectrum density. Similarly, the scalogram, the time-frequency energy density representation of the CWT (figure 2.9), is defined as the normalized modulus squared of the wavelet transform [45], [48]:

$$E_X(\tau, s) = \frac{|W_X(\tau, s)|^2}{\sum_{\tau' \in T} \sum_{s' \in S} |W_X(\tau', s')|^2} \quad (2.10)$$

Figure 2.9: Computation of scalogram by averaging the result of wavelet transform over time ($\tau' \in T$).

A simple example may help to showcase the utility of this transform. We are considering a traffic light signal that has either the red or green lights turned on. The DFT will be able to show at an infinite precision the frequencies regarding the time intervals where lights were turned on. Therefore, we will know that the light was red or green at a given frequency, but not the instant that occurs. Even if the traffic lights start malfunctioning and both lights get simultaneously turned on, we will know that they both got turned on, but not when.

The CWT will sacrifice some frequency domain precision to earn some temporal context. Considering the obtained wavelet transform W , at the timestamp $t = 10s$, for the frequency component f_{red} , W is large and non-zero. At the same instant, the transform for the component f_{green} is close to zero. Then, for $t = 10s$, the red light will most likely be on and the green off. Still, for $t = 70s$, if $W(f_{\text{red}})$ is close to zero and $W(f_{\text{green}})$ is large, the green light is on, while the red is off. Hence, it is possible to perceive with a considerable amount of accuracy if the traffic light is appropriately working.

2.5 Machine Learning

The term Machine Learning (ML) is often jumbled with Artificial Intelligence (AI). AI is a way of making machines start thinking and carrying out tasks in an intelligent manner [49]. Its main goal is to make machines and the software they are running mimic as much as possible the behavior of the human brain. Nevertheless, ML is nothing more than a subset of AI, which effectively regards how to teach the software to carry out some tasks by providing

it with examples. Furthermore, we now live in a data-driven world, so having the ability to find correlations between vast amounts of data, making it even more popular in other fields, such as statistics and medical analysis.

In 1997, Tom Mitchell [50] summarized ML with the following statement: “A computer program is said to learn from experience E with respect to some task T and some performance measure P , if its performance on T , as measured by P , improves with experience E .”

Unlike what one might think, ML has been around for many years. The first-ever neural network machine was developed in 1951 by Marvin Minsky and Dean Edmonds, and it was known as Stochastic Neural Analog Reinforcement Computer (SNARC) [51]. The machine was built of valves, motors, gears, and wires, and it was designed to look like the neuron connections of the human brain. Its main goal was to perform a simulation where a rat searches for food on a maze.

Despite many techniques being designed in the '90s, the “ML boom” did not happen until the new millennium. The recent advances in hardware design, which dramatically increases performance and reduces power consumption, was the critical factor for this trend. Graphic Processing Units (GPUs) and their ability to efficiently parallelize computations allowed data scientists to go even further than the old ML techniques, thus the popularization of Deep Learning (DL).

Machine Learning systems are often classified based on [52]:

- Whether they are trained under human supervision or not, being classified as either supervised, unsupervised, semisupervised or reinforcement learning techniques;
- Whether they are continually learning on the fly or not, thus are either classified as online or batch learning systems;
- Whether they work by comparing new data to the already known, or by building a predictive model to infer and classify new data.

2.5.1 Machine Learning Use Cases

Consider the scenario where a spam filter is to be developed using traditional programming techniques [52]. Spam filters usually work by storing a database of words that are potentially indicative of a spam email and compare every word of the body, sender’s name, and title, with this word repository.

The typical approach would be to write a set of rules according to the patterns above mentioned, test it and check if it effectively detects spam emails. If necessary, the developer would tinker the set of rules to improve the performance.

This is not an easy task, as the code would likely be very extense. Furthermore, it is tough to update it every time a new spam-related set of words has to be taken into account.

On the other hand, if the spam filter is built using ML techniques, the developer only has to feed the algorithm with a set of emails labeled as spam, and it will be able to detect patterns and words characteristic of these messages. Also, if new words or patterns start to appear in spam emails, the process can be automated so that the algorithm can detect them and update the current detection model.

Besides this typical use case, ML is also useful in complex problems where it is hard to find a solution. It is also helpful in cases where the amount of data is more than what a typical human can single-handed process, since it can provide insights about the correlation between the data, a process also denoted as *data mining*.

2.5.2 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 a given model is trying to decode are hidden in the features of the input data. A feature corresponds to a single numerical series of a multivariate problem.

Using a high number of features is not always beneficial, especially if the number of training examples is relatively small. It seems counter-intuitive that removing features may lead to better classification results, as usually more information is often better.

Even though there is no rule to define the optimal number of features, some algorithms, particularly those who determine non-linear boundaries, are prone to overfitting when using an excessive amount. Furthermore, the obtained data is sometimes artificially mapped to higher dimensions, when, in reality, it belongs to a smaller number of dimensions. Using fewer features may also lead to faster training and more time to test multiple variations, which, in the end, may result in overall better performance [53].

Feature Selection

Selecting the features that will feed a ML 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 important features, and they are often split into **filter**, **wrapper**, and **embedded** categories. **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 [54].

Embedded methods, 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. Figure 2.10 illustrates an example of the return values of these methods.

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 the feature relevance.

One of the most popular Python machine learning frameworks, Scikit-learn ¹⁰, uses the impurity C_j of a node j in a decision tree and its weighted number of samples w_j , where

¹⁰<https://scikit-learn.org/stable/>

Figure 2.10: Feature importance using tree based classifiers.

$left(j)$ and $right(j)$ are j children nodes, to compute the importance of the node ni ¹¹:

$$ni_j = w_j C_j - w_{left(j)} C_{left(j)} - w_{right(j)} C_{right(j)} \quad (2.11)$$

The importance of a given feature i , also known as Gini importance, is given by:

$$fi_i = \frac{\sum_j \xi_{j,i} ni_j}{\sum_j ni_j} \quad (2.12)$$

where $\xi_{j,i}$ denotes the node j splits on feature i , and Ξ denotes the total set of nodes.

Feature Dimensionality Reduction

Even though both feature selection and dimensionality reduction seek to reduce the number of attributes in a dataset, the first removes attributes while the latter does so by creating a new combination of attributes.

The most popular technique is Principal Component Analysis (PCA), whose task is to map a higher dimension set of features to a smaller one which contains the information of the original dataset. Similarly to Variational Autoencoders (VAEs) - both will be further discussed in section 2.5.3 -, the objective is to minimize the reconstruction error of the original dataset based on the dimensionally-reduced representation (figure 2.11).

PCA uses eigenvalue decomposition to compute the eigenvectors of the target covariance matrix, which is defined as [55]:

$$\Sigma = \frac{1}{m} \sum_{i=1}^m n(x^{(i)})(x^{(i)})^T \quad (2.13)$$

¹¹<https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>

Figure 2.11: Dimensionality reduction from the original 3D to a 2D space, using PCA.

The first d_{reduced} eigenvectors are then used as base vectors W for the mapped data, as they point to the data with the highest variance. The dimensionally reduced object z can be obtained from the original d -dimension x by:

$$z = W^T x \quad (2.14)$$

The original d -dimension vector can be computed with the following approximation:

$$x_{\text{approx}} = Wz \quad (2.15)$$

Therefore, the reconstruction error of an object x can be obtained as the squared distance from the original and the dimensionally reduced vectors [55]:

$$\begin{aligned} d_{PCA}(x) &= \|x - (W(W^T W)^{-1} W^T)x\|^2 \\ &= \|x - (W W^T)x\|^2 \end{aligned} \quad (2.16)$$

2.5.3 One-class Classification

Anomalies, or outliers, are objects or samples that differ from the majority of the others. According to Hawkins [56], “an outlier is an observation, which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism.”

An anomaly can be individual instances that show to be very different from the rest, as exemplified in figure 2.12 on the left, but it can also be a deviation from the normal behavior on a particular context, as illustrated on the right of the same figure.

Anomaly detection can be performed using either one of three approaches:

- **Supervised learning**, when both normal and anomalous data are available and respectively labeled. This methodology usually performs well; however, its detection range is limited to the type of anomalies fed to the algorithm.
- **Unsupervised learning**, when none of the data is labeled, but one assumes that it contains both normal and anomalous data.

Figure 2.12: Examples of point (left) and context (right) anomalies.

- **Semi-supervised learning**, when only a few samples are labeled, but from both regular and outlier datasets.

Even though many studies have been performed using these techniques, they take into account that there is always anomalous data available for training the model. There is an approach, considered a blend of both supervised and unsupervised learning, that tries to mitigate this issue. **Novelty detection**, or **One-Class Classification (OCC)**, checks for novel or outlier samples in the test set, i.e., unseen phenomena, with respect to the distribution of the training data. OCC can be seen as supervised learning since clean data in the training set is labeled; yet, the algorithm is not aware of what an outlier looks like, thus, it can also be considered unsupervised learning.

Novelty detection has been increasing in popularity in a variety of fields, such as medical diagnostic, failure detection in complex industrial systems, fraud detection, or video surveillance. The difficulty in modeling what an anomalous pattern is *a priori* or even the lack of samples in that regard makes **OCC** systems ideal for these cases [57]. For instance, there is a need for a model for fault detection regarding some industrial machinery. If we took a multi-class approach, samples of both standard and faulty states would have to be available for training. Still, that could mean that, in order to obtain negative class data, a faulty machine would have to be available, or ultimately a functional one destroyed. That is not a feasible approach, so the model would have to be constructed using positive labeled samples only.

There are four main approaches to OCC problems: density/probabilistic-based, distance-based, reconstruction/model-based and domain-based methods.

Density-based Methods

When a significant amount of data is available, the most straightforward 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

or Poisson distributions, or even a mixture of them to create a more flexible model, like a Gaussian Mixture Model (GMM) [55], [57].

The most common and often simple model is the **Gaussian distribution**, which is represented in figure 2.13. The probability of a univariate Gaussian distribution of an object x is denoted as:

$$p_{\mathcal{N}}(x, \mu, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{\frac{1}{2}}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad (2.17)$$

For a d dimension distribution, where M is the mean and Σ the covariance matrix, it can be represented as [55]:

$$\begin{aligned} \left(\frac{x-\mu}{\sigma}\right)^2 &= (x-\mu)(\sigma^2)^{-1}(x-\mu) \\ &\Rightarrow (x-M)^T \Sigma^{-1} (x-M) \\ \Rightarrow p_{\mathcal{N}}(x, M, \Sigma) &= \frac{1}{(2\pi)^{d/2} |\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(x-M)^T \Sigma^{-1} (x-M)} \end{aligned} \quad (2.18)$$

Figure 2.13: Threshold on a 1-dimensional Gaussian distribution [55].

To create a model which is more versatile and flexible model, one can use a **mixture of Gaussians (GMM)** (figure 2.14), i.e., a linear combination of M probability density functions $p_{\mathcal{N}}$ [55]:

$$p_{\text{GMM}}(x) = \sum_i^M \alpha_i p_{\mathcal{N}}(x, \mu_i, \sigma_i^2) \quad (2.19)$$

where α_i denotes the mixing weights, and μ_i, σ_i^2 the mean and standard deviation of the i -th normal distribution. Lu and Traore [58] developed a real-time anomaly detection framework by converting IP packets data into a four-dimensional numerical feature space and using an algorithm based on GMM. Yang *et al.* [59] conducted a study to assert how GMMs perform in real-life scenarios and datasets, concluding its validity.

Figure 2.14: Edge definition of a GMM [60].

A non-parametric approach, such as the **Kernel Density Estimation (KDE)** (or Parzen windows) [61], is also popular. Considering $D = x_1, x_2, \dots, x_N$, a d dimensional set of N examples, and ϕ , the kernel function, whose window size is denoted as h , the Parzen window density estimation $p_{\text{Parzen}}(x)$ is defined as:

$$p_{\text{Parzen}}(x) = \frac{1}{N} \sum_{i=1}^N \phi(x - x_i, h) \quad (2.20)$$

The rectangular and the Gaussian window functions are generally the most commonly used. Parzen density estimation with the Gaussian function is given by [55], [62]:

$$p(x) = \frac{1}{N} \sum_i p_{\mathcal{N}}(x, x_i, \Sigma, h) \quad (2.21)$$

$$p_{\mathcal{N}}(x, \mu, \Sigma, h) = \frac{1}{(2\pi)^{d/2} h^d |\Sigma|^{\frac{1}{2}}} e^{-\frac{(x-\mu)^T \Sigma^{-1} (x-\mu)}{2h^2}}$$

Opposite to the GMM whose model is defined by a series of parameters, the Parzen density model is non-parametric. Therefore, the training cost is almost zero since the data model is weak, and the training set gives all the density representativity. On the other hand, the testing cost is expensive as all the training objects have to be stored, and all distances to those objects have to be calculated and sorted.

Cao *et al.* [63] proposed an anomaly-based NIDS that modeled the network system using KDE combined with a Self-Organizing Map (SOM), whose task was to create a well-defined cluster to reduce the computational cost of the KDE. They tested their combined model in the Ohio University's network detective service and concluded that it could detect anomalous behavior effectively, and more efficiently when compared to a single Parzen estimator with a higher number of kernels.

Distance-based Methods

Distance-based methods usually include clustering or nearest-neighbor approaches, which essentially compute distances between data points, i.e., the similarity measure between them.

k-Nearest Neighbours (kNN) is one of the examples of nearest-neighbor methods. It often uses Euclidean or Mahalanobis distance to calculate the similarity between samples. One of the advantages of this approach is that it does not require building any model since the training samples are stored to compute the distances to testing samples later. However, it requires more computational resources than the majority of the approaches during the test phase, since, just like the Parzen density model, all the training samples have to be stored and the distance between points has to be computed (figure 2.15).

Figure 2.15: Classification of a new example using kNN on a 2-class problem.

The one-class kNN algorithm is a slight modification of the multi-class approach [64]. Consider a test example z and the distance $d(z, y)$ to its nearest neighbour $y = \text{NN}(z)$. Since the default value of δ is often 1, a new sample belongs to the normal class if:

$$\frac{d(z, y)}{d(y, \text{NN}(y))} < \delta \quad (2.22)$$

The one-class kNN is different from the multi-class approach since it averages the distance of the k -nearest neighbors.

Even though it can be considered a density-based algorithm, **Local Outlier Factor (LOF)** [65] computes the local density deviation of a given sample concerning its neighbors rather than the whole data distribution, and it considers that an outlier has a substantially lower density than clean data. It is classified as a distance-based approach since the density values are calculated 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 $N_k(o) = \{o' | o' \in D, \text{dist}(o, o') \leq$

$\text{dist}_k(o)\}$, the local reachability density ψ of o is given by [65]:

$$\psi_k(o) = \frac{||N_k(o)||}{\sum_{o' \in N_k(o)} \text{reachdist}_k(o' \leftarrow o)} \quad (2.23)$$

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

The LOF can then be computed as the average ratio of local reachability of o and its k -nearest neighbors [65]:

$$\text{LOF}_k(o) = \frac{\sum_{o' \in N_k(o)} \frac{\psi_k(o')}{\psi_k(o)}}{||N_k(o)||} \quad (2.24)$$

This means that, the lower the local reachability density of o and the higher of its k -nearest neighbors, the higher the LOF; thus, a higher probability of o being an outlier. Since this algorithm ranks points by its neighbors, it is possible to miss outliers whose ψ is similar to other neighbor outliers ψ .

Prakobphol and Zhan [66] proposed a NIDS that used, among other algorithms, a LOF approach to detect unauthorized access, DoS, and probing attacks, even if the training set only contained anomaly-free data. Their tests showed that it is able to detect almost every outlier behavior with low false-positive rates, with only 20 neighbors.

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. This novelty score, also known as reconstruction error, is the distance or similarity between the test vector and the output of the system [55], [67].

Many algorithms fit this category, which take different approaches regarding the data clustering and its distribution on the subspaces. To name a few, PCA and autoencoders fit this category.

PCA is often used for data dimensionality reduction, as it finds the optimal orthogonal subspace which captures the features with the highest variance in data. It can also be used as a reconstruction technique, thus an outlier detection tool.

Autoencoders (AEs) [68] (figure 2.16) use a similar approach to PCA since they are trained with backpropagation to reconstruct the input data at the output layer. A test sample is classified as the target class by computing the difference between the input and the output, and comparing it with a given threshold. If it is above the threshold, it is considered an outlier or anomaly. This was the approach taken by Aygun and Yavuz [69], where they used deep autoencoders to perform network intrusion on the *NSL-KDD*¹² dataset and compared them with state-of-the-art solutions. They concluded that their model outperformed all the non-hybrid existent approaches.

¹²<https://www.unb.ca/cic/datasets/ns1.html>

Figure 2.16: Basic AE architecture.

Each neuron of a neural networks calculates a “weighted sum” of its inputs, and adds a bias according to an activation function. There many activation functions, such as the Hyperbolic Tangent and Sigmoid functions, denoted in equation 2.25. These are great to use in classifiers, and they bound their output to a fixed range - $[-1; 1]$ and $[0; 1]$, respectively - so the activations will not explode.

$$\begin{aligned}\text{sigmoid}(x) &= \frac{1}{1 + e^{-x}} \\ \tanh(x) &= \frac{2}{1 + e^{-2x}} - 1\end{aligned}\tag{2.25}$$

Rectified Linear Unit (ReLU), on the other hand, is a linear function when $x > 0$, which means that its output ranges from $[0; +\infty[$, according to equation 2.26. ReLU enables sparsity on the activations of the neurons, as some will yield the value 0. This is particularly useful on networks with a vast number of neurons per layer, reducing the computational cost. However, gradients of negative values of x will stop responding to variations, and they will possibly make some parts of the neural network passive. Leaky ReLU was developed as a way to minimize this problem by changing the negative portion of the ReLU activation function into a small derivative linear function, such as $f(x) = 0.01x$, when $x < 0$.

$$\begin{aligned}\text{ReLU}(x) &= \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases} \\ \text{LeakyReLU}(x) &= \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases}\end{aligned}\tag{2.26}$$

The goal of neural networks is minimize a given metric, or error. For reconstruction-like approaches, one typically uses the Mean Squared Error (MSE):

$$\text{MSE}(y, \hat{y}) = \sum_{i=1}^n (y_i - \hat{y}_i)^2\tag{2.27}$$

Although having an autoencoder-like architecture, probabilistic generative models such as **Variational Autoencoders (VAEs)** [70] define a neural network that, by running a set of latent variables z through a decoder applying a process of data generation $g(z)$, are able

to reproduce the input data x . As observed in figure 2.17, $q_\phi(z|x)$ denotes the approximate posterior and $p_\theta(z)$ the prior distribution of the latent variable z , in which $p_\theta(x|z)$ is the likelihood of the data x . In sum, the VAE models the parameters of the distribution of the latent variables rather than the actual values, by minimizing the error between the prior and posterior distributions through the Kullback-Leibler (KL) divergence and the reconstruction of x after being exposed to the posterior distribution and its likelihood value (equation 2.28, figure 2.17). For example, the function defining the encoder neural network $f(x, \phi)$ outputs the parameters of the approximate posterior distribution $q_\phi(z|x)$, not the latent variables z value, which are only obtained by sampling from $q(z; f(x, \phi))$.

Figure 2.17: Simple VAE architecture.

$$E = -D_{KL}(q_\phi(z|x^{(i)})||p_\theta(z)) + E_{q_\phi(z|x^{(i)})}[\log p_\theta(x|z)] \quad (2.28)$$

An and Cho [70] tested the VAE against other encoder-like architectures such as vanilla autoencoders, and PCA, on the *NSL-KDD*¹³ dataset, and showed that the first outperformed all the others.

