



Internet of Things demonstrator based on the OpenMote platform

Arturo Medina

Máster universitario de Ingeniería de Telecomunicación
Telemática

Jose Lopez Vicario

Xavi Vilajosana Guillen

18/04/2018

GNU Free Documentation License (GNU FDL)

Copyright © 2018 Arturo Medina Merino

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.

A copy of the license is included in the section entitled "GNU Free Documentation License".

FICHA DEL TRABAJO FINAL

| | |
|------------------------------------|--|
| Título del trabajo: | Internet of Things demonstrator based on the OpenMote platform |
| Nombre del autor: | <i>Arturo Medina Merino</i> |
| Nombre del consultor/a: | José Lopez Vicario |
| Nombre del PRA: | |
| Fecha de entrega (mm/aaaa): | 23/05/2018 |
| Titulación:: | Máster universitario de Ingeniería de Telecomunicación |
| Área del Trabajo Final: | <i>Telemática</i> |
| Idioma del trabajo: | <i>Inglés</i> |
| Palabras clave | <i>iot, cloud computing, edge computing</i> |

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.*

Abstract (in English, 250 words or less):

In recent years, the number of systems which leverage Internet of Things technologies has experienced an enormous growth. From industrial machine-to-machine applications to personal data metrics, the connectivity of all kinds of devices has become an incredibly interesting tool for improving the performance of existing systems, and, in many occasions, it has led to the creating of whole new business areas.

In this context, open source and open hardware frameworks and devices have become a key driving force of innovation. Thanks to their use, it has been possible to accelerate the research and development of new technologies and protocols, rapidly closing the existing gap between the academia and commercial applications.

The purpose of this project is to develop an Internet of Things (IoT) demonstrator which makes use of currently relevant open source and hardware technologies in the field, from the physical layer to the application layer, as a way of demonstrating such potential. A sensor network, including data collection, transmission and centralization will be studied. In addition, two current trends in the IoT research area will be analyzed and compared, namely the different approaches of edge computing and cloud computing.

In this manner, a small scale but close to real-world scenario will be presented, acting as a valid example of the potential impact of IoT systems as an enabler for increasing the intelligence and efficiency of applications across different industries.

Acknowledgments

Thanks to all my friends and family for their unconditional support,
to Ricardo, Virgilio and David for accompanying me in this journey,
to María and Pablo, for their charm and help,
and to José Lopez for his always helpful and sincere feedback.

Index

| | |
|---|-----|
| Acknowledgments | iii |
| 1 Introduction | 1 |
| 1.1 Objectives | 3 |
| 1.2 Planning..... | 4 |
| 1.3 Gantt Chart | 5 |
| 1.4 Related work..... | 1 |
| 2 IoT components, technologies and protocols..... | 2 |
| 2.1 Open Hardware and Open Source solutions for the IoT | 2 |
| 2.2 Operating systems for embedded devices and wireless sensors | 2 |
| 2.2.1 TinyOS | 3 |
| 2.2.2 RIOT OS | 3 |
| 2.2.3 Contiki OS | 4 |
| 2.2.4 Comparison of WSN operating systems..... | 5 |
| 2.3 IoT Protocols..... | 6 |
| 2.3.1 Brief overview of the OSI model | 6 |
| 2.3.2 Infrastructure protocols..... | 7 |
| 2.3.2.1 RPL..... | 7 |
| 2.3.2.2 IEEE Std 802.15.4: Low-Rate Wireless Personal Area Networks | 8 |
| 2.3.2.2.1 Network nodes..... | 8 |
| 2.3.2.2.2 Topologies..... | 8 |
| 2.3.2.2.3 Physical layer (PHY)..... | 9 |
| 2.3.2.2.4 MAC sublayer | 9 |
| 2.3.3 Application Protocols..... | 9 |
| 2.3.3.1 MQTT..... | 9 |
| 2.3.3.2 CoaP protocol | 10 |
| 2.4 Strategies for data processing and storage | 11 |
| 2.4.1 Cloud computing | 12 |
| 2.4.2 Edge computing | 14 |
| 2.4.3 Cloud computing vs edge computing | 17 |
| 2.5 Hardware platforms for IoT | 18 |
| 2.5.1 Raspberry Pi 3 B+ | 19 |
| 2.6 IoT Cloud Platforms | 21 |
| 2.6.1 thethings.iO | 21 |
| 2.6.2 thethings.iO overview | 22 |
| 2.7 Data storage: the database | 25 |
| 2.8 Ethics of the IoT..... | 26 |
| 3 Proposed design | 27 |
| 3.1 System overview..... | 27 |
| 3.2 System processes | 28 |
| 3.2.1 Cloud computing model..... | 29 |
| 3.2.1.1 Retrieve and send to the cloud sensor data from motes..... | 29 |
| 3.2.1.2 Analyze stored information to compute relevant statistical metrics | 30 |
| 3.2.1.3 Data representation | 31 |

| | | |
|---------|---|----|
| 3.2.1.4 | Intruder detection | 31 |
| 3.2.2 | Edge computing model..... | 32 |
| 3.2.2.1 | Retrieve and store sensor data from motes | 32 |
| 3.2.2.2 | Analyze stored information to compute relevant statistical metrics | 34 |
| 3.2.2.3 | Actuate over the air conditioning system based on the temperature | 36 |
| 3.2.2.4 | Data cleanup..... | 37 |
| 3.2.3 | Comparison of designed features..... | 37 |
| 3.3 | Deployment | 37 |
| 3.3.1 | Summary of collected and generated data | 39 |
| 3.4 | Database model..... | 39 |
| 3.4.1 | Tables | 40 |
| 3.4.1.1 | MOTES | 40 |
| 3.4.1.2 | SOURCE_TYPES..... | 40 |
| 3.4.1.3 | READINGS | 40 |
| 3.4.1.4 | TEMPERATURE..... | 40 |
| 3.4.1.5 | HUMIDITY | 41 |
| 3.4.1.6 | PRESENCE | 41 |
| 3.4.2 | Relationship between tables | 42 |
| 4 | Implementation | 43 |
| 4.1 | Simulation of motes | 43 |
| 4.1.1 | Web interface | 44 |
| 4.1.1.1 | Motes | 44 |
| 4.1.1.2 | Event bus..... | 45 |
| 4.1.1.3 | Topology | 45 |
| 4.1.1.4 | Routing | 45 |
| 4.1.1.5 | Connectivity | 46 |
| 4.1.1.6 | Documentation..... | 46 |
| 4.2 | Developing OpenWSN applications..... | 46 |
| 4.2.1 | C Implementation | 47 |
| 4.2.1.1 | Initialization | 48 |
| 4.2.1.2 | Reception callback..... | 48 |
| 4.2.2 | Compilation | 49 |
| 4.2.3 | Implementation of computation models..... | 49 |
| 4.2.4 | Communication using CoAP | 50 |
| 4.2.5 | Interfacing with PostgreSQL from Python | 50 |
| 4.2.6 | Interfacing with thethings.io | 52 |
| 4.2.7 | Scheduling of periodic operations | 52 |
| 4.3 | Cloud Computing implementation..... | 53 |
| 4.3.1 | Server-side computation | 54 |
| 4.4 | Edge Computing implementation..... | 56 |
| 5 | Comparison of models | 61 |
| 5.1.1 | Latency..... | 61 |
| 5.1.1.1 | Edge computing | 61 |
| 5.1.1.2 | Cloud computing..... | 61 |
| 5.1.2 | Network traffic | 62 |
| 5.1.2.1 | Cloud computing | 62 |
| 5.1.2.2 | Edge computing | 63 |
| 5.1.3 | Power consumption..... | 63 |

| | | |
|---------|------------------------------------|----|
| 5.1.3.1 | Cloud computing | 63 |
| 5.1.3.2 | Edge computing | 63 |
| 5.1.4 | Cost..... | 64 |
| 5.1.4.1 | Cloud computing | 64 |
| 5.1.4.2 | Edge computing | 64 |
| 5.1.5 | Security | 65 |
| 5.1.5.1 | Edge computing | 65 |
| 5.1.5.2 | Cloud computing | 65 |
| 5.1.6 | Summary of comparison | 66 |
| 6 | Conclusions | 67 |
| 7 | Future work | 68 |
| 8 | Glossary..... | 69 |
| 9 | References..... | 70 |
| 10 | Appendix..... | 74 |
| 10.1 | psycopg2 setup | 74 |
| 10.2 | Raspberry Pi 3 Model B+ setup..... | 74 |
| 10.3 | OpenSim setup..... | 74 |
| 10.4 | PostgreSQL setup | 74 |

Figures

| | |
|--|----|
| Figure 1: Project Gantt chart, describing the planned work packages..... | 1 |
| Figure 2: Partition into ROM and RAM memory in Contiki OS. Adapted from [20]. | 5 |
| Figure 3: two example message exchanges. In both cases, the message type, CON, indicates that a confirmation is required. In the first case, the request ends successfully, while in the second it does not. However, in both occasions, an ACK message is sent. Adapted from [25]...... | 11 |
| Figure 4. Overview of Cloud Computing Architecture. Adapted from [26]. | 12 |
| Figure 5: Edge computing. Edge computing is performed between the source (devices) and the cloud platform. Some of the load, in both upstream and downstream directions, is handled by the network edge and not by the cloud platform, thus effectively reducing response time and making a more efficient use of network capacity. Adapted from [29]. | 14 |
| Figure 6: configuration of an IoT product in thethings.iO. The chosen format .. | 23 |
| Figure 7: Definition of a “thing”. The Thing Token identifies the device which is sending the information. | 23 |
| Figure 8: Example data received at thethings.iO. As it can be seen, the data is represented with its associated timestamp, so that the evolution of the measurement over time can be observed. In addition, several sources can be represented at the same time (temperature and humidity, in this case). Consequently, it becomes easy to observe correlations between them. | 24 |
| Figure 9: thethings.iO Dashboard, showing temperature and OpenMote location | 24 |
| Figure 10: system overview. Sensor data is collected by OpenMote devices distributed along a property. Such data is sent to a central point, the Raspberry Pi, which acts as a gateway, processing the data and sending it to the thethings.iO platform. | 27 |
| Figure 11: flow diagram which describes how the process of retrieving data from sensors and sending it to the cloud is performed | 30 |
| Figure 12: flow diagram which describes how the sub-process of reading a value from a mote is performed. | 30 |
| Figure 13: flow diagram showing the process of detecting an intruder..... | 32 |
| Figure 14: flow diagram which describes how information is read from the motes in the Edge Computing model. | 33 |
| Figure 15: flow diagram which describes the sub-process of reading a resource value from a mote..... | 34 |
| Figure 16: flow diagram which describes how statistical analysis is performed on the stored data. | 35 |
| Figure 17: flow diagram which describes the sub-process of calculating different statistical metrics on the stored data. | 36 |
| Figure 18: flow diagram which describes how the air conditioning system is actuated based on the temperature..... | 36 |
| Figure 19: proposed home deployment for the application..... | 38 |
| Figure 20: Relationship between tables in the database model | 42 |
| Figure 21: High level overview of the implementation | 43 |
| Figure 22: Mote overview | 44 |

| | |
|---|----|
| Figure 23: Location and topology of motes | 45 |
| Figure 24: Overview of OpenSim and its components. Adapted from OpenWSN documentation..... | 46 |
| Figure 25: CoAP GET request captured with Wireshark | 49 |
| Figure 26: Interaction between Python scripts and the PostgreSQL database | 50 |
| Figure 27: Interaction between Python scripts and the PostgreSQL database | 52 |
| Figure 28: summarized syntax of crontabs..... | 52 |
| Figure 29: overview of the different parts which compose the cloud computing model. Triggers execute code upon data reception and perform some action as a response (in this case, sending an alert email). Jobs are executed periodically an interact with the internally stored resources to calculate KPIs (key performance indicators)..... | 53 |
| Figure 30: representation of the maximum temperature calculated using a job | 55 |
| Figure 31: example of received email..... | 56 |
| Figure 32: overview of the different parts which compose the edge computing model. edge_readings.py fetches data from OpenSim and stores it internally into the Raspberry Pi. On the other hand, the analyze_readings.py scripts is executed every hour to extract relevant information for data collected during the previous hour..... | 56 |
| Figure 33: creation of a dashboard for representing a resource calculated on the edge | 59 |
| Figure 34: panel displaying number of people at home | 60 |
| Figure 35: CoAP messaging for reading resource values. | 61 |
| Figure 36: severe communication delay in the communication with thethings.iO | 62 |
| Figure 37: encrypted conversation between Raspberry Pi and Thethings.iO ... | 62 |
| Figure 38: bytes sent from Raspberry Pi to thethings.iO in the edge model..... | 63 |
| Figure 39: 802.15.4 frame containing presence reading | 65 |

1 Introduction

Nowadays, it is undeniable that the Internet of Things, abbreviated as IoT, has become one of the most exciting and prominent areas in the field Information Technology and its related industries. Far from being a mere buzzword, IoT-enabled devices are already a key component of many real-world systems, both in industrial and home environments.

One of the reasons why this has happened is that such devices and systems can be applied to a significant number of situations. From factories to hospitals, the power offered by IoT-enabled networks offers great opportunities for improving, optimizing and even redefining many different types of tasks.

It is worth stopping for one second to present a brief definition of what Internet of Things is, since some confusion might appear. However, as with any industry trend, the term “IoT” is used broadly, and what is considered as IoT by some actors could not be deemed as such by others. For this purpose, [1] will be cited verbatim:

IoT can be defined as the ever-growing network of Things (entities) that feature Internet connectivity and the communication that occurs between them and other Internet-enabled devices and systems

IoT networks are often characterized by its closed range of action and almost complete lack of infrastructure. They are usually composed of small devices which communicate with each other sending small amounts of information, which is usually collected in a central node or gateway. Due to these features, such devices must usually be small, inexpensive and power-efficient, since this allows for its use in applications such as health monitoring, home automation and sensor networks.

Under the umbrella of this definition, many things can be considered IoT devices. Typical examples include a fridge connected to the Internet. In this case, the fridge might be able to monitor what amount of a certain product is left. Moreover, if it detects that the quantity of such product, for example milk, falls below a certain level, it has the possibility of autonomously ordering more milk bottles from a chosen provider. In this example, the fact that the fridge has network connectivity has allowed for a concrete improvement of our everyday life, namely that we no longer need to worry about not having milk for our morning coffee.

By means of this example, many of the elements most commonly found in IoT applications can be identified. It can be appreciated that a *sensor* has been used to capture a concrete piece of information (quantity of milk left), which has then been *collected* (in this case by the fridge itself, although other possibilities exist, such as a central home hub for collecting information from different appliances). Once the information has been gathered, it has been *processed* (compared to a predefined level) and the conclusions obtained has been *communicated via the*

Internet. Not all applications will present all these elements, but this can help to shed some light into what can be considered an IoT-enabled system.

In summary, IoT transforms ordinary objects into smart and cognitive systems, capable of monitoring their environment and actively actuating to transform it. This can be applied to almost any device, and therefore it can be applied to almost any application. For that reason, IoT embraces many different vertical markets, including, but not limited to, transport, manufacturing industries, healthcare, agriculture, smart cities and smart homes.

With so many options to choose, it is necessary to select the area of application for this project. Nonetheless, this dissertation will focus on only a small part of the IoT world, attempting to develop a **smart home** system. There are several reasons for this:

- Smart homes, a rarity some years ago, are become increasingly common. The irruption of Wi-Fi controlled light bulbs, continuous remote surveillance and monitoring, and even assistants with artificial intelligence capabilities, are creating a thriving environment for innovation, pushing the evolution of mere homes to smart homes [2]
- Smart homes is a trending research area, with many different disciplines studying how to maximize the comfort and minimize resource consumption [3] [4].
- A smart home application can be easily simulated with scarce resources. Unlike other fields such as healthcare, which deal with very complex signals and systems, a smart home application can monitor and control simple metrics such as room temperature.
- The application of IoT to home environments goes beyond monitoring and controlling comfort metrics. Thanks to the fact that precise information can be collected in a periodic and consistent manner, advanced studies can be performed on the habits of citizens on their homes. This can lead to the creation of advanced models which can help to better understand how does society make use of their homes, allowing the optimize energy consumption and providing insights into how to better design houses. However, this is not only limited to homes, being also applicable in public building such as hospitals, schools, train stations, museums, government buildings, etc. In these areas, IoT can also be used to model and study the relationship between different indicators, such as occupancy, temperature and energy consumption, also leading to better designs in the future [5].

Consequently, the elements that compose the project have been selected with the idea of being representative of the technological areas to which they belong, resulting in the following choices for the different parts of the system:

- **Hardware:**
 - OpenMote: IoT-oriented open-hardware prototyping ecosystem

- Raspberry Pi: low-cost single-board computer suitable for cheap and open-source oriented projects
- **Software:**
 - Contiki: IoT-oriented operating system for low-power microcontrollers
 - thethings.iO: cloud platform for storage, analytics and tools for managing IoT devices
 - OpenWSN: operating system which provides simulation capabilities through the OpenSim component.
- **Telecommunication standards:**
 - 802.15.4e: communication standard especially suitable for close range communications (such as in sensor networks) using low-powered devices
 - CoAP: web transfer protocol for constrained devices.

The combination of the aforementioned elements results in a rich ecosystem of open source and open hardware which nonetheless provides a valid approach to IoT-related technologies.

1.1 Objectives

The main objective of this project can be described as:

Develop an Internet of Things demonstrator which makes use of currently relevant technologies, resembling real-world applications as much as possible, to demonstrate the applicability and relevance of open-source and open-hardware platforms as key enablers of real-world IoT deployments.

In addition, several other objectives and motivations have been defined:

- Perform a comparison between different approaches to the analysis and computation of results obtained from IoT-generated data, namely the **edge computing** and **cloud computing** paradigms.
- Investigate the status of open-hardware platforms in IoT, with on a focus on aspects such as availability, ease of use and viability as solutions for real-world applications.
- Investigate the current ecosystem of open-source utilities, operating systems and web platforms in IoT. In particular, the *Contiki* operating system will be thoroughly used and therefore studied throughout the duration of the project.
- Gain an insight into the IEEE 802.15.4e standard, analyzing its significance in IoT applications and the implementation of such standard provided by hardware platforms.
- Make use of the *OpenMote* platform as a key component of a IoT demonstrator, including its configuration, programming and interconnection with other elements.
- Adapt a *Raspberry Pi* SBC for its use as in IoT applications.

- Gain an insight into available commercial-off-the-shelf websites, such as *thethings.iO* which can act as IoT getaways and provide both data aggregation and data display capabilities, analyzing its impact into accelerating IoT developments.
- Analyze the integration of different pieces of hardware, potentially provided by different vendors, to create a unique IoT-oriented product.
- Enhance the authors ability to plan an undertake Information Technology projects.

1.2 Planning

In this section, a description of the work packages which the project will be composed of is presented, serving as companion explanation to the Gantt chart included in the following section. A hierarchy of tasks has been defined, divided in three level, namely *work areas* (e.g. Project Implementation), *work packages* (e.g. simulator set-up) and *low-level activities*.

- Project Planning:
 - Initial Planning: develop understanding of project description and purpose and create a draft of project planning.
- Project Implementation:
 - Research state-of-the-art: analyze current trends in IoT and possible implementations at hardware and software level.
 - Design of demonstrator: determine components of the demonstrator, interfaces between them, data to be collected, how data will be stored, how devices will communicate and how the user will access the stored information.
 - Create software repositories: work on the code will be performed on software repositories (Github, Bitbucket, etc.) so that they are under version control and are accessible from multiple places.
 - Simulator set-up: install and configure the emulator.
 - First tests with simulator: familiarization with the simulator by implementing different basic scenarios.
 - Implementation of demonstrator in simulator: implementation of the designed demonstrator in the simulation.
 - Acquisition of required physical material: purchase/obtain the required hardware for implementing the demonstrator. It is necessary to consider possible delays in the delivery of such hardware.
 - Hardware set-up: upon reception of the hardware, it will be necessary to configure it, including OS installation (e.g. installing an OS for Raspberry Pi on a SD card).
 - Implementation of demonstrator using real hardware: final implementation of the demonstrator using the received components.
- Project Documentation: the documentation associated to each submission will be prepared in parallel to everyday work performed on the project. However, it has been considered necessary to allocate specific work

packages to reflect the fact that documenting and formatting all the generated information will require a non-negligible amount of time, being necessary to assign resources to it.

- Submissions: milestones for documentation submission, as well as objectives for percentage of field work finalized (60% for PEC 2 and 100% for PEC3).

1.3 Gantt Chart

The project timeline has been represented by means of a Gantt chart, as shown in Figure 1. For this purpose, the Microsoft Project 2013 software was used. The Gantt chart includes the work packages introduced in the previous section, assigning them a duration and defining the interdependencies existing between them.

Estimated duration is provided for work packages, which make up the estimated duration of work areas.

It is considered that the definition of low-level activities can be performed as the project progresses, and a deeper understanding of the project is developed.

| | | | | |
|---|---------|--------------|--------------|-------|
| ▸ Project Planning | 6 días | mié 28/02/18 | mié 07/03/18 | |
| ▸ Project Implementation | 56 días | jue 08/03/18 | jue 24/05/18 | 1 |
| ▸ Research state-of-the-art | 5 días | jue 08/03/18 | mié 14/03/18 | 2 |
| ▸ Design of demonstrator | 7 días | jue 15/03/18 | vie 23/03/18 | 4 |
| ▸ Create software repositories | 1 día | jue 15/03/18 | jue 15/03/18 | |
| ▸ Simulator set-up | 3 días | lun 26/03/18 | mié 28/03/18 | 9 |
| ▸ First tests with simulator | 3 días | jue 29/03/18 | lun 02/04/18 | 17;15 |
| ▸ Implementation of demonstrator in simulator | 6 días | mar 03/04/18 | mar 10/04/18 | 20 |
| Acquisition of required physical material | 15 días | mié 21/03/18 | mar 10/04/18 | |
| ▸ Hardware set-up | 7 días | mié 11/04/18 | jue 19/04/18 | 23;27 |
| Implementation of demonstrator using real hardware | 25 días | vie 20/04/18 | jue 24/05/18 | 28 |
| ▸ Project Documentation | 53 días | mié 04/04/18 | sáb 16/06/18 | |
| Project memory (PEC 2) | 10 días | mié 04/04/18 | mar 17/04/18 | |
| Project memory (PEC 3, candidate for final version) | 9 días | vie 11/05/18 | mié 23/05/18 | |
| Project memory (Final version) | 12 días | vie 25/05/18 | sáb 09/06/18 | |
| Project presentation | 6 días | lun 11/06/18 | sáb 16/06/18 | |
| ▸ Submissions | 73 días | mié 07/03/18 | dom 17/06/18 | |
| Submission of PEC 1 | 0 días | mié 07/03/18 | mié 07/03/18 | |
| Submission of PEC 2 | 0 días | mié 18/04/18 | mié 18/04/18 | 35 |
| Submission of PEC 3 | 0 días | jue 24/05/18 | jue 24/05/18 | 36 |
| Submission of project report | 0 días | lun 11/06/18 | lun 11/06/18 | 37 |
| Submission of project presentation | 0 días | dom 17/06/18 | dom 17/06/18 | 38 |
| ▸ Project Assessment | 5 días | lun 18/06/18 | dom 24/06/18 | |
| Review Board | 6 días | lun 18/06/18 | dom 24/06/18 | |

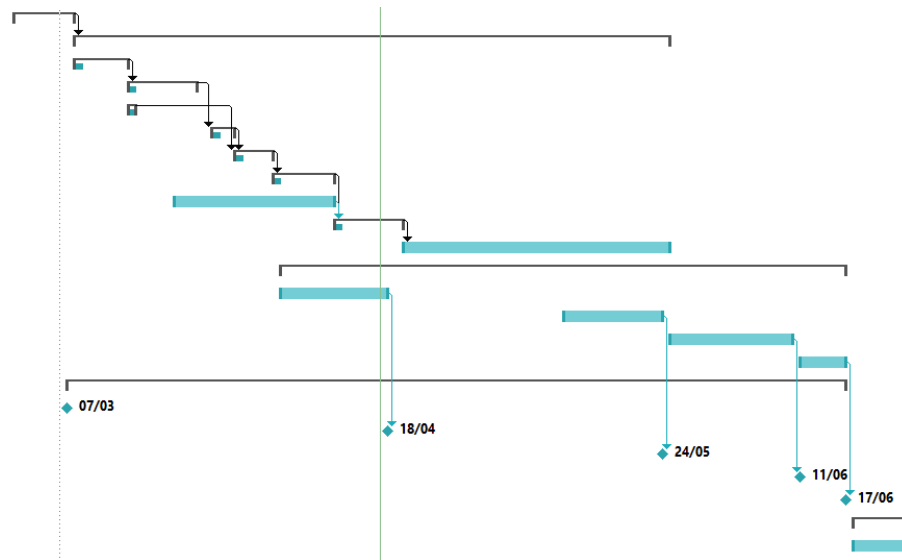


Figure 1: Project Gantt chart, describing the planned work packages

1.4 Related work

IoT demonstrators based on OpenWSN are commonly found in the literature. In particular, other students of the UOC has undertaken master's dissertations under the same topic.

