PH.D. THESIS IN THE DOCTORAL PROGRAM IN NETWORK AND INFORMATION TECHNOLOGIES

ANOMALY DETECTION IN SMART CITY WIRELESS SENSOR NETWORKS

AUTHOR:
VICTOR GARCIA FONT

ADVISORS:
CARLES GARRIGUES
HELENA RIFÀ POUS

Universitat Oberta de Catalunya

Doctoral program in Network and Information Technologies

Anomaly detection in smart city wireless sensor networks

Advisors:
Carles Garrigues, PhD
Helena Rifà Pous, PhD

Author:
Victor Garcia Font

Deposit authorization: December 22, 2016

Defense: February 8, 2017

Abstract

Over the last few years, cities around the world have been building smart city systems in order to improve their operational structure and to acquire a data-driven management perspective. At an early stage, cities started deploying a few sensors of non-critical services (e.g. atmospheric monitoring), which were considered innocuous from a global security perspective. Nevertheless, nowadays, cities deploy sensors with a wide range of purposes (e.g. parking, safety, lighting) and some areas have become densely populated with wireless sensor networks (WSN). Thus, WSNs turn into an important data source for many applications and, consequently, also become more attractive targets for attacks. Aware of this issue, IT administrators are looking now for security solutions both for the WSNs that are already spread throughout the city and for the ones that will be deployed in the future.

Furthermore, WSNs are normally installed and operated by external providers. This fact complicates security management from the global perspective of the smart city administrators, because different providers implement different solutions using different devices, configurations, protocols, etc., which results in a highly heterogeneous environment. Traditionally, WSN security has been approached as an independent problem for each specific type of network and, therefore, no security solution exists that can be applied in a generalizable manner to all the possible WSNs in a smart city. In this context, security solutions implemented by the providers are currently the main barrier to defend the networks. However, it is also of paramount importance to provide smart city administrators with tools to verify that providers are indeed applying the necessary security measures and also to check that data received from the WSNs are correct. In this thesis, we take a first step in this direction and, taking the point of view of the smart city administrators, we propose an intrusion detection platform to disclose attacks in the WSNs.

In this dissertation, we identify the principal components of an architecture to handle intrusion detection in the heterogeneous context of a smart city. The solution that we propose is based on a centralized system that gathers all data from the WSNs. Then, a rule-based and an anomaly-based detection engines are configured to trigger alarms in the case of attack. This architecture does not add extra requirements for the already deployed WSNs and it is, thereby, compatible with the existing infrastructure of the providers.

Between the two aforementioned detection engines, we focus our analysis on the anomaly-based engine, because it is more generalizable to different smart city configurations. This detection engine generates mathematical models to identify deviations from the normal behavior of the WSN data in attack situations. In this thesis, we compare several anomaly detection algorithms and we observe that, in this context, one-class support vector machines results the most suitable technique.

Furthermore, we identify the various necessary steps from gathering WSN data until running the detection techniques. We evaluate the whole procedure under the processing requirements of this scenario and we attest that: (1) the proposed architecture is capable of handling smart city data and (2) that the entire procedure is scalable.

Finally, by studying the effects of the most popular attacks in WSNs (these effects include the malfunction traces and the anomalies that can be detected with the detection engines of the proposed architecture) we derive seven different attack models. Then, we propose a schema to help smart city administrators to classify the alarms received from the detection engines into one of the attack models, thereby narrowing down the list of the likely attacks and sources compromising the networks.

Acknowledgements

I would like to use this section to express my profound gratitude to the people who supported and helped me during my Ph.D. studies and the creation of this dissertation.

First and foremost, I would like to thank my thesis supervisors Carles Garrigues and Helena Rifà for their guidance and motivation. Thanks for your critical advice and for helping me to overcome the difficulties during the research process. Your counsel has always been valuable and has helped me to improve the quality of my research, my articles and my dissertation.

Thanks to Joaquin Jimenez, Manel Garcia, Carlos Puga, Cast Info and openTrends for starting this project and for their assistance during its development. Manel Mendoza and Institut Municipal d'Informatica deserve particular thanks for sharing their knowledge of smart cities. I also gratefully acknowledge the funding provided by the Ministry of Economy and Competitiveness through the projects CO-PRIVACY (TIN2011-27076-C03-02) and SMARTGLACIS (TIN2014-57364-C2-2-R), and the Government of Catalonia through the industrial doctorate subsidy ECO/2497/2013.

Last, but not least, my colleagues and friends at UOC deserve a special mention for their rewarding company. The last three years have been a period of intense learning in both scientific and personal level. I am certain that without all of you, working on this thesis would have not been such a great experience.

Contents

List of Figures

List of Tables

Abbreviations

6LoWPAN	**Lo**w power **W**ireless **P**ersonal **A**rea **N**etworks
ACK	**Ack**nowledgement
AODV	**A**d hoc **O**n-demand **D**istance **V**ector
API	**A**pplication **P**rogramming **I**nterface
ARIMA	**A**utoregressive **I**ntegrated **M**oving **A**verage
AT	**A**larm **T**ype
BSI	**B**ritish **S**tandards **I**nstitution
CCA	**C**lear **C**hannel **A**ssessment
CoAP	**Co**nstrained **A**pplication **P**rotocol
CSMA-CA	**C**arrier **S**ense **M**ultiple **A**ccess with **C**ollision **A**voidance
CRC	**C**yclic **R**edundancy **C**heck
CTP	**C**ollection **T**ree **P**rotocol
CTS	**C**lear **T**o **S**end
DoS	**D**enial **o** **S**ervice
DBMS	**D**ata**b**ase **M**anagement **S**ystems
FFD	**F**ull-**F**unction **D**evice
FPR	**F**alse **P**ositive **R**ate
FP	**F**alse **P**ositive
FV	**F**eature **V**ector
HDFS	**H**adoop **D**istributed **F**ile **S**ystem
HTTP	**H**ypertext **T**ransfer **P**rotocol
ICMP	**I**nternet **C**ontrol **M**essage **P**rotocol
ICT	**I**nformation and **C**ommunication **T**echnologies
IDS	**I**ntrusion **D**etection **S**ystem
IoT	**I**nternet **o**f **T**hings

IP	**I**nternet **P**rotocol
JSON	**J**ava**S**cript **O**bject **N**otation
KNN	**K**-**N**earest **N**eighbor
LAN	**L**ocal **A**rea **N**etwork
LOF	**L**ocal **O**utlier **F**actor
LPWAN	**L**ow-**P**ower **W**ide-**A**rea **N**etwork
M2M	**M**achine-to-**M**achine
MAC	**M**edia **A**ccess **C**ontrol
NoSQL	**N**ot **o**nly **SQL**
OC-SVM	**O**ne-**C**lass **S**upport **V**ector **M**achines
OWD	**O**ne-**W**ay **D**elay
PFPR	**P**ermitted **F**alse **P**ositive **R**ate
RBF	**R**adial **B**asis **F**unction
RECSI	**R**eunión **E**spañola sobre **C**riptología y **S**eguridad de la **I**nformación
ReST	**Re**presentational **S**tate **T**ransfer
RFD	**R**educed-**F**unction **D**evice
RFID	**R**adio-**F**requency **Id**entification
RSSI	**R**eceived **S**ignal **S**trength **I**ndicator
RTS	**R**eady **T**o **S**end
SCADA	**S**upervisory **C**ontrol **A**nd **D**ata **A**cquisition
SIEM	**S**ecurity **I**nformation and **E**vent **M**anagement
SNR	**S**ignal-to-**N**oise **R**atio
SOA	**S**ervice **O**riented **A**rchitecture
SSL	**S**ecure **S**ockets **L**ayer
SVM	**S**upport **V**ector **M**achines
TDMA	**T**ime **D**ivision **M**ultiple **A**ccess
TPR	**T**rue **P**ositive **R**ate
TSOD	**T**emporal and **S**patial real-data-based **O**utlier **D**etection
UDP	**U**ser **D**atagram **P**rotocol
WLAN	**W**ireless **L**ocal **A**rea **N**etwork
WMAN	**W**ireless **M**etropolitan **A**rea **N**etwork
WPAN	**W**ireless **P**ersonal **A**rea **N**etwork
WSAN	**W**ireless **S**ensor and **A**ctuator **N**etwork

WSN	**W**ireless **S**ensor **N**etwork

Chapter 1

Introduction

In the coming decades, cities are facing new challenges characteristic of contemporary societies: population aging, reduction in energy consumption and carbon emissions, the struggle for greater sustainability, economic growth, etc. In addition, migratory movements are rapidly increasing the size of cities. Nowadays, 50% of the world's population lives in cities and it is foreseen that by 2050 this percentage will be around 70% [1].

To address these challenges, smart city initiatives have emerged proposing new ways of looking at development and city management. Currently, an internationally accepted definition of smart city does not exist. However, the authors of [2] have proposed a definition that has become popular: *We believe a city to be smart when investments in human and social capital and traditional (transport) and modern (ICT) communication infrastructure fuel sustainable economic growth and a high quality of life, with a wise management of natural resources, through participatory governance.*

Generally, smart city projects have the goal of improving metropolitan infrastructure planning, automatizing urban operations, reducing costs, increasing city competitiveness, opening new business lines, creating employment and enhancing transparency and openness [3]. Depending on specific needs, each city implements smart city initiatives focusing on different sectors. *The Smart City Project*[1], which aims at profiling and benchmarking medium and large cities in Europe (it has covered almost 1,600 cities),

[1] "The Smart City Project",$http://www.smart-cities.eu$

has proposed a smart city model including six key sectors: smart economy, smart mobility, smart environment, smart people, smart living and smart governance. From this list, European cities are mainly implementing smart environment and smart mobility [4].

From a technological point of view, information systems are being deployed to transform infrastructure management towards a data-driven approach following four basic building blocks: data, analysis, feedback and adaptability [5]. In order to feed the information systems, smart cities use elements of the Internet of things (IoT) as the main data source, such as mobile phones, radio-frequency identification (RFID) cards and wireless sensor networks (WSNs). The data collected by the latter are used in a plethora of applications. For example, traffic monitoring sensors are used to control traffic lights [6] and wireless meters are installed in pipes to monitor leaks and ruptures [7]. Moreover, these data give city managers and other stakeholders the opportunity to plan future facilities based on a better picture of citizens' behavior and the real use of the current infrastructures.

The clear benefits provided by smart city technology have prompted many cities to devote a considerable part of their innovation efforts to developing their concept of smart city. This has caused a significant and rapid increase in the number of WSN deployments on the streets, which has resulted in the emergence of new applications with many different technologies, solutions, requirements, etc.

However, this accelerated deployment of smart city technology has often resulted in putting security aside as a secondary issue. For instance, some studies [8, 9] have proven that traffic control systems can be manipulated in real deployments in the United States due to the lack of cryptographic and authentication systems in the sensors and, in general, because of a systematic lack of security consciousness.

Moreover, in order to rapidly deploy WSNs and smart city technology, cities have taken advantage of services procured from external providers. Nevertheless, outsourcing public services has also raised security-related concerns [10].

The impact of these outsourcing policies on security can be attributed mainly to two key factors: the loss of control over network devices and the lack of visibility over the potential security problems affecting these devices. Indeed, public administrations usually outsource not only the implementation and deployment of their WSNs, but also the administration thereof. In this way, security countermeasures and system logs are exclusively operated by external providers. Although service providers are contractually obliged to ensure certain levels of security, in practice, smart city administrators cannot determine the extent to which received data are precise and accurate. In fact, the Royal Academy of Engineering has identified data quality as one of the six major barriers to effectively optimize smart infrastructures [5].

Therefore, this thesis aims to improve smart city WSN security from the centralized point of view of smart city administrators. The thesis outlines the principal barriers to achieve secure WSNs in this context, and it provides the schema for an architecture that analyzes WSN data to detect intrusions in networks operated by external providers. This thesis also focuses on analyzing the most convenient algorithms to detect certain intrusions and it provides smart city administrators with guidelines to locate the source of security problems on their WSNs.

In this chapter, Section 1.1 reviews the main smart city initiatives. Secondly, Section 1.2 describes common features in smart city architectures focusing on WSN data collection. Thirdly, Section 1.3 presents the objectives of the thesis. Fourthly, Section 1.4 describes the research methodology used in this thesis. Finally, Section 1.5 contains the main contributions of the thesis and Section 1.6 outlines the rest of the chapters.

1.1 Smart city initiatives

Cities, private companies and other institutions are already involved in smart city projects to provide solutions to the contemporary challenges that cities are facing. The following pages describe some prominent initiatives.

The PlanIT Urban Operating System [2] is a multilayered operating system for urban environments. Its control layer is responsible for responding with low latency to incidents in the sensor/actuator infrastructure. A supervisory layer offers an application programming interfaces (API) and also modules of management, analytics, storage, simulation, security, etc. *PlaceApps* is a layer to publish applications. All the layers are designed following service-oriented architectures (SOA) in order to facilitate application creation, platform service usage and third-party module integration.

[2] "Living PlanIT OS", $http://living-planit.com$

Rio Operation Center [3] has been developed by IBM in Rio de Janeiro to integrate public information from multiple governmental institutions. This center is aimed at improving public safety and increasing incident response efficiency, mainly in the face of natural disasters.

In [11], the authors describe a middleware implemented in Oulu (Finland). This middleware is a layer deployed on top of a series of communication networks (i.e. local area network (LAN), Bluetooth, Wi-Fi), responsible for enabling connectivity with these networks and giving access to data collected by city sensors. The ultimate goal of this project is to build an actual testbed to improve communication between citizens and the government.

Ubiquitous city (u-city)[12] is a South Korean non-intrusive, user-centered project to interconnect urban services divided by area of interest (e.g. building automation, business, governance).

In [13], the author proposes a four-layer architecture to integrate elements of the IoT into smart cities. A key feature of this solution is the inclusion of instruments to stimulate collaboration among elements of the system. For example, low power devices, such as smartphones, send parts of complex processes to the cloud to be computed.

From a futuristic theoretical point of view, the authors of [14] present a framework based on cloud computing middleware and a highly interconnected IoT network. Basically, the authors claim that the IoT will be used to sense and interact with the environment with applications from all kinds of areas (e.g. home automation, transport, community services, operation of infrastructures, health care). The middleware will use paradigms such as software-as-a-service, platform-as-a-service and infrastructure-as-a-service to bind the applications and the IoT together.

SmartSantander[4] is an initiative based in the city of Santander, in the north of Spain, where IoT elements have been massively deployed as a test field for smart city projects. As a result, researchers can experiment in an environment that takes into account real smart city circumstances: large-scale deployment, device heterogeneity, static and mobile sensors, real users, etc. In [15], the authors present more details about the architecture of the system.

[3]"Rio Operation Center", $http://www-03.ibm.com/press/us/en/pressrelease/33303.wss$

[4]"SmartSantander",$http://www.smartsantander.eu/$

Barcelona is taking a leadership role and has proposed *CityOS*[5], an operating system for cities that aggregates modules of data processing, analytics, historical data management, business intelligence, etc. A major objective of the smart city of Barcelona is to deploy a system for the easy integration of third-party modules. For example, CityOS includes the module *City Service Development Kit* [6] (CitySDK), which offers a set of open source tools to aid cities in opening their data and to help developers to create digital services for the city. Other remarkable projects included in the smart city of Barcelona are: *Sentilo*[7], a platform to gather urban sensor data; *iCity*[8], a platform to incentivize third-party projects using public information; and *Open Cities*[9], which is a project to validate user-centered methodologies to use open data in the public sector.

1.2 Generic smart city architecture

In general, analyzing the initiatives presented in the previous section, it can be seen that the architecture of smart city information systems follows certain common patterns. This section outlines these patterns, easily found in most smart cities.

First of all, ICT systems normally are deployed in a so-called silo perspective. This means that an independent new system is designed for each infrastructure. Therefore, cooperation and inter-connectivity among infrastructures remains very limited. Smart city frameworks have sprung up with the goal of breaking these silos, easing application development involving several stakeholders and providing a platform with common services.

Secondly, smart city systems are normally designed as service oriented architectures divided into three layers. The first layer includes the elements that collect information from the city (e.g. sensors, surveillance cameras, social networks, citizen complaint applications, supervisory control and data acquisition (SCADA) systems). The second layer acts as a middleware, which provides the city with an API to connect the elements of the first layer to the services offered in this layer. Among others, these services include relational and non-relational storage, geographic information systems, data analysis, cloud

[5] "CityOS", http://ibarcelona.bcn.cat/ca/o-government/city-os
[6] "CitySDK", http://www.citysdk.eu
[7] "Sentilo", http://www.sentilo.io
[8] "iCity", http://www.icityproject.com
[9] "Open Cities", http://opencities.net

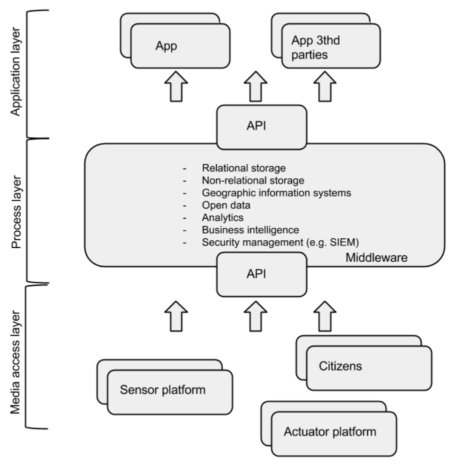


FIGURE 1.1: Generic smart city architecture

computing, natural language processing, business intelligence or open data. Finally, the third layer is an application layer, in which the city council and third parties implement applications based on data and services offered by the middle layer. A scheme of this architecture is shown in Figure 1.1. In general, these architectures aim to maximize interoperability among modules with SOA, in order to encourage development of third-party applications and to facilitate access to services and city data.

The communication channel between street sensors and smart city central servers is represented in Figure 1.2. As shown in the figure, some WSNs are also equipped with actuators, which can be operated from the central servers with a downlink transmission or triggered by other first layer systems using machine-to-machine (M2M) communication. For example, vehicle detection sensors embedded in the asphalt send information to traffic controllers installed in traffic lights [16]. Nevertheless, principally, the infrastructure shown in Figure 1.2 is designed to collect information generated by sensors and send it to the city servers. The elements in this schema are part of the elements from the first and second layers in Figure 1.1. The information flow in this schema begins in the sensors, which gather data about their environment and then send them to a gateway. Gateways finally deliver sensor data to the smart city premises.

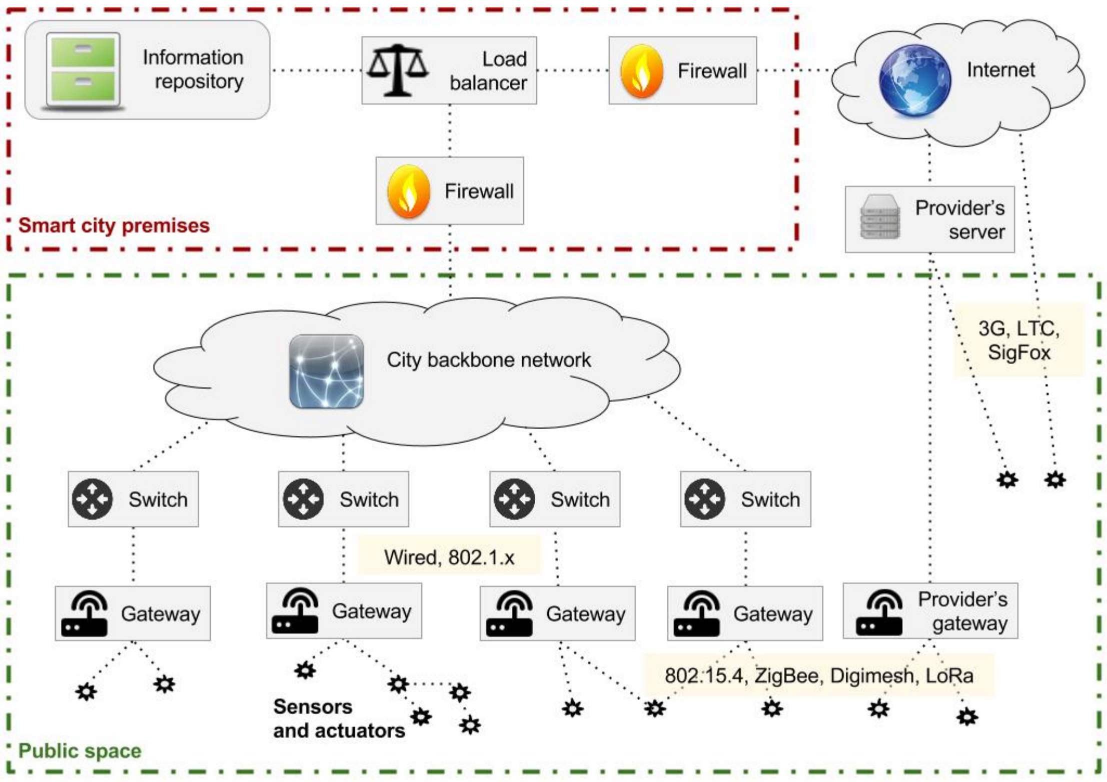


FIGURE 1.2: WSN data collection infrastructure

1.3 Objectives

As mentioned above, the technology deployed in smart cities has great potential as a means of improving their economic progress, social welfare and quality of life while ensuring a more rational and efficient approach to the way services are operated and delivered. However, the current architecture of the information systems poses some security challenges due to the following facts:

- The city includes many systems from different services (e.g. street lighting, garbage collection, water supply), each of which with specific needs and requirements. Consequently, the WSNs from each system are implemented with different technology and are deployed by different providers.
- As it will be seen in Chapter 2, security solutions in the field of WSNs are usually focused on protecting scenarios with very specific characteristics. Nowadays, security solutions for WSNs are not capable of protecting a whole system as heterogeneous as the smart city.

- From the point of view of the smart city administrators, outsourcing the deployment and maintenance of WSNs results in a loss of the visibility of actual effectiveness of the security measures implemented in the providers' networks.
- Due to computational power and battery constraints of sensor nodes, the WSNs often avoid sending system status information, which hinders subsequent security analysis.

Thus, it can be seen that the inherent characteristics of a smart city pose additional security challenges that are not easily overcome with traditional solutions only. Therefore, the main objective of this thesis is to contribute to the security of smart cities by designing an intrusion detection platform for the urban WSNs. As far as we know, this is the first research work approaching this problem in the context of smart cities, where WSNs have become a major data source but are known to be vulnerable at the same time. Below, details of the specific objectives of our work are given:

1. Definition of the architecture of an intrusion detection platform.

   The first goal is to define the main modules of an intrusion detection platform that is:

   - capable of collecting, indexing and processing WSN data,
   - scalable,
   - capable of handling big data,
   - transparent for the providers, and
   - compatible with the existing infrastructure.

2. Definition of the pipeline of the subprocesses involved in attack detection.

   Attack detection involves several subprocesses. These need to be defined and the interactions between different subprocesses have to be identified and studied to guarantee the sustainable and scalable execution of the pipeline.

3. Identification of suitable algorithms for an anomaly-based intrusion detection system.

   This thesis has to identify suitable algorithms for the anomaly-based intrusion detection system, taking into account the requirements of the smart city context and the characteristics of WSN technology.

4. Provision of a mechanism to identify attacks.

   In the case of a security incident in the WSNs, it is essential not only to detect that the network has been compromised, but also to identify the attack and the compromised equipment. A mechanism must be provided to guide smart city administrators in identifying the most likely attacks.

It is worth mentioning that the study of algorithms and techniques to solve security problems for specific smart city business cases, attack types or network configurations is beyond the scope of this thesis. For instance, it is important to identify thresholds for some system status variables, beyond which smart city administrators are confident that certain WSN protocols are not working properly. It is also important to find the best algorithms to discover malfunctions for each of the services offered by the smart city. As these algorithms may be very different depending on the specific service, they have not been considered in our research.

Hence, this thesis aims to contribute with generalizable solutions applicable to diverse smart cities, different services, technologies, etc.

1.4 Research methodology

To achieve the thesis objectives, a *design and creation* [17] research methodology have been applied in an incremental process, where each contribution have been sequentially proposed and validated. For each contribution of this thesis, the five steps that this research methodology involves have been followed: awareness, suggestion, development, evaluation and conclusion. The first stage of the research process included identifying the research problems. Basically, meetings were held with smart city administrators and providers from Barcelona, literature was reviewed and technological solutions related to WSNs and smart cities were critically evaluated. Then, we sketched and proposed solutions and we developed an ICT artefact. The artefact includes models and instantiations that were evaluated with simulations and proofs of concept. In general, in order to make this research comparable and the simulations coherent, wide-known metrics were used to evaluate the results and real data were used in the simulations to the extent possible. More than 10Gb of data were gathered for several months from different urban WSNs from Barcelona for this purpose. Finally, for each solution, the process concluded by identifying the main contributions to the state of the art and also pointing out future research lines.

1.5 Contributions

This section summarizes the contributions of the thesis and the publications derived from it.

The main contributions are:

- Identification of a research challenge relevant to smart city viability: the lack of visibility over the WSN security issues from a holistic perspective such as the one of the smart city administrators.
- A design of an architecture to detect intrusions in smart city WSNs that combines two detection engines: a rule-based and an anomaly-based engines. The design has taken into account the viability and scalability of the system in terms of data volume and computational complexity of the subprocesses involved in the detection pipeline.
- A comparative study of the most suitable anomaly detection techniques to discover attacks in smart city WSNs. The study has considered the detection capabilities of diverse algorithms and has also determined the minimum WSN data required to obtain valid detection results.
- An attack classification schema and a procedure to analyze the security alarms triggered by the proposed architecture to help smart city administrators to identify attacks and compromised nodes.

The thesis contributions have been presented and published in conferences and journals. In this introductory chapter and in the other chapters of this dissertation, we identify new security concerns and barriers arising from smart city technological models and, more specifically, from the adoption of WSNs as a key urban data collection system. The analysis of these security problems was presented on 2014 at the XIII Reunión Española sobre Criptología y Seguridad de la Información (RECSI) [10] and it was published in [121]. RECSI is a biannual Spanish scientific congress in the field of cryptology and security in ICT.

The main schema of the proposed architecture was described in [122] and it was presented on 2015 at the First IEEE International Smart City Conference [11].

The results of the comparative study of anomaly detection algorithms were published in [124], in a Special Issue of the Sensors Journal [12] on the "Security and Privacy in Sensor Networks". Sensors is an international, peer-reviewed journal on science and technology of sensors and biosensors. Moreover, the results of the study were extended and the need of gathering system status and application layer data from the WSNs in order to cover the detection of different types of attack was shown using the technique with the best performance in the comparative study. This was presented on 2016 at the XIV RECSI [13] and it was published in [123].

The description of the attack classification schema has been described in [125], which is currently under review.

1.6 Thesis organization

The thesis is organized as follows:

- Chapter 2 provides the background for the subsequent chapters of the thesis. The main subjects reviewed in the chapter are: big data, WSNs and intrusion detection.
- In Chapter 3, an architecture with the necessary modules to handle the requirements of intrusion detection in the heterogeneous context of the smart city is proposed. This chapter also contains a schema of a pipeline with the main steps needed to disclose attacks from WSN data using the proposed architecture.
- Chapter 4 contains a comparative study of several anomaly detection algorithms to uncover popular attacks in the WSNs of the smart cities. This study has been

[10]"XIII Reunión Española sobre Criptología y Seguridad de la Información",$https://web.ua.es/recsi2014$

[11]"First IEEE International Smart City Conference",$sites.ieee.org/isc2-2015/$

[12]"Sensors",$http://www.mdpi.com/journal/sensors$

[13]"XIV Reunión Española sobre Criptología y Seguridad de la Información",$http://recsi16.uib.es/$

carried out simulating different attacks affecting the main communication layers and using common network metrics to perform the anomaly analysis. In this way, the results can be generalized to other attacks affecting these same communication layers and the techniques can be used with data from any WSN manufacturer.

- Chapter 5 contains a study of the computational complexity of the subprocesses included in the intrusion detection pipeline presented in Chapter 3. The viability of the proposed pipeline and the scalability of its subprocesses are analyzed. For the most complex processes, a schema based on MapReduce is presented to prove that the most critical components can be parallelized.
- In Chapter 6, an attack classification schema for smart city WSNs is proposed. The schema should help smart city administrators to basically narrow down the possible causes of the anomalies detected in the WSNs and identify compromised network components.
- In Chapter 7, the dissertation is concluded and possible future research directions are given.

Chapter 2

Background and related work

This chapter includes the background and related work required to frame the rest of the chapters of this thesis. Firstly, Section 2.1 introduces big data in the context of the smart city. Secondly, Section 2.2 gives an overview of WSNs, showing the most typical protocols and configurations. Thirdly, Section 2.3 illustrates the most relevant topics related to WSN security. Fourthly, Section 2.4 shows related work about intrusion detection. Finally, Section 2.5 presents the role of the standardization organizations related to smart cities and Section 2.6 concludes the chapter.

2.1 Big data and the smart city

Datasets of large **v**olume, high **v**elocity and wide **v**ariety that cannot be processed using traditional methods have been pointed out as *big data* by the *3 Vs* definition [18]. Although the most advanced smart cities are still in their early stages, they already have projects collecting datasets that can be considered big data.

For instance, the *Oyster card* is used to access public transport in London. This system basically registers a user's ID, the location and a timestamp every time a citizen enters or exits the public transportation network. A six-month sample of this dataset contains 7 million records a day, 40 million a week, 160 million a month and almost 1,000 million every half year [19].

Nowadays, some big data properties can also be seen in smart building projects. Many of these projects are based on measuring electrical consumption of home appliances and send the measurements to a centralized location to gather aggregated information and improve electrical system management. These projects deal with large volume and high velocity datasets. For example, a house collecting data from 30 appliances every minute would send 43,000 records a day. A small city of 25,000 houses would send more than 1 billion records a day [20].

Data variety is also present in smart city datasets. Smart cities involve projects that fall under the scope of multiple areas, such as energy efficiency, transportation, environmental protection, etc. Each project has different needs and requirements and, therefore, diverse types of IoT elements are used to fulfill these needs and acquire the necessary data. This creates large-scale and very heterogeneous systems, including devices deployed in a distributed manner that generally collect unstructured or semi-structured data.

Furthermore, other characteristics of smart cities pose obstacles to data processing and show big data properties [21]. For example: useful data are only a small portion of all the gathered data; data are valuable in the long term, which requires historical data management and storage; IoT data show strong temporal and spatial correlation; etc.

The next section reviews the most popular big data management mechanisms. Afterwards, it introduces MapReduce as a basic tool to process in parallel large volumes of data. Finally, it describes the main characteristics of security information and event management (SIEM) systems. As it will be seen in the following chapters, these are key components for the architecture proposed in this thesis.

2.1.1 Big data management mechanisms

In order to process, store and manage big data, it is necessary to rely on mechanisms that go beyond traditional database management systems (DBMS). The most popular mechanisms to store big data are NoSQL databases (not only SQL), which can be classified in three basic classes [21]:

- **Key-value systems** use data structures to store single values indexed with a unique key. Popular products are DynamoDB [1] or Redis [2].

[1]"Amazon DynamoDB", *https : //aws.amazon.com/documentation/dynamodb/*
[2]"Redis", *http : //redis.io/*

- **Document-oriented databases** are similar to key-value systems, but they are capable of storing more complex structures, such as semi-structured JSON or XML documents. MongoDB [3], SimpleDB [4] and CouchDB [5] are popular examples of this type of databases.
- **Column-oriented databases** optimize storage in columns rather than in rows like traditional DBMS. In this way, these systems are capable of scaling horizontally. A few examples are BigTable [6], Cassandra [7] and HBase [8].

Other prominent mechanisms to store big data are implemented as file systems, such as Google's GFS [22] or Hadoop Distributed File System (HDFS) [23].

2.1.2 MapReduce

Besides the storage capabilities described in the previous section, it is also necessary that tools to manage smart city data be capable of executing processes within distributed programming paradigms in order to make the execution scalable. The most popular among these programming paradigms is MapReduce.

MapReduce [24] uses a divide and conquer strategy, where the programmer basically has to implement two functions: *map* and *reduce*. In the *map* function, the programmer uses input data to create an intermediate output dataset with a $< key, value >$ structure. The MapReduce framework sorts, groups by *key* and sends these intermediate datasets to the *reduce* function. The programmer has to implement the *reduce* function to merge data with the same *key*.

For instance, MapReduce can be applied to count the number of word occurrences in a set of documents [24]. In this example, for each word in a document, *map* creates the tuple $< word, 1 >$. The *reduce* function receives all the tuples grouped by *key* and sums up all the values. Figure 2.1 shows a schema of this algorithm.

[3]"MongoDB", $https://www.mongodb.com/$
[4]"Amazon SimpleDB", $https://aws.amazon.com/simpledb/$
[5]"Apache CouchDB", $http://couchdb.apache.org/$
[6]"BigTable", $https://cloud.google.com/bigtable/$
[7]"Apache Cassandra", $http://cassandra.apache.org/$
[8]"Apache HBase", $https://hbase.apache.org/$

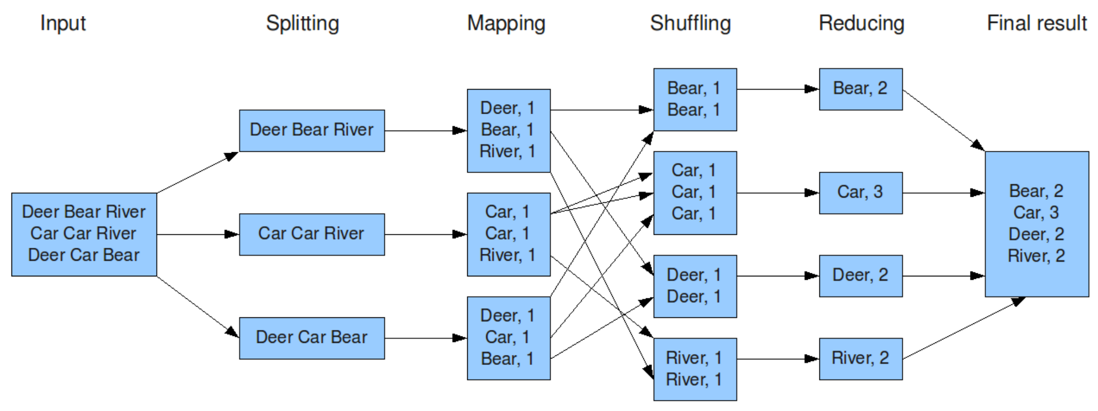


FIGURE 2.1: MapReduce word count example [25].

In this way, *map* and *reduce* jobs can be parallelized using a large number of computer clusters and, therefore, operations that can be divided into these two functions become scalable for large datasets.

MapReduce has been used in this way in many applications. For example in [26], the authors use this programming paradigm in three different contexts: in a movie recommendation engine, in an earthquake simulator and in a large-scale processing task to automatically georeference images.

In [20], the authors describe Scallop4SC, a hybrid architecture using HBase and MySQL [9] to store and process smart city household data with the primary purpose of performing aggregated statistical queries in a scalable manner, such as *amount of energy consumed by air conditioning units*. The authors use MapReduce for the implementation of the data processing programs.

In [27], the authors use MapReduce as a data preprocessing method to index taxi routes in MongoDB. Afterwards, they query MongoDB to obtain the most taken k routes in a certain time interval.

[28] describes a big data platform deployed for Santander's smart city project. The platform can process semi-structured sensor data in real time with MapReduce using CouchDB. Non-structured data from other sources, such as images or social network posts, are processed with HDFS.