Both these approaches tend to overfit the training data since they do not have any idea of the portrays of an outlier. **Generative Adversarial Networks (GANs)** try to mitigate this problem and to improve the performance of the binary classifier, also know as discriminator, by using a different model, the generator, for generating fake negative-class samples, as illustrated in figure 2.18. The goal is that, while training, both models can “challenge” each other and improve their performance; the generator by creating samples that are even more similar to the original data, and the discriminator by being able to differentiate samples that are even more alike [71], [72]. GANs often use a different error metric, as their goal is minimize the classification difference between two classes. Binary cross-entropy is the most typically used metric:

$$H(y, \hat{y}) = -\frac{1}{N} \sum_{i=1}^N y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i) \quad (2.29)$$

¹³<https://www.unb.ca/cic/datasets/ns1.html>

Figure 2.18: Simplified version of a GAN architecture.

The majority of model-based approaches are able to profile clean instances to later identify anomalies as samples that are distinct from them. In contrast, **Isolation Forests (IFs)** isolate anomalous instances rather than building a model solely based on normal data [73]. In order to detect outliers, the algorithm assumes that a tree structure can isolate every instance. Since anomalies are susceptible to the isolation, they are usually in the root of the tree; therefore, if fewer steps are needed to isolate a sample, the chances of it being an anomaly increase. Figure 2.19 shows the tree structure used to isolate a normal and an outlier.

Figure 2.19: On the left, the tree structure steps to isolate a normal sample x_i . On the right, the less complex structure to isolate an outlier x_0 [73].

The anomaly score of an IF is given by:

$$s(x, n) = 2^{-\frac{E(h(x))}{c(n)}} \quad (2.30)$$

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, and n is the number of external nodes. Scores s closer to 1 indicate possible anomalies, but a threshold must be defined to separate them from clean data.

Calheiros *et al.* [74] explored the feasibility of using IFs in large-scale cloud datacenter anomaly detection since other algorithms were not sufficiently efficient due to the vast amount

of data. The authors resorted to IFs due to their low complexity for training and anomaly identification.

Domain-based Methods

These methods try to create a boundary surrounding the training data, and they are usually agnostic to its density [57]. The most popular approaches are One-Class Support Vector Machines (OC-SVMs) and Support Vector Data Descriptors (SVDDs).

Multi-class Support Vector Machines (SVMs) define a boundary between classes that maximizes the margin between the two classes. However, **OC-SVMs** are only fed with one class, and in theory, it would not be possible to perform this separation through the conventional method, but only by finding a hyperplane of maximal distance from the origin. One common approach to improve the usual overfitting is by establishing *a priori* the percentage of positive data allowed to be outside the boundary of clean samples, making the model more tolerant to outliers [75].

A classical multi-class SVM is able to separate the points of two classes by projecting the data to a higher-dimensional space using a non-linear function ϕ . In that context, data points can then be separated by a vertical plane (or line), which is defined by $w^T x + b = 0$. Given that ξ_i variables are used as a way of introducing soft noise to prevent overfitting and that the constant $C > 0$ controls the penalty for misclassified training examples, the objective of the SVM is to minimize:

$$\begin{aligned} \min_{w,b,\xi_i} \frac{\|w\|^2}{2} + C \sum_{i=1}^n \xi_i \\ \text{subject to:} \\ y_i(w^T \phi(x_i) + b) \geq 1 - \xi_i \quad i = 1, \dots, n \\ \xi_i \geq 0 \quad i = 1, \dots, n \end{aligned} \tag{2.31}$$

If this problem is solved using Lagrange multipliers, commonly used for finding the local maxima and minima of a function subject to equality constraints, the decision function for every point x is denoted as:

$$f(x) = \text{sgn}\left(\sum_{i=1}^n \alpha_i y_i K(x, x_i) + b\right) \tag{2.32}$$

where α_i corresponds to the Lagrange multipliers and $K(x, x_i)$ to the kernel function (often the Gaussian Radial Base Function (GRBF)).

According to Schölkopf *et al.* [75], an **OC-SVM** uses a feature map $\phi \implies X \rightarrow F$ 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 to the origin of the obtained hyperplane characterized by w and ρ . The minimization function is now defined as:

$$\begin{aligned} \min_{w, \xi_i, \rho} & \frac{\|w\|^2}{2} + \frac{1}{vN} \sum_{i=1}^N \xi_i - \rho \\ \text{subject to:} & \\ (w \cdot \phi(x_i)) & \geq \rho - \xi_i \quad i = 1, \dots, N \\ \xi_i & \geq 0 \quad i = 1, \dots, N \end{aligned} \tag{2.33}$$

where ξ_i are slack variables and the coefficients α_i are found as the solution of:

$$\begin{aligned} \min_{\alpha} & \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j K(x_i, x_j) \\ \text{subject to:} & \\ 0 \leq \alpha_i & \leq \frac{1}{vN} \quad \sum_i \alpha_i = 1 \end{aligned} \tag{2.34}$$

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

$$f(x) = \text{sgn}\left(\sum_{i=1}^N \alpha_i K(x, x_i) - \rho\right) \tag{2.35}$$

On the other hand, **SVDDs** were proposed by Tax and Duin [76], and work by creating a boundary corresponding to the minimum hypersphere that encloses the majority of the clean data. The model is optimized by generating artificial unlabeled data that uniformly surrounds the positive class. This hypersphere is defined by a center a and a radius $R > 0$, and the main objective of the SVDD is to minimize the volume R^2 . The minimization problem is given by:

$$\begin{aligned} \min_{R, a, \xi_i} & R^2 + C \sum_{i=1}^n \xi_i \\ \text{subject to:} & \\ \|x_i - a\|^2 & \leq R^2 + \xi_i \quad i = 1, \dots, n \\ \xi_i & \geq 0 \quad i = 1, \dots, n \end{aligned} \tag{2.36}$$

To verify if a point z belongs to the target class, the distance, computed using the GRBF, to the center has to be smaller than or equal than the radius:

$$\|z - x\|^2 = \sum_{i=1}^n \alpha_i e^{\frac{-\|z - x_i\|^2}{\sigma^2}} \geq -\frac{R^2}{2} + C_R \tag{2.37}$$

OC-SVMs and SVDDs are similar approaches, especially when using Gaussian kernels in OC-SVM models since it usually results in the creation of the same hypersphere boundary as

the SVDD. In figure 2.20, one can see the typical boundary created by the OC-SVM when using linear kernels. When using non-linear approaches, the boundary is similar to the one created by the SVDD, on the right. Regarding network security, there are multiple proposals of NIDS using OC-SVMs and SVDDs [77], [78].

Figure 2.20: Boundary representation of an OC-SVM (left) and a SVDD (right) [79].

2.5.4 Ground-truth Evaluation

In order to accurately evaluate the performance of a classifier, it is necessary a perfect ground-truth, i.e., training and testing samples must be correctly labeled. Otherwise, it would impact the accuracy of the considered models. However, obtaining perfect ground truth is very challenging, especially when considering the evolution and the heterogeneity of network technologies. In not so past years, network applications could be easily distinguished by their use of a well-known TCP/User Datagram Protocol (UDP) port. Nowadays, due to the use of dynamic port applications, this is no longer a suitable strategy.

There has been a series of approaches to generate ground-truth for anomaly detection, ranging from manual inspection of network traces, the blend of real attack-free traffic with emulated attack traffic or even fully emulated traces [80]. Manual inspection of traffic consumes too much time and requires an analysis of different factors to label traces correctly. This is often not easy taking into account the unexpected nature of an attack. Using emulated traffic usually provides a more reliable ground-truth, even though it does not always accurately represent the behavior of real-life traffic. Furthermore, it is difficult to verify the correctness of the simulated data when compared to real traces [81]. On the other hand, one can also argue that it is very challenging to obtain real attack-free traces; thus using simulation can be the way to go in these situations.

Having a perfect ground-truth is not always a must, or even possible to obtain. Consequently, the model classification results must be further analyzed. Consider an OCC machine learning model, whose inputs correspond to a statistical analysis performed over sets of t -seconds observation windows of attack-free network traces. In order to evaluate the performance of the model when it comes to anomaly detection, an attack vector was simulated.

Figure 2.21: Timeline example of an anomaly showing up every 4 windows.

It should appear every four windows, i.e., it happens approximately every $4t$ seconds. Since the attack only showed up once every four windows, three of them would be labeled as an attack, even though they represented the normal behavior of the network. This situation led to an imperfect ground-truth, since it would be very impractical and time-consuming to label each observation window manually. If the model was perfectly accurate, it would report an anomaly once every four windows, even if the anomaly is present at all times. Hence, there is the need for further analysis upon the output results. This example is illustrated in figure 2.21.

The use of an imperfect ground-truth on the performance evaluation of a model already trained is not as critical as if it was used to train the model. The latter can lead to significant estimation errors. Oliveira *et al.* [80] proposed a latent class model that combined the results of different classifiers in order to reduce the effects of using an imperfect ground-truth. Their model was based on a previous work that assumed that, while a given categorical variable is not observable at a given moment, its effects were showcased by other categorical variables, and it was able to perform very well under moderate levels of ground-truth imperfection.

2.6 Summary

In this chapter, one comprehends how the increased use of IoT devices in critical environments is seen as a liability to the network. Given their low-power design, administrators use monitoring strategies to detect the presence of outliers and many known attacks. The majority of these approaches use only data from layer 2 or above, possibly compromising the privacy of network nodes, or limiting their range of detection if the communication channel is encrypted. On the other hand, detection systems that deal exclusively with layer 1 data often use it as a distance metric, which only fires an alert when there is a variation in the node's relative distance. Those taking a step forward, and feeding their models with raw IQ data or power density vectors so that they can infer the behavior of the network, often consider purely synthetic anomalous behaviors, facilitating their detection.

This dissertation proposes privacy-focused mechanisms to detect the presence of outliers by using that physical-layer data as a behavioral metric, not as a relative-distance indicator, and by extracting relevant features from the spectrum behavior, rather than feeding them blindly to models.

The techniques presented in section 2.4 will be used for converting registered IQ values to RSSI indicators, and to perform a periodic-based analysis over silent and activity periods on the network.

The following chapter describes privacy-focused mechanisms to identify the presence of outliers at the physical layer by extracting features from the behavioral patterns of the network. A few feature engineering techniques presented in section 2.5 will be employed to filter out non-relevant inputs; the most relevant data will feed the majority of the mentioned OCC models, used to discern “clean” from outliers behavior.

Physical Layer Anomaly Detection Mechanisms

As stated throughout chapter 2, with the advent of IoT and wireless mesh networks, security risks inherent to these types of networks have grown in number, and severity, especially when attackers use network resources in an illicit manner, or perform data exfiltration. Most of the approaches currently available for anomaly detection perform frame and packet inspection, which may inadvertently reveal the private behavioral patterns of its users. Moreover, in situations where the communication channel is encrypted, these strategies would be ineffective. Physical layer analysis makes it more challenging 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 MAC or IP addresses.

This section 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 OCC models.

3.1 System Overview

In order to rightly present it, we split the overall system into a series of steps. All the showcased pipeline in figure 3.1 is further detailed below. In sum, the sniffer registers IQ values to be later converted to RSSI power indicators through an FFT processing, which are then turned into a binary activity timeline. From it, we extract a series of statistical and frequency-time-related features. Clean-labeled data should then be used to train multiple one-class classifiers, to be later used as anomaly detectors.

Figure 3.1: Overall system overview.

3.2 RSSI Data Collection

As mentioned in chapter 2, 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 was essential owing to the ability to monitor an environment without the need to authenticate a sniffer client in the network, assuring that data exchanges between clients stayed private as they cannot be individually identified. The RSSI is the amount of power in a received radio signal [82]. Therefore, it is not bound to any specific Radio Frequency (RF) technology, and one can use it with many others, such as Bluetooth ¹ and Zigbee ².

In order to capture RF data, a probe with a radio capable of recording its environment is needed, i.e., a sniffer. That radio must be able to sweep an array of frequencies $F = f_1, f_2, \dots, f_n$ in the same instant t , whose components may correspond to, as an example, a Wi-Fi channel, and return the corresponding amplitude values $A(F, t)$ for those very same frequencies of interest. Traditional transceivers are not capable of such task; hence, the use of SDRs is perfect for these situations.

The SDR must then be configured 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 convert the amplitude values belonging to the time-domain to the frequency-domain to compute power indicators (RSSI) later on, one must use the DFT. Performing FFT computations over the full signal is not a wise choice, as it can be extremely taxing on the CPU, besides only retrieving the results after stopping the capture. So, an FFT window size (ζ) has to be defined, indicating how many consecutive samples the DFT will effectively use to convert the time series to the frequency domain. Figure 3.2 represents a 1024-point logarithmic power FFT over the recorded SDR IQ components of the signal.

Moreover, the majority of SDRs are powered by an amplifier, whose configuration must

¹<https://www.bluetooth.com/>

²<https://www.zigbee.org/>

Figure 3.2: RSSI spectrum (right) obtained by computing an 1024-point logarithmic power FFT over the IQ signal outputted by an SDR (left).

be carefully done. It is essential to assure that it is easy to distinguish the ground noise from network activity. Determining the optimal parameters is usually accomplished by inspecting the outputted signal, while testing multiple combinations of the settings of the amplifier. It is nearly impossible to define a set that works across a variety of different radios, respective amplifiers, and that accurately capture signals in both noisy and clean environments. An overall well-configured amplifier facilitates the choice of the silence/activity threshold, hence an overall better system performance.

Figure 3.3: Extraction of multiple time-series data from the FFT output for each of the selected frequencies of interest.

The FFT will output ζ points; however, not all of these may be relevant to the problem. For instance, when monitoring an entire Wi-Fi channel, one can average all ζ points to obtain a single RSSI value. If one wants to monitor multiple individual frequencies, one does so by selecting a subset of the FFT output, or even by mixing averaged subsets of frequencies with individual points; thus, creating multiple time-series. The goal is to extract data with

relevance to machine learning models from these time-series. This procedure is illustrated in figure 3.3.

3.3 Data Preprocessing

In the ML algorithm standpoint, the data outputted by the sniffer or after the FFT processing does not have any representative meaning. Therefore, it has to be transformed into relevant features. Given that layer 1 data are usually subject to variations due to real-world distance changes, the addition of physical obstacles between nodes and overall environment background noise (devices like microwave ovens and televisions may be a source of potential 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 favored, as they reveal the behavioral patterns of normal and anomalous traffic. For instance, the data extracted from an IoT network with sensors synchronously uploading data every minute may reveal that 90% of silent periods persist almost 60 seconds. If a node that got tampered transmitted data every 30 seconds, the mentioned metric would be nearly 30 seconds, which can be seen as an anomaly. In order to differentiate activity and silence periods, one must define a RSSI threshold, as it will separate ground noise from network activity, as regarded in figure 3.4. That threshold is given by equation 3.1, where κ corresponds to the mean of the RSSI time-series, and $\varkappa$ to the mean of the activity spikes (recall that RSSI values are negative, and that the average of these spikes will be approximately the mean RSSI of an activity). The threshold is obtained by averaging $\varkappa$ and κ .

Figure 3.4: Extraction of activity periods (on the right) from the original RSSI values (on the left) with a threshold of -80.0dBm .

$$\begin{aligned}
\kappa &= \frac{1}{m} \sum_{i=0}^{m-1} \text{RSSI}_i \\
\kappa &= \frac{1}{n} \sum_{\text{RSSI}_i > 0.99 * \kappa} \text{RSSI}_i \\
thr &= \frac{(\kappa + \kappa)}{2}
\end{aligned} \tag{3.1}$$

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 that threshold. During a transmission, there are small silence gaps between activity periods, due to driver and application latency, wait times while accessing the network, or the nature of wireless communications; therefore, the activity periods can be grouped into one more extended activity period. Likewise, there is activity in the network that can be considered unaccounted, such as non-requested Address Resolution Protocol (ARP) and Multicast DNS (mDNS) packets or discovery protocols like Simple Service Discovery Protocol (SSDP). So, activity periods whose duration fits under a given threshold can be discarded. This *non-relevant activity filter* may also delete TCP Keep Alive and other session packets, which essentially describe the flow of data in the network. Taking this into consideration, it is proposed that, before settling on any filtering strategy, both methods (with or without removal of small activity periods) should be tested (figure 3.5).

Figure 3.5: Post processing of activity periods, joining activity peaks and removing short isolated activities (on the right). The original activity signal is represented on the left.

3.4 Feature Extraction

Even though the activity timelines show the behavior of the network, they do not highlight changes and variations in such a way that the machine learning algorithm is able to model those patterns.

With this in mind, the first step is to split the samples of the activity timelines 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 θ , where a window n would start on sample i and end on sample $i + \theta$, and window $n + 1$ would start on sample $i + \theta + 1$. However, this “hard” separation has some setbacks, as different samples may occasionally show very different patterns, making it harder for the algorithm to create an overall model.

In order to increase the throughput and the similarity between consecutive windows, resorting to a “sliding” strategy can be beneficial. 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 (figure 3.6). In sum, if we consider a slide value $\Delta < \theta$, window n would start on sample i and end on sample $i + \theta$, whereas window $n + 1$ would start on sample $i + \Delta$ that, in reality, $i + \Delta < i + \theta$.

Figure 3.6: Comparison of how the timeline samples are distributed across data windows. On the top, an example of “hard” separation, while on the bottom, the “sliding” windows approach is showcased.

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 (figure 3.7). Nonetheless, the dataset must be split according to the most extensive window size and the defined window slide. Then, for every sub-window ending on the same data sample, the statistical measurements are computed. For training purposes, only one window size is considered, given that all these features are grouped, even though every sub-window provides insights over different periods. For example, according to the Nyquist samples theorem [22], on a 30-second window, one will identify a tampered node sending data every 10 seconds by checking that its mean silent periods trend to 10 seconds. However, when the network behavior is also periodic, in this particular case, 60 seconds, the majority of silent periods on 30 seconds windows will be close to 30 seconds, thus making it easy to detect outlier behavior.

As showcased in figure 3.7, mean, median, standard deviation, 75th, 90th, 95th and

99th percentiles, minimum, and maximum values regarding activity and silent periods are computed. These features from the different sub-windows are then joined and treated as a single window. Additionally, a scalogram over the activity values on the biggest sub-window size used is obtained so that pseudo-periodic components can be used as features. As explained in section 2.4.2, the scalogram is computed as the normalized squared modulus of the CWT, using a Morlet [83] mother wavelet on a set of scales related to the sub-window size. It then outputs values of energy 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. For instance, a network trace of an outlier generating data exchanges every 60 seconds will have a high energy value on the scale corresponding to 60 seconds.

Figure 3.7: Diagram of features extracted for every sub-window size. Even though statistical features are extracted in all of them, the local maxima energy and scales of the CWT are only computed for the biggest defined sub-window size.

It is common practice to set the difference between scales as a fixed interval. However, the resolution of the scalogram is highly coupled with the number of considered scales. Consider a set of scales in the $[1, 600]$ seconds interval. Extracting 128 values will return a plentiful accurate representation of the energy spectrum, and all the relevant local maxima will be easily identifiable, but this process would take an excessive amount of time. On the other hand, if we set it to only 16 scales, it will result in a low accuracy representation. Therefore, there has to be a compromise between performance and resolution. Also, using logarithmic spaced values will often produce better results when covering a substantial range of scales.