- [6] presented an IoT demonstrator based on 802.15.4e, OpenMote, OpenWSN and Thethings.iO.
- [7] also presented an IoT demonstrator based on 802.15.4e, OpenMote, OpenWSN and Thethings.iO.

Both dissertations are in essence similar, although some implementation differences exist. For example, one of them sent alarms from Thethings.iO platform using SMS, while the other us emails for the same purpose.

In contrast to those works, the current dissertation provides the following novelties.

- More variety of statistics, going beyond the average/maximum/minimum value of the comfort indicators.
- More detailed database design, which allows readers to build on top of the obtained results in an easier manner.
- Use of PostgreSQL instead of alternatives such as MySQL. PostgreSQL is a highly influential database system, and therefore the study of its suitability in an IoT demonstrator can prove beneficial for research on the topic.
- Power consumption has been considered for the comparison of cloud and edge models.
- Analysis of the use of Contiki OS for the OpenMote devices. Although that it has not finally being possible to implement the design in real hardware, some progress has been made into analyzing the current relevant of Contiki OS in the context of IoT and WSNs.
- Although a minor difference that might not be noteworthy, the fact that the master's dissertation is written in English can help to increase its potential impact, and help to create a more connected and international research environment.

Besides master's dissertations at UOC, many investigators are applying open source technologies such as OpenWSN and OpenMote to different fields. Some examples include:

- [8] uses OpenWSN to design a WSN system for monitoring patient health.
- [9] used OpenMote and OpenWSN to perform energy measurements during operation of the 802.15.4e protocol in TSCH mode.
- [10] analyzes the implementation of the TSCH extension of the 802.15.4e protocol in different devices, including OpenWSN and Contiki OS, as well as different hardware platforms, such as TelosB, Zolertia Z1, and OpenMote.

2 IoT components, technologies and protocols

The purpose of this section is presenting the main elements of which the system is composed. Therefore, a theoretical introduction to the relevant technologies and protocols will be provided.

2.1 Open Hardware and Open Source solutions for the IoT

The status of open source applications shows that they are not only a valid replacement for existing proprietary products, but also a crucial point of innovation. In fact, many of the most interesting and innovating technologies are being developed as open source projects. Some examples include the GNU/Linux operating system, which is quintessential to many different aspects of modern computing, including IoT, the open hardware Arduino prototyping board and RabbitMQ, and opensource message which is typically used in some IoT applications.

On the other hand, the open hardware community is not as mainstream, most probably due to the increased difficulty of working with a fully proprietary ecosystem of existing electronic devices. However, great progress is being made, with systems such as Arduino and Raspberry Pi becoming referents in the hobbyist and educational areas, but also in professional applications [11].

In the context of wireless sensors, the trend seems to go open hardware and open source [12], with valid solutions existing for prototyping and implementation of recent wireless protocols, including those which are still in draft version. This allows for a faster adoption and in-depth research of new technologies, since open hardware and source allows to share acquired knowledge easily, usually creating a thriving community of academics and enthusiasts that can provide accurate feedback to the industry.

2.2 Operating systems for embedded devices and wireless sensors

The purpose of this section is to present the most relevant open source operating systems which are currently available, highlighting the main differences between them. As it was introduced in the previous section, the use of open source and open hardware platforms is a strong trend among wireless sensor networks researches. Therefore, despite their differences, all the following operating systems share some common traits, which form a common theme in this research area:

- Low memory requirements.
- Low energy consumption.
- Off-the-shelf support for popular network stacks and standards.
- Flexibility, understood as the possibility of easily installing, updating or modifying embedded applications in the sensors.
- Ease of use.
- Public and accessible documentation.

2.2.1 TinyOS

TinyOS is a hardware-agnostic operating system aimed at microcontroller-based sensor devices with networking capabilities, developed under the BSD open source license [13].

More specifically, it is a programming framework rather than a fully-pledged operating system. This approach provides developers with a set of components that make it possible to *create* a specific OS suited for each hardware platform and application. The OS required only 400 B of storage, while applications built on top of it require around 15 KiB [14].

TinyOS supports many network protocols: FTSP (time synchronization) [15] , CTP (data collection) [16], Trickle (data dissemination) [17], RPL (packet routing) [18] and IPv6 support for 802.15.4. These features support low-power operation, making it possible to use TinyOS in battery-constrained applications.

The programming model of TinyOS is Event. It is based on non-blocking function calls, i.e. execution of applications does not stop when calling a function. Instead, the normal flow of operation continues after the system call, but the function is not actually executed until sometime after, being notified by means of events. This model is useful due to the resource constraint of the devices to which it is intended. The programming model is implemented using the NesC language, developed by the TinyOS working groups, which is an extension of C.

This has the disadvantage of not being suitable for CPU intensive applications, since they lock the execution of the program for too long, preventing other events from being processed. To tackle this problem TinyOS implements a threads library.

TinyOS supports many hardware platforms: telos family, micaZ, IRIS, mica2, the shimmer family, epic, mulle, tinynode, span, and iMote2.

2.2.2 RIOT OS

RIOT OS main goal is to create a modern operating system which respects the constraints present in IoT applications while presenting a developer friendly environment, as well as providing access to features of general purpose operating systems such as native multithreading [19].

Design aspects:

- The kernel can be built so that is monolithic, layered or even take the form of a microkernel.
- The scheduling strategy can be varied to support real-time applications and different degrees of user interaction.
- Tasks can be executed either in the same context or in their own memory stack.

Architecture:

- It natively supports multithreading.
- It implements a full TCP/IP stack.
- The modular approach to the kernel allows to isolate components of the system so that a failure in one part of it does not translate to the whole system.

Technical Details:

- RIOT enforces constant periods for kernel tasks to fulfill strong real-time requirements.
- Static memory allocation is used for the kernel, but applications can make use of dynamic memory allocation.
- RIOT implements a scheduler that works without any periodic events, so that it can switch to the idle state whenever there are no more pending tasks, and stays in that state as long as possible.

2.2.3 Contiki OS

As any other operating systems aimed at WSNs, Contiki [20] is designed for memory constrained systems, with a focus on low energy consumption. Additionally, one of Contiki's main design drivers is to make it easy to replace or update running programs directly on the network, i.e. dynamically download code without requiring a full reload of Contiki's core and the application. As a result, it is considerably easier to manage existing deployments, since it is no longer necessary to remove each sensor, re-program it and install it again on its previous location.

Moreover, due to the smaller amount of memory required for applications, energy consumption during transmission is reduced, as well as transmission time, which optimizes energy use in battery-powered devices.

Contiki's architecture is event-driven, with optional preemptive multi-threading capabilities, implemented as a library which is linked to application programs. Therefore, multithreading is not part of the kernel, reducing its complexity and avoiding the overhead of managing multiple stacks.

Contiki is not intended for a specific hardware platform. Instead, it is designed for portability. Consequently, Contiki's approach is based on only abstracting key components, namely CPU multiplexing and support for loadable programs and services. Therefore, a reduced number of system components need to be ported: boot up code, device drivers, architecture specific parts of the program loader, and the stack switching code of the multi-threading library.

The following components can be found in Contiki: kernel, libraries, program loader and processes (either application programs or services). Services are special processes that provide common functionality to be used by several application programs. An example of how memory is partitioned during an application execution is shown in Figure 2.

A process is composed of an event handler function and an optional poll handler function. Processes can keep a state. Inter-process communication is done by posting events.

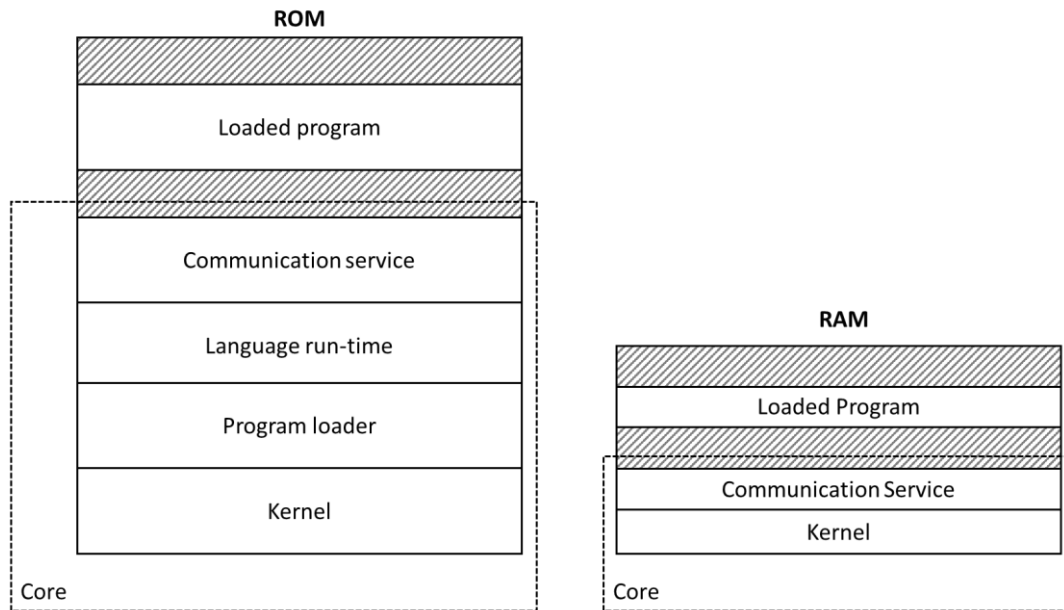


Figure 2: Partition into ROM and RAM memory in Contiki OS. Adapted from [20].

2.2.4 Comparison of WSN operating systems

A comparison between TinyOS and Contiki with other operating systems is presented in [21], which has been adapted and summarized in Table 1. The row detailing RIOT OS characteristics is entirely new.

| OS/Feature | Architecture | Programming model | Scheduling | Memory Management and Protection | Resource Sharing |
|------------|------------------------------|---|--|---|--|
| TinyOS | Monolithic | Primarily event Driven, support for, TOS threads has been added | FIFO | Static Memory Management with memory protection | Virtualization and Completion Events |
| Contiki | Modular | Protothreads and events | Events are fired as they occur. Interrupts execute w.r.t. priority | Dynamic memory management and linking. No process address space protection. | Serialized Access |
| RIOT | Either monolithic or layered | Real-time. Support for multithreading | Minimized scheduler for interrupt service routines | Dynamic memory management | Either all task in the same context or one context for each task |

Table 1: Comparison of most popular WSN operating systems

One of the main advantages of Contiki OS over other options is that it has been designed to be updatable via low-rate connections. This makes it attractive for

IoT applications, since it removes the necessity of retrieving the devices, flashing them and putting them back in place.

Another advantage of Contiki over RIOT OS is that Contiki fully supports board simulation by means of Cooja. The possibility of simulating sensor running the operating system which will be finally used greatly facilitates the initial phases of the design, due to the fact that it can be validated even before real hardware is used.

Finally, Contiki has been in development since 2002, and has a solid codebase, with hundreds of contributors, guaranteeing the health of the project.

Therefore, it can be concluded that Contiki as valid choice and a worthy operating system to be studied.

2.3 IoT Protocols

In spite of the fact that IoT shares many protocols and technologies, such as the 802.11 suite of protocols (Wi-Fi) with other areas of computer sciences, its particularities have created the necessity of designing specific protocols that are more suitable for energy and resource constrained devices that operate in the context of IoT.

For that reason, this section presents some of those protocols, which will also be used as part of the proposed design and implemented solution.

2.3.1 Brief overview of the OSI model

Although knowledge of the OSI model should be considered basic for any form of study of IT systems, it has been deemed useful to include at least a brief description of the OSI layers in this dissertation.

The OSI (Open System Interconnection) model is a theoretical representation of how communication functions performed by network-aware devices can be divided. Table 2 describes the seven layers which are defined by the model, and describes its function [22].

| | Name | Description |
|---|--------------|---|
| 1 | Physical | Defines the physical and electrical characteristics of the network. |
| 2 | Data Link | In the context of 802 standards, this layer is divided in two sub-layers: <i>Medium access control (MAC)</i> layer, which controls how and when do devices gain access to the medium, and the <i>Logical link control (LLC)</i> layer, which controls error checking, frame synchronization and data encapsulation of upper layers. |
| 3 | Network | Provides the required functionality for establishing, maintaining and terminating connections in the context of networks of devices. |
| 4 | Transport | Provides functions regarding how data reliability and integrity are to be ensured. |
| 5 | Session | Defines how continuous exchange of information is performed, understood as a sequence of multiple transmissions between the same hosts. |
| 6 | Presentation | It provides the means for translating the information received so that it can be understood by the application. This includes aspects such as data compression and encryption. |
| 7 | Application | End-user protocols such as ftp or mail. |

Table 2: overview of the layers which are defined by the OSI model

2.3.2 Infrastructure protocols

Infrastructure protocols are a key component of all IoT-based systems. The reason for this is that they provide the low-level transport capabilities for the upper layers. Hence, they also play a crucial role in fulfilling the requirements of resource constrained systems, as they directly influence in how much energy is spent transmitting and receiving information from the medium.

Therefore, a common design driver in these protocols is the focus on low power consumption.

2.3.2.1 RPL

The Routing Protocol for Low Power and Lossy Networks is a link-independent routing protocol based on IPv6 for resource-constrained nodes called, designed for building a robust topology over lossy links.

The core of RPL is the Destination Oriented Directed Acyclic Graph (DODAG), which describes how information is routed between nodes. The DODAG has a single root, and each node in it are aware of its parents, but not of its childer.

To maintain the routing topology, four types of control messages are used:

- DODAG Information Object (DIO) is used to keep the current rank (level) of the node and choose the preferred parent path.
- Destination Advertisement Object (DAO): provides upward traffic as well as downward traffic support.

- DODAG Information Solicitation (DIS): used by a node to acquire DIO messages from a reachable adjacent node.
- DAO Acknowledgment (DAO-ACK): response to a DAO message.

2.3.2.2 IEEE Std 802.15.4: Low-Rate Wireless Personal Area Networks

The goal of the 802.15.4 standard, developed by the IEEE, is to define the physical (PHY) layer and Medium Access Control (MAC) sublayer layer of the OSI model so that it allows for low-cost, low-power communications for Wireless personal area networks (WPANs) [23].

As it can be concluded, its focus on low-power consumption, low data rate, low cost and high message throughput make it a suitable protocol for the IoT.

Another key feature which makes it interesting is that it supports a significant number of nodes, in the order of tens of thousands. Therefore, it can be particularly interesting for extended IoT deployments, where thousands of devices and sensors collect and send information through the network. An example of this can be monitoring networks in agricultural areas.

802.15.4 also provides security capabilities, allowing for the encryption of the sent data as well as providing authentication mechanisms to control access to the network. However, 802.15.4 lacks any kind of processes to provide an agreed level of QoS.

2.3.2.2.1 Network nodes

802.15.4 can support two different kinds of nodes, namely Full and Reduced Function Devices (FFD and RFD, respectively).

- FFD: full-pledged nodes which can act as coordinators of the network.
 - PAN Coordinator: node responsible for the creation, control and maintenance of the network. An IEEE 802.15.4 network has exactly one PAN coordinator.
 - Coordinator: a device in an LR WPAN that provides synchronization services to other devices in the LR WPAN.
 - Normal node: node which can have bidirectional communication with other nodes but which cannot act as a coordinator.
- RFD: they are restricted to communication with coordinator nodes, and they can only be part of star topologies.

2.3.2.2.2 Topologies

Two different topologies are supported:

- **Star topology:** the communication is established between devices and a single central controller, the PAN coordinator. Each which is usually either the initiation point or the termination point for network communications, except for the PAN coordinator, which is focused on enabling routing of information on the network.

- **Peer-to-peer topology:** although a PAN coordinator exists, any device is able to communicate with any other device as long as they are in range of one another. A mesh networking topology can be mounted on top of the Peer-to-peer topology.

2.3.2.2.3 Physical layer (PHY)

Functions of the Physical Layer can be divided in two groups:

- The **PHY data service** enables the transmission and reception of PHY protocol data units (PPDUs) across the physical radio channel.
- The **PHY management** service provides activation and deactivation of the radio transceiver, ED, LQI, channel selection, clear channel assessment (CCA), and transmitting as well as receiving packets across the physical medium.

2.3.2.2.4 MAC sublayer

Functions of the MAC sublayer can be divided in two groups:

The MAC sublayer provides two services: the MAC data service and the MAC management service interfacing to the MAC sublayer management entity (MLME) service access point (SAP) (known as MLME-SAP).

- The MAC data service enables the transmission and reception of MAC protocol data units (MPDUs) across the PHY data service.
- The MAC management service provides beacon management, channel access, GTS management, frame validation, acknowledged frame delivery, association, and disassociation. In addition, the MAC sublayer provides hooks for implementing application-appropriate security mechanisms.

2.3.3 Application Protocols

Protocols in the application layer aim to provide services which are requested by users. Therefore, the possible forms in which application protocols manifest are very divers, and can depend on the vertical market. However, many IoT protocols are designed to be suitable for many different environments.

This section briefly presents some of such protocols.

2.3.3.1 MQTT

The Message Queue Telemetry Transport (MQTT) [24] is an application protocol in which information is formatted in for of messages. Although it was created in 1999 by independent researchers, it was standardized in 2013.

MQTT can be used to connect embedded devices with each other, as well as with applications from upper layers. It supports different routing mechanisms, including one-to-one, one-to-many and many-to-many.

The message exchange is based on a publish/subscribe, i.e. nodes in the network subscribe to a “topic” to listen to all information sent (published) to that particular topic. For example, one device can subscribe to the “temperature” channel, and therefore will receive all “temperature” readings from other devices.

Since MQTT is built on top of the TCP protocol and it does not make heavy use of resources, MQTT is suitable for resource constrained devices that use unreliable or low bandwidth links.

2.3.3.2 CoaP protocol

The CoaP (Constrained Application Protocol) [25] is a web transfer protocol aimed at offering an optimum solution for enabling communication in constrained devices. It is designed under the REST model, and therefore it works in a very similar way to HTTP. Consequently, starting to use CoAP is straightforward for people familiarized with existing web technologies. The main features of CoAP are described below:

- Suitable for machine-to-machine (M2M) communication.
- Transport over UDP, extended by means of Datagram Transport Layer Security (DTLS) to optionally support unicast and multicast requests.
- Asynchronous message exchanges.
- Simple and light header to reduce complexity and power consumption.
- In a similar manner to HTTP, a CoAP client requests an action (using a Method Code) on a resource (identified by a URI) on a server, to which the server response with the corresponding Response code, indicating the status of the request.
- Support of Content-type.
- Simple proxy and caching capabilities.

CoAP requests can make use of different methods, depending on the type of action that is to be applied on the destination resource. There are four CoAP methods available:

- **GET**: retrieves a representation of the resource identified by the request URI. Possible response codes include 2.05 (Content) or 2.03 (Valid) upon success.
- **POST**: requests that the resource transported in the request to be processed. Although the function performed by the POST method is defined on the server side, it usually corresponds with creating or updating a new resource. As a result, the 2.01 (Created), 2.04 (Changed) or 2.02 (Deleted) response code should be sent as an answer upon success.
- **PUT**: requests that the resource identified by the request URI be updated or created with the enclosed data. If the resource is indeed updated, the 2.04 (Changed) response code is to be returned. On the other hand, if the resource does not exist, and the server decides to create it with that URI, a 2.01 (Created) response code is returned.

- **DELETE**: requests that the resource identified by the request URI be deleted. If the request succeeds, A 2.02 (Deleted) response code SHOULD be returned.

A thorough list of all available response codes is out of the scope of this document. However, it is worth mentioning that error codes common in HTTP do exist, e.g. 4.03 (Forbidden) or 4.04 (Not Found).

In addition to the request methods described, CoAP messages can be classified according to their desired reliability. Messages which need to be confirmed are labeled as CON (Confirmable), while those who do not need to be confirmed are labeled as NON (Non-confirmable). CON messages are answered with an ACK (Acknowledgment) message, indicating that the server has received the request. An example of communication between client and server using CoAP is shown in Figure 3.

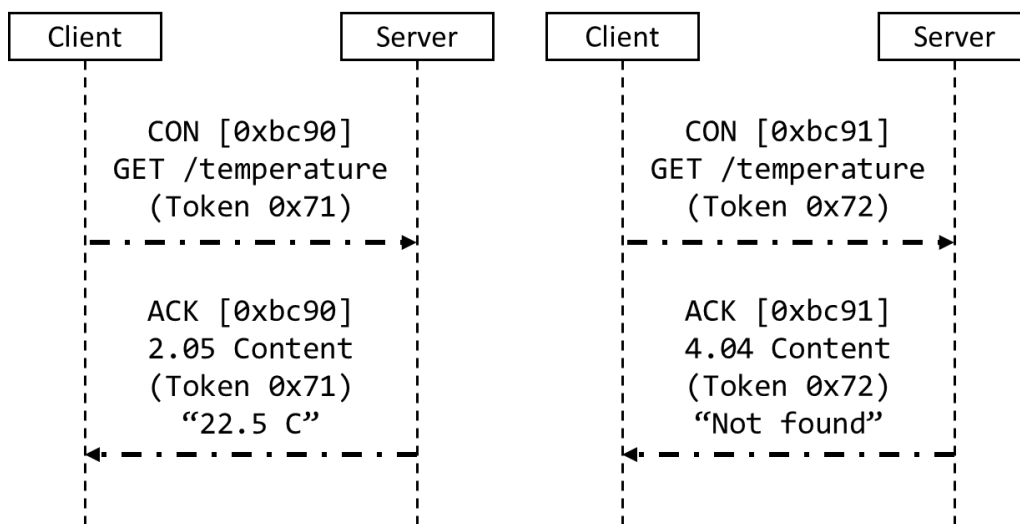


Figure 3: two example message exchanges. In both cases, the message type, CON, indicates that a confirmation is required. In the first case, the request ends successfully, while in the second it does not. However, in both occasions, an ACK message is sent. Adapted from [25].

2.4 Strategies for data processing and storage

In the context of IoT, the traditional approach has been collecting data on the edge of the network by means of sensors, and then centralizing such information transmitting it to a main server for later processing. In recent times, this has evolved to become what is known as Cloud Computing, in which applications can make use of a vast amount of cloud platforms for storage, analysis, processing and display of information.

However, the increase in complexity and capacity of edge nodes has allowed for a new approach, edge computing. In the following sections, both Cloud and Edge computing models will be introduced, highlighting their differences, strengths and weaknesses.