In addition, multiple new systems have emerged proposing layers of abstraction in order to simplify programming tasks using the MapReduce programming paradigm as the

[9]"MySQL", *https : //www.mysql.com/*

basis. Generally, in these systems, users program in high-level languages and the code gets processed and converted to MapReduce jobs. Examples of these layers are Pig [10] and Hive [11].

Furthermore, in order to facilitate enterprise-oriented data analysis with big data, some business intelligence (BI) tools such as Pentaho [12] have developed connectors to the main big data storage mechanisms. Along the same lines, SIEM systems are data analysis platforms focused on information security. Logically, SIEMs are relevant tools in the context of the present thesis, so the next section briefly reviews their most important characteristics.

2.1.3 Security information and event management

Security information and event management systems are designed for log management, IT regulatory compliance, event correlation, active response and endpoint security[29]. SIEMs contribute to the security administration of organizations by gathering and correlating security information of several types, formats and sources into a single system. As a result, administrators may leave behind traditional analysis using security mechanisms in a silo perspective. With a SIEM, security practitioners carry out complex monitoring and incident inquiries involving multiple devices and protection mechanisms. Popular SIEM tools are: Splunk [13], AlientVault Open Source SIEM (OSSIM) [14], HPE Security ArcSight [15] and IBM Security QRadar SIEM [16].

For this thesis, the most relevant capabilities of SIEMs are the storage of large datasets and high-velocity data collection. In general, SIEM systems can index big data running as standalone platforms or they can be configured as a top-layer data analysis tool connected to one of the storage mechanisms described in Section 2.1.1. For instance,

[10]"Pig", *https : //pig.apache.org/*
[11]"Hive", *https : //hive.apache.org/*
[12]"Pentaho", *http : //www.pentaho.com/*
[13]"Splunk", *http : //www.splunk.com*
[14]"AlientVault Open Source SIEM (OSSIM)", *http : //www.alienvault.com/open − threat − exchange/projects*
[15]"HPE Security ArcSight", *http : //www8.hp.com/us/en/software − solutions/siem − security − information − event − management/*
[16]"IBM Security QRadar SIEM", *http : //www − 03.ibm.com/software/products/en/qradar − siem*

Splunk offers indexing and parallel processing capabilities to handle large datasets [30], but it also offers connectors to big data mechanisms, for example Hadoop Connect [17].

Furthermore, SIEM systems provide tools to capture data from remote locations or locally. These tools are commonly capable of parsing, compressing, securely processing and sending data from the source to the server. For example, OSSIM offers *Collectors* to classify and normalize events gathered in other external systems [31].

2.2 Wireless sensor networks

Now that the main characteristics of big data management systems and the suitability of SIEMs for processing security-related data have been reviewed, this section sets the focus on the WSN technology that is used to gather these data. This section first gives a general overview of the different technologies enabling this type of network (Section 2.2.1). Then, the subsequent sections briefly describe the most important layers in the communication stack: the physical layer in Section 2.2.2, the data link layer in Section 2.2.3, the network layer in Section 2.2.4 and the application layer in Section 2.2.5.

2.2.1 General overview

Wireless sensor networks are networks that communicate using wireless technology, where nodes, also known as motes, are equipped with one or several sensors to capture information about their environment. When the motes are also equipped with actuators that enable them to perform a certain action, then these networks are known as *wireless sensor and actuator networks* (WSAN).

In smart cities, it is common for motes to have autonomous cooperative communication to send values read by their sensors to a device at the edge of the WSN known as the gateway or base station. Gateways are equipped with several communication interfaces with the aim of transmitting WSN data to the smart city data centers through a conventional and reliable network (e.g. Internet).

[17]"Hadoop Connect",$http : //www.splunk.com/en_us/solutions/solution-areas/big-data/splunk-hadoop-connect.html$

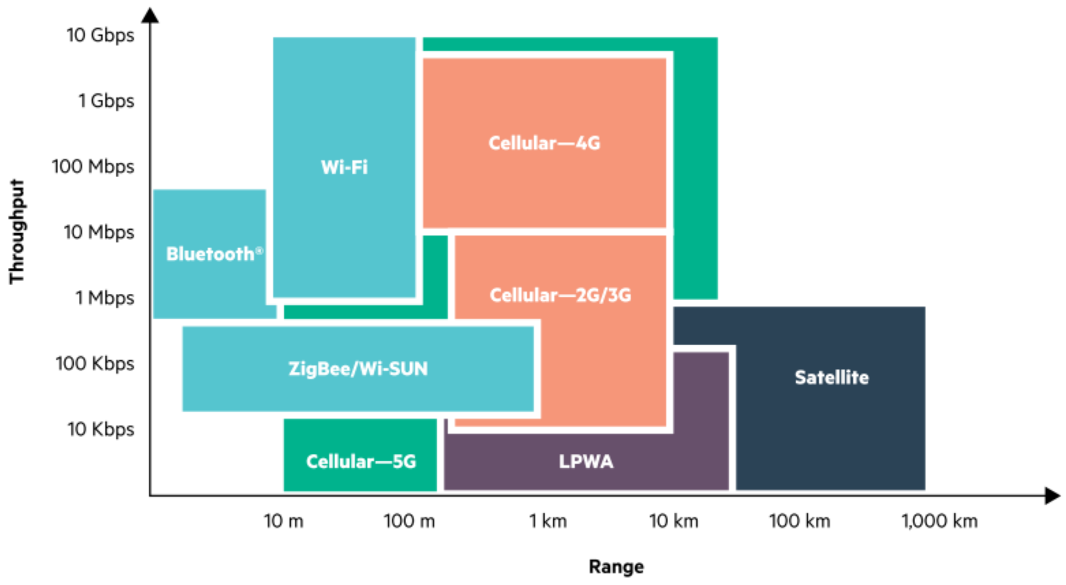


FIGURE 2.2: Range and throughput comparison among wireless technologies [32].

There are multiple types of wireless communication protocols that build different kinds of wireless networks. Figure 2.2 compares the most important technologies according to their range and throughput. Basically, the most relevant WSN technologies in smart cities are:

- *Wireless personal area networks* (WPAN) are low-power, low-throughput, short-range (up to few meters) wireless networks that are based on the standard IEEE 802.15. Relevant technologies included in this category are ZigBee, IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) and Bluetooth. There are many use cases in smart cities using these protocols. For instance, ZigBee is used in [33] to control street lighting.

- *Wireless local area networks* (WLAN), such as Wi-Fi, provide low-range but broad-throughput wireless networks. Some cities, like San Jose in California [34], deploy Wi-Fi connectivity on the streets not only to offer Internet to citizens, but also to connect IoT elements that require larger bandwidth than the usual sensor applications, such as IP-based traffic cameras.

- *Wireless metropolitan area networks* (WMAN) follow IEEE 802.16 standards. Protocols following these family of standards are popularly known as WiMAX. This technology is mainly used for applications requiring large deployments (up to 25

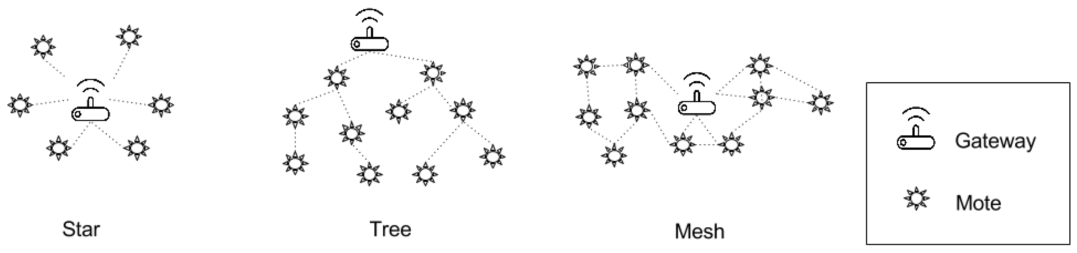


FIGURE 2.3: WSN communication topologies

km) and non-restringed throughput ($<$ 150 Mb/s). In [35], the authors propose to use WiMAX for smart grid projects.

- *Low-power wide-area networks* (LPWAN) are low-power, long-range (up to 10 km), low-throughput ($<$5 Kb/s) wireless networks [36]. SigFox [18] and LoRaWAN [19] are the most popular technologies at the moment. Both technologies have been used in many smart parking applications [37].

Among these types of network, WPAN are considered especially vulnerable. These are made of low-power devices and short-range communication modules, which rely in many cases on multi-hop capabilities to build an extensive network and deliver packets from the most remote nodes to the base station. Besides, motes are frequently battery-powered and, therefore, are designed also with restringed processing capacity to save power. Therefore, this thesis focuses on this type of WSN to perform attacks and intrusion analysis. However, results are generalizable to the other types of WSN.

As mentioned above, many WSNs rely on multi-hop capabilities to deliver packets from one end of the network to the other end. This allows three basic topologies shown in Figure 2.3: star, tree and mesh. These topologies can include three types of nodes: gateways, motes with routing capabilities and leaf motes. Gateways and routing nodes consume large amounts of energy-forwarding packets and, thereby, they are generally plugged into the electrical grid. However, leaf nodes can be battery-powered, because their sole responsibility is sensing the environment and sending their own packets towards the gateway.

[18]"SigFox",$https://www.sigfox.com/$
[19]"LoRaWAN",$https://www.lora-alliance.org/$

The following sections summarize the responsibilities of the most relevant layers in the communication stack for WSNs. These are the physical, data link, network and application layers.

2.2.2 Physical layer

The physical layer handles the way that bits are transmitted over the medium (the air in WSNs). Thus, its main responsibilities include defining operating frequencies, modulation, carrier sense, bit rate, etc. In ZigBee and 6LoWPAN this layer is defined by the IEEE 802.15.4 standard [38].

2.2.3 Data link layer

The data link layer is responsible for transferring data between adjacent nodes in a network. In WSNs, the media access control (MAC) sublayer is of particular importance. This sublayer organizes the network nodes so that the transmission medium is accessed in an ordered manner, which allows for correct communication. As with the physical layer, the MAC layer is also defined by the IEEE 802.15.4 standard [38].

According to this standard, nodes can take the role of a full-function device (FFD), which has routing capabilities, or the role of a reduced-function device (RFD), which limits the nodes to just transmit their own data. In this way, two types of topologies are possible in this layer: star and peer-to-peer. In a star topology, a single FFD receives messages from several FFDs or RFDs. In a peer-to-peer topology, several FFDs can communicate with each other. It is important to note that FFDs normally consume more energy than RFDs and, therefore, generally, they cannot be battery-powered.

Regarding media access, protocols use two main types of strategies. On the one hand, protocols based on time division multiple access (TDMA) divide time in slots and transmitters reserve a slot before transmitting. This type of protocol requires beacons to synchronize transmitters and receivers. In other types of protocols, however, nodes cannot reserve slots, but they are provided with a mechanism to effectively transmit packets without creating interference on communications from the other nodes in the network. For example, carrier sense multiple access with collision avoidance (CSMA-CA) basically specifies that, before transmitting, a node needs to sense the medium and simply start a transmission if the channel is free. In [39], authors analyze multiple MAC protocols for WSNs.

Furthermore, IEEE 802.15.4 standard also defines data verification (i.e. cyclic redundancy check (CRC)) and security mechanisms to ensure data confidentiality, authenticity and replay protection in single link communications.

2.2.4 Network layer

The network layer enables multi-hop network topologies. Nodes implementing this layer can become intermediaries, developing routing capabilities and forwarding packets from other nodes. In WPAN, there are two main standards for embedded systems that include the specification for a network layer: ZigBee (protocol stack shown in Figure 2.4) and 6LoWPAN (protocol stack shown in Figure 2.5).

As shown in the figures, these standards define all the necessary layers in the communication stack. Although it is not mandatory, both protocols are defined on top of IEEE 802.15.4. Both offer typical network services, such as neighbor discovery, route discovery, addressing, routing, etc. ZigBee [40] has been proposed by the ZigBee Alliance [20] and more information can be found in [40].

6LoWPAN has been defined by the IETF [21] in [41]. This protocol definition proposes an interoperability layer to send IPv6 packets over low-power and lossy networks. Thus, this protocol is easily integrable with conventional networks: gateways are simple; the same network addressing space can be used; and protocols above IP, such as User Datagram Protocol (UDP) or Internet Control Message Protocol (ICMP), are compatible.

The network layer contains multiple protocols to deal with the different responsibilities of the layer. For example, *Ad hoc On-Demand Distance Vector* (AODV) [43] is a very popular routing protocol that not only can be used in ZigBee and 6LoWPAN, but also in other mobile networks.

[20]"ZigBee Alliance", $http://www.zigbee.org/$
[21]"IETF", $https://www.ietf.org/$

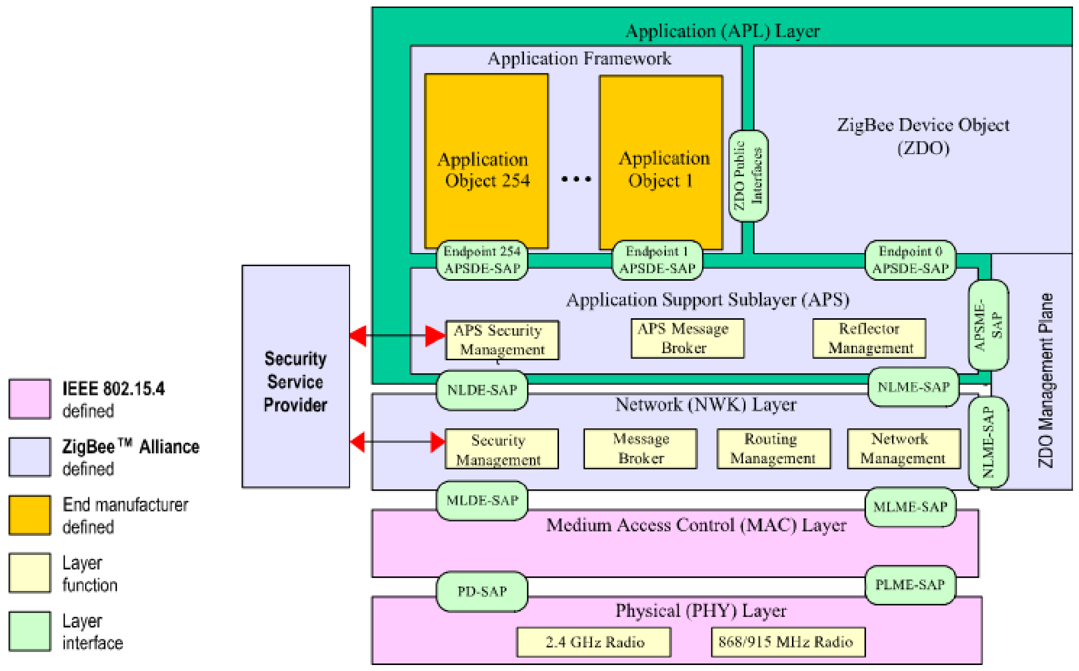


FIGURE 2.4: ZigBee communication stack [40].

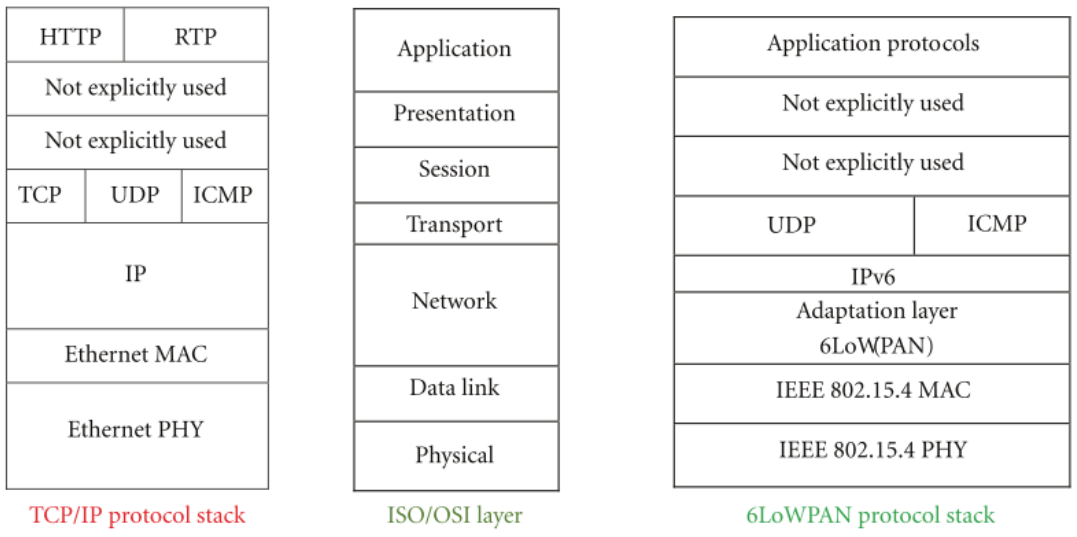


FIGURE 2.5: 6LoWPAN communication stack [42].

2.2.5 Application layer

The application layer is on top of the communication stack. In this layer, application-specific operations are implemented. Furthermore, the Constrained Application Protocol (CoAP) [44] has emerged as a protocol similar to Hypertext Transfer Protocol (HTTP) for the conventional Internet. CoAP works over UDP, it supports the Representational

State Transfer (ReST) methods of HTTP, and it provides subscription and push notifications. Thus, CoAP provides an interoperable constrained application protocol for the IoT.

2.3 WSN security

This section reviews attacks on WSNs (Section 2.3.1) and it also provides basic countermeasures that have been proposed against these attacks (Section 2.3.2).

2.3.1 Attacks on WSNs

The limited computational and energetic constraints of nodes are an obstacle to applying conventional computer network security countermeasures in WSNs. Furthermore, in these networks, nodes become more vulnerable when they are placed in unprotected environments like streets. In these circumstances, attackers can easily capture nodes, access confidential information in their memory (e.g. cryptographic keys) and reprogram their behavior. It is also common that attackers benefit from the wireless nature of the communications to intercept the messages or to obstruct frequency bands to impede the proper reception of some packets. In [45, 46], the authors survey the most popular attacks on WSNs. In the next sections, the most outstanding attacks affecting each of the layers of the communication protocols are reviewed.

2.3.1.1 Attacks against the physical layer

- **Data tampering:** Data in transit between two nodes are modified.
- **Node tampering:** A node is captured in order to damage it or to extract information from its memory.
- **Node replication:** New nodes are added to the network by copying nodes that are already legitimate network members.
- **Jamming:** Attackers send a high-power signal in order to generate interference and avoid correct reception of legitimate packets.

2.3.1.2 Attacks against the data link layer

- **Sybil:** A node takes several identities to change the behavior of data link protocols. This has consequences for communication protocols relying on data aggregation (i.e. in order to forward fewer packets, intermediate nodes aggregate received data from several nodes and send a new packet with the aggregated data) or on voting (i.e. intermediate nodes make decisions, such as deciding the best link, according to the votes casted by other nodes).
- **Interrogation:** Attackers exploit MAC protocols based on a two-way handshake (e.g. protocols that send control packets, such as Ready To Send (RTS) and Clear To Send (CTS)). Attackers send many RTS, so that listening nodes answer with a CTS for each received RTS and, therefore, consume resources.
- **Exhaustion:** Attackers constantly occupy the communication channel. Thereby, legitimate nodes using carrier sense protocols (which are used before transmitting to check whether the transmission medium is free) get their transmissions delayed or even canceled.
- **Collision:** Attackers create interference during legitimate transmissions. In this way, checksum mechanisms discard received messages and transmitters have to resend messages.
- **Unfairness:** Persistence on attacks such as exhaustion or collision in order to highly decrease quality of service and create a total or partial Denial of Service (DoS).

2.3.1.3 Attacks against the network layer

- **Sleep deprivation:** Attackers generate a lot of traffic by means of broadcast packets or by creating network loops in order to keep many nodes awake re-transmitting packets.
- **Internet smurf:** Attackers impersonate a node and then they send multiple ECHO requests in broadcast. ECHO replays saturate the impersonated victim.
- **Misdirection:** Attackers with routing capabilities forward packets towards links where the final destination is not reachable.

- **Acknowledgement spoofing:** Attackers impersonate a legitimate node and send Acknowledgement (ACK) packets indicating reception of incorrectly received packets. This keeps transmitters from re-sending the packets.
- **Spoofed, altered, or replayed routing information:** Routing information sent between legitimate nodes is captured, changed and re-sent in order to create loops, attracting traffic to target nodes, segmenting the network, etc.
- **Wormhole:** A low-latency transmission channel is created between two far-apart attackers. The attacker located at greater distance from the base station benefits from better communications than its neighbors to obtain better routing metrics. Therefore, this attacker becomes the best routing node in the area and attracts traffic.
- **Sybil:** Attackers take diverse identities from legitimate nodes. Then, attackers can mislead other routing nodes to change the routing path towards or away from the impersonated nodes.
- **Selective forwarding and blackhole:** Attackers in a routing position discard some (selective forwarding) or all (blackhole) packets from certain nodes.
- **Hello flood:** Attackers use a powerful transmitter to send HELLO messages to join the network to a large amount of nodes. Listener nodes reply to this fake request with futile messages.
- **Sinkhole:** Some nodes in an area are misled to believe that either a target node or an attacker is the best routing link. In the first case, the target node has to consume extra resources forwarding packets. In the second case, the attacker starts forwarding packets from many nodes and, therefore, can perform other attacks, such as selective forwarding.
- **Homing:** Traffic and network analysis is performed in order to determine key network nodes. These nodes become the best candidates for other attacks.

2.3.1.4 Attacks against the transport layer

- **De-synchronization:** First, attackers impersonate a legitimate node. Then, they request retransmission of properly transmitted packets in a legitimate connection established with another node in the network. In this way, legitimate nodes misuse their resources in futile re-transmissions.

- **Flooding:** Attackers repeatedly send connection requests to other nodes, so that these reserve and exhaust their resources.

2.3.1.5 Attacks against the application layer

- **Deluge:** Attackers take advantage of over-the-air systems to remotely reprogram nodes.
- **Path-based DoS:** Duplicated application packets are injected in leaf nodes. Thereby, packets are forwarded up to the base station, where they are discarded. This is resource-consuming and prevents other nodes from sending their packets.
- **Overwhelm:** Over-stimulation of sensors in leaf nodes to generate large amounts of packets traversing and saturating multiple paths.
- **Eavesdropping:** Attackers read packets transmitted between two legitimate nodes.
- **Re-play:** Attackers re-transmit already sent legitimate packets.

2.3.2 Basic countermeasures

As Table 2.1 shows, the attacks mentioned in the previous section can be used to compromise data confidentiality, integrity, availability and non-repudiation. In order to protect networks against these attacks, researchers have proposed many countermeasures [45, 46]. This section discusses the main protection mechanisms found in the literature.

Basically, confidentiality attacks have two main origins: physically accessing node memory or eavesdropping on wireless transmissions. In smart city WSNs, physical access is easy in many cases since sensor nodes are deployed unprotected in the streets. Tamper-resistant hardware is a strong countermeasure in this case. However, for most smart city services it is too expensive to be implemented in all the nodes. Other more economical alternatives have been proposed: code obfuscation and code attestation [47]. In code obfuscation, some techniques are used to make code and data more difficult to read, increasing, thereby, the amount of time required to perform an attack. Code attestation is used to check if running code has been altered.

Table 2.1: Information security principles compromised by WSN attacks

Physical layer

Attack	**Compromised principles**
Data tampering	Integrity
Node tampering	Confidentiality, integrity, availability
Node replication	Integrity
Jamming	Availability

Data link layer

Attack	**Compromised principles**
Sybil	Integrity, availability
Interrogation	Availability
Exhaustion	Availability
Collision	Availability
Unfairness	Availability

Network layer

Attack	**Compromised principles**
Sleep deprivation	Availability
Internet smurf	Integrity, availability
Misdirection	Availability
Acknowledgement spoofing	Integrity, availability
Spoofed, altered, or replayed routing information	Integrity, availability
Wormhole	Availability
Sybil	Integrity, availability
Selective forwarding and blackhole	Availability
Hello flood	Availability
Sinkhole	Availability
Homing	Confidentiality

Transport layer

Attack	**Compromised principles**
De-synchronization	Integrity, availability
Flooding	Availability

Application layer

Attack	**Compromised principles**
Deluge	Integrity
Path-based DoS	Availability
Overwhelm	Availability
Eavesdropping	Confidentiality
Re-play	Non-repudiation

Confidentiality problems due to the wireless nature of WSNs are normally tackled with cryptographic solutions. Since the first WSN nodes were designed with minimum processing power, legacy systems based on these networks are incapable of running any cryptographic algorithm. However, in the last few years, manufacturers have developed more powerful nodes and new protocols have been designed to take into account cryptographic requirements. For example, the specification of the most popular communication protocols for WSN, e.g. IEEE 802.15.4 standard [38] and ZigBee [40], include different security modes based on symmetric cryptography. Asymmetric cryptography has also been proposed for some situations. In [48], the authors propose to implement asymmetric cryptography through a Secure Sockets Layer (SSL) protocol for WSNs. Libelium [22] proposes using symmetric cryptography for data exchange, and then regularly renewing the cryptographic keys using asymmetric cryptography with RSA 1024 [49].

Cryptography is also a mechanism to avoid integrity and non-repudiation attacks. Checksums and message authentication codes are the usual countermeasures to impede unnoticed modifications of packets in transit. The destination node of an altered packet discards it if the received packet and the code generated by the message integrity mechanism do not match. However, integrity attacks are hardly noticed by city administrators, since most WSNs do not send information to the base station indicating the reasons why packets are dropped. Thereby, from the centralized point of view of smart city administrators, the traces of this type of attack can be assimilated to the traces of attacks against data availability.

Availability attacks normally focus on breaking communication in certain areas and depleting node batteries. Although there are solutions in the literature to avoid this type of attacks, they are not always effective or applicable. For instance, frequency hopping spread spectrum [50] is used to avoid certain types of jamming attacks by constantly changing the transmission channel within the frequency band of the protocol. However, jammer devices currently available on the market can jam all the channels used by several protocols at the same time.

Hence, it can be seen that attacks can succeed and impact data confidentiality, integrity, availability and non-repudiation. Although there are countermeasures to stop or at least slow down the attacks, in this context security barriers are often penetrable. Therefore, the best mitigation approach is a good detection strategy. Even though much information to identify attacks is already lost when it reaches the smart city data centers, it is important at the very least to detect that the networks are under attack in order to increase the strength of the applied security measures and push WSN providers to

[22]"Libelium",*http : //www.libelium.com/*

improve their network security. Indeed, in a smart city context, discovering that certain WSN components are under attack is notably important, since many of the components are shared among different networks (e.g. gateways). Thus, some attacks do not stay isolated in a single system and can have consequences for several services and providers. Next section describes intrusion detection techniques that can be used from the smart city centralized point of view to analyze WSN data and point out attacks.

2.4 Intrusion detection

Within the research field of intrusion detection, two types of techniques can be distinguished: misuse detection and anomaly detection. Whereas the former looks for traces left by the attackers in the security data (e.g. system logs), the latter analyzes the normal behavior of the system and points out unusual changes.

Intrusion detection techniques looking for misuses rely on an extensive database of attack signatures. An attack signature is a sequence of typical actions that can be recorded in a security log. The signature can be used to identify an attacker's attempt to exploit a known network, operating system or application vulnerability. Alarms are raised when the detection system discovers a sequence of events that matches any of the signatures [51]. The main advantage of this type of detection is the low rate of false positives. In the context of WSNs in smart cities, signature-based detection is useful in identifying attacks targeting networks with regular behavior (e.g. environmental sensors sending readings every day at the same hour) or highly reliable services. Simple rules can be created in these two cases to trigger alerts when the expected readings are not received or when a certain number of packets are lost. Nonetheless, many smart city services do not follow a regular pattern and WSN is an unreliable technology, where some packets are occasionally not delivered.

Alternatively, intrusion detection techniques looking for anomalies are able to identify changes in the system that do not match the normal behavior. Given the significance of this type of intrusion detection for this thesis, next section provides more details about it.

2.4.1 Anomaly detection

Anomaly detection has been widely used in many application domains (see a survey on anomaly detection techniques in [52]). The most common techniques fall into the scope of statistics, clustering and machine learning. Depending on the types of samples necessary to process the data, these techniques are divided into supervised, semi-supervised or unsupervised.

Supervised techniques require a training dataset with labels indicating the category of each sample (e.g. "no attack", "jamming" or "selective forwarding"). Then, a model is generated to classify new unlabeled samples into one of the defined categories. Semi-supervised techniques require a training dataset with samples of a single category in order to create a model that classifies new samples as belonging to that category or not. Finally, unsupervised techniques do not require labeled training data and are capable of dividing a dataset into various subsets without a previously learnt model.

Furthermore, anomaly detection algorithms are also clearly separable between univariate algorithms (only one variable is used in the analysis) and multivariate algorithms (several variables are used in the analysis). In univariate algorithms, computing a higher and a lower bound beyond which data are considered anomalous is common. As an example, *Tukey's method* [53] is popular for computing these boundaries from a numeric dataset. In this method, two types of boundaries are defined: the inner fences and the outer fences. The former are computed by subtracting and adding 1.5 times the interquartile distance of the dataset (i.e. distance between the first and the third quartile) to the first and the third quartiles respectively. This defines very strict thresholds, which implies a high probability of identifying some normal instances as outliers. Outer fences represent a more loose way to define the boundaries. The outer fences are computed by subtracting and adding 3 times the interquartile distance to the first and the third quartiles respectively. In order to compute the boundaries with this method, it is recommended that the large datasets not be highly skewed.

Another way to compute thresholds with univariate algorithms is using autoregressive models [54], such as autoregressive integrated moving average (ARIMA). These models are based upon the assumption that each value is somehow correlated with the previous recorded values. In this way, autoregressive models use previous values to predict future values within a confidence interval. The lower and higher boundary in the interval can be used as thresholds to point out anomalies. Autoregressive models are very common in time series analysis.

Multivariate anomaly detection is generally handled by machine learning and clustering techniques. Widely used algorithms are support vector machines [55], nearest neighbor [56] and local outlier factor [57].

Depending on the characteristics of the specific scenario and on the requirements of the application, some algorithms perform better than others. For instance, the authors of [58] compare several unsupervised approaches based on local outlier factor, near neighbors, Mahalanobis distance and support vector machines to detect intrusions in conventional computer networks. Their experiments show that the local outlier factor approach is the most adequate in this context.

Anomaly detection has been also used in intrusion detection systems (IDS) for WSNs. In [59], the authors survey the most popular techniques. Generally, the nodes that contain IDS components gather and/or analyze network status data concerning anomalous operation activities of their neighbors. When this occurs, the nodes trigger an alarm at the base station.

Anomaly detection techniques have been applied in multiple applications related to WSNs. As an example, the authors of [60] use geostatistics and time series analysis to detect outliers in readings of meteorological sensors. The authors select temporal and spatial real-data-based outlier detection (TSOD) as the most appropriate technique in this context. In their experiments, the authors claim that TSOD has a high performance and it is able to identify all the outliers with a low false positive rate, around 3%. However, these techniques are only applicable to certain scenarios in which there exists a spatio-temporal correlation and the WSN is dense enough.

Other studies focus on anomaly detection applied to single sensors. By way of example, in [61], the authors propose a two-phased algorithm. In the first phase, the algorithm seeks temporal anomalies with one-class support vector machines (OC-SVM) and, in the second phase, the algorithm reduces false positives and classifies the anomalies with a supervised K-Nearest Neighbor (KNN) approach. For the first phase, the authors compare OC-SVM with other techniques (i.e. logistic regression, random forest, linear support vector classification, complexity invariant distance-based KNN and Euclidean distance-based

KNN). The authors conclude that OC-SVM outperforms the other techniques, achieving a 96% detection rate in their experiments.

In [62], Mahalanobis distance is used to detect insider attacks with high detection accuracy and robustness (i.e. the false positive rate stays low even though the number of outlying sensors increases). Some authors claim that anomaly detection techniques based on the distance to the neighbors should not be used in WSN due to high computational complexity [63]. Nevertheless, from the point of view of smart city administrators, these techniques can be considered because anomaly analysis can be computed in data centers using powerful computers.

In [64], the authors use one-class quarter-sphere support vector machines in two new anomaly detection algorithms: lightweight anomaly detection algorithm using sort and lightweight anomaly detection algorithm using quick select. These algorithms are suitable to run in constrained nodes due to their low computational complexity. Moreover, their experiments show a high performance, e.g. a 95% true positive rate and a false positive rate of below 10%.

The authors of [65] use an improved ARIMA model to predict anomalies in WSN through network traffic analysis in the nodes. The experiments in the article show an accuracy of over 96% and a false positive rate of less than 3%.

Although some of the previously mentioned anomaly detection techniques and IDS perform well in detecting attacks, they are not a generalizable solution in an heterogeneous context such as the smart city. This is due to the fact that, on the one hand, some techniques excessively depend on the context of the WSN. On the other hand, IDS are normally designed ad-hoc to be embedded in some or all of the nodes of specific WSNs. Therefore, IDS can only be considered as a first protection mechanism to be implemented by WSN providers for their specific networks. From the centralized perspective of smart city administration, the solution must not require access to the WSN nodes nor knowledge of the specific technology used by each external provider.

2.5 The role of standards

With the aim of contributing to smart city construction in a progressive, incremental and scalable manner, leading standardization corporations have started to write guides and regulations. These documents are generally written based on principles of consensus, openness, transparency and non-discrimination. Taking these into account, standards about smart cities are written with the main purposes of improving interoperability among components, promoting secure ways of sharing data, easing entrance to new application development players, removing commercial monopolies, reducing costs, etc. [66]

Although, nowadays, the drafting of such texts is immature and it does still not cover all areas, there are already some relevant documents related to the smart city. For instance, *ISO 37120* [67] describes a suite of indicators to measure and compare different city services in a verifiable manner. *ITU-T Y.4400 series* [68] offers a terminological guide about ICT for sustainable smart cities and *ITU-T L.1600 series* [69] is similar to *ISO 37120* and lists ICT key performance indicators.

Local standardization organizations are also contributing with documentation. The British Standards Institution (BSI)[23] has published guides to facilitate data interoperability[70] and to improve planning and smart city development[71]. In the Spanish sphere, *AEN/CTN 178* an AENOR[24] working group has published a set of standards about smart cities. Among them are some related to ICT: *UNE 178102-1:2015* [72] on multiservice city networks; *UNE 178104:2015* [73] on integrated systems for smart city management; *UNE 178107-4:2015 IN* [74] on access and transport in WSNs; *UNE 178301:2015* [75] on open data.

Beyond traditional standardization organizations, it is also important to highlight some of the initiatives and contributions of other entities. For example, the *City Protocol Society* [25] is a community of cities, companies, academia and nonprofit organizations with the aim of offering recommendations, technical information and use cases as a model for other cities. The *Open & Agile Smart Cities initiative* [26] involves 89 cities from 19 countries in Europe, Latin America and Asia-Pacific. This initiative recommends the adoption of: a driven-by-implementation approach, where communities and developers

[23]"British Standards Institution", http://www.bsigroup.com
[24]"AENOR", http://www.aenor.es/
[25]"City Protocol Society", *http : //cityprotocol.org*
[26]"Open & Agile Smart Cities initiative", *http : //oascities.org*

work together creating services; a technical API; a data model set; and an open data platform.

