TÉCNICO LISBOA

EnergyBrick: The smart low-cost energy meter for the IoT

Tiago Edgar Sôra Barbosa

Thesis to obtain the Master of Science Degree in

Telecommunications and Informatics Engineering

Supervisor: Prof. Paulo Jorge Fernandes Carreira

Examination Committee

Chairperson: Prof. Paulo Jorge Pires Ferreira
Supervisor: Prof. Paulo Jorge Fernandes Carreira
Member of the Committee: Prof. Mário Serafim dos Santos Nunes

May 2014

Dedicated to my family

Agradecimentos

Gostaria de agradecer ao meu orientador Prof. Paulo Carreira pelo seu apoio, pela exigência, pelas críticas, pelos comentários, pelas sugestões, pelos elogios, e sobretudo por ter acreditado desde o início em mim e nas potencialidades deste projeto. Gostaria também de agradecer à ajuda finânceira prestada para a realização desta tese. Por tudo isto, um sincero obrigado.

Gostaria também de agradecer a todos os colegas da sala de orientação, especialmente ao Diogo Anjos pelo magnífico trabalho na construção dos gráficos de energia, e ao Vitor Mansur pelo seu enorme apoio ao longo de toda a tese.

Um especial agradecimento à Sílvia Resendes por toda a sua paciência em reler o documento, e por ter sido uma excelente amiga ao longo de todo o curso e que espero manter para o resto da vida.

Finalmente gostaria de agradecer também à minha família por todo o apoio prestado não só agora durante a universidade, mas ao longo de toda a minha vida, tendo sido sempre uns pais exemplares e com uma paciência infindável para que pudesse concluir com êxito esta etapa tão importante. Este trabalho é dedicado a vocês.

Maio 2014

Resumo

Diversas soluções tecnológicas, tais como sistemas de gestão de energia (SGE), tem vindo a surgir para permitir a monitorização de recursos energéticos. Embora permitam às empresas gerir e reduzir custos energéticos, estas soluções são ainda dispendiosas, devido não só ao preço dos equipamentos, mas também a instalação dos mesmos, que tem de ser efetuado por pessoal qualificado.

Na prática, a maioria dos EMS tem um conjunto limitado de medidores que são insuficientes para uma análise aprofundada dos consumos energéticos. O desenvolvimento de uma solução de baixo-custo baseada em eletrónica e protocolos abertos poderá permitir a sua adoção em grande escala.

Esta tese propõe o desenvolvimento de um medidor energético de baixo-custo e fácil instalação, que permitirá também a criação de uma rede mesh sem fios de medidores energéticos escalável que permitirá efectuar sub-medições energéticas com o detalhe desejado.

Para validar esta solução de baixo-custo diversos medidores energéticos foram desenvolvidos e ligados entre eles, formando uma rede mesh sem fios. Cada um deles suporta comunicação e medições energeticas em diferentes cenários. Na solução criada, é possível efectuar medições energeticas de diferentes modos: *i)* medindo a energia activa e reactiva ou *ii)* requisitando essas mesmas medições a um medidor inteligente previamente instalado no edifício Essas medições são então depois encaminhadas usando o protocolo Modbus sobre a rede mesh sem fios. Adicionalmente, a integração de novos módulos é efectuada de forma muito simples: ao ligar, ele registar-se-á automaticamente no sistema.

Este trabalho demonstra que é possível criar um SGE de baixo-custo com custos de integração, aliviando assim o custo inicial, e reduzindo o tempo de retorno do investimento.

Palavras-chave: Energia, Sistemas de Gestão Energética, Baixo-custo, medidor energético

Abstract

Several technological solutions, such as Energy Management Systems (EMS), have been developed to allow the monitoring of energy resources. Despite enabling energy managers to monitor and reduce energy costs, these solutions are still quite expensive, mostly due to the equipment price but also to their installation and commissioning effort, which must be performed by qualified personnel. Moreover in practice, most EMS feature a limited number of meters which are insufficient for an effective energy consumption analysis. The design of a low-cost EMS solution based on open electronics and protocols will likely enable the adoption of large number of meters. This thesis proposes to develop an easy to install and low-cost meter that will also allow the creation of a scalable wireless mesh network of energy meters to perform sub-metering with a desirable resolution.

To validate the low-cost meter solution several energy data metering modules were developed and connected to each other, forming a wireless mesh network. Each one handles communications and energy metering were tested in different scenarios. Indeed, the created solution is able to perform energy metering in different ways: *i)* measuring active and reactive energy or *ii)* request to an already installed smart-meter. Those readings are then forwarded using the Modbus over a wireless mesh network. In addition, commissioning of a new module is performed in a very simple way: turning it on it will register it automatically in the system.