The most relevant information from the scalogram concerns the local maxima scales and its energy values (figure 3.8). The optimal number of extracted local maxima is another compromise, between the amount of information and the performance of the learning algorithm. Moreover, using a very high number of local maxima 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.

Figure 3.8: Scalogram performed over 64 scales distributed across 1 and 600 seconds. Only the top five local maxima values and respective scales are used as features for the ML algorithm, even though all of them are clearly identified.

3.5 Feature Dimensionality

As mentioned in section 2.5.2, it is often a good practice to perform some sort of dimensionality reduction in the feature set used to train the classifiers, especially if the number of parameters is too high compared to the number of samples, or when using classical classifiers that determine non-linear boundaries, which can result in overfitted models. Additionally, the learning process will be significantly faster, with a smaller number of features. According to the selected frequencies of interest, it is essential to notice that any feature engineering has to be performed on the multiple timelines exclusively.

As an example, we consider that there are n sub-windows in each data window, and that m features per sub-window are extracted (section 3.4). This totals $n \times m$ features on each window. Most likely, many of them are correlated or do not have enough statistical relevance even to be considered. In this situation, it can be beneficial to use only a subset $k < m$ of those features to train the classifier. To select the k most relevant features, one can resort to various feature selection embedded models [84], [85]. The goal is to minimize k since the optimal scenario is to find the minimum set of features that can adequately model a clean network behavior.

In section 2.5.2, we presented two strategies to reduce the total amount of inputs used to train the models. One was Principal Component Analysis (PCA), which maps a higher dimension set of features to a smaller one without losing the information of the original dataset. The second strategy is feature selection, i.e., using only those given by a series of models that output the relevance of each input feature, to train the classifiers. Both strategies should be tested to assess which one performs better.

Regarding feature selection mechanisms, the idea behind using multiple models is to ensemble different perspectives into a reliable global selection methodology. Furthermore, since the goal is to develop an anomaly detection system, anomaly samples should not be used

during training in order for the classifier to solely learn to model clean data and differentiate it from outlier data, rather than using samples with contamination in the learning process to help it distinguish normal behavior from outliers. However, the datasets must have negative and positive class examples, so that feature selection models can check which features will potentially be the most useful in telling them apart. To avoid having the models biased towards a specific type of anomaly, one should use anomalous samples of different types, even though it is not the optimal approach. Nonetheless, it is feasible, as one cannot obtain examples of every possibly imaginable anomaly.

Figure 3.9: Classification system used across different anomaly datasets.

The majority of embedded feature selection models return a normalized importance value. Still, these values do not translate very well across different datasets (e.g., a given feature may be the most important one in different datasets but with different relevance values); thus, we propose using a classification system. This way, the most relevant feature gets 1 point, and the least important receives $n \times m$ points (figure 3.9). In the end, the k features with fewer points are selected as the most relevant.

To determine k , i.e., the minimum number of features to obtain a good classification performance, yet with reasonably fast learning times, different models with j features can be trained, belonging in the $1 < j < m$ range, which correspond to the j most important features pointed out by the already mentioned feature selection ensemble model, and verifying which j optimizes its classification *F1-Score*.

On the other hand, neural network approaches often require high dimensional data to be able to detect patterns and correctly model input data. Since the total number of extracted features will be around $\mathcal{O}(10^3)$, performing dimensionality reduction may deteriorate the

performance of the algorithm, especially when many neural network designs for anomaly detection already take on similar procedures.

3.6 Model Training and Evaluation

Once the data is processed, it may be used to train ML algorithms that fit the problem. The first step is to split the data into training (μ) and testing ($1 - \mu$) sets. Even though we are only considering OCC models, anomaly-labeled data is also split into training and testing sets for Cross-validation (CV) purposes.

Each model is trained τ times, so that using different training and testing sets may reduce the performance metrics variance and ultimately return a confidence interval of those results.

Selecting an optimized model is not a straightforward procedure. In each training iteration, after splitting the datasets (one only containing “normal” behavior on the network and the other having anomalies) into training and testing sets, 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, and v to the number of splits, that defines a model ψ maximizing the average *F1-Score* (F_1) on multiple sets of cross-validation (CV):

$$\lambda_{opt} = \arg \max_{\lambda \in \Lambda} \frac{1}{v} \sum_{\substack{X \in CV_X \\ y \in CV_y}} F_1(\psi, \lambda, X, y) \quad (3.2)$$

For each combination, the overall “clean” training set is divided into training and CV sets using the Monte Carlo cross-validation splitting [86] (figure 3.10), which randomly selects a percentage γ of the overall set to serve as CV data, while the remaining is used for training the classifier.

Figure 3.10: Illustration of mini-batch approach cross-validation splitting.

Even though only clean network data is used to train the classifier, anomaly data is required to perform the model evaluation. Therefore, the CV set has also a ρ percentage of

anomalous samples. For every combination of hyperparameters, the splitting exemplified in equation 3.3 and illustrated in figure 3.11 is repeated v times.

$$\begin{aligned}
 \text{trainSet}_{\text{size}} &= \mu \times \text{cleanSet}_{\text{size}} \\
 \text{train}_{\text{size}} &= (1 - \gamma) \times \text{trainSet}_{\text{size}} \\
 \text{outlier}_{\text{size}} &= \frac{\text{trainSet}_{\text{size}} \times \gamma \times \rho}{1 - \rho} \\
 \text{cv}_{\text{size}} &= \text{trainSet}_{\text{size}} - \text{train}_{\text{size}} + \text{outlier}_{\text{size}}
 \end{aligned} \tag{3.3}$$

Figure 3.11: Splitting of the training datasets into training and cross-validation sets.

Besides feature selection or dimensionality reduction, an important technique to improve the performance of some models is feature scaling. This method essentially rescales the features such that they have the properties of a zero mean and standard deviation of one normal distribution [87]. This is a critical step since some models, such as SVMs and kNN, do not deal well with features whose values are on very distinct scales.

The goal is to model the “normal” behavior of the network as accurately as possible; therefore, the models should ideally not be learning with any specific anomalies, making one-class classifiers ideal for this case. With a proper set of features, they are able to detect unseen phenomena, hence, triggering alarms to possible outliers. As presented in section 2.5.3, algorithms such as One-Class Support Vector Machines [88], Isolation Forests [73], Local Outlier Factor [65], Kernel Density Estimation [60], and Gaussian Mixture Model [55] have proven to produce satisfactory results, even with their much different methods. Furthermore, deep neural networks approaches such as Variational Autoencoders [68] and Generative Adversarial Networks [71] [72] were also successfully used in outlier detection.

The built models should be as broad as possible; in an optimal scenario, they would be able to detect any anomalies they faced and avoid sending alarms when real “normal” traffic is flowing through the network, i.e., they would ideally have a small value of false positives. One way to avoid biasing the models is by tuning their hyperparameters according to a variety of different anomalies, as explained in section 3.5. 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 3.4.

$$\begin{aligned} \text{precision} &= \frac{TP}{TP + FP} \\ \text{recall} &= \frac{TP}{TP + FN} \\ F_1 &= \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \end{aligned} \tag{3.4}$$

As already mentioned, each model is trained τ times. This allows the dataset to be shuffled and to use different data in both training and testing. Then, the final metrics results are averaged and a confidence interval is computed according to equation 3.5, where σ is the standard deviation of a set of given metric values, n its total number and Z , a tabulated value, is the upper $\frac{1-C}{2}$ critical value for the standard normal distribution for a confidence interval of $C\%$.

$$CI_C = Z \frac{\sigma}{\sqrt{n}} \tag{3.5}$$

3.7 Summary

The mechanisms presented throughout this chapter aim at filling a gap regarding physical-layer anomaly detection strategies, since, up until this moment, we were not able to find any work that used RSSI data for pattern and behavioral analysis, rather than location-based outlier detection. There are similar strategies using spectral analysis; however, the majority use spectrum values as raw input data of their models. With this in mind, the main features of these mechanisms are described below:

- These are privacy-focused mechanisms, since it makes it very challenging to discern individual behavior at the physical layer when multiple devices share the same spectrum;
- It focuses on behavioral analysis, i.e., modeling silent and activity periods extracted from the RSSI time-series;
- It can be adapted to other radio technologies with relative ease, not only Wi-Fi, mainly because it uses Software Defined Radios (SDRs) for capturing physical-layer data;
- It does not force the administrator to feed the models with every imaginable type of anomalous behavior, as it only trains them with “clean” data;
- It resorts to multiple feature engineering techniques to avoid overflowing some models with non-relevant data;
- It is designed to support any one-class classifier;
- It focuses on optimizing the model detection performance based on metrics that work even on unbalanced datasets, such as the *F1-Score*.

The rationale for using OCC models is to make them as broad as possible against multiple out-of-normal behaviors. Nonetheless, one also needs anomalous samples to evaluate the performance of the model, and even perform feature engineering.

These mechanisms will be further explored in chapter 4, where they will be employed to perform outlier analysis in two different scenarios, one regarding a home automation set, and the other regarding a work-related environment.

Anomaly Detection in IoT Networks

To validate the anomaly detection methodology presented in chapter 3, we tested it in very distinct scenarios. In both presented situations, the goal is to verify how accurate the proposed methodology identifies the presence of periodic outlier behavior on a Wi-Fi channel containing IoT devices network data.

Furthermore, for each scenario, we will perform a thorough analysis of the recorded data; this will include all the pipeline, from the initial RSSI time-series until obtaining the relevance of the extracted features.

4.1 IoT Home Scenario

This scenario was designed initially to work as proof-of-concept. It mainly reflected a home automation scenario, where the network was essentially built of 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 is given by the usual interaction of the user with Alexa¹, Amazon’s personal assistant, consisting of casual requests, such as weather and sports updates, and Spotify streaming.

The first step was creating datasets of isolated events. For instance, one dataset would only contain clean traffic, i.e., sole interaction with Alexa, while each of the remaining would exclusively carry network traces of different anomalies. The next step was to blend clean and anomalous behavior, by keeping the tampered Raspberry Pi 3 generating traffic in the background at the same time the user interacts with the Amazon Echo. At last, a second “clean” node was added to add background traffic and make this scenario substantially more

¹<https://developer.amazon.com/en-GB/alexa>

realistic. This node, a laptop running Linux, would keep the channel occupied by also generating periodic data exchanges from uninterruptedly playing YouTube videos (figure 4.1).

Figure 4.1: Home scenario network diagram.

The tampered node - the Raspberry Pi 3 - 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, after referred as *scheduled 10s* and *scheduled 60s*, respectively;
- A file transfer which occurred with exponential distribution with 10 seconds mean, after referred as *exp 10s*.

We used a small data file, since, for short-term anomalies, it would not make sense to use big data transfers. Identifying the tampered node is the primary focus of our simulation work, however it may also be used to locate other types of network anomalies or behavioral changes. Additionally, we created a dataset composed by random samples of the already mentioned datasets with anomalous behavior, after referred as *mixture of samples*.

In all these cases, the sniffer that we used to perform all the signal captures is the HackRF One ², an open-source hardware SDR, adequately configured according to the environment, and connected to a laptop performing live FFT computations over the outputted data. For older device compatibility purposes, the tests were performed over the 2.4GHz band and on a “free” channel, i.e., where, before these experiments, there were no APs or devices working on that specific frequency. Another aspect to consider is the sample rate, which was set to 20MHz, so that it could cover the maximum range of frequencies. This was also the maximum allowed value by the HackRF.

With a 20MHz sample rate, every $1/20e6 \approx 50\text{ns}$, a new 8-bit quadrature sample (8-bit Q and 8-bit I) is generated. The HackRF fills a 262,144-bytes buffer - 131,072 samples - about 152 times per second. Storing all this data is not feasible as an hour-long dataset

²<https://greatscottgadgets.com/hackrf/>

could potentially reach hundreds of gigabytes. With this in mind, we only converted to the frequency domain the first 1024 of those 131,072 samples. Besides reducing the CPU load over the intensive FFT computations, it also reduces disk usage, as only 1024 samples of each buffer are converted and stored, which means that 1024 power samples are going to be stored approximately every 6.55ms. The number of samples used by the FFT are usually power of 2 - typically 512, 1024, and 2048. In this range of values, a 1024-sample FFT will produce 19.5kHz bins, while a 2048-sample FFT will return 9.75kHz bins. Since we are working on the 2.4GHz frequency domain, the difference in precision is not relevant.

When calibrating the HackRF, the goal is to separate the ground noise from activity in the spectrum. In the conducted tests, its amplifier Low-Noise Amplifier (LNA) gain was set to 8dB, and the Variable-Gain Amplifier (VGA) gain to 0dB. The first corresponds to the main gain, while the latter amplifies the ground noise. We tested multiple combinations of these parameters, and through graphical representations of the outputted signal, those were chosen. Nonetheless, these proved to correctly operate on both the proof-of-concept scenarios, the one with much less noisy environment, and the scenario presented in section 4.2.

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. Since the AP was wall-mounted in a place with limited physical access, the sniffer was placed about 3 meters from the nodes and 10 from the AP, in-between them.

4.1.1 Dataset Exploration

All datasets regarding the home automation scenario had 6 to 8 hours worth of data. The FFT processing of the IQ data outputted by the HackRF resulted in a file containing RSSI values of every frequency of interest approximately every 6.55ms. In reality, the stored file would contain a matrix. Since we were working with Wi-Fi, those 1024 power values can then be averaged to obtain the mean RSSI over the entire channel. This is a great way of checking if the captured data is valid and can be used for training and testing the classifiers.

As illustrated in figure 4.2, the outliers have completely different behavior than the typical traffic characteristics of the network. The first plot mainly shows an empty channel with a single data exchange - a possible interaction with Alexa - after 40 seconds, whose duration was close to 15 seconds. In contrast, both the second and third plots illustrate the behavior of two different outliers, the first with short data bursts every 10 seconds, and the second every 60 seconds.

Figure 4.3 now shows the shape of the channel occupancy when a tampered node is running in the background, masked by the usual behavior of the network. Besides the typical interactions with Alexa, which show up at 100 (*scheduled 10s*) and 50 seconds timestamps (*scheduled 60s*), the anomalous behavior is also present - in the first case, every 10 seconds, and in the second, which is overlapping the non-anomaly behavior, every 60 seconds. Even though the typical network traffic masks anomalies, figure 4.4 shows that the channel has far more activity when the third node, the laptop streaming YouTube videos uninterruptedly,

Figure 4.2: RSSI power values over time in three different datasets, the first containing only clean traffic, the others exclusively with different anomalous behaviors.

Figure 4.3: RSSI power values over time in two anomalous datasets, masked behind typical network traffic.

is present. Also, the additional YouTube traffic is periodic, much like the anomalies, which makes it harder for the models to identify these patterns.

The next step in the process is to extract activity and silent periods from the RSSI time-series data. As mentioned in chapter 3, we need to establish a threshold that separates activity from silence. Using equation 3.1 in section 3.3, activity and silent periods are easily differentiated.

The activity and silence thresholds were yet other parameters that had to be set. The first was set 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 drivers 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, features were also extracted using a zero-sample silence threshold, i.e., not filtering out non-relevant network activity. Figure 4.5 shows the output of applying

Figure 4.4: RSSI power values over time in two anomalous datasets, masked behind typical network traffic of two nodes, the Amazon Echo and Linux laptop streaming YouTube videos.

non-relevant activity filtering to an RSSI time-series describing the behavior of a 60-second periodic anomaly. Even though the data on the left seem to contain more noise, that behavior may describe other underlying mechanics or characteristics of this network that were removed, as illustrated on the right plot.

Figure 4.5: Activity timeline without (left) and with (right) *non-relevant activity filtering*, extracted from a RSSI time-series data.

The data is 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. As showcased in section 3.4, mean, median, standard deviation, 75th, 90th, 95th and 99th percentiles, minimum, and maximum values regarding activity and silent periods over any of the sub-windows are extracted from the time-series data.

The majority of the datasets had around 6 to 8 hours worth of data. The dataset with a “clean” network was a mixture of the multiple datasets to capture multiple normal scenarios, resulting in a much bigger set when compared to datasets with contamination. To balance their size, the samples of the smallest sets were duplicated until their size did not exceed the size of the biggest set. For example, considering that the “clean” set has 20,000 samples and an anomaly set has 9,000, the latter will be doubled to 18,000 samples.

Not all these features are equally relevant for detecting outliers. Figure 4.6 compares the mean silence and activity periods on two distinct datasets, the left corresponding to a

Figure 4.6: Comparison of the mean silence and activity periods on clean and outlier behaviors, without applying *non-relevant activity filtering*.

clean environment, i.e., only Amazon Echo data transfers, and the right corresponding to the tampered node uploading a 1MB file every ten seconds. In both cases, no filtering regarding unaccounted activity was applied. For any of the sub-windows sizes (15 seconds, 1, 5, and 15 minutes), silent periods ranged from 1 to 5 seconds in both datasets, while activity periods show a slightly higher tendency on the 10-seconds outlier. Even though the uploaded data were small, if there is background noise when the sniffer registers the IQ values between data transfers, the length of activity periods may increase. Furthermore, averages tend to decrease when there are outlier values in the data frame. This clearly shows that mean values, especially when not using non-relevant activity filtering, are possibly not a good enough feature to differentiate normal from outlier behavior.

On the other hand, there are noticeable differences between those two datasets when *non-relevant activity filtering* was applied, as shown in figure 4.7. Even though there is not much distance between the length of activity periods - slightly higher on the clean dataset -, we are now able to differentiate silent periods. The length of silent periods considerably increased when removing small activities in the dataset that were masking the behavior of the Amazon Echo, such that some reached 60 seconds, while the majority situated between 10 and 20 seconds. One noticeable thing on these plots is that there is a correlation between activity and silence periods; considering that window sizes are fixed, more extended activity periods results in shorter silent periods, and vice-versa.

Figure 4.7: Comparison of the mean silence and activity periods on clean and outlier behaviors, when applying *non-relevant activity filtering*.

Still, there are features that better describe the behavior of the network in those datasets. For instance, when comparing a dataset free of anomalies with one which contains the tampered

node sending data every 60 seconds, through the 99th percentile, we are able to perceive the differences between the two behaviors, even when not filtering out non-relevant activities, as illustrated in figure 4.8.

Figure 4.8: Comparison of the 99th percentile silence and activity periods on clean and outlier behaviors, without applying *non-relevant activity filtering*.

Figure 4.9 outputs the very same data, but with unaccounted activity filtering. These values show that the silent periods of the outlier never surpass the 60-seconds mark, as one would expect, with short-duration activities, since the uploaded file was considerably small. On the contrary, a network free of these anomalies may sometimes have silence periods that go over 60 seconds. Regular Spotify music streaming patterns, or regular interaction with Alexa, may justify this behavior.

Figure 4.9: Comparison of the 99th percentile silence and activity periods on clean and outlier behaviors, when applying *non-relevant activity filtering*.

As mentioned in section 3.4, we extracted a scalogram over the activity values on the largest sub-window size used: in this case, 15 minutes. The scales data frame consisted of 64 logarithmic spaced values between 0 and 900 seconds - the window size. As observed in figure 4.10, filtering out non-relevant activities does not affect the output of the scalogram, as it seems to maintain the periodic properties of the original unfiltered timeline.

Looking closely at each of the plots, one can notice that clean behavior does not have a clear tendency, despite some minor local maxima at 60 and 240 seconds. On the opposite, the *scheduled 10s* has a remarkable local maxima at around the 10-seconds timestamp. On a similar note, the scalogram regarding the outlier uploading data every 60 seconds also contains the absolute maximum around this value. Similar to the clean network traces, the anomaly samples with tampered node uploading data with a 10-seconds mean exponential distribution does not seem to have relevant local maxima; yet, it maintains higher energy values on scales

around the 10-seconds timestamp as it would be expected.