2.4.1 Cloud computing

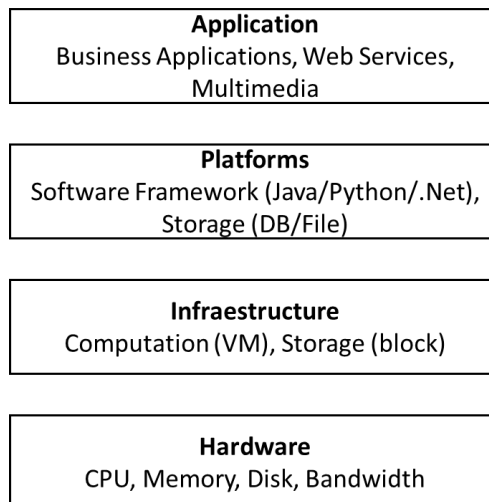


Figure 4. Overview of Cloud Computing Architecture. Adapted from [26].

Cloud computing has been the most extended system for exploiting IoT-based applications until date. It can be defined as a computation model where a shared pool of network-accessible computing resources can be autonomously and effortlessly provisioned on-demand, thus allowing for desired characteristics such as ubiquity and full availability of computing capabilities. The main features that define a Cloud Computing services are listed below [27]:

- **On-demand self-service:** cloud users can access to automatically assigned computing capabilities without required human interaction.
- **Broad network access:** cloud services can be accessed from commonly available standard network protocols.
- **Resource pooling:** resources are shared among users, but ensuring isolation between application and guaranteeing service levels by means of virtualization of such resources.
- **Rapid elasticity:** computing capabilities must be capable of dynamically adapting to the evolving requirements of an application in real time.
- **Measured service:** service elements such as storage and network capacity need to be monitorable and configurable.

Although cloud services are usually associated with outsourcing, this does not need to be the case. Different deployment models can be followed, as described in []:

- **Private cloud:** a single organization manages, owns and operates the cloud.
- **Community cloud:** a group of organizations with shared concerns operates the cloud.
- **Public cloud:** the cloud is available for open use by general public, being usually provided by public entities like governments of academic institutions.

- **Hybrid cloud:** a combination of the aforementioned types of clouds, with the capability of exchanging data and providing application interoperability.

As it can be concluded from the previously described characteristics, the essence of cloud computing is that it allows a company to outsource its computing and storage necessities, which are provided by an external company. Many advantages can be obtained from this approach, mainly because the centralizing and sharing resources between a significant number of applications, belonging to different companies, helps to reduce costs. In addition, the complexity of managing the computation systems is delegated into an external company, allowing the cloud users to focus on their own products without having to worry about its deployment aspects. Other critical benefits obtained from cloud computing include [26]:

- **It reduces the necessity of planning ahead:** cloud services elasticity means that users do not need to be concerned about over-provisioning or under-provisioning risks. The use of computing resources can grow or be reduced depending on the application needs, and, more importantly, it is not necessary to re-negotiate or re-define the resource usage, since this adaptation happens automatically and autonomously.
- **It follows a “pay-as-you-go” model:** companies which make use of cloud services can decide when and how increase the contracted capacity. As a result, the application can be rapidly start working and generating benefits without having to worry about infrastructure deployment.
- **It is highly scalable:** from a user perspective, infinite resources are available. If the application resource demands are increased, the cloud platform will provide them without issues.
- **Easy access:** cloud services are designed to be easy to use, allowing users to access them using traditional methods such as web browsing from a variety of devices, including computers and laptops.

Finally, even though cloud computing brings multiple benefits, it comes with its associated risks and challenges to be solved [28]:

- **Business continuity and service availability:** users demand extremely high standards of availability, perceiving even small service disruptions as major threads.
- **Data Lock-in:** reliable methods need to be provided to achieve platform interoperability and allow for data migrations between providers.
- **Data Confidentiality/Auditability:** the privacy and security of data stored in external platforms needs to be guaranteed, and methods for auditing it must be available.
- **Data transfer bottlenecks:** transferring huge amounts of information between locations (as required for redundancy, for example) involves a high cost.
- **Performance unpredictability:** the fact that computing and network resources are shared creates some level of concern regarding fluctuations in performance if such resources become insufficient for serving all users simultaneously. Virtualization and containerization are key for addressing this issue.

- **Scalable storage:** the fact that “infinite storage capacity” needs to be provided results in severe storage demands.
- **Bugs in large-scale distributed systems:** unlike traditional centralized infrastructures, identifying and fixing bugs becomes extremely difficult due to the overwhelming number of actors and entities involved at the same time.
- **Scaling quickly:** dynamically providing the appropriate amount of resources can prove difficult in some applications, such as algorithmically-intensive applications.
- **Software licensing:** purchased software licenses (e.g. Oracle database licenses used by cloud providers) are limited to particular machines and periods of time. This does not fit well with the “pay-as-you-go” philosophy of cloud platforms, since it reduces its operation flexibility. New licensing models or open source alternatives need to be explored.

2.4.2 Edge computing

In simple terms, edge computing differs from the traditional cloud-based approach in which computing and storage is performed close to where data is generated [29]. Thus, under this paradigm, part of the intelligence of the network is moved closer to the edge nodes of the network.

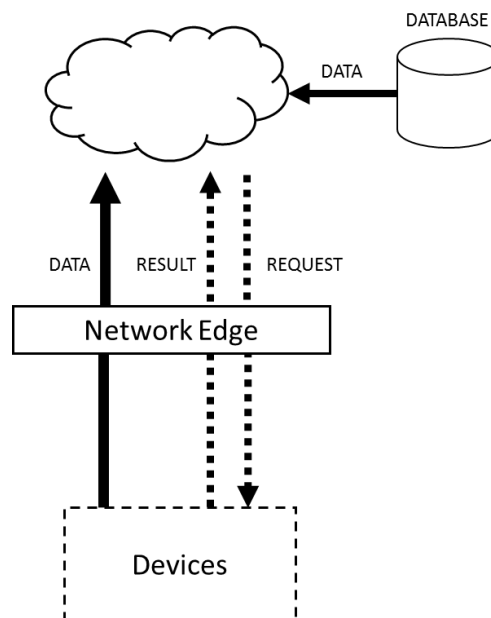


Figure 5: Edge computing. Edge computing is performed between the source (devices) and the cloud platform. Some of the load, in both upstream and downstream directions, is handled by the network edge and not by the cloud platform, thus effectively reducing response time and making a more efficient use of network capacity. Adapted from [29].

The necessity of edge computing arises from some of today’s threats to the development of IoT applications, which are presented below:

- **Network capacity:** although the processing capacity of cloud platforms has increased considerably over the years, this has not been accompanied by an equal amount of improvement in the network capacity. Due to the sheer amount of IoT devices that will be connected

to the internet, the network used to transmit data from data sources to processing centers will become a bottleneck, limiting the growth of the IoT ecosystem.

- **Privacy:** as IoT spreads to more contexts, concerns is raised about how to handle sensitive information collected by sensors, which is to be transmitted to data centers. In fields such as healthcare, it would be desirable to protect the data collected before sending it, or even more, not even having to transmit such sensitive data over the network, avoiding potential threats.

The processing centers can take many forms used in edge computing, and range from fully-pledged, although smaller, data centers (micro-datacenters), to devices very close to the end user, such as smartphones. Their common characteristic is that they need to be placed between data sources and cloud platforms. This is shown in Figure 5. As it can be observed, data, requests and services pass through the edge processing centers before continuing to the cloud. In this manner, part of the services required by the application can be provided by the edge devices without requiring the intervention of the cloud platform.

Several advantages can be obtained from this model:

- **Reduced network load:** since edge devices handle part of the application services, it is not necessary to transmit all the data to the cloud. Consequently, cloud platforms can provide service to a bigger number of applications, therefore guaranteeing the scalability of IoT applications.
- **Reduced latency:** due to the fact that processing takes place closer to the end devices, the communication between them and processing centers is much faster. This is especially important in time-constrained applications which require a quick response time.
- **Power usage:** some applications can benefit from reduced consumption, since processing data closer to the edge can prove to be more energy efficient than performing such task at the cloud and then transferring result to the end application.
- **Improved privacy:** sensitive data can be better protected by processing it before sending it to the cloud. Additionally, in some cases, all the computation can take place at the edge, so that it is not necessary to transmit it, greatly reducing the threat of data theft in the network middle nodes.
- **Greater redundancy:** the load distribution among cloud platforms and edge processing centers makes it possible to provide continuous service, because in case of failure in the cloud platform, part or all of the computing activities can take place at an alternative edge center.

As it has been described, there are many benefits in edge computing. However, this means that edge computing systems need to be designed to fulfil the same requirements as traditional cloud-based applications, because of the necessity of availability and speed of response, which cannot be sacrificed. As a result, aspects such as differentiation, extensibility, isolation and reliability must be considered [29].

In particular, some specific issues need to be solved before edge computing can become completely mainstream. Some of the most relevant ones are listed below [30]:

- **Disperse ecosystem of protocols and technologies:** many solutions exist for performing communication, processing and storage tasks. While this is good for innovation, it also makes it more difficult to design for interoperability and simplicity.
- **Distributed management:** losing cloud computing centralized architecture increases management complexity and costs. New approaches and technical solutions need to be developed efficiently to communicate and handle an increasing number of distributed processing centers.
- **Weaker security:** less centralization also means increasing the attack surface. Security in distributed systems is more difficult to guarantee, and therefore should be considered a priority in the design of edge applications.
- **Undeveloped ecosystem:** for developers and companies to seriously consider edge computing, a rich ecosystem of edge computing providers and applications must be put in place. However, edge computing providers need real-world applications to justify edge computing deployments, so the situation is blocked. Research, in conjunction with public and private investment can help to clear the situation.

Before concluding this section, it is worth mentioning that in spite of its nobility, real edge computing deployments are already in place. Some examples include the Radio Applications Cloud Server (RACS) for 4G/LTE networks and the Living Edge Lab [30].

From the perspective of IoT applications, Edge Computing is a specially interesting model, mainly because of its close to real-time nature, which allows for instantaneous response and subsequent decision making. This is particularly useful in industrial applications where collected information needs to be applied instantly, as the delay of sending the data back to the cloud, processing it and receiving the results might be unacceptable.

In addition, its distributed layout makes it possible to use collected data, and associated computations, even if no network connectivity exists, which can be the case in remote deployments like agricultural settings.

Another reason why Edge Computing is useful for IoT is because of its scalability. In order to leverage the power of IoT, it is often required to deploy hundreds, or even thousands of devices. This causes traffic to grow exponentially, therefore exhausting networking, storage and computing capabilities. Edge Computing reduces this threat by reducing the amount of data that needs to be transferred and stored in central nodes, allowing for the growth of the application without putting the infrastructure network to its limits.

2.4.3 Cloud computing vs edge computing

Previous sections have described the characteristics of both Cloud and Edge computing paradigms. While both of them are valid approaches for implementing IoT solutions, the suitability of each model will depend on the application at hand. Table 3 presents a possible criterion for choosing between both approaches depending on which is the key design driver of the application. Different applications have different requirements, and knowing which of them are the most relevant makes it easier to select the model to choose.

| Design driver | Choice | Description |
|--------------------------------|-----------------|---|
| Low latency | Edge Computing | The proximity between processing centers and data sources greatly reduces the time required to transfer data and thus improves the overall system response time. |
| Low network traffic | Edge Computing | Only computed data is transferred to central nodes. Consequently, the amount of network traffic is significantly smaller. |
| High scalability | Edge Computing | Networks can grow naturally by adding more edge nodes. In this manner, computation power grows as the same rate as the application. |
| High redundancy | Edge Computing | Although a high level of redundancy can be achieved in Cloud Platforms (e.g. distributed datacenters), redundancy is more naturally achieved in Edge Computing deployments. |
| Complex data processing | Cloud Computing | When complex data analysis is required, such as in big data applications, the reduced computation power of edge devices make them unsuitable for performing the required operations. |
| Centralized management | Cloud Computing | Cloud Computing deployments are comparatively easier to manage and maintain, since all resources can be controlled from a reduced set of nodes. On the other hand, debugging applications implemented in a Edge Computing philosophy can prove to be difficult. |
| Cost reduction | Cloud Computing | Since Cloud computing centers can be shared by multiple applications, even from different companies, it is possible to optimize resource consumption and therefore reduce costs. |

Table 3: comparison of cloud and Edge computing models based on different key design drivers.

In summary, choosing Edge or Cloud computing depends on the requirements of the application. In broad terms, if such application requires computation in real-time, or with very small delay, then the Edge computing model is more suitable. Another typical use of Edge Computing is when end devices already have a significant amount of computation power, such as in smartphones [31]. If instead the application is not time constrained, and other aspects such as cost or

management complexity are more important, the traditional cloud computing model can be more beneficial.

However, it must be noticed that despite their differences, they are not mutually exclusive. Using mixed models can help to obtain all their respective benefits without sacrifices.

2.5 Hardware platforms for IoT

One of the key enablers of IoT has been the emergence of small, cheap and flexible computer hardware which facilitates the rapid design and deployment of IoT applications. A rich ecosystem of development platforms exists, from general purpose platforms, oriented for prototyping, to fully customized systems.

In the context of this dissertation, only prototyping boards have been considered, since it makes it possible to work on an initial version of the project, validating its model and design, without having to worry about secondary aspects like manufacturing and testing the platform. Once the model is validated, application-specific platforms can be designed, if the cost saving are deemed significant enough.

Some of the most relevant prototyping platforms include [32]:

- **Arduino:** open source, microcontroller-based computing platform. The board includes a microcontroller and all related components that are required to operate and interface with it, including memory, oscillators for generating the clock signal and input/output pins, which can be connected to sensors and actuators. In addition, a development environment is provided, which allows the user to program applications using a dialect of C++, instead of having to write assembly code.
- **BeagleBone Black:** single-board computer based on low-power Texas Instruments processors, using the ARM Cortex - A8 core. It has a size (roughly the size of a credit card), and it is able to run full operating systems, including GNU/Linux and Android. It has a much higher processing power compared to Arduino, as it is more like a computer than a controller.
- **Phidgets:** rather than a pre-built, Phidgets follows a modular philosophy. Systems are formed by combining building blocks (temperature sensors, RFID tags, switches, etc.), all of which communicate using a USB interfaces. The resulting system can be programmed using C++ by means a collection of library and a API. Due to these features, Phidgets enables programmers to rapidly develop physical interfaces without the need for extent knowledge in electronics design issues.
- **Udoo:** open hardware minicomputer based on the ARM i.MX6 Freescale process, being compatible with Android and GNU/Linux. In addition, it has and embedded Arduino-compatible board. Therefore, it combines elements of computers and controllers in a single device. It provides connectivity using the usual interfaces, such as Ethernet, WiFi and USB,

while also providing connectivity with sensors and actuators by means of the Arduino-compatible board.

Despite the suitability of these platforms for many applications, the selected platform for the purpose of this project has been **Raspberry Pi**. Its main characteristics, as well as its advantages over the options, are described in the following section.

2.5.1 Raspberry Pi 3 B+

The Raspberry Pi [33] is a Single Board Computer (SBC) designed to be open and accessible, making it suitable for education purposes, as well as for DIY projects, research and prototyping. It enjoys an enormous popularity, mainly because of its high-quality documentation and convenient price (around 40€). Consequently, a significant number of examples, libraries and off-the-shelf software exists for this platform, greatly simplifying the learning curve and the bootstrap of new developments.

The reasons for choosing the Raspberry Pi as the hardware for this project are summarized below:

- It can run GNU/Linux, providing an environment with an enormous amount of existing software (e.g. Python, PostgreSQL, etc.), which in addition feels familiar for developers.
- Large RAM memory, making it suitable for applications with non-negligible complexity, such as those present in Edge Computing systems.
- It supports expandable memory, thus allowing storage of up to 128 GiB. This is more than enough memory for the purposes of this project.
- It has 4 cores, and therefore supports multithreading, increasing performance.
- Off-the-shelf WiFi, Bluetooth, Ethernet and USB connectivity.
- Expandable by means of widely available shields, aimed at specific purposes like LTE connectivity, which can be very useful in IoT applications.
- It can be battery-powered.
- More importantly, it is a highly popular project with a thriving community of enthusiasts. Consequently, documentation is highly available and of great quality. This also includes tutorials for developing common applications. Hence, it becomes very easy to start using Raspberry Pi for prototyping purposes. These features, combined with a competitive price (see Table 4), makes it a good choice.

| | Size (mm) | Weight (g) | Cost (\$) |
|------------------|-------------------|------------|-----------|
| Raspberry Pi | 85.6 x 53.98 x 17 | 45 | 25-35 |
| Arduino Uno | 75 x 53 x 15 | 30 | 30 |
| BeagleBone Black | 86.3 x 53.3 | 39.68 | 45 |
| Phidgets | 81.3 x 53.3 | 60 | 50-200 |
| Udoo | 110 x 85 | 120-170 | 99-135 |

Table 4: Comparison of different SBC in terms of size, weight and cost.

The main specifications of the Raspberry Pi 3 Model B+ can be consulted in Table 5 [34]. As it can be seen, the Raspberry Pi implements many of the key features that a regular personal computer has, such as 802.11 connectivity and HDMI connectivity, all compressed in a size of about 47.6 cm^2 . Moreover, it provides some characteristics that make it suitable for IoT projects, including 40 GPIO pins and support for communication protocols such as Bluetooth 4.2.

With regard to the Operating System, Raspberry Pi can be operated with many different options, including officially supported: Raspbian; semi-officially supported: Ubuntu Core, Windows IoT Core, RISC OS, Ichigo Jam Pi, Ubuntu Mate; and compatible but not officially supported: Arch Linux, Chromium OS, Diet Pi and many more.

As it has been mentioned, Raspbian is the official distribution of Raspberry Pi. It is based on Debian, the widely known GNU/Linux distribution for servers and personal computers. Due to its official status and its ease of use, it has been selected as the OS for this project.

| | Value |
|--------------------------|--|
| Processor | Broadcom BCM2837B0, Cortex-A53 64-bit SoC @ 1.4 GHz |
| Memory | 1GB LPDDR2 SDRAM |
| Connectivity | <ul style="list-style-type: none"> • 2.4 GHz and 5 GHz IEEE 802.11.b/g/n/ac wireless, LAN, Bluetooth 4.2, BLE • Gigabit Ethernet over USB 2.0 (maximum throughput 300 Mbps) • 4 x USB 2.0 ports |
| Access | Extended 40-pin GPIO header |
| Video & Sound | <ul style="list-style-type: none"> • 1 x full size HDMI • MIPI DSI display port • MIPI CSI camera port • 4 pole stereo output and composite video port |
| Multimedia | H.264, MPEG-4 decode (1080p30); H.264 encode (1080p30); OpenGL ES 1.1, 2.0 graphics |
| Input power | <ul style="list-style-type: none"> • 5 V/2.5 A DC via micro USB connector • 5 V DC via GPIO header • Power over Ethernet (PoE)–enabled (requires separate PoE HAT) |
| SD card support | Micro SD format for loading operating system and data storage |

Table 5: Raspberry Pi 3 Model B+ echnical specifications

2.6 IoT Cloud Platforms

Several crucial aspects of IoT applications cannot be directly managed at the edge of the network, since they required a significant amount of storage and computation capacity which is out of the scope of sensor devices and gateways. Therefore, a centralized point of control is required, from which activities such as data monitoring, gathering and processing, as well as device management are to be performed.

One way of addressing this issue is by using commercial IoT Cloud providers, which offer off-the-shelf tools for performing such tasks. A significant number of options exist in the market, but not all of them are aimed at the same markets or applications. Consequently, the suitability of each IoT Cloud platform needs to be investigated as part of the design of the application.

The comparison to be drawn between different platforms can be performed by evaluating the features that each platform offers. Such features can be classified according to the following groups [35]:

- Application Development
- Device Management
- System Management
- Heterogeneity Management
- Data Management
- Analytics
- Deployment Management
- Monitoring Management
- Visualization
- Research

2.6.1 thethings.iO

The IoT Cloud platform selected for this dissertation has been thethings.iO [36]. This platform provides back end solution for IoT applications, allowing its users to connect its IoT devices to the cloud for monitorization, data analysis and device management.

One of the main advantages of the platform is that it is device agnostic, i.e. any type of device can communicate with the platform, mainly thanks to the use of industry standard protocols, such as MQTT. Moreover, is it not limited to data representation, since it includes application development capabilities by means of an exposed API. Tools are provided for accessing such interfaces, including Python libraries.

A brief description of the capabilities of the platform is presented below [37]:

- *Connectivity and Normalization*: an API is provided which allows users to connect any type of device to the platform. This includes libraries for languages such as Python and Node.js, in addition to libraries for popular embedded platforms like Arduino and Raspberry Pi. Furthermore, various communication protocols can be used to perform the communication between devices and platform, including HTTP, Websockets, MQTT, CoAP and more.

- *Device & Licenses Management*: devices can be individually monitored, allowing for remote management. Moreover, thethings.iO does not limit the amount of storage that an application can make use for, therefore guaranteeing scalability.
- *Cloud Code Processing & Action Management*: the platform can be programmed to trigger specific actions upon data reception. Actions can be complemented with notifications through SMS or even Twitter.
- *Data Monitoring and Visualization*: customizable dashboards for displaying real-time information collected from the connected devices.
- *Analytics, AI, Predictive analysis*: processing of stored data using AI and machine learning techniques.
- *Interoperability and integrations*: possibility of interacting with third party services.

2.6.2 thethings.iO overview

As it has been mentioned in previous sections, communication between devices and the platform is performed by means of APIs provided by thethings.iO.

Access to devices is divided in “Products” and “Things”. Products are groups of devices which serve a common purpose. In this case, the defined product has been “Comfort Metrics”, understood as the installation of a set of sensors in a building with the objective of collecting different metrics related to comfort conditions. On the other hand, “Things” are the end devices that collect the information, which corresponds to the OpenMote devices deployed on the building.

The creation of a new product is shown in *Figure 6*. As it can be observed, the chosen serialization format has been JSON, although other options exist, including Sigfox, MessagePack and Protocol Buffer. However, given the enormous flexibility and popularity of JSON, it has been deemed as the most suitable solution for the purpose of this project.

Edit your IoT product
Here you can edit your product

Product name
Comfort Metrics

Serialization Format: JSON

Opendata

Product public resources

Thing properties (you can define default properties for all the things of the product and edit them in the Edit Description view easier)

temperature humidity light_intensity

CANCEL SAVE

Figure 6: configuration of an IoT product in thethings.io. The chosen format

Once the product has been created, “Things” can be added to it. Whenever a “Thing” is created and activated, a Thing Token is obtained, uniquely identifying it and allowing for the communication between the platform and the Raspberry Pi. An overview of the “Thing” definition is shown in *Figure 7*.

Details Edit Description ^

Name: Mote-1

Thing Id: xmQA0vt_Rg-NWHBUxBfQH7dEDkbQPux5SKOnFMRswlk

Thing Token: _KkoHNV4qR1a_WMUUFPPjcVHgcR0JqS691Zlxll8Cqg

Thing Tags: [+ Add Tag](#)

Figure 7: Definition of a “thing”. The Thing Token identifies the device which is sending the information.

While thethings.io offers several methods for transferring information to and from it, the chosen communication methods have been the provided Python library [38], mainly because of the availability and ease of use of Python, and the rapid development speeds that can be achieved thanks to the functions contained in the library. Listing 1 describes the process of installing the thethings.io Python library.

```
$ git clone https://github.com/theThings/thethings.io-python-library.git
$ sudo python setup.py install
Password:
...
>>> help("thethings")
Help on package thethings:
NAME
  thethings
FILE
  /Library/Python/2.7/site-packages/thethings/__init__.py
PACKAGE CONTENTS
```

After the Python library has been installed, it is possible to send and receive data from the platform using the write() and read() functions. Figure 8 shows how data is displayed when it is received. It is important to notice that one “Thing” can handle multiple “resources”, such as temperature and humidity, which are identified by the information sent in the JSON message.