In general, it can be seen that the standards published to date do not deal with security in depth, since none provide a solution to the above mentioned problems derived from the heterogeneous context, externalized services, and so on.

2.6 Conclusions

This chapter has shown that traditional security protocols are necessary to protect smart city subsystems from different types of attacks. These solutions are generally designed by embedding in the WSNs some countermeasures based on cryptography, code obfuscation and frequency hopping, among others. However, existing solutions cannot holistically cover highly heterogeneous environments with multiple and diverse network technologies and communication protocols. To respond to the needs of this type of system, this thesis proposes to use intrusion detection techniques based on signatures and also on anomaly analysis. In the following chapters, a schema of the architecture will be defined to integrate these detection techniques, and a study of the most suitable anomaly detection algorithms for smart city WSNs will be performed. Finally, a classification system will also be proposed to assist smart city administrators in recognizing the specific attacks in the case of a network compromise.

Chapter 3

Architecture

The previous chapters have shown how smart city systems increase interconnectivity among infrastructures, and create new ways to spread vulnerabilities and to exploit infrastructure dependencies, causing damage to third parties. We have also seen that WSNs are basic components for gathering urban data, but, at the same time, they use technology which is less reliable and easier to attack than conventional networks. In order to achieve rapid deployment of WSNs and smart city technology, cities have taken advantage of services procured from external providers; this outsourcing of public services has, however, raised security-related concerns. Basically, we have observed that smart city administrators have lost control over the network devices, and the visibility over the WSNs to identify potential security issues has decreased.

Moreover, it is not feasible to design security strategies and countermeasures applicable to all types of WSNs that can be deployed in a city. Although smart cities are still in their inception, their rapid growth has led to the deployment of state of the art technology in a new field (i.e. the city), in a manner that makes it hard to apply universal security solutions to the WSNs. The following characteristics of smart cities are the three main impediments to deploy generalizable solutions:

- **Heterogeneity:** Multiple providers implement different technological solutions, under diverse security requirements. Traditional security measures cannot be applied to WSNs, and WSN countermeasures do not cover all circumstances.

- **Limited access:** Providers usually restrain public administrations from accessing their equipment. Accordingly, thorough security analysis in the WSNs can only be performed by the providers. Nevertheless, smart city managers are the only actors with information from all the providers and services. Therefore, incidents involving multiple providers or provoking cascading effects can only be studied by the smart city managers.

- **Difficulty to update:** System updates become very costly and sometimes even impossible in WSNs, because some networks do not offer a downlink communication channel from the data centers to the sensors and, therefore, sensors have to be physically access in order to update their software. Hence, newly discovered vulnerabilities in WSNs frequently go unpatched once they have been deployed.

Therefore, there are currently no security solutions that smart city administrators can use to successfully manage WSN security in a holistic manner. Thus, the main security barriers are implemented and controlled only by the providers. In this thesis, we propose to improve WSN security in smart cities with an intrusion detection platform for the smart city administrators. In this way, WSN data sent by the providers can be analyzed looking for attacks and other failures and, therefore, smart city administrators can push providers to implement the most adequate security countermeasures in their networks. This chapter presents an architecture for this platform. The proposed architecture is envisaged as an additional layer in the smart city architecture above the elements deployed by the providers. Thereby, the architecture is transparent for the providers, it does not add extra requirements for the WSN nodes, which are very limited in terms of processing power and battery, and it is compatible with the WSNs that are already deployed on the streets.

This chapter presents the main requirements for the architecture in Section 3.1. Then, Section 3.2 gives an overview of the proposed architecture. Section 3.3 discusses the major considerations of designing an anomaly-based detection engine, which is one of the principal components of the proposed architecture. Section 3.4 provides a schema with the most relevant subprocesses involved in the intrusion detection pipeline, from data reception at the central server to the generation of alarms due to detected anomalies. Finally, Section 3.5 presents a use case based on a public car park as proof of concept of the architecture.

3.1 Main architecture requirements

A generic platform to detect intrusions in smart city WSNs has to be designed from the point of view of the smart city administrators. Firstly, this means taking into account that smart city administrators have a centralized perspective. Hence, the architecture has to be capable of collecting and processing a large amount of unstructured and semi-structured data sent by urban WSNs. Secondly, it entails avoiding the barriers related with the high heterogeneity and the limited access of the systems and the difficulties in updating them. Finally, the architecture has to be transparent for external WSN providers. This makes it compatible with already deployed networks and with the low processing capabilities of some sensor nodes.

Regarding data processing requirements, this can be considered in a big data context. The architecture has to be ready to collect, index, and process a large volume of data with high velocity and variety. We assume that smart cities have, in their central servers, high computational power, a large amount of storage space, and the other hardware and network requirements necessary to deploy big data solutions.

As seen in Section 2.1, a principal characteristic of smart cities is that their data are highly heterogeneous and distributed. The proposed architecture has to take into account these characteristics and it has to be able to acquire data from diverse source types.

3.2 Architecture overview

This section takes into account the requirements presented in the previous section and outlines a centralized architecture to collect WSN data and to process them with a hybrid detection engine combining a rule-based and an anomaly-based engine to disclose incidents in the third-party WSNs.

The proposed solution, shown in Figure 3.1, is based on an enhanced SIEM system (See Section 2.1.3) contained within the city council facilities in order to make use of the collection, storage, processing and big data services offered by the smart city. The main components and the data flow represented in the figure are:

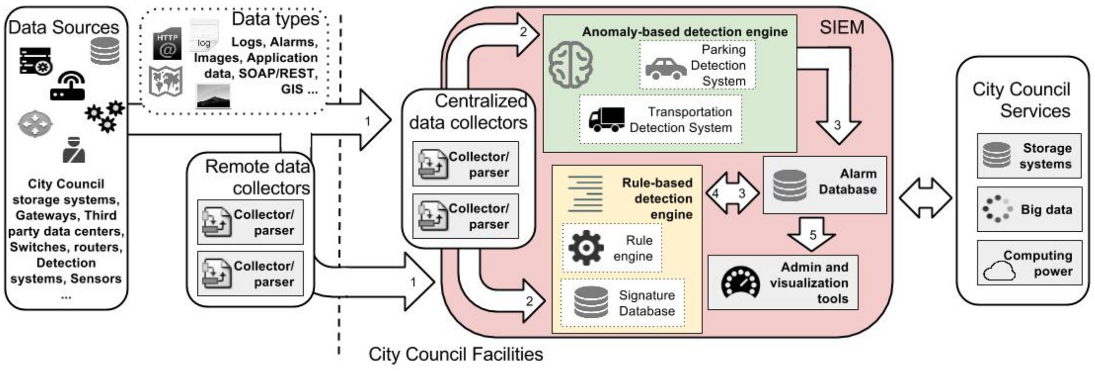

Data Sources
City Council storage systems, Gateways, Third party data centers, Switches, routers, Detection systems, Sensors ...
Data types
Logs, Alarms, Images, Application data, SOAP/REST, GIS ...
Remote data collectors
Collector/ parser
Collector/ parser
Centralized data collectors
Collector/ parser
Collector/ parser
SIEM
Anomaly-based detection engine
Parking Detection System
Transportation Detection System
Rule-based detection engine
Rule engine
Signature Database
Alarm Database
Admin and visualization tools
City Council Services
Storage systems
Big data
Computing power
City Council Facilities
1
2
3
4
5

FIGURE 3.1: Architecture of the proposed solution

1. Data originate from several **sources** in different **data types**. Generally, application data come from the sensor readings and network status data come from gateways, watchdogs[76] or other devices with enough capacity to monitor WSNs. In certain cases, in order to get a precise picture of the network, WSN nodes log system status information, which is sent regularly or on request[77, 78]. These data, then, are gathered, parsed and normalized by **remote data collectors** distributed near the sources, or by **centralized data collectors** installed near the processing engines.

2. Normalized data are the input of the two detection engines. On the one hand, there is the **rule-based detection engine**, the objective of which is to detect known attacks and to correlate data from different sources, and, on the other hand, the **anomaly-based detection engine**, which uses machine learning and statistical techniques for the detection of anomalies and unknown attacks.

3. The detection engines independently analyze the input data and trigger alarms that are stored in a common **alarm database**.

4. Alarms from the database are correlated by the rule-based detection engine generating new alarms, which are more reliable, have higher priority and become candidates for correlation in future iterations.

5. The **administration and visualization tools** offer interfaces (e.g. dashboards, SMS alerts) and subscription mechanisms to provide information about the alarms and to manage the system.

Our solution deploys a new layer in the servers of smart city administrators. This layer is conceptually above the devices used by the different providers. Therefore, it is not affected by the heterogeneity of the different configurations, it does not require special permissions over third-party devices and it is easily accessible and updatable.

Moreover, a SIEM is the core of the architecture and, as seen in Section 2.1.3, this type of system is capable of processing big data as required and it offers mechanisms to collect data from local and remote locations suitable for deployment in a smart city context. Concerning WSNs, data can be found stored in smart city servers or in remote devices on the streets. Sensor readings, for example, are normally of value to specific city departments. These data, therefore, are generally stored in a conventional server and they can be accessed within the smart city premises. However, system status data (e.g. device logs), which may not be relevant to any other smart city department, have to be directly gathered from the WSNs. Therefore, the proposed SIEM-based architecture is capable of gathering remote and local data using different data collectors, and it offers a centralized single platform on which to process and correlate all the information together. Furthermore, data correlation and historical data management, which are generally very relevant in intrusion detection analysis, are especially efficient on this type of platform. Finally, parallel programming paradigms, such as MapReduce, are a normal characteristic of SIEM systems. In Chapter 5, we take advantage of these paradigms to deal with complex intrusion detection algorithms managing large amounts of data in a scalable manner.

Regarding the intrusion detection characteristics of the proposed architecture, two situations are covered. Firstly, the rule-based detection engine is capable of finding patterns in the data to identify attacks that have already been reported in the literature and that are known to information security researchers. Secondly, the anomaly-based detection engine is capable of warning administrators in the case of situations that do not follow normal system behavior, although there are no patterns matching known attacks. In this way, popular attacks can be easily identified and prevented, and new attacks, exploiting unknown vulnerabilities, trigger alarms that give smart city administrators the first warning signs in order to start more in-depth inquiries. Additionally, at the time of creating an alarm, administrators can associate a severity level to the alarm and also assign an action to execute as soon as the alarm is triggered.

The following sections describe the basic WSN data available at the central servers that can be used by the detection engines to disclose security incidents. Then, more details about the two detection engines are given.

3.2.1 Data types

The following are the most relevant types of data that are received from the WSN at the smart city data centers. Some examples of how they can be used to detect intrusions are provided:

- **Basic information about the nodes:** ID, latitude, longitude, etc. The geographical position is a basic parameter to determine the area affected by the attacks.
- **Basic information about the WSN:** service purpose (e.g. parking, environmental monitoring), communication protocol (e.g. ZigBee, LoRa), etc. From this information, other information about the WSN can be extracted. For example, ZigBee has two possible topologies (i.e. tree and star) and its frequency bands in Europe are either 868 MHz or 2.4 GHz. This information can be used to discard possible attacks against a service and to cluster data.
- **Basic information about the packets:** packet number, gateway ID, timestamps, etc. Additional fields can be computed from this basic information, such as the one-way delay (OWD), which indicates the time taken by the packets from the sensor nodes to the server. This is an important field for the detection of some attacks, such as DoS, since these attacks tend to slow down packet reception.
- **Basic information about system status:** received signal strength indicator (RSSI), signal-to-noise ratio (SNR), etc. Some attacks have a direct impact on some of these variables. For instance, attacks generating interference impact RSSI.
- **Information about the services:** sensor readings, service data aggregated in time intervals, etc. These data are sent either in scheduled regular time intervals (e.g. environmental data) or when sensors have reacted to an environmental condition (e.g. parking activity). Anomalies in these data can indicate badly calibrated sensors and data integrity attacks.

- **Battery status:** The objective of many WSN attacks is to exhaust sensor batteries. Therefore, this information is very useful for the detection of this type of attack.

3.2.2 Rule-based detection engine

The rule-based detection engine provides the system with an alarm module capable of identifying attacks that are recorded as signatures in a database. The rules that define the signatures in the database specify the traces that have to appear in the data in order to trigger an alarm. Additionally, alarms are implemented setting up a schedule, a level of severity and the actions to execute (e.g. administrator warning, execution of certain processes).

Rules are built with two purposes; firstly, to find evidence of undesirable situations (e.g. traces of refused connections, parameters surpassing a threshold). Secondly, rules are also built to correlate multiple pieces of evidence found in different network components and/or moments in time. The correlation rules take advantage of the fact that some attacks leave traces in several parts of the system within a limited time window. These traces are normally a consequence of the several steps required to perform an attack or the persistence of the attacker after failing. The following are high-level descriptions of some example alarms:

- This alarm is triggered by a rule that looks for the string “Authentication failed” in a log in real time. The alarm is set with low severity and runs a script.

```
IF ∃ Event E
  FROM Log L ON Real-time
  WHERE Field F
     CONTAINS "Authentication failed"
THEN Alarm(Severity: Low,
          Action: Run Script S)
```

- This alarm is triggered by a rule that checks if a certain field in a log goes over a threshold in real time. The alarm is set with medium severity and warns the administrator with a message in the alarm panel.

```
IF ∃ Event E
  FROM Log L ON Real-time
  WHERE Field F > Threshold
THEN Alarm(Severity: Medium,
          Action: Show in alarm panel)
```

- This alarm is triggered by a rule that correlates events from two logs. The rule looks for situations where the first log does not contain events for the last two hours and the second log contains at least one event in the last hour. The alarm is set as highly severe and sends an SMS to system administrators.

```
IF ∄ Event E1 FROM Log L1 Last 2 Hours
  AND ∃ Event E2 FROM Log L2 Last Hour
THEN Alarm(Severity: High,
          Action: Send SMS)
```

The proposed architecture gathers all the evidence of suspicious behavior in the WSNs of the smart city in a single system. Both detection engines trigger alarms in the case of theoretical undesirable events. However, a large number of these alarms can be classified as false alarms, or they are due to unimportant or transient situations. Therefore, the real challenge is not only to write effective alarms, but also to warn administrators only when the reliability and the severity of the alarms are high enough. This can be achieved by creating alarms based on correlation rules, as Figure 3.1 shows. In this figure, 4 in the data flow schema implies that alarms, which have already been triggered, are used again in the rule-based detection engine. In this way, alarms affecting the same components, area, bandwidth, etc., are correlated, triggering a more reliable alarm. In Chapter 6 a framework is defined to assist administrators in creating correlation rules to increase alarm reliability and to be able to classify security incidents into attack types.

To simplify the definition of rules for an entire smart city, system administrators can use publicly available signature databases. For instance, Snort[1] is a popular IDS that offers regularly updated signature databases for the most common protocols. Quickdraw[2] offers signatures for SCADA. As far as we know, however, there are no signature databases

[1] "Snort", https://www.snort.org/

[2] "Quickdraw", http://www.digitalbond.com/tools/quickdraw/

specifically designed for WSNs or for smart city applications (e.g. parking, environmental monitoring). This complicates the management and the detection of anomalies in contexts with a vast number of incident types.

Moreover, a smart city is a very mutable scenario. Public databases can help administrators maintaining an updated collection of signatures including recently discovered vulnerabilities, new components, network configurations, etc. However, it is still an open problem to find an adequate manner to manage signature databases in large and highly heterogeneous systems like smart cities. The next section presents the anomaly-based detection engine, which can detect unknown attacks and, therefore, can aid administrators in discovering that some signatures are missing or out of date.

3.2.3 Anomaly-based detection engine

The rule-based detection engine provides administrators with a handy tool to identify attacks, looking for the traces that the attacks leave on data. However, this engine has several limitations, which preclude reliance solely on this intrusion detection mechanism for effective attack detection in the context described in this thesis. The principal limitations are:

- The rule-based detection engine is especially useful against attacks that are clearly identifiable through thresholds and which are considered stable in the long term. However, in a changing environment such as the smart city, static thresholds are usually hard to define because the environment is dynamic and changes according to the time of day, the season of the year, the weather conditions, etc.
- Unknown attacks, for which no rules are defined, remain undetected.
- Rules that involve many variables become too complex and are difficult to maintain.
- Finding predefined thresholds for certain variables is sometimes not possible.
- The set of rules defined ad hoc by the administrators requires manual maintenance, which is costly and does not scale well.

In order to overcome these problems, it is essential to complement the rule-based detection engine with other mechanisms. For this purpose, the proposed architecture includes an anomaly-based detection engine. This engine has the responsibility of computing thresholds automatically and training complex machine learning models capable of identifying anomalous data instances due to attacks or failures in the WSNs of the smart city. As seen in Section 3.2.1, data analyzed with this engine include sensor readings, network status logs, etc. Henceforth, the fields in a dataset that can be used for the anomaly analysis are referred as *variables*.

Section 2.4.1 has shown that, in the literature, there are multiple anomaly detection techniques capable of performing analysis on data to disclose these types of situation. Generally, these techniques require a dataset of samples to train models. The models are subsequently used to predict whether new samples are normal or not. Taking this into account, the main characteristics of the projected detection engine can be summarized in the following four points:

1. It uses unsupervised or semi-supervised algorithms.
2. The vast majority of the samples in the training datasets are captured during normal non-attack situations.
3. Training datasets are sampled including observations from most of the different states that can occur in the monitored service.
4. Training datasets are large.

Regarding the first point, as seen in Section 2.4.1, supervised machine learning algorithms require the labeling of each training sample with the class that it belongs to. For these algorithms, in the intrusion detection context, an additional field should be included in the samples indicating whether the sample is normal or anomalous or, in the case of computing a model with one class for each type of anomalous situation, the label should be more specific and indicate the type of situation (e.g. "jamming attack", "selective forwarding attack"). In the smart city context, it is very difficult to gather a large amount of labeled data. Administrators could perform some attack simulations on a testbed or even against the real WSN infrastructure; however, it is unrealistic to systematically gather comprehensive datasets, including many samples from all attacks reported in the literature every time that new models have to be trained. Moreover, attacks that are as yet undiscovered, or that exploit unknown vulnerabilities, would go unnoticed.

Accordingly, the anomaly-based detection engine mainly has to use semi-supervised or unsupervised algorithms.

Regarding the second point, training datasets have to contain a very large proportion of normal samples, because anomaly detection models are normally computed by finding a boundary that encloses most of the samples in the training dataset. If a training dataset contains an excessive proportion of abnormal samples, then it is likely that some of these samples will be included within the boundaries of normality.

Regarding the third point, training datasets have to contain a comprehensive representation of samples from the various possible normal states and situations in the monitored service. Otherwise normal situations not represented in the training dataset can fall outside the computed normality boundaries.

Finally, with regard to the fourth point, training datasets have to be large for the reasons noted in the previous two points. Using only a few samples can result in models trained with too many samples from transient network states or unimportant transitory errors that do not capture the normal behavior of the system. Furthermore, with small training datasets, the proportion of variables relative to the number of samples increases and, therefore, overfitting is more likely to occur [79].

3.3 Designing the anomaly-based detection engine

Taking into account the complexity behind intrusion detection based on disclosing anomalies in the data, this section describes the major considerations to bear in mind when designing the anomaly-based detection engine.

With the purpose of performing a complete and effective anomaly analysis in a smart city, it is necessary to deploy at least two types of algorithms:

- **Univariate algorithms**. With this type of algorithm, administrators can monitor the behavior of a single numeric variable and detect anomalous values falling outside its normal range. For example, detecting an anomalous reading of −50°C in a temperature sensor located in a city with a mild climate.

- **Multivariate algorithms**. With this type of algorithm, administrators can point out anomalies taking into account several variables at the same time, even when each of the analyzed variables stays within its normal boundaries. For example, a reading of 0°C in a temperature sensor can be considered normal in the winter, but it can also be considered as anomalous in the summer if the detection algorithm also takes into account the season of the year, the readings of other temperature sensors, etc.

These are simple examples of typical anomaly analysis. However, we must bear in mind that anomaly analysis in smart cities can be very complex. Anomalies detected by univariate algorithms studying a single variable can provide valuable information for disclosing the source of an incident. For instance, it is widely known that abnormal values in the RSSI can be due to interference [80]. Also, univariate analysis in sensor readings enables the identification of extreme values coming from integrity attacks or sensors that are not properly calibrated. However, there are many attacks against WSN that cannot be directly detected with univariate algorithms, because the attacks have a reduced impact on just one of the received variables. Using multivariate algorithms is more adequate for considering the impact on several variables at the same time, but an alarm triggered by one of these algorithms is generally more difficult to link with the specific causes of the problem. Therefore, intrusion detection in smart city WSNs is not a straightforward task, and cannot be fully automatized and treated like a black-box in which the set of anomaly detection algorithms provided can be used for any context. Administrators have to select the variables to monitor, draw conclusions in the case of alarms, create correlation rules to trigger meaningful alarms that deserve the administrator's attention, etc.

Another difficulty in deploying a system like this is choosing specific algorithms for the different analyses. Regarding multivariate analysis, as it will be seen in Chapter 4, smart city administrators should consider implementing detection based on OC-SVM. This is a semi-supervised algorithm, which gives good detection results in this context. Chapter 4 provides more information about the application of this algorithm and it compares it with other multivariate algorithms.

Regarding univariate algorithms, two types of situations have to be covered depending on the monitored variable:

- Situations where the distribution of the variable is relatively stable in time. In this way, thresholds can be computed from large amounts of relevant values and they can be used to predict anomalies in many samples in the future. In this type of situation, smart city administrators should consider using Tukey's method to define thresholds. Numerous studies in the literature have successfully used algorithms based on Tukey's method to find outliers with good results [81, 82, 83]. This statistical method is adequate in this context because it is simple, has a low level of computational complexity and it does not make assumptions about the statistical distribution of the variable.

- Situations where new values of the variable show strong correlation with immediately previous values. In this type of situation, smart city administrators should consider using autoregressive algorithms, such as ARIMA. Autoregressive algorithms have been widely used to detect outliers in WSN data [60, 65]. These models require very small training datasets, but they have to be recomputed every time a new value is received.

All these types of models, to varying degrees, are not perennial and eventually have to be recomputed to adapt to system changes. The next section provides some indications on how to carry out model maintenance.

3.3.1 Maintenance of machine learning models

The dynamic behavior of cities needs to be taken into account when it comes to keeping models up-to-date. Therefore, it is necessary to recompute the models when these cease to capture the normal behavior of the system.

On the one hand, some types of model are already designed to be transient. For instance, autoregressive models are recomputed after each new observation. On the other hand, other types of model are more durable and suitable for less mutable data. In order to know when these models become out-of-date, it is necessary to compute certain metrics regularly to assess the performance of each model.

Any set of metrics selected for the purpose of evaluating anomaly detection models has to take into account four situations: cases where attacks are not detected (false negatives),

Table 3.1: Metrics to asses anomaly detection algorithms

True positive rate (TPR)	$\frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$
False positive rate (FPR)	$\frac{\text{false positives}}{\text{false positives} + \text{true negatives}}$
F-score	$\frac{\text{true positives}}{\text{true positives} + (\text{false negatives} + \text{false positives})/2}$

cases where the algorithms incorrectly point out attacks that have not occurred (false positives), cases where the attacks are correctly detected (true positives) and cases that are correctly identified as not being under attack (true negatives). Taking these into account, the metrics shown in Table 3.1 have been widely used to assess IDSs and machine learning algorithms [58]. The true positive rate (also known as detection rate, sensitivity, or recall) measures the percentage of attacks that have been properly detected. The false positive rate (also known as the false alarm rate) indicates the percentage of normal samples misclassified as attacks. Finally, the f-score is used as a general overview of the performance of the algorithm. This metric takes into account the number of true positives over the arithmetic average of predicted positives and real positives.

Using these metrics, administrators can establish limits, beyond which the models have to be considered out-of-date and must be recomputed. In order to compute these metrics and establish the limits, it is important to have labeled test datasets. The predictions for each sample in a test dataset have to be compared with the labels indicating the actual class of the sample, resulting in one of the four cases mentioned above: false negative, false positive, true negative or true positive. The purpose of these labels must not be confused with the labels required to train models with supervised algorithms, which require much larger datasets. To compute these metrics, shorter datasets can be used. The following are examples of methods that can be used to label a small dataset:

- Perform attacks against real smart city WSNs under supervised situations, against testbeds or in simulators.
- Deploy honeypots to entice attackers to attack the system and monitor their activity.
- Use a system to manage alarms in which administrators can mark alarms as false positives or true positives.

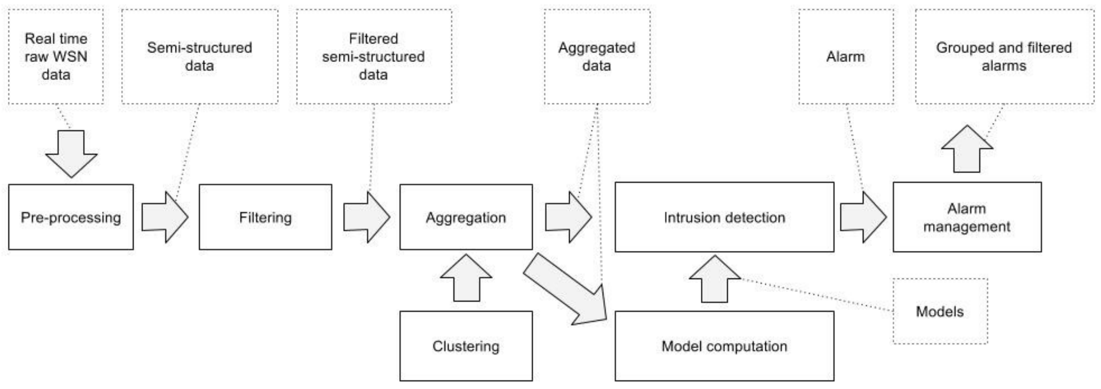


FIGURE 3.2: Pipeline describing the general process to detect intrusions from smart city WSN data.

As seen in these sections, intrusion analysis involves several steps, such as training models or predicting anomalies in new observations. The next section presents a pipeline with the necessary steps and the subprocesses involved in the intrusion analysis with the proposed architecture.

3.4 Intrusion analysis outline

Figure 3.2 shows a pipeline including the necessary subprocesses to process smart city WSN data from their arrival at the servers until an alarm can be triggered. First, when data reach the servers they have to be **preprocessed** in real time. Afterwards, data can be **filtered** and **aggregated**. Data aggregation can be performed by several criteria. A usual criterion is according to some previously defined **clusters**. Preprocessed, filtered, and aggregated data are used to **compute models**, which will later be used to detect anomalies in order to disclose attacks and other failures in an **intrusion detection** subprocess. The alarms triggered by the anomaly detection analysis are processed by an **alarm management** subprocess in order to correlate them, to reduce the amount of false positives and to warn administrators only in the case of relevant situations. Below each of these subprocesses will be briefly described.

3.4.1 Preprocessing

As a first step before performing any type of analysis, it is necessary to conduct some kind of data transformation in order to ease subsequent operations:

- **Parsing:** messages enter the system with multiple formats (e.g. string containing several variables separated by #). These data need to be parsed and transformed into at least a semi-structured data type (e.g. JavaScript object notation (JSON)) linked with some metadata to give context to the different fields.

- **Indexing:** messages enter the system in a disorderly way, but they contain a field with its creation timestamp. In order to ease some other preprocessing operations that require previously received messages and also to ease subsequent subprocesses, messages have to be indexed according to a temporal component. WSN data do not arrive perfectly in order because:

  – Some sensors store several readings in the memory, which are sent jointly after a certain amount of time has elapsed.

  – Providers have several alternatives for sending data from WSN to smart city central servers (e.g. via municipal telecommunication networks or via private infrastructure through servers owned by the providers). The different alternatives generate different delays between the initial data transmission and its reception at the final destination.

  – Within a single multi-hop WSN, packets can take different routes, which results in different delays.

  – Transmission errors necessitate the resending of packets, which increases message delays.

- **Simple variable creation:** new variables are created applying simple functions to aggregate or transform one or more fields from the current message (e.g. unit conversion).

- **Complex variable creation:** new variables are created applying complex functions to aggregate or transform one or more fields from the current message or from a previous message (e.g. computing battery consumed since last received message).

3.4.2 Filtering

In this subprocess, data are selected by standard filtering operations, such as comparing variables with user defined values or with other variables. In this way, administrators can extract just the important samples from the plethora of data that arrive to the system. For example, administrators can set filters to get data only from certain sensors, from a certain area, etc.

3.4.3 Clustering

The principal purpose of this subprocess is to create certain divisions in data in order to draw meaningful conclusions after anomaly analysis. The divisions can be created ad-hoc by system administrators, or using clustering algorithms. The purpose of this subprocess is twofold. Firstly, using clusters, nodes that can be affected by the same attacks can be grouped and analyzed together (e.g. neighboring nodes transmitting in the same frequency band). Secondly, search space gets reduced, which simplifies finding the root cause of security incidents.

3.4.4 Aggregation

This subprocess is responsible for combining data extracted in the filtering subprocess, in such a way that data stay grouped according to a certain criterion, such as time intervals, clusters, etc. Typical data aggregation operations are: the minimum, the maximum, the sum, the mean, the median, the mode, etc. For instance, summarized information from an environmental WSN can be periodically obtained by computing the minimum, the maximum and the mean of the sensor readings.

3.4.5 Model computation

As previously seen, anomaly detection is based on using previously computed models to predict whether an observation is normal or not. These models can be very simple, such as thresholds, or complex, such as machine learning models. In both cases, the cost of computing models is normally not insignificant. Therefore, models must already be computed prior to their usage in real-time applications.

3.4.6 Intrusion detection

Intrusion detection is performed on preprocessed, filtered, clustered and/or aggregated data using either the rule-based detection engine or the anomaly-based detection engine. As previously seen, the former uses attack signatures to look for traces of attacks, and the latter uses the models computed beforehand to predict whether new data has to be considered abnormal.

3.4.7 Alarm management

The intrusion predictions reported in the previous step are used by an alarm management subprocess to create alarms and warn administrators. This subprocess can use variables such as WSN criticality or a degree of abnormality extracted from certain anomaly detection algorithms to sort alarms by importance. Furthermore, administrators can correlate alarms with the rule-based detection engine in order to concentrate several alarms into a single and more relevant alarm, to look for faults in various networks at the same time, etc.

3.5 Use case: attack on a parking WSN

This section shows how the proposed architecture can be applied to a smart city scenario using a use case based on a public car park. To build this use case, we used Castalia 3.3[3] to implement typical WSN configurations and to simulate several parking scenarios. We also implemented popular attacks against these network configurations. In this way, we used the resulting simulated data in a prototype of an intrusion detection platform following the architecture described in this chapter. The SIEM Splunk was the core of the prototype: its indexing tools were used to store the data, its alarm module was used as the rule-based detection engine, and two new modules to train and test OC-SVM were developed, within the SIEM Splunk, as the anomaly-based detection engine.

This use case shows the usage of the proposed architecture following the main steps in the pipeline described in Section 3.4. In this way, this use case provides a proof of concept of the architecture, it shows the feasibility of implementing and using it to detect attacks in

[3]"Castalia", http://castalia.npc.nicta.com.au/

Table 3.2: Summary use case scenarios

Scenario	parking sensors	Other sensors	MAC	Topology
1	9	2 CO_2, 3 light, 1 mass, 2 humidity	802.15.4	Star
2	9	2 CO_2, 3 light, 1 mass, 2 humidity	TMAC	Star
3	30	2 CO_2, 3 light, 1 mass, 2 humidity	802.15.4	Star
4	30	2 CO_2, 3 light, 1 mass, 2 humidity	TMAC	Star
5	100	None	802.15.4	Star
6	100	None	TMAC	Star
7	100	None	802.15.4	Tree

typical WSNs, such as the car park, and it shows how data from all the communication layers can be combined for effective detection of different types of attack in WSNs.

3.5.1 Scenarios

In order to build our scenario, first we studied the dynamics of the data sent by the sensors of different services in Barcelona and also the information fields included in the packets by the providers. Additionally, the most common WSN configurations and protocols were taken into account. As a result, we configured seven different parking WSNs. Table 3.2 shows a summary of the scenarios. As it can be seen, to capture the variability of layouts and the diversity of technologies in the smart city, the networks were designed relying on two types of data link protocol (802.15.4 and TMAC) and two types of topology (star and tree). The number of parking sensors varies in each configuration between 9 and 100 sensors. In addition, in scenarios 1 - 4, sensors from a miscellany of applications (environmental monitoring, light, and mass in a container) shared a single WSN with different sending behaviors. While parking, light and mass sensors were reactive, CO_2 and humidity sensors were programmed to send readings at regularly scheduled intervals.

These scenarios were implemented in the WSN simulator, Castalia. As a result of the simulations, a dataset was generated containing data aggregated in 2-hour intervals. The resulting dataset included application data (e.g. amount of times that each parking slot was used within each time interval) and also variables from the other communication protocol layers (e.g. number of packets not received due to interference).

3.5.2 Attack model

We used Castalia to simulate various scenarios under normal circumstances. We then also used this WSN simulator to implement and simulate several attacks against the networks. In this way, test datasets were generated to evaluate the performance of the proposed detection algorithms.

In this simulation, we assumed that attackers intend to gain advantage over other users of the public parking spaces. To this end, attackers try to disrupt communication between several nodes during high occupancy hours. Thus, parking applications cannot receive updates when parking slots become free and, therefore, attackers have a higher probability of finding available slots in certain areas.

In order to disrupt communication, we conducted three different attacks aiming at the three first layers of the communication stack (i.e. physical, data link and network). In WSNs, these layers are the most vulnerable ones, specially in smart cities, where the sensor nodes lay unprotected on public places and, therefore, the nodes and the communication medium are easily accessible. The conducted attacks are:

- **Jamming.** This attack at the physical layer consisted of sending a high-power signal to the gateway in order to corrupt legitimate packets.
- **Unfairness.** At the data link layer, the attackers exploited the channel access protocols to prevent transmissions from legitimate nodes. In 802.15.4 configurations, legitimate nodes use clear channel assessment (CCA) to check whether the channel is free before transmitting. The attackers continuously occupied the communication channel impeding other transmissions. In TMAC configurations, the attackers corrupted reference control packets used by legitimate nodes to initiate transmission.
- **Selective forwarding and blackhole.** In these attacks at the network layer, the attackers captured a node that stopped retransmitting certain packets from some nodes (selective forwarding), or from all of them (blackhole).

3.5.3 Intrusion detection process

The following sections briefly describe how the steps in the pipeline presented in Section 3.4 were taken to process the data.

3.5.3.1 Preprocessing, filtering, clustering and aggregation

In this use case, the preprocessing, the filtering, the clustering, and the aggregation subprocesses were done in an ad hoc and simplified manner. Firstly, it must be taken into account that the results of a Castalia simulation are variables related to the communication status aggregated during a time interval of 2 hours in this case. Therefore, preprocessing and aggregation are carried out by the simulator automatically. Secondly, the simulation was designed including only the relevant nodes from a small area sharing the 2.4 GHz bandwidth. Therefore, the scenario was already designed creating a single cluster by area and bandwidth. Filtering was also simple in this case, since for each computed model, filters consisted of sub-selecting the required variables.

3.5.3.2 Model computation

An anomaly-based detection engine was implemented using OC-SVM from the scikit-learn 0.15.2[84] library. In order to train models, we implemented a new custom command in Splunk: *svmtrain*. A subset of samples from the attack-free dataset was used to train the models. Figure 3.3a shows the usage of this command to train a model using training data from Scenario 7. For each of the scenarios we trained an OC-SVM model with a radial basis function (RBF) kernel, which we stored within the Splunk server.