This work demonstrates that it is possible to create a low-cost EMS with reduced implementation and commissioning costs, thus alleviating the initial investment, and improving the payback time.

Keywords: Energy, Energy Management Systems, Low-cost, energy meter

Contents

List of Tables

List of Figures

Acronyms

AC Alternating Current

AP Access Point

CSMA/CA Carrier Sense Multiple Access / Collision Avoidance

CT Current Transformer

DSSS Direct-Sequence Spread Spectrum

DIY Do It Yourself

DHCP Dynamic Host Configuration Protocol

EMS Energy Management Systems

FFD Full Functional Device

GTS Granted Time Slots

HDMI High-Definition Multimedia Interface

I2C Inter-Integrated Circuit

IP Internet Protocol

ISO International Organization for Standardization

IST Instituto Superior Técnico

I/O Input/Output

kWh kilowatt hour

LAN Local Area Networks

LCD Liquid-Crystal Display

M2M Machine to Machine

MAC Media Access Control

OSI Open Systems Interconnection

PAN Personal Area Network

PDU Protocol Data Unit

PCB Printed Circuit Board

PWM Pulse Width Modulation

PLC Programmable Logic Controller

QoS Quality of Service

RFD Reduced Functional Device

SD Secure Digital

SPI Serial Peripheral Interface

TCP Transmission Control Protocol

UDP User Datagram Protocol

VT Voltage Transformer

WMN Wireless Mesh Networks

Chapter 1

Introduction

Active energy consumption reduction is at the top of energy policy agendas. The European Commission has created the “Energy 2020” program with the goal of reducing energy consumption by 20% until 2020 [Eur11]. A solution for efficient energy usage, is to use technology-based products that assist people in taking actions to reduce its usage, without compromising their comfort. Not only technology is needed, but also the mindset must change. People currently take energy as granted, and do not take actions towards not wasting it. When the energy provider fails, that period of time can make people think. Organizations consider energy costs as an overhead, rather than controllable costs, and most of them do not take actions towards energy saving. Many are unaware of its potential economical and ecological advantages.

Aiming at reducing energy consumption, an international standard was also created. The ISO 50001 [ISO11] standard was created by the International Organization for Standardization (ISO) [Int47] to provides a framework with best practices to establishing, implementing, maintaining and improving Energy Management Systems (EMS).

EMS provide a set of tools that give an overview of the energy usage status of a building. Having that information available, it is possible to take action towards reducing energy consumption, reducing operational costs without affecting the comfort of people. EMS performs three main tasks: *i)* gathering energy data, *ii)* interpreting collected data, *iii)* presenting results to the users in the form of reports. To perform the first task, energy meters are used to collect and store energy consumption for further analysis. When just some energy meters reports data, only a limited view of the building consumption is acquired. The larger the building, the lower the obtained detail. Ideally, spreading hundreds in a large building, implementing sub-metering techniques, can lead to a very well detailed report that can actually help to reduce energy consumption.

With many low-cost energy meters, it is possible to correctly measure all the energy that is being spent in a building, with an adequate level of accuracy and a reduced cost. The more meters are used, the better the result will be. It is a better approach to the problem to use many meters, than just one, and the herein proposed solution uses several low-cost meters instead of an expensive one. It “divides to conquer" instead of using just one highly advanced and expensive meter.

However, these meters have highly costs, that hinder a detailed consumption report. Indeed, a large range of smart energy meters exists in the market, but all of them share a common problem: their high price makes it unreasonable to spread hundreds of them in a large building to control the electricity costs. This situation can lead to poor reports that do not efficiently support the building manager to detect and correct possible problems. Such problems results in energy waste, which is the problem that the EMS is supposed to prevent.

This thesis aims to create, document and validate a wireless low-cost energy meter to report energy data consumption to the EMS. Such low-cost energy meters can be spread all over the building, with the cost of just some smart energy meters available in the market.

1.1 Problem Statement

Meter pricing commissioning smart-meters also represents a problem. However installation of smart-meters in existing building, in most cases, can be problematic. If the EMS must be implemented in a building that has several years, and passing communications cable can represent a problem, specially if it is a historical building. In new buildings, spreading communication cable is a minor a problem, since that it can still be included in the project.

A smart meter does not need to offer several complex functions to be effective. In fact, the simpler it is, the less it will cost. A simple meter with energy measurement and server reports as only tasks shows a great potential. With the information provided by the meter, the server can perform those complex functions and also produce better reports. Besides of this advantage, the information will be reported not just by one meter but also by dozens of meters spread all over the building, having in the end a much better view of the energy spent.

The key for a low-cost meter is to implement the simplest meter solution that only reports its measures to a central-server. This solution will allow the development of simple energy meters, while the main intelligence of the system is provided by software in the central-server.