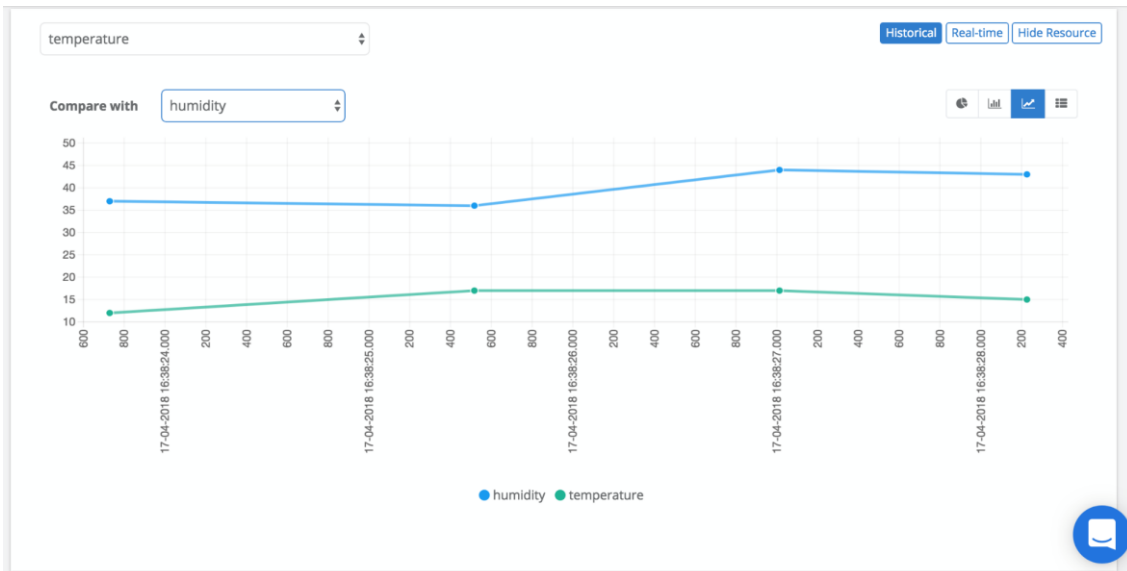


Figure 8: Example data received at thethings.iO. As it can be seen, the data is represented with its associated timestamp, so that the evolution of the measurement over time can be observed. In addition, several sources can be represented at the same time (temperature and humidity, in this case). Consequently, it becomes easy to observe correlations between them.

A custom dashboard can be configured to represent the data captured from remote devices. The dashboard is composed of widgets which are available off-the-shelf, with different types of widgets being suitable for different types of data. A new widget can be configured from the Dashboard menu using the “Add Widget” button. As it can be seen, it is simply a matter of selecting which resource is to be represented, and how is it going to be visualize (pie chart, historical evolution, etc.). A temporary dashboard example is displayed on Figure 9.

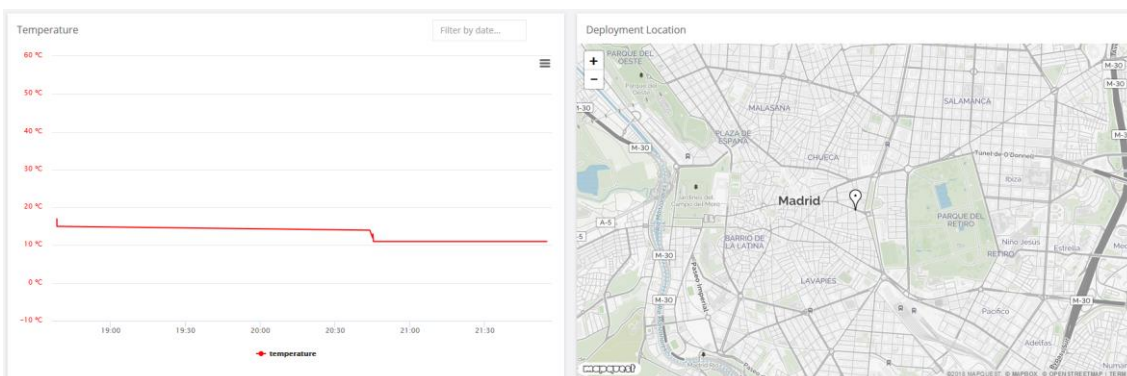


Figure 9: thethings.iO Dashboard, showing temperature and OpenMote location

2.7 Data storage: the database

The implementation of the edge computing approach requires storing the information collected from the sensors on a local database located in the Raspberry Pi. Consequently, it becomes necessary to select a database framework which fulfils all the requirements of the application. Since the operations performed by the database system for the purpose of this project are rather basic, the resulting requirements are very simple in nature. Accordingly, the following requirements have been defined.

- It is an open source solution, being freely distributable and with publicly available documentation.
- It can be controlled using the Python programming language.
- It can be installed on the Raspbian operating system.
- It is actively developed.

The current ecosystem provides many possible alternatives, from traditional proprietary software (Oracle, Microsoft's SQL Server) to fully open source solutions (MySQL, MariaDB, MongoDB). This allows users to select the tool that best suites them, with very fine-grained control of the features that are desired.

Any of the open source solutions described above could fulfil the requirements of this project. Therefore, selecting one or the other boils down to personal preference. In this regard, the database of choice has been PostgreSQL [39], mainly because it is one of the most active databases in the open source scene, with a rich community of contributors and regular global events. In addition, updated and detailed documentation is readily available, making it very easy to develop new database applications from scratch.

Some of PostgreSQL's main features are [40]:

- It is possible to create custom data types and query methods.
- It provides functionality to run stored procedures in many popular programming languages, including Java, Perl, Python, C/C++, etc.
- It implements the GiST (Generalized Search Tree) system, which allows the user to sort and search for data using different algorithms, i.e. B-tree, B+-tree, R-tree, partial sum trees, and ranked B+-trees.
- Highly extensible.
- Tools for data recovery: Write-ahead Logging (WAL), Tablespaces, Point-in-time-recovery (PITR), active standbys, etc.
- Security features: Robust access-control System, Column and row-level security, etc.

The installation process of PostgreSQL on the Raspberry Pi is presented in section 10.4 for the interested reader.

2.8 Ethics of the IoT

Any technological development, whether small or big, must address its ethical implications and its effects on society. Otherwise, technology might stop being a tool of improvement, becoming something else, however dangerous or harmful it might be.

In the case of the Internet of Things, endless possibilities are opened, with the potential of heavily determining the shape of the daily lives of the population. This, however, poses some crucial challenges that must be address. In the present section, the focus will be places on privacy and data protection.

As it has been mentioned, IoT has the potential of influencing how we live. This is achieved thanks to the collection of an enormous amount of personal information, including what we eat, how much do we sleep, what we do at home, who we talk with, and the list goes on.

In the industrial field, privacy is also relevant, although the implications are different. An IoT-enabled factory can store detailed information about all the processes taking place in it, such as number of workers, number and type of machines, temperature of operation, pieces being manufactured, etc. An external agent with access to such information has an undoubtable advantage over its competitors.

Therefore, privacy of collected and transmitted data must always be ensured.

Although manufacturers must guarantee such security by design, it is also useful to mention the existing regulation regarding data protection. The European Union Data Protection Directive [41] defines the rights that a subject (in this the user of a IoT-enabled application), has. Violation of such rights represents a law infringement action.

It has been considered useful to include a summary of such rights, since it can be considered as a checklist to have in mind when designing IoT projects:

- Breach Notification
- Right to Access
- Right to be Forgotten
- Data Portability
- Privacy by Design

3 Proposed design

3.1 System overview

The proposed system aims to provide an insightful example of a typical IoT applications, in which several elements, from the edge of the network to the cloud platform, collaborate using standard communication protocols in order to achieve a final goal. For this reason, a typical IoT home deployment is proposed, as shown in Figure 10.

The purpose of this deployment will be twofold: monitoring and actuation. Typical comfort indicators, including temperature and humidity, will be frequently logged so that its status and evolution can be observed. Based on the registered temperature, the system will be actuated by means of disabling or enabling the air conditioning system.

Furthermore, the presence in each of the rooms of the house will also be monitored. Although this is not directly related to comfort indicators, this information can be significantly beneficial for creating occupancy models which, with the appropriate analysis, can help to optimize several aspects such as energy consumption. This not only applies to the monitored home, since aggregated data from thousands of homes can provide a useful input for different disciplines which aim to improve the design of existing buildings to minimize energy consumption.

In addition, another actuation method based on presence can alert the owners in intruders are detected at home.

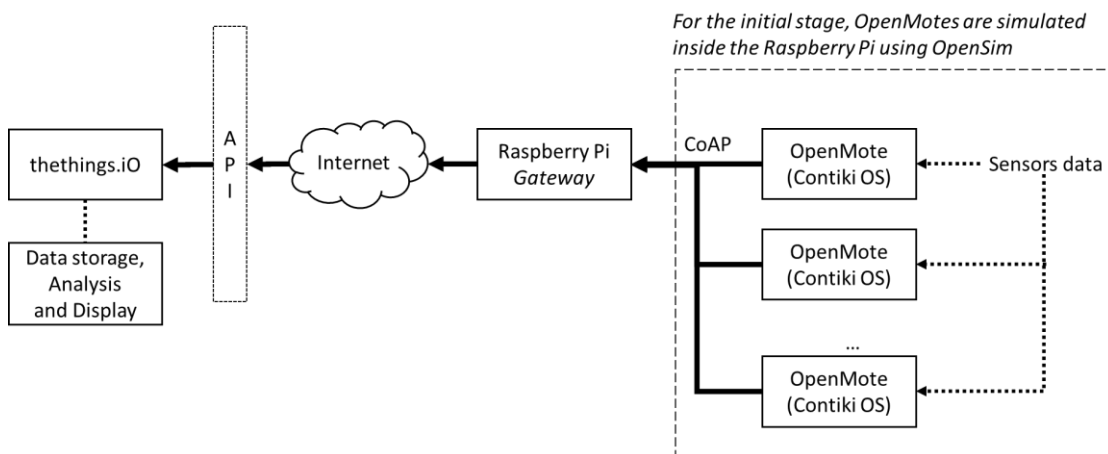


Figure 10: system overview. Sensor data is collected by OpenMote devices distributed along a property. Such data is sent to a central point, the Raspberry Pi, which acts as a gateway, processing the data and sending it to the thethings.iO platform.

As it can be observed, the system is composed of the following elements:

- **OpenMote nodes:** wireless sensor nodes which read sensors (temperature, humidity, etc.) and communicate with the Raspberry Pi

using the 802.15.4 and CoAP protocols. Motes are distributed around a house to provide a full picture of the living conditions inside it.

- **Raspberry Pi:** playing the role of a network gateway, this node is in charge of collecting the data coming from OpenMote sensors. When validating the edge computing paradigm, heavy computation operations will be performed on the Raspberry Pi, with the results being transferred to the cloud. On the other hand, when validating the cloud computing paradigm, the processing operations performed by Raspberry Pi on the collected data will be minimum. Since this node is the one connecting to the Internet, it will contain the applications requiring for communicating with thethings.iO API.
- **thethings.iO:** it acts as the cloud platform of the system, providing storage, analysis and visualization of generated and processed data. Additionally, data will be processed in this platform when validating the cloud computing paradigm.

To maximize the project outcome, and to guarantee that key objectives are achieved, two distinct project phases are proposed:

- **Phase 1: simulation.** Instead of using a real network of OpenMote nodes, sensor information is generated using OpenWSN's simulator, OpenSim, on the Raspberry Pi. This allows for the validation of the computing model before the final deployment is performed, since the source of the data does not affect the performance indicators used for the comparison of paradigms (edge versus cloud computing).
- **Phase 2: physical deployment.** in case that the simulation implementation is successful, and valid conclusions can be drawn from it, the paradigms can be further validated using real sensor data, collected by OpenMote devices. In addition, this provides a deeper insight into IoT technologies, since it is necessary to set up, program and configure the sensor data collection and the connectivity using the 802.15.4 protocol.

The presented design allows to undertake the study of the two proposed computation models, edge and cloud, since the role of its different components, specially the Raspberry Pi, will depend on how applications are implemented in the system. This flexibility comes from the fact that the applications running on the mote and running on the Raspberry Pi are programmable using common languages, namely C and Python, so that the control over their behavior can be fine-grained.

3.2 System processes

The purpose of this section is presenting the functions and processes that have been described as part of the application. this section is presenting how the different process which compose the computation models under study have been designed.

The design is presented in the form of flow diagrams, which describe the sequence of operations which are performed. It is important to note that their

purpose is not to present low-level details of how the application is designed. Such details can be consulted in the provided source code.

3.2.1 Cloud computing model

The design of the cloud computing component of the simulation has the objective of presenting the classical features associated to cloud computing Systems. Consequently, no data processing will be undertaken on the Raspberry Pi, which will simply act as a gateway, collecting data from different motes and sending it to the cloud platform, thethings.iO.

The following processes have been identified.

3.2.1.1 Retrieve and send to the cloud sensor data from motes

The information collected by sensors is retrieved from the motes by the gateway Raspberry Pi using the CoAP protocol. The following types of data is considered:

- *Temperature*: indicates room temperature in degrees Celsius humidity level in the air.
- *Humidity*: indicates relativity humidity level in the air expressed as a percentage.
- *Presence*: indicates whether a presence has been detected in the room (1) or not (0).

Once the sensor readings are received, their format is adapted to make it compatible with thethings.iO. Once the format is adapted, each reading is added to a JSON object, where the key indicates the type of data and the mote to which it corresponds, and the value is the value read from the sensors.

When all data has been collected, the JSON object is sent to the cloud platform.

Figure 11 and Figure 12 depict the flow diagrams for this process.

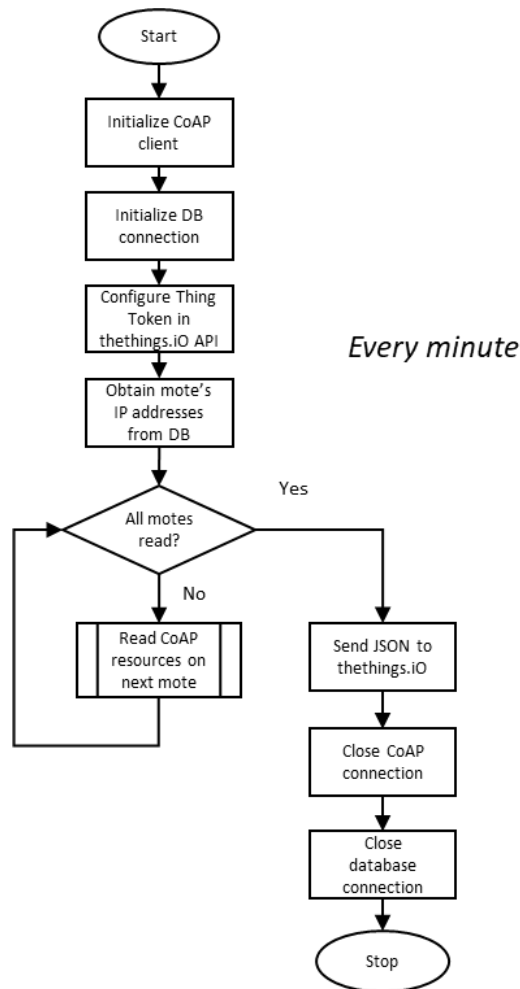


Figure 11: flow diagram which describes how the process of retrieving data from sensors and sending it to the cloud is performed

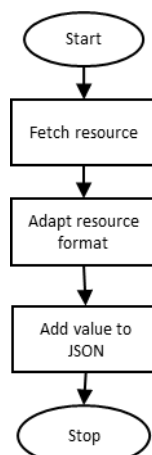


Figure 12: flow diagram which describes how the sub-process of reading a value from a mote is performed.

3.2.1.2 Analyze stored information to compute relevant statistical metrics

The data collected from the sensors, which has been stored in the cloud platform, is further analyzed using tools provided by thethings.io. In this case, it is not

required to have continuous updates on the computed statistics. Instead, the analysis is performed with a period of 1 hour. In this manner, it becomes possible to create statistical profiles with a step size of one hour.

For example, this could be useful to plot the average temperature in the house for a 24-hour period, showing the average temperature for each hour.

The following statistics are calculated for the *temperature* and *humidity* sensor data types:

- Average value
- Maximum value
- Minimum value

3.2.1.3 Data representation

One of the key features of most cloud platforms is the possibility of displaying stored data in different formats. For this reason, the readings obtained from the home deployment will be plotted using line diagrams, which show their evolution as a function of time.

3.2.1.4 Intruder detection

Another aspect of the cloud computing model which has been applied to the design is the possibility of actuating on the monitored system based on its status. In particular, a naïve intruder detection mechanism is proposed. If presence is detected in the entrance hall between 00:00 and 06:00, a log message is to be registered in *thethings.iO*, and an email message is sent informing the owner about the intruder detection.

The flow diagram for this process is shown in Figure 13.

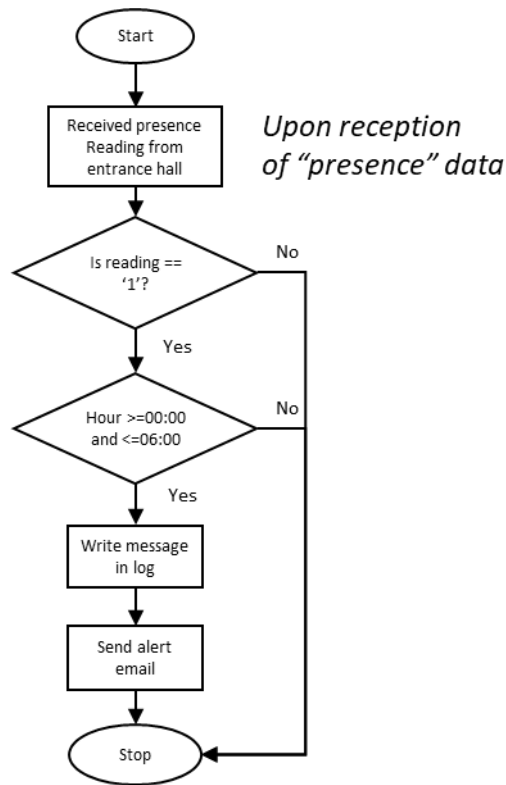


Figure 13: flow diagram showing the process of detecting an intruder

3.2.2 Edge computing model

The design of the cloud computing component of the simulation aims to highlight the main differences between classic cloud computing and edge computing model. For this reason, a significant amount of the computation tasks will be done directly on the Raspberry Pi. However, some information will still be sent to the cloud, which will act as a central point for monitoring the summarized information provided by the edge computations.

To implement the edge computing model, the following processes have been identified.

3.2.2.1 Retrieve and store sensor data from motes

The information collected by sensors is retrieved from the motes by the gateway Raspberry Pi using the CoAP protocol. The same data sources as in the cloud computing model (see 3.2.1) are defined, with the addition of:

- *Air conditioning status*: indicates whether the air condition system is activated (1) or not (0).

Once sensor data has been fetched, its format is adjusted so that it can it is suitable for storage. After the adaptation has taken place, the read values are stored internally in the Raspberry Pi, using the PostgreSQL database created for such purpose. The following information regarding the read value is stored:

- Date

- Type of reading (temperature, humidity or presence)
- Mote from where the value was read.

Consequently, a single table will contain all readings from different motes and resource types. Further analysis will read values from this table, potentially filtering by date, mote or resource type.

This activity needs to be frequent enough to provide a realistic view of how the different environmental conditions in the house evolve with time. As a consequence, a polling period of *3 minutes* has been defined. It is important to notice that the data collected at this rate will not be sent to the cloud. Instead, it will only be stored internally in the Raspberry Pi.

The flow diagrams for this activity is presented in Figure 14 and Figure 15.

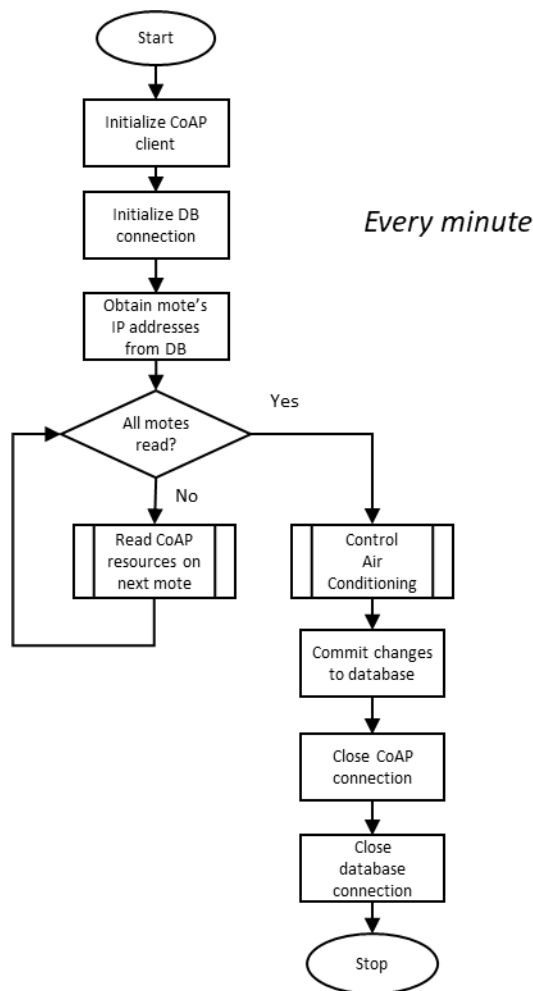


Figure 14: flow diagram which describes how information is read from the motes in the Edge Computing model.

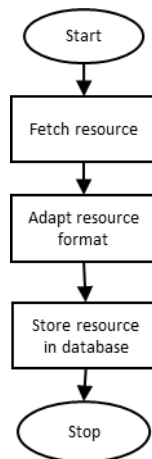


Figure 15: flow diagram which describes the sub-process of reading a resource value from a mote.

3.2.2.2 Analyze stored information to compute relevant statistical metrics

The data collected from the sensors is further analyzed by calculating different metrics for a defined period. As in the Cloud Computing model, it is not required to have continuous updates on the computed statistics.

The following statistics are calculated for the *temperature* and *humidity* sensor data types:

- Average value
- Maximum value
- Minimum value
- Standard deviation

With regard to the *presence*, a different approach has been undertaken. In this case, the statistical analysis attempts to provide information that allows to create occupancy models of the house. This information can prove useful to determine how citizens make use of their home, and how their behavior and behavior patterns can influence aspects such as the temperature of the room. In this regard, three indicators have been included:

- Least active room: room where presence has been detected the lowest amount of times during the last hour.
- Most active room: room where presence has been detected the highest amount of times during the last hour.
- Number of people at home: maximum number of rooms where presence has been detected at the same time during the last hour.

As it can be appreciated, these indicators are simple to calculate. Although more complex computations might be developed, these three simple indicators have been deemed appropriate for the scope of this dissertation, as they represent an example of the type of information that can be extracted using sensors in a connected home.

Unlike the basic resource value reading, this activity does not need to be executed every minute, since that would not provide any significant information. Instead, a period of *1 hour* has been defined for these calculations, since it presents a good balance between producing data frequently enough and covering a period that can be used to analyze the evolution of values during the day.

Flow diagrams for this activity are presented in Figure 16 and Figure 17.

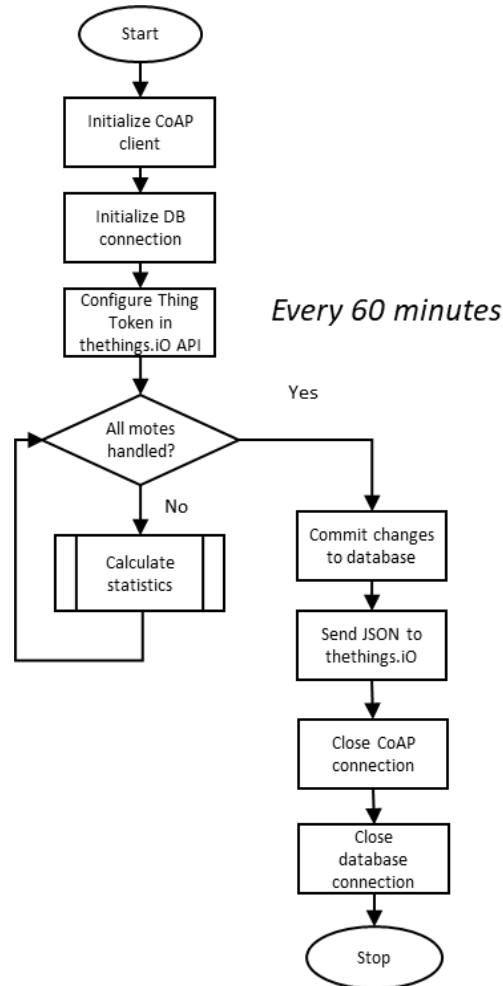


Figure 16: flow diagram which describes how statistical analysis is performed on the stored data.