When available, the selected variables from the training dataset included: the time of day; the number of state changes (free/occupied) per sensor; the percentage of lost application packets per sensor; the number of ACK, CTS and RTS packets sent by the gateway; and the number of packets received with and without interference. These system status variables were received and recorded in the base station using current protocols implemented in Castalia. Therefore, these variables can easily be accessed and sent to the smart city servers without having to redesign and deploy new protocols in the sensor nodes.

```
index=20150325-100nodes802154-multihop sim_type=train
| fields - _* | fields + "hour", *_appsend, *_applossrate
| svmtrain file_name=scenario7 nu=0.01 gamma=0.01
```

(A) Example of *svmtrain* command to compute an OC-SVM model. The training dataset corresponds to Scenario 7 in Table 3.2 including variables: hour of the day, number of sent application packets and loss rate for each node.

```
index=20150325-100nodes802154-multihop sim_type=test
| fields - _* | fields + "hour", *_appsend, *_applossrate
| svmtest file_name=scenario7
```

(B) Example of *svmtest* command to predict if each sample in a test dataset is anomalous. The test dataset contains samples from Scenario 7 in Table 3.2 including variables: hour of the day, number of sent application packets and loss rate for each node.

FIGURE 3.3: Training and test custom command examples

3.5.3.3 Intrusion detection

This section explains how we used the two detection engines to disclose intrusions.

Rule-based detection engine

In this use case, we implemented rules in Splunk's alarm mechanism to look for traces of missing received data from the sensors that regularly send their readings. For instance, Figure 3.4a shows a rule that checks whether the environmental sensor reading data programmed at 19:00 was properly received. If this reading was not received, then a medium-severity alarm was triggered.

Anomaly-based detection engine

As previously mentioned, a major problem of a rule-based detection engine is the infeasibility of detecting unknown attacks. Therefore, we used the models trained with the custom command *svmtrain* to detect attacks that, unlike the examples from the previous section, do not leave clear or easily identifiable traces in the data.

In order to make predictions with the trained models, we implemented another custom command: *svmtest*. The datasets used to test the models included an equal proportion of instances from the attack-free data that were not used to train the models and instances from the simulation with attacks. Figure 3.3b shows the usage of *svmtest*. With this command, we first load the model previously computed with *svmtrain*, then, for each sample in the test dataset from Scenario 7, we predict if the sample is normal or anomalous.

```
index="parking_simulation" host=20150113-9sensors802154-2h
| eval count_co2_sent = '10_appsend' + '11_appsend' + '12_appsend' | search count_co2_sent<3
```

(A) **CO_2 not received:** Medium-severity alarm scheduled at 20:00 to verify that the regular CO_2 readings from 19:00 have been received.

```
index="parking_simulation" host=20150113-9sensors802154-2h  | fields - _*
| fields + "hour", "0_radiopckfailedNoInt",
"0_radiopckfailedInt", "0_radiopckfailedBelowSens", "0_radiopckfailedNonRX",
"0_radiopckreceivedWithInt", "0_radiopckreceivedWithoutInt", *_appsend
| svmtest file_name=20150113-9sensors802154-2h | search test=-1
```

(B) **SVM outlier:** Medium-severity alarm to identify outliers in real time using a trained OC-SVM within a two-hour window.

```
index=_audit action=alert_fired | eval ttl=expiration-now() | search ttl>0
| eval is_not_co2=if(ss_name=="Scenario 1 - CO2 Not Received",1,0)
| eval is_svm=if(ss_name=="Scenario 1 - SVM outlier",1,0)
| stats sum(is_not_co2) as sum_alert1 sum(is_svm) as sum_alert2
| eval num_alerts=sum_alert1+sum_alert2 | search num_alerts>1
```

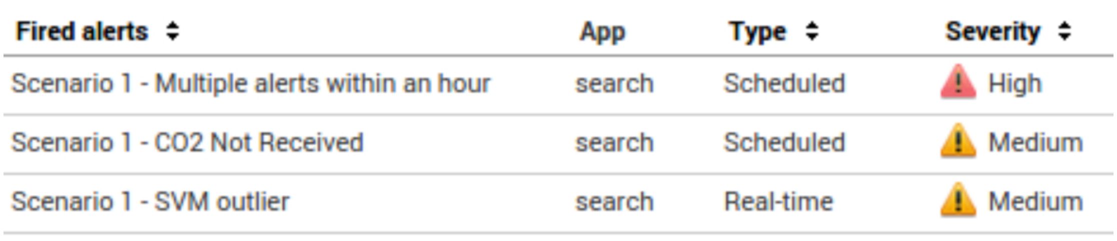
(C) **Multiple alerts within an hour:** High-severity alarm to detect that several pieces of evidence are affecting the network within a one-hour window.

FIGURE 3.4: Examples of rules defining alarms

In this use case, for each of the 7 scenarios, we trained an OC-SVM model and we also implemented a medium-severity alarm triggered when the anomaly-based detection engine discovers an anomaly using the trained model. Figure 3.4b shows the rule implementing the alarm for Scenario 1.

3.5.3.4 Alarm management

As seen in the previous section, the rule-based and the anomaly-based detection engines trigger alerts when certain conditions are met. In order to warn administrators only in reliable situations, we created correlation rules to join the alerts from the two engines. In Figure 3.4c, we defined a correlation rule that triggers a high severity alarm when multiple alarms are triggered within a 1 hour window. Figure 3.5 shows Splunk's alarm panel after triggering several alarms using the test dataset from Scenario 1. As the figure shows, first the anomaly-based detection engine detected an anomalous circumstance in real time and triggered an alarm. Next, a rule looking for missing data triggered an alarm when the CO_2 readings were not received in their scheduled time. Finally, a correlation rule that regularly checks if there are several alarms in the last hour triggered a highly severe alarm.



Fired alerts	App	Type	Severity
Scenario 1 - Multiple alerts within an hour	search	Scheduled	⚠ High
Scenario 1 - CO2 Not Received	search	Scheduled	⚠ Medium
Scenario 1 - SVM outlier	search	Real-time	⚠ Medium

FIGURE 3.5: Alarms in the alarm panel in Splunk

3.5.4 Results and discussion

This section discusses the benefits of the proposed architecture, which combines rule-based and anomaly-based detection. Firstly, detection results of the anomaly detection process with OC-SVM will be analyzed. Then, the positive effects of bringing together the alarms from the two detection engines with correlation rules will be evaluated.

To evaluate the detection results of the anomaly-based detection engine, we calculated the three standard metrics shown in Table 3.1 on the test dataset containing instances with and without attacks. As Table 3.3 shows, this evaluation yielded high detection rates in all the scenarios.

It is worth noticing that all available variables were included in the detection models. This is important because the impact on the variables is different depending on the type of attack. For example, the jamming attack has a clear impact on physical layer variables and the blackhole attack has no impact on them. Implementing feature selection methods that systematically reduce the number of variables included in the models can have a negative impact on the detection of certain attacks that are unknown at the time of designing the feature selection methods. Therefore, this shows that OC-SVM can be used, including variables from all communication layers and that detection results for different types of attack, network configuration and communication protocol are generally good.

However, the false alarm rate in the scenarios with more nodes is slightly high. As it can be seen in the tree topology, the lack of information about widely separated nodes at the base station reduces the detection rate when a small percentage of the nodes are affected by the attacker (i.e. selective forwarding), and it increases the false alarm rate.

In order to reduce the number of false alarms, correlation rules can be implemented to look for multiple alarms affecting the same area or the same components. Since

Table 3.3: Detection results of the anomaly-based analysis in the use case

Scenario	Attack	Detection rate (%)	False alarm rate(%)	F-score (%)
1	Jamming	98	2	97.51
1	Unfairness	97.5	2	97.26
2	Jamming	98.5	3.33	96.81
2	Unfairness	99	3.33	97.06
3	Jamming	99	3.67	96.82
3	Unfairness	99	3.67	96.82
4	Jamming	100	3.67	97.32
4	Unfairness	99.5	3.67	97.07
5	Jamming	100	4.91	93.14
5	Unfairness	95.56	4.91	90.89
6	Jamming	100	13.43	83.24
6	Unfairness	90.84	13.43	78.61
7	Blackhole	100	14.54	82.29
7	Sel.forward.	82.78	14.54	73.31

WSNs share the radio spectrum and sometimes they even share network devices, signs of abnormal behavior in various networks in the same area can be caused by a single attack. In this use case, we correlated the alarms from a rule that was regularly checking for missing data from the CO_2 WSN with the alarms triggered by the anomaly-based detection engine analyzing the other nodes. An alarm from this correlation rule can be set as highly severe. Although CO_2 sensor nodes sometimes transiently fail to communicate with the base station, this can be considered normal in WSN and it is not enough reason to warn administrators; similarly, the anomaly-based detection engine triggers a certain amount of false alarms. However, the chance of these two circumstances occurring together, in a short time interval, becomes unlikely.

3.6 Conclusions

In this chapter, we have proposed an architecture, which provides tools to ease the complex problem of disclosing intrusions in a large and heterogeneous environment such as the smart city.

We have seen that traditional security needs to be enhanced in order to detect anomalies in smart city WSNs operated by third parties. The reduced access to the service provider network devices limits the visibility of the WSNs to smart city administrators, and prevents a conventional security analysis. To overcome this, we have proposed a non-intrusive architecture that combines a rule-based and an anomaly-based detection engine. This architecture deploys a new security layer in the central servers above the miscellaneous equipment of the providers. Thus, problems due to the heterogeneity, limited access, or updating difficulties of certain devices are avoided. The proposed architecture is compatible with the already deployed infrastructure, as it does not add any requirements to the existing infrastructure. Additionally, we described a pipeline with the necessary subprocesses to process WSN data and disclose intrusions using the proposed architecture with conventional anomaly detection techniques.

Furthermore, we have implemented a prototype of the architecture on top of Splunk and we have presented a use case structured around a public car park that shows the benefits of combining the two types of detection engine. On the one hand, the anomaly-based detection engine (implemented with OC-SVM) is capable of detecting unknown attacks and its unsupervised learning nature provides it with flexibility in a changing environment like the smart city. In the use case presented in this chapter, OC-SVM included variables from several communication layers and showed good detection results for various attacks aimed at different vulnerabilities affecting the most important layers. Nevertheless, in some situations, anomaly-based detection triggers excessive false alarms.

On the other hand, the rule-based detection engine does not have as much flexibility as the anomaly-based detection engine, but it triggers highly reliable alarms. In the use case, we implemented several rules verifying the correct arrival of WSN data with a regular behavior. Additionally, we created correlation rules joining the alarms generated by both detection engines. This increased the reliability of the anomaly-based alarms and allowed system administrators to be warned in the case of more important situations.

We have seen that intrusion detection in the smart city is a very complex problem. A black-box solution with a multipurpose detection algorithm that covers most of the attacks for most of the configurations is not feasible. To tackle intrusion detection in this context, it is first necessary to select the most adequate algorithms. In this chapter, we have indicated some suitable algorithms for implementing anomaly analysis in the context of the smart city. With these indications, this thesis contributes to simplifying the system administrator's task at the time of setting up the anomaly-based detection engine. Chapter 4 extends the study of some of these algorithms by comparing several multivariate anomaly detection techniques. Secondly, it is necessary to select the variables that will be used with each algorithm and that will trigger the first alarms pointing to possible security problems. Thirdly, it is necessary to create correlation rules in order to group alarms to identify those that are meaningful and therefore require the attention of system administrators. Chapter 6 provides more details about correlation rules.

Chapter 4

A comparative study of anomaly detection techniques

One of the most important components of the architecture proposed in the previous chapters is the anomaly-based detection engine. Section 3.3 has shown that this engine can be implemented using several types of algorithms, being multivariate anomaly detection an essential technique to disclose intrusions by taking into account the behavior of several variables at the same time. Although there are many studies analyzing the behavior of multivariate techniques in other contexts, as far as we know, there is no study analyzing these techniques with WSN data in a smart city scenario from the point of view of smart city administrators.

Hence, the study presented in this chapter has the main goal of comparing four semi-supervised multivariate anomaly detection techniques under the point of view of smart city administrators. This study evaluates the techniques taking into account scenarios with different available network status variables. It is also a main objective of this study to find the minimum amount of network status variables that providers have to send to the smart city data centers in order to be capable to perform a proper centralized anomaly detection analysis.

The rest of this chapter is structured as follows: Section 4.1 introduces the techniques used in this comparative study. Section 4.2 explains the simulation and the experimental procedure. Section 4.3 contains the results of the study. Finally, Section 4.4 concludes the chapter.

4.1 Description of anomaly detection techniques

This section describes the anomaly detection techniques that will be used in this comparative study. The compared techniques are the most frequently used methods in the literature for this purpose, and they are based on Mahalanobis distance, local outlier factor, hierarchical clustering and OC-SVM.

In the smart city context, it is not possible to assume that data from all the possible attack categories are available to create comprehensive training datasets (e.g. some attacks are unknown until new vulnerabilities are disclosed). Accordingly, supervised techniques are not suitable in this context. Regarding semi-supervised and unsupervised techniques, it is also necessary not to base the attack detection on the previous knowledge of the problem. For instance, some statistical techniques assume specific distributions of the data. However, this is unknown in many smart city services and sometimes data behavior is variable depending on the time of day, the season of the year, the weather conditions, etc.

Apart from the aforementioned techniques, we have considered other more recent methods from the area of machine learning. For example, algorithms based on random forests [85] have been used successfully in many scenarios of different domains, but their current popularity has not reached the levels of support vector machines. Another family of algorithms that is certainly worth considering is deep learning. In this regard, recent advances show that this is a very promising field. As an example, algorithms based on deep belief networks [86] convolutional neural networks [87] or recursive neural networks [88] have been used successfully in several scenarios to improve the performance obtained with previous techniques. In the area of anomaly detection, additionally, the authors of [89] have used deep learning in combination with other techniques to identify outliers, and they have obtained promising detection results.

However, the use of deep learning for anomaly detection is a research field still in its infancy. Deep learning techniques, in general, require costly training processes, which is something easily attainable in fields such as computer vision, speech recognition, etc. However, in the case of smart city WSNs, obtaining large training datasets is much more complex due to their dynamic nature. This dynamic behavior involves frequently retraining the models generated by the machine learning algorithms, thus making it difficult to work with huge training datasets and apply the deep learning techniques successfully. Therefore, for these reasons, the study of deep learning techniques in this scenario is beyond the scope of this thesis.

4.1.1 Mahalanobis distance

Mahalanobis distance measures the number of standard deviations that an observation is from the mean of a distribution. This measure can be used to detect outliers in multivariate data, because outlier observations do not have normal values in one or more dimensions. [90] surveys outlier detection methodologies and compares Mahalanobis distances with other proximity-based outlier detection techniques.

4.1.2 Local outlier factor

Local outlier factor (LOF) is a degree measuring the isolation of a point in a vector space with respect to its neighbors [57]. In order to compute this degree of isolation, LOF is based on the concepts of reachability distance and local reachability density (lrd). The reachability distance (Equation 4.1) between two points p and o is the maximum value between the distance between p and o and the farthest distance between o and its k nearest neighbors (k-distance). The lrd for point p is the inverse of the average reachability distance between p and its $MinPts$ neighbors, where $MinPts$ is a parameter of the algorithm. Equation 4.2 shows the lrd formula, where $N_{MinPts}(p)$ is the $k - distance$ neighborhood of p with $k = MinPts$, which is a set including the points that have a distance to p equal or lower than $k - distance$. Finally, the LOF (Equation 4.3) computes the average ratio of the lrd of p with the lrd of its k neighbors. LOF values smaller than 1 indicate high densities, LOF values greater than 1 indicate low densities and LOF values close to 1 indicate average density spaces. Outliers are considered to be in low density regions.

$$reach\_dist_k(p,o) = max\{k\_distance(o), distance(p,o)\} \tag{4.1}$$

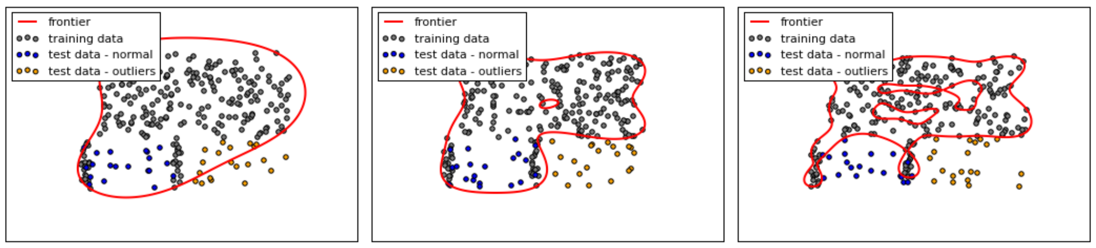
$$lrd_{MinPts}(p) = \frac{1}{\sum\limits_{o \epsilon N_{MinPts}(p)} reach\_dist_k(p,o)/|N_{MinPts}(p)|} \tag{4.2}$$

$$LOF_{MinPts}(p) = \frac{\sum\limits_{o \epsilon N_{MinPts}(p)} lrd_{MinPts}(o)/lrd_{MinPts}(p)}{|N_{MinPts}(p)|} \tag{4.3}$$

In [57], the authors of LOF suggest a lower and an upper bound for the k value. The lower bound for k can be considered as the minimum amount of nearby points that can mark out a more isolated nearby point as an outlier. It is considered good practice to select a k higher than 10 to remove unwanted statistical fluctuations. Conversely, the upper bound for k indicates the maximum number of nearby points that can potentially be considered outliers. A group of $k-1$ or less nearby points require other points in the vector space to have k points to compute the LOF. This implies that the LOF values for the points in the group increases and becomes similar to the LOF of the isolated points. Therefore, either some isolated points are considered normal or the points in the group are considered outliers. In their experiments, the authors of LOF indicate that the algorithm performs well selecting values of k between 10 and 20.

4.1.3 Hierarchical clustering

Hierarchical clustering is a type of analysis that aims to partition a dataset in groups of data (i.e. clusters) according to a similarity measure and creating a tree-based structure that eases the anomaly analysis. This clustering analysis is performed using two types of approaches: top-down or bottom-up [91]. In this thesis we focus on agglomerative hierarchical clustering, which is a bottom-up approach, where initially each sample of the dataset falls in a different cluster and, in each step of the algorithm, two clusters are selected according to a similarity measure and combined in a new cluster. This process ends when there is only one cluster that includes all the samples. A common similarity measure can be computed using the Euclidean distance in Ward's method [92]. With this method, two clusters with the minimum average distance from any sample in one cluster to any sample in the other cluster are merged in each step.



(A) $\nu = 0.01$, $\gamma = 0.2$ (B) $\nu = 0.01$, $\gamma = 0.8$ (C) $\nu = 0.01$, $\gamma = 2$

FIGURE 4.1: RBF kernel OC-SVM trained with different parameters. In *a*, the influence area of the support vectors is wide, so many outliers are incorrectly classified as normal. In *b*, the frontier is very near the support vectors, so there is a reduced number of misclassified outliers and all the normal test samples are properly classified. In *c*, the model is overfitting the training data, so many normal test samples are classified as outliers.

Agglomerative hierarchical clustering can be used to compute outlier ranking factors for the samples in the dataset. Outliers are theoretically more dissimilar to other observations and they should be more resistant to be merged in a new cluster. Thereby, various methods have been proposed to obtain the outlier factors with this type of clustering, such as *linear*, *sigmoid* or *sizeDiff* [93].

4.1.4 Support vector machines

Classification techniques based on support vector machines (SVM) have proven to be effective in several contexts related to intrusion detection [94, 95]. Basically, classification techniques based on machine learning require two steps. First, a dataset is used to train a learning model. Then, the trained model is used to classify new data samples. Several features define each sample of the datasets. The SVM classification process represents the training dataset in a n-dimensional vector space, n being the number of features of the training data. Then, it defines a hyperplane (i.e. a $n-1$ dimensional plane) that separates (with a maximum margin) the samples from the different classes. The support vectors are the subset of training samples that are near the hyperplane and that define it. Finally, the hyperplane acts as a frontier to classify other samples.

In this thesis, we use OC-SVM, which are a special case of semi-supervised SVM that do not require attack labeled data. OC-SVM build a frontier to classify new samples as normal or outlier. In SVM, different types of kernel functions are available to build the most adequate hyperplane for each application. In this work, we use a RBF kernel, which can learn complex regions [52]. In order to build the frontier, RBF kernel OC-SVM use basically two parameters[84]. On the one hand, ν defines the maximum fraction of outliers present in the training data. On the other hand, γ establishes the influence area of the support vectors on the classification. Figure 4.1 exemplifies the impact on the learned frontier that γ has for a fixed value of ν. As shown in Figure 4.1b, increasing the value of γ implies adjusting the frontier closer to the training samples. This reduces the number of misclassified outliers as normal samples. However, Figure 4.1c shows that increasing γ too much causes the training data to be overfitted. A usual approach to select optimum parameters is grid search [96]. This method uses a grid with parameter values that is exhaustively explored in order to select the values that give the best performance of the SVM over a set of samples.

4.2 Simulation and anomaly detection analysis

This section contains the core steps of this comparative study. First, the challenges of gathering the necessary data to perform security studies in smart cities are reviewed. Section 4.2.1 discusses the main challenges and, Section 4.2.2 explains how we overcome them in this study, besides providing a brief description of the entire procedure taken to evaluate the different anomaly detection techniques. The other sections contain information about the different steps in the analysis: the data collection (Section 4.2.3), the simulation (Section 4.2.4), the feature selection (Section 4.2.5) and the anomaly analysis (Section 4.2.6).

4.2.1 Smart city security simulation challenges

As previously seen, smart cities can be considered to be very heterogeneous scenarios, where many technologies, applications and different suppliers coexist. Thus, the implementation of software simulators that realistically reflect the complexity of smart city WSNs is very complicated. In recent years, simulators have been used to test new protocols and to assess the security techniques that protect simple WSNs in very specific contexts [97, 98]. OMNET++ [99], Castalia [100], Cooja [101] and NS-2 [102] are among the most popular WSN simulators.

From a technological perspective, replicating WSN configurations from different providers to simulate several smart city scenarios is a very arduous task. This is motivated by the extensive variety of existing hardware on the market and the wide availability of communication protocols for WSNs. Furthermore, although some of the previously mentioned simulators implement realistic signal propagation algorithms, none does so with a model that can take into account complex urban components, such as walls, traffic, etc. Moreover, simulators also lack realistic event generation engines to reproduce the dynamics of the citizens and the other elements interacting with the urban WSN. For example, Castalia offers multiple distribution functions to simulate the events sensed by the sensors. Nonetheless, selecting the appropriate distribution and modeling the appropriate behavior for the different applications is complex and can lead to unrealistic conclusions. Recently, the authors of CupCarbon [103] proposed a simulator for an easy integration of WSNs and elements of the IoT in smart cities. However, this simulator, which is intended as a supplement to other simulators, is still immature and it does not implement all the layers of the communication stack.

Performing simulations to test security components poses additional difficulties. On the one hand, reproducing computer attacks requires a high technological expertise and a high level of investment in manpower. On the other hand, many attacks exploit unknown vulnerabilities and, therefore, they are not a priory replicable in controlled simulated environments.

Moreover, the testing of security issues in controlled contexts using real hardware is complicated in a smart city. For example, many WSN applications in the cities cannot be easily deployed at a similar scale in a realistic testbed because they would require an infrastructure as big and dynamic as a city. In addition, attack tests in operational WSN are generally incompatible with some application requirements (e.g. 24/7 availability) and can be detrimental to third parties (e.g. jamming attacks to ZigBee can also provoke interferences to Wi-Fi users).

Therefore, it is necessary to combine the large amount of data that is already gathered by smart city providers on the streets with the use of existing simulators to evaluate the consequences of computer attacks and to determine the most appropriate intrusion detection techniques and the appropriate security procedures to resolve these issues.

4.2.2 Experimental procedure

In order to overcome the barriers discussed in the previous section, we use real data from deployed services in Barcelona to feed a WSN simulator that will generate data following realistic patterns. Running experiments in the simulator provides the flexibility necessary to test different communication protocols and network configurations, and it is also a safe way to execute computer attacks. This section presents how we use this mechanism to collect a smart city WSN dataset with and without attacks, and how we compare four anomaly detection techniques to detect intrusions based on Mahalanobis distance, local outlier factor, hierarchical clustering and OC-SVM. We also compare the performance of these techniques under different amounts of available network status information, taking into account three different levels of permitted false positive rates.

The pipeline in Figure 4.2 shows a general picture of the complete process of the analysis. This process consists of the following steps:

1. Data collection: raw sound data are gathered over a period of 14 days from the streets of Barcelona (Section 4.2.3).
2. Simulation (Section 4.2.4):
   (a) Raw data are used in a simulator to generate WSN data with comprehensive information about all the communication layers. The simulation is executed multiple times (one time without including any attack and one time for each of the attacks), resulting in a dataset containing samples with and without attacks.
   (b) The simulation data are aggregated in time intervals.
3. Feature selection: the features of the dataset are filtered according to those available at the simulated WSN (Section 4.2.5). As previously stated, the main goal of this comparative study includes minimizing the amount of network status information required to detect anomalies. Thus, the features are selected taking into account the simulated availability of network status information.
4. Anomaly analysis (Section 4.2.6). One of the available detection techniques is selected and we proceed with the following sub-steps:

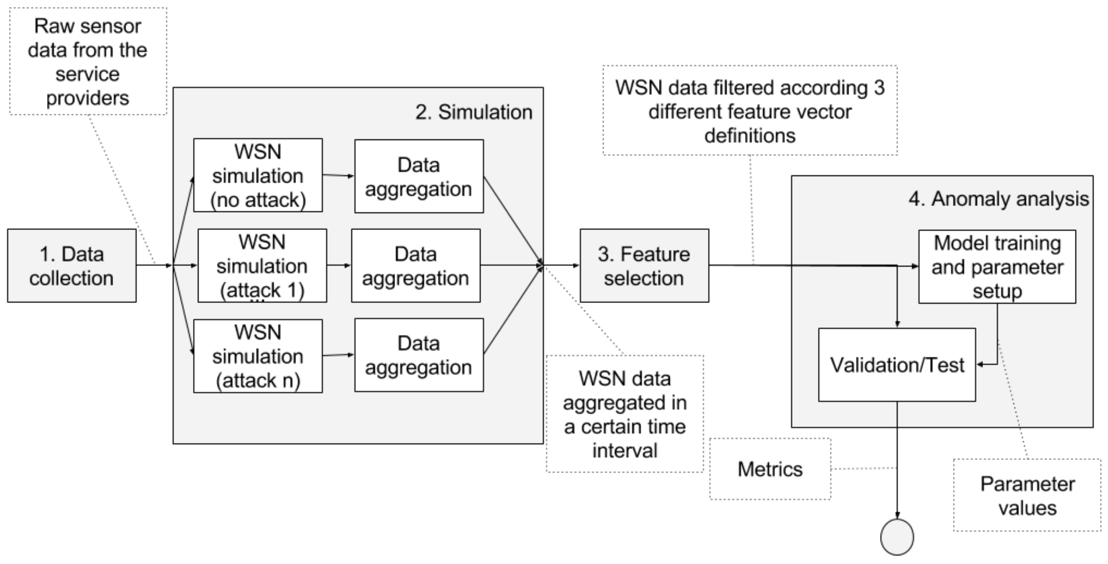

Raw sensor data from the service providers
1. Data collection
2. Simulation
WSN simulation (no attack)
WSN simulation (attack 1)
WSN simulation (attack n)
Data aggregation
Data aggregation
Data aggregation
WSN data aggregated in a certain time interval
3. Feature selection
WSN data filtered according 3 different feature vector definitions
4. Anomaly analysis
Model training and parameter setup
Validation/Test
Metrics
Parameter values

FIGURE 4.2: Pipeline of the simulation and the experimental process.

(a) Training phase: a model is trained or the parameters required by the detection technique are setup.

(b) Validation/Test phase: the performance of the technique to distinguish between the samples that were generated with and without attacks is tested. At this stage, the metrics to compare the different techniques are computed.

We repeat steps three and four with three different feature sets for each of the four detection techniques compared in this study. The results taking into account the different situations are discussed in Section 4.3.

4.2.3 Data collection

The first step in this study is the collection of real urban data. The scenario for this simulation is based on data gathered during 14 days from sound meters deployed in the city of Barcelona. The sound meters, which are installed on the streets by a service provider, send their readings to the smart city central servers. The layout of the sensor nodes is represented in Figure 4.3. The outcome of this first step is a dataset with raw sound data.

As Figure 4.3 shows, the WSN contains a reduced number of sensors belonging to a single provider, even though networks gathering data from a city service can be much more complex, involving many more nodes and several providers. In case of anomalies,

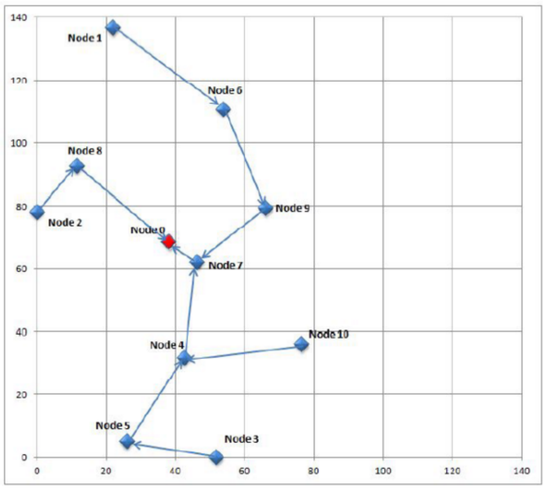


FIGURE 4.3: Schema and topology of the simulated WSN. The layout of the sensor nodes (i.e. nodes 1-10) reproduces the layout of real sound meters deployed in Barcelona over a 140m x 140m terrain. The topology and the base station (i.e. node 0) location are setup ad-hoc for the simulation.

however, the network should be divided into smaller sections, because this allows administrators to reduce the search for the specific compromised equipment to a smaller area with fewer nodes and less providers.

4.2.4 Simulation

The raw data from the sound readings obtained in the previous step are used in the second step to build a realistic simulated scenario of a smart city service with Castalia 3.3 simulator [100]. Castalia is an appropriate simulator for these experiments because it has a highly accurate radio physical model [104], it is specialized in WSNs and, therefore, it includes the most popular MAC and routing protocols for this type of networks, it is widely used and it offers a moderate complexity. This simulator can aggregate information from all the layers involved in the communication between the sensors and the base station using different configurations in a WSN. In the studied real WSN implementations, most of this network status information is currently not disclosed by service providers and, therefore, it is unavailable at the smart city data centers. Thus, this study analyzes the effects of including this information to detect attacks.

In order to use the real sound readings in the simulations, we implemented an application module [105] in Castalia that replays the exact sending behavior of the real sound devices.

In this way, the simulated sensors acquire the same sending patterns as the real sensors deployed on the streets.

The simulation also takes into account that WSNs are unreliable networks in which packets can be lost even in non-attack circumstances. To mimic this behavior, Castalia's physical and communication layers lose some packets. This circumstance is paramount in order to evaluate the detection techniques in a realistic scenario, where communications are not always perfect. Moreover, we also include two nodes from which no messages are received because of failed communication and inactivity. The simulated WSN uses the CC2420 [106] communication module, configured in TMAC [107] and follows a multihop tree topology, as it can be seen in Figure 4.3.

In step 2, the simulation runs to generate data with and without attacks. The implemented attacks exemplify two easy ways to attack WSNs, which can also be disruptive for third-party WSNs in smart cities. Moreover, the attacks cover different levels of affectation in terms of the number of compromised nodes in the network and in terms of disrupted packets. The following are the implemented attacks:

- **Constant jamming.** Attack at the physical and link layers, where the attackers send a high power signal to a legitimate node in order to avoid the correct reception of legitimate packets from other nodes. Besides disrupting application packets, this attack also has an effect on MAC protocols, because the attacker also jams control packets and occupies the channel for a long time, which disrupts the coordination among nodes and impedes other nodes from starting their transmission. We implemented this attack in three situations: near node 4 (affecting 4 nodes in the lower area of the network), near node 9 (affecting 3 nodes in the higher area of the network) and near the base station (affecting all the nodes in the network).
- **Selective forwarding.** Attack at the network layer, where the attackers have captured the base station and they drop a percentage of random packets before re-transmitting them to the smart city control center. We implemented this attack in four levels: a selective forwarding dropping 30%, 50%, 70% and 90% of the packets.

Besides simulating the WSN events in step 2, Castalia aggregates the outcome in time intervals. This outcome is mainly a set of variables containing network status information about the communication protocol for each node. For instance, the number of radio packets received with interferences during a certain period of time.

The size of the time window is a paramount parameter in the detection process of attacks concerning data availability. On the one hand, short attacks can get obscured among a plethora of data in large time windows. On the other hand, datasets in small time windows sometimes do not contain enough variability to be able to distinguish between normal and attack situations.

Having a too large or too small time window also depends on the type of monitored service. For instance, during the 14 days of the sound data gathering process, we measured in Barcelona an average of 30.85 messages per hour, per sensor from a parking service and 1,000.57 messages per hour, per sensor from an electrical meter service. This implies that an attack against the electrical meters lasting a few minutes drops several messages and becomes easily visible, whereas the same attack against the parking sensors does not always leave traces in the data since a lack of messages from the parking sensors can be normal for several minutes. In the simulation for this study, we divide the 14-day sound data in 30-minute time windows. As a result, the dataset contains 5,344 samples of eight classes (i.e. one class for the 668 samples with no attack and one class for each of the seven attacks). Each sample contains information like the number of received application packets and the battery used during the interval.

4.2.5 Feature selection

As previously stated, the main goal of this study is to evaluate the performance of several semi-supervised and unsupervised techniques in different situations considering different degrees of network status information availability. To achieve this goal, this status information is converted into features in a vector space, which is then explored by the anomaly detection algorithms to identify the attacks. The feature vectors extracted from the WSN data determine the set of variables included in the learning models of these algorithms. These variables are the basic knowledge with which to decide if each sample in the dataset contains anomalies.

In other machine learning applications, a large number of features are gathered and a feature extraction transformation is executed prior to classification in order to reduce the dimensionality of the vector space. However, in our context, the necessary features have to be chosen from the inception of the process. This is due to the fact that adding extra features requires computing, sending and forwarding more information from the WSN nodes, therefore having a negative impact on the network performance and the sensors battery life. Therefore, this study compares the performance of the detection algorithms, taking into account three different situations related with the available features:

- **Feature Vector 1 (FV1)**. This includes data aggregated from the minimum information that any WSN always sends (i.e. the sensor readings and the timestamp). The aggregated features are: the number of application packets received at the central server and the hour of the day.
- **Feature Vector 2 (FV2)**. This includes FV1 fields plus the data extracted and aggregated from supplementary fields included in the packets (i.e. the sequence number of the application packet and the battery level). The aggregated features are: the ones in FV1 and also the number of lost application packets and the consumed energy.
- **Feature Vector 3 (FV3)**. This includes FV2 fields plus data aggregated from the principal components of the WSN communication protocol in the physical, link, network and application layers. The additional features included in this feature vector per node are: the number of proper radio transmissions with and without interferences; the number of failed radio transmissions due to interference, the low sensitivity and incorrect reception state; the number of received MAC ACK and CTS.