Figure 4.10: Scalogram of the activity timeline on clean and multiple outlier behaviors.

4.1.2 Feature Dimensionality

Given the number of features, we analyzed the importance of statistical and wavelets data. As mentioned in section 2.5.2, there are a variety of models that provide feature-importance information. Ensemble models, such as Extra Trees, Decision Trees, and Random Forests classifiers, complement perfectly the feature classification system proposed in section 3.5. We used the Scikit-learn ³ Python implementation of the mentioned algorithms to conduct this process. To determine the k most relevant features, one must test multiple values of k , select the top features, and use them for training multiple models, and record their $F1$ -Scores. Since the models need anomalous data to tune their hyperparameters and perform their final performance evaluation, samples of all the anomalous datasets were merged into a single dataset.

Regarding statistical features, we trained the models, in this particular case, an One-Class Support Vector Machine (OC-SVM) and an Isolation Forest (IF), with $k \in \{1, 3, 5, 10, 15, 20, 25, 72\}$ most relevant inputs, with five cross-validation splits ($v = 5$). For datasets without *non-relevant activity filtering*, figure 4.11 shows that the optimal number of features ranges between 5 and 10, as the $F1$ -Score between those numbers has a slight difference. We opted for using 10 out of the 72 original features since it can provide more insights about the input data, even at the cost of longer training times. On the other hand, when not excluding unaccounted activities, the difference in performance between using 5 and 10 features becomes steeper; thus, for these datasets, we only selected 5 out of the original 72. We also tested PCA for dimensionality reduction with a set $\{1, 3, 5, 10, 15, 20, 25, 72\}$

³<https://scikit-learn.org/stable/>

of components. It turns out that using PCA components as input features produces worse results; therefore, only feature selection mechanisms will be used in classification.

Figure 4.11: *F1-Score* for multiple statistical feature subsets, using feature selection and PCA.

The same methodology was applied to local maxima scales of the CWT and its respective energy values. The models were also trained with 5 cross-validation splits, and $k \in \{2, 4, 6, 10, 20\}$. As observed in figure 4.12, for data with or without filtering, the optimal number of scales is 5, which gives a total of 10 features. Once again, PCA components perform poorly when compared to feature selection methods.

Figure 4.12: *F1-Score* for multiple wavelet local maxima feature subsets, using feature selection and PCA.

Figure 4.13: *F1-Score* for multiple scalogram feature subsets, using feature selection and PCA.

Finally, the same procedure was conducted with all the scalogram energy data, not only the local maxima scales and its energy values, as explained above. The value of v was also 5, and k belonged to the $\{2, 4, 6, 10, 20, 64\}$ set. Figure 4.13 shows that the optimal number of features is using all the available scales. For neural network approaches, having high-dimensional data such as these energy values may be beneficial. Despite the differences between the *F1-Scores* when using 20 or 64 scales being relatively small, there is a slight upper tendency when using 20 scales. With or without filtering, PCA produces the worst results out of all.

4.1.3 Classification Results

The final step is to train each model with the selected features and to extract the results. The tested models were OC-SVMs, IFs, LOF, KDE, and GMM, using Scikit-learn ⁴ Python implementations, running on a Jupyter⁵ notebook. For each dataset and each model, the data was randomly split into 80% training and 20% testing five times ($\mu = 0.8$, $\tau = 5$), and split according to the Monte Carlo cross-validation method also five times ($v = 5$) for every tested set of hyperparameters. The batch size used for the CV sets was equal to 30% of the training set ($\gamma = 0.3$), and so, 70% of the training set with “clean” data was used for training, while the remaining 30%, plus an equal amount of outlier samples ($\rho = 0.5$), were used as CV data. Table 4.1 showcases which ranges of parameters were used for tuning the classical models. Values over or under the present ranges performed worse than the presented tested combinations.

Model	Parameters ranges
OC-SVM	sigma: [0.01, 3.0], nu: [0.01, 0.5]
IF	ms: [1, 4999]
LOF	nn: [10, 5000], ct: [0.1, 0.5]
KDE	kernel: guassian, bw: [0.01, 10.0], cutoff: [0.10, 0.35]
GMM	n_components: [1.0, 50], cutoff: [0.1, 0.35]

Table 4.1: Ranges of parameters used on classical OCC models tuning.

Since this scenario was considered a test run, we only considered classical models. Their performance does not scale well with higher-dimensional feature spaces. With this in mind, the input features vector corresponded to a concatenation between the most relevant statistical features, and the most relevant local maxima and respective energies from the extracted scalogram, as exemplified in the previous section.

Tables 4.2 and 4.3 show that KDE outperforms all other models, reaching an almost perfect *F1-Score* across the different tested datasets. The OC-SVM almost matches its performance, especially when using *non-relevant activity filtering*. In general, the GMM was the worst performer, even if its detection rates increased considerably, and were on pair with the IF models when the filtering was active.

Table 4.4 shows that, even though KDE was able to produce the best detection rates, it does so at the cost of considerably longer training times, averaging about 64 minutes to perform hyperparameter tuning using 5-fold Monte-Carlo cross-validation strategy and modeling the entire training set. Additionally, it has longer testing times, averaging about 15 seconds to classify about 7555 samples, while the remaining models took less than 3 seconds to do it. IFs were the fastest to converge, but the ones producing the worse classification results. The OC-SVM seems to have excellent performance while maintaining reasonable

⁴<https://scikit-learn.org/stable/>

⁵<https://jupyter.org/>

training times. GMMs have less optimal detection rates but are about three times as fast to train as OC-SVMs.

Type of anomaly		WITHOUT non-relevant activity filter				
		OC-SVM	IF	LOF	KDE	GMM
1 licit node + scheduled 10s	<i>Precision</i>	98.3±0.3	86.5±0.7	86.9±0.5	97.0±0.2	65.9±1.5
	<i>F1-Score</i>	99.1±0.2	92.8±0.4	93.0±0.3	98.5±0.1	79.5±1.1
1 licit node + exp 10s	<i>Precision</i>	98.3±0.3	86.6±0.6	86.8±0.6	96.9±0.3	65.7±1.7
	<i>F1-Score</i>	99.1±0.2	92.8±0.3	92.9±0.4	98.4±0.1	79.3±1.2
1 licit node + scheduled 60s	<i>Precision</i>	98.6±0.1	65.3±8.7	87.2±0.5	97.0±0.3	65.1±0.9
	<i>F1-Score</i>	99.2±0.1	47.1±1.9	92.8±0.2	98.4±0.1	78.8±0.6
2 licit nodes + scheduled 10s	<i>Precision</i>	93.8±0.4	83.4±0.5	83.6±0.7	96.0±0.3	59.2±2.0
	<i>F1-Score</i>	96.5±0.2	90.9±0.3	91.0±0.4	98.0±0.2	74.4±0.9
2 licit nodes + exp 10s	<i>Precision</i>	98.0±0.2	87.9±0.4	88.4±0.4	97.4±0.2	69.3±1.8
	<i>F1-Score</i>	99.0±0.1	93.5±0.2	93.8±0.2	98.7±0.1	81.8±0.6
2 licit nodes + scheduled 60s	<i>Precision</i>	98.8±0.2	87.4±0.7	88.7±0.4	97.6±0.2	68.0±1.4
	<i>F1-Score</i>	99.4±0.1	93.3±0.4	94.0±0.2	98.8±0.1	80.9±1.0
Mixture of samples	<i>Precision</i>	95.2±0.2	87.4±0.8	90.6±0.5	97.9±0.1	72.4±1.2
	<i>F1-Score</i>	94.8±0.3	77.7±0.6	94.5±0.3	98.8±0.0	84.0±0.8

Table 4.2: Home automation scenario classification results (in percentage, with a 95% confidence interval), when not using *non-relevant activity filtering*.

Type of anomaly		WITH non-relevant activity filter				
		OC-SVM	IF	LOF	KDE	GMM
1 licit node + scheduled 10s	<i>Precision</i>	98.4±0.2	84.3±0.3	86.6±0.8	98.9±0.1	84.8±0.8
	<i>F1-Score</i>	99.2±0.1	91.5±0.2	92.8±0.4	99.5±0.1	91.8±0.5
1 licit node + exp 10s	<i>Precision</i>	98.1±0.2	84.7±0.4	87.3±0.4	99.0±0.2	84.2±0.6
	<i>F1-Score</i>	99.0±0.0	91.6±0.2	93.2±0.2	99.5±0.1	91.4±0.4
1 licit node + scheduled 60s	<i>Precision</i>	96.1±0.3	78.4±2.0	86.7±0.2	99.0±0.0	84.8±0.9
	<i>F1-Score</i>	97.9±0.2	73.5±5.3	92.7±0.2	99.5±0.0	91.7±0.5
2 licit nodes + scheduled 10s	<i>Precision</i>	98.1±0.0	81.1±1.1	83.9±0.9	98.8±0.1	80.3±1.5
	<i>F1-Score</i>	99.0±0.0	89.6±0.7	91.2±0.6	99.4±0.1	89.1±0.9
2 licit nodes + exp 10s	<i>Precision</i>	98.6±0.1	86.5±0.3	88.7±0.4	99.2±0.1	87.0±1.0
	<i>F1-Score</i>	99.3±0.0	92.8±0.2	94.0±0.2	99.6±0.1	93.0±0.6
2 licit nodes + scheduled 60s	<i>Precision</i>	98.7±0.2	86.8±1.0	88.7±0.1	99.2±0.2	85.9±0.9
	<i>F1-Score</i>	99.4±0.1	92.9±0.6	94.0±0.1	99.6±0.1	92.4±0.5
Mixture of samples	<i>Precision</i>	97.2±0.2	86.9±0.4	90.7±0.3	99.4±0.1	88.6±1.4
	<i>F1-Score</i>	98.4±0.1	86.1±1.7	95.0±0.2	99.7±0.0	94.0±0.8

Table 4.3: Home automation scenario classification results (in percentage, with a 95% confidence interval), when using *non-relevant activity filtering*.

In sum, this was a simple scenario, since it only monitored three nodes maximum, one of them not licit. The usage of the spectrum is relatively low, which makes it easier for some models to detect outlier behavior, especially boundary models such as the OC-SVM. In a scenario in which outlier and “clean” behavior are not so easy to discern, it may lower its

performance, even when using GRBF kernels. Moreover, looking at the results, one may assume they are overfitted. Even though we do not present classification results with the training sets, they show that this bias is often less than one percentage point.

Model	Training time	Training set size	Testing time	Test set size
OC-SVM	18min28s $\pm$ 3min36s	16371	<1s	7555 $\pm$ 207
IF	0min23s $\pm$ <1s	16371	<1s	7555 $\pm$ 207
LOF	24min51s $\pm$ 2min0s	16371	0min5s $\pm$ 0min1s	7555 $\pm$ 207
KDE	64min50s $\pm$ 8min2s	16371	0min15s $\pm$ 0min1s	7555 $\pm$ 207
GMM	5min10s $\pm$ 0min33s	16371	<1s	7527 $\pm$ 208

Table 4.4: Average (with 95% confidence interval) training and testing durations in the home automation scenario, corresponding to a $\tau = 1$ and $v = 5$.

Overall, using *non-relevant activity filtering* seems to improve the results given that, in a channel with considerably few data exchanges, removing unaccounted activity may promptly reveal the underlying communication periodicities of some devices.

4.2 IoT Work/Lab Environment Scenario

Contrasting with the scenario presented in section 4.1, which was initially thought as a trial-run to validate the proposed methods in chapter 3, the following scenario showcases an attempt at detecting the same type of outlier behavior but in an environment with a higher number of nodes connected to the same AP, specifically a Cisco AIR-AP1231G-E-K9. As the amount of non-tampered connected devices is superior - always more than five, in comparison to the maximum two in the previous case -, the Wi-Fi channel will be far more often occupied, making it harder to detect the presence of periodic outlier behavior.

Figure 4.14: Work/lab network diagram.

The goal was to also identify a periodic outlier, within an environment containing interactions with Amazon Alexa - mainly Spotify streaming, which is also periodic - and multiple laptops browsing the web and performing a multitude of tasks. In sum, the initial scenario is masked behind the day-to-day of the researches in a room at Instituto de Telecomunicações ⁶ in Aveiro (figure 4.14). The same tampered node was used, with the following distributions:

- A file transfer which occurred every 10 or 60 seconds, after referred as *scheduled 10s* and *scheduled 60s*, respectively;
- A file transfer which occurred with Gaussian distribution with 10 mean, and 5 seconds standard deviation, after referred as *gaussian 10s*;
- A file transfer which occurred with exponential distribution with 10 or 60 seconds mean, after referred as *exp 10s* and *exp 60s*, respectively.

We also resorted to a HackRF with 20MHz sample rate, outputting a RSSI sample per 6.55 milliseconds, and a FFT window size of 1024 samples. The HackRF LNA gain was set to 8dB, and the VGA gain to 0dB, just like in the previous case. It was placed about three meters away from the access point, and both were in a central position in the room.

4.2.1 Dataset Exploration

Even though this was a more complex scenario, except for recording about 6 hours of “clean” data, we only captured 2 to 3 hours of data per dataset. Due to schedule incompatibilities, it was sometimes challenging to conciliate a time of the day where a high number of people were in the room and used their devices to generate traffic. Nevertheless, just like the home automation scenario, since all devices are connected to the same 2.4GHz Wi-Fi AP, they were all on the same channel. This channel corresponded to one that was previously free - in this case, channel 11 -, and all the captured RSSI data was averaged over the frequencies of interest.

Figure 4.15 shows that there are some differences when compared with the same figure in the previous chapter (figure 4.2). Activities seem to be less noticeable, but cranking up the VGA gain only made the ground noise more noticeable, given that the latter is far superior due to the higher number of nodes and overall background noise. We conducted the tests regarding the home automation scenario in a location where there was few to none interference in the 2.4GHz band; thus, making it easier to discern activity from ground noise. Despite recurrently having small activities at all times, only more significant data transfers are undoubtedly clear. However, these independent plotted points may represent how the network behaves and how it communicates with all the connected nodes.

Next on the pipeline is the discretization of activity and silent periods. After, the thresholds corresponding to the maximum time frame without sign of activities in the network was also set to 75 samples (500ms), and the one corresponding to the minimum activity duration to be considered non-relevant activity was set to 5 samples (33ms). In comparison with figure 4.5,

⁶<https://www.it.pt/>

Figure 4.15: RSSI power values over time in three different datasets, the first containing only clean traffic, the others containing a masked anomalous behavior.

in figure 4.16, the amount of activities present on the timeline is vastly superior, even when removing unaccounted activity.

Figure 4.16: Activity timeline obtained from not applying (left) and applying (right) *non-relevant activity filtering* on RSSI time-series data.

Once again, similar to the work done in section 4.1, the data is split into sliding windows of 15 minutes with 2 seconds sliding distance, with a set of sub-windows of 30, 60 and 300 seconds. Additionally, besides the usual statistical features (mean, median, standard deviation, 75th, 90th, 95th and 99th percentiles, minimum, and maximum values regarding activity and silent periods), relevant data from the scalogram was also used. Similarly, dataset sizes were also normalized.

When not applying *non-relevant activity filtering*, for the exception of some longer activity periods on the network traces free of anomalies - which can appear when there are many devices that are trying to transmit and receive data simultaneously -, the latter with and without a tampered node sending data every 10 seconds is considerable identical as the average activity duration is close to 5, and the average silence length sits between 1 and 3 seconds (figure 4.17).

In this case, using unaccounted activity filtering does not seem to produce better results

Figure 4.17: Comparison of the mean silence and activity periods on clean and outlier behaviors, without applying *non-relevant activity filtering*.

when it comes to visually discerning different behaviors (figure 4.18). The only noticeable difference is that silence periods average slightly increased, as one would expect since some activity points were removed.

Figure 4.18: Comparison of the mean silence and activity periods on clean and outlier behaviors, when applying *non-relevant activity filtering*.

Once again, taking a look at figures 4.19 and 4.20, one may observe that there is not much difference between removing non-relevant activities or leaving them intact. However, it is still possible to visually differentiate both datasets. For instance, there are slightly longer silent periods on the clean dataset, since the outlier behavior on the figure corresponds to a node sending data every 10 seconds, which maximizes silent periods at around 10 seconds. Nevertheless, it still surpasses the 10 seconds mark, possibly due to background noise generated by other devices, generating collisions and delays when communicating with the AP.

Figure 4.19: Comparison of the maximum silence and activity periods on clean and outlier behaviors, without applying *non-relevant activity filtering*.

Additionally, we did not expect the observable vertical non-correlated features in the

“clean” dataset. Similar to the data presented in the previous scenario, we expected a correlation between silence and activity durations due to fixed window sizes (a longer activity produces shorter silence periods). We assume that this behavior results from the more prolonged activities due to more extended interactions without having a node transmitting data recurrently and prevent other devices from accessing the AP. Furthermore, as already mentioned, this environment had much more uncontrollable background noise, which may also contribute to these results.

Figure 4.20: Comparison of the maximum silence and activity periods on clean and outlier behaviors, when applying *non-relevant activity filtering*.

Just like the scalogram extracted in the previous scenario, it is also clear that the differences between obtaining it on a timeline with or without filtering are almost indistinguishable (figure 4.21). We can assume that local maxima at the 60-second mark and its resonance frequencies - 120 and near 240 seconds - correspond to the typical duration of a person reading a web page and Spotify streaming on the Amazon Echo - it buffers an entire song right off the bat. However, it is mixed with other activities in the network, either related to human behavior or the operating system functioning of the connected devices. Additionally, it is essential to notice that, due to the internal operation of the CWT, higher frequencies showcase higher energy values.

Figure 4.21: Scalogram of the activity timeline on clean and multiple outlier behaviors.

4.2.2 Feature Dimensionality

The next step was to perform an analysis of the relevance of input features. Besides using the feature selection mechanisms proposed in section 3.5, we also test feature dimensionality reduction techniques such as PCA. Similarly to the previous example, we used the Scikit-learn ⁷ Python implementation of Extra Trees, Decision Trees, and Random Forests classifiers, and PCA. For any subset of features, the number of cross-validation splits was 5 ($v = 5$), and for both the number of relevant features and PCA components, k belonged to the $\{1, 3, 5, 10, 15, 20, 25, 72\}$ set.

Figure 4.22 shows that, compared to feature selection, PCA has sub-optimal performance. On the other hand, regarding feature selection methods, using 5 or 10 features on datasets without unaccounted activities produces similar $F1$ -Score results. Even though it might increase training times, we opted for using 10 features, as the extra information may be helpful for other models not tested up until this point, like KDE or GMM. When not filtering out those activities, the plot shows that there is not much difference between using the 5, 10, or 15 most relevant features to perform classification. Although it produced an every so slightly worse $F1$ -Score result, we opted for a compromise and considered the 10 most relevant features. Even following the same strategy as in the previous example - choosing the maximum possible number of features out of those that maximized the F_1 metric -, using 15 features could make training times even longer, especially in distance or density-based approaches.

Figure 4.22: $F1$ -Score for multiple statistical feature subsets, using feature selection and PCA.

Both the previous techniques were applied to local maxima scales of the CWT and its energy values. As observed in figure 4.23, for data with or without filtering, the number of features that maximized performance was 10, which matches a total of 5 scales and respective energy values. Again, we will not use PCA components as input features since their performance is worse than using the most relevant features exclusively.