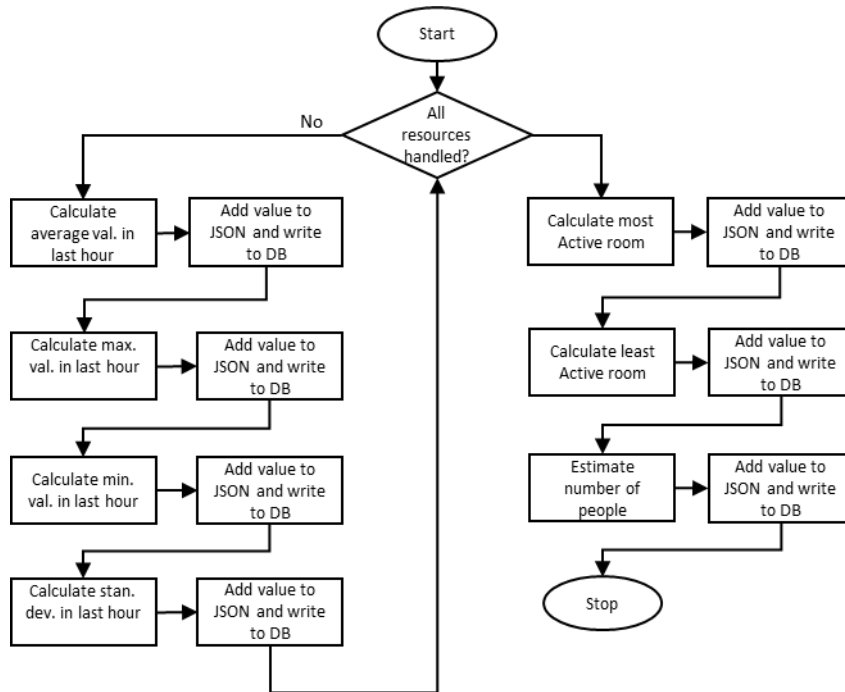


Figure 17: flow diagram which describes the sub-process of calculating different statistical metrics on the stored data.

3.2.2.3 Actuate over the air conditioning system based on the temperature

As an example of an actuation mechanism based on read data, the edge computing application is designed to actuate on the air conditioning system based on the temperature of the Living Room.

The flow diagrams for this activity is presented in Figure 18.

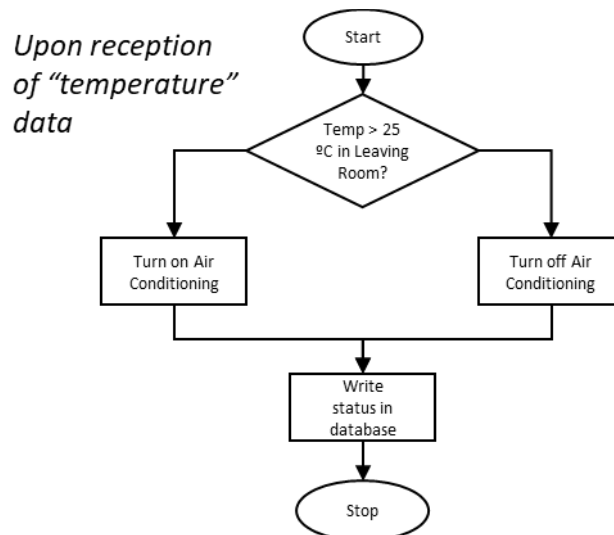


Figure 18: flow diagram which describes how the air conditioning system is actuated based on the temperature.

3.2.2.4 Data cleanup

In order to avoid excessive data storage that could potentially exhaust the storage capacity of the Raspberry Pi, a database cleanup operation is scheduled every 12 hours, cleaning entries in the table older than 12 hours, since they are considered unnecessary.

3.2.3 Comparison of designed features

In principle, both solutions should be equal, resulting in the same generated data, e.g. average, standard deviation, etc. However, two factors have influenced the features finally implemented in each model:

1. Greater expertise in the development environment used for the edge model. In particular, previous experience with both Python and PostgreSQL allowed for a more ambitious scope regarding the calculated metrics. On the other hand, the development of the cloud model features occurs in the things.io platform, with Javascript being used as the main development language. Despite the high quality of the documentation, the learning curve for this platform has been steeper than in the case of the edge computing model.
2. The steep learning curve in the case of the cloud computing model is combined with an nonoptimal planning, which has caused a lack of time for developing more features in the cloud computing model.

3.3 Deployment

Considering the processes described in previous sections, the deployment depicted in Figure 19 is proposed. As it can be appreciated, the deployment is composed of 6 motes and 1 Raspberry Pi, which acts as the gateway of the system. Each of the motes have some sensors associated to them, which are listed in Table 6.

With this deployment, it becomes possible to effectively monitor the comfort indicators of the house, since all motes are assumed to have access to temperature, humidity and presence sensors. Moreover, mote number 5 can also control the air conditioning status, which will be used for controlling the temperature based on the measured value in that room.

With regard to the network topology, the mote with ID 1 is considered the DAG root of the network, i.e. the hierarchy will be formed dynamically taking such node as the root. In addition, it has been considered that all of the motes have direct connectivity with each other. This is a safe assumption, since the distance between the nodes is not too long, and there are not any significant obstacles between them, with the exception of walls. In summary, the nodes form a *fully-meshed topology*, thus guaranteeing a high level of connectivity.

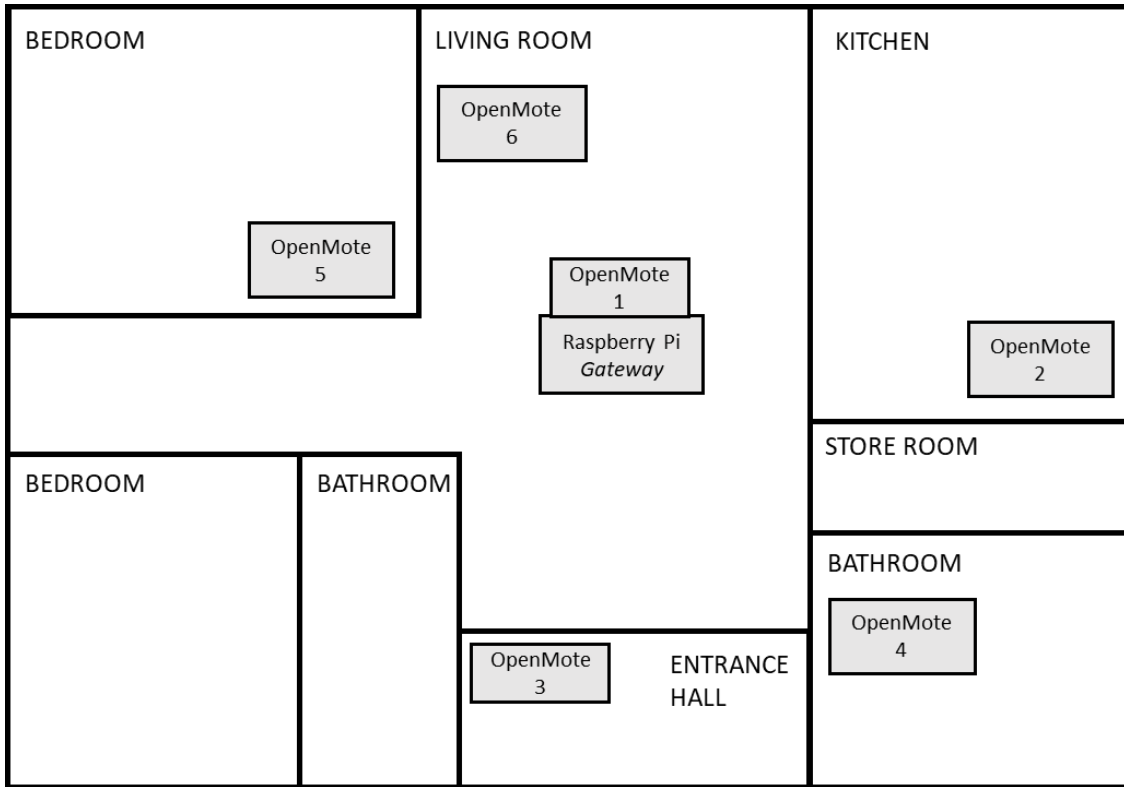


Figure 19: proposed home deployment for the application.

| Mote ID | Location | Sensors |
|--------------|---------------|--|
| 1 (DAG root) | Living Room | None |
| 2 | Kitchen | <ul style="list-style-type: none"> • Temperature • Humidity • Presence |
| 3 | Entrance Hall | <ul style="list-style-type: none"> • Temperature • Humidity • Presence |
| 4 | Bathroom | <ul style="list-style-type: none"> • Temperature • Humidity • Presence |
| 5 | Bedroom | <ul style="list-style-type: none"> • Temperature • Humidity • Presence • Air conditioning status |
| 6 | Living Room | <ul style="list-style-type: none"> • Temperature • Humidity • Presence |

Table 6: summary of deployed motes. All of the 7 motes form a fully-meshed topology.

3.3.1 Summary of collected and generated data

As a summary of the information presented in section 0, Table 7 shows the different data sources that exist in the system, including the frequency at which they are collected. In the same manner, Table 8 presents the results obtained from the computations which take the collected data as input.

| Name | Type | Units | Associated notes | Frequency (minutes) |
|-------------------------|-------------------------|--|------------------|---------------------|
| Temperature | Environmental condition | ° C | 2, 3, 4, 5, 6 | 1 |
| Humidity | Environmental condition | % | 2, 3, 4, 5, 6 | 1 |
| Presence | Environmental condition | Boolean (<i>presence</i> or <i>not presence</i>) | 2, 3, 4, 5, 6 | 1 |
| Air conditioning status | Device status | Boolean (<i>active</i> or <i>not active</i>) | 5 | 1 |

Table 7: summary of data sources which are collected from the simulated nodes

| Name | Description | Associated sources | data | Frequency (minutes) |
|--------------------------|---|---------------------------------|------|---------------------|
| Average values | Average value of collected values during a period of 1 hour | Temperature, humidity, presence | | 60 |
| Maximum values | Maximum value of collected values during a period of 1 hour | Temperature, humidity, presence | | 60 |
| Std. deviation of sample | Standard deviation of the collected values during a period of 1 hour | Temperature, humidity, presence | | 60 |
| Least active room | Room where presence has been detected the lowest amount of times during the last hour | Presence | | 60 |
| Most active room | Room where presence has been detected the highest amount of times during the last hour | Presence | | 60 |
| Number of people at home | Maximum number of rooms where presence has been detected at the same time during the last hour. | Presence | | 60 |

Table 8: summary of generated results

3.4 Database model

A key component of any application is how is information stored and accessed. For this reason, the database model which has been designed for this project is presented. Such model describes what tables exist, what is their function, what data do they store and how do they relate to each other.

An important consideration is that, except for the data types, the database model is framework-agnostic, being it possible to implement it in any of the available database systems (PostgreSQL in this case). The only requirement would be to find the equivalent data type in the corresponding database system. For example, the `cidr` data type in PostgreSQL would become `VARBINARY(16)` in MariaDB.

The code for implementing the database model in PostgreSQL can be found in

3.4.1 Tables

3.4.1.1 MOTES

Table to register the mote devices which exist on the network.

| Column name | Description | Type | Can be Null? |
|-------------|---|-------------|--------------|
| ID | Unique numeric identifier of the mote, e.g. 1, 2. etc. | INT | NO |
| NAME | Arbitrary name to identify the mote. It usually corresponds to the location where the mote is placed, e.g. 'Kitchen'. | varchar(80) | NO |
| ADDRESS | IPv6 address of the mote | cidr | YES |

Table 9: Database table "MOTES"

3.4.1.2 SOURCE_TYPES

Table to identify which type of data sources exists in the system. It typically corresponds to the types of sensors associated to the motes.

| Column name | Description | Type | Can be Null? |
|-------------|---|-------------|--------------|
| NAME | Unique identifier of the data source, e.g. 'temperature'. | varchar(80) | NO |
| UNITS | Units in which the data source is expressed | varchar(80) | YES |

Table 10: Database table "SOURCE_TYPES"

3.4.1.3 READINGS

Table to store read values from the different data sources of the deployment.

| Column name | Description | Type | Can be Null? |
|-------------|---|-------------|--------------|
| ID | Identifier of the mote device which performs the reading. It must be one of the identifiers defined in the 'MOTE' table | int | NO |
| DATE | Date and time on which the reading was performed | timestamp | NO |
| TYPE | Type of data source (e.g. temperature). It must correspond to one the entries in the SOURCE_TYPES table | varchar(80) | NO |
| VALUE | Numeric value of the reading | int | NO |

Table 11: Database table "SOURCE_TYPES"

3.4.1.4 TEMPERATURE

Table to store statistics regarding temperature.

| Column name | Description | Type | Can be Null? |
|------------------|---|-----------|--------------|
| ID | Identifier of the mote device which performs the reading. It must be one of the identifiers defined in the 'MOTE' table | int | NO |
| TIME_RANGE_START | Initial date and time of the range where the statistic has been calculated | timestamp | NO |
| TIME_RANGE_STOP | Final date and time of the range where the statistic has been calculated | timestamp | NO |
| AVERAGE | Average value in the period under study | real | YES |
| MIN | Minimum value in the period under study | real | YES |
| MAX | Max value in the period under study | real | YES |
| STDDEV | Standard deviation of the sample in the period under study | real | YES |

Table 12: Database table "SOURCE_TYPES"

3.4.1.5 HUMIDITY

Table to store statistics regarding humidity.

| Column name | Description | Type | Can be Null? |
|------------------|---|-----------|--------------|
| ID | Identifier of the mote device which performs the reading. It must be one of the identifiers defined in the 'MOTE' table | int | NO |
| TIME_RANGE_START | Initial date and time of the range where the statistic has been calculated | timestamp | NO |
| TIME_RANGE_STOP | Final date and time of the range where the statistic has been calculated | timestamp | NO |
| AVERAGE | Average value in the period under study | real | YES |
| MIN | Minimum value in the period under study | real | YES |
| MAX | Max value in the period under study | real | YES |
| STDDEV | Standard deviation of the sample in the period under study | real | YES |

Table 13: Database table "SOURCE_TYPES"

3.4.1.6 PRESENCE

Table to store statistics regarding home occupancy.

| Column name | Description | Type | Can be Null? |
|-------------------|---|-----------|--------------|
| TIME_RANGE_START | Initial date and time of the range where the statistic has been calculated | timestamp | NO |
| TIME_RANGE_STOP | Final date and time of the range where the statistic has been calculated | timestamp | NO |
| LEAST_ACTIVE_ROOM | Room where presence has been detected the smallest amount of times | real | YES |
| MOST_ACTIVE_ROOM | Room where presence has been detected the biggest amount of times | real | YES |
| N_PEOPLE_HOME | Estimated number of people at home, calculated as the number of rooms where presence has been detected at the same time | real | YES |

Table 14: Database table "SOURCE_TYPES"

3.4.2 Relationship between tables

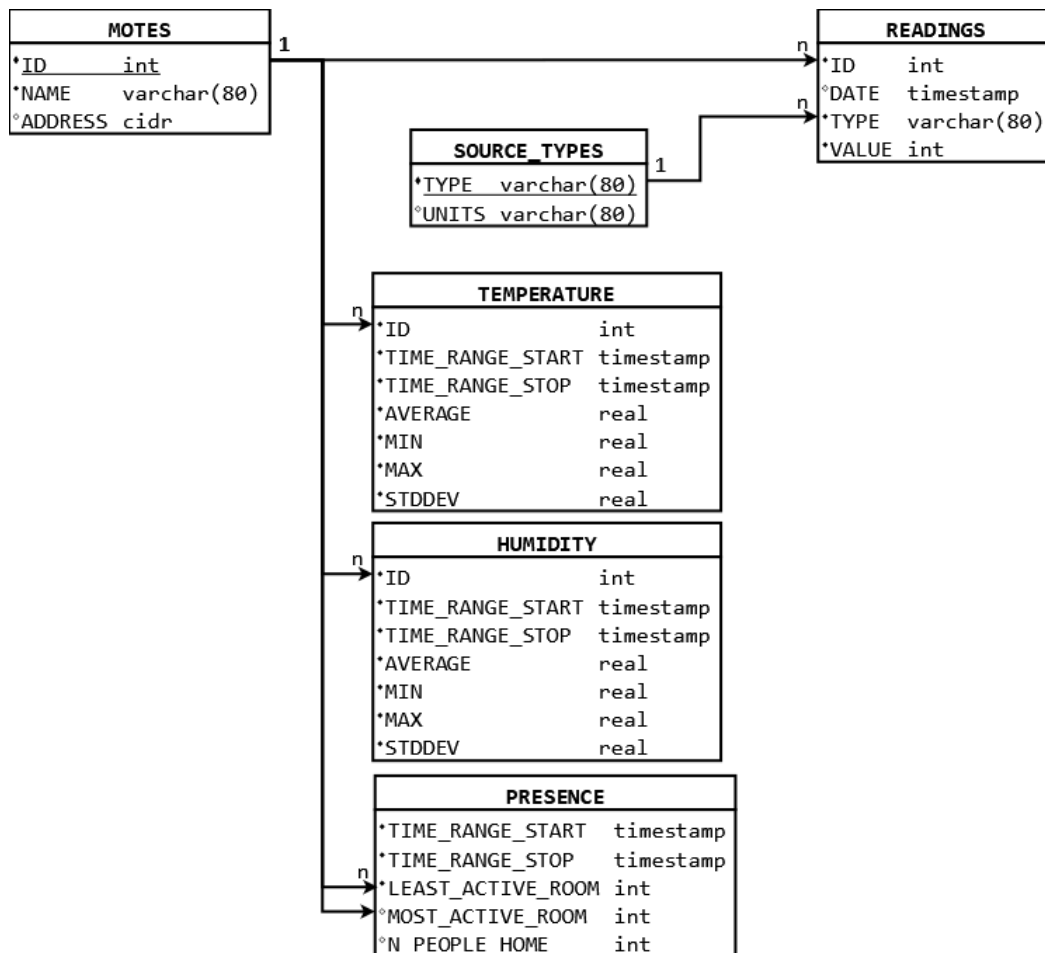


Figure 20: Relationship between tables in the database model

The relationship between tables of the database model is presented in Figure 20. As it can be observed, the following foreign keys have been defined:

- **ID**: whenever a reference to a mote is presented in a column, such column references the ID column of table “MOTES”. In this manner, data integrity in the database is ensured, since it becomes impossible to store readings or data that correspond to an undefined mote. The 1 to N relationship indicates that motes cannot share ID in the MOTES table, but multiplies entries referring to the same ID can be stored in other tables.
- **SOURCE_TYPE**: when indicating the type of resource in the READINGS table, it must correspond to one of the types defined in the SOURCE_TYPE table. This ensures that all data stored in the READINGS table correspond to known types of resources. The 1 to N relationship indicates that two different types cannot have the same name in the SOURCE_TYPES table, but multiplies entries referring to the same source type can be stored in other tables.

In addition, Figure 20 also contains information regarding data types in the database and a flag indicating whether the column can be null or not.

4 Implementation

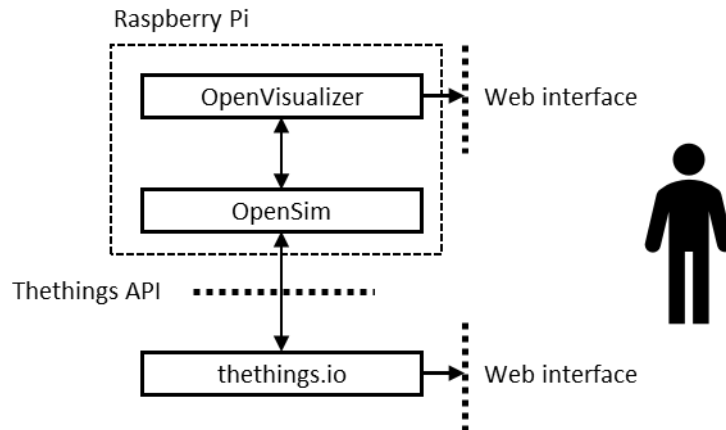


Figure 21: High level overview of the implementation

In order to provide a high-level overview of the system, Figure 21 is presented. As it can be appreciated, three major levels can be identified:

- **OpenSim**: emulate motes to generate data and inject to the event bus.
- **OpenVisualizer**: provides a management and monitoring interface for the simulated motes. It provides several user interfaces, namely native GUI, CLI and a web interface. The later will be used for the purpose of this project.
- **thethings.io**: cloud platform which will store and display sensor data, as well as the resulting computations.

In this section, the implementation of the design proposed in section 0 is presented. Firstly, the discussion will focus on aspects of the implementation which apply to both models. Secondly, the development specifics of each model will be described.

4.1 Simulation of motes

In the first stage of the process, the OpenMote devices are simulated, allowing to validate the design and study the models without requiring complex hardware setup. For this purpose, OpenSim, the simulation framework of OpenWSN, in combination with OpenVis, will be used. The undertaken approach has the additional advantage of being hardware-agnostic, being it possible to reuse the applications developed for the simulation, which shall behave in the same manner whether it is run as a simulation or flashed on a real device.

It is assumed that OpenVis is already installed and configured (see section 10.3). The basic invocation of the OpenSim can be achieved with the following command.

```
$ cd openwsn-sw/software/openvisualizer
$ sudo scons runweb --sim
```

Once the simulation is loaded, it can be accessed from a web browser through port 8080. This will show the result shown in Figure 22. The panel on the left allows the user to configure several aspects of the simulation.

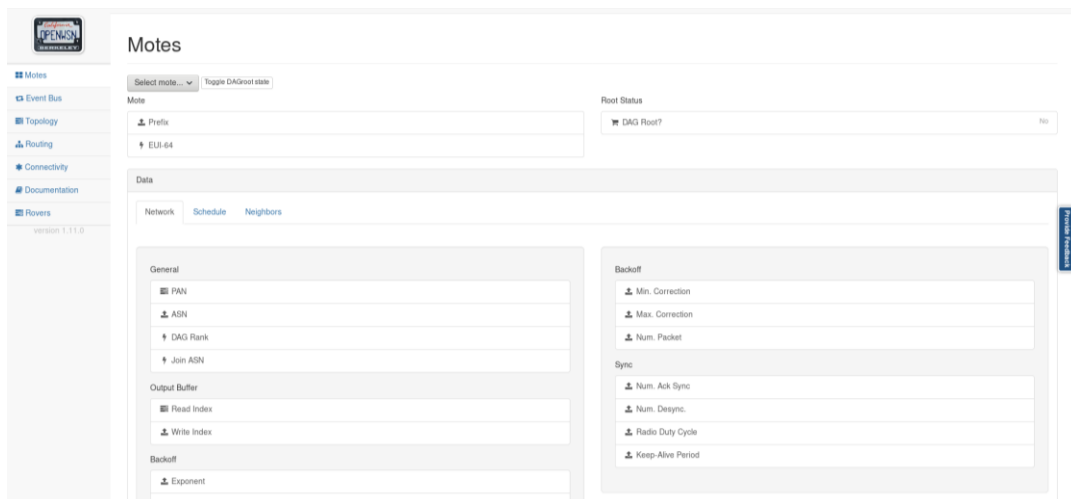


Figure 22: Mote overview

To further configure how the simulation is run, the following options can be used:

- **simCount**: number of simulated motes.
- **simTopology**: topology which the simulated motes form. Possible values are:
 - *linear*: each mote connects to 2 other motes at most.
 - *fully-meshed*: each mote connects to all other motes.