With this kind of solution, a complete system can be produced with a low price, and this document will further discuss and detail how to implement an low-cost meter. An advantage of having low-cost meters is the possibility of performing *sub-metering* in a room by using multiple energy meters, and therefore know how much each equipment is consuming, instead of having just one meter that costs the same as the sum of low-cost ones, but only is able to identify how much the room is consuming.

Besides existing many EMS solutions in the market, all of them have common problems that need to be solved in order to increase their adoption:

- **Energy meters price** is too high to install hundreds of them in a large building. This limitation does not allow to perform *sub-metering* with a desirable resolution at a low-price.

- **Installation cost** is yet another important aspect that has to be taken into account and that can be further divided into:

  - Qualified personal has to perform the electrical installation of the energy meters representing a cost that can be avoided if the installation process were simplified
  - Powering down the building may also be needed which can represent a problem to the client.
  - Communication cable installation may also be a problem in older buildings, where adding new cable may not be a trivial.

- **Commissioning** effort required for each meter may require several interactions with specialized software.

All these problems still exist in EMS solutions, and there are not a single solution that does not have the mentioned problems, however we think is possible to solve them using low-cost open electronics hardware that can be easily installed without previous specialized knowledge about electricity and that also provides auto-commissioning. This is the main objective of this thesis, and this document will focus on the first main task of EMS, by presenting a low-cost solution for gathering energy data using open hardware, including details about current measurements and deliverance to the main server using a low-cost wireless mesh network.

1.2 Goals and Contributions

The major contributions of this thesis are as follows:

- Develop, document and validate a RS-485 and Modbus compatible low-cost meter for the EMSs, that is also easy to install.
- Develop and validate a wireless mesh network protocol using low-cost hardware that ease up the energy meters commissioning.
- Field test a metering solution that can be used in a large building such as the Instituto Superior Técnico (IST) Taguspark campus
- Integrate smart-meters in a EMS that may be used in the building.

1.3 Document Organization

The rest of this document is structured as follows:

- **Chapter 2** presents the most important concepts related with this thesis to better understand it. It starts to introduce energy concepts in Section 2.1, then the Arduino Platform in Section 2.2, Energy Metering Protocols in Section 2.3, and finally Wireless Mesh Network concepts in Section 2.4.
- **Chapter 3** describes some related work regarding our thesis. There are many different energy meter solutions that are already available. Besides being impossible to study them all, the most similar are presented.

- **Chapter 4** presents the solution to the problem. It starts by presenting a Solution Overview in Section 4.1, and a Block Diagram in Section 4.1.3. After describing each block, several models EnergyBricks are presented in Section 4.2, where their the internal memory is detailed in Section 4.3.1, and the used protocols are detailed in Section 4.3.2.
- **Chapter 5** describes all the experiments made using the developed solution. To test all solution features, several different tests were performed, to validate them. Some of the tests were executed to discover the solution limitations.
- **Chapter 6** concludes this document by presenting the Achievements in Section 6.1, the Lessons Learned during this work in 6.2, and considerations about Future Work in Section 6.3.

Chapter 2

Concepts

In this section, some important concepts regarding this document will be presented. This section is divided in four main topics: *i)* energy and *ii)* Arduino platform, *iii)* Energy Metering Protocols, and *iv)* Wireless Mesh Networks.

The concepts about energy usage are detailed in Section 2.1, where it starts to detail the electric power (Section 2.1.1), different kinds of electric loads (Sections 2.1.2), followed by an overview of which equipments are used to measure electricity consumption (Section 2.1.3). The concept of EMS and their capabilities are presented in Section 2.1.4, followed by the concepts of data acquisition frequency (Section 2.1.5), sub-metering (Section 2.1.6) and data quality (Section 2.1.7).

The second topic, regarding concepts about the Arduino platform, details its most relevant hardware aspects in Section 2.2.1, while its software features are described in Section 2.2.2.

The Energy Metering Protocls are presented in Section 2.3 where the Modbus protocol is detailed in Section 2.3.1, then the M-bus is introduced in Section 2.3.2, and a comparison between both are presented in Section 2.3.3.

The Wireless Mesh Networks concepts are presented in Section 2.4, where it starts to present the mesh network over WiFi in Section 2.4.1 followed by mesh networks over ZigBee in Section 2.4.2.

2.1 Energy

2.1.1 Electric Power

Consumption

The total energy needed in a given time interval is known as *consumption*, and can be monitored using energy meters. Energy meters measure the energy flows through in some point of an electric circuit, being kilowatt hour (kWh) the most generally used unit for measuring and billing, commonly known as the billing unit. This concept is usually called consumption, and it is sometimes confused with *demand*, which will be explained later.