Figure 4.24 illustrates the same study conducted on all the scalogram scales. It shows that there is a slight difference between using 20 or the total number of scales. In this case, PCA does not show such big performance losses when compared to feature selection mechanisms.

⁷<https://scikit-learn.org/stable/>

On the other hand, it is not advisable to use a fixed subset of scales. It may work on these data, as we are comparing it with only a few types of anomalies whose periodicities may coincide with those scales, but it may not work with different anomalous behaviors. It is worth noticing that this is not a problem for local maxima scales, as the only study we are performing regarding those features is how many maximum points to use, and not which specific local maxima scales. One more time, with or without filtering, PCA shows not to be suitable for this problem.

Figure 4.23: *F1-Score* for multiple wavelet local maxima feature subsets, using feature selection and PCA.

Figure 4.24: *F1-Score* for multiple scalogram feature subsets, using feature selection and PCA.

4.2.3 Classical Models Classification Results

The data were randomly split into 80% training, and 20% testing ($\mu = 0.8$) for each of five conducted training procedures on every model ($\tau = 5$). For classical models, it was then split according to the Monte Carlo cross-validation method five times ($v = 5$) for every set of hyperparameters, with a batch size equal to 30% of the training set ($\gamma = 0.3$), and a 50% rate of outlier samples ($\rho = 0.5$) in the CV data. As these models demand low-dimensional inputs, only statistical and scalogram local maxima were used, totaling 16 features. Identical to the previous scenario, we resorted to Scikit-learn implementations of these models and Jupyter to run the tests. The parameters used to tune the models were the same as in the previous scenarios (table 4.1).

As illustrated in tables 4.5 and 4.6, and identical to the previous scenario, KDE outperforms all other models, seizing an almost perfect detection score. Opposite to the results obtained in section 4.1.3, GMMs are now able to correctly model the “behavior” of anomaly-free data, and they now perform slightly better than the OC-SVMs, and also slightly worse than the KDE models. One common aspect to the majority of the results is that the lower *F1-Scores* come from lower *Precision* rates, which means they sometimes produce false positives, even though they are able to identify almost every anomaly (high *Recall*).

Opposite to what happened in the home automation scenario, using *non-relevant activity filtering* does not have the same impact, as the classification results only show a small variation when it is active. There was only one significant difference when using IFs to classify the

anomalous behavior of sending data every minute, even though it was the model that obtained the worst detection results.

In sum, density-based models, namely KDE and GMM, obtained the best classification results, closely followed by OC-SVMs. Classical models proved that, even in scenarios with a higher number of devices and increased background noise, they are still capable of discerning outlier behavior, when they are trained with data that corresponds to the standard network operation.

Type of anomaly		WITHOUT non-relevant activity filter				
		OC-SVM	IF	LOF	KDE	GMM
Scheduled 10s	<i>Precision</i>	95.1±0.5	80.2±0.5	87.4±0.4	99.5±0.2	96.2±0.3
	<i>F1-Score</i>	97.4±0.3	89.0±0.3	93.5±0.2	99.7±0.1	98.1±0.2
Gaussian 10s	<i>Precision</i>	95.7±0.4	82.4±0.6	89.8±0.6	99.6±0.1	97.1±0.7
	<i>F1-Score</i>	97.1±0.2	90.1±0.3	94.6±0.3	99.8±0.0	98.5±0.4
Exp 10s	<i>Precision</i>	94.2±1.3	83.7±0.8	84.7±0.9	99.4±0.1	96.2±0.5
	<i>F1-Score</i>	96.2±0.5	91.0±0.5	94.6±0.3	99.7±0.0	98.1±0.3
Scheduled 60s	<i>Precision</i>	90.3±0.5	55.7±3.4	89.0±0.9	99.6±0.1	96.6±0.2
	<i>F1-Score</i>	94.3±0.2	33.9±3.6	91.6±1.1	99.8±0.1	98.3±0.1
Exp 60s	<i>Precision</i>	94.7±1.1	83.3±0.3	89.5±0.5	99.6±0.1	97.0±0.2
	<i>F1-Score</i>	96.5±0.4	90.7±0.2	94.5±0.3	99.8±0.1	98.5±0.1
Mixture of samples	<i>Precision</i>	93.2±0.7	82.3±0.5	90.4±0.6	99.8±0.0	96.8±0.0
	<i>F1-Score</i>	95.2±0.4	83.6±0.5	94.2±0.3	99.8±0.1	98.4±0.1

Table 4.5: Work/lab scenario classical models' classification results (in percentage, with a 95% confidence interval), when not using *non-relevant activity filtering*.

Type of anomaly		WITH non-relevant activity filter				
		OC-SVM	IF	LOF	KDE	GMM
Scheduled 10s	<i>Precision</i>	94.9±0.5	80.1±1.0	87.0±0.7	99.5±0.2	95.2±0.7
	<i>F1-Score</i>	96.4±0.4	88.9±0.6	91.4±0.5	99.7±0.1	97.5±0.4
Gaussian 10s	<i>Precision</i>	95.8±0.6	81.6±1.0	90.0±0.7	99.5±0.1	95.9±0.4
	<i>F1-Score</i>	97.1±0.2	86.1±1.2	94.2±0.5	99.8±0.0	97.9±0.2
Exp 10s	<i>Precision</i>	92.8±0.4	82.7±1.7	90.2±0.6	99.6±0.1	95.8±0.5
	<i>F1-Score</i>	96.1±0.2	86.7±1.2	93.8±0.7	99.8±0.1	97.8±0.3
Scheduled 60s	<i>Precision</i>	92.9±0.4	74.6±1.4	87.6±2.9	99.6±0.2	96.4±0.3
	<i>F1-Score</i>	95.2±0.2	64.6±2.8	89.1±2.8	99.8±0.1	98.1±0.2
Exp 60s	<i>Precision</i>	92.5±0.6	81.1±0.6	90.1±0.6	99.7±0.1	95.7±0.5
	<i>F1-Score</i>	96.0±0.3	85.8±0.9	93.6±0.5	99.8±0.1	97.8±0.3
Mixture of samples	<i>Precision</i>	93.3±0.1	82.4±0.6	90.2±0.3	99.6±0.2	96.6±0.4
	<i>F1-Score</i>	96.1±0.1	83.8±1.4	93.3±0.5	99.8±0.1	98.3±0.2

Table 4.6: Work/lab automation scenario classical models' classification results (in percentage, with a 95% confidence interval), when using *non-relevant activity filtering*.

4.2.4 Neural Networks Classification Results

In contrast with the previous scenario where we only resorted to classical models - OC-SVMs, IFs, LOF, KDE, and GMMs - to look for outlier behavior in network traces, in this case, we will also use neural network approaches, such as AEs, Variational Autoencoders (VAEs), and Generative Adversarial Networks (GANs). To build these models, we used Keras ⁸, a deep learning Python framework that is able to work on top of other low-level frameworks, such as Tensorflow ⁹, and allows easy prototyping of deep learning models.

The process of tuning neural networks was not as automated as the one used for classical models. The number of hyperparameters to properly set in a neural network is vastly superior than the one on a classical model. In sum, when fine-tuning a deep neural network, we have to take into consideration the number of hidden layers, and the number of units per layer, which activation functions to use in each of the layers and its parameters, the amount of regularization and which type (L1, L2, or even Dropout), which loss function one should use, which optimizer should minimize the loss of the model and its numerous parameters, such as the learning rate. All of this should be tested, and carrying out an automated process could waste much time in useless combinations; therefore, we manually tested the majority of these parameters, while trying to minimize the loss function. To fine-tune these hyperparameters, we tested multiple combinations on a dataset containing a mixture of different anomalous traffic (*Mixture of samples*), and use the architecture that best fitted it on the remaining datasets.

Nevertheless, on each of the five training procedures per dataset ($\tau = 5$), the input data was divided into training and testing with an 80/20 split. The majority of these models return a reconstruction error, like the MSE; therefore, one must determine a threshold that separates anomalies from non-anomalies. Usually, clean network behavior would produce lower reconstruction errors, while outliers would produce higher values. This threshold is the only hyperparameter whose discovery is automated, which is also done by using Monte Carlo cross-validation with a $v = 5$, a CV batch size of about 30% of the training set ($\gamma = 0.3$), and a 50% rate of outlier samples ($\rho = 0.5$). Additionally, all models were trained with 150 epochs, as it seemed sufficient for the losses to converge. There is at least one update to the neuron weights in the neural network every epoch. However, there are multiple update strategies, as those weights can be updated:

- Once per epoch, by updating them after computing forward and backpropagation for all samples, at the expense of additional computational time, i.e., a batch gradient;
- Once every sample, i.e., a stochastic gradient. This approach is not always able to converge at the local minimum loss value, and does not always mean that there is a speedup when compared with the batch gradient, as we lose the vectorization properties of a batch approach;

⁸<https://keras.io/>

⁹<https://www.tensorflow.org/>

- Multiple times per epoch in the form of mini-batches. Theoretically, it is the fastest approach as it is able to use vectorization, and it can make progress without the entire training set.

There are multiple optimizers built as extensions of the vanilla Stochastic Gradient Descent (SGD) by either introducing momentum, such as the Momentum optimizer, or by adding adaptive learning rate, such as Adagrad, and RMSProp. However, Kingma and Ba [89] combined both these strategies into a fast and low-memory consumption algorithm, called Adam, that works particularly well when the input data is sparse, which is the case, as clean traffic may take multiple forms. Besides all the mentioned reasons, a 256-minibatch Adam was the optimizer that produced the best results on our tests.

Opposite to classical models, neural networks are able to find correlations in high-dimensional input data. Moreover, AEs and VAEs encode the data into lower dimensions, similarly to PCA; therefore, one should use more input features than the 16 on the classical models. Consequently, we will use all the available features, except data from wavelet local maxima, as we would end up with duplicate information. Then, neural network models will be fed with 136 inputs, $18 \times 4 = 72$ corresponding to statistical features extracted from multiple sub-windows, plus 64 scalogram energy values.

Autoencoder

The first tested model was a classical Autoencoder (AE). Its goal is to learn how to reconstruct the input data after compressing it into a lower-dimensional set. The final architecture is depicted in figure 4.25. The 136 inputs were encoded into a 9-dimensional vector by alternate layers using ReLU and Hyperbolic Tangent (equation 2.25) activation functions, reducing by half the number of neurons. The decoder followed a mirror-like approach of the encoder to reconstruct the initial 136-dimensional vector.

Figure 4.25: Final AE architecture.

The final layer is provisioned with a Hyperbolic Tangent activation since the model is trying to replicate the inputs that also belong to the $[-1; 1]$ interval. Moreover, we are not

looking to having any sparsity on the output layer; therefore, not considering the ReLU function or any of its variations. Then, one would compute the MSE between the output and input, according to equation 2.27, and determine a threshold that could properly separate outliers from non-outliers, as the first would presumably have higher reconstruction errors, as illustrated in figure 4.26.

Figure 4.26: AE MSE values of clean and anomalous samples.

The models were trained using Adam optimizer with a learning rate of 0.001, $\beta_1 = 0.9$, and $\beta_2 = 0.999$, corresponding to the provided default values on Keras.

Non-relevant activity filtering		Type of anomaly					
		Scheduled 10s	Gaussian 10s	Exp 10s	Every 60s	Exp 60s	Mixture of samples
Active	<i>Precision</i>	81.6±1.5	80.8±1.5	92.8±0.4	89.7±0.5	91.3±0.5	85.2±0.7
	<i>F1-Score</i>	88.3±0.9	87.1±0.7	83.9±1.6	87.0±0.9	88.4±0.9	89.9±1.2
Non-active	<i>Precision</i>	81.6±2.9	85.7±1.2	87.2±0.9	86.5±1.6	87.1±0.8	86.5±1.2
	<i>F1-Score</i>	86.6±1.3	90.7±0.8	91.6±0.8	89.4±2.1	90.5±1.1	91.5±0.9

Table 4.7: Work/lab automation scenario AE classification results (in percentage, with a 95% confidence interval).

When looking carefully at the results of using an AE to detect anomalies, presented in table 4.7, one can observe that it produces lower *F1-Scores* than classical models, primarily due to low *Precision* rates. Using figure 4.26 as a visual aid, AEs are able to detect the majority of anomalies - even though we do not display *Recall* values -, but they fail to classify some of the anomaly-free behavior, resulting in high false-positive rates, especially for samples displaying higher values of MSE.

Generative Adversarial Network
However, these results are satisfactory, as, even with a mixture of outlier samples, AEs still managed to break the 90% *F1-Score* barrier. Additionally, the results show that filtering unaccounted activity results in a minimal performance loss, especially in *F1-Scores*, and some *Precision* values increase when using it.

Variational Autoencoder

Secondly, we tested a Variational Autoencoder (VAE). Its goal is similar to a vanilla AE, but it also tries to learn a probability distribution of the input variables. The architecture is similar to the one depicted in the previous model, with the only significant difference being the latent-dimensional layer, which uses a λ function to input the latent distribution points into the decoder. The final architecture is illustrated in figure 4.27.

Figure 4.27: Variational Autoencoder (VAE) architecture.

The other notable difference is that, besides computing the loss function forcing the AE to match the initial samples, it also computes the KL-divergence between the learned latent distribution and the prior distribution. This function was already depicted in equation 2.28, in section 2.5.3. This model was also optimized by Adam, using the same parameters as the vanilla AE already presented.

Non-relevant activity filtering		Type of anomaly					
		Scheduled 10s	Gaussian 10s	Exp 10s	Every 60s	Exp 60s	Mixture of samples
Active	<i>Precision</i>	75.6±0.6	77.4±0.9	80.4±1.0	79.9±1.4	79.8±1.4	80.5±1.1
	<i>F1-Score</i>	85.2±0.5	85.5±0.8	86.5±1.0	87.6±0.6	87.9±0.9	87.0±1.0
Non-active	<i>Precision</i>	75.2±0.7	81.5±1.2	82.8±1.9	80.9±1.9	82.3±1.7	82.2±0.9
	<i>F1-Score</i>	84.8±0.7	88.2±0.8	89.2±0.5	87.5±1.0	87.7±0.9	88.2±0.4

Table 4.8: Work/lab automation scenario VAE classification results (in percentage, with a 95% confidence interval).

When comparing the results obtained by the vanilla AE and the VAE, it is possible to see a minimal deterioration in both the *Precision* and *F1-Score* rates. Furthermore, they do not show very significant fluctuation when using *non-relevant activity filtering*. Even though they are reasonable detection rates, the *Precision* rates are sub-optimal and flag almost a fifth of the training set as false positives.

Generative Adversarial Network

A Generative Adversarial Network (GAN) works by playing an adversarial game between two neural networks, the generator and the discriminator, so that both can improve their performance. The first does it by generating more realistic samples, to make more laborious the task of discerning between real and fake inputs for the discriminator, and the second by improving its classification performance, correctly labeling real data and samples outputted by the generator.

Figure 4.28: Generative Adversarial Network (GAN) architecture.

The final architecture of the generator and discriminator are both illustrated in figure 4.28. The discriminator is a simple Multi Layer Perceptron (MLP) with three hidden layers with Leaky ReLU activation (equation 2.26), and 68, 34, and 17 neurons respectively, and an output layer with a Sigmoid activation with a single neuron outputting a value between $[0; 1]$, where one is certainly an anomaly, and zero a clean sample. Its inputs are 136-dimensional vectors whose values come from the anomaly-free dataset and the generator. The latter also uses an MLP architecture, whose inputs correspond to pseudo-random noise vectors of five positions, with five fully connected Leaky ReLU layers, each with increasing number of neurons, followed by a `BatchNormalization` layer that normalizes the output with a momentum equal to 0.85.

The optimizer that produced the best results was Adam. In reality, there are two optimizers on a GAN architecture: one relative to the discriminator, and the other regarding the entire combined model. During training, the generator creates a batch of fake vectors that, alongside real data, are used to train the discriminator. Then, the discriminator weights are frozen, and

the generator is trained with random noise, some samples labeled as clean data, and others as anomalies. The procedure of training a GAN is depicted in algorithm 1.

Algorithm 1 Generative Adversarial Network (GAN) training procedure.

```

1: procedure FIT_GAN(model, X, num_epochs, batch_size, latent_dim)
2:   num_samples, num_inputs = X.dims
3:   steps_per_epoch = num_samples / batch_size
4:   ones = get_vector(size=batch_size, val=1)
5:   zeros = get_vector(size=batch_size, val=0)
6:   for each epoch in range(num_epochs) do
7:     for each step in range(steps_per_epoch) do
8:       X_batch = X[step*batch_size:(step+1)*batch_size, :]
9:       noise = get_random_noise(batch_size, latent_dim)
10:      X_gen = model.generator.predict(noise)
11:      model.discriminator.trainable = True // Train discriminator
12:      model.discriminator.train_on_batch(X_batch, ones)
13:      model.discriminator.train_on_batch(X_gen, zeros)
14:      model.discriminator.trainable = False
15:      noise = get_random_noise(batch_size, latent_dim) // Train GAN
16:      model.gan.train_on_batch(noise, ones)

```

Training an adversarial model is not a trivial task, as there is a need for a balance between models. In an optimal scenario, neither model should overfit, and both discriminator and adversarial losses should converge. However, according to our tests, we obtained the best results when the discriminator was slightly overfitted, i.e., the discriminator loss converged to values close to zero, and the generator loss diverged. With this in mind, the discriminator optimizer had a 0.0005 learning rate, and the adversarial model a 0.0001 learning rate. Given that the output of the GAN is a Sigmoid activation function, we used the binary cross-entropy loss, depicted in equation 2.29, as the metric Adam should optimize.

Non-relevant activity filtering		Type of anomaly					
		Scheduled 10s	Gaussian 10s	Exp 10s	Every 60s	Exp 60s	Mixture of samples
Active	<i>Precision</i>	94.2±3.3	63.0±2.0	98.3±0.7	98.6±0.6	95.7±1.2	97.6±1.5
	<i>F1-Score</i>	81.3±6.6	68.4±0.9	80.7±1.9	84.6±7.9	81.8±5.0	83.6±3.8
Non-active	<i>Precision</i>	97.6±1.4	98.0±0.9	97.0±1.0	98.5±0.7	95.4±4.4	97.6±0.7
	<i>F1-Score</i>	84.7±3.7	80.4±7.1	84.9±6.4	93.0±1.3	79.2±9.1	88.3±2.6

Table 4.9: Work/lab automation scenario GAN classification results (in percentage, with a 95% confidence interval).

GANs are not able to achieve *F1-Scores* as high as AE or VAE, mainly due to poor *Recall* values. Table 4.9 shows that, overall, GAN models have meager false-positive rates when compared to other neural network approaches. In any anomaly-detection scenario, it is great to have high *Precision* values since it lessens the need to have the administrator regularly dealing with false-positive alarms.

Except for the exponential distribution with 60 seconds mean outlier, GANs show a considerable performance boost when using the *non-relevant activity filtering*, ranging from about 3 to 12 percentage points.

Dual Autoencoders

The rationale behind this architecture is to improve the classification results of the vanilla AE by training a new Autoencoder (AE) using only clean samples that had a high MSE. We use the lowest MSE amongst anomalous training samples as the threshold that separates the “clean” samples that are going to feed the second AE, as exemplified in algorithm 2. All the parameters of the AEs are identical to those defined in section 4.2.4.