The selected configuration for the implementation of the design has been assigning a value of 6 to `simCount` and use `simTopology` fully-meshed. Therefore, the invocation script becomes:

```
cd openwsn-sw/software/openvisualizer
$ sudo scons runweb --simCount=6 --simTopology=fully-meshed
```

The reason for selecting a fully-meshed topology is that it guarantees the connectivity of all devices. If any of them malfunctions, the RPL algorithm will be able to recalculate the root with the remaining nodes.

4.1.1 Web interface

This section provides a description of the web interface and the configuration options that it offers.

4.1.1.1 Motes

This page contains information regarding each of the motes which compose the simulation. Some of the most relevant pieces of information that can be consulted are:

- Whether the mote is DAG root or not.

- IPv6 address of the mote.
- Status indicators of the mote.

From this page, it is also possible to turn a mote into the DAG (Directed Acyclic Graph) root of the network. In the context of the RPL protocol [18], the DAG root node is the only one that does not have outgoing edge. This means that it acts as the centralizing point of the network, as other motes form a connectivity hierarchy with the DAG root in the upper level.

4.1.1.2 Event bus

It contains events that have been produced by the devices which take part in the simulation. Thanks to the shared Event Bus, motes in the network can communicate by means of a publish-subscribe messaging mechanism. An advantage of this approach is that it is hardware-agnostic, being it possible to communicate real and simulated motes [42].

4.1.1.3 Topology

The topology of the network can be configured in a visual manner, as displayed in Figure 23. A map is displayed, showing the existing motes and their location (by default the UC Berkeley building for Electrical Engineering), as well as the connections between them.

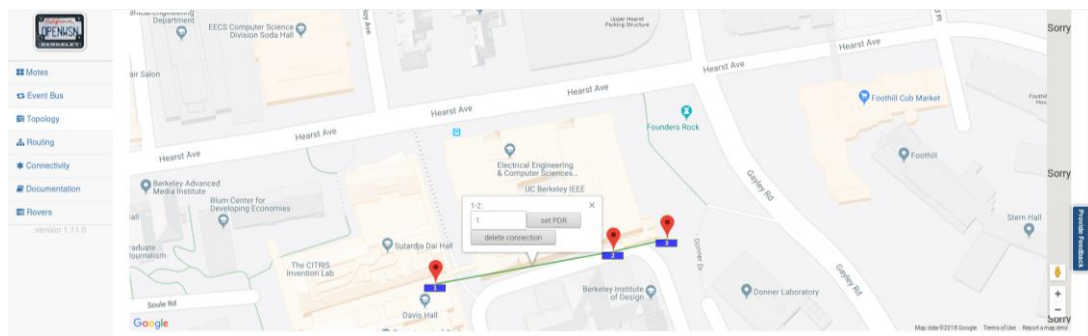


Figure 23: Location and topology of motes

The quality of the connection between motes can be configured by means of controlling its associated PDR, which indicates the reliability of a link, calculated as the ratio between the number of received packets and the number of sent packets. Consequently, a PDR of 1 indicates a perfect link [43].

For the purpose of this project, a PDR of 1 will be assumed between all motes.

4.1.1.4 Routing

It contains a graph showing the DAG of the RPL protocol. After setting the root, RPL DAO messages are sent between the motes in order to form the DAG, indicating the hierarchy between nodes. These messages are sent periodically to keep the routing functional (e.g. if a node goes down).

```

received RPL DAO from bbbb:0:0:0:1415:92cc:0:2
- parents:
  bbbb:0:0:0:1415:92cc:0:1
- children:

```

4.1.1.5 Connectivity

Shows which motes are reachable and which are not. Additionally, the connectivity with the simulated OpenMote devices can be tested with the ping command, sending to the IPv6 associated to the mote.

```

$ ping -s 10 bbbb:0:0:0:1415:92cc:0:2
PING bbbb:0:0:0:1415:92cc:0:2(bbbb::1415:92cc:0:2) 10 data bytes
18 bytes from bbbb::1415:92cc:0:2: icmp_seq=1 ttl=64
18 bytes from bbbb::1415:92cc:0:2: icmp_seq=2 ttl=64

$ ping -s 10 bbbb:0:0:0:1415:92cc:0:3
PING bbbb:0:0:0:1415:92cc:0:3(bbbb::1415:92cc:0:3) 10 data bytes
18 bytes from bbbb::1415:92cc:0:3: icmp_seq=1 ttl=64
18 bytes from bbbb::1415:92cc:0:3: icmp_seq=4 ttl=64

```

4.1.1.6 Documentation

External link's to OpenSim's documentation.

4.2 Developing OpenWSN applications

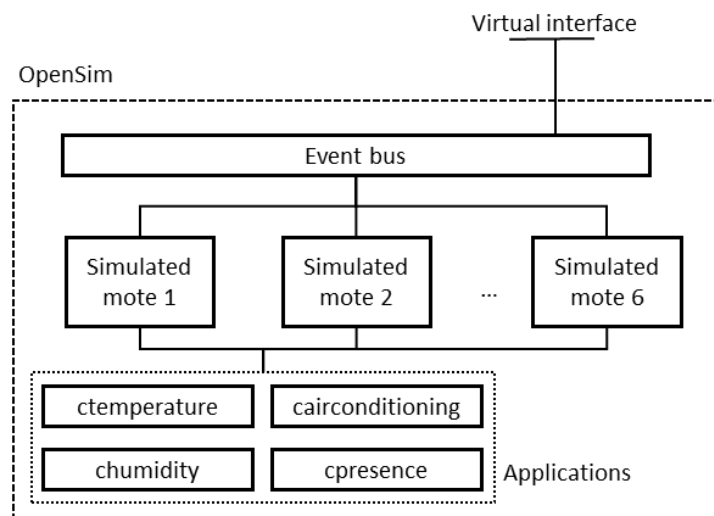


Figure 24: Overview of OpenSim and its components. Adapted from OpenWSN documentation.

As it can be appreciated in Figure 24, each simulated mote has access to different applications. In this case, such applications are to be used for simulating sensor data, i.e. temperature, humidity, presence and the status of the air conditioning system. Once implemented, each application is initialized and creates a new

resource that can be accessed by its URI using CoAP, thanks to the fact that a virtual interface is exposed to the Raspberry Pi, allowing communication with OpenSim.

While some applications are already provided by the `openwsn-fw` repository, it is possible to develop new ones tailored for the specific purposes of the project. The implemented applications, written in C, will work as sensor simulators, providing random values for the different readings. Developed applications are listed below:

- **ctemperature**: simulation of temperature.
 - A random number between +5 and -5 is added to the nominal temperature, which has a value of 23 degrees Celsius during the day (08:00 AM - 22:00 PM) and 18 degrees Celsius during the night.
- **chumidity**: simulation of humidity.
 - A random number between +5 and -5 is added to the nominal humidity level, which has a value of 45 % during the day (08:00 AM - 22:00 PM) and 55 % during the night.
- **cpresence**: simulation of presence.
 - The probability of detecting a presence (1) is 75 % during the day (08:00 AM - 22:00 PM) and 25 % during the night.
- **cairconditioning**: status of the air conditioning system.
 - A value of 1 indicates that the air conditioning system is enabled, while a value of 0 indicates that it is disabled. It is possible to modify this value using CoAP.

As in any other software development, it is highly recommended to use version control for keeping track of the evolution of the code. Since the `openwsn-fw` is hosted as a git project, this will be the used tool.

```
$ git checkout -b feature_to_be_developed  
Checkout out new branch 'feature_to_be_developed'
```

The codification of the applications has been based on existing apps. In particular, the `cinfo.c` application, can be used as the base of the development, since they already contain all the required elements for sending a desired value upon a CoAP request.

4.2.1 C Implementation

Instead of adding the code, it has been considered more effective to include an explanation of how the implementation has been performed. In the case of the temperature, humidity and presence sensors, the implementation is rather similar, and the illustrated concepts will apply to all of them. In the case of the status of the air conditioning system, the code is also very similar, but since the application can be controlled via CoAP, the additional code for handling CoAP requests will be explained.

For the sake of brevity, only the most relevant aspects of the code will be explained.

4.2.1.1 Initialization

Function `ctemperature_init()` configures some important parameters of the sensor. They are stored in a struct which is used for the registration of the application (`opencoap_register`). The most important are:

- **path**: URI path which identifies the “temperature” resource. CoAP GET methods will use such URI to select which value is to be read.
- **componentID**: identifier of the component within the openwsn-fw environment. It is configured in `inc/opendefs.h` and it must be unique.
- **callbackRx**: callback for handling CoAP requests.

4.2.1.2 Reception callback

It is in charge of handling CoAP requests. Different actions are performed depending on the CoAP code contained in the `coap_header`. The GET and PUT methods are the most commonly used:

- **GET**: firstly, the message payload is reset to allow for the storage of the new value to be sent. Then, space is reserved for the CoAP header. Afterwards, the value to be sent is transform to a character array and inserted into the payload (`payload[0]`, `payload[1]`, etc.). Finally, the CoAP code is set as “RESP_CONTENT” in the CoAP header.
- **PUT**: the contents of the payload are inspected as char variables. Depending on the value received, the global internal variables used for storage are updated after converting them from their char representation.

```
owerror_t cpresence_receive( OpenQueueEntry_t* msg,
    coap_header_iht* coap_header,
    coap_option_iht* coap_incomingOptions,
    coap_option_iht* coap_outgoingOptions,
    uint8_t* coap_outgoingOptionsLen)
{
    owerror_t outcome;

    switch (coap_header->Code) {
        case COAP_CODE_REQ_GET:
            // AMM: we are receiving a GET request
            // Reset packet payload so that packetBuffer can be reused
            msg->payload = &(msg->packet[127]);
            msg->length = 0;

            // AMM: Reserve space for the string which contains the presence.
            packetfunctions_reserveHeaderSize(msg, PRESENCE_N_DIGITS);

            presenceReading = read_presence();
            snprintf(presenceReadingStr, PRESENCE_STR_LEN, "%d", presenceReading);

            msg->payload[0] = presenceReadingStr[0];

            coap_header->Code = COAP_CODE_RESP_CONTENT;
            outcome = E_SUCCESS;

            break;
        default:
            // AMM: unkown CoAP code, return an error message
            outcome = E_FAIL;
    }
}
```



```

    }
    return outcome;
}

```

4.2.2 Compilation

After the source files have been created, it is necessary to modify the Python-simulated firmware in OpenSim in order to make it aware of the new application. The process for doing so is described in [44], and it requires the modification of the following files:

- **inc/opendefs.h**: add an identifier for the new component. This uniquely identifies the application in the firmware.
- **openapps/SConscript**: add the path to the new application with `path.join` so that source files can be found.
- **openapps/openapps.c**: indicate that the new application needs to be initiated, e.g. calling `ctemperature_init()`;
- **projects/python/Sconscript.env**: all the functions defined in the new application need to be added to the list of `functionsToChange`, in order for them to be objectified. Otherwise, compilation errors will appear.

4.2.3 Implementation of computation models

The implementation of the proposed computation models will take the form of Python scripts running on the Raspberry Pi. Their function will be communicating with the sensors simulated in OpenWSN, perform some operations on the read data, and then either store it locally in a database or send it to the cloud (or both in some cases).

The communication with a running application is possible thanks to the CoAP client implementation provided by Berkeley, which can be imported into the Python scripts to communicate with the sensors in an easy manner. Documentation about such library can be found in [45]. As it can be appreciated in Figure 25, a CoAP request is sent to the mote device to obtain the temperature reading.

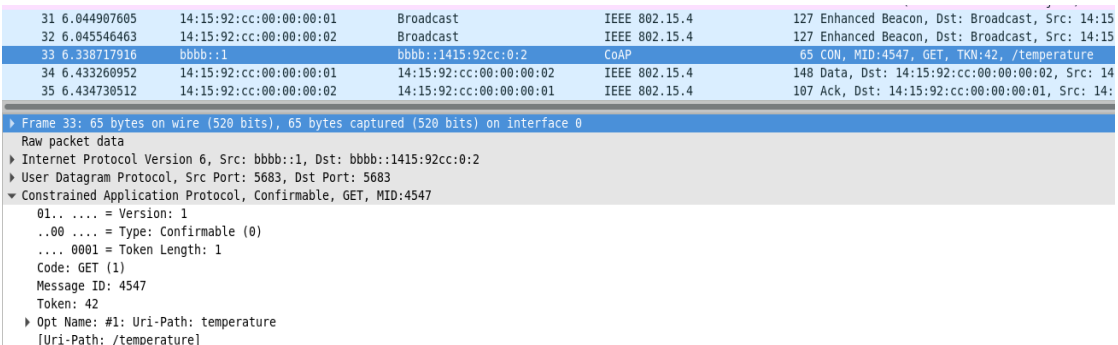


Figure 25: CoAP GET request captured with Wireshark

4.2.4 Communication using CoAP

To connect with a mote, the following steps are required.

1. Import coap package: `from coap import coap`
2. Initiate CoAP client with `c = coap.coap()`
 - a. Use the GET method provided in the CoAP client object. Its only parameter is the URI path to the resource to be read: `<IP_ADDRESS_OF_MOTE> + name of the resource (e.g. "COAP_PATH_TEMPERATURE.format(MOTE_IP2))"`.
3. Adapt the format received from char variables to a single integer variable. This can be done using the `int()` operator in Python.

To facilitate the task of interpreting received data, the functions defined in Table 15 have been implemented, being shared by both edge and cloud applications.

| File | Name | Description |
|-------------------|-------------------------|---|
| readings_utils.py | convert_temperature | Convert a temperature to int from its ASCII representation |
| | convert_humidity | Convert a relative humidity to int from its ASCII representation |
| | convert_presence | Convert a presence value (1 or 0) to int from its ASCII representation |
| | convert_airconditioning | Convert air conditioning status (1 or 0) to int from its ASCII representation |

Table 15: readings_utils.py

4.2.5 Interfacing with PostgreSQL from Python

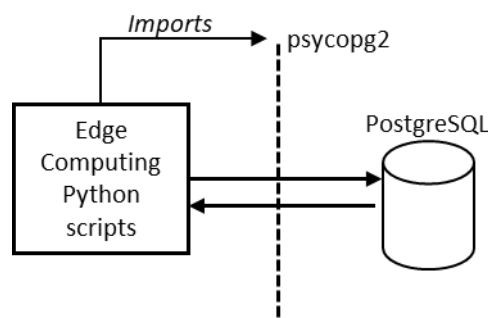


Figure 26: Interaction between Python scripts and the PostgreSQL database

Due to the fact that Python is the selected language for retrieving data from the simulators, it is required to use a library which allows communicating with PostgreSQL from Python.

The most popular PostgreSQL Python library is psycopg2 [46]. It provides functions to easily connect and interact with PostgreSQL database. Once it has been installed, it is enough with importing the psycopg2 package in the Python script.

1. Import psycopg2 package
2. Create a database connection, indicating the database name and the user: `psycopg2.connect("dbname=edge_computing user=pi")`
3. Prepare the query to be performed (e.g. SELECT, INSERT, etc.). The query can contain dynamic values which can be set using local variables at execution time. For example: `db_cursor.execute("SELECT ADDRESS FROM MOTES WHERE ID = '%s'", [id])`. In this case, the ID used for filtering will depend on the value of variable "id".
4. Fetch the result of the query using `fetchone()`. Since more than one column can be retrieved, a tuple is returned, which each value in the tuple corresponding to one column. If more than one row is selected, a list of tuples is returned.
5. Extract the desired value from the tuple and apply any format transformations, if required.
6. Commit changes to the database (only required if there has been insertions) with `conn.commit()`.
7. Close the database connection with `db_cursor.close()`.

To facilitate the task of interfacing with the database, the functions defined in Table 16 have been implemented, being shared by both edge and cloud applications.

| File | Name | Description |
|-------------|----------------|---|
| database.py | insert_reading | Insert an entry into the READING table. The database connection must be already established. |
| | get_mote_ip | Read the IPv6 address of a mote from the database. The database connection must be already established. |

Table 16: database.py

4.2.6 Interfacing with thethings.iO

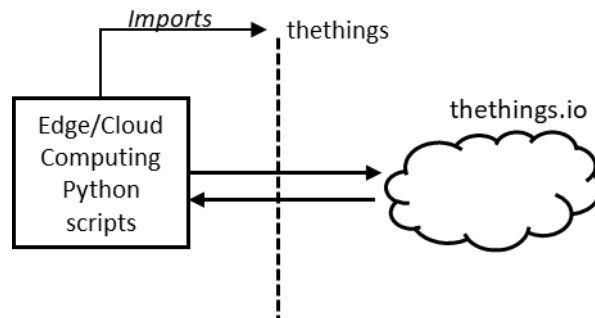


Figure 27: Interaction between Python scripts and the PostgreSQL database

The data collected at the Raspberry Pi needs to be sent to thethings.iO. This can be achieved using thethings.iO API for Python. Once installed, the steps that need to be followed to send data to the platform are:

1. Import the thethings library
2. Create a ThethingsAPI object using the Thing Token which identifies our Raspberry Gateway: `thethings = ThethingsAPI(<THING_TOKEN>)`. In this manner, all information received at the platform will be associated to that particular Thing, becoming available for analysis and representation.
3. Add key-value pairs to the JSON object which will be sent to thethings.iO. The key identifies the resource, while the value indicates the value of such resource at the time of the reading. For example: `thethings.addVar('temperature_mote_2', r.convert_temperature(p))`
4. Send the JSON object using `thethings.write()`. The API automatically handles all the process.

4.2.7 Scheduling of periodic operations

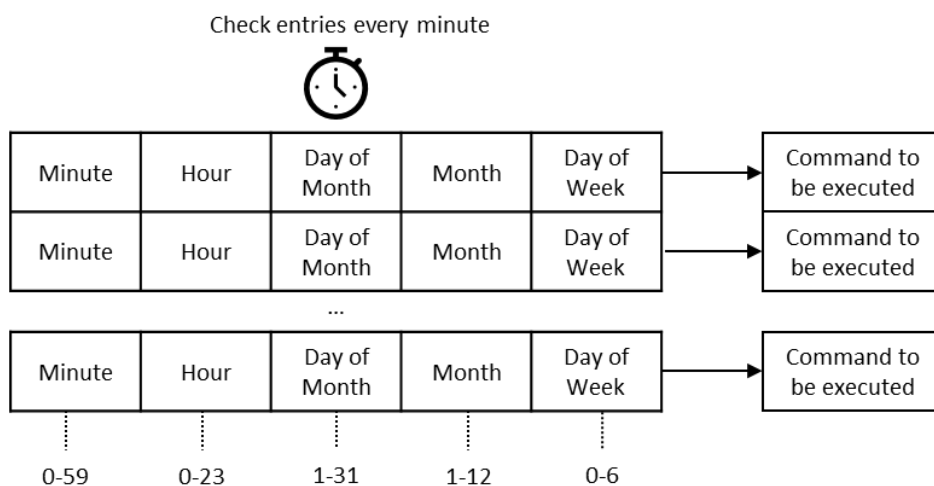


Figure 28: summarized syntax of crontabs

As described in the design sections, the different tasks required for implementing each model are scheduled in a periodic manner. For example, the readings are performed every minute for both the cloud and edge models, while the analysis of statistics in the edge model is performed every hour.

In order to implement this behavior, `cron` has been used. This utility makes it possible to schedule the execution of tasks (jobs) in a computer (such as the Raspberry Pi) with the desired frequency and timing conditions. Such rules are expressed as crontabs (cron tables), which indicate at what times should a given action be executed.

For example, in order to run a job every 5 minutes on all Mondays of every month, from 13:00 PM to 14:00 PM, the following crontab would be used:

```
*/5 13 * * 1 test_command.sh
```

Crontabs can be edited with `crontab -e` and listed with `crontab -e`.

4.3 Cloud Computing implementation

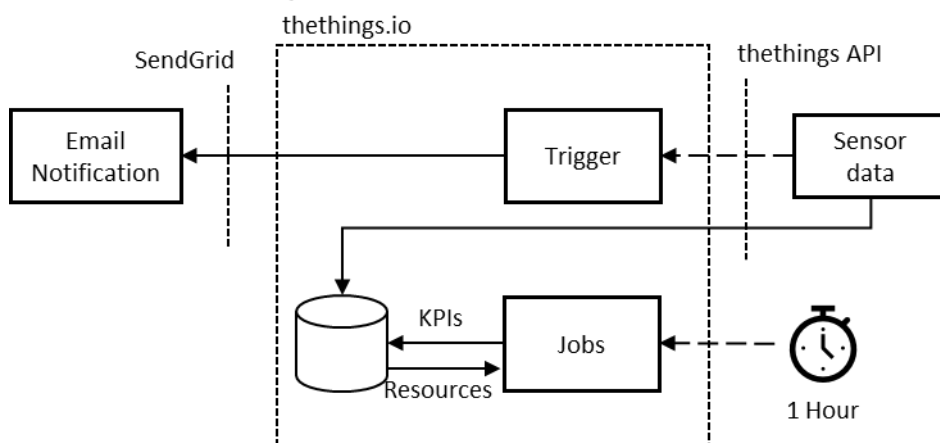


Figure 29: overview of the different parts which compose the cloud computing model. Triggers execute code upon data reception and perform some action as a response (in this case, sending an alert email). Jobs are executed periodically and interact with the internally stored resources to calculate KPIs (key performance indicators).

The implementation of the Cloud computing processes described in 3.2.1 is manifested in Python script `cloud_readings.py`. Its main purpose is reading the sensor values for all motes and sending them to the cloud platform. It simply becomes a matter of using the tools presented in the preceding sections.

The only additional requirements are defining correctly the path to the resources to be fetched and obtaining the IPv6 addresses of all motes. Both steps are listed below:

```
# Read ip addresses from the database
MOTE_IP2 = database.get_mote_ip(cur, 2)
MOTE_IP3 = database.get_mote_ip(cur, 3)
MOTE_IP4 = database.get_mote_ip(cur, 4)
MOTE_IP5 = database.get_mote_ip(cur, 5)

#Path to the temperature resource
COAP_PATH_TEMPERATURE = 'coap://[{}]/temperature'
COAP_PATH_HUMIDITY = 'coap://[{}]/humidity'
```

```
COAP_PATH_PRESENCE      = 'coap://[{}]/presence'  
COAP_PATH_AIRCONDITIONING = 'coap://[{}]/airconditioning'
```

With regard to the periodicity of this tab, it is executed every minute of every hour of every day. Therefore, the crontab used for scheduling this job is:

```
*/1 * * * * /home/pi/openwsn/openwsn-fw/readings/cloud_readings.py
```

4.3.1 Server-side computation

In the cloud computing model, the intelligence of the network is placed in the platform. As it can be seen in Figure 29. There are two different ways in which Thethings.io will be used to process data:

- **Jobs:** snippets of code written in Javascript which are executed every hour or once a day. Execution time is limited to 10 minutes.
- **Triggers:** snippets of code written in Javascript which are executed every hour or once a day. Execution time is limited to 2 seconds.

The analysis of statistics has been implemented using jobs which run every hour. Three types of jobs have been created, depending on whether the average value, the minimum value or the maximum value is calculated. All of them are similar except for the operation applied on the data.

```
function job(params, callback){  
  analytics.events.getValuesByName('temperature_mote_2', function(error,  
data){  
  var max = data.max();  
  analytics.kpis.create('temperature_avg_mote_2-kpi', max);  
  callback();  
});  
}
```

Therefore, a total of 36 jobs will exist, calculating the average, maximum and minimum for the temperature and humidity of every of the 6 motes.

Calculated metrics can be represented in the dashboard using the Custom Metric (Cloud Code) data source, which gives access to the KPIs defined in the jobs. An example of the maximum temperature is shown in Figure 30.