To better understand consumption, consider a 100 W lamp that is turned on for 10 hours. This lamp,

to produce work (in this case, light) will consume 100 W in an hour, i.e., 100 Wh. If this lamp is turned on for 10 hours, the total consumption will be 100 W $\times$ 10 h = 1000 Wh = 1 kWh. At this point, it is clear to understand that after these 10 hours the energy meter will consider one more kWh, for which an electricity rate will be charged.

Consider now a heat radiator. Expectably, such equipment consumes a lot more than a simple lamp, typically around 1500 W. In one hour, it will consume 1500 Wh. An important fact is that a heater is adjustable, so at maximum heat it has a *consumption* of 1500 Wh, but at minimum heat it has a significantly smaller consumption, e.g., 750 Wh. Therefore, set to the minimum for 2 hours results in the same *consumption* (1500 Wh = 750 W $\times$ 2 h) as 1 hour at maximum heat (1500 Wh).

In the latter example, we are assuming that the heat radiator is turned on during the entire hour, however, it can have a different behavior: once reached the desirable temperature in a room, it automatically turns off, remaining off until the temperature in that room reaches a minimum value that affects the occupants' comfort, at which point it automatically turns on again. If, for instance, reaching the desired temperature takes 10 minutes, and reaching the minimum limit afterwards also takes 10 minutes, then the heater will be turned on for 10 minutes and then turned off for 10 minutes. Therefore, during an hour, we can expect half the consumption of the previous examples.

In the previous examples, the consumption was obtained using the following expression:

$$C = P \times t$$

The expression shows that the total consumption (C), measured in kWh, can be obtained by multiplying the power that the device needs to produce work (P), measured in kW, by the time that the device is working (h), [RGK02, Nat12], as used in previous examples.

Demand

Demand represents the maximum energy *required* in a given instant of time. When signing a contract with an electrical provider, one choice that has to be made is how much energy is required (demanded) in order to have the desirable equipments turned on at the same time. It is important for the electrical provider to know the maximum power required by each client, to have a good notion of how much energy it needs to produce in order to assure the electrical distribution, even during peak hours. In order to control the maximum energy provided to each client, a device called *circuit breaker* is installed in each client. For each maximum value, a different circuit breaker is used.

A circuit breaker acts as an interrupter, allowing the energy to pass until it reaches the maximum demand value defined in the contract with the electrical operator, cutting off the power if this maximum value is exceeded. By doing this, the electrical operator ensures that each client can only use that maximum demand energy level at any time. This maximum demand value is another important component of the billing, since the higher this value is, the more expensive is the monthly contract fee.

Each circuit breaker has a limit value, measured in amperes (A), which limits the electric current flow. All buildings and houses have circuit breakers. This mechanism also prevents damage from short circuits. A short-circuit occurs when the electricity has no resistance in the electric circuit, usually due

to some anomaly, resulting in an excessive current in the circuit that can damage electronic devices connected to it. If this happens, too much current will pass through the circuit breaker, and it will act by cutting the power. Usually, a circuit breaker has the information of how much current it should take before breaking the circuit.

The maximum demand in a circuit is calculated using the following expression:

$$MaxD = V \times CBV$$

The maximum demand (*MaxD*) can be obtained with a simple multiplication of the voltage level at which the energy provider operates (V), measured in volts, by the limit value of the circuit breaker (CBV), measured in amperes. The maximum demand is measured in kiloVolt-Ampere (kVA), which represents the *apparent power*, as will be explained later. For now, consider that 1 kVA equals 1 kWh, to simplify this explanation of demand.

As an example, consider the typical voltage value of electric energy, which in the case of Portugal is 230 V, and a circuit breaker of 15 A installed in a house. In this case, it is possible to get a maximum demand of 3.45 kVA (230V $\times$ 15A) and if this value is exceeded the circuit breaker will cut the power. The circuit will be broken if, for instance, three 1500 W heaters, as the one from the previous example, are turned on at same time, since their demand is 4.5 kVA (3 $\times$ 1500 KVA). If only two of those heaters are on, at the same time as two of our example lamps, the total demanded energy will be 3 kVA for the heaters (2 $\times$ 1.5 kVA), plus the lamps' demand, 0.2 kVA (2 $\times$ 0.1 kVA), totalling 3.2 kVA. This is tolerable by the 15 A circuit-breaker, since there are just 13.9 A (3200 VA / 230 V) passing through it.

2.1.2 Electrical Loads

In electricity, especially in Alternating Current (AC) circuits, which is the case of the energy provided by the electrical operators, the voltage and the current assume wave forms with a given frequency (depending on the country). This wave forms are usually perfectly aligned in phase, changing polarity at the same instance in each cycle, but when an electrical load is connected to the circuit, the perfect alignment could be lost, depending on the electrical load type.