The necessary information to build FV1 and FV2 is already available in some real WSN implementations in Barcelona, whereas the extra information required to aggregate the data to build FV3 is currently not available in any implementation. In fact, with FV3, we are evaluating the case where administrators use all the available features to train and test the models. Even though not all the features are necessarily relevant to disclose attacks, we are testing the resistance of the algorithms to increasing the vector dimensionality with non-relevant features.

The outcome of the feature selection step is the dataset from the previous step filtered according to one of the feature vector descriptions.

4.2.6 Anomaly analysis

The anomaly analysis [105] step compares four different techniques using the programming language R [108]. The first technique is implemented with the *stats* [108] package and it is based on Mahalanobis distance (See Section 4.1.1 for more information). The second technique computes the LOF score with *DMwR* [93] (See Section 4.1.2 for more information). For the third technique, we compute an outlier score using agglomerative hierarchical clustering analysis according to Ward's clustering method [92] (See Section 4.1.3 for more information). This score is obtained with the *sizeDiff* method through the function *outliers.ranking* in *stats*. Finally, we use the *e1071* [109] package for the fourth technique: a one-class classification with OC-SVM (See Section 4.1.4 for more information).

The anomaly analysis comprises three basic sub-steps for each of the compared techniques: the training, the validation and the test phases. In order to perform these sub-steps, first of all, our study takes the filtered dataset obtained in the feature selection step and divides it as shown in Figure 4.4. As this figure shows, the attack samples are not included in training dataset (a), because the detection techniques used in this comparative study are semi-supervised or unsupervised. Regarding the validation and test datasets, each of them is divided into 8 additional datasets ((b) to (i) in the figure), resulting in a total of 17 datasets (16 + 1 training dataset). As will be described in the next section, these datasets are used to run 72 experiments to evaluate the four anomaly detection techniques described in Section 4.1.

Once these dataset partitions are obtained, basically, we will use the training dataset to tune the parameters required by the algorithms. We will use the validation dataset internally during the development of the experiments to estimate the performance of the algorithms. Finally, we will use the test dataset just once to obtain the final results of this study. The following sections include more details about these datasets and the actions taken in the training (Section 4.2.6.1), validation and test phases (Section 4.2.6.2).

4.2.6.1 Training phase

The main responsibility of the training phase is to find the best parameters for the algorithms and to fit the models. We use the training dataset, which contains only

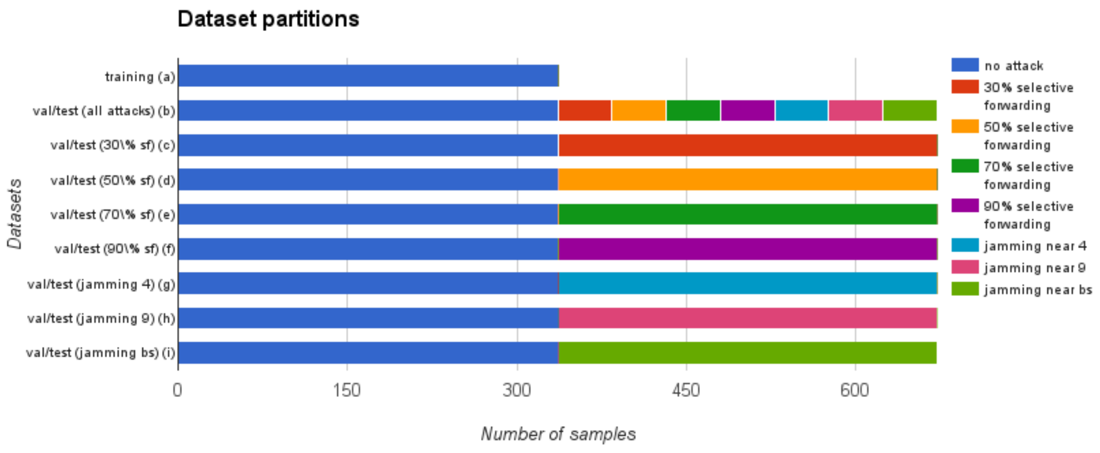


FIGURE 4.4: Size of the dataset partitions. The validation and test (val/test) datasets are partitioned in the same manner and contain the same number of samples of each attack type.

samples without attacks, to perform these two tasks.

Before training the models and selecting the parameters, we first normalize and standardize the features in all the datasets (i.e. subtracting the mean and dividing by the standard deviation for each feature) and then we identify the features that have a zero variance in the training dataset. These features are removed from the three datasets (i.e. training, validation and test). Thereby, the features that do not provide any information for the detection process are eliminated. We use the remaining features in the training dataset to train the models and to find the best parameters for the algorithms considering three different levels of false positive rate: permissive (false positive rate $< 15\%$), restrictive (false positive rate $< 10\%$) and very restrictive (false positive rate $< 5\%$). From now on, we will refer to these levels as the permitted false positive rate (PFPR). We consider that a rate of more than 15% overwhelms administrators with an excessive number of false alarms.

In order to select the optimum parameters for the OC-SVM, we use grid search [96]. In this method, a grid with parameter values is exhaustively explored in order to select the values that give the best performance using the training dataset. OC-SVM requires the set up of two parameters: ν and γ. We fix the value of ν to the PFPR, since the training dataset does not contain any samples with attacks and the ν value is a higher limit on the fraction of outliers in the training dataset [110]. We use grid search to find the best value for γ using *svm.tune* [109], configured in a 10-fold cross-validation repeated

3 times [111]. This function uses the classification error as a performance measure to decide the best value for γ.

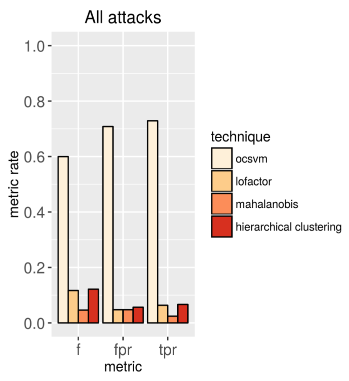
Before the different detection techniques can be compared, an additional step has to be carried out, since the OC-SVM technique returns a binary value (which simply indicates if the sample is an outlier or not) and the LOF, Mahalanobis and hierarchical clustering techniques return an outlier score (in our context, outliers will be considered as attacks). Thus, the outlier score must be translated into a decision about whether the sample is considered an outlier or not. In order to do so, for each of these score-based techniques, we select a threshold score beyond which the sample is considered an outlier. This threshold score is determined as the threshold where the false positive rate in the training dataset is equal to the PFPR. The procedure is as follows:

1. The outlier score is computed for each sample in the training dataset using the corresponding detection algorithm. This results in a list L of scores.
2. Any sample in the training dataset identified as attacked should be considered a false positive (FP), since this dataset does not contain any attack. Therefore, the maximum amount of allowed false positives in the training dataset is defined by the PFPR (i.e. $FP \leq |L| * PFPR$).
3. The $|L| * PFPR$ highest score in L is set as the threshold.

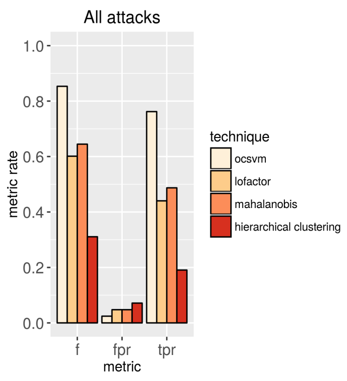
Furthermore, LOF also requires the definition of the parameter k. In this study, we have determined that $k = 10$ is a good choice following the indications of [57], as described in Section 4.1.2.

4.2.6.2 Validation and test phase

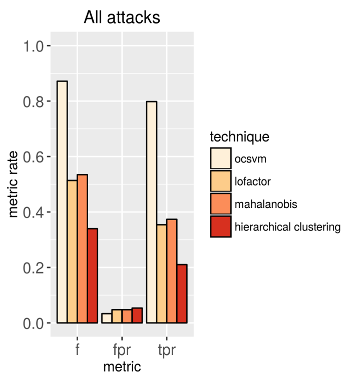
The validation and test datasets are used to evaluate the performance of the algorithms in 72 experiments: (1 with all the attacks together + 7 with each attack separately) x 3 feature vector definitions x 3 PFPR levels. As Figure 4.4 shows, these datasets contain the same number of samples and each is divided into several partitions. Dataset (b) contains half of the dataset without attacks and the other half with attacks, with equal proportion of samples from each of the seven attack types. This dataset allows us to



(A) FV1.



(B) FV2.



(C) FV3.

FIGURE 4.5: Results using the test dataset with samples of all the attacks filtering the features according to the three feature vector definitions with a very restrictive PFPR. The plots show the metrics f-score (f), the false positive rate (FPR) and the true positive rate (TPR). The captions below each plot indicate the feature vector definition used in each case.

validate and test the behavior of the detection algorithms in a general way, taking into account all the attacks. Moreover, we also create validation and test datasets (c) to (i), which only include samples of a single attack. These datasets allow us to evaluate the performance of the algorithms for each of the different attacks separately. We balance the number of samples with and without attack in each of the datasets using sampling with replacement [112].

In the validation and test phases, we use the detection algorithms to decide whether each sample has to be considered as an attack or not. Then, we count the correct identifications of attacks as true positives, the incorrect identifications of attacks as false positives, the correct identifications of no attacks as true negatives and the incorrect identifications of no attacks as false negatives. Afterwards, we compute the metrics described in Section 3.3.1. The training and validation phases are iteratively conducted to explore the most suitable configurations of the algorithms. These configurations are then applied in the test phase to obtain the results shown in the next section. To compare the different algorithms we use the true positive rate (TPR), the false positive rate (FPR) and the f-score.

Table 4.1: Results sorted by TPR using test dataset (b) with samples of all the attacks.

FV	PFPR	technique	F-score	FPR	TPR
FV3	very restrictive	ocsvm	0.872	0.033	0.798
FV3	restrictive	ocsvm	0.857	0.033	0.774
FV2	very restrictive	ocsvm	0.853	0.024	0.762
FV2	restrictive	ocsvm	0.853	0.024	0.762
FV3	permissive	ocsvm	0.843	0.030	0.750
FV1	very restrictive	ocsvm	0.6	0.708	0.729
FV1	restrictive	ocsvm	0.599	0.696	0.723
FV2	permissive	ocsvm	0.809	0.024	0.696
FV1	permissive	ocsvm	0.583	0.681	0.690
FV2	permissive	hierarchical clustering	0.665	0.211	0.552
FV2	permissive	mahalanobis	0.670	0.149	0.542
FV2	restrictive	mahalanobis	0.655	0.098	0.511
FV2	permissive	lofactor	0.641	0.149	0.507
FV3	permissive	hierarchical clustering	0.616	0.220	0.495
FV2	very restrictive	mahalanobis	0.645	0.048	0.487
FV3	permissive	mahalanobis	0.621	0.149	0.484
FV2	restrictive	lofactor	0.631	0.098	0.484
FV3	restrictive	mahalanobis	0.598	0.098	0.448
FV2	very restrictive	lofactor	0.601	0.048	0.44
FV3	permissive	lofactor	0.569	0.149	0.428
FV3	restrictive	hierarchical clustering	0.545	0.140	0.401
FV3	restrictive	lofactor	0.547	0.098	0.395
FV3	very restrictive	mahalanobis	0.535	0.048	0.374
FV2	restrictive	hierarchical clustering	0.517	0.098	0.366
FV3	very restrictive	lofactor	0.514	0.048	0.354
FV1	permissive	hierarchical clustering	0.394	0.158	0.265
FV3	very restrictive	hierarchical clustering	0.340	0.054	0.210
FV2	very restrictive	hierarchical clustering	0.311	0.071	0.191
FV1	restrictive	hierarchical clustering	0.258	0.101	0.156
FV1	permissive	lofactor	0.251	0.149	0.154
FV1	restrictive	lofactor	0.195	0.098	0.113
FV1	permissive	mahalanobis	0.124	0.149	0.071
FV1	very restrictive	hierarchical clustering	0.122	0.057	0.067
FV1	very restrictive	lofactor	0.117	0.048	0.064
FV1	restrictive	mahalanobis	0.112	0.098	0.062
FV1	very restrictive	mahalanobis	0.046	0.048	0.024

4.3 Results and discussion

This section shows the most relevant results of the 72 experiments. As previously mentioned, these experiments evaluate the detection algorithms using the different feature vector definitions for the different PFPR on the test dataset partitions shown in Figure 4.4. Only the most important information is included in this section (Figure 4.5,

Table 4.2: Results of several cases exceeding the PFPR. Cases where PFPR<FPR are highlighted

FV	attack	PFPR	technique	F-score	FPR	TPR
FV2	Selective forwarding 30%	very restrictive	ocsvm	0.811	0.117	0.762
FV2	Selective forwarding 30%	very restrictive	lofactor	0.218	0.048	0.125
FV2	Selective forwarding 30%	very restrictive	mahalanobis	0.598	0.048	0.437
FV2	Selective forwarding 30%	very restrictive	hierarchical clustering	0.003	0.071	0.002
FV2	Selective forwarding 50%	very restrictive	ocsvm	0.82	0.054	0.732
FV2	Selective forwarding 50%	very restrictive	lofactor	0.502	0.048	0.343
FV2	Selective forwarding 50%	very restrictive	mahalanobis	0.609	0.048	0.449
FV2	Selective forwarding 50%	very restrictive	hierarchical clustering	0.003	0.071	0.002
FV2	Selective forwarding 30%	restrictive	ocsvm	0.811	0.117	0.762
FV2	Selective forwarding 30%	restrictive	lofactor	0.348	0.098	0.221
FV2	Selective forwarding 30%	restrictive	mahalanobis	0.613	0.098	0.464
FV2	Selective forwarding 30%	restrictive	hierarchical clustering	0.111	0.098	0.062

Tables 4.1 and 4.2). The remaining results are shown in Appendix A.

Filtering the datasets according to the feature vector definition FV2 and using OC-SVM is the optimal approach for attack detection in the context described in this chapter. Table 4.1 (sorted by the true positive rate column) presents the performance of the algorithms using samples with all the attack types in test dataset (b) and filtering the features according the three feature vector definitions. The top rows in the table show that OC-SVM is the technique performing the best in terms of true positive rate and false positive rate. For all the PFPR, OC-SVM performs better than any of the other techniques. The minimum difference in the performance occurs with a permissive PFPR. In this case the true positive rate using OC-SVM is 37% higher than with LOF, 28% higher than with Mahalanobis distance and 26% higher than with hierarchical clustering. With a very restrictive PFPR (Figure 4.5), which is the most challenging configuration to disclose attacks, the difference in the performance is the highest. In this case the true positive rate using OC-SVM is 73% higher than with LOF, 56% higher than with Mahalanobis distance and 300% higher than with hierarchical clustering. In this last configuration, the true positive rate using OC-SVM is over 75% and the f-score over 85%.

From a theoretical point of view, the results suggest that a large amount of samples with attack lie too close to samples without attack in the vector space. Therefore, techniques based on distances (i.e. Mahalanobis, LOF and hierarchical clustering) cannot distinguish between the two types of samples in many cases (especially in the most restrictive situations). However, in OC-SVM, the results suggest that the separating hyperplane resulting from the training process is closely adjusted to the data without attacks. As a result, samples with attacks lie, in most cases, outside the frontier defined by this hyperplane, even when these samples are very near to the ones without attacks.

Furthermore, as it can be seen in Appendix A, using the features defined by FV2, OC-SVM also gives the best results for all the metrics in 18 of the 21 experiments when using test datasets (c) to (i), which contain only samples of a single type of attack (i.e. 7 attacks $\times$ 3 feature vector definitions). However, in three experiments (Table 4.2), the false positive rate exceeds the PFPR. In the experiment with 30% selective forwarding, the false positive rate exceeds the very restrictive PFPR by 6.7 percentage points and the restrictive PFPR by 1.7 percentage points. In the experiment with 50% selective forwarding, the false positive rate exceeds the very restrictive PFPR by less than 1 percentage point. Although these three experiments show the false positive rate as slightly over the PFPR, the other 18 experiments show that OC-SVM is generally the most suitable in this context.

Unlike filtering features with FV2 or FV3, when filtering is performed with FV1, datasets with and without attacks show only a slight variation. For example, as Table 4.3 shows,

Table 4.3: Mean of the standard deviation of all the features of the training dataset (a) and the test dataset (b) with all the attacks for each feature vector definition

FV	**Dataset**	**Std. Mean**
FV1	training dataset (a)	0.48
FV1	test dataset (b)	0.45
FV2	training dataset (a)	0.39
FV2	test dataset (b)	0.60
FV3	training dataset (a)	0.57
FV3	test dataset (b)	0.79

when data in the feature vectors are normalized and we compute the mean of the standard deviation among all the features, with FV1 the difference between including and excluding attacks is minimal (i.e. 0.48 for the training dataset and 0.45 for the test dataset). However, this difference is larger with the other two feature vector definitions (i.e. 0.39 for the training dataset and 0.60 for the test dataset with FV2). Including only features from FV1 makes attack samples and normal samples to lie very close in the vector space. Therefore, the performance of all the compared techniques is generally very poor in this case as it can be seen in Table 4.1. With FV1, the highest true positive rate is achieved when dealing with attacks that affect a large number of nodes (i.e. 90% selective forwarding attack and jamming attack near the base station). In this case, the technique based on hierarchical clustering achieves a true positive rate of around 30% if PFPR is permissive. Hence, in the scenarios where only the features in FV1 are available, none of these techniques is suitable. Therefore, we can conclude that public administrations should never allow WSN providers to supply so little network status data.

Finally, Figure 4.5 also shows that OC-SVM is the only technique resistant to the inclusion of too many features for the algorithms. With the extra features included in FV3, the rest of the algorithms decrease their performance. SVMs do not depend on the size of the vector space to be able to properly generalize [113]. This technique shows more resistance to high dimensionality and to the inclusion of correlated features.

Furthermore, in the scenario using the extra features included in FV3, the detection performance of the OC-SVM algorithm slightly improves in some cases. However, as previously stated, these features are currently not sent in any of the analyzed services in Barcelona. Besides, sending extra features can be detrimental for the network nodes and, therefore, the slight increment in the performance is not worth the effort of adding the extra features in FV3.

4.4 Conclusions

In this chapter, we compared diverse techniques to analyze whether the data received from smart city WSNs are the result of the normal operation of the network or whether they contain some type of anomaly due to computer attacks. The compared techniques are either semi-supervised or unsupervised, because obtaining a large dataset of labeled data with attacks for training purposes is difficult in a smart city. Therefore, supervised techniques are not practical in this context. In this comparative study, we used real data from the smart city of Barcelona to simulate WSNs and implement typical attacks. Then, using these data, we compared four anomaly detection techniques based on different principles: Mahalanobis distance, local outlier factor, hierarchical clustering and OC-SVM. We used various feature vector definitions to identify the optimal network status fields that the service providers have to include to effectively detect attacks. We also considered three scenarios with different maximum levels of permitted false positive rates. As a result of this work, we conclude that OC-SVM is the most suitable technique in the smart city scenario described in this thesis. Moreover, we justified that the optimal network status information that should be gathered for proper attack detection must include the sequence number of the application packet and the battery level. Considering the most restrictive case with a permitted false positive rate of less than 5%, our experiments achieved a true positive rate of over 75%. This value is at least 56% higher than the rates achieved with any of the other compared techniques.

Chapter 5

Intrusion detection pipeline viability

So far, an architecture capable of receiving and indexing big data from smart city WSNs has been proposed in order to analyze the data in the search for intrusions. Moreover, a pipeline to carry out these analyses has also been presented. This chapter includes a study of the viability and scalability of the execution of this pipeline and its subprocesses. Section 5.1 analyzes the criticality of each subprocess in the pipeline and, for the ones showing a high critical nature, it is shown that they can be parallelized, providing an implementation with MapReduce. Section 5.2 analyzes the main time constraints between subprocesses. Finally, Section 5.3 includes a simulation of the most critical subprocesses and it confirms that, using mid-range servers, a volume of data similar to that required by a typical smart city service can be processed. Thus, provided enough hardware resources, the pipeline is scalable.

5.1 Principal subprocesses

Chapter 3 has shown that the pipeline in Figure 5.1 is the basic processing flow from data reception until the triggering of warnings due to intrusions. This section analyzes the scalability of the subprocesses in the figure. For each subprocess, we examine its computational complexity and, for the most complex ones, we provide a MapReduce schema in order to show that the subprocesses are parallelizable and the general pipeline is scalable and viable. A SIEM system is the base of the proposed architecture. Therefore, it can be assumed that the system is capable of gathering and indexing big data.

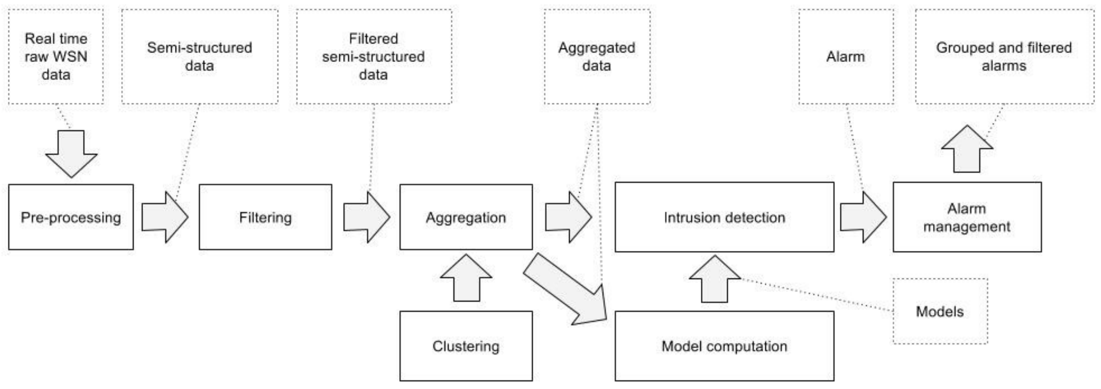


FIGURE 5.1: Intrusion detection pipeline.

5.1.1 Preprocessing

The raw WSN data have to be transformed and indexed in order to ease the subsequent steps. The transformation operations for this subprocess described in Section 3.4.1 generally have a very low computational complexity, i.e. $O(1)$, since many are simple operations applied to a single sample (e.g. unit conversions).

Other operations also require previously received data. However, generally, only the immediately prior sample is necessary (e.g. battery consumption can be computed by subtracting previous battery level from current level). This can also be computed in $O(1)$.

We consider this subprocess as non-critical. We assume that, if the system is capable of gathering and indexing large amounts of data, then it is also capable of preprocessing them.

5.1.2 Filtering

In this subprocess, preprocessed data are filtered using comparison operations with other variables or constants (e.g. $variable > value$), which have a computational cost of $O(1)$. Moreover, more complex filter functions, such as a dichotomic search where a variable is compared against multiple other ordered values have a cost of $O(log(n))$.

We consider that this is not a critical subprocess. As in the previous case, if the system is capable of gathering and indexing large amounts of data, then it is also capable of filtering them.

5.1.3 Clustering

The aim of this subprocess is dividing data into groups (i.e. clusters) in order to segment analysis according to the different groups. In this way, administrators reduce the search space and draw conclusions from a short amount of nodes with common features.

Regarding computational costs, clustering involves algorithms that can be computationally intensive. Nonetheless, once clustering is performed, the clusters do not have to be recomputed often. Basically, clusters are no longer valid at the moment that variables for which data have been clusterized change or when nodes are joined or removed from the network. This does not happen very often, because these variables tend to be stable (e.g. frequency band, location).

5.1.4 Aggregation

Data aggregation operations have computational costs that depend on the number of samples to aggregate. For instance, aggregating the last n samples on a single variable using an operation such as *mode* has a maximum computational cost of $O(nlog(n))$. Other aggregation operations have lower computational costs (e.g. *minimum*, *maximum*, *sum*, *mean* and *median* can be calculated in time $O(n)$). The most complex aggregation operations require sorting the list of samples, which is the sub-operation with the highest cost. However, as this sub-operation can be, at least, partially reused between subsequent aggregation operations on the same data source, the total computational complexity of these aggregation operations can be substantially reduced.

Furthermore, n is generally not large for aggregation operations that have time restrictions. In these cases, higher time restrictions imply shorter intervals and, therefore, also imply including fewer samples in the time interval.

Hence, we consider that this subprocess is non-critical and can be done by any big data system.

5.1.5 Model computation

As seen in Section 3.3, several types of anomaly detection techniques are suitable for detecting intrusions in this context. This subprocess can be computationally critical using any of the suggested techniques for the following three reasons:

- In the case of univariate models, the complexity arises due to the fact that a different model needs to be computed for each variable that requires monitoring. In this way, although univariate techniques have a low computational complexity (e.g. Tukey's method can be computed in $O(nlog(n))$ or less if the list of samples is already sorted), the models have to be computed for each of the necessary m variables in the dataset and, therefore, it is convenient to parallelize the computation.
- In the case of univariate autoregressive models, although they require few historical samples and they have a low training computational complexity (ARIMA models on n training samples can be computed in $O(n)$ [114]), their usage has an additional critical factor because the models can not be reused several times and, therefore, they have to be recomputed every time new data arrive.
- In the case of multivariate models, the training process is complex. With n as the number of training samples, the computational complexity of OC-SVM lies between $O(n^2)$ and $O(n^3)$ [115].

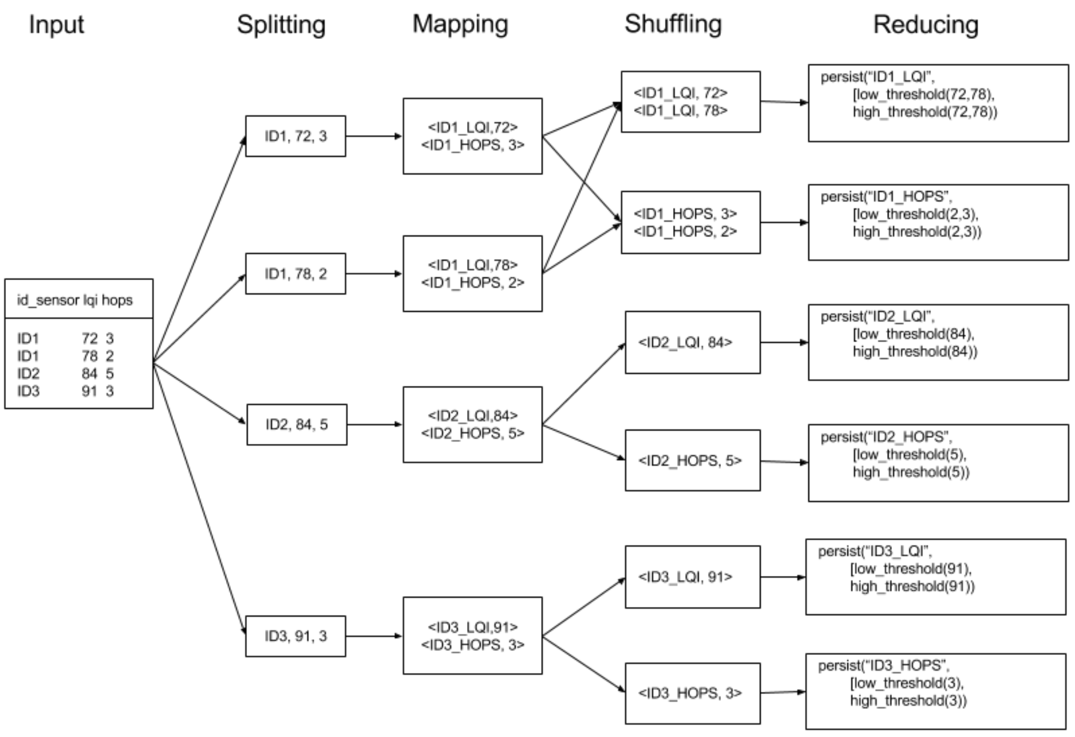
Hence, relying on both univariate and multivariate models can be computationally expensive and requires a systematic parallelizable approach to be able to compute all the models required by administrators with large historical datasets with many variables. Therefore, we propose a procés based on MapReduce to divide the input data and train the models in parallel. Algorithm 1 presents the pseudocode for this process. The basic structure of this algorithm can be used to train either simple univariate or complex machine learning multivariate models. Algorithm 2 contains auxiliary functions used in Algorithm 1. Figure 5.2 shows two examples of the usage of the algorithm. In this manner, with enough hardware resources, the system is capable of parallelizing and scaling this subprocess to train the necessary anomaly detection models.

Algorithm 1 MapReduce algorithm pseudocode to train anomaly detection models.

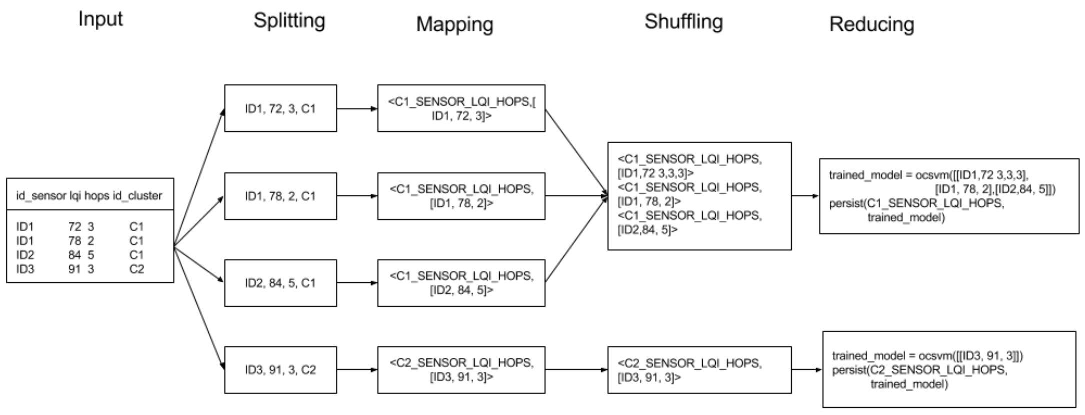
```
Require: $record$, a sample from the training dataset
Require: $G$, a tuple with the variable names to group the $record$ variables
Require: $F$, a tuple of tuples with the variable names of the features selected to generate
  the models
  procedure MAP
      $g\_values \leftarrow get\_variable\_values(record, G)$
      for each $feature\_names \in F$ do
          $feature\_values \leftarrow get\_variable\_values(record, feature\_names)$
          $new\_key \leftarrow concat(g\_values, feature\_names)$
          $emit(new\_key, feature\_values)$
      end for
  end procedure
Require: $key$, a new key generated in a $map$ function.
Require: $values$, a tuple with the tuples emitted in a map function with the same
  $new\_key$.
  procedure REDUCE
      $trainned\_model \leftarrow train(values)$
      $persist(key, trained\_model)$
  end procedure
```

Algorithm 2 High-level description of the auxiliary functions used in Algorithm 1

```
  procedure GET_VARIABLE_VALUES($record$,$variable\_names$)
      Returns a subset of $record$ with the variable values defined by $variable\_names$
  end procedure
  procedure CONCAT($values$)
      Concatenates all input $values$ separating them by "_".
  end procedure
  procedure EMIT($key$,$value$)
      Publishes intermediate results containing $key$ and $value$. These are shuffled with
  other intermediate results and fed to a reducer.
  end procedure
  procedure TRAIN($training\_dataset$)
      Trains a model using $training\_dataset$ with a predefined anomaly detection algo-
  rithm. The model is returned by this function.
  end procedure
  procedure PERSISTS($key$,$value$)
      Persists $value$ with $key$ in a key-value system.
  end procedure
```


Input
Splitting
Mapping
Shuffling
Reducing
id_sensor lqi hops
ID1 72 3
ID1 78 2
ID2 84 5
ID3 91 3
ID1, 72, 3
ID1, 78, 2
ID2, 84, 5
ID3, 91, 3
<ID1_LQI,72> <ID1_HOPS, 3>
<ID1_LQI,78> <ID1_HOPS, 2>
<ID2_LQI,84> <ID2_HOPS, 5>
<ID3_LQI,91> <ID3_HOPS, 3>
<ID1_LQI, 72> <ID1_LQI, 78>
<ID1_HOPS, 3> <ID1_HOPS, 2>
<ID2_LQI, 84>
<ID2_HOPS, 5>
<ID3_LQI, 91>
<ID3_HOPS, 3>
persist("ID1_LQI", [low_threshold(72,78), high_threshold(72,78))
persist("ID1_HOPS", [low_threshold(2,3), high_threshold(2,3))
persist("ID2_LQI", [low_threshold(84), high_threshold(84))
persist("ID2_HOPS", [low_threshold(5), high_threshold(5))
persist("ID3_LQI", [low_threshold(91), high_threshold(91))
persist("ID3_HOPS", [low_threshold(3), high_threshold(3))

(A) Example of the execution of Algorithm 1 to compute univariate models based on low and high thresholds for all the variables of the input records. The input dataset contains the variables: *id_sensor*, *lqi* and *hops*. In this example, the required parameters in *map* function are: *record* $= < id\_sensor, lqi, hops >$, $G = < "id\_sensor" >$ and $F = << "lqi" >, < "hops" >>$.


Input
Splitting
Mapping
Shuffling
Reducing
id_sensor lqi hops id_cluster
ID1 72 3 C1
ID1 78 2 C1
ID2 84 5 C1
ID3 91 3 C2
ID1, 72, 3, C1
ID1, 78, 2, C1
ID2, 84, 5, C1
ID3, 91, 3, C2
<C1_SENSOR_LQI_HOPS,[ID1, 72, 3]>
<C1_SENSOR_LQI_HOPS, [ID1, 78, 2]>
<C1_SENSOR_LQI_HOPS, [ID2, 84, 5]>
<C2_SENSOR_LQI_HOPS, [ID3, 91, 3]>
<C1_SENSOR_LQI_HOPS, [ID1,72 3,3,3]> <C1_SENSOR_LQI_HOPS, [ID1, 78, 2]> <C1_SENSOR_LQI_HOPS, [ID2,84, 5]>
<C2_SENSOR_LQI_HOPS, [ID3, 91, 3]>
trained_model = ocsvm([[ID1,72 3,3,3], [ID1, 78, 2],[ID2,84, 5]]) persist(C1_SENSOR_LQI_HOPS, trained_model)
trained_model = ocsvm([[ID3, 91, 3]]) persist(C2_SENSOR_LQI_HOPS, trained_model)

(B) Example of the execution of Algorithm 1 to compute multivariate models based on OC-SVM per each cluster. The input dataset contains the variables: *id_sensor*, *lqi*, *hops* and *id_cluster*. In this example, the required parameters in *map* function are: *record* $= < id\_sensor, lqi, hops, id\_cluster >$, $G = < "id\_cluster" >$ and $F = << "id\_sensor", "lqi", "hops" >>$.

FIGURE 5.2: Examples of training models with Algorithm 1

5.1.6 Intrusion detection

Intrusion detection using the models trained in the previous section requires steps similar to those in the training process. The steps for dividing the original data until training the model (i.e. splitting, mapping and shuffling) are equivalent. Then, a model stored in the previous subprocess is loaded and used to test if the input data in the intrusion detection subprocess contain anomalies.

Hence, taking into account these differences, the schema of the algorithm shown in the previous section is also valid for this subprocess. Moreover, actions involved in this subprocess are computationally less expensive than for the training subprocess. Loading a model from a key-value structure (e.g. a hash table) can be done in $O(1)$ on average. Testing if a value falls within two thresholds can also be done in $O(1)$. For more complex multivariate models, computational complexity varies depending on the specific algorithm. The computational cost of testing with non-linear SVMs with a low number of dimensions increases linearly with the number of support vectors of the model [116]. This number is limited by the size of the training set multiplied by the training set error rate. This is clearly lower than the computational complexity of training the model.