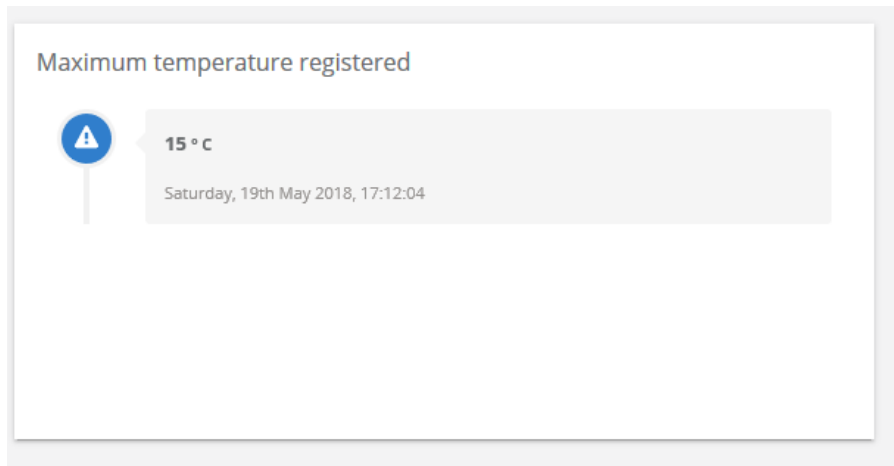


Figure 30: representation of the maximum temperature calculated using a job

With regard to the detection system, it has been implemented as a Trigger. Upon reception of presence readings from mote 3 (entrance), the value is analyzed based on the time of day. If it night (between 00:00 and 06:00 AM) and a presence is detected, an email is sent.

```
function trigger(params, callback) {
  console.log('Starting trigger event...')

  var values = params.values
  var thingToken = params.thingToken

  var date = new Date();
  var current_hour = date.getHours();

  if (current_hour >= 0 && current_hour <= 6) {
    console.log('It is night. Checking if presence has been detected in the entrance
hall.');
```

```
    for(var i=0; i<values.length; ++i) {

      if(values[i].key === 'presence_mote_3' && values[i].value == 1) {
        console.log('Presence detected in the entrance hall.');
```

```
        email(
          {
            service : 'SendGrid',
            auth: {
              api_user: 'artmedmer',
              api_key: 'APIKEY'
            }
          },
          {
            from: 'EMAIL_ADDRESS',
            to: 'EMAIL_ADDRESS_2',
            subject : 'Presence detected in the entrance hall.',
            text : 'An unusual activity has been detected. (Mote: ' +
thingToken + ')'
          }
        )
      }
      else {
        console.log('No presence detected');
      }
    }
  }
  else {
    console.log('It is day. No need to check for intruders.');
```

```
  }
  //end the trigger
  callback()
}
```

Using the email feature requires registering in an external service, SendGrid. However, due to the simplicity of the process and its scarce relationship with the project, this process is not described. An example of a received email is presented in Figure 31.

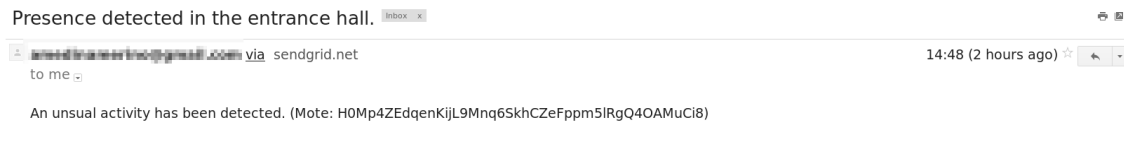


Figure 31: example of received email

4.4 Edge Computing implementation

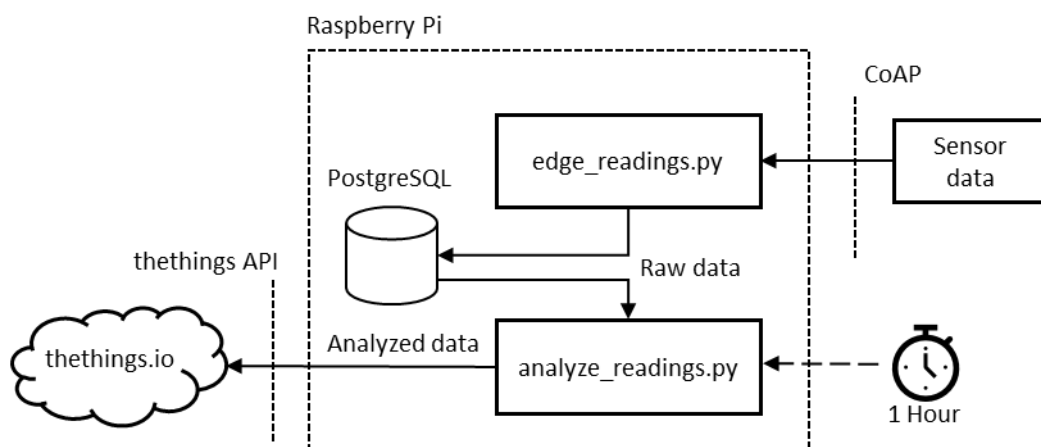


Figure 32: overview of the different parts which compose the edge computing model. `edge_readings.py` fetches data from OpenSim and stores it internally into the Raspberry Pi. On the other hand, the `analyze_readings.py` scripts is executed every hour to extract relevant information for data collected during the previous hour.

The implementation of the Edge computing processes described in 3.2.2 is manifested in Python script **edge_readings.py**. Its main purpose is reading the sensor values for all motes and storing them locally on the database. This script can be very easily implemented using the functions described in previous sections.

With regard to the statistical analysis, it has been implemented in **analyze_readings.py**, it has been possible to use functions natively supported by PostgreSQL. Therefore, the query directly incorporates the type of statistic that is to be calculated. For example, to calculate the average value, AVG can be used.

```
query = "SELECT AVG(VALUE) FROM READINGS WHERE ID = (%s) AND TYPE = (%s) AND (DATE >= (%s) and DATE <= (%s))"
```

For the presence statistics, specific calculations have been required. For this purpose, the `get_room_activity` and `get_n_people` functions have been defined:

```
def get_room_activity(db_cursor, start_date, end_date):
```



```

    """Find the most active and least active rooms, understood as the mote
    (room)
    where presence has been detected the biggest (smallest) amount of
    times"""

    query = "SELECT id, COUNT(*) from READINGS WHERE TYPE = 'presence' AND
    VALUE = '1' AND DATE >= (%s) AND DATE <= (%s) GROUP BY id"
    db_cursor.execute(query, (start_date, end_date))
    count_tuples = db_cursor.fetchall()
    # A list of tuples is returned:
    # (id, number of occurrences of the ID in the table)
    # Extract the second part, the number of occurrences, to see the max and
    the min
    count = [x[1] for x in count_tuples]
    min_val = min(count)
    max_val = max(count)
    min_val_index = count.index(min_val)
    max_val_index = count.index(max_val)
    # The index where the min/max of occurrences occurred can be used to
    identify
    # which mote (room) correspond to the least and most active rooms
    least_active_room = count_tuples[min_val_index][0]
    most_active_room = count_tuples[max_val_index][0]
    return (least_active_room, most_active_room)

def get_n_people_home(db_cursor, start_date, end_date):
    """Estimate the number of people at home, understood as the highest
    number
    of motes (rooms) where presence has been detected at the same time"""

    # Fetch number of motes where presence has been 1 for each reading
    (identified
    # by the date when the reading was performed)
    query = "SELECT DATE, COUNT(*) from READINGS WHERE TYPE = 'presence' AND
    VALUE = '1' AND DATE >= (%s) AND DATE <= (%s) GROUP by DATE"
    db_cursor.execute(query, (start_date, end_date));
    presence_tuples = db_cursor.fetchall()
    # We are only interested in the count value, extract it from the tuple
    presence = [x[1] for x in presence_tuples]
    max_presence = max(presence)
    return max_presence

```

Additional utility functions for calculating statistics are presented in Table 17.

| File | Name | Description |
|---------------------------|-------------------|--|
| | get_statistic | Generic wrapper for calculating any type of statistic |
| | get_avg | Obtain the average value in the last hour |
| | get_stddev | Obtain the standard deviation of the sample in the last hour |
| | get_max | Obtain the maximum value in the last hour |
| edge_statistics.py | get_min | Obtain the minimum value in the last hour |
| | insert_statistics | Insert avg, max and min statistics in the database |
| | get_room_activity | Find the most active and least active rooms, understood as the mote (room) where presence has been detected the biggest (smallest) amount of times |

| | |
|----------------------------|--|
| get_n_people_home | Estimate the number of people at home, understood as the highest number of motes (rooms) where presence has been detected at the same time |
| insert_presence_statistics | Insert presence statistics in the database |

Table 17: edge_statistics.py

Thanks to functions in Table 17, it is possible to calculate statistics in a concise manner.

```
# Calculate statistics of temperature and humidity readings for the last hour,
# for all motes
for source in ['temperature', 'humidity']:
    for mote in [2, 3, 4, 5, 6]:
        avg_val = stat.get_avg(cur, mote, source, start_date, now)
        sd_val = stat.get_stddev(cur, mote, source, start_date, now)
        max_val = stat.get_max(cur, mote, source, start_date, now)
        min_val = stat.get_min(cur, mote, source, start_date, now)
        statistic_name = source + '_avg_' + 'mote_' + str(mote)
        thethings.addVar(statistic_name, float(avg_val))
        statistic_name = source + '_max_' + 'mote_' + str(mote)
        thethings.addVar(statistic_name, float(max_val))
        statistic_name = source + '_min_' + 'mote_' + str(mote)
        thethings.addVar(statistic_name, float(min_val))

[least_active, most_active] = stat.get_room_activity(cur, start_date, now)
n_people_home = stat.get_n_people_home(cur, start_date, now)
stat.insert_presence_statistics(cur, start_date, now, least_active, most_active, n_people_home)
thethings.addVar('least_active_room', least_active)
thethings.addVar('most_active_room', most_active)
thethings.addVar('n_people_home', least_active)
```

Finally, another detail is that the calculation of statistics considers only the last hour of readings. Therefore, the datetime package from Python has been used to get the current hour and calculate the previous hour, using such range in the SQL queries.

```
# Get current date and time. Although some seconds might pass between
readings,
# it is assumed that all are performed at the same time for simplicity
now = datetime.datetime.now()
# To avoid leaving entries without being considered, the microseconds will
# be rounded to zero
now = now.replace(microsecond=0)
# We will read entries from the previous hour, so a time delta of one hour
# is defined
one_hour = datetime.timedelta(hours=1)
start_date = now - one_hour
```

The data cleanup operation consists on a simple DELETE query applied to the last 12 hours of readings.

```
# Get current date and time
now = datetime.datetime.now()
# To avoid leaving entries without being considered, the microseconds will
# be rounded to zero
now = now.replace(microsecond=0)
# Entries older that 16 hours will be removed
twelve_hours = datetime.timedelta(hours=12)
threshold_date = now - twelve_hours
# Delete entries older that 16 hours
query="DELETE FROM READINGS WHERE DATE <= (%s)"
cur.execute(query, [threshold_date])
```

With regard to the periodicity of these tasks, the readings are executed every minute of every hour of every day. Furthermore, the statistics are calculated each hour, and the database cleanup is performed every 12 hours. Therefore, the crontab used for scheduling this job is:

```
*/1 * * * * /home/pi/openwsn/openwsn-fw/readings/edge_readings.py
* */1 * * * /home/pi/openwsn/openwsn-fw/readings/analyze_readings.py
*/12 * * * * /home/pi/openwsn/openwsn-fw/readings/database_cleanup.py
```

Once information about the statistics has been uploaded to thethings.iO, it can be represented, as seen in Figure 39. An example could be a panel that displays the number of people at home in the last hour, as shown in Figure 34.

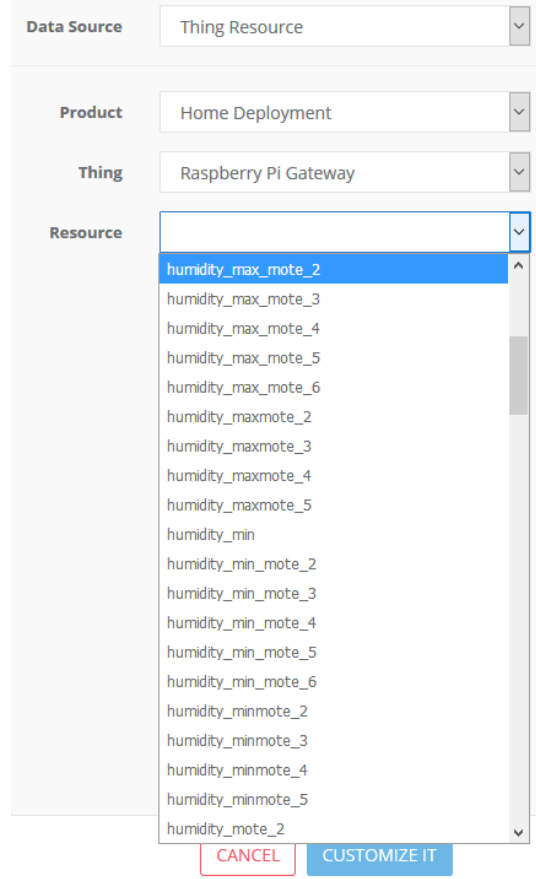


Figure 33: creation of a dashboard for representing a resource calculated on the edge

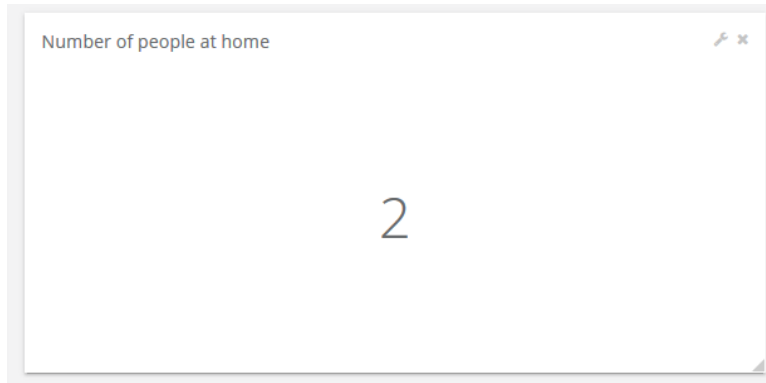


Figure 34: panel displaying number of people at home

5 Comparison of models

5.1.1 Latency

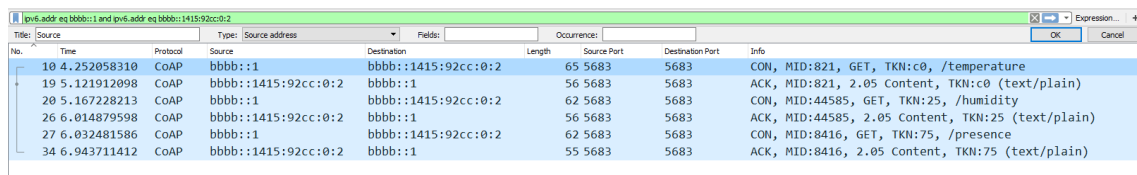
5.1.1.1 Edge computing

In the Edge computing model, latency has been calculated as the sum of consulting the resources using CoAP and writing them to the database.

Due to the simplicity of the application, with only one client writing at any given time, the database is far from being a bottleneck. However, it is still interesting to measure its performance. PostgreSQL provides pgbench for such purpose.

A simple read-write test suite was run for one minute, with a single concurrent client and a single thread. The obtained results were an average latency of 4.063 milliseconds and an average of 199.80 transactions per second. Several iterations of the test returned similar results, with the latency never being bigger than 6 milliseconds and the transactions per seconds never smaller than 180.

In addition, the time taken by CoAP to read or write a resource was measured using tcpdump. As it can be observed in Figure 35, each GET method takes around **1 second** to complete.



| No. | Time | Protocol | Source | Destination | Length | Source Port | Destination Port | Info |
|-----|-------------|----------|---------------------|---------------------|--------|-------------|------------------|---|
| 10 | 4.252058310 | CoAP | bbbb::1 | bbbb::1415:92cc:0:2 | 65 | 5683 | 5683 | CON, MID:821, GET, TKN:c0, /temperature |
| 19 | 5.121912098 | CoAP | bbbb::1415:92cc:0:2 | bbbb::1 | 56 | 5683 | 5683 | ACK, MID:821, 2.05 Content, TKN:c0 (text/plain) |
| 20 | 5.167228213 | CoAP | bbbb::1 | bbbb::1415:92cc:0:2 | 62 | 5683 | 5683 | CON, MID:44585, GET, TKN:25, /humidity |
| 26 | 6.014879598 | CoAP | bbbb::1415:92cc:0:2 | bbbb::1 | 56 | 5683 | 5683 | ACK, MID:44585, 2.05 Content, TKN:25 (text/plain) |
| 27 | 6.032481586 | CoAP | bbbb::1 | bbbb::1415:92cc:0:2 | 62 | 5683 | 5683 | CON, MID:8416, GET, TKN:75, /presence |
| 34 | 6.943711412 | CoAP | bbbb::1415:92cc:0:2 | bbbb::1 | 55 | 5683 | 5683 | ACK, MID:8416, 2.05 Content, TKN:75 (text/plain) |

Figure 35: CoAP messaging for reading resource values.

Consequently, the total time spent on reading a value and storing it in the database is 1 second, with the time taken by PostgreSQL to write on disc having been considered negligible in comparison with CoAP times.

5.1.1.2 Cloud computing

In order to study latency in the Cloud model, the average time required to send data to thethings.iO was considered. *Figure 37* shows the encrypted conversation which occurs between the Raspberry Pi and thethings.iO when sending the values read through the provided Python API. As it can be seen, the transaction takes less than 0.5 seconds to complete. Additionally, it is important to notice that all sensor data is sent using a single call to thethings.iO API, so the time spent on each reading is considerably small.

Nevertheless, a big disadvantage to be considered is the fact that thethings.iO sometimes experiments **severe delays** in its service. *Figure 36* shows a case where the total conversation time was of almost **10 seconds**. Therefore, the suitability of cloud platforms for time-sensitive applications is put at risk.

| IPv4 Conversations | | | | | | |
|--------------------|----------------|----------|-------------|-------------|---------------|-------------|
| Address A | Address B | Duration | Packets A → | Bytes A → B | Packets B → A | Bytes B → A |
| 104.199.77.107 | 192.168.42.122 | 9,2672 | 12 | 4 876 | 14 | 2 376 |
| 192.168.42.122 | 192.168.42.129 | 0,0025 | 2 | 152 | 2 | 244 |

Figure 36: severe communication delay in the communication with thethings.io

However, it must be considered that this time depends on the level of network congestion, and therefore it can vary significantly. As an illustrative example, the traceroute command indicates that there are more than 70 hops between the Raspberry Pi and thethings.io, so varying levels of latency and jitter are expected.

| No. | Time | Protocol | Source | Destination | Length | Source Port | Destination Port | Info |
|-----|--------------|----------|----------------|----------------|--------|-------------|------------------|--|
| 9 | 12.973413464 | TCP | 192.168.42.122 | 104.199.77.107 | 74 | | 34560+443 | [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK... |
| 10 | 13.143750742 | TCP | 104.199.77.107 | 192.168.42.122 | 74 | | 443+34560 | [SYN, ACK] Seq=0 Ack=1 Win=28160 Len=0 MS... |
| 11 | 13.143774261 | TCP | 192.168.42.122 | 104.199.77.107 | 66 | | 34560+443 | [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=2... |
| 12 | 13.143922913 | TLSv1.2 | 192.168.42.122 | 104.199.77.107 | 583 | | | Client Hello |
| 13 | 13.189678925 | TCP | 104.199.77.107 | 192.168.42.122 | 66 | | 443+34560 | [ACK] Seq=1 Ack=518 Win=29312 Len=0 TSval... |
| 14 | 13.193851889 | TLSv1.2 | 104.199.77.107 | 192.168.42.122 | 1464 | | | Server Hello |
| 15 | 13.193864064 | TCP | 192.168.42.122 | 104.199.77.107 | 66 | | 34560+443 | [ACK] Seq=518 Ack=1399 Win=32128 Len=0 TS... |
| 16 | 13.193166187 | TCP | 104.199.77.107 | 192.168.42.122 | 1464 | | | [TCP segment of a reassembled PDU] |
| 17 | 13.193169789 | TCP | 192.168.42.122 | 104.199.77.107 | 66 | | 34560+443 | [ACK] Seq=518 Ack=2797 Win=35072 Len=0 TS... |
| 18 | 13.196279687 | TLSv1.2 | 104.199.77.107 | 192.168.42.122 | 491 | | | CertificateServer Key Exchange, Server Hello Done |
| 19 | 13.196284840 | TCP | 192.168.42.122 | 104.199.77.107 | 66 | | 34560+443 | [ACK] Seq=518 Ack=3222 Win=37888 Len=0 TS... |
| 20 | 13.198886209 | TLSv1.2 | 192.168.42.122 | 104.199.77.107 | 192 | | | Client Key Exchange, Change Cipher Spec, Encrypted ... |
| 21 | 13.249932242 | TLSv1.2 | 104.199.77.107 | 192.168.42.122 | 117 | | | Change Cipher Spec, Encrypted Handshake Message |
| 22 | 13.250360796 | TLSv1.2 | 192.168.42.122 | 104.199.77.107 | 859 | | | Application Data |
| 23 | 13.336687041 | TLSv1.2 | 104.199.77.107 | 192.168.42.122 | 844 | | | Application data |
| 24 | 13.336839789 | TCP | 104.199.77.107 | 192.168.42.122 | 66 | | 443+34560 | [FIN, ACK] Seq=4051 Ack=1437 Win=30848 Le... |
| 25 | 13.337072755 | TCP | 192.168.42.122 | 104.199.77.107 | 66 | | 34560+443 | [FIN, ACK] Seq=1437 Ack=4052 Win=40704 Le... |
| 26 | 13.376654929 | TCP | 104.199.77.107 | 192.168.42.122 | 66 | | 443+34560 | [ACK] Seq=4052 Ack=1438 Win=30848 Len=0 T... |

Figure 37: encrypted conversation between Raspberry Pi and Thethings.io

5.1.2 Network traffic

5.1.2.1 Cloud computing

In the cloud computing model, all readings are sent to the cloud. Therefore, the following data is uploaded each minute:

- Temperature of 5 motes.
- Humidity of 5 motes.
- Presence status of 5 motes.

A total of 15 measurements are sent. By trace inspection using Wireshark, it has been possible to determine that the size of the Application Data packets sent by the thethings.io for this quantity of measurements is **around 2200 bytes** (see Figure 36 for example).

Since data is sent every minute, the following quantities are expected over the year.

| | Bytes sent | API Calls |
|---------------|------------|-----------|
| Minute | 2200 | 1 |
| Hour | 132000 | 60 |
| Day | 3168000 | 1440 |
| Year | 1156320000 | 525600 |

Table 18: network traffic in cloud model

5.1.2.2 Edge computing

In the edge computing model, only statistical metrics are sent to the cloud. Therefore, the following data is uploaded each hour:

- Average value, max value, min value and standard deviation of the temperature for 5 motes.
- Average value, max value, min value and standard deviation of the humidity for 5 motes.
- Least active room, most active room and number of people at home for 1 mote.

A total of 41 values are sent every hour. By trace inspection using Wireshark, it has been possible to determine that the size of the Application Data packets sent by the thethings.iO for this quantity of values is **around 3300 bytes** (see Figure 38 for an example).

| Address A | Address B | IPv4 Conversations | | | | |
|----------------|----------------|--------------------|---------|-------------|-------------|-------------|
| | | Packets A → | Packets | Bytes A → B | Packets B → | Bytes B → A |
| 192.168.42.122 | 192.168.42.129 | 2 | 4 | 152 | 2 | 244 |
| 104.199.77.107 | 192.168.42.122 | 10 | 21 | 4 719 | 11 | 3 358 |

Figure 38: bytes sent from Raspberry Pi to thethings.iO in the edge model

Since data is sent every hour, the following quantities are expected over the year.

| | Bytes sent | API Calls |
|-------------|------------|-----------|
| Hour | 3300 | 1 |
| Day | 79200 | 24 |
| Year | 28908000 | 8760 |

Table 19: network traffic in cloud model

As it can be seen, the data sent in the cloud computing model is **40 times bigger** than in the edge computing model.