There are three electrical load types: resistive, which behave purely as resistors, e.g., a lamp or a heater; capacitive, which have capacitors, e.g., flash of the camera; inductive which have coils, e.g., a motor. Resistive electrical loads do not affect the wave form's phase, but capacitive and inductive loads affect it, causing a phase shift between voltage and current. Capacitive and inductive loads, also known as reactive electrical loads, can cause a shift variation in the current wave, introducing reactive power. According to Nicholas Piotrowski *"Capacitors generate reactive power, and inductors consume reactive power"* [Pio10], and since electrical devices need both types of components, their design usually takes into account the need of avoiding this reactive power.

"Reactive power is the lazy brother of true power" [Pio10]. While true power (or real power) is the power presented in the system used to power up the devices, the reactive power has a negative impact on true power, draining true power in the system. The wave forms' alignment, and the consequences of phase shifts producing reactive power, are depicted in Figure 2.1. In *a)*, a perfect alignment is achieved,

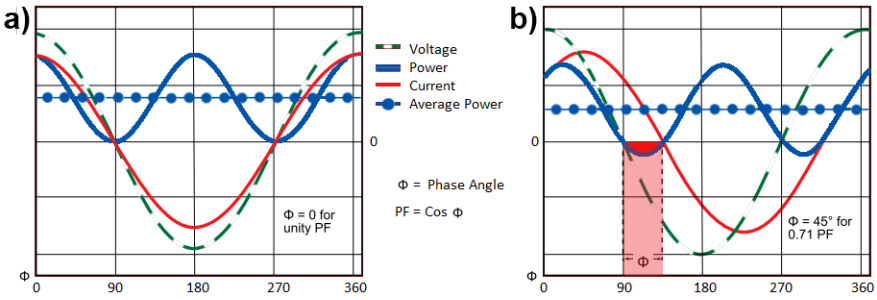

Figure 2.1: Electrical wave forms in AC - *a)* Representation of a purely resistive load. Both current and voltage waves are perfectly aligned ($\phi = 0$), hence, a unity power factor (PF) is achieved. *b)* Representation of a reactive load, depicting a shift phase variation that produces reactive power in the system (energy wave below 0). In this case, the 45 degree shift causes a lower power factor.

due to the presence of purely resistive loads in the circuit, which do not produce reactive power. In this case, only true power is presented (which can be obtained through the Ohm's Law), because the power wave is always positive. In *b)*, the phase alignment of voltage and current waves is not perfect. This can happen due to the introduction of reactive electrical loads (capacity and inductive loads), which cause the phase shift between both waves. In this case, it is possible to notice that the power wave, goes below the 0 value, becoming negative, which represents the reactive power, and therefore the average power in diagram *b)* is less than the average power in diagram *a)*.

Power factor

The Power factor indicates how efficient an equipment is. This concept is important to understand the apparent power, explained next. The Power factor is a value that varies between 0 and 1. An equipment having a power factor of 1 means that all the energy used is actually converted in work, and not wasted by heat dissipation in the system. Equipments usually have a power factor close to 1, which is the reason why 1 kWh is approximate to 1 kVA. With a power factor of 1, all energy consumed (apparent power) produces work (true power or real power), and in this case the kVA equals kWh. This is the case of resistive electrical loads, as mentioned before, in which the current and voltage waves are perfectly aligned in phase, thus not producing reactive power and resulting in a power factor of 1. Examples of resistive electrical loads are incandescent light bulbs, electric water heaters, and electric stoves.

Other types of devices consisting of reactive loads, such as fridges and computers, and many others with inductive or capacitive components, such as coils and capacitors, may have a power factor closer to 1. In these devices, it is desirable to thoroughly chose the components, in order to minimize the reactive power in the circuit, thus achieving a better power factor.

The higher the power factor is, the less the device will consume while producing the same work.

Apparent power

The apparent power is the total energy consumed by a device to produce work. As mentioned before, it has an inverse relation with the power factor: the higher the power factor is, the lower the apparent

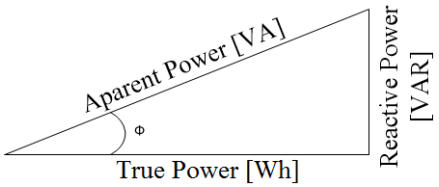

Figure 2.2: Complex power triangle - The bottom vertex represents the Real Power measured in Wh. The right vertex is the Reactive Power measured in VAR. On the top: the Apparent Power, measured in VA. The angle ϕ represents the Power Factor. The Power factor affects directly the consumption of the system. A Power factor of 1 means 0^o, and the lower the Power factor is, the higher the angle will be.

power will be, to produce the same work. This can be observed in Figure 2.2, which depicts the complex power triangle.