Algorithm 2 Dual Autoencoders (DualAEs) training procedure.

```

1: procedure FIT_DUALAE(model, X, num_epochs, batch_size)
2:   model.ae.fit(X, num_epochs, batch_size)
3:   mse['clean'] = compute_mse(X['clean'], model.ae.predict(X['clean']))
4:   mse['outlier'] = compute_mse(X['outlier'], model.ae.predict(X['outlier']))
5:   min_thr = min(mse['outlier'])
6:   X_filtered = X['clean'][mse['clean'] > min_thr, :]
7:   model.filtered_ae.fit(X_filtered, num_epochs, batch_size)

```

Non-relevant activity filtering		Type of anomaly					
		Scheduled 10s	Gaussian 10s	Exp 10s	Every 60s	Exp 60s	Mixture of samples
Active	<i>Precision</i>	71.3±2.3	74.9±1.8	75.6±3.1	80.5±2.3	76.0±2.4	76.2±1.8
	<i>F1-Score</i>	81.4±4.1	85.6±1.2	82.2±5.2	89.1±1.4	83.9±2.7	86.5±1.2
Non-active	<i>Precision</i>	74.6±3.1	80.7±1.0	81.0±1.2	79.8±2.1	80.9±0.8	79.0±0.7
	<i>F1-Score</i>	83.8±3.3	86.6±0.9	87.6±2.8	88.3±1.7	88.9±0.4	88.0±0.3

Table 4.10: Work/lab automation scenario DualAE classification results (in percentage, with a 95% confidence interval).

The results presented in table 4.10 highlight the fact that these models perform better when not using unaccounted activity filtering, except for the outlier uploading data every minute. In general, there is a slight loss in performance when compared to the pure vanilla AE. Each AE has sub-optimal *Precision* rates but, using them in sequence, may decrease those metrics even more.

AE-GAN Sequence Model

Although anomaly detection rates are not as good as the ones of the classical models attained, neural network approaches are not far behind. However, autoencoders presented low *Precision*, and the adversarial model was not able to detect all anomalies, even though it rarely produced false-positives.

This model stacks a GAN after an Autoencoder (AE), similar to the strategy used in the DualAE models, but this time it uses a GAN as the final classifier to improve *Precision* and,

consequently, the overall *F1-Score*. Thus, the minimum threshold is also determined by the MSE amongst anomalous training samples, and it uses training data from the anomaly-free dataset producing an MSE higher than the threshold to train the GAN. The architecture and hyperparameters used for any of these models were the same as the ones presented above.

Non-relevant activity filtering		Type of anomaly					
		Scheduled 10s	Gaussian 10s	Exp 10s	Every 60s	Exp 60s	Mixture of samples
Active	<i>Precision</i>	96.0±2.3	95.4±2.0	98.0±0.5	98.7±0.6	98.3±0.6	97.2±1.2
	<i>F1-Score</i>	94.6±1.4	83.9±2.9	95.7±1.7	96.4±0.5	96.9±1.2	94.3±0.7
Non-active	<i>Precision</i>	99.3±0.4	99.2±0.5	99.5±0.3	99.5±0.2	98.9±0.2	98.2±0.6
	<i>F1-Score</i>	96.6±0.9	88.4±2.3	97.3±0.4	97.0±1.0	97.1±0.5	94.9±0.8

Table 4.11: Work/lab automation scenario AE-GAN classification results (in percentage, with a 95% confidence interval).

Table 4.11 shows that there was a considerable positive jump in outlier behavior detection when compared to the other tested neural network approaches. The AE-GAN attained *F1-Scores* that were on par with the OC-SVM and GMM. When removing unaccounted activity, its results suffered a minor increase across all tests, a tendency that all neural network approaches showed, especially stacked models like this and the DualAE.

Neural Networks Results

Tables 4.12 and 4.13 summarize the classification results of the neural network approaches. As already mentioned, AE-GAN models outperform considerably any other neural network model, both when using or not *non-relevant activity filtering*. Additionally, there is a slight performance gain when keeping those small activities, something visible across all models, likely due to removing activities corresponding to TCP Keep-Alive and other packet exchanges that describe the usual network behavior.

On the other hand, the vanilla AE is also able to detect considerably well the majority of anomalous behavior; however, considering that the network administrator does not want to deal with false-positive red-flags regularly, a lower *Precision* rate penalizes this approach.

Overall, classical approaches seem to perform better than neural networks. However, AE-GAN models almost matched the detection rate of both the OC-SVM and GMM and fell shortly behind the KDE. However, the last two models have better performance to training-time ratio than the KDE, which is the classical model that takes the longest to train. Once again, it is essential to notice that the times showcased in table 4.14 are relative to tests performed on a Jupyter notebook, running on a desktop with an Intel Core i5-6500 CPU, 8GB of Random Access Memory (RAM), and a NVIDIA ¹⁰ GTX 1050TI GPU.

In comparison to the home automation scenario, IFs remain the fastest method to converge, despite its low classification performance. On average, classical models training times are lower than neural network approaches, except for KDE. However, as seen in the previous

¹⁰<https://www.nvidia.com/en-eu/geforce/>

Type of anomaly		WITHOUT non-relevant activity filter				
		AE	VAE	GAN	DualAE	AE-GAN
Scheduled 10s	<i>Precision</i>	81.6±2.9	75.2±0.7	86.9±0.5	74.6±3.1	99.3±0.4
	<i>F1-Score</i>	86.6±1.3	84.8±0.7	93.0±0.3	83.8±3.3	96.6±0.9
Gaussian 10s	<i>Precision</i>	85.7±1.2	81.5±1.2	86.8±0.6	80.7±1.0	99.2±0.5
	<i>F1-Score</i>	90.7±0.8	88.2±0.8	92.9±0.4	86.6±0.9	88.4±2.3
Exp 10s	<i>Precision</i>	87.2±0.9	82.8±0.7	87.2±0.5	81.0±1.2	99.5±0.3
	<i>F1-Score</i>	91.6±0.8	89.2±0.5	92.8±0.2	87.6±2.8	97.3±0.4
Scheduled 60s	<i>Precision</i>	86.5±1.6	80.9±1.9	83.6±0.7	79.8±2.1	99.5±0.2
	<i>F1-Score</i>	89.4±2.1	87.5±1.0	91.0±0.4	88.3±1.7	97.0±1.0
Exp 60s	<i>Precision</i>	87.1±0.8	82.3±1.7	88.4±0.4	80.9±0.8	98.9±0.2
	<i>F1-Score</i>	90.5±1.1	87.7±0.9	93.8±0.2	88.9±0.4	97.1±0.5
Mixture of samples	<i>Precision</i>	86.5±1.2	82.2±0.9	88.7±0.4	79.0±0.7	98.2±0.6
	<i>F1-Score</i>	91.5±0.9	88.2±0.4	94.0±0.2	88.0±0.3	94.9±0.8

Table 4.12: Work/lab scenario neural networks’ classification results (in percentage, with a 95% confidence interval), when not using *non-relevant activity filtering*.

Type of anomaly		WITH non-relevant activity filter				
		AE	VAE	GAN	DualAE	AE-GAN
Scheduled 10s	<i>Precision</i>	81.6±1.5	75.6±0.6	86.9±0.5	71.3±2.3	96.0±2.3
	<i>F1-Score</i>	88.3±0.9	85.2±0.5	93.0±0.3	81.4±4.1	94.6±1.4
Gaussian 10s	<i>Precision</i>	80.8±1.5	77.4±0.9	86.8±0.6	74.9±1.8	95.4±2.0
	<i>F1-Score</i>	87.1±0.7	85.5±0.8	92.9±0.4	85.6±1.2	83.9±2.9
Exp 10s	<i>Precision</i>	92.8±0.4	80.4±1.0	87.2±0.5	75.6±3.1	98.0±0.5
	<i>F1-Score</i>	83.9±1.6	86.5±1.0	92.8±0.2	82.2±5.2	95.7±1.7
Scheduled 60s	<i>Precision</i>	89.7±0.5	79.9±0.5	83.6±0.7	80.5±2.3	98.7±0.6
	<i>F1-Score</i>	87.0±0.9	87.6±0.6	91.0±0.4	89.1±1.4	96.4±0.5
Exp 60s	<i>Precision</i>	91.3±0.5	79.8±1.4	88.4±0.4	76.0±2.4	98.3±0.6
	<i>F1-Score</i>	88.4±0.9	87.9±0.9	93.8±0.2	83.9±2.7	96.9±1.2
Mixture of samples	<i>Precision</i>	85.2±0.7	80.5±1.1	88.7±0.4	76.2±1.8	97.2±1.2
	<i>F1-Score</i>	89.9±1.2	87.0±1.0	94.0±0.2	86.5±1.2	94.3±0.7

Table 4.13: Work/lab scenario neural networks’ classification results (in percentage, with a 95% confidence interval), when using *non-relevant activity filtering*.

scenario, KDE performance does not adequately scale when increasing the number of samples. On the other hand, neural networks tend to perform better when fed with a considerable amount of training examples.

Model	Training time	Training set size	Testing time	Test set size
OC-SVM	1min1s $\pm$ <1s	5885	<1s	2795 $\pm$ 65
IF	0min13s $\pm$ <1s	5885	<1s	2795 $\pm$ 65
LOF	3min9s $\pm$ 0min2s	5885	<1s	2795 $\pm$ 65
KDE	4min43s $\pm$ 0min5s	5885	0min1s $\pm$ <1s	2795 $\pm$ 65
GMM	0min27s $\pm$ 0min1s	5885	<1s	2795 $\pm$ 65
AE	6min58s $\pm$ 1min22s	5885	0min1s $\pm$ <1s	2793 $\pm$ 71
VAE	4min26s $\pm$ 0min32s	5885	<1s	2831 $\pm$ 73
GAN	14min6s $\pm$ 2min53s	5885	<1s	2776 $\pm$ 64
DualAE	3min17s $\pm$ 1min12s	5885	0min1s $\pm$ <1s	2811 $\pm$ 56
AE-GAN	4min13s $\pm$ 0min19s	5885	<1s	2802 $\pm$ 69

Table 4.14: Average (with a 95% confidence interval) training and testing durations in the work/lab scenario, corresponding to a $\tau = 1$ and $v = 5$.

4.3 Conclusion

In both the tested scenarios, a simplistic home automation example with a reduced number of devices, and a scenario where the number of background noise was considerably higher due to the higher number of nodes, the privacy-focused mechanisms proposed in chapter 3 fulfilled their goal of detecting periodic anomalous behavior. Additionally, depending on the use-case, removing unaccounted activity may deteriorate the performance of the classifiers. Even though models perform better without it in the first scenario, the second showed the exact opposite behavior. Therefore, it is advisable to test both strategies before settling on any of them.

Realistically, it is impossible to test all types of existant anomalies. This work only covered a few variations, more specifically, small-volume periodic behavior, simulating an attacker monitoring a network regularly. Anomalies with a larger volume of data exchange would be more easily detected, as the respective activity periods would be relatively long, and they would stand out amongst the characteristical regular low-volume exchanges of IoT networks, even if that behavior occurred less frequently. Obviously, if an attacker is aware of the periodic behavior of the network, it would be easier to masquerade the outlier behavior; yet, we are assuming that an external entity does not have that information. Additionally, the attacker may disguise the anomaly by mimicking the typical operation of a specific device, namely an Amazon Alexa, if he was aware that such a node existed in the network. Even if he managed to do it, he would only mimic the device bot-like behavior, not the human interactions that also describe its usual operation.

Moreover, these tests were conducted in very controlled environments. The home scenario occurred in a location with zero to none background noise or interference, and a reduced

number of devices connected to the network. The work environment tests were always conducted at the same period of the day - usually between 2 and 5 p.m. - and the clients connected to the network were always the same, despite slight variations on their behavior. Besides, the amount of data available was relatively low - about 6 hours worth of “clean” data and 2 to 3 of each anomaly type. In order to create a more reliable model, we would have to record a lot more training data, as well as more diverse. Ideally, it would contemplate different periods of the day or weekdays, and register a timestamp, or use a one-hot encoding of the moment of capture to make it possible for the models to separate the different behaviors along the day. Furthermore, the models were trained by modeling some specific “clean” behavior. If by any means, the typical network usage changed, they would have to be re-trained to compensate those changes.

In sum, it is possible to detect outlier behavior at the physical layer with a careful study of the typical network behavior. Still, if one wants to deploy these models into production, there has to be a compromise between training/classification spans and detection performance. Some models accurately model “clean” data, but take an excessive amount of time to do so (e.g., KDE, GAN). Other models, such as OC-SVMs or even the proposed AE-GAN, work considerably well and have reasonable training times. However, neural network approaches take a significant more amount of time to properly tune, as that process is not totally automated, as mentioned in this chapter, but scale better with a bigger amount of samples.

Framework for Live Anomaly Detection in IoT Networks

After conducting the tests showcased in chapter 4, one should deploy the models in a production environment. In this chapter, we propose a framework which contains the models previously trained and validated, like the ones in the previous chapter, and use them in production to detect anomalies as they occur. Furthermore, this framework is prepared to re-train the models periodically, so that they are able to adapt themselves as the network behavior changes.

We also conducted a series of performance tests to evaluate the scalability of the platform. In sum, our intuition is to know if it can handle the increase on the number of sniffers, and consequently, if it maintains the message throughput without losing a significant amount of performance.

5.1 Architecture Overview

The rationale behind the architecture depicted in figure 5.1 is to integrate into a production environment the models that are able to detect anomalies in data that was extracted and carefully explored before performing classification. Even though they work with “offline” data, they will only be useful once they are deployed in a real-world scenario. Additionally, if the platform administrator finds it adequate, the models can be re-trained with samples that the platform is periodically storing in the database.

The core of the framework is the Anomaly Detection Server (ADS), which is mostly responsible for classifying and storing windows of features read from the Network Windows Message Broker (NWMB). The latter receives data from multiple probes spread around the network that are initially recording IQ data, and ultimately transforming it into feature vectors so that the machine learning models stored in the ADS can classify them. The ADS

also contains the Feature Verification Application Programming Interface (API), which will be detailed further below, as well as the Resource Monitor (RM), that captures and offers an interface to visualize performance metrics of the server in real-time.

Figure 5.1: Overall live anomaly detection framework overview.

5.1.1 Probe Data Extractor (PDE)

A *probe* is an isolated device equipped with a suitable SDR. The Probe Data Extractor (PDE) must be capable of sniffing IQ data on a range of frequencies of interest, convert them to the frequency domain using the FFT, extract features that the administrator considers relevant, and send them to the NWMB. For instance, if the goal is to monitor a Wi-Fi network, then we would place a probe close to every AP of the network and record data exchanges in the same channel of the associated AP. Additionally, every probe would have a matching Classification Manager (CM) instance in the ADS, responsible for the classification of feature windows and model re-training.

The PDE is composed of three founding blocks: the sniffer, the FFT processor, and the feature extractor. They work as three independent threads, communicating via monitors that contain buffers to share data between them. Given that the probe may be a low-power device, the PDE was developed in C¹ to minimize the performance overhead of other higher-level programming languages. The PDE resorts to the FFTW² subroutines to compute the FFT, and `librdkafka`³, a C/C++ Kafka client, to send messages to the NWMB. Since we tested the anomaly detection mechanisms presented in chapter 3 using an HackRF One, we used a modified version of their `hackrf-tools`⁴ to read data from the sniffer. The pseudo-code of the PDE is split across algorithms 3 and 4.

Algorithm 3 Probe Data Extractor (PDE) *modus operandi*, part 1.

```

1: procedure PDE(radio_configs, writes_per_buffer, fft_size, freqs_interest, broker_topic)
2:   qam_buffer = get_mutex_buffer()
3:   timeline_buffer = get_mutex_buffer()
4:   config_radio(radio_configs)
5:   config_radio_callback(sniffer_callback, qam_buffer, writes_per_buffer, fft_size)
6:   start_thread(compute_fft, qam_buffer, timeline_buffer, fft_size, activity_thr,
   freqs_interest)
7:   start_thread(compute_features, timeline_buffer, freqs_interest, broker_topic)
8:   while radio_is_streaming() do
9:     sleep(1)
10: procedure SNIFFER_CALLBACK(sniffer_data, qam_buffer, writes_per_buffer, fft_size)
11:   bytes_read = sniffer_data.length
12:   bytes_interval = bytes_read / writes_per_buffer
13:   for each w in range(writes_per_buffer) do
14:     lock(qam_buffer.mutex)
15:     while qam_buffer.length == MAX_SIZE do
16:       wait(qam_buffer.mutex)
17:     write_idx = qam_buffer.producer_idx * fft_size * 2
18:     qam_buffer.buffer[write_idx] = sniffer_data.buffer[:fft_size * 2]
19:     qam_data.producer_idx = (qam_buffer.producer_idx + 1) % MAX_SIZE
20:     signal(qam_buffer.can_consume)
21:     unlock(qam_buffer.mutex)

```

¹<http://www.open-std.org/jtc1/sc22/wg14/>

²<http://fftw.org/>

³<https://github.com/edenhill/librdkafka>

⁴<https://github.com/mossmann/hackrf/tree/master/host>

Algorithm 4 Probe Data Extractor (PDE) *modus operandi*, part 2.

```
22: procedure COMPUTE_FFT(qam_buffer, timeline_buffer, fft_size, activity_thr,
    freqs_interest)
23:   num_freqs = length(freqs_interest)
24:   while 1 do
25:     lock(qam_buffer.mutex)
26:     while qam_buffer.length == 0 do
27:       wait(qam_buffer.mutex)
28:     read_idx = qam_buffer.consumer_idx * fft_size * 2
29:     qam_data = qam_buffer.buffer[read_idx:read_idx + fft_size * 2]
30:     qam_data.consumer_idx = (qam_buffer.consumer_idx + 1) % MAX_SIZE
31:     signal(qam_buffer.can_produce)
32:     unlock(qam_buffer.mutex)
33:     fft_data = compute_logpower_fft(qam_data)
34:     timeline_data = convert_to_timeline(fft_data, activity_thr, freqs_interest)
35:     lock(timeline_buffer.mutex)
36:     while timeline_buffer.length == MAX_SIZE do
37:       wait(timeline_buffer.mutex)
38:     write_idx = timeline_buffer.producer_idx * num_freqs
39:     timeline_buffer.buffer[write_idx] = timeline_data
40:     timeline_data.producer_idx = (timeline_buffer.producer_idx + 1) % MAX_SIZE
41:     signal(timeline_buffer.can_consume)
42:     unlock(timeline_buffer.mutex)
43: procedure COMPUTE_FEATURES(timeline_buffer, freqs_interest, broker_topic)
44:   num_freqs = length(freqs_interest)
45:   while 1 do
46:     lock(timeline_buffer.mutex)
47:     while timeline_buffer.length == 0 do
48:       wait(timeline_buffer.mutex)
49:     read_idx = timeline_buffer.consumer_idx * num_freqs
50:     timeline_data = timeline_buffer.buffer[read_idx:read_idx + num_freqs]
51:     timeline_buffer.consumer_idx = (timeline_buffer.consumer_idx + 1) %
MAX_SIZE
52:     signal(timeline_buffer.can_produce)
53:     unlock(timeline_buffer.mutex)
54:     features_data = []
55:     for each f in freqs_interest do
56:       features_data += [extract_features(timeline_data[f])]
57:     while 1 do
58:       msg_sent = send_to_broker(broker_topic, features_data)
59:       if msg_sent then
60:         break
```

5.1.2 Network Windows Message Broker (NWMB)

The Network Windows Message Broker (NWMB) is, in its essence, an instance of Apache Kafka ⁵, a publish-subscribe streaming platform, able to process and store, in a fault-tolerant way, streams of records as they occur. It is widely popular in multiple applications, especially when a reliable, low-latency data pipeline is needed. Kafka can run in a cluster of servers, and it works by queuing messages containing a key, a value, and a timestamp in streams of records, known as *topics*. Opposite to the majority of messaging systems, Kafka is designed around a distributed commit log, where data are not immediately flushed to disk, allowing high throughput and very efficient Input/Output (I/O) operations [90].