5.1.3 Power consumption

5.1.3.1 Cloud computing

Since thethings.iO is a cloud platform, infrastructure details such as used hardware are difficult, if not impossible, to know. As it could be expected thethings.iO does not disclose such information, and therefore it has not been possible to estimate the power consumption. Even though this information is not known, it seems reasonable to assume that Thethings.iO uses external data centers operated by third parties for its operation. Therefore, energy consumption is considerably diluted, as any processing node in the data center can potentially be shared by thousands of applications.

5.1.3.2 Edge computing

In the case of Raspberry Pi, a rough estimation of its power consumption is presented.

- Although the maximum current that the Raspberry Pi 3 Model B+ can draw from the power supply is 1A, power consumption under heavy load typically has values of around 700 mA [47]. This only occurs on peak periods, but it has been taken as the worst case (a less conservative value would be 500 mA).
- Since the supply operates at 5 V, the power consumption has a value of $55 \cdot 0.7 = 3.5 \text{ W}$.
- At this consumption rate, it would take the Raspberry Pi around 286 hours to use 1 kWh.
- Assuming a leap year of 8790 hours (worst case), the Raspberry Pi would consume $\frac{8790}{286} = 30.73 \text{ kWh}$.
- Using a cost simulator [48] and not taking into account the cost of the subscription to the power line, a resulting cost of operating the Raspberry Pi for a period of 1 year is **4,73 €**.

While this is a very small value for a simple deployment, it can grow significantly as more devices are connected. For systems where thousands of devices are operated, energy consumption costs stop being negligible, and need to be considered.

5.1.4 Cost

5.1.4.1 Cloud computing

The pricing model in thethings.iO is based on three different tiers, differentiated by the number of “things”, as well as the maximum number of API calls. Such model is summarized in Table 20.

| | Price/month (€) | Things | API calls/month | Cloud code executions/month |
|--------------------|-----------------|--------|-----------------|-----------------------------|
| Prototyping | 29 | 10 | 50K | 10K |
| Advanced | 169 | 1.000 | 6M | 50K |
| Corporate | 499 | 5.000 | 60M | 500K |

Table 20: thethings.iO pricing model

For the purpose of this project, the appropriate category would be “Prototyping”. Nevertheless, more professional applications would very easily fall in the two subsequent categories, significantly increasing the price per month.

5.1.4.2 Edge computing

The Price in the Edge computing model is determined by the initial investment in hardware and the cost of operating the system. The cost of OpenMotes is not

taken into account since it is common for both the Cloud and the Edge computing model.

The price of an individual Raspberry Pi 3 Model B+ can range from 35 to 45 € depending on the supplier. Such quantity, added to the cost of operating to the Raspberry Pi for one year, is significantly smaller than the lowest subscription to thethings.iO (348 €).

5.1.5 Security

5.1.5.1 Edge computing

As it can be observed in Figure 39 (and previously in Figure 35), it is possible to inspect 802.15.4 frames and see its contents. Therefore, an eavesdropper would be able to consult the values returned by sensors. This is caused by the lack of implementation of 802.15.4 MAC security features in the communication stack provided by OpenWSN [49]. Therefore, the edge model does not provide strong protection mechanisms for sensible data.

This is especially relevant since 802.15.4 is a wireless protocol which operates in the free frequency bands, and therefore it is much more prone to attacks. Threats are not limited to data theft, since it would also be possible to the attacker to intercept the communications and transmit falsified data to the gateway, representing a sever threat to the security of the system.

5.1.5.2 Cloud computing

As it could be observed in Figure 37, interaction with thethings.iO through its API is protected with SSL over TCP. Therefore, communication is secure and it cannot be tampered or spied.

```
> Frame 18: 155 bytes on wire (1240 bits), 155 bytes captured (1240 bits) on interface 0
  Raw packet data
  > Internet Protocol Version 6, Src: bbbb::1, Dst: bbbb::1
  > User Datagram Protocol, Src Port: 0, Dst Port: 17754
  > ZigBee Encapsulation Protocol, Channel: 25, Length: 75
  > IEEE 802.15.4 Data, Dst: 14:15:92:cc:00:00:00:04, Src: 14:15:92:cc:00:00:00:01
  v Data (52 bytes)
    Data: f18003141592cc000000047a55110000000000000011415...
    [Length: 52]
```

| | | | |
|------|-------------------------|-------------------------|------------------------|
| 0000 | 60 00 00 00 00 73 11 08 | bb bb 00 00 00 00 00 | ~ s |
| 0010 | 00 00 00 00 00 00 00 01 | bb bb 00 00 00 00 00 | |
| 0020 | 00 00 00 00 00 00 00 01 | 00 00 45 5a 00 73 c5 24 |EZ.s.\$ |
| 0030 | 45 58 02 01 19 00 01 01 | ff 01 01 01 01 01 01 | EX. |
| 0040 | 01 02 02 02 02 00 00 00 | 00 00 00 00 00 00 4b | K |
| 0050 | 21 ec 66 fe ca 04 00 00 | 00 cc 92 15 14 01 00 00 | ! . f |
| 0060 | 00 cc 92 15 14 f1 80 03 | 14 15 92 cc 00 00 00 04 | |
| 0070 | 7a 55 11 00 00 00 00 00 | 00 00 01 14 15 92 cc 00 | zU. |
| 0080 | 00 00 03 16 33 16 33 00 | 16 42 5e 41 01 d0 2d c2 | 3.3. .B^A. . . |
| 0090 | b8 70 72 65 73 65 6e 63 | 65 c3 a4 | .presenc e.. |

Figure 39: 802.15.4 frame containing presence reading

5.1.6 Summary of comparison

| | Cloud | Edge |
|------------------------------|---|---|
| Latency (ms) | 1.05* | 1 |
| Network Traffic (GiB/year) | 1102.7527 | 27.5688 |
| Power consumption (kWh/year) | ** | 30.73 |
| Cost (€/year) | 348 | 50*** |
| Security | High by default, but provides less control over how data is protected | Low by default, but can be hardening according to needs |

Table 21: * Severe delays (up to 10 seconds) have been observed in some occasions. **It has not been possible to estimate the power consumption of thethings.iO. ***Including the cost of buying the Raspberry Pi (i.e. it would only affect the first year).

Both models under study, the edge computing model and the cloud computing model, have proven to have both advantages and disadvantages. As a consequence, it is not possible to determine a winner, since this is entirely relative to the purpose of and scope of each individual application. However, some high-level conclusions can be drawn.

Commercial cloud platforms such as thethings.iO greatly simplify the design, prototyping and implementation of IoT projects. It provides a set of tools that allow its users to very easily setup data retrieval, monitoring and business intelligence, analyzing storing data and actuating as a response to registered changes. More importantly, many of these features are managed in a very convenient manner, being based mostly on drag-and-drop widgets and easy configuration forms.

In the same manner, interoperability with other platforms is provided off-the-shelf by thethings.iO. One example is the integration with SendGrid, which has been used in this project for sending email alarms.

On the other hand, the edge computing model allows to use any possible technology that is supported by the used platform (Raspberry Pi in this case), providing greater flexibility to the designers, in exchange of a steeper design curve, with increased complexity.

For small deployments, the edge computing model can be more convenient due to its reduced cost and tighter control over the resources. However, for larger applications other factors will influence. In particular, deployments with number of devices in the order of hundreds can benefit from the shared costs of processing all data at the cloud.

When generated network traffic is a design driver, it is clear that the edge computing model puts a greatly smaller burden on the network, with the number of gigabytes sent to the cloud being 40 times bigger in the case of cloud computing. Therefore, in cases where network resources are constrained, and edge computing model can be more appropriate.

In terms of security, thethings.iO provides security by default, encrypting the data that is sent to the cloud. Therefore, the weaker link in the chain is located at the edge, since readings performed using CoAP are performed unencrypted, being susceptible to both theft and falsifications.

As shown in Table 1, latencies in both cases are considerably low, and have similar values. However, experiments have demonstrated that the cloud platform offers significant delays of up to 10 seconds in some occasions. Therefore, whenever the response time is critical, the edge computing model is the logic choice, as the proximity of end devices with the intelligence of the network results in close-to-zero latencies. In cases where the response time is not critical, the cloud computing model can greatly facilitate the operation of the application, while ensuring scalability and continuous availability of the collected data. The tradeoffs need to be studied in each case.

6 Conclusions

The present document has described the design and implementation of an IoT demonstrator, characterized by its use of currently relevant technologies in the area. In particular, the focus has been put into open source and open hardware solutions, which represent a crucial trend in IoT.

From a technical perspective, this project has made it possible to gain an insight into IoT and its related technologies. While IoT is already an established term, it is often confused or used too broadly. For this reason, it becomes even more important to get hands on with the technology, learning about its inner workings.

With the objective of enabling meaningful results, the proposed design has focused on the use of IoT at home, in what is known as the Smart Homes vertical market. This approach has been embodied in the design system for collecting and monitoring comfort metrics, like temperature and humidity, including actuation to control such comfort metrics.

Moreover, two different approaches to data collection and analysis in the area of IoT, namely the Edge Computing model and the Cloud Computing model, have been presented and compared by applying them to the designed application. The results obtained have allowed to shed some light into what are their main strengths and weaknesses, and which application areas are more suitable for each of them.

As it has been mentioned, the implementation has been performed using relevant open source technologies like OpenWSN, which implements stacks of crucially important protocols in the field of IoT, e.g. 802.15.4 and CoAP. Particularly significant has been the role of the simulator, OpenSim, and its associated application for visualization, OpenVis, which have been key enablers for analyzing the models under study without real hardware.

In summary, this project has provided firsthand experience with a topic as tending and promising as IoT. Some broader conclusion that can be drawn from this experience are:

- IoT is a vibrant research trend, with a huge ecosystem of technologies and areas.
- IoT is not only a research topic, it is also a crucial industry market, where companies are investing and also developing commercially-available products which are bound to affect how we live our lives in the years to come.
- Open Source and Open Hardware initiatives, especially in the area of WSN, are key driving factors of innovation. They allow knowledge to be more easily transferred between laboratories and industry, as well as accelerating the adoption rate of new and convenient technologies by society as a whole.

7 Future work

The presented project is nothing but an insignificant portion of what is possible in the field of IoT. For this reason, many possible paths could be followed to continue the work initiated by this project:

- Probably the most important line of future work would be validating the simulation results with real hardware. Although this was planned in the initial scope, using Contiki OS over the OpenMote platform, it has not been possible to include it as part of the dissertation.
- Many of the presented utilities, such as the actuation over the air conditioning, although suitable for serving as archetypical examples of IoT applications, are rather naïve in their implementation. Much more complex applications could be developed, for example based on historical data, seasonal information, occupancy patterns, etc. However, this could require a significant amount of time, as a complex application could constitute a final year project on its own.
- For the Edge Computing model, very basic features have been developed, mainly concerned with data retrieval and storage. Nevertheless, much more intelligence can be translated to the edge of the network, due to the high capacity of SBC such as the one used.
- The code used for applications does not reach industry quality standards. Code could be heavily improved by means of protecting it against contingencies (for example using the exception mechanism in Python) and its behavior could be further assured by means of unit testing.
- Another possibility would be comparing the features and performance offered by different cloud platforms instead of thethings.iO.

8 Glossary

- **CoAP**: Constrained Application Protocol
- **DAG**: Directed Acyclic Graph
- **GPIO**: General Purpose Input Output
- **GUI**: Graphical User Interface
- **HTTP**: Hypertext Transfer Protocol
- **IEEE**: Institute of Electrical and Electronic Engineering
- **IETF**: Internet Engineering Task Force
- **JSON**: Javascript Object Notation
- **MAC**: Medium Access Control
- **MQTT**: Message Querying Telemetry Transport
- **OSI**: Open System Interconnection
- **PAN**: Personal Area Network
- **SBC**: Single Board Computer
- **TCP**: transmission Control Protocol
- **TSCH**: Time Slotted Channel Hopping
- **UDP**: User Datagram Protocol
- **URI**: Unified Resource Identifier
- **WPAN**: Wireless Personal Area Network
- **WSN**: Wireless Sensor Networks

9 References

- [1] J. Moolayil, *Smarter Decisions - The Intersection of Internet of Things and Decision Science*, Packt Publishing Ltd, 2016.
- [2] C. a. A. J. C. a. S. D. Ramos, "Ambient intelligence - the next step for artificial intelligence," *IEEE Intelligent Systems*, vol. 23, no. 2, pp. 15-18, 2008.
- [3] D.-M. a. L. J.-H. Han, "Smart home energy management system using IEEE 802.15. 4 and ZigBee," *IEEE Transactions on Consumer Electronics*, vol. 56, no. 3, 2010.
- [4] G. a. H. B. K. a. o. Demiris, "Technologies for an aging society: a systematic review of "smart home" applications," *Yearb Med Inform*, vol. 3, pp. 33-40, 2008.
- [5] K. a. G. I. a. A. R. a. P. N. a. K. A. Akkaya, "IoT-based occupancy monitoring techniques for energy-efficient smart buildings," in *Wireless Communications and Networking Conference Workshops (WCNCW)*, IEEE, 2015, pp. 58-63.
- [6] R. C. Chacón, "Demostrador de Internet of Things con la tecnología IEEE 802.15.4e utilizando la plataforma OpenMote, sistema operativo Openwsn y TheThings.iO," UOC, 2018.
- [7] M. M. Salas, "Demostrador de Internet of Things con la tecnología IEEE 802.15.4e utilizando la plataforma OpenMote, sistema operativo Openwsn y TheThings.iO," UOC, 2017.
- [8] M. a. B. M. a. L. M. Vahabi, "A Heterogeneous IoT-Based Architecture for Remote Monitoring of Physiological and Environmental Parameters," *Internet of Things (IoT) Technologies for HealthCare*, vol. 225, p. 48, 2018.
- [9] G. a. M. E. a. V. d. V. B. a. E. G. a. W. Daneels, "Accurate Energy Consumption Modeling of IEEE 802.15. 4e TSCH Using Dual-Band OpenMote Hardware," *Sensors*, vol. 18, no. 2, p. 437, 2018.
- [10] P. a. P. G. a. S. D. a. G. L. A. Boccadoro, "Experimental comparison of Industrial Internet of Things protocol stacks in Time Slotted Channel Hopping scenarios," 2018.
- [11] A. Powell, "Democratizing production through open source knowledge: from open software to open hardware," *Media, Culture & Society*, pp. 691-708, 2012.
- [12] R. a. L. L. a. H. G. a. K. C. Fisher, "Open Hardware: A Role to Play in Wireless Sensor Networks?," *Sensors*, pp. 6818-6844, 2015.
- [13] S. M. J. P. R. S. K. W. A. W. P. Levis, "TinyOS: An Operating System for Sensor Networks," in *Ambient Intelligence*, Berlin, Springer Berlin Heidelberg, 2005, pp. 115-148.
- [14] TinyOS Development Team, "TinyOS FAQ," TinyOS Development Team, 2011. [Online]. Available: <http://tinyos.stanford.edu/tinyos-wiki/index.php/FAQ>. [Accessed April 2018].

- [15] M. e. a. Maróti, "The flooding time synchronization protocol," in *Proceedings of the 2nd international conference on Embedded networked sensor systems*, 2004.
- [16] R. e. a. Fonseca, "The collection tree protocol (CTP)," TinyOS TEP, 2006.
- [17] P. e. a. Levis, "Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks," in *Proc. of the 1st USENIX/ACM Symp. on Networked Systems Design and Implementation*, 2004.
- [18] T. Winter, "RPL: IPv6 routing protocol for low-power and lossy networks," Internet Engineering Task Force (IETF), 2012. [Online]. Available: <https://tools.ietf.org/html/rfc6550>. [Accessed April 2018].
- [19] E. a. H. O. a. G. M. a. W. M. a. S. T. C. Baccelli, "RIOT OS: Towards an OS for the Internet of Things," in *Computer Communications Workshops (INFOCOM WKSHPS)*, 2013.
- [20] A. a. G. B. a. V. T. Dunkels, "Contiki-a lightweight and flexible operating system for tiny networked sensors," *Local Computer Networks*, pp. 455-462, 2004.
- [21] M. O. a. T. K. Farooq, "Operating systems for wireless sensor networks: A survey," *Sensors*, pp. 590-593, 2011.
- [22] N. Briscoe, "Understanding the OSI 7-layer model," *PC Network Advisor*, 2000.
- [23] IEEE 802.15.4 Working Group, "IEEE Standard for Local and metropolitan area networks--Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)," *IEEE Std 802.15.4-2011 (Revision of IEEE Std 802.15.4-2006)*, pp. 1-314, 2011.
- [24] U. a. T. H. L. a. S.-C. A. Hunkeler, "MQTT-S—A publish/subscribe protocol for Wireless Sensor Networks," in *Communication systems software and middleware and workshops*, 2008.
- [25] IETF, "The Constrained Application Protocol (CoAP)," 2014. [Online]. Available: <https://tools.ietf.org/html/rfc7252>. [Accessed April 2018].
- [26] Q. L. C. a. R. B. Zhang, "Cloud computing: state-of-the-art and research challenges," *Journal of internet services and applications 1.1*, pp. 7-18, 2010.
- [27] P. a. T. G. Mell, "The NIST definition of cloud computing," 2011.
- [28] M. e. a. Armbrust, "A view of cloud computing," *Communications of the ACM 53.4*, pp. 50-58, 2010.
- [29] W. a. S. D. Shi, "The promise of edge computing," *Computer 49.5*, pp. 78-81, 2016.
- [30] M. Satyanarayanan, "The emergence of edge computing," *Computer 50.1*, pp. 30-39, 2017.
- [31] C. a. L. C. a. Y. F. R. a. C. Q. a. T. L. Wang, "Computation offloading and resource allocation in wireless cellular networks with mobile edge computing," *IEEE Transactions on Wireless Communications*, vol. 16, no. 8, pp. 924-938, 2017.
- [32] M. a. V. V. Maksimovi, "Raspberry Pi as Internet of things hardware: performances and constraints," *Design Issues*, vol. 3, p. 8, 2014.

- [33] Raspberry Pi Foundation, "Raspberry Pi," <https://www.raspberrypi.org/>, 2018. [Online]. [Accessed April 2018].
- [34] Raspberry Pi Foundation, "Raspberry Pi 3Model B+ Product Brief.pdf," Raspberry Pi Foundation, 2018. [Online]. Available: <https://static.raspberrypi.org/files/product-briefs/Raspberry-Pi-Model-Bplus-Product-Brief.pdf>. [Accessed April 2018].
- [35] P. P. Ray, "A survey of IoT cloud platforms," *Future Computing and Informatics Journal*, vol. 1, no. 1, pp. 35-46, 2016.
- [36] thethings.io, "thethings.io Landing page," thethings.io, [Online]. Available: <https://thethings.io/>. [Accessed April 2018].
- [37] thethings.io, "thethings.io Features," thethings.io, [Online]. Available: <https://thethings.io/iot-dashboards-features/>. [Accessed April 2018].
- [38] thethings.io, "Python API for <http://thethings.io>," thethings.io, 2018. [Online]. Available: <https://github.com/theThings/thethings.io-python-library>. [Accessed April 2018].
- [39] The PostgreSQL Global Development Group, "PostgreSQL," The PostgreSQL Global Development Group, 2018. [Online]. Available: <https://www.postgresql.org/>. [Accessed April 2018].
- [40] The PostgreSQL Global Development Group, "PostgreSQL About," The PostgreSQL Global Development Group, 2018. [Online]. Available: <https://www.postgresql.org/about/>. [Accessed April 2018].
- [41] European Union, "Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data," European Union, [Online]. Available: <https://eur-lex.europa.eu/legal-content/en/TXT/?uri=CELEX%3A31995L0046>. [Accessed March 2018].
- [42] OpenWSN, "Architecture," OpenWSN, June 2013. [Online]. Available: <https://openwsn.atlassian.net/wiki/spaces/OW/pages/7471117/Architecture>. [Accessed 2018].
- [43] OpenWSN, "IEEE802.15.4e," OpenWSN, November 2012. [Online]. Available: <https://openwsn.atlassian.net/wiki/spaces/OW/pages/688145/IEEE802.15.4e>. [Accessed 2018].
- [44] OpenWSN, "How to write a module in OpenWSN firmware," OpenWSN, 2015. [Online]. Available: <https://openwsn.atlassian.net/Togglewiki/spaces/OW/pages/97878065/How+to+write+a+module+in+OpenWSN+firmware>. [Accessed April 2018].
- [45] OpenWSN, "CoAP Python Library," OpenWSN, [Online]. Available: <https://openwsn.atlassian.net/wiki/spaces/OW/pages/32014351/CoAP+Python+Library>. [Accessed April 2018].
- [46] D. Varrazzo, "psycopg," 2018. [Online]. Available: <http://initd.org/psycopg/>. [Accessed April 2018].
- [47] A. Eames, "How Much Power Does Raspberry Pi 3B+ Use? Power Measurements," 2018. [Online]. Available: <http://raspi.tv/2018/how-much-power-does-raspberry-pi-3b-use-power-measurements>. [Accessed May 2018].

- [48] CNMC, "Simulador de la factura de la luz," Comisión Nacional de los Mercados y la Competencia (CNMC), 2018. [Online]. Available: <https://factualuz.cnmc.es/factualuz1.html#datos>. [Accessed May 2018].
- [49] S. a. P. G. a. B. G. a. G. L. Sciancalepore, "Application of IEEE 802.15. 4 security procedures in OpenWSN protocol stack," *IEEE Standards Education e-Magazine*, vol. 4, no. 2, pp. 1-9, 2015.
- [50] resin.io, "Etcher," resin.io, 2018. [Online]. [Accessed April 2018].
- [51] OpenWSN, "Kickstar Linux," OpenWSN, 2014. [Online]. Available: <https://openwsn.atlassian.net/wiki/spaces/OW/pages/29196302/Kickstart+Linux>. [Accessed April 2018].

10 Appendix

10.1 psychopg2 setup

The installation of psychopg2 Its installation can be performed using pip.

```
$ pip install psychopg2
...
Successfully installed psychopg2-2.7.4
```

10.2 Raspberry Pi 3 Model B+ setup

The first step is flashing a SD card with the Raspbian OS. The method chosen has been Etcher [50], which is a multiplatform application for flashing SD cards in an easy manner. Etcher is officially recommended by the Raspberry Pi Foundation and, therefore, the setting up Raspbian is straightforward.

Once the image has been flashed, it is inserted in Raspberry Pi's SD slot and the Raspberry Pi can be powered up.

The GUI can be disabled to save resources using the `raspi-config` utility. As a consequence, the system resources will be more efficiently used for the computing tasks at hand.

10.3 OpenSim setup

The process for installing OpenSim in Linux (as it is the case in the Raspberry Pi) can be found in [51].

```
$ mkdir openwsn; cd openwsn
$ git clone https://github.com/OpenWSN-berkeley/openwsn-fw
$ git clone https://github.com/OpenWSN-berkeley/openwsn-sw
$ git clone https://github.com/openwsn-berkeley/coap.git
$ sudo apt-get install scons
$ sudo /usr/bin/pip2.7 install -r requirements.txt
```

The openwsn software can be built for the python board, i.e. instead of building it for a real device, such as the CC2538, a Python-based simulation will be run instead.

```
$ cons board=python toolchain=gcc oos_openwsn
```

10.4 PostgreSQL setup

PostgreSQL is supported off-the-shelf on Rasbian.

```
$ sudo apt install postgresql libpq-dev postgresql-client postgresql-client-common
```

Once installed, a new database user must be created. For simplicity, the username will be the same as Raspbian's default username, "pi".

```
$ sudo su postgres
postgres@openwsn-gateway:/home/pi$ createuser pi -P -interactive
```