The total consumption of a device depends on its power factor. A lower power factor means that the device wastes more energy in the network grid e.g., by heat dissipation in the lines, thus consuming more apparent power in order to produce the same work (real power). This is important, since electric providers do not charge in kWh (real power), but rather in kVA (apparent power). The total consumption (apparent power) of a device regarding its power factor, can be obtained with the expression:

$$AP = RP/PF$$

The apparent power (AP), measured in VA (Volt-ampere), can be obtained with a division of the real power (RP) measured in Watt-hour by the power factor (PF). As a practical example, consider a 500 W computer power source with a power factor of 0.9. If the power factor were 1, then this equipment would consume 0.5 kWh, but with a power factor of 0.9, it will actually consume 555.55 VA (500 Wh / 0.9). The extra 55.55 W is wasted in heat form in the circuit or in the power lines, and it will be charged by the power provider, since, as explained before, they charge the apparent power (in kVA).

The herein detailed concepts were the most relevant to understand how energy is measured and charged, thus providing the needed basis to understand how an electricity meter works and how it can be used to our advantage, to reduce energy waste, and consequently, energy costs.

2.1.3 Energy Metering

Energy metering is the act of measuring the electricity that passes in a given point of the circuit where the meter is installed. Energy meters are cumulative devices that give information about the total energy consumption since their installation. Typically, a building has energy meters on its energy entry point, belonging to the utility company, which uses it to charge the consumer according to the measured energy. It is also possible to acquire additional energy meters for personal use to perform measurements in specific places, therefore attaining more detailed information about the consumption.

There are also energy meters for large buildings, such as factories or office buildings, which perform measurements of different locations, implementing the sub-metering concept. These systems are usually complex, since the meters are all interconnected, forming a network, and all of them report their

measurements to a central-server, making the data acquisition a complex challenge in these systems. These systems are known as EMS, and will be further detailed in Section 2.1.4. The communication protocols used in EMS are presented in Section 2.3, to enable a better understanding of how measurement reports are made.

2.1.4 Energy Management Systems

EMS provide a set of tools to monitor the energy consumption of a building, allowing building managers to obtain overviews and relevant estimates. With such information, it is possible to take action towards reducing energy consumption and operational costs, without affecting the comfort of people. In order to do so, EMS perform three main tasks:

1. Gathering energy data
2. Interpreting collected data
3. Presenting reports

Regarding energy data gathering, we can classify data by three source types:

1. Energy meters, which measure energy consumption, as presented before.
2. Environment sensors, which measure environmental factors, such as weather aspects and building occupation. These factors are highly related with energy consumption. In a highly populated building, a higher energy consumption is expectable, especially when the weather is too hot or too cold, affecting the occupants' comfort.
3. Equipment status sensors, which enable EMS to obtain information about how equipments are operating and in which conditions. These measurements can also be correlated with energy consumption, to predict electricity consumption.

The combination of these three data sources allows EMS to correlate information, thus allowing building managers to better understand how the energy is being used and why, by interpreting the collected data. The combination of an environment sensor, e.g. a temperature sensor, with an energy meter that reports the energy measured in a room, can be easily correlated. For instance, if the temperature is too low and then starts rising at the time the energy meter registered a peak of energy demand, this can be understood as the heat radiator being turned on. The same correlation can be performed with light sensors, and many other equipment. The gathered information is then used to produce reports, which present the obtained results.

An EMS can encompass four types of energy meters: electricity, gas, water and enthalpic. In this document, the presented solution is for electricity, but it will be designed with expandability in mind, thus including tools that can allow additional functionalities, and possible measurement all energy types. In EMS, energy meters measure the energy in the locations where they are installed, and report their

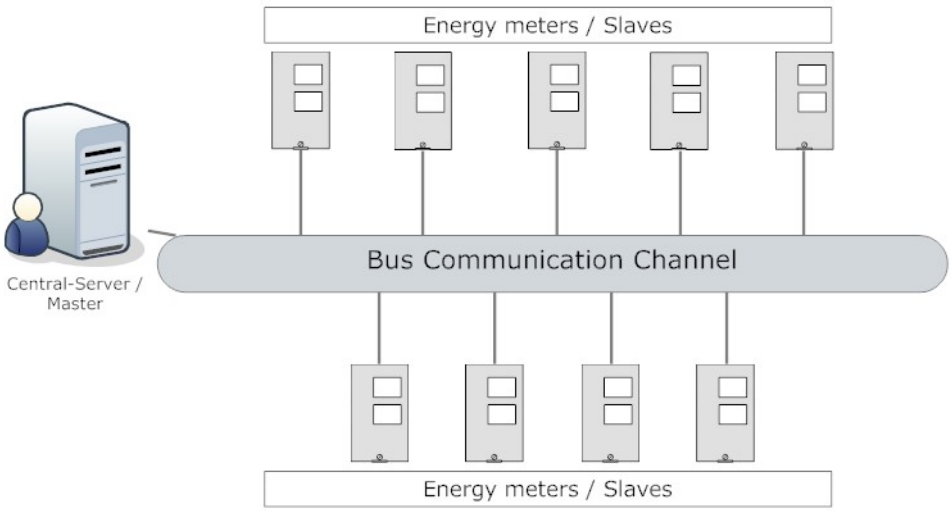

Figure 2.3: EMS Block Diagram. On the left, the central server that collects all the data and performs operations to produce reports. On the right, the energy meters that report the energy consumption to the server.