5.1.7 Alarm management

With the proposed architecture, the detection engines generate alarms. The main responsibilities of this subprocess are to combine and correlate these alarms into new and more reliable alarms, which are finally warning system administrators. Although the amount of alarms generated by the detection engines can be too high to be handled by a human, it is not too high to be handled by a machine using conventional processing techniques. Hence, this subprocess is not considered critical in terms of processing requirements.

5.2 Temporal constraints

This section studies the strongest temporal constraints that must be satisfied to be able to execute the pipeline shown in Figure 5.1 in a sustainable manner. When implementing and configuring detection algorithms, selecting window sizes, etc., it is of utmost importance to take into account these constraints, in order to avoid undesirable situations, such as testing for anomalies with outdated models. These constraints are especially important in the face of real-time analysis. In the scope of this study, we only take into account the constraints related with the life cycle of a single model.

First, the variables used in this section are:

- T_stream: average time between consecutive data arrivals from services monitored by the model.
- T_window: time window used to aggregate data. This variable is not used when data are not aggregated.
- T_exp_models: average time until the model expires.
- T_train: average time taken training the model.
- T_test: average time taken to test new data with the trained model.
- $T\_redo\_detection$: average time taken until the model is reused to test for anomalies on new data.

Below we show basic time constraints between the different steps in order to keep a sustainable execution cycle of the subprocesses in the pipeline:

$$T\_window >> T\_stream \tag{5.1}$$

$$T\_redo\_detection \geq T\_stream \tag{5.2}$$

$$T\_redo\_detection \geq T\_test \tag{5.3}$$

$$T\_exp\_models \geq T\_train \tag{5.4}$$

Constraints 5.1 and 5.2 are basic in order to perform computation only when there are new input data and new results can be obtained. Constraint 5.3 ensures that the input queue at the intrusion detection subprocess does not constantly increase. Constraint 5.4 ensures that models do not expire before they can be trained.

Complying with these constraints is easy in situations without additional real-time restrictions, where intrusion analysis subprocesses can be programmed at the most convenient time in terms of system resources. Typical examples of this kind of subprocess are cluster generation or model training with a large expiring time ($T\_exp\_models >> T\_train$).

However, these constraints are particularly important in cases where additional real-time restrictions have to be considered. Ideally, all input data samples are analyzed in real-time looking for anomalies. As previously seen, training models and intrusion detection are the most critical subprocesses in the pipeline. The time required for these subprocess depends on the specific type of analysis and on the algorithm selected to implement it. Using detection methods that consume much computational time in networks with a high data reception rate is generally incompatible with real-time analysis of all the data samples. In real-time situations, i.e. $T\_redo\_detection = T\_stream$, another constraint needs to be considered:

$$T\_test \leq T\_stream \tag{5.5}$$

In these circumstances, detection algorithms with a very low computational complexity for testing are adequate. Thus, using precomputed thresholds with a computational cost of $O(1)$ is suitable for discovering anomalies on single variables. For multivariate analysis, we proposed using OC-SVM in Chapter 4. We have seen that the complexity of this machine learning algorithm is also low for testing (i.e. linear with the number of support vectors of the model).

Nevertheless, there are situations where, ideally, detection results have to be processed in real time for all new data entering the system, but Constraint 5.5 cannot be satisfied. In these cases, either the computational cost of the selected algorithm or the data rate is too high. Therefore, ideally $T\_redo\_detection = T\_stream$, but $T\_stream < T\_test$, which would break Constraint 5.5. In these situations, in order to maintain a viable anomaly analysis in real time, it is necessary to decrease the number of times that the test is executed (i.e. increase $T\_redo\_detection$). To that end, we distinguish between analysis with direct streaming data or with aggregated data.

Detecting anomalies on direct streaming data without aggregation requires a sampling method that does not analyze all data but just selects the most appropriate samples. For instance, systematic sampling can be used to test one in n samples. In such a way, system administrators can adjust $T\_redo\_detection$ according to T_test.

Testing for anomalies on aggregated data relaxes some of the previous constraints. On the one hand, provided that $T\_test = T\_redo\_detection < T\_window$, every new data sample is eventually included in at least one time window for aggregation and subsequently used for testing. For example, a service sending data every minute on average ($T\_stream = 1m$), where these data are aggregated using 15-minute windows ($T\_window = 15m$). If, on average, running an anomaly test on new data takes 6 minutes ($T\_test = 6m$), then, by Constraint 5.3, system administrators need to set $T\_redo\_detection \geq 6$. In this way, although $T\_redo\_detection >> T\_stream$, every new observation is aggregated and included in a test sample more than once. However, if $T\_test = T\_redo\_detection > T\_window$, then the sampling method must select the most appropriate aggregated data to run the detection tests.

Finally, it is worth noting that there are some types of models that expire shortly, such as autoregressive models, where $T\_model\_exp = T\_stream$ in the most critical scenario. In theses cases, besides the previous considerations, $T\_train \leq T\_stream$ by Constraint 5.4. This may imply that models that require large historical training datasets become unaffordable. However, using models that expire shortly is suitable in situations where previous samples are highly correlated with current samples. Therefore, small historical datasets are better suited in these cases. Autoregressive models have already been used to predict anomalies on WSN data, proving that this type of model can be executed in very constrained devices and that the best performance results are obtained including very few previous samples in the models [65].

5.3 Temporal analysis

As previously stated, the most time-critical subprocess in the pipeline is anomaly detection, especially when there are real-time constraints. It has also been mentioned that

- Operating System: Ubuntu 14.04 LTS
- Memory (RAM): 8 GB
- Processor: Intel ®Core ™ i5-4210U CPU @ 1.70GHz x 4
- Programming language: Python 2.7
- Machine learning library: Scikit-learn 0.17 [84]

FIGURE 5.3: Characteristics of the experimental environment

the subprocess with the highest computational complexity is training machine learning models. Within the MapReduce schema for these subprocesses, the most critical execution units are in the *reduce* function. For these reasons, this section includes an empirical analysis of the time required for these *reduce* functions.

For this experiment, we generated several datasets with which we trained and tested OC-SVM. We created a dataset that contained 100,000 samples of 1,000 variables for training purposes. This represents the equivalent of more than 2 months of observations from a system that sends 1 observation per minute on average. We also created 2 more datasets of 1,000 samples and 1,000 variables each. One of the datasets contained anomalies and the other was anomaly-free. The datasets were created generating random numbers following a normal distribution $X \sim N(\mu, \sigma^2)$. For the training dataset and the anomaly-free test dataset, the normal distribution followed $X \sim N(0, 0.9)$, and for the test data with anomalies it followed $X \sim N(3, 1)$. The environment used for this experiment is shown in Figure 5.3.

In the experiments, we measured the time required to train the models and test for anomalies. The next section presents the results and discusses the viability of these algorithms in the smart city context.

5.3.1 Results and discussion

In this experiment, fitting the model using the training dataset required 92.91 minutes and testing the models on 2,000 samples from the test datasets required 0.019 seconds per sample. Considering a real-time analysis and setting $T\_redo\_detection = T\_stream = 60s$ (1 observation per minute on average), the temporal constraints 5.3 and 5.5 are satisfied. Taking into account these measures, a single machine like the one used in this experiment (Figure 5.3) would be able to handle testing anomalies on 3,157 samples every minute. This is 4,546,080 samples every day and more than 31 million samples every week. For samples of 4 KB, this implies processing 18.18 GB every day and 6,6 TB every year.

Framing this in the context of a smart city, it can be seen that detection via this route is viable for typical urban services. In Section 2.1, it was explained that the Oyster card system in London gathers 7 million registers every day, including information about the user's ID, the location and a timestamp recorded every time a citizen enters or exits the public transportation network. Assuming 5 variables for these fields and using large data types of 10 bytes, such as long doubles, this would result in registers of 50 bytes, which would yield 350 MB of data every day.

Section 2.1 also mentioned a smart building use case, where about 1 billion registers are gathered every day. Consider 50-byte registers again, then this generates 50 GB of data every day.

Therefore, anomaly detection with OC-SVM would be feasible with a single machine with the characteristics in Figure 5.3 on datasets with a data volume similar to real urban services like the Oyster card. A small cluster of 3 machines would also be able to handle the analysis for datasets such as the one in the smart building use case.

These measurements were computed considering non-aggregated data. In many situations, OC-SVM is tested on aggregated data. Using the testing method described in Section 5.2 for aggregated data, where $T\_stream << T\_redo\_detection < T\_window$, then testing datasets like those would become less time intensive. For example, if data were aggregated in time windows of 20 minutes ($T\_window = 20m$) and tested for anomalies every 10 minutes ($T\_redo\_detection = 10m$), then every data sample would be aggregated and analyzed twice within two time windows. Through this aggregation process, a test dataset such as the smart building one would be reduced from 50 GB and 1 billion registers every day to 5 GB and 100 million a day. This falls within the values that a single machine like Figure 5.3 can handle. Furthermore, it should be noted that the server in Figure 5.3 has reduced processing and memory power compared to conventional servers in data processing centers.

As seen at the beginning of this section, the training time was 92.91 minutes, which is much longer than the testing time. This is logical taking into consideration that the training computational complexity stays between $O(n^2)$ and $O(n^3)$, with n being the size of the training dataset. Thus, extremely large historical training datasets are not viable. Due to Constraint 5.4, it is important that $T\_exp\_models \geq 92.91$ in order to keep a sustainable cycle concerning the training process. It seems unreasonable to believe that models trained using 2-month data (86,400 minutes) will expire before 92.91 minutes, which is 0.001% of the time interval of data on which the model is based. If that were the case, then other models using fewer samples would be more recommendable.

5.4 Conclusions

This section has reviewed the computational complexity of the subprocesses involved in the anomaly detection pipeline. We have identified the most critical subprocesses: model computation and intrusion detection. For these subprocesses, we have presented algorithms based on the MapReduce paradigm to parallelize them and, therefore, make them scalable. Furthermore, we have described the principal time constraints of the pipeline. Taking into account this constraint is important in selecting the most adequate algorithms to maintain a viable anomaly detection cycle. OC-SVM has proven to be very suitable in this context. Although the training complexity of this algorithm is high, the test complexity is low, which is essential in real-time situations. We have empirically tested that performing anomaly detection using OC-SVM in a big data scenario like the Oyster card in London could be handled using a server with modest computational capacity, and the scenario of the city with smart buildings could be processed with a small cluster of 3 machines.

Chapter 6

Attack Classification schema

In the previous chapters, we have proposed an architecture to gather evidence of malfunctions in urban WSNs. The two main components of the architecture are a rule-based detection engine and an anomaly-based detection engine. As previously seen, these engines are capable of triggering alarms in the case of attacks in smart city WSNs. However, the real challenge is to not only show warnings that indicate that a network has been compromised, but identify the cause of the problems and the affected components.

This chapter provides guidelines to assist smart city administrators in their response to incidents, when the detection tools trigger an alarm that involves multiple services, nodes or frequency bands. These alarms, triggered through a correlation rule as it will be shown below, not only have a high severity level, but are also difficult to resolve. Administrators need to find out the attack type, to locate the source of the incident and the involved devices. In order to facilitate these tasks, this chapter presents a schema with the basic steps to classify attacks.

The approach followed in this chapter takes into account the requirement that no assumption must be made regarding specific network configurations, available countermeasures, installed IDS, etc. Considering a generic smart city architecture, the guidelines presented in this chapter are designed in a general manner in order to be easily adaptable to many smart city scenarios. Section 6.1 describes the assumptions taken in this regard.

In order to abstract the schema from particular implementations and make it more generalizable, Section 6.2 identifies the basic alarm types that can be triggered in the architecture proposed in Chapter 3. Thus, in the rest of the chapter, we refer to these alarm types instead of specific alarms caused by specific intrusion detection techniques.

In Section 6.3, we propose seven different attack models based on the study of the effects that attacks have in the components of smart city WSNs in a generic way. The models include the most popular attacks against WSNs. Section 6.4 outlines a procedure with the steps required to classify the received alarms into one of the attack models. This procedure has been devised taking into account the aforementioned assumptions and alarm types. In Section 6.5, we propose a set of contingency plans to mitigate the attacks.

Section 6.6 shows an experimental demonstration with two types of urban WSNs as a proof of concept of the benefits of the proposed classification schema. This proof of concept shows how the classification procedure determines the most likely attack types. Furthermore, it also demonstrates that simple correlation rules, combining alarms from the two detection engines, can significantly increase the detection rate and, therefore, generally improve the reliability of the detection system. Finally, Section 6.7 concludes the chapter.

6.1 Assumptions

As mentioned above, the contributions proposed in this chapter intend to be a guide for how smart city administrators should behave in the case of attack, taking into account the effects that attacks cause to the elements of a generic smart city. In any case, each specific scenario will have to be examined individually and administrators will have to adapt the models, the schema and the list of contingency measures to their smart city context. The most relevant assumptions taken to create the guidelines included in this chapter are listed below:

- Each service uses a single WSN configuration with the same communication protocols in their nodes. If a single urban service is implemented by two providers with different networks types, in this chapter, it is considered to be two different services.

- Gateways are shared by different WSNs and providers. These devices are assumed to be connected to the electricity grid, have enough computational power and a good communication network from and towards the city central servers. Therefore, gateway providers can use conventional security measures.
- The smart city is considered to be in an advanced state of development with a high density of sensors and networks. Therefore, if attack traces involve several providers, networks in several frequency bands, etc.; then the analyzed scenario includes these required elements.
- Large scale attacks are disclosed as several different attacks. For instance, a jamming attack affecting several frequency bands is considered as several jamming attacks.

6.2 Alarms

As seen in the Chapter 3, one of the main functions of the rule-based detection engine is to offer a way to create alarms that are triggered when a rule is fulfilled. In addition, administrators can set a severity level and an action to execute when the alarm is triggered (i.e. run a script). These two properties are very relevant in a smart city, because it is a very complex system that includes multiple subsystems and devices, and, therefore, it is likely that many events trigger a plethora of alarms. Obviously, many of these alarms may not be relevant as a consequence of false alarms or ephemeral malfunctions. Hence, administrators can label the alarms with a severity code to have a first filter to distinguish the alarms that require their immediate attention and also assign an automatic action to activate a security measure.

The following subsections firstly describe the general alarm types that can be generated with the detection engines of the proposed architecture. Secondly, the section will focus on the alarms triggered by correlation rules.

6.2.1 General alarm types

This section will outline the different types of alarms that can be triggered with the architecture for an intrusion detection platform sketched in Chapter 3. In order to select the different categories of alarm types, we have considered the characteristics of the techniques that trigger the alarms, the analyzed variables and the dynamics of the data. Thus, in the rest of the chapter, a high level of abstraction is achieved by referring to the different alarm types instead of mentioning specific intrusion detection techniques, which would make the schema proposed in this chapter less generalizable.

The different types of alarms that we consider are:

- **AT1 - Alarms triggered by simple thresholds**

  This type of alarm is triggered by simple rules that check whether a variable stays within pre-set thresholds. These thresholds can be computed manually for certain variables, for which administrators know a priori their normal boundaries; or they can be computed using methods to find outliers, such as Tukey's method (see Section 2.4.1 for more information). In the case of an attack that triggers an alarm of this type, administrators can have a valuable hint to find the origin of the incident, because the rule that triggers the alarm is generally associated with a single variable of a single node. However, by focusing on just one variable, several attacks may show the same effects and, therefore, other evidence needs to be gathered.

- **AT2 - Alarms triggered by complex models**

  The alarms triggered by complex anomaly detection models, such as OC-SVM, allow administrators to disclose attacks that leave more subtle evidence than just single variables going over thresholds. Thus, these allow monitoring the normal state of several variables, involving different nodes and services at the same time. Compared to AT1 alarms, these alarms allow administrators to know that data are anomalous as a whole and, therefore, they have less information about the specific variables or nodes that are affected.

- **AT3 - Alarms triggered by time series analysis**

  Time series analysis is useful in smart cities because many services generate times series data. In these cases, new observations are closely related to previous ones.

For these types of data, static thresholds are sometimes inadequate. Mathematical models, capable of predicting future values, are more suitable to point out the real values that deviate from the forecasts, which, therefore, can be considered anomalous.

- **AT4 - Alarms triggered by correlation**

  These types of alarms result from establishing certain relationships between an attack and its effects in an area or in some network components. These effects are in turn detected by the other alarm types. Therefore, these types of alarms are created to aggregate other alarms into a single one.

When system administrators create any of the alarms described above, they have to assign a severity level to the alarm. Administrators then have an easy way to filter the alarms that will actually receive their attention. Identifying the severity of the alarms is generally a specific task dependant on the specific smart city and on the security policies. For instance, administrators can flag an alarm as severe if it is implemented on an especially critical service or node; or if the alarm is implemented on a highly reliable protocol and, therefore, any sign of anomaly is a clear sign of attack. Moreover, as it will be seen in the next section, it is also recommended to create AT4 alarms with a high-severity level, because they are more reliable and can affect several systems.

6.2.2 Alarms triggered by correlation rules

A correlation rule is a type of rule that is used to group alarms that have some kind of relationship. The new alarm triggered by a correlation rule can be considered more trustworthy than the alarms that it is composed of, since it gets triggered only if several unwanted situations have already triggered some alarms. Therefore, alarms triggered by correlation rules can be considered more critical, not only because they are more reliable, but also because the alarms that it is composed of may come from several services, devices, providers, etc. Moreover, correlation rules allow administrators to reduce the number of alarms that require their attention. For instance, creating a correlation rule to group the alarms by location in a WSN reduces all the alarms triggered by a source of interference to a single alarm, instead of receiving individual alarms from each of the sensor nodes in the area that receives the interference.

Moreover, administrators can use correlation rules to implement signatures for known attacks, for which administrators can clearly identify the attack traces. In this way, alarms triggered by these rules will not only be reliable, but they will also straightaway point to a specific attack type. The creation of correlation rules is highly dependent on the WSNs that are deployed together, their configurations, and, in general, the specific scenario. Therefore, defining a methodology to create correlation rules falls out of the scope of this thesis.

However, as seen in the examples in Chapter 3, general purpose correlation rules can be easily implemented in most scenarios and, as the proof of concept of Section 6.6 will show, this can significantly improve the detection success. As a general guideline, we recommend that smart city administrators create correlation rules to group the alarms received within a certain time interval triggered by devices sharing certain characteristics, such as a nearby location, the frequency band, the provider, the gateway, etc.

In this way, once a severe alarm built from a correlation rule is received, administrators can begin to collect more insights about the compromised components, providers, etc. The next sections provide some guidelines to assist administrators to this end.

6.3 Attack models

Attacks in WSNs are traditionally detected analyzing particular parameters from each of the affected communication layers. Nevertheless, as previously mentioned, from the smart city administrators' perspective, it is unrealistic to count on the availability of all the parameters and maintaining very specific detection systems for each WSN would be unmanageable. Therefore, the models in this attack schema are a general classification, based on the anomaly traces that most common attacks in WSN leave on the affected data described in Section 3.2.1 and on the attack's geographical influence. For each attack model, a list of attack candidates is provided. These are the most common attacks reported in the literature (see more details in Section 2.3.1). Figure 6.1 shows graphical representations of the seven models. These models are described as follows:

Model 1: Vertical attacks

Description: Attacks that show vertical attack traces (i.e. from a group of node leaves to the base station) on a single WSN. The main aims of these attacks is to obstruct one or more paths in order to increase the arrival time of the packets from the target leaves, to crash intermediate routing nodes, to decrease node batteries or to provoke a general DoS.

Affected data: Application data, packet latency, battery level.

Geographical influence: Attack traces along large network paths starting near the leaves and ending near the base station.

Attack candidates: Path-based DoS, overwhelm, misdirection, spoof, alter or replay routing information, wormhole, sybil to an important routing node, blackhole on an important routing node, sinkhole.

Model 2: Transmission medium attacks

Description: Attacks that affect nearby nodes using the same frequency bands or MAC protocols. Other bands or protocols are not affected. Basically, attackers take advantage of the transmission medium in order to prevent the proper delivery or reception of packets from certain nodes. These attacks applied to routing nodes also hamper the correct communication of other nodes outside of the attacker's direct influence area.

Affected data: Network status data (e.g. RSSI, SNR), application data, packet latency, battery level.

Geographical influence: Reduced area of nearby nodes.

Attack candidates: Unfairness, collision, jamming.

Model 3: Locally dispersed attacks

Description: Attacks that affect dispersed nodes from a single WSN with the main goal of creating delays, dropping packets and depleting node batteries.

Affected data: Application data, packet latency, battery level.

Geographical influence: No geographical influence.

Attack candidates: Misdirection, spoof, alter or replay routing information, sybil, data tampering, wormhole, selective forwarding, sinkhole.

Model 4: Widely dispersed attacks

Description: Attacks that affect dispersed nodes from several WSNs. Attackers aim to reduce the proper operation of one or several WSNs. In this case, attackers do not use constant attack techniques, which would be more effective, in order to cover up their intentions and delay the moment of their discovery.

Affected data: Application data.

Geographical influence: No geographical influence.

Attack candidates: Selective forwarding at a gateway, unfairness at a gateway (not-constant), collision at a gateway (not-constant), jamming at a gateway (not-constant).

Model 5: Widely intensive attacks

Description: Attacks that affect a great percentage of nodes from several WSNs using the same gateway. Attackers use these techniques to completely stop the service provided by one or more WSNs.

Affected data: Application data, battery level.

Geographical influence: Wide area of nearby nodes.

Attack candidates: Blackhole at a gateway, unfairness at a gateway (constant), collision at a gateway (constant), jamming at a gateway (constant), other attacks that crash or isolate the gateway.

Model 6: Local service alteration attacks

Description: Attacks that affect several nearby nodes from the same WSN. The main goal is to alter application information from an area. The attackers either drop application packets or send false information.

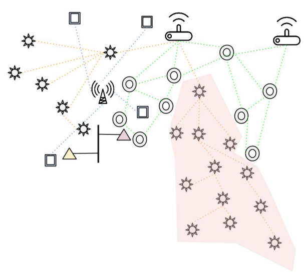
Affected data: Application data.

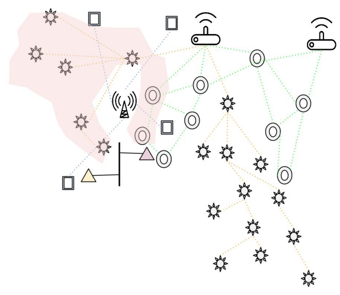
Geographical influence: Reduced area of nearby nodes.

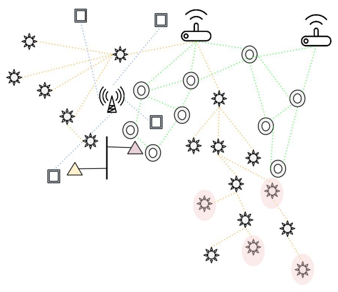
Attack candidates: Blackhole, sinkhole, sybil, data tampering.

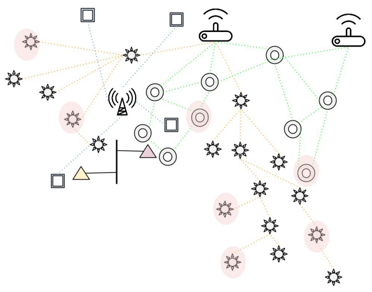
Model 7: Single node attacks

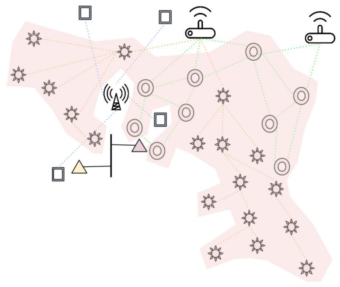
Description: Attacks that aim at depleting the batteries of a single node. This becomes very critical when attackers aim at an important router node in

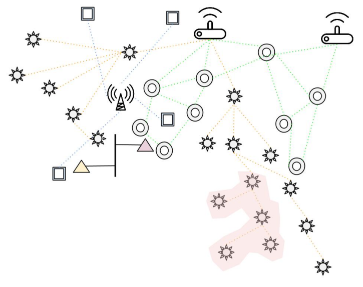

(A) Model 1: vertical attacks

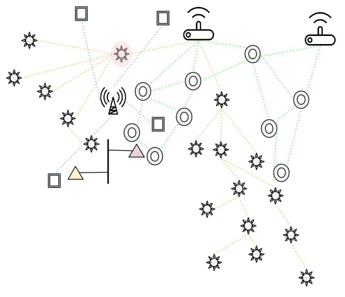

(B) Model 2: transmission medium attacks


(C) Model 3: locally dispersed attacks


(D) Model 4: widely dispersed attacks


(E) Model 5: widely intensive attacks


(F) Model 6: local service alteration attacks


(G) Model 7: single node attacks

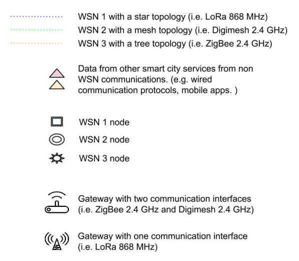

WSN 1 with a star topology (i.e. LoRa 868 MHz)
WSN 2 with a mesh topology (i.e. Digimesh 2.4 GHz)
WSN 3 with a tree topology (i.e. ZigBee 2.4 GHz)
Data from other smart city services from non WSN communications. (e.g. wired communication protocols, mobile apps.)
WSN 1 node
WSN 2 node
WSN 3 node
Gateway with two communication interfaces (i.e. ZigBee 2.4 GHz and Digimesh 2.4 GHz)
Gateway with one communication interface (i.e. LoRa 868 MHz)

FIGURE 6.1: Graphical representation of the seven attack models

network areas with few paths to the sink. Several of these attacks on each path divide the network.

Affected data: Battery level.

Geographical influence: No geographical influence.

Attack candidates: De-synchronization, flooding, sinkhole, collision.

6.4 Classification procedure

As previously seen, the proposed architecture can gather a plethora of AT1, AT2 and AT3 alarms from many different nodes triggered by the same attack. Section 6.2.2 has shown that administrators can implement correlation rules in order to group some of these alarms and, thus, trigger a high severity AT4 alarm to warn them when the evidence of attacks is strong. Then, at the point of receiving an AT4 alarm, administrators have to start inquiring to find out the type of attack and which elements in the network are compromised. This section provides guidelines to classify the alarms into one of the seven models described above. In this way, using the models provide administrators with information about the most likely attacks that fit the evidence.

Figure 6.2 presents the workflow with the classification schema. As the figure shows, the process begins when administrators receive a high severity AT4 alarm (step (a)). Then, administrators have to decide which other alarms registered in the system could be related to the incident and, therefore, have to be collected for analysis (step (b)). In general, these related alarms are the ones triggered by nearby nodes during a short time interval before or after the incident, or by other components of the same network, or the same provider, etc. As a result of step (b) administrators identify a set of alarms that are relevant for the classification of the incident. In the subsequent steps, this set of alarms is analysed to answer the following questions:

- Are multiple nodes affected? (step (c)).
- Are the affected nodes geolocalized together? (step (d)).
- Are multiple services affected? (steps (e) and (k)).
- Are multiple frequency bands affected? (step (f)).
- Is the gateway involved in the attack? (steps (g), (l) and (m)).
- Are the alarms in the set of alarms relevant? (step (i)).
- Do vertical paths show signs of being affected? (step (j)).

It is worth noting that verifying whether a gateway is involved in the attack is usually the responsibility of the gateway provider, who has full access to the gateway. Moreover, as

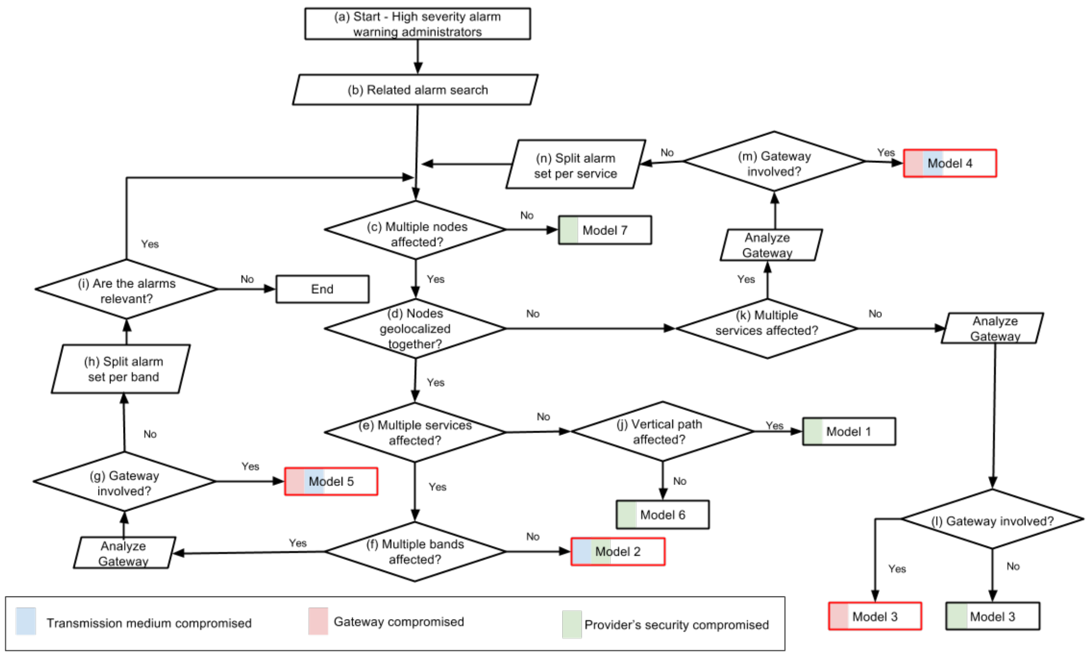

(a) Start - High severity alarm warning administrators
(b) Related alarm search
(n) Split alarm set per service
No
(m) Gateway involved?
Yes
Model 4
(c) Multiple nodes affected?
No
Model 7
Analyze Gateway
Yes
Yes
(i) Are the alarms relevant?
No
End
Yes
(d) Nodes geolocalized together?
No
(k) Multiple services affected?
No
Analyze Gateway
(h) Split alarm set per band
Yes
(e) Multiple services affected?
No
(j) Vertical path affected?
Yes
Model 1
No
No
(g) Gateway involved?
Yes
Model 5
Yes
Model 6
Analyze Gateway
Yes
(f) Multiple bands affected?
No
Model 2
(l) Gateway involved?
Yes
No
Model 3
Model 3
Transmission medium compromised
Gateway compromised
Provider's security compromised

FIGURE 6.2: Procedure to classify attacks into seven attack models according to the evidence in the alarms triggered by the detection engines.

mentioned in Section 6.1, we consider that gateways are not constrained in terms of processing power, are connected to the electricity grid and have reliable telecommunication connections. Therefore, providers can perform complete analysis on the gateways with conventional security tools (e.g. antivirus, IDS) to test if they have been compromised.

In addition to the analysis through the aforementioned questions, the workflow eventually requires splitting the alarm set into several subsets according to a certain criterion (e.g. by frequency band) (steps (h) and (n)). Splitting the alarm set is important to differentiate between several alarms due to different incidents occurring in the same devices around the same time. When there is no model clearly identifiable from the alarm set, as loops (c) - (i) and (c) - (n) show, we propose an iterative process that divides the alarm set into several partitions and each of them is used to start the classification procedure again. Thus, alarms that are not related are divided into different partitions and, therefore, analyzed separately. These unrelated alarms from the same area at the same moment in time can be due to concurrent attacks, false alarms, etc.

Lastly, the final states of the procedure in Figure 6.2 show the attack model that fits the evidence given by an alarm set. The states where third party services or important communication nodes are affected are indicated by a red box.

Below, in order to illustrate the classification process, an example incident is resolved:

1. A high severity alarm (AT4) calls the smart city administrators' attention (step (a)) to a parking WSN controlled by provider A. This AT4 alarm is defined with a correlation rule that groups three or more AT1 alarms (triggered if the received RSSI is above a threshold) in a two hour window. Administrators look in the system for other alarms in the same time window sharing the same area or equipment (step (b)). They find other AT1 alarms from an environmental WSN, which is controlled by provider B. In this case, the AT1 alarms were programmed with a simple rule that would check that the data from the sensors was received in the scheduled intervals. All these alarms make up the alarm set used by the administrators to classify the incident.

2. At this point, administrators start analysing the alarm set by answering the questions proposed in the schema. Firstly, they run a simple query to determine if *multiple nodes are affected* (step (c)). The result of this query is positive.

3. Administrators visually inspect the nodes to figure out if the affected *nodes are geolocalized together* (step (d)). The result of this query is positive.

4. Administrators verify if *multiple services are affected* in all the alarms (step (e)). The result of this query is positive (because the parking service and the environmental monitoring service are affected).

5. Administrators make another simple query to check if *multiple bands are affected* (step (f)). The result of this query is negative (because the WSNs from the two services are configured in the same frequency band).

6. Administrators conclude that the analyzed WSNs are under an attack from model number 2. Moreover, the affected data that triggered the alarms supports this conclusion, i.e. RSSI over the limit in the parking WSN and packet latency increased in the environmental WSN. This is a *transmission medium attack*, where the source of the attack affects nearby nodes using the same frequency bands and/or MAC protocols. In this type of attack, the affected nodes can be from different WSNs and different providers. Likely attacks are: unfairness, collision and jamming.

With this procedure, we point out the most likely attack model, we limit the affected nodes, the area and the providers. Additionally, we also indicate if the attack is compromising the transmission medium (i.e. blue mark), a gateway (i.e. red mark) or the provider's infrastructure (i.e. green mark). Moreover, the previous section listed the attack candidates for each model. The next section recommends contingency plans to mitigate the short and medium term consequences of the attacks.

6.5 Contingency plans

At the end of the procedure in the previous section, the administrators have a sharper picture of the cause of the alarms and, therefore, they can contact the service providers to look for a solution to the problem. In general, solving this type of attack can be a long process, since it involves coordinating several parties and analyzing many devices, which can be difficult to access. Hence, at this moment, besides looking for a solution to patch the possible vulnerabilities, it is also paramount to mark the data from the compromised services and to avoid new data becoming compromised.

Below, basic recommendable strategies to mitigate the possible negative consequences of the attacks are listed. These contingency plans are divided in short and medium-term actions depending on whether other services are affected.

Short-term actions:

Transmission medium compromised

- Data from the compromised service in quarantine
- Data from nearby services in quarantine
- Exclusion area required

Gateway compromised

- Data from the compromised service in quarantine
- Data from other services in the compromised gateway in quarantine
- If other gateways available, then compromised gateway excluded

Provider's security compromised

- Data from all the services of the provider in quarantine

Medium-term actions:

Transmission medium compromised

- Data from the compromised service and the nearby services excluded from learning, statistical and analysis processes

Gateway compromised

- Data from the compromised service and other services in the compromised gateway excluded from learning, statistical and analysis processes

Provider's security compromised

- Data from the services of the compromised provider excluded from learning, statistical and analysis processes

Putting the measures listed above in place avoids the situations where other services compromise new data and where compromised data are used in an urban operation. For instance, in the short-term, we propose establishing exclusion areas when attacks compromise the transmission medium. In this way, mobile devices using WSN technology avoid entering areas where their transmissions can be in danger. In the medium-term, we propose that data from compromised devices should be excluded from business analytical tasks to avoid drawing wrong conclusions. For example, the municipality could decide to expand a parking facility, basing the analysis on compromised WSN data from a parking lot indicating overuse.