We also considered other messaging platforms, such as RabbitMQ ⁶, a lightweight and easy to deploy message broker. However, it does not support throughputs as high as Kafka. Moreover, RabbitMQ was designed to be vertically scaled since its performance depends on the CPU processing power, and it is harder to scale horizontally, as it forces the developer to configure the cluster, rather than managing it automatically. On the other hand, Kafka relies on Apache Zookeeper ⁷ to manage the distributed cluster. Additionally, in RabbitMQ, if a given queue that is not mirrored is running on a node that shuts down, that queue becomes unavailable. Kafka will deviate all the traffic to another node on the cluster due to its inherent distributed commit log architecture [90].

In the context of this platform, the PDEs will work as producers, and stream data to a specific *topic* identified by the unique name or numerical identifier of the PDE. Each message value will be an array of `double`, containing the feature set, and two timestamps, the first corresponding to the timestamp of when the sniffer thread at the PDE recorded the data, and the second corresponding to the moment before sending the message to the broker.

Furthermore, using a message broker instead of plain TCP socket allows us to use load balancing of CMs. For instance, we would have two instances of CM on different machines consuming data from the same *topic*, which corresponds to the same PDE.

5.1.3 Anomaly Detection Server (ADS)

The Anomaly Detection Server (ADS) is formed by several instances of CMs, and time-series persistence database that will store all the relevant data regarding windows classification. Each Classification Manager (CM) is responsible for handling the message reading from the NWMB, retrieving a collection of features and timestamps stored in the associated *topic* of that node or probe. In reality, for any available probe, a model, or multiple models if there are multiple ranges of frequencies of interest, they should be previously trained and made available for the CM instance, so that, after retrieving a given window of features, it can adequately classify it. Then, it will save those features, the label given by the chosen model, and the anomaly score (e.g., the MSE) in the database.

⁵<https://kafka.apache.org/>

⁶<https://www.rabbitmq.com/>

⁷<https://zookeeper.apache.org/>

Despite not being appropriate to use in a production system due to the overhead of being an interpreted programming language, we developed the ADS in Python 3 ⁸. Our choice was made considering the already existing code-base for the mechanisms presented in chapter 3, and the fact that all models were built either using Scikit-learn ⁹ or Keras ¹⁰. Even though Keras runs on top of Tensorflow ¹¹, which essentially creates a Python wrapper around C or Compute Unified Device Architecture (CUDA) ¹² routines, and there are frameworks on more efficient programming languages that can load them, the majority of Scikit-learn models are written purely in Python.

The Classification and Features database was implemented using Timescale ¹³, an open-source database based on PostgreSQL ¹⁴, that provides fast and efficient time-series storing, with the convenience of the Structured Query Language (SQL) language. It is capable of ingesting millions of data points per second and extends the SQL language with unique queries based on the temporal nature of data. Each CM will store the retrieved statistics in the associated table in the database; thus, isolating data from every probe.

The CM is also responsible for, on a given schedule, re-training the stored models using the data saved in the database. In order to avoid overloading the server resources, the training schedule of each CM must be carefully thought, as one should avoid overlapping different instances re-trainings. Certainly, if there are enough resources, this should not pose a problem. Typically, a modern desktop computer is not able to handle multiple machine learning processes at the same time, as they represent resource-intensive operations. Otherwise, if the number of nodes is high enough to avoid it, the administrator of the platform should think about vertically scaling the server, e.g., add more CPU cores or increase the memory capacity, or offloading some instances to another server, i.e., perform horizontal scaling. Additionally, to avoid storage limitations, data already used for training purposes (only valid for neural networks), or non-validated, is deleted from the database. Finally, the obtained classification results are saved into a separate table, so that the administrator may consult them later if necessary.

Sometimes, for several reasons, anomalous behavior can appear in the network, and the system classifies it as if it was regular traffic. This could be a potential risk since the models would be learning this behavior as being anomaly-free. Consequently, each entry in the database has a verification label, whose goal is to force administrators to review the data before its use in re-training.

On a separate note, even though we denote the ADS as a *server*, it is not aimed to be deployed on a single machine. Its modular architecture makes it possible to horizontally scale it by spreading CMs across different physical hosts. One way to do it is to deploy these

⁸<https://www.python.org/>

⁹<https://scikit-learn.org>

¹⁰<https://keras.io/>

¹¹<https://www.tensorflow.org/>

¹²<https://developer.nvidia.com/cuda-zone>

¹³<https://www.timescale.com/>

¹⁴<https://www.postgresql.org/>

components as Docker containers, just like we designed them, on a Docker Swarm ¹⁵ cluster.

5.1.4 Feature Verification API

The Feature Verification API works mainly as a communication interface between the administrator and the Classification and Features database. Its goal is to provide a way for the administrator to verify the captured data features and labels before re-training the models and validate them accordingly.

For instance, the model in production is not able to detect an anomaly present in the network; then, any captured window of features would be saved in the database as a clean entry, even though it is mislabeled. If the network administrator is aware of this outlier behavior, we would not want to use that as anomaly-free data to re-train the classifier. The Feature Verification API provides an interface to validate which entries are used for training, and which are discarded.

The REST API was developed using Flask ¹⁶, a lightweight Web Server Gateway Interface (WSGI) framework, and provides the following endpoints:

- `/entries/<node_id>/<int:hours>/`: this endpoint allows `GET`, and `DELETE` Hypertext Transfer Protocol (HTTP) methods for retrieving or deleting entries in the `node_id` table - corresponding to a probe identifier - over the last `hours` hours.
- `/entries/<node_id>/`: this endpoint only allows the `PUT` method, and it accepts a JavaScript Object Notation (JSON) body containing a list of tuples (`timestamp`, `validated`) that will be used for updating the validation status of the entries on the `node_id` table.
- `/results/<node_id>/`: after each training process, the CM stores the classification results of each model into the `results` table. This endpoint provides an interface for reading and deleting data from that particular table (`GET` and `DELETE` methods), given a particular `node_id`.

5.1.5 Resource Monitor

The Resource Monitor (RM) is a component whose goal is to monitor the status of the ADS, especially regarding resource usage. Assuming that the CM instances are deployed as Docker ¹⁷ containers, it is vital to monitor the total CPU and memory consumption. We use cAdvisor ¹⁸, an open-source tool developed by Google, to access the resource usage of running containers, and export it to Prometheus ¹⁹, an open-source bundle of tools for system monitoring and alerting. It provides efficient storage of time-series metrics data, and it is compatible with many data sources, such as cAdvisor, besides an excellent integration with Grafana ²⁰, a dashboard for visualizing and monitoring the registered metrics. Additionally,

¹⁵<https://docs.docker.com/engine/swarm/>

¹⁶<https://flask.palletsprojects.com/en/1.1.x/>

¹⁷<https://www.docker.com/>

¹⁸<https://github.com/google/cadvisor>

¹⁹<https://prometheus.io/>

²⁰<https://grafana.com/>

we also used Node exporter ²¹, a tool developed by the Prometheus team to monitor the Docker host hardware.

5.2 Scalability Tests

To make sure that the platform can handle labeling data from multiple network probes simultaneously, we perform some scalability tests. Since we did not have multiple HackRF Ones in our possession, we emulated multiple nodes by developing a simple script that sent an array of features every three seconds. According to our tests, it takes less than one second to read data from the sniffer buffer, process it, and extract features (for instance, we extracted features from 15-minutes window in under 400 milliseconds on a desktop with an Intel Core i5-6500 and 8GB of RAM); thus, assuming the use of a sliding distance of two seconds, and an additional second for data processing, it totals about three seconds.

On our tests, the ADS, including the multiple Classification Manager (CM) instances, the Feature Validation API, the database, and the Resource Monitor (RM), as well as the NWMB were running on the same desktop with an Intel Core i5-6500 3.20 GHz CPU, and 8GB of RAM, all deployed as Docker containers. We extracted messaging times, corresponding to how long it took for a message to be sent by a PDE and read by a CM, and PDE processing times, i.e., the amount of time since the PDE read a feature-window and saved the classification results into the Classification and Features database, for variable number of nodes/probes for about ten minutes. These emulated probes were running on a separate Linux machine connected to the same local network, just reading features from a file and sending them to the broker every three seconds.

Figure 5.2 highlights the differences between having one or twenty probe nodes sending data to the broker, while their corresponding CMs are consuming and processing it. Even though the time frame is relatively short, increasing the number of nodes does not seem to affect the linearity of the average messaging time, as it seems to be constant throughout the timeline. However, having more nodes provokes much more fluctuation in those values, sometimes up to six times higher.

Once more, increasing the number of nodes does not seem to affect processing times at the CM instances. Identically to messaging times, there is a noticeable variation in processing times when increasing the number of nodes, but without affecting their linearity. Those values are depicted in figure 5.3.

Table 5.1 showcases the minimum, maximum, and average and 95% confidence interval values of messaging and CM processing times, for multiple numbers of probes working simultaneously. Additionally, we tested those same metrics while re-training an AE model concurrently in one of the CM instances.

Regarding messaging times without re-training models, although minimum values tend to be flat with a variable number of nodes, maximum values increase as we add more probes to the cluster - almost an eightfold jump when adding nineteen nodes -, as well as the average,

²¹https://github.com/prometheus/node_exporter

Figure 5.2: Comparison of messaging times (difference between sent and received message timestamps), when there are one and twenty consuming nodes.

Figure 5.3: Comparison of feature window processing times at the CMs, when emulating one node and twenty nodes.

			1 node	2 nodes	5 nodes	10 nodes	20 nodes
Consuming	Messaging time	Minimum	0.002	0.002	0.002	0.002	0.002
		Maximum	0.008	0.009	0.015	0.031	0.067
		Mean	0.003±0.0001	0.004±0.0001	0.005±0.0001	0.008±0.002	0.017±0.0004
	CM processing time	Minimum	0.004	0.003	0.002	0.002	0.002
		Maximum	0.012	0.014	0.014	0.025	0.044
		Mean	0.007±0.0001	0.007±0.0001	0.006±0.0001	0.007±0.0002	0.012±0.0002
Consuming and Training	Messaging time	Minimum	0.002	0.001	0.001	0.003	0.001
		Maximum	51.102	49.883	52.396	98.880	55.379
		Mean	6.381±1.706	2.945±0.862	1.305±0.393	1.308±0.403	0.377±0.107
	CM processing time	Minimum	0.004	0.003	0.002	0.002	0.002
		Maximum	7.154	8.411	8.169	7.801	7.494
		Mean	0.142±0.105	0.084±0.0002	0.032±0.022	0.019±0.011	0.014±0.006

Table 5.1: Comparison of average (with a 95% confidence interval), minimum and maximum messaging and CM processing times (in seconds) for multiple number of nodes.

that sees a sixfold increase when jumping from one to twenty nodes. When taking into consideration the absolute values, these differences are almost negligible since probes are sending data only three seconds at a time. On the other hand, when the CMs consumed data from the broker while a model was re-training in one of the instances, there is a high increase in the registered maximum and mean values. They averaged about 55 seconds - except for the peak when using 10 probes -, while the mean messaging times decreased as the number of

probes increased. Given that the primary source of these delays comes from the CM container performing model re-training, adding more nodes decreases the overall mean since we have more nodes with lower latencies. Still, considering the absolute values, there is a significant loss in performance.

In both scenarios, when training the AE or not, the evolution of the CM processing times shows the same behavior as the one produced by messaging times. However, there is not such a severe performance loss. When adding 19 extra nodes, the maximum processing time only increases three times, while the average only sees a 60% increase. When training simultaneously, maximum processing times average around 8 seconds, and the mean time for 20 nodes is only 17% bigger than when only consuming data from the broker.

		1 node	2 nodes	5 nodes	10 nodes	20 nodes
CPU Usage	<i>Idle</i>	2.83%	3.70%	4.63%	6.62%	10.71%
	<i>Consuming</i>	3.27%	3.40%	4.48%	7.02%	11.91%
	<i>Consuming/Training</i>	94.37%	94.34%	93.58%	95.20%	95.61%
Memory Usage	<i>Idle</i>	17.09%	20.24%	29.30%	45.45%	77.46%
	<i>Consuming</i>	17.89%	20.75%	30.52%	46.39%	76.91%
	<i>Consuming/Training</i>	43.92%	48.18%	55.44%	72.02%	93.62%

Table 5.2: Comparison of Docker host resource usage, for multiple number of nodes.

Table 5.2 showcases the resource usage at the host machine, also when there is a variation in the number of PDEs. When idle, the CPU usage sees a 5 times increase as the number of nodes increases 20 times, and memory usage increases 6 times, likely due to having to fit more models in memory to perform classification. However, different models are more memory-taxing than others, so it ultimately depends on the type of model or the number of models being loaded to memory per CM.

The expenditure of host resources when consuming data from the broker is very similar to when the CMs are idle. However, the CPU usage increases significantly when training a single model in a separate process, in this case, an AE with approximately 3,000 training samples; thus, we recommended either using a powerful machine capable of running multiple parallel processes, or re-training the models in separate intervals.

Additionally, memory usage takes an extra toll when a model is being re-trained, as it has to fit two of its instances simultaneously - one which is consuming data from the broker, and the one that is being re-trained. Consuming data from 20 emulated probes while training a model almost saturates the memory of the host used for testing, as it reached 93%. Nevertheless, these metrics show that either the machine hosting the ADS is equipped with a considerable amount of RAM, or CM instances should be split across a cluster of ADSs.

Finally, it is important to view these values not as absolute ground truth, but as comparative metrics. Our goal is to show that, even though it is possible to run the ADS on a single machine for a network containing 20 probes, fitting all the corresponding models on a network with a higher number of nodes will only be possible with the appropriate hardware resources.

5.3 Conclusion

The goal of the framework here presented was to make sure that the mechanisms proposed in chapter 3 can be used in a production environment. Despite the initial long process of training the base models, migrate them into the platform is as easy as firing a new Docker container up, and setting a new PDE at an appropriate location.

When carefully analyzing the results obtained in section 5.2, it is possible to see that, on average, the platform does not suffer a significant performance loss when adding more PDEs. However, especially maximum values showcase a higher variance, even if these changes are almost negligible. On the other hand, while re-training models in the background, these variations are far more noticeable, and can impact the classification throughput of the platform.

Undoubtedly, the testing timeframe is small; yet, it still manages to capture the essential behavior of the server. Moreover, the NWMB and ADS were running on the same Docker host, which may have deteriorated the overall performance, which is especially noticeable when re-training the models at the same time, CM instances are consuming and classifying feature windows. Ideally, they would run on different machines, to reduce the performance overhead caused by the broker, and nodes re-training their stored models. Additionally, when using a considerable amount of CMs, splitting them across a cluster of servers will also increase performance, notably by reducing the overhead caused by model re-training.

It is essential to notice that it is not recommendable to use Python as the core language in the proposed framework, as it reduces the general system performance considerably, and increases the consumption of memory resources. However, besides allowing to reuse the already developed code-base, this framework was thought of as proof-of-concept; thus, the overall results may be considered satisfactory.

Moreover, since we are performing physical-layer data analysis, with a focus on user privacy, detecting which type of anomaly is present in the network or which node is tampered is a challenging task and is out of the scope of this work. We aim to alert the administrator that there is an out-of-normal behavior on the network, but will not act upon that alert.

Conclusions and Future Work

The research conducted in chapter 2 showed that IoT and other wireless mesh networks are vulnerable to a variety of attacks, as their heterogeneity and low-powered CPUs force manufacturers to overlook potential liabilities and implement trivial security mechanisms. To overcome these issues, administrators were forced to use monitoring strategies to keep networks secure, even if at the same time they are disrespecting users' privacy as they are able to correlate network patterns to an individual node, according to their MAC or IP address. Moreover, those performing OSI layer 1 monitoring often detect anomalies by only correlating the signal power to the relative distance between nodes. On the other hand, the proposed anomaly detection mechanisms focus on physical layer data, where such task becomes more complex as the signal mixes multiple devices data exchanges.

The mentioned mechanisms rely on analyzing and extracting statistical and time-frequency features that characterize radio silence and activity periods. Moreover, the goal is to only model the network when it is free of anomalies, so that the trained classifier is not bound to any specific type of outlier behavior. Another issue of current anomaly detection systems in IoT networks is that most rely on supervised learning techniques, thus training a classifier to specific types of attack vectors. With this in mind, we proposed a series of feature engineering procedures to optimize the classification performance of OCC models. Our choice for one-class classifiers is justified by only having to train them with “clean” data, therefore, avoiding biasing them towards some determined anomalous behavior. This also made it possible to create a more robust model against unseen phenomena. Nonetheless, to conduct proper research, one needs datasets containing anomalous behavior too, either to perform feature engineering or to optimize the performance of the models. We proposed using multiple different anomalous samples to make the models as broad as possible. In an ideal scenario, one would have access to all existent anomalies; yet, that is not a feasible approach.

Furthermore, as we suggested the use of SDRs as the primary tool to record IQ data, these mechanisms are not bound to any specific radio technology, and they can be used with Wi-Fi,

Bluetooth, or any other that works on the frequency range supported by the provided sniffer.

To validate those same mechanisms to outlier detection at the physical layer, we designed two IoT network scenarios, where the goal was to detect periodic outlier behavior in a Wi-Fi channel. The first worked as a proof-of-concept scene since, besides the low number of connected devices, it was conducted in an environment isolated from background noise and interferences. The second was conducted on an environment with a significant amount of nodes generating background noise and activity on the channel, thus making it harder for the classifiers to identify outliers.

In the first scenario, the OCC classical models were able to detect the anomalous behavior with ease. However, even in the more noisy environment as the one depicted in the second scenario, both classical and neural network approaches tested were able to produce very satisfactory results. Some models were able to obtain a 99% detection rate, even if at the cost of higher training and evaluation times, as it was the case of the KDE. On the other hand, classical models such as OC-SVMs and GMMs were able to grasp 97 to 98% *F1-Scores* while maintaining reasonable training spans. Neural network approaches did not perform as well as classical ones did; nevertheless, the Autoencoder-Generative Adversarial Network (AE-GAN) model - an approach stacking a GAN after an AE to improve the *Precision* rate of the latter -, was able to obtain similar results to the OC-SVM. Besides the higher memory usage and longer training times, they require a more careful configuration, as we did not resort to auto machine learning techniques (i.e., automatically testing millions of combinations of number of layers, number of neurons, and other hyperparameters) due to time and hardware resources constraints. One positive note about the developed models is that the majority also presented satisfactory *Precision* rates, which decreases the number of false-positive alerts.

These tests showed the validity of the mechanisms proposed in chapter 3, even if it is realistically impossible to test all types of existing anomalies. 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. Higher-volume data exchanges would produce longer activity periods; thus, they would be more easily detected. Moreover, we had a small amount of data for the last scenario - about 6 hours of anomaly-free traces and 2 to 3 hours of traces with every type of anomalous behavior that was tested -, and the tests were always conducted with the same devices, with similar behaviors every time, at the same periods of the day. This makes it harder for the model to generalize to other periods of the day where there would be fewer connected devices, or even differences in behavior between weekdays and weekends.

Nevertheless, IoT devices are very heterogeneous, and their behavior on the network can take many forms. Our tests were conducted using an Amazon Echo with regular interactions and Spotify streaming. Some of its operations were periodic, which also made it easier for the models to learn how the network behaves correctly.