results to the system, periodically or by demand. The system collects all the information from those multiple energy meters, which is then used to create reports, which give the users or building managers a better notion of how to act towards reducing energy waste.

EMS are already available in the market, but they are still too expensive to be widespread and adopted by organizations or residential users. In fact, such an investment usually takes some years to reach a break-even point and become profitable, which can influence the adoption of this type of solutions.

Since the goal of this document is to present a low-cost energy meter, we will only focus on the first task of EMS: gathering energy data. This problem can currently be solved, as will be showed in later sections. Having the problem of gathering energy data solved, the intelligence of the system is provided by software on a main server. Therefore, it is possible to develop not only a smart low-cost energy meter, but also a low-cost Energy Management System.

An EMS can be represented in a simple block diagram, as presented in Figure 2.3, for a better understanding of the system's architecture. There are also some known models that can be applied, such as the Open Systems Interconnection (OSI) (ISO/IEC 7498-1) model for networks, as represented in Table 2.1. In this section, we will try to merge both of these ideas, in order to give a more complete view of the system.

The topology presented in Figure 2.3 is a typical block diagram of an EMS. The energy meters therein represented are responsible for collecting, storing and transmitting the energy consumption to the central server, on demand. Regarding Table 2.1, this represents the Data Acquisition process (Layer 1).

The information is transmitted through the bus communication channel. Typically, an EMS' communication channel is wired and shared by all participants in the network. They are all connected to the same link, so only one can transmit information at each time, in order to avoid transmission collisions.

In this case, the messages are exchanged in a half-duplex master-slave topology. This is the reason why the energy meters in Figure 2.3 are called slaves, as they only transmit when the master asks for information. Regarding Table 2.1, this corresponds to Layer 2: Data Transmission.

The master and the central server in Figure 2.3 can be also divided in two different blocks, if needed. This way, the central server can be responsible only for the energy-related operations, while the network master is responsible for the bus network. When the central server wants to collect information from the meters, it sends a message to the network master, which in turns requests data from the energy meters, one at a time, and reports the results back to the central server.

The central server, represented on the left, is the brain of the system. It gathers information from several energy meters by requesting the meters stored data and storing the values reported by them on a database, after a correct data interpretation (Layer 3). This server is also responsible for presenting reports to the building managers about the energy consumption (Layer 4) in order to provide a global view of energy consumption in the building, which assists them on managing energy in a more informed and efficient way, thus enabling the reduction of energy waste (Layer 5).

Layer 5	Energy Optimization
Layer 4	Performance Evaluation
Layer 3	Data Interpretation
Layer 2	Data Transmission
Layer 1	Data Acquisition

Table 2.1: OSI-like layers of an EMS adapted from [MCS+10]. Each layer is independent and groups similar operations.

2.1.5 Data Acquisition Frequency

Data Acquisition is the process responsible for gathering data, in this case, regarding energy consumption. The adequate frequency for meter data reporting does not have an exact value. Usually, the acquisition frequency can be configured, but it is limited by the communication channel. When several meters are installed in a bus channel, where meters can only be accessed one at a time, the time needed to communicate with all of them can amount to several minutes, therefore, many EMS solutions reading all the meters in a bus with a periodicity of around 15 minutes. Using other communication channels, such as the mesh network described in Section 2.4, the amount of time required to read all meters can be significantly reduced. In turn, reading all meters in a smaller amount of time allows a better view of the energy consumption, and can thus result in a more accurate data correlation.

2.1.6 Sub-Metering

Energy meters can be placed in any point of the circuit that needs to be measured. Sometimes, installing just one meter in a building room is not enough to obtain reliable energy consumption information. The solution for this problem is installing more meters, to measure some points of the circuit more accurately. The concept of sub-metering can be better understood in Figure 2.4, which depicts sub-metering in an

office center. In an office center, the energy used by each office room is different, depending on several factors, e.g., its usage or the equipment that is turned on. Using a single global energy meter, is not enough to understand how energy is being used, but performing sub-metering with an additional energy meter in each office, allows a better knowledge of how the energy is being used, and were it is potentially being wasted.