Depending on the critical nature of the affected services, it is possible that other contingency strategies are required. Hence, administrators have to analyze the details of each use case to figure out the necessary additional countermeasures.

6.6 Proof of concept

This section includes a proof of concept to demonstrate the use of the proposed classification schema to detect and locate a data availability attack. This type of attack is difficult to detect, can involve several services from different WSNs from different providers and requires the analysis of multiple types of data. The implemented attack in this proof of concept simulates a 20% selective forwarding attack that affects a wide area with dispersed nodes from several WSNs from different providers. Selective forwarding attacks become more obvious as the dropped packet rate and the number of affected nodes increase, and when it affects a reduced and non-dispersed area. Therefore, we analyze the benefits of using the defined schema in a highly complex detection scenario.

6.6.1 Scenario description

In this section, we build a scenario for the demonstration and we analyze the data using the core packages included in R [108] and the packages *e1071* [109] and *caret* [117] for the OC-SVM classification and *FPC* [118] for the clustering algorithms. The original data came from two Barcelona service providers from July to November 2015. The two services collect data from street parkings and sound meters. The data from the parking service include the gateway identifier used by each packet to send the data from the parking sensors to the central servers. The sound data do not include this information. For this proof of concept, it is necessary to extend the real data to have a dataset with the minimum amount of information to be able to simulate attacks and, subsequently, perform anomaly analysis. Therefore, in order to have a proper dataset for this demonstration, the following actions are performed:

- The sound meters are placed in the area of the parking network respecting the layout of both networks. Figure 6.3 shows the node position in both networks.
- A gateway identifier is assigned to each packet received from the sound network. The assignment of a gateway to each sound packet is based on the gateway computational load. Thus, gateways processing many parking network packets receive fewer sound packets. As it will be seen later on, this gateway identifier is only used to simulate the attack. When we perform the analysis later, the gateway identifier is considered unknown in the sound packets.
- The data types described in Section 3.2.1 are received at the data center from diverse services and are generally considered available to any service. However, in this scenario, the sound service does not provide the sequence number of the application packets, which can be used to easily calculate the application packet loss. Therefore, the packet loss rate is added to the sound meter data.
- A normal packet loss rate (i.e. not due to attacks) is also considered. There is a wide variability on ordinary packet loss rates in WSNs, depending on several

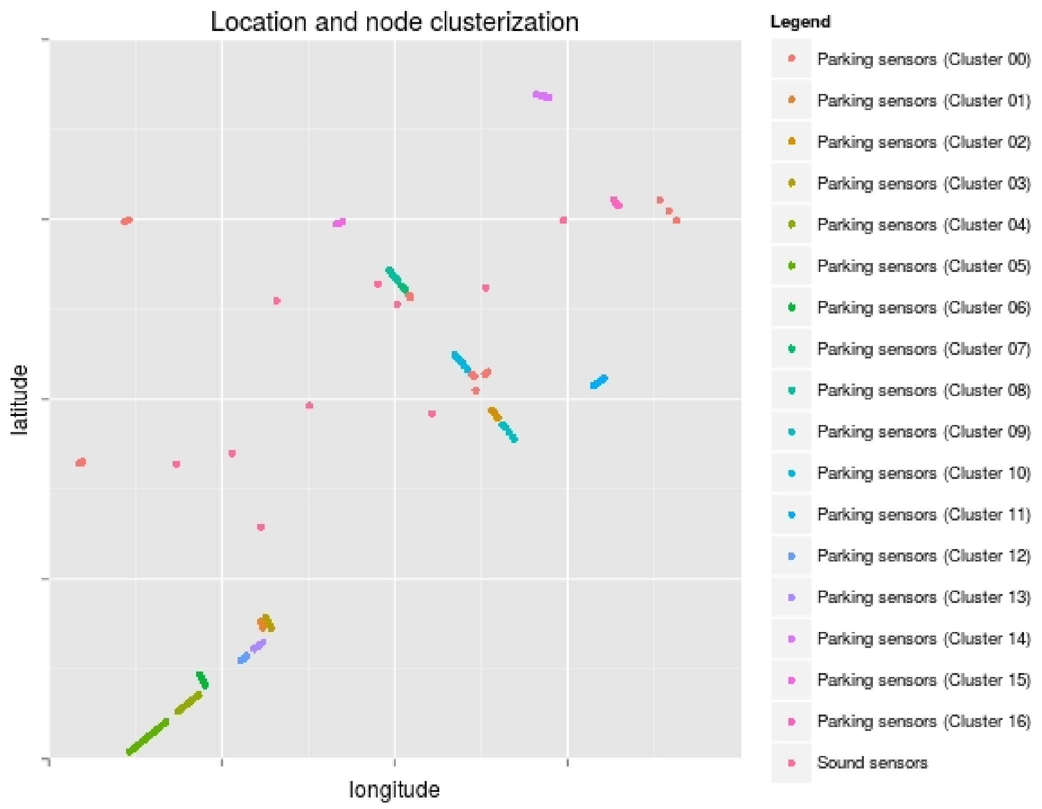


FIGURE 6.3: Sensor positions and division of the parking sensor nodes in clusters.

network characteristics, such as the communication protocols or the node density. In [119], the authors performed measurements using the Collection Tree Protocol (CTP), which is one of the most popular routing protocols in WSNs, and they found end-to-end packet delivery rates from 90.5% to 99.9%. Thus, in this simulation, the sound monitoring service has a conservative 90% packet delivery rate (i.e. 10% packet loss rate without attack).

In this scenario, we simulate a selective forwarding attack in one of the gateways, where 20% of the received packets are randomly dropped. This causes data loss from sensors belonging to the two services spread throughout the neighborhood. This scenario, with only two services, is a basic configuration for a smart city and demonstrates the value of the proposed classification schema. Scenarios including more services and providers would increase the complexity of the analysis, but, at the same time, more alarms from other WSNs would be trigger and, then, there would be more evidence of the attack, which would enhance the value and the results of the classification procedure.

6.6.2 Analysis

This section presents the basic procedures to detect anomalies and trigger alarms (Section 6.6.2.1). Thus, this proof of concept briefly shows how to apply the techniques proposed and reviewed in the previous chapters. Moreover, it shows the importance of implementing correlation rules and demonstrates how to use the classification schema proposed in this chapter (Section 6.6.2.2).

6.6.2.1 Basic detection analysis

In this section, a rule-based technique with a predefined threshold (AT1 type in Section 6.2.1) and an anomaly-based technique (AT2 type in Section 6.2.1) are used on aggregated data from the parking and the sound services to unveil the attacks affecting the smart city.

In order to perform the detection analysis, we first divide the data from this simulation into three sets. 50% of the data is used for training purposes, 25% is used as a validation data to tune the parameters of the algorithms and the last 25% is used as a test dataset. The selective forwarding attack is only applied to the validation and the test data.

We implement three types of alarms to detect the attacks:

- We use **rule-based detection on the sound data**. From this service, we aggregate the number of lost application packets in one hour windows and we define a threshold to determine the maximum number of lost application packets that is considered normal (i.e. not due to an attack). Setting a high threshold implies decreasing the detection rate (i.e. number of detected attacks divided by the number of total attacks) and the false positive rate (i.e. instances that are incorrectly classified as attacks divided by the number of total instances that are not attacks).
- We use **OC-SVM on the parking data**. In this case, the procedure to detect attacks has the following steps:
    1. We group the sensor nodes into clusters by location using DBSCAN[120]. Figure 6.3 shows the division by clusters.
    2. We aggregate the number of changes in the parking spots per cluster in one hour windows.

3. We use OC-SVM to train and test a model in order to verify if the number of changes in each cluster in each window at the corresponding hour of the day is normal. The OC-SVM hyperparameters are used to test different combinations of detection and false positive rates.

- We create a simple **correlation rule** (AT4 type in Section 6.2.1) that is triggered when various alarms from any of the previous two detection techniques have been triggered in one hour. In order build this correlation rule, we compare the detection rate and the false positive rate of varying amounts of alarms triggered by any of the two detection engines. As shown below, by decreasing the number of required alarms in the correlation rule, we can gradually increase both the detection rate and the false positive rate. The results are shown in Figure 6.4.

Generally, alarms based on thresholds are more reliable than alarms triggered by machine learning techniques. If an attacker performing a selective forwarding in the gateway drops a large amount of packets, then the AT1 alarm implemented with a threshold for the number of lost application packets from the sound service would be enough in order to discover the attack. However, the most challenging situations arise when the attacks affect few sound packets and many parking packets, which is the scenario that is implemented in this proof of concept. Thus, we demonstrate the advantages of using correlation rules and the classification schema proposed in this chapter. Figures 6.4 and 6.5 show the performance of the detection techniques in this scenario. Figure 6.4 shows that, in general, for a false positive rate lower than 25%, the combination of alarms from the first two techniques in a correlation rule outperforms the other techniques operating separately. In the smart city context, the false positive rate must be kept low to avoid overwhelming system administrators. As seen in Figure 6.5, setting a maximum 5% false positive rate implies that the detection rate using correlation detection is more than 1.5 times higher than using only OC-SVM and more than 3.5 times higher than using only rule-based detection. Therefore, this outcome shows that administrators can significantly improve detection results by implementing simple correlation rules as shown in this example.

These results are particularly interesting when taking into account that the detection scenario is highly complex due to a 20% selective forwarding dropping rate (only 10% more than the normal loss rate for the sound network) and the difficulty of detecting

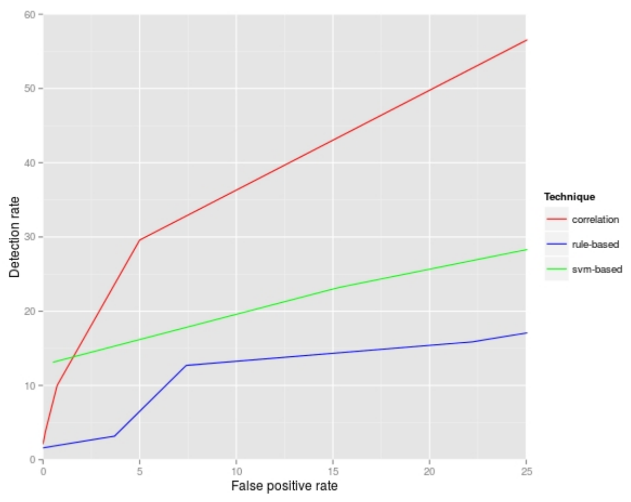


FIGURE 6.4: Evolution of the detection rate and the false positive rate of the three techniques. In the rule-based detection, increasing the packet loss threshold decreases the detection rate and the false positive rate. In the OC-SVM detection, a higher value of the hyperparameter ν increases the detection rate and the false positive rate. In the detection by correlation, including more alarms in the correlation rule decreases the detection rate and the false positive rate.

anomalies in nodes spread in a wide area. There is also a need to detect the attacks within a one hour window, which allows time to apply short-term contingency actions. Therefore, the performance of the proposed detection process would increase in scenarios with a higher dropping rate, where the attack is focused on a narrow area or without short-term contingency requirements.

This simulated detection process also shows that alarms triggered by the correlation rule must be considered as highly severe. The other alarms are also important to trace incidents in the smart city, but they are less reliable and provide less information about the incidents than correlation rules. In the next section, starting from the reception of a highly severe alarm, the proposed schema is used to classify the incident into an attack model.

6.6.2.2 Enhanced analysis with attack classification

In this proof of concept, AT1 and AT2 alarms can be constantly triggered and, therefore, do not deserve administrators' attention until an AT4 alarm is received. Upon receiving an AT4 alarm (step (a) in the schema shown in Figure 6.2), administrators can proceed with the other steps in the classification procedure:

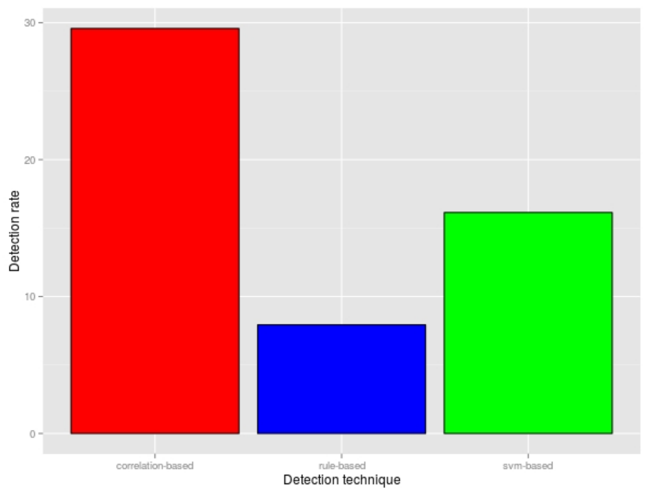


FIGURE 6.5: Detection rate comparison between the three techniques at a 5% false positive rate.

1. Administrators must retrieve other alarms from the same service, from the same area or from some network that shares important components with the network that triggered the alarm (step (b)). Thus, in this scenario, alarms from the parking and sound service have to be retrieved and make up the base alarm set for this analysis.

2. The answer to the question *Multiple nodes affected?* is *Yes* (step (c)).

3. The answer to the question *Nodes geolocalized together?* is *No* (step (d)), since Figure 6.6 shows that the anomalies detected in the sound sensors and parking clusters are sparsely distributed.

4. The answer to the question *Multiple services affected?* is *Yes* (step (k)).

5. At this point administrators could have already taken preventive countermeasures involving the transmission medium and the gateway. They should request the gateway provider to undertake a comprehensive analysis of the gateway (step (m)) to confirm that the networks are under an attack model number 4. As it can be seen in Section 6.3, this corresponds to a *widely dispersed attack*, where the most likely attack candidates aim at the gateway and match the traces of a selective forwarding, a non-constant unfairness, a non-constant collision or a non-constant jamming. In this situation, the basic recommended contingency strategies involve marking an exclusion area near the gateway, redirecting WSN traffic to other gateways (if

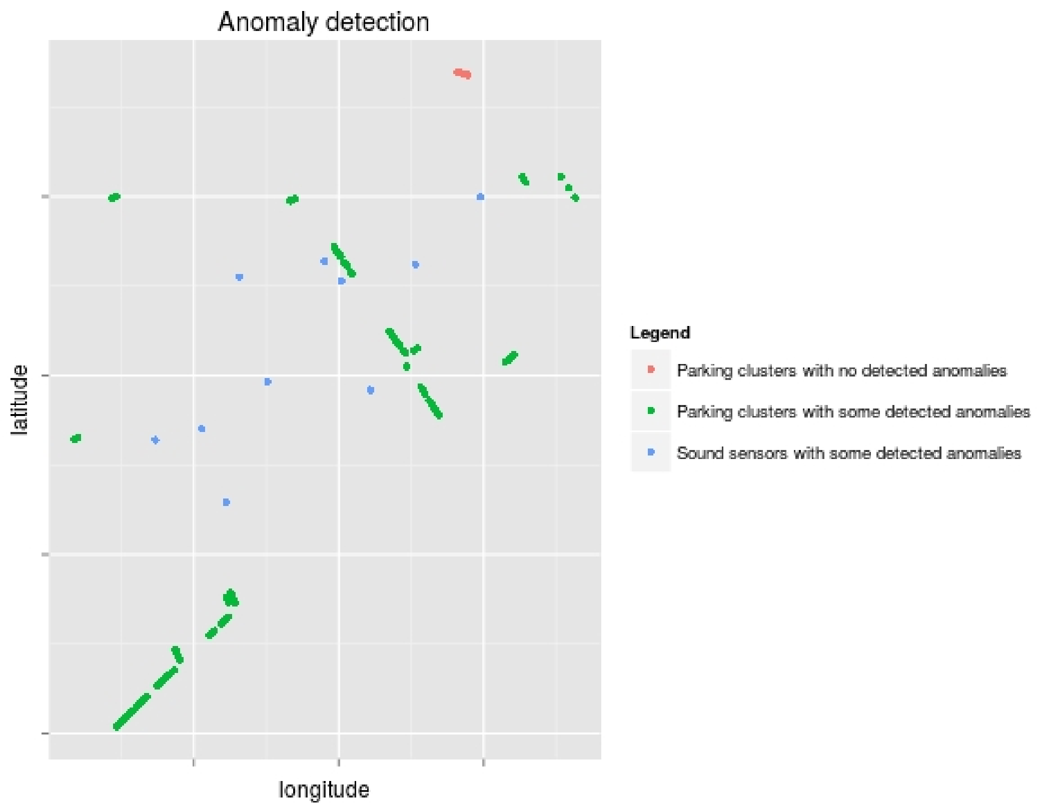


FIGURE 6.6: Sound sensors and parking sensor clusters where the attack has been detected.

possible) and quarantining the data from the compromised gateway, the nearby services and other services that were using this gateway.

As this example has shown, using the classification schema and taking into account the alarms from the smart city WSNs as a whole, smart city administrators not only detect anomalies in a more reliable way, but are also provided with a clearer picture of the attacks causing the incidents.

6.7 Conclusions

In this chapter, we have extended the intrusion detection platform presented in the previous chapters with a schema to classify the evidence left by attacks against smart city WSNs into seven different attack models. This schema provides smart city administrators with guidelines to identify the attacks and the compromised network components. These models have been proposed by taking into account the effects that the attacks have on the components of smart city WSNs in a generic way. For each attack model, we have provided a list of attack types to narrow down the most likely cause of attack and we have provided a set of contingency plans to mitigate short and medium term consequences of the attacks. This schema does not claim to be comprehensive and, therefore, it does not include all the possible attacks for all possible smart city configurations. This schema has been designed to be adaptable to many smart cities and it should be treated as a guideline to develop a security and incident response system for smart city WSNs.

This chapter has also shown that the combined use of rule-based detection and OC-SVM by means of simple correlation rules can significantly improve detection results. The combination of these techniques in a correlation rule can outperform the other techniques operating separately, and this has been shown in a complex detection scenario with a 20% selective forwarding dropping rate (only 10% more than the normal loss rate for the sound network). To the best of our knowledge, this is the first approach towards studying the effects of correlating WSN security analysis of different services in the smart city. The proof of concept has also exemplified the procedure to follow to figure out the most likely attacks and components compromising two WSNs.

Chapter 7

Conclusions

In recent years, smart city projects are gaining importance in the urban development of many cities around the world. This has involved acquiring new ways of managing the cities, which have generally been based on technological solutions that gather and process large amounts of city data. To this end, public administrations, which aim at developing smart city solutions, normally deploy WSNs in order to collect data from the streets and, in this way, obtain information about the operation of the metropolitan infrastructures.

Nonetheless, a massive deployment of WSNs in an unprotected environment, like the streets, raises some security concerns. Furthermore, public administrations are generally outsourcing the installation and maintenance of the WSNs to external providers. These facts create scenarios with several barriers to security, from which, this thesis has highlighted three. Firstly, outsourcing potentiates the heterogeneous environment of the smart city. Each urban service demands a different level of security and each provider offers a different solution to implement a system. Secondly, network devices administered by the providers become less accessible for the public administration. This, in many cases, is an impediment to access system logs and to monitor the security health of the network. And thirdly, WSNs are generally designed to be highly efficient in order to reduce energy consumption and extend battery life. This results, in some cases, in the fact that downstream communication is not implemented, which hardens software updates, key exchange, etc. Therefore, finding generalizable security solutions to protect the WSNs that can cope with the heterogeneity of the smart city and that are also efficient and adaptable enough to be installed in the sensor nodes is unfeasible.

Currently, in order to protect the WSNs, public administrations include security clauses in the service-level agreements with the external providers. Accordingly, security mechanisms are in hands of the providers. Generally, the providers embed countermeasures based on cryptography, obfuscation, frequency hopping and so on in the sensor nodes. However, these security measures are only effective if they are properly applied and maintained, and, in front of severe attacks, they are totally futile. Thus, in this scenario, smart city administrators must have mechanisms to verify their WSNs operation, so that they can urge, if necessary, external providers to apply the required security measures. In this context, this research work aims at contributing to increase smart city WSN security from the point of view of the smart city administrators. The rest of this chapter presents the conclusions of this thesis in Section 7.1, and future work directions in Section 7.2.

7.1 Conclusions

As a first contribution in this dissertation, we have proposed a centralized architecture to gather all the available application and network status data from the urban WSNs in order to analyze them and disclose attacks. In this way, this architecture contributes towards a centralised intrusion detection platform for smart cities. The proposed architecture has been designed to be non-intrusive and transparent to the WSN providers. The architecture design also takes into account that different smart cities require different services and that different providers use different technologies. The architecture and the algorithms included in this thesis intend to be portable to many smart city models. Consequently, we studied the characteristics of current smart city projects and we have abstracted the proposed architecture from any specific smart city configuration. Hence, the proposed system is easily integrable and adaptable to many smart cities, and the proposed detection algorithms can be applied in many WSN types.

In the proposed architecture, intrusion detection is basically handled by two detection engines: a rule-based detection engine and an anomaly-based detection engine. The rule-based detection engine looks for patterns of attacks that have previously been recorded in signature databases. Although this mechanism is highly effective to detect certain attacks, it has the main drawback that unknown attacks, for which there are still no signatures, go unnoticed. Moreover, creating rules that involve many variables becomes too complex and difficult to maintain, and defining static thresholds for highly dynamic systems is sometimes unfeasible. On the other hand, anomaly-based detection normally uses machine learning and statistical techniques to discover data that deviate from normality. In this way, these types of techniques are capable of disclosing unknown attacks. However, they are not fully reliable and they trigger a certain amount of false alarms. Therefore, it is necessary to combine the two types of detection engines in order to avoid an excessive amount of false positives and also to be able to detect unknown attacks. Incorporating a correlation system which brings together alarms triggered by both detection engines has significantly shown an increase of the detection rate. Additionally, it reduces the number of relevant alarms that need the administrator's attention.

This thesis has put more focus on the anomaly-based detection engine because it can offer more flexibility and adaptability to different WSNs than the rule-based engine. Instead of using static rules, the anomaly-based engine uses mathematical models that are constantly updated using the data gathered from the WSNs. In this way, this engine is responsible to create the models that define the normal behavior of the variables and, then, use these models to verify that new data from the WSNs come without anomalies. This engine has to be capable of finding normality boundaries for single variables and also identify not normal situations considering the relationship between several variables at the same time. In this thesis, we have studied several multivariate anomaly detection techniques, and we have concluded that OC-SVM is very suitable in this context. This is a semi-supervised machine learning technique, which has given good detection results in several scenarios incorporating in the models different application and network status variables from the different layers of the WSN communication protocols. As a result of the studies included in this dissertation, we have determined that the sequence number of the application packet and the battery level are the minimum network status information that providers have to send from the sensor nodes to the smart city central servers to be able to run successful anomaly analysis.

Furthermore, this thesis has shown that intrusion detection with the proposed methods requires several steps. For instance, data have to be pre-processed and aggregated, and machine learning models need to be trained and, then, anomaly analysis can be performed. We have analyzed the computational complexity of the different steps and we have identified model computation and intrusion detection as the most critical subprocesses among these steps. On the one hand, model computation has to be considered critical, because the computational complexity of training machine learning models is generally very high. However, this action is executed very seldom. On the other hand, anomaly detection is generally computationally inexpensive, but it has to be executed very often. This thesis validates that the pipeline involving all the required steps is viable even in scenarios involving big data without having to rely on a hardware architecture with exceptionally high computational resources.

Additionally, one of the main challenges for smart city administrators is not only to detect that an attack is compromising the WSNs of the external providers, but also to identify the specific attack. This thesis has provided guidelines to gather the evidences of the attack and then point out one of seven proposed attack models. In this way, smart city administrators narrow down the possible attack type affecting their networks and they can also figure out the compromised devices and some mitigation strategies to limit short and medium term harmful consequences of the attacks.

Summing up, in this thesis we have proposed a system that contributes to improve smart city WSN security in a generic manner. The solutions proposed in this thesis are suitable to be adapted and deployed to several smart city models. In order to adapt the proposed solutions, smart cities need to further study the consequences of attacks in their particular scenarios and extend or reduce the solutions proposed in this thesis according to their circumstances. In this way, the proposed solutions can help smart city administrators to enhance security, to mitigate the consequences of attacks, to increase data quality, to monitor that providers apply the necessary security countermeasures in their networks, and, in general, to improve WSN security as a whole.

As a result of our work, we have seen that intrusion detection in a smart city is a very complex problem. A black-box solution with a multipurpose detection algorithm that covers most of the attacks for most of the configurations is not feasible. This thesis is a first contribution on this research field and it does not aim to include a comprehensive intrusion detection system capable of disclosing any attack targeting any possible business case in any type of smart city. Therefore, smart city administrators can use the tools proposed in this thesis as a basis and adapt these solutions to the particular cases in their cities.

7.2 Future work

This thesis has presented generic methods that can be used in many smart city situations focusing on problems concerning typical attacks that can affect several smart city WSNs. As future work, it would be of great value to provide the best ways to detect attacks taking into account the specificities of the typical services included in the smart cities. Although we have already suggested certain algorithms to set thresholds up, data integrity attacks have different effects for application data in the different services and, therefore, in order to have more effective detection, it is necessary to study how attacks affect the most important variables on each service or in the network, and then find the best algorithm to detect them. For example, it would be useful to find the best way to automatically determine the boundaries for sound meters or for the RSSI in the most used WSNs configurations.

Moreover, it is also important to enhance the guidelines provided in this thesis with a methodology to configure the necessary parameters to run the proposed intrusion detection processes. This would be very useful, for instance, to set up the OC-SVM parameters or to find the most adequate time window size to aggregate data. Furthermore, the guidelines of the thesis have to also be enhanced with a methodology to build correlation rules integrating alarms from the two detection engines.

It also remains as future work to find a clustering methodology to divide the networks in small areas, with which administrators can apply the detection techniques and withdraw relevant conclusions bearing in mind the compromised frequency bands, protocols, node location, etc. The experiments performed in this thesis have used small networks or the scenarios were divided ad-hoc. However, in a real smart city situation with multiple WSNs, these tasks need to be handled by algorithms in order to make them scalable.

Finally, we have proposed guidelines that narrow down the list of candidate attacks compromising a system, the affected components and providers. However, in a real smart city it is necessary to have an automatic system including a comprehensive list of attacks and other possible causes of network malfunctioning. The rule-based detection engine is theoretically capable of identifying any type of known attack if it leaves traces on the data. However, as far as we know, there are currently no public signature databases to identify attacks in this context. Additionally, if large datasets of labeled data from the smart city WSNs were available, then, it would be possible to use supervised machine learning algorithms to train models to classify the anomalies into specific attacks. The availability of large datasets of labeled data would also allow studying the performance of deep learning techniques in this context. The analysis of these type of techniques also remains as future work.

Appendix A

Supplementary materials for Chapter 4

This Appendix contains supplementary materials for Chapter 4. Each table shows the results of the experiments considering a different feature vector and/or a different PFPR.

Table A.1: Comparative study results for the FV1 dataset with a very restrictive PFPR.

attack	**technique**	**F-score**	**FPR**	**TPR**
All attacks	ocsvm	0.60	0.71	0.73
All attacks	lofactor	0.12	0.05	0.06
All attacks	mahalanobis	0.05	0.05	0.02
All attacks	hierarchical clustering	0.12	0.06	0.07
Jamming near 4	ocsvm	0.63	0.67	0.77
Jamming near 4	lofactor	0.21	0.05	0.12
Jamming near 4	mahalanobis	0.10	0.05	0.05
Jamming near 4	hierarchical clustering	0.18	0.06	0.10
Jamming near 9	ocsvm	0.62	0.70	0.76
Jamming near 9	lofactor	0.13	0.05	0.07
Jamming near 9	mahalanobis	0.10	0.05	0.05
Jamming near 9	hierarchical clustering	0.07	0.06	0.04
Jamming near BS	ocsvm	0.77	0.26	0.79
Jamming near BS	lofactor	0.04	0.05	0.02
Jamming near BS	mahalanobis	0.02	0.05	0.01
Jamming near BS	hierarchical clustering	0.08	0.06	0.04
Selective forwarding 30%	ocsvm	0.54	0.97	0.73
Selective forwarding 30%	lofactor	0.11	0.05	0.06
Selective forwarding 30%	mahalanobis	0.03	0.05	0.01
Selective forwarding 30%	hierarchical clustering	0.06	0.06	0.03
Selective forwarding 50%	ocsvm	0.57	0.95	0.79
Selective forwarding 50%	lofactor	0.08	0.05	0.05
Selective forwarding 50%	mahalanobis	0.05	0.05	0.03
Selective forwarding 50%	hierarchical clustering	0.07	0.06	0.03
Selective forwarding 70%	ocsvm	0.58	0.90	0.78
Selective forwarding 70%	lofactor	0.05	0.05	0.03
Selective forwarding 70%	mahalanobis	0.03	0.05	0.02
Selective forwarding 70%	hierarchical clustering	0.14	0.06	0.08
Selective forwarding 90%	ocsvm	0.61	0.72	0.75
Selective forwarding 90%	lofactor	0.07	0.05	0.04
Selective forwarding 90%	mahalanobis	0.03	0.05	0.02
Selective forwarding 90%	hierarchical clustering	0.15	0.06	0.08

Table A.2: Comparative study results for the FV1 dataset with a restrictive PFPR.

attack	**technique**	**F-score**	**FPR**	**TPR**
All attacks	ocsvm	0.6	0.7	0.72
All attacks	lofactor	0.2	0.1	0.11
All attacks	mahalanobis	0.11	0.1	0.06
All attacks	hierarchical clustering	0.26	0.1	0.16
Jamming near 4	ocsvm	0.63	0.67	0.76
Jamming near 4	lofactor	0.31	0.1	0.19
Jamming near 4	mahalanobis	0.19	0.1	0.11
Jamming near 4	hierarchical clustering	0.32	0.1	0.2
Jamming near 9	ocsvm	0.63	0.69	0.77
Jamming near 9	lofactor	0.3	0.1	0.19
Jamming near 9	mahalanobis	0.17	0.1	0.09
Jamming near 9	hierarchical clustering	0.2	0.1	0.12
Jamming near BS	ocsvm	0.79	0.23	0.79
Jamming near BS	lofactor	0.11	0.1	0.06
Jamming near BS	mahalanobis	0.07	0.1	0.04
Jamming near BS	hierarchical clustering	0.2	0.1	0.12
Selective forwarding 30%	ocsvm	0.54	0.97	0.72
Selective forwarding 30%	lofactor	0.18	0.1	0.1
Selective forwarding 30%	mahalanobis	0.09	0.1	0.05
Selective forwarding 30%	hierarchical clustering	0.18	0.1	0.1
Selective forwarding 50%	ocsvm	0.57	0.94	0.78
Selective forwarding 50%	lofactor	0.15	0.1	0.08
Selective forwarding 50%	mahalanobis	0.11	0.1	0.06
Selective forwarding 50%	hierarchical clustering	0.19	0.1	0.11
Selective forwarding 70%	ocsvm	0.58	0.9	0.78
Selective forwarding 70%	lofactor	0.12	0.1	0.07
Selective forwarding 70%	mahalanobis	0.07	0.1	0.04
Selective forwarding 70%	hierarchical clustering	0.24	0.1	0.14
Selective forwarding 90%	ocsvm	0.61	0.7	0.76
Selective forwarding 90%	lofactor	0.11	0.1	0.06
Selective forwarding 90%	mahalanobis	0.06	0.1	0.03
Selective forwarding 90%	hierarchical clustering	0.3	0.1	0.19

Table A.3: Comparative study results for the FV1 dataset with a permissive PFPR.

attack	**technique**	**F-score**	**FPR**	**TPR**
All attacks	ocsvm	0.58	0.68	0.69
All attacks	lofactor	0.25	0.15	0.15
All attacks	mahalanobis	0.12	0.15	0.07
All attacks	hierarchical clustering	0.39	0.16	0.26
Jamming near 4	ocsvm	0.62	0.64	0.73
Jamming near 4	lofactor	0.38	0.15	0.25
Jamming near 4	mahalanobis	0.22	0.15	0.13
Jamming near 4	hierarchical clustering	0.40	0.16	0.27
Jamming near 9	ocsvm	0.62	0.67	0.74
Jamming near 9	lofactor	0.35	0.15	0.23
Jamming near 9	mahalanobis	0.19	0.15	0.11
Jamming near 9	hierarchical clustering	0.29	0.16	0.18
Jamming near BS	ocsvm	0.79	0.16	0.76
Jamming near BS	lofactor	0.14	0.15	0.08
Jamming near BS	mahalanobis	0.09	0.15	0.05
Jamming near BS	hierarchical clustering	0.42	0.16	0.28
Selective forwarding 30%	ocsvm	0.52	0.96	0.69
Selective forwarding 30%	lofactor	0.23	0.15	0.14
Selective forwarding 30%	mahalanobis	0.11	0.15	0.06
Selective forwarding 30%	hierarchical clustering	0.26	0.16	0.16
Selective forwarding 50%	ocsvm	0.56	0.91	0.75
Selective forwarding 50%	lofactor	0.21	0.15	0.13
Selective forwarding 50%	mahalanobis	0.11	0.15	0.06
Selective forwarding 50%	hierarchical clustering	0.28	0.16	0.18
Selective forwarding 70%	ocsvm	0.57	0.85	0.74
Selective forwarding 70%	lofactor	0.19	0.15	0.11
Selective forwarding 70%	mahalanobis	0.08	0.15	0.05
Selective forwarding 70%	hierarchical clustering	0.36	0.16	0.24
Selective forwarding 90%	ocsvm	0.62	0.66	0.74
Selective forwarding 90%	lofactor	0.15	0.15	0.09
Selective forwarding 90%	mahalanobis	0.06	0.15	0.03
Selective forwarding 90%	hierarchical clustering	0.44	0.16	0.30

Table A.4: Comparative study results for the FV2 dataset with a very restrictive PFPR.

attack	**technique**	**F-score**	**FPR**	**TPR**
All attacks	ocsvm	0.85	0.02	0.76
All attacks	lofactor	0.60	0.05	0.44
All attacks	mahalanobis	0.64	0.05	0.49
All attacks	hierarchical clustering	0.31	0.07	0.19
Jamming near 4	ocsvm	0.86	0.00	0.75
Jamming near 4	lofactor	0.69	0.05	0.53
Jamming near 4	mahalanobis	0.67	0.05	0.52
Jamming near 4	hierarchical clustering	0.49	0.07	0.33
Jamming near 9	ocsvm	0.88	0.00	0.78
Jamming near 9	lofactor	0.67	0.05	0.52
Jamming near 9	mahalanobis	0.67	0.05	0.52
Jamming near 9	hierarchical clustering	0.35	0.07	0.22
Jamming near BS	ocsvm	0.86	0.00	0.76
Jamming near BS	lofactor	0.68	0.05	0.52
Jamming near BS	mahalanobis	0.67	0.05	0.51
Jamming near BS	hierarchical clustering	0.62	0.07	0.47
Selective forwarding 30%	ocsvm	0.81	0.12	0.76
Selective forwarding 30%	lofactor	0.22	0.05	0.13
Selective forwarding 30%	mahalanobis	0.60	0.05	0.44
Selective forwarding 30%	hierarchical clustering	0.00	0.07	0.00
Selective forwarding 50%	ocsvm	0.82	0.05	0.73
Selective forwarding 50%	lofactor	0.50	0.05	0.34
Selective forwarding 50%	mahalanobis	0.61	0.05	0.45
Selective forwarding 50%	hierarchical clustering	0.00	0.07	0.00
Selective forwarding 70%	ocsvm	0.87	0.03	0.79
Selective forwarding 70%	lofactor	0.60	0.05	0.44
Selective forwarding 70%	mahalanobis	0.63	0.05	0.47
Selective forwarding 70%	hierarchical clustering	0.23	0.07	0.14
Selective forwarding 90%	ocsvm	0.84	0.01	0.73
Selective forwarding 90%	lofactor	0.64	0.05	0.48
Selective forwarding 90%	mahalanobis	0.64	0.05	0.48
Selective forwarding 90%	hierarchical clustering	0.44	0.07	0.29