Lastly, we designed a framework to deploy the trained machine learning models into a production environment. We proposed a full data pipeline - multiple probes, also known as PDEs, that are split across the network recording IQ data, transforming it into windows

of relevant features that are sent to the broker; they are then proxied to the ADS, running multiple instances of CMs, which are responsible for handling the classification of an individual PDE data stream. The framework was designed to be scalable, modular, and able to handle re-training of the models. Furthermore, adding a new node to the network must be as simple as adding a new sniffer, and starting a new Docker container with the corresponding model at the ADS.

After the performance tests, designed to validate the scalability of the mentioned framework, we concluded that due to memory constraints - as we add more CMs to the ADS pool, the number of models stored in memory also increases -, the ADS should be distributed across different physical machines. Furthermore, when re-training a given model, there is a considerable increase in the usage of host resources, especially CPU-wise. However, as we designed its components to be deployed as Docker containers, it is easily horizontally scaled on a Docker Swarm cluster.

6.1 Future Work

Although this Dissertation covered the base approach to allow the detection of anomalies at the physical layer, there is still a few points where it can be improved:

- Extend the anomaly detection mechanisms proposed in chapter 3 to allow the differentiation of multiple periods of the day or week. For instance, a series of different models can be used to classify the data according to the weekday. Another alternative is to add a one-hot encoded timestamp to the feature set (e.g., seven binary fields corresponding to the day of the week), thus “informing” the model about the date of capture.
- In our test scenarios, only short-length and high-frequency anomalies were considered. It is impossible to test all imaginable variations of anomalous behavior, especially since it depends on the regular operation of the network. Nevertheless, having a longer list of anomalies, particularly long-duration and low-frequency ones (e.g., uploading a 1GB video every ten minutes) increases the chances of creating a model that is properly tuned.
- Our tests were conducted with essentially two IoT devices: an Amazon Echo and a Raspberry Pi. It would be interesting to study how other devices work, and what is their impact on the spectrum usage. It is challenging for the classifiers to understand and model some behaviors correctly.
- Even though we tested combinations of models, namely DualAE and AE-GAN, the second model was only trained with subsets of the overall training data. A similar approach, yet often successful, is to create ensemble solutions that work by combining the classification results of multiple independent models into a single one. For instance, we could have combined two approaches with not so great *F1-Scores*, but that were fast to converge into an ensemble model that was fast and more accurate than any of its individual components.

- Regarding the developed framework for deploying models into production environments, there are some issues that could be improved. First, the core of the ADS, the CMs, should be migrated into a faster and more efficient programming language, since Python makes it harder to control memory usage and concurrency between processes. Furthermore, one could develop a web interface to manage the ADS containers considering, at the moment, that launching or deleting them is done manually. Finally, the PDE was developed on top of HackRF One libraries. To make it compatible with other SDR devices, one should modify the sniffer callback to support the different data types outputted by different sniffers.

References

- [1] K. Ashton, *That 'Internet of Things' Thing*, Jun. 2009. [Online]. Available: <https://www.rfidjournal.com/articles/view?4986> (visited on 10/07/2018).
- [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*, IET, 2018, pp. 1–4.
- [3] L. Columbus, *10 Charts That Will Challenge Your Perspective Of IoT's Growth*, Jun. 2018. [Online]. Available: <https://www.forbes.com/sites/louiscolumbus/2018/06/06/10-charts-that-will-challenge-your-perspective-of-iots-growth/#132436083ecc> (visited on 10/01/2018).
- [4] K. Zhao and L. Ge, "A survey on the internet of things security", in *2013 Ninth international conference on computational intelligence and security*, IEEE, 2013, pp. 663–667.
- [5] M. A. Khan and K. Salah, "Iot security: Review, blockchain solutions, and open challenges", *Future Generation Computer Systems*, vol. 82, pp. 395–411, 2018.
- [6] F. Y. Yavuz, "Deep learning in cyber security for internet of things", PhD thesis, 2018. [Online]. Available: <http://hdl.handle.net/11498/51652> (visited on 04/28/2019).
- [7] M. F. Muhammad, W. Anjum, and K. S. Mazhar, "A critical analysis on the security concerns of internet of things (iot)", *International Journal of Computer Applications (0975 8887)*, vol. 111, no. 7, 2015.
- [8] M. A. Al-Garadi, A. Mohamed, A. Al-Ali, X. Du, and M. Guizani, "A survey of machine and deep learning methods for internet of things (iot) security", *arXiv preprint arXiv:1807.11023*, 2018.
- [9] J. Wallen, *Five nightmarish attacks that show the risks of IoT security*, Jun. 2017. [Online]. Available: <https://www.zdnet.com/article/5-nightmarish-attacks-that-show-the-risks-of-iot-security/> (visited on 01/20/2018).
- [10] J. du Toit, *Active versus Passive Network Monitoring: An infographic guide, revisited*, Sep. 2016. [Online]. Available: <https://www.irisns.com/active-versus-passive-network-monitoring-an-infographic-guide-revisited/> (visited on 01/28/2019).
- [11] B. B. Lowekamp, "Combining active and passive network measurements to build scalable monitoring systems on the grid", *ACM SIGMETRICS Performance Evaluation Review*, vol. 30, no. 4, pp. 19–26, 2003.
- [12] C. Chaudet, E. Fleury, I. G. Lassous, H. Rivano, and M.-E. Voge, "Optimal positioning of active and passive monitoring devices", in *Proceedings of the 2005 ACM conference on Emerging network experiment and technology*, ACM, 2005, pp. 71–82.
- [13] 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.
- [14] M. Collins, *Network Security Through Data Analysis: From Data to Action*. " O'Reilly Media, Inc.", 2017.
- [15] S. Axelsson, "Intrusion detection systems: A survey and taxonomy", Technical report, Tech. Rep., 2000. [Online]. Available: http://neuro.bstu.by/ai/To-dom/My_research/Paper-0-again/For-research/D-mining/Anomaly-D/Intrusion-detection/taxonomy.pdf (visited on 04/20/2019).

- [16] H. Debar, M. Dacier, and A. Wespi, “Towards a taxonomy of intrusion-detection systems”, *Computer Networks*, vol. 31, no. 8, pp. 805–822, 1999.
- [17] F. Sabahi and A. Movaghar, “Intrusion detection: A survey”, in *2008 Third International Conference on Systems and Networks Communications*, IEEE, 2008, pp. 23–26.
- [18] 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.
- [19] S. Raza, L. Wallgren, and T. Voigt, “Svelte: Real-time intrusion detection in the internet of things”, *Ad hoc networks*, vol. 11, no. 8, pp. 2661–2674, 2013.
- [20] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, A. Shabtai, D. Breitenbacher, and Y. Elovici, “N-baiot—network-based detection of iot botnet attacks using deep autoencoders”, *IEEE Pervasive Computing*, vol. 17, no. 3, pp. 12–22, 2018.
- [21] R. B. Randall, “Spectral analysis and correlation”, in *Handbook of signal processing in acoustics*, Springer, 2008, pp. 33–52.
- [22] J. Semmlow, “The fourier transform and power spectrum”, in *Signals, and Systems for Bioengineers*, Academic, 2012, pp. 253–268.
- [23] 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)*, IEEE, 2014, pp. 175–182.
- [24] N. Sehatbakhsh, M. Alam, A. Nazari, A. Zajic, and M. Prvulovic, “Syndrome: Spectral analysis for anomaly detection on medical iot and embedded devices”, in *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, IEEE, 2018, pp. 1–8.
- [25] 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)*, IEEE, 2018, pp. 1–9.
- [26] Q. Feng, Y. Zhang, C. Li, Z. Dou, and J. Wang, “Anomaly detection of spectrum in wireless communication via deep auto-encoders”, *The Journal of Supercomputing*, vol. 73, no. 7, pp. 3161–3178, 2017.
- [27] I. 8. W. Group *et al.*, “Ieee standard for information technology–telecommunications and information exchange between systems–local and metropolitan area networks–specific requirements–part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications amendment 6: Wireless access in vehicular environments”, *IEEE Std*, vol. 802, no. 11, 2010.
- [28] *Wi-Fi Channels, Frequency Bands & Bandwidth | Electronics Notes*. [Online]. Available: <https://www.electronics-notes.com/articles/connectivity/wifi-ieee-802-11/channels-frequencies-bands-bandwidth.php> (visited on 02/04/2019).
- [29] J.-J. DeLisle, *What’s the Difference Between IEEE 802.11af and 802.11ah?*, Apr. 2015. [Online]. Available: <https://www.mwrf.com/active-components/what-s-difference-between-ieee-80211af-and-80211ah> (visited on 02/04/2019).
- [30] R. Parada, J. Meliá-Seguí, and R. Pous, “Anomaly detection using rfid-based information management in an iot context”, *Journal of Organizational and End User Computing (JOEUC)*, vol. 30, no. 3, pp. 1–23, 2018.
- [31] 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*, IEEE, 2007, pp. 2684–2687.
- [32] 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*, 2007, pp. 105–116.
- [33] 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*, IEEE, 2007, pp. 2783–2786.

- [34] 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*, ACM, 2006, pp. 191–204.
- [35] Y. Sheng, K. Tan, G. Chen, D. Kotz, and A. Campbell, "Detecting 802.11 mac layer spoofing using received signal strength", in *IEEE INFOCOM 2008-The 27th Conference on Computer Communications*, IEEE, 2008, pp. 1768–1776.
- [36] F. K. Jondral, "Software-defined radio: Basics and evolution to cognitive radio", *EURASIP journal on wireless communications and networking*, vol. 2005, no. 3, pp. 275–283, 2005.
- [37] D. Valerio, "Open Source Software-Defined Radio: A survey on GNUradio and its applications", Forschungszentrum Telekommunikation Wien, Vienna, Austria, Tech. Rep. FTW-TR-2008-002, Aug. 2008. [Online]. Available: https://www.researchgate.net/publication/228781324_Open_Source_Software-Defined_Radio_A_survey_on_GNU-radio_and_its_applications (visited on 04/29/2019).
- [38] O. Anjum, T. Ahonen, F. Garzia, J. Nurmi, C. Brunelli, and H. Berg, "State of the art baseband dsp platforms for software defined radio: A survey", *EURASIP Journal on wireless communications and networking*, vol. 2011, no. 1, p. 5, 2011.
- [39] M. Robert, Y. Sun, T. Goodwin, H. Turner, J. H. Reed, and J. White, "Software frameworks for sdr", *Proceedings of the IEEE*, vol. 103, no. 3, pp. 452–475, 2015.
- [40] W. T. Cochran, J. W. Cooley, D. L. Favon, H. D. Helms, R. A. Kaenel, W. W. Lang, G. C. Maling, D. E. Nelson, C. M. Rader, and P. D. Welch, "What is the fast fourier transform?", *Proceedings of the IEEE*, vol. 55, no. 10, pp. 1664–1674, 1967.
- [41] K. M. Ribeiro, R. A. B. Júnior, T. Sáfiadi, and G. Horgan, "Comparison between fourier and wavelets transforms in biospeckle signals", *Applied Mathematics*, vol. 4, no. 11, pp. 11–22, 2013.
- [42] P. Welch, "The use of fast fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms", *IEEE Transactions on audio and electroacoustics*, vol. 15, no. 2, pp. 70–73, 1967.
- [43] A. V. Oppenheim, *Discrete-time signal processing*. Pearson Education India, 1999.
- [44] M. Akin, "Comparison of wavelet transform and fft methods in the analysis of eeg signals", *Journal of medical systems*, vol. 26, no. 3, pp. 241–247, 2002.
- [45] E. Sejdic, I. Djurovic, *et al.*, "Quantitative performance analysis of scalogram as instantaneous frequency estimator", *IEEE Transactions on Signal Processing*, vol. 56, no. 8, pp. 3837–3845, 2008.
- [46] B. Cazelles, M. Chavez, G. C. d. Magny, J.-F. Guégan, and S. Hales, "Time-dependent spectral analysis of epidemiological time-series with wavelets", *Journal of the Royal Society Interface*, vol. 4, no. 15, pp. 625–636, 2007.
- [47] P. Brémaud, *Mathematical principles of signal processing: Fourier and wavelet analysis*. Springer Science & Business Media, 2013.
- [48] M. I. Hossain, J. Liu, and R. Lee, "A study of multilingual speech features: Perceptive scalogram based on wavelet analysis", in *IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No. 99CH37028)*, IEEE, vol. 2, 1999, pp. 178–183.
- [49] P. Joshi, *Artificial intelligence with python*. Packt Publishing Ltd, 2017.
- [50] T. M. Mitchell, *Machine learning*, en, International ed., [Reprint.], ser. McGraw-Hill series in computer science. New York, NY: McGraw-Hill, 1997.
- [51] 1951 - SNARC Maze Solver - Minsky / Edmonds (American), Nov. 2009. [Online]. Available: <http://cyberneticzoo.com/mazesolvers/1951-maze-solver-minsky-edmonds-american/> (visited on 02/06/2019).
- [52] A. Géron, *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. " O'Reilly Media, Inc.", 2017.
- [53] W. Richert, *Building machine learning systems with Python*. Packt Publishing Ltd, 2013.

- [54] 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.
- [55] D. M. Johannes, “One-class classification”, PhD thesis, Delft University of Technology, Delft, Netherlands, Jun. 2001.
- [56] D. M. Hawkins, *Identification of outliers*. Chapman and Hall London ; New York, 1980.
- [57] M. A. Pimentel, D. A. Clifton, L. Clifton, and L. Tarassenko, “A review of novelty detection”, *Signal Processing*, vol. 99, pp. 215–249, 2014.
- [58] W. Lu and I. Traore, “A new unsupervised anomaly detection framework for detecting network attacks in real-time”, in *International Conference on Cryptology and Network Security*, Springer, 2005, pp. 96–109.
- [59] X. Yang, L. J. Latecki, and D. Pokrajac, “Outlier detection with globally optimal exemplar-based gmm”, in *Proceedings of the 2009 SIAM International Conference on Data Mining*, SIAM, 2009, pp. 145–154.
- [60] D. Miljković, “Review of novelty detection methods”, in *The 33rd International Convention MIPRO*, IEEE, 2010, pp. 593–598.
- [61] D.-Y. Yeung and C. Chow, “Parzen-window network intrusion detectors”, in *Object recognition supported by user interaction for service robots*, IEEE, vol. 4, 2002, pp. 385–388.
- [62] N. Kwak and C.-H. Choi, “Input feature selection by mutual information based on parzen window”, *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 12, pp. 1667–1671, 2002.
- [63] Y. Cao, H. He, H. Man, and X. Shen, “Integration of self-organizing map (som) and kernel density estimation (kde) for network intrusion detection”, in *Unmanned/Unattended Sensors and Sensor Networks VI*, International Society for Optics and Photonics, vol. 7480, 2009, 74800N.
- [64] M. Yousef, N. Najami, and W. Khalifav, “A comparison study between one-class and two-class machine learning for microrna target detection”, *Journal of Biomedical Science and Engineering*, vol. 3, no. 03, p. 247, 2010.
- [65] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, “Lof: Identifying density-based local outliers”, in *ACM sigmod record*, ACM, vol. 29, 2000, pp. 93–104.
- [66] K. Prakobphol and J. Zhan, “A novel outlier detection scheme for network intrusion detection systems”, in *2008 International Conference on Information Security and Assurance (isa 2008)*, IEEE, 2008, pp. 555–560.
- [67] F. Sohrab, J. Raitoharju, M. Gabbouj, and A. Iosifidis, “Subspace support vector data description”, in *2018 24th International Conference on Pattern Recognition (ICPR)*, IEEE, 2018, pp. 722–727.
- [68] N. Japkowicz, C. Myers, M. Gluck, *et al.*, “A novelty detection approach to classification”, in *IJCAI*, vol. 1, 1995, pp. 518–523.
- [69] R. C. Aygun and A. G. Yavuz, “Network anomaly detection with stochastically improved autoencoder based models”, in *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)*, IEEE, 2017, pp. 193–198.
- [70] J. An and S. Cho, “Variational autoencoder based anomaly detection using reconstruction probability”, *Special Lecture on IE*, vol. 2, no. 1, 2015.
- [71] S. H. Kim, H.-J. Kim, and J.-Y. Kim, “Gan-based one-class classification for personalized image retrieval”, in *2018 IEEE International Conference on Big Data and Smart Computing (BigComp)*, IEEE, 2018, pp. 771–774.
- [72] M. Sabokrou, M. Khalooei, M. Fathy, and E. Adeli, “Adversarially learned one-class classifier for novelty detection”, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 3379–3388.
- [73] F. T. Liu, K. M. Ting, and Z.-H. Zhou, “Isolation forest”, in *2008 Eighth IEEE International Conference on Data Mining*, IEEE, 2008, pp. 413–422.

- [74] R. N. Calheiros, K. Ramamohanarao, R. Buyya, C. Leckie, and S. Versteeg, “On the effectiveness of isolation-based anomaly detection in cloud data centers”, *Concurrency and Computation: Practice and Experience*, vol. 29, no. 18, e4169, 2017.
- [75] 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*, 2000, pp. 582–588.
- [76] D. M. Tax and R. P. Duin, “Support vector domain description”, *Pattern recognition letters*, vol. 20, no. 11-13, pp. 1191–1199, 1999.
- [77] P. Winter, E. Hermann, and M. Zeilinger, “Inductive intrusion detection in flow-based network data using one-class support vector machines”, in *2011 4th IFIP international conference on new technologies, mobility and security*, IEEE, 2011, pp. 1–5.
- [78] Y. Zhang, N. Meratnia, and P. Havinga, “Adaptive and online one-class support vector machine-based outlier detection techniques for wireless sensor networks”, in *2009 International Conference on Advanced Information Networking and Applications Workshops*, IEEE, 2009, pp. 990–995.
- [79] P. Oza and V. M. Patel, “One-class convolutional neural network”, *IEEE Signal Processing Letters*, vol. 26, no. 2, pp. 277–281, 2018.
- [80] M. R. Oliveira, J. Neves, R. Valadas, and P. Salvador, “Do we need a perfect ground-truth for benchmarking internet traffic classifiers?”, in *2015 IEEE Conference on Computer Communications (INFOCOM)*, IEEE, 2015, pp. 2452–2460.
- [81] S. Floyd and V. Paxson, “Difficulties in simulating the internet”, *IEEE/ACM Transactions on Networking (ToN)*, vol. 9, no. 4, pp. 392–403, 2001.
- [82] M. Sauter, *From GSM to LTE.: An Introduction to Mobile Networks and Mobile Broadband*. John Wiley & Sons, 2010.
- [83] 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.
- [84] G. Chandrashekar and F. Sahin, “A survey on feature selection methods”, *Computers & Electrical Engineering*, vol. 40, no. 1, pp. 16–28, 2014.
- [85] 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.
- [86] Q.-S. Xu and Y.-Z. Liang, “Monte carlo cross validation”, *Chemometrics and Intelligent Laboratory Systems*, vol. 56, no. 1, pp. 1–11, 2001.
- [87] *Importance of Feature Scaling — scikit-learn 0.21.1 documentation*. [Online]. Available: https://scikit-learn.org/stable/auto_examples/preprocessing/plot_scaling_importance.html (visited on 05/16/2019).
- [88] 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.
- [89] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization”, *arXiv preprint arXiv:1412.6980*, 2014.
- [90] P. Dobbelaere and K. S. Esmaili, “Kafka versus rabbitmq: A comparative study of two industry reference publish/subscribe implementations: Industry paper”, in *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems*, ACM, 2017, pp. 227–238.