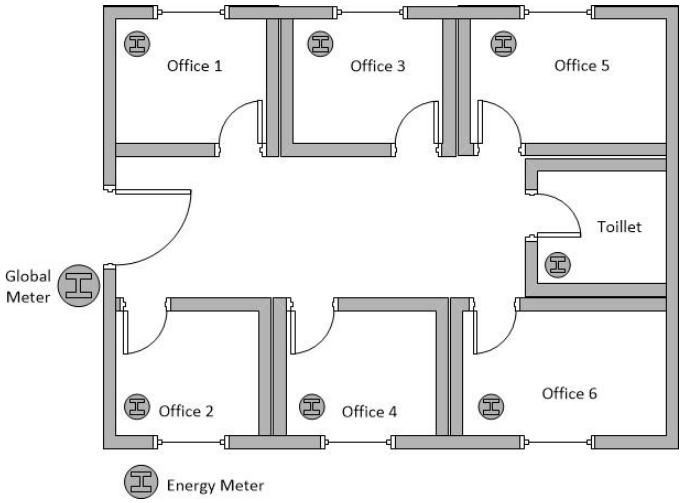

Figure 2.4: Sub-metering performed in an office center. Outside the office center, a global meter accounts for the global energy consumption of all offices. In each room, an energy meter tracks the energy usage of that room.

Usually large buildings are logically divided into several different spaces. For instance, a school can be divided into spaces such as common areas, classrooms, and faculty offices. It is important to know where energy is being used in order to take localized action to improve its usage. For instance, in a common area we can have bars, restaurants, and toilets, that may have a constant consumption that can be improved; on the other hand, classrooms and teacher departments are groups of rooms that can also be inspected in more detail, in order to realize where the energy is being consumed and wasted, e.g., not wasting energy in lights or heating when a classroom is not being used.

Sub-metering can also be used to identify which type of logical functionality is consuming more energy. For instance, in the previous office example, the HVAC system and the lighting system may be the two main causes for energy consumption, but more devices with other functionalities can be compared. It is also possible in these cases, to know which logical functionality inside a given office is using more energy.

The implementation of sub-metering may also bring many advantages in several different levels, such as increasing user awareness on the energy consumption issue, and more easily promoting among them good principles and practices for energy saving.

2.1.7 Data Quality

Poor data quality can lead to ineffective or wrong decisions. If an EMS cannot report a correct energy usage, it can lead the managers to take wrong decisions about energy management, and thus be a

useless system. The integration of different metering and measuring technologies required by most EMS can also be a reason for poor data quality. In fact, in an EMS there are often many different meters, which store information internally in different ways, and use different types of protocols or instructions to access data over a possibly noisy communication channel, which in turn that can corrupt the data. However, the EMS may obtain a valid value and remain unaware of its inaccuracy, thus delivering poor results to the energy managers.

Data quality has several dimensions [BS06], all of which must be taken into account to fully characterize it:

- **Accuracy** is related with precision. Some problems regarding accuracy, besides the meters' precision, is if a meter is placed in one part of a building, and the system has it registered as being in another physical place. Another problem that is considered in accuracy is the presence of duplicated information in the system. In some cases, such as in energy metering, this duplication can lead to information about abusive consumption in a given point of the building, caused by wrong data transmission due to a problem in the network. Usually, these problems can be easily detected.

- **Consistency** is related to data evaluation in order to know if it is valid and makes sense. A problem of consistency is when the system's data is valid but senseless. For instance, if a meter reports a value that is valid and accepted by the system, but lower that the previous one reported, we can assume there is a consistency problem, since meters are cumulative. It is normal to break this rule if the meter reaches the end of its internal counter and resets it back to 0, but this exception can actually create even more confusion if this kind of error occurs. For instance, when a reported value is lower than the previous one, two things may have happened: It could have been a consistency error, the reported value must be rejected by the system, or the meter's counter has reached is limit and been reset back to 0, being a valid value. These problems are usually harder to detect, compared with accuracy problems.

- **Completeness** is a dimension that considers that the data is completed only if all the related data is available. In energy metering, some meters can have more features and produce more information than others. In this case, it is imperative to know the exact data produced by each, in order to enable the detection of whether a missing value in the system is due to the lack of that feature in a particular meter or due to an unavailable chunk of information, which can be caused by an acquisition problem.

- **Timeliness** is related with the time during which a piece of information can be considered valid. When the energy meter is reporting the data, by the time the information enters the system, more energy has been consumed, meaning that the information is already outdated for a real time presentation. It is important to understand for how long this information can be considered valid. This is not a simple question, since ideally information would be immediately reported to enable real-time knowledge about the system's state, but the price trade-off for this to happen must also be considered.