Table A.5: Comparative study results for the FV2 dataset with a restrictive PFPR.

attack	**technique**	**F-score**	**FPR**	**TPR**
All attacks	ocsvm	0.85	0.02	0.76
All attacks	lofactor	0.63	0.10	0.48
All attacks	mahalanobis	0.66	0.10	0.51
All attacks	hierarchical clustering	0.52	0.10	0.37
Jamming near 4	ocsvm	0.86	0.00	0.75
Jamming near 4	lofactor	0.69	0.10	0.55
Jamming near 4	mahalanobis	0.68	0.10	0.54
Jamming near 4	hierarchical clustering	0.68	0.10	0.54
Jamming near 9	ocsvm	0.88	0.00	0.78
Jamming near 9	lofactor	0.67	0.10	0.53
Jamming near 9	mahalanobis	0.68	0.10	0.55
Jamming near 9	hierarchical clustering	0.66	0.10	0.52
Jamming near BS	ocsvm	0.86	0.00	0.76
Jamming near BS	lofactor	0.69	0.10	0.55
Jamming near BS	mahalanobis	0.67	0.10	0.53
Jamming near BS	hierarchical clustering	0.68	0.10	0.54
Selective forwarding 30%	ocsvm	0.81	0.12	0.76
Selective forwarding 30%	lofactor	0.35	0.10	0.22
Selective forwarding 30%	mahalanobis	0.61	0.10	0.46
Selective forwarding 30%	hierarchical clustering	0.11	0.10	0.06
Selective forwarding 50%	ocsvm	0.82	0.06	0.73
Selective forwarding 50%	lofactor	0.56	0.10	0.41
Selective forwarding 50%	mahalanobis	0.63	0.10	0.49
Selective forwarding 50%	hierarchical clustering	0.29	0.10	0.18
Selective forwarding 70%	ocsvm	0.87	0.03	0.79
Selective forwarding 70%	lofactor	0.61	0.10	0.47
Selective forwarding 70%	mahalanobis	0.65	0.10	0.50
Selective forwarding 70%	hierarchical clustering	0.54	0.10	0.39
Selective forwarding 90%	ocsvm	0.84	0.01	0.73
Selective forwarding 90%	lofactor	0.65	0.10	0.50
Selective forwarding 90%	mahalanobis	0.66	0.10	0.51
Selective forwarding 90%	hierarchical clustering	0.59	0.10	0.44

Table A.6: Comparative study results for the FV2 dataset with a permissive PFPR.

attack	**technique**	**F-score**	**FPR**	**TPR**
All attacks	ocsvm	0.81	0.02	0.70
All attacks	lofactor	0.64	0.15	0.51
All attacks	mahalanobis	0.67	0.15	0.54
All attacks	hierarchical clustering	0.67	0.21	0.55
Jamming near 4	ocsvm	0.81	0.00	0.67
Jamming near 4	lofactor	0.69	0.15	0.57
Jamming near 4	mahalanobis	0.68	0.15	0.55
Jamming near 4	hierarchical clustering	0.73	0.21	0.63
Jamming near 9	ocsvm	0.83	0.00	0.70
Jamming near 9	lofactor	0.67	0.15	0.55
Jamming near 9	mahalanobis	0.69	0.15	0.56
Jamming near 9	hierarchical clustering	0.73	0.21	0.64
Jamming near BS	ocsvm	0.83	0.00	0.70
Jamming near BS	lofactor	0.69	0.15	0.57
Jamming near BS	mahalanobis	0.68	0.15	0.55
Jamming near BS	hierarchical clustering	0.73	0.21	0.63
Selective forwarding 30%	ocsvm	0.77	0.11	0.70
Selective forwarding 30%	lofactor	0.38	0.15	0.26
Selective forwarding 30%	mahalanobis	0.63	0.15	0.50
Selective forwarding 30%	hierarchical clustering	0.41	0.21	0.28
Selective forwarding 50%	ocsvm	0.78	0.05	0.67
Selective forwarding 50%	lofactor	0.57	0.15	0.43
Selective forwarding 50%	mahalanobis	0.66	0.15	0.53
Selective forwarding 50%	hierarchical clustering	0.60	0.21	0.47
Selective forwarding 70%	ocsvm	0.83	0.03	0.73
Selective forwarding 70%	lofactor	0.62	0.15	0.49
Selective forwarding 70%	mahalanobis	0.65	0.15	0.52
Selective forwarding 70%	hierarchical clustering	0.66	0.21	0.54
Selective forwarding 90%	ocsvm	0.80	0.01	0.68
Selective forwarding 90%	lofactor	0.65	0.15	0.52
Selective forwarding 90%	mahalanobis	0.67	0.15	0.54
Selective forwarding 90%	hierarchical clustering	0.69	0.21	0.58

TABLE A.7: Comparative study results for the FV3 dataset with a very restrictive PFPR.

attack	**technique**	**F-score**	**FPR**	**TPR**
All attacks	ocsvm	0.87	0.03	0.80
All attacks	lofactor	0.51	0.05	0.35
All attacks	mahalanobis	0.53	0.05	0.37
All attacks	hierarchical clustering	0.34	0.05	0.21
Jamming near 4	ocsvm	0.90	0.00	0.82
Jamming near 4	lofactor	0.69	0.05	0.54
Jamming near 4	mahalanobis	0.57	0.05	0.40
Jamming near 4	hierarchical clustering	0.63	0.05	0.47
Jamming near 9	ocsvm	0.90	0.00	0.82
Jamming near 9	lofactor	0.68	0.05	0.52
Jamming near 9	mahalanobis	0.58	0.05	0.42
Jamming near 9	hierarchical clustering	0.61	0.05	0.45
Jamming near BS	ocsvm	0.90	0.00	0.81
Jamming near BS	lofactor	0.68	0.05	0.53
Jamming near BS	mahalanobis	0.47	0.05	0.31
Jamming near BS	hierarchical clustering	0.65	0.05	0.50
Selective forwarding 30%	ocsvm	0.82	0.15	0.80
Selective forwarding 30%	lofactor	0.12	0.05	0.07
Selective forwarding 30%	mahalanobis	0.44	0.05	0.29
Selective forwarding 30%	hierarchical clustering	0.01	0.05	0.00
Selective forwarding 50%	ocsvm	0.87	0.09	0.83
Selective forwarding 50%	lofactor	0.22	0.05	0.12
Selective forwarding 50%	mahalanobis	0.51	0.05	0.35
Selective forwarding 50%	hierarchical clustering	0.01	0.05	0.01
Selective forwarding 70%	ocsvm	0.88	0.05	0.82
Selective forwarding 70%	lofactor	0.41	0.05	0.27
Selective forwarding 70%	mahalanobis	0.55	0.05	0.39
Selective forwarding 70%	hierarchical clustering	0.04	0.05	0.02
Selective forwarding 90%	ocsvm	0.88	0.02	0.80
Selective forwarding 90%	lofactor	0.58	0.05	0.42
Selective forwarding 90%	mahalanobis	0.55	0.05	0.39
Selective forwarding 90%	hierarchical clustering	0.15	0.05	0.08

Table A.8: Comparative study results for the FV3 dataset with a restrictive PFPR.

attack	**technique**	**F-score**	**FPR**	**TPR**
All attacks	ocsvm	0.86	0.03	0.77
All attacks	lofactor	0.55	0.10	0.39
All attacks	mahalanobis	0.60	0.10	0.45
All attacks	hierarchical clustering	0.54	0.14	0.40
Jamming near 4	ocsvm	0.89	0.00	0.80
Jamming near 4	lofactor	0.69	0.10	0.55
Jamming near 4	mahalanobis	0.61	0.10	0.46
Jamming near 4	hierarchical clustering	0.71	0.14	0.59
Jamming near 9	ocsvm	0.89	0.00	0.80
Jamming near 9	lofactor	0.67	0.10	0.53
Jamming near 9	mahalanobis	0.63	0.10	0.49
Jamming near 9	hierarchical clustering	0.68	0.14	0.55
Jamming near BS	ocsvm	0.89	0.00	0.80
Jamming near BS	lofactor	0.69	0.10	0.55
Jamming near BS	mahalanobis	0.53	0.10	0.38
Jamming near BS	hierarchical clustering	0.69	0.14	0.57
Selective forwarding 30%	ocsvm	0.80	0.15	0.77
Selective forwarding 30%	lofactor	0.17	0.10	0.10
Selective forwarding 30%	mahalanobis	0.56	0.10	0.41
Selective forwarding 30%	hierarchical clustering	0.28	0.14	0.17
Selective forwarding 50%	ocsvm	0.86	0.09	0.83
Selective forwarding 50%	lofactor	0.30	0.10	0.18
Selective forwarding 50%	mahalanobis	0.58	0.10	0.43
Selective forwarding 50%	hierarchical clustering	0.27	0.14	0.17
Selective forwarding 70%	ocsvm	0.87	0.05	0.81
Selective forwarding 70%	lofactor	0.48	0.10	0.33
Selective forwarding 70%	mahalanobis	0.62	0.10	0.47
Selective forwarding 70%	hierarchical clustering	0.44	0.14	0.30
Selective forwarding 90%	ocsvm	0.87	0.02	0.78
Selective forwarding 90%	lofactor	0.60	0.10	0.45
Selective forwarding 90%	mahalanobis	0.60	0.10	0.45
Selective forwarding 90%	hierarchical clustering	0.55	0.14	0.41

Table A.9: Comparative study results for the FV3 dataset with a permissive PFPR.

attack	**technique**	**F-score**	**FPR**	**TPR**
All attacks	ocsvm	0.84	0.03	0.75
All attacks	lofactor	0.57	0.15	0.43
All attacks	mahalanobis	0.62	0.15	0.48
All attacks	hierarchical clustering	0.62	0.22	0.49
Jamming near 4	ocsvm	0.88	0.00	0.78
Jamming near 4	lofactor	0.68	0.15	0.56
Jamming near 4	mahalanobis	0.62	0.15	0.49
Jamming near 4	hierarchical clustering	0.71	0.22	0.62
Jamming near 9	ocsvm	0.88	0.00	0.79
Jamming near 9	lofactor	0.67	0.15	0.54
Jamming near 9	mahalanobis	0.65	0.15	0.51
Jamming near 9	hierarchical clustering	0.71	0.22	0.61
Jamming near BS	ocsvm	0.88	0.00	0.78
Jamming near BS	lofactor	0.68	0.15	0.56
Jamming near BS	mahalanobis	0.54	0.15	0.40
Jamming near BS	hierarchical clustering	0.70	0.22	0.60
Selective forwarding 30%	ocsvm	0.80	0.14	0.75
Selective forwarding 30%	lofactor	0.23	0.15	0.14
Selective forwarding 30%	mahalanobis	0.60	0.15	0.46
Selective forwarding 30%	hierarchical clustering	0.37	0.22	0.25
Selective forwarding 50%	ocsvm	0.85	0.08	0.81
Selective forwarding 50%	lofactor	0.35	0.15	0.22
Selective forwarding 50%	mahalanobis	0.60	0.15	0.46
Selective forwarding 50%	hierarchical clustering	0.45	0.22	0.32
Selective forwarding 70%	ocsvm	0.87	0.04	0.80
Selective forwarding 70%	lofactor	0.50	0.15	0.36
Selective forwarding 70%	mahalanobis	0.62	0.15	0.49
Selective forwarding 70%	hierarchical clustering	0.55	0.22	0.42
Selective forwarding 90%	ocsvm	0.86	0.02	0.77
Selective forwarding 90%	lofactor	0.61	0.15	0.48
Selective forwarding 90%	mahalanobis	0.63	0.15	0.49
Selective forwarding 90%	hierarchical clustering	0.63	0.22	0.51

Bibliography

[1] Milind Naphade et al. "Smarter cities and their innovation challenges". In: *Computer* 44.6 (2011), pp. 32–39.

[2] Andrea Caragliu, Chiara Del Bo, and Peter Nijkamp. "Smart cities in Europe". In: *Journal of urban technology* 18.2 (2011), pp. 65–82.

[3] Constantin Gurdgiev, S Dirks, and M Keeling. "Smarter cities for smarter growth". In: *IBM Institute for Business Value* (2010).

[4] C Manville. *Mapping Smart Cities in the EU. European Parliament, Directorate General for Internal Policies, Policy Department–Economic and Scientific Policy.* Tech. rep. IP/A/ITRE/ST/2013-02, available http://www. europarl. europa. eu- /RegData/etudes/etudes/join/2014/507480/IPOLITRE_ET (2014) 507480_EN. df, 2014.

[5] The Royal Academy of Engineering. *Smart infrastructure: the future.* Tech. rep. The Royal Academy of Engineering, 2012, pp. 16–17.

[6] Malik Tubaishat et al. "Wireless sensor-based traffic light control". In: *Conf. Consumer Communications and Networking.* IEEE. 2008, pp. 702–706.

[7] Ivan Stoianov et al. "PIPENET: A wireless sensor network for pipeline monitoring". In: *Int. Symp. Information Processing in Sensor Networks.* IEEE. 2007, pp. 264–273.

[8] N. Perlroth. *Smart City Technology May Be Vulnerable to Hackers.* http://bits.blogs.nytimes.com/2015/04/21/smart-city-technology-may-be-vulnerable-to-hackers/. Accessed: 2016-02-08. 2015.

[9] Branden Ghena et al. "Green lights forever: analyzing the security of traffic infrastructure". In: *Workshop on Offensive Technologies.* USENIX. 2014.

[10] Jhoana Mutiangpili. *Government Sector Outsourcing*. Tech. rep. Tholons, 2010, p. 18.

[11] Felipe Gil-Castineira et al. "Experiences inside the ubiquitous Oulu smart city". In: *Computer* 44.6 (2011), pp. 48–55.

[12] Yong Woo Lee and Seungwoo Rho. "U-city portal for smart ubiquitous middleware". In: *Int. Conf. Advanced Communication Technology (ICACT)*. Vol. 1. IEEE. 2010, pp. 609–613.

[13] Min Chen. "Towards smart city: M2M communications with software agent intelligence". In: *Multimedia Tools and Applications* 67.1 (2013), pp. 167–178.

[14] Jayavardhana Gubbi et al. "Internet of Things (IoT): A vision, architectural elements, and future directions". In: *Future Generation Computer Systems* 29.7 (2013), pp. 1645–1660.

[15] Luis Sanchez et al. "SmartSantander: IoT experimentation over a smart city testbed". In: *Computer Networks* 61 (2014), pp. 217–238.

[16] Inc. Sensys Networks. *Sensys Networks VDS240 Wireless Vehicle Detection System. Design Guidelines for Intersection Applications*. Tech. rep. Sensys Networks, Inc., 2014.

[17] Briony J Oates. *Researching information systems and computing*. Sage, 2005.

[18] Doug Laney. "3D data management: Controlling data volume, velocity and variety". In: *META Group Research Note* 6 (2001), p. 70.

[19] Michael Batty. *Smart Cities and Big Data*. `http://www.spatialcomplexity.info/files/2013/07/BATTY-AESOP-ACSP-20131.pdf`. Accessed: 2016-08-09. 2012.

[20] Shintaro Yamamoto, Shinsuke Matsumoto, and Masahide Nakamura. "Using cloud technologies for large-scale house data in smart city". In: *IEEE 4th Int. Conf. on Cloud Computing Technology and Science (CloudCom)*. IEEE. 2012, pp. 141–148.

[21] Min Chen, Shiwen Mao, and Yunhao Liu. "Big data: a survey". In: *Mobile Networks and Applications* 19.2 (2014), pp. 171–209.

[22] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. "The Google file system". In: *ACM SIGOPS operating systems review*. Vol. 37. ACM. 2003, pp. 29–43.

[23] Konstantin Shvachko et al. “The Hadoop distributed file system”. In: *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*. IEEE. 2010, pp. 1–10.

[24] Jeffrey Dean and Sanjay Ghemawat. “MapReduce: simplified data processing on large clusters”. In: *Communications of the ACM* 51.1 (2008), pp. 107–113.

[25] Xiaochong Zhang. *The overall MapReduce word count process*. http://xiaochongzhang.me/blog/wp-content/uploads/2013/05/MapReduce_Work_Structure.png. Accessed: 2016-08-31. 2013.

[26] Shimin Chen and Steven W Schlosser. “Map-reduce meets wider varieties of applications”. In: *Intel Research Pittsburgh, Tech. Rep. IRP-TR-08* 5 (2008).

[27] Fadhilah Kurnia Putri et al. “Finding Frequent Route of Taxi Trip Events Based on MapReduce and MongoDB”. In: *KIPS Transactions on Software and Data Engineering* 4.9 (2015), pp. 347–356.

[28] Bin Cheng et al. “Building a Big Data Platform for Smart Cities: Experience and Lessons from Santander”. In: *2015 IEEE Int. Congress on Big Data*. IEEE. 2015, pp. 592–599.

[29] David Miller et al. *Security information and event management (SIEM) implementation*. McGraw Hill Professional, 2010.

[30] Stephen Sorkin. *Large-Scale, Unstructured Data Retrieval and Analysis Using Splunk*. Tech. rep. Accessed: 2016-08-09. Splunk Inc., 2011, p. 7. URL: https://www.splunk.com/web_assets/pdfs/secure/Splunk_and_MapReduce.pdf.

[31] Juan Manuel Lorenzo. *AlienVault Installation Guide*. Tech. rep. AlienVault LC, 2010, p. 52.

[32] Chris Meering and Paolo Balella. *Smart cities and the Internet of Things. Municipal transformation with the HPE Universal IoT Platform*. Tech. rep. Hewlett Packard Enterprise Development LP, 2016.

[33] Fabio Leccese, Marco Cagnetti, and Daniele Trinca. “A smart city application: A fully controlled street lighting isle based on Raspberry-Pi card, a ZigBee sensor network and WiMAX”. In: *Sensors* 14.12 (2014), pp. 24408–24424.

[34] Inc. Ruckus Wireless. *Public Access: City of San Jose*. Tech. rep. Ruckus Wireless, Inc., 2014.

[35] Mark Anderson. "WiMax for smart grids". In: *IEEE Spectrum* 47.7 (2010), pp. 14–14.

[36] Inc. LinkLabs. *A comprehensive look at Low Power, Wide Area Networks For Internet of Things Engineers and Decision Makers.* Tech. rep. LinkLabs, Inc., 2016.

[37] Libelium Comunicaciones Distribuidas S.L. *Plug & Sense! Smart parking.* Tech. rep. Libelium Comunicaciones Distribuidas S.L., 2016.

[38] IEEE Computer Society. *IEEE Standard for Local and metropolitan area networks - Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs).* Tech. rep. IEEE Computer Society, 2011, p. 294.

[39] Lamia Chaari and Lotfi Kamoun. "Wireless sensors networks MAC protocols analysis". In: *arXiv preprint arXiv:1004.4600* (2010).

[40] ZigBee Standards Organization. *Zigbee specification.* Tech. rep. ZigBee Standards Organization, 2012, p. 594.

[41] Gabriel Montenegro et al. *Transmission of IPv6 packets over IEEE 802.15. 4 networks.* Tech. rep. The IETF Trust, 2007.

[42] Vinay Kumar and Sudarshan Tiwari. "Routing in IPv6 over low-power wireless personal area networks (6LoWPAN): A survey". In: *Journal of Computer Networks and Communications* 2012 (2012).

[43] Charles Perkins, Elizabeth Belding-Royer, and Samir Das. *Ad hoc on-demand distance vector (AODV) routing.* Tech. rep. The IETF Trust, 2003.

[44] Zach Shelby, Klaus Hartke, and Carsten Bormann. *The constrained application protocol (CoAP).* Tech. rep. The IETF Trust, 2014.

[45] T Kavitha and D Sridharan. "Security vulnerabilities in wireless sensor networks: A survey". In: *Journal of information Assurance and Security* 5.1 (2010), pp. 31–44.

[46] Hero Modares, Rosli Salleh, and Amirhossein Moravejosharieh. "Overview of security issues in wireless sensor networks". In: *Int. Conf. Computational Intelligence, Modelling and Simulation.* IEEE. 2011, pp. 308–311.

[47] Javier Lopez and Jianying Zhou. "Overview of wireless sensor network security". In: *Wireless sensor network security. IOS Press, incorporated* (2008), pp. 1–21.

[48] Wooyoung Jung et al. "SSL-based lightweight security of IP-based wireless sensor networks". In: *Int. Conf. Advanced Information Networking and Applications Workshops*. IEEE. 2009, pp. 1112–1117.

[49] David Gascón. *IoT Security Infographic – Privacy, Authenticity, Confidentiality and Integrity of the Sensor Data. "The Invisible Asset"*. http://libelium.com/downloads/security_infographic.pdf. Accessed: 2016-08-10. 2015.

[50] Aristides Mpitziopoulos et al. "Defending wireless sensor networks from jamming attacks". In: *Int. Symposium Personal, Indoor and Mobile Radio Communications*. IEEE. 2007, pp. 1–5.

[51] Ilkyu Kim et al. "A Distributed Signature Detection Method for Detecting Intrusions in Sensor Systems". In: *Sensors* 13.4 (2013), pp. 3998–4016.

[52] Varun Chandola, Arindam Banerjee, and Vipin Kumar. "Anomaly Detection: A Survey". In: *ACM Comput. Surv.* 41.3 (2009), 15:1–15:58.

[53] John W Tukey. *Exploratory data analysis*. Reading, Mass., 1977.

[54] Douglas C Montgomery, Cheryl L Jennings, and Murat Kulahci. *Introduction to time series analysis and forecasting*. John Wiley & Sons, 2015.

[55] Corinna Cortes and Vladimir Vapnik. "Support-vector networks". In: *Machine learning* 20.3 (1995), pp. 273–297.

[56] Edwin M Knorr and Raymond T Ng. "Finding intensional knowledge of distance-based outliers". In: *VLDB*. Vol. 99. 1999, pp. 211–222.

[57] Markus M Breunig et al. "LOF: identifying density-based local outliers". In: *ACM sigmod record*. Vol. 29. ACM. 2000, pp. 93–104.

[58] Aleksandar Lazarevic et al. "A Comparative Study of Anomaly Detection Schemes in Network Intrusion Detection." In: *Int. Conf. Data Mining*. SIAM. 2003, pp. 25–36.

[59] Miao Xie et al. "Anomaly detection in wireless sensor networks: A survey". In: *Journal of Network and Computer Applications* 34.4 (2011), pp. 1302–1325.

[60] Yang Zhang et al. "Statistics-based outlier detection for wireless sensor networks". In: *Int. Journal of Geographical Information Science* 26.8 (2012), pp. 1373–1392.

[61] Jing Su et al. "Anomaly Detection of Single Sensors Using OCSVM_KNN". In: *Big Data Computing and Communications*. Springer, 2015, pp. 217–230.

[62] F. Liu, X. Cheng, and D. Chen. “Insider Attacker Detection in Wireless Sensor Networks”. In: *Int. Conf. on Computer Communications*. IEEE. 2007, pp. 1937–1945.

[63] Nauman Shahid, Ijaz Haider Naqvi, and Saad Bin Qaisar. “Characteristics and classification of outlier detection techniques for wireless sensor networks in harsh environments: a survey”. In: *Artificial Intelligence Review* 43.2 (2015), pp. 193–228.

[64] Pu Cheng and Minghua Zhu. “Lightweight anomaly detection for wireless sensor networks”. In: *Int. Journal of Distributed Sensor Networks* 2015 (2015), p. 3.

[65] Qin Yu, Lyu Jibin, and Lirui Jiang. “An Improved ARIMA-Based Traffic Anomaly Detection Algorithm for Wireless Sensor Networks”. In: *Int. Journal of Distributed Sensor Networks* 2016 (2016).

[66] Bordeaux Digital City. *EUROCITIES Guidance webinar Standards WG*. Tech. rep. Bordeaux Digital City, 2016.

[67] International Organization for Standardization. *Sustainable development of communities – Indicators for city services and quality of life*. ISO 37120:2014. Geneva, Switzerland: International Organization for Standardization, 2014.

[68] International Telecommunication Union (ITU). *ITU-T Y.4400 series – Smart Sustainable Cities - Setting the framework for an ICT architecture*. Tech. rep. Geneva, Switzerland: International Telecommunication Union (ITU), 2016.

[69] International Telecommunication Union (ITU). *ITU-T L.1600 - Key performance indicators definitions for smart sustainable cities*. Tech. rep. Geneva, Switzerland: International Telecommunication Union (ITU), 2015.

[70] British Standards Institution. *Guide to establishing a model for data interoperability*. Guide PD 182:2014. London, United Kingdom: British Standards Institution, 2014.

[71] British Standards Institution. *Guide to the role of the planning and development process*. Guide PD 8101:2014. London, United Kingdom: British Standards Institution, 2014.

[72] AENOR. *UNE 178102-1:2015. Smart cities. Infrastructures. Telecommunication systems. Part 1: Multiservice city networks*. Tech. rep. AENOR, 2015.

[73] AENOR. *UNE 178104:2015. Smart cities. Infrastructures. Comprehensive systems for a Smart City management.* Tech. rep. AENOR, 2015.

[74] AENOR. *UNE 178107-4:2015 IN. Guidelines on smart cities infrastructures. Access and transport networks. Part 4: Wireless Sensor Networks, WSN.* Tech. rep. AENOR, 2015.

[75] AENOR. *UNE 178301:2015. Smart Cities. Open Data.* Tech. rep. AENOR, 2015.

[76] Sergio Marti et al. "Mitigating routing misbehavior in mobile ad hoc networks". In: *Int. Conf. Mobile Computing and Networking.* ACM. 2000, pp. 255–265.

[77] Sumit Gupta, Rong Zheng, and Albert MK Cheng. "ANDES: an anomaly detection system for wireless sensor networks". In: *Int. Conf. Mobile Adhoc and Sensor Systems.* IEEE. 2007, pp. 1–9.

[78] Nithya Ramanathan et al. "Sympathy for the sensor network debugger". In: *Int. Conf. Embedded Networked Sensor Systems.* ACM. 2005, pp. 255–267.

[79] Pedro Domingos. "A few useful things to know about machine learning". In: *Communications of the ACM* 55.10 (2012), pp. 78–87.

[80] Sven Zacharias et al. "Identifying sources of interference in RSSI traces of a single IEEE 802.15. 4 channel". In: *Int. Conf. on Wireless and Mobile Communications, Venice, Italy.* 2012.

[81] Georgy Shevlyakov et al. "Robust versions of the Tukey boxplot with their application to detection of outliers". In: *2013 IEEE Int. Conf. on Acoustics, Speech and Signal Processing.* IEEE. 2013, pp. 6506–6510.

[82] Chengwei Wang et al. "Statistical techniques for online anomaly detection in data centers". In: *12th IFIP/IEEE Int. Symp. on Integrated Network Management (IM 2011) and Workshops.* IEEE. 2011, pp. 385–392.

[83] Osman Salem et al. "Online anomaly detection in wireless body area networks for reliable healthcare monitoring". In: *IEEE journal of biomedical and health informatics* 18.5 (2014), pp. 1541–1551.

[84] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[85] Chesner Désir et al. "One class random forests". In: *Pattern Recognition* 46.12 (2013), pp. 3490–3506.

[86] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. "A fast learning algorithm for deep belief nets". In: *Neural computation* 18.7 (2006), pp. 1527–1554.

[87] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc., 2012, pp. 1097–1105.

[88] Richard Socher et al. "Parsing natural scenes and natural language with recursive neural networks". In: *Int. conf. on machine learning (ICML-11)*. 2011, pp. 129–136.

[89] Sarah M Erfani et al. "High-dimensional and large-scale anomaly detection using a linear one-class SVM with deep learning". In: *Pattern Recognition* (2016).

[90] Victoria J Hodge and Jim Austin. "A survey of outlier detection methodologies". In: *Artificial Intelligence Review* 22.2 (2004), pp. 85–126.

[91] Lior Rokach and Oded Maimon. "Data mining and knowledge discovery handbook". In: Springer, 2005. Chap. Clustering methods, pp. 321–352.

[92] Fionn Murtagh and Pierre Legendre. "Ward's Hierarchical Agglomerative Clustering Method: Which Algorithms Implement Ward's Criterion?" In: *Journal of Classification* 31.3 (2014), pp. 274–295.

[93] L. Torgo. *Data Mining with R, learning with case studies*. Chapman and Hall/CRC, 2010. URL: http://www.dcc.fc.up.pt/~ltorgo/DataMiningWithR.

[94] Alice Este, Francesco Gringoli, and Luca Salgarelli. "Support vector machines for TCP traffic classification". In: *Computer Networks* 53.14 (2009), pp. 2476–2490.

[95] Sophia Kaplantzis et al. "Detecting selective forwarding attacks in wireless sensor networks using support vector machines". In: *Int. Conf. Intelligent Sensors, Sensor Networks and Information*. IEEE. 2007, pp. 335–340.

[96] Ling Zhuang and Honghua Dai. "Parameter optimization of kernel-based one-class classifier on imbalance learning". In: *Journal of Computers* 1.7 (2006), pp. 32–40.

[97] Smriti Joshi, Anant Kr Jaiswal, and Pushpendra Kr Tyagi. "A Novel Analysis of T Mac and S Mac Protocol for Wireless Sensor Networks Using Castalia". In: *Int. Journal of Soft Computing and Engineering* 2.6 (2013), pp. 128–131.

[98] Yulia Ponomarchuk and Dae-Wha Seo. “A Lightweight and Effective Jamming Detection in Electronic Shelf Label Systems”. In: *Int. Conf. Ubiquitous Information Technologies & Applications*. IEEE. 2009, pp. 1–6.

[99] András Varga. “The omnet++ discrete event simulation system”. In: *In ESM’01*. 2001.

[100] Dimosthenis Pediaditakis, Yuri Tselishchev, and Athanassios Boulis. “Performance and scalability evaluation of the Castalia wireless sensor network simulator”. In: *Int. Conf. Simulation Tools and Techniques*. ICST. 2010, p. 53.

[101] Fredrik Osterlind et al. “Cross-level sensor network simulation with cooja”. In: *Conf. Local Computer Networks*. IEEE. 2006, pp. 641–648.

[102] Lee Breslau et al. “Advances in Network Simulation”. In: *IEEE Computer* 33.5 (May 2000), pp. 59–67.

[103] Kamal Mehdi et al. “CupCarbon: a multi-agent and discrete event wireless sensor network design and simulation tool”. In: *Int. Conf. Simulation Tools and Techniques*. ICST. 2014, pp. 126–131.

[104] Ivan Minakov et al. “A comparative study of recent wireless sensor network simulators”. In: *ACM Transactions on Sensor Networks (TOSN)* 12.3 (2016), p. 20.

[105] Victor Garcia-Font, Carles Garrigues, and Helena Rifà-Pous. *A comparative study on anomaly detection techniques for smart city wireless sensor networks (Source code)*. http://einfmark.uoc.edu/technology/get/id/2. Accessed: 2016-06-09. 2016.

[106] Texas Instruments. *CC2420. 2.4 GHz IEEE 802.15.4 / ZigBee-Ready RF Transceiver*. Tech. rep. Accessed: 2016-06-09. Texas Instruments, 2014, p. 85. URL: http://www.ti.com/lit/ds/symlink/cc2420.pdf.

[107] Tijs Van Dam and Koen Langendoen. “An adaptive energy-efficient MAC protocol for wireless sensor networks”. In: *Int. Conf. Embedded networked sensor systems*. ACM. 2003, pp. 171–180.

[108] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2015. URL: https://www.R-project.org/.

[109] David Meyer et al. *e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*. R package version 1.6-6. 2015. URL: https://CRAN.R-project.org/package=e1071.

[110] Chih-Chung Chang and Chuan-bi Lin. "Training v-support vector classifiers: theory and algorithms". In: *Neural computation* 13.9 (2001), pp. 2119–2147.

[111] Payam Refaeilzadeh, Lei Tang, and Huan Liu. "Encyclopedia of Database Systems". In: Springer US, 2009. Chap. Cross-Validation, pp. 532–538.

[112] Ranjan K Som. "Practical sampling techniques". In: CRC press, 1995. Chap. Simple random sampling, pp. 38–40.

[113] Vladimir Cherkassky and Yunqian Ma. "Practical selection of SVM parameters and noise estimation for SVM regression". In: *Neural networks* 17.1 (2004), pp. 113–126.

[114] Hani Omar, Van Hai Hoang, and Duen-Ren Liu. "A Hybrid Neural Network Model for Sales Forecasting Based on ARIMA and Search Popularity of Article Titles". In: *Computational Intelligence and Neuroscience* 2016 (2016).

[115] Scikit-learn developers. *Scikit-learn. Support Vector Machines*. http://scikit-learn.org/stable/modules/svm.html. Accessed: 2016-08-14. 2014.

[116] Marc Claesen et al. "Fast prediction with SVM models containing RBF kernels". In: *arXiv preprint arXiv:1403.0736* (2014).

[117] Max Kuhn. Contributions from Jed Wing et al. *caret: Classification and Regression Training*. R package version 6.0-47. 2015. URL: https://CRAN.R-project.org/package=caret.

[118] Christian Hennig. *fpc: Flexible procedures for clustering*. R package version 2.1-9. 2014. URL: http://CRAN.R-project.org/package=fpc.

[119] Omprakash Gnawali et al. "Collection tree protocol". In: *Proceedings of the 7th ACM Conf. on Embedded Networked Sensor Systems*. ACM. 2009, pp. 1–14.

[120] Martin Ester et al. "A density-based algorithm for discovering clusters in large spatial databases with noise." In: *Kdd*. Vol. 96. 34. 1996, pp. 226–231.

List of publications

Conference proceedings

[121] Victor Garcia Font, Carles Garrigues, and Helena Rifà Pous. “Seguridad en smart cities e infraestructuras críticas”. In: *Actas de la XIII Reunión Española sobre Criptología y Seguridad de la Información (RECSI 2014)*. Universidad de Alicante. 2014, pp. 221–226.

[122] Victor Garcia-Font, Carles Garrigues, and Helena Rifà-Pous. “An architecture for the analysis and detection of anomalies in smart city WSNs”. In: *Proceedings of the First IEEE International Smart Cities Conference (ISC2)*. IEEE. 2015, pp. 207–2012.

[123] Victor Garcia-Font, Carles Garrigues, and Helena Rifà-Pous. “Anomaly detection in smart city parking data: A case study”. In: *Actas de la XIV Reunión Española sobre Criptología y Seguridad de la Información (RECSI 2016)*. Universitat de les Illes Balears. 2016, pp. 81–85.

Journal articles

[124] Victor Garcia-Font, Carles Garrigues, and Helena Rifà-Pous. “A Comparative Study of Anomaly Detection Techniques for Smart City Wireless Sensor Networks”. In: *Sensors* 16.6 (2016). Impact Factor=2.033(2015), 1st quartile, p. 868.

[125] Victor Garcia-Font, Carles Garrigues, and Helena Rifà-Pous. *Attack classification schema for smart city WSNs (under review)*. 2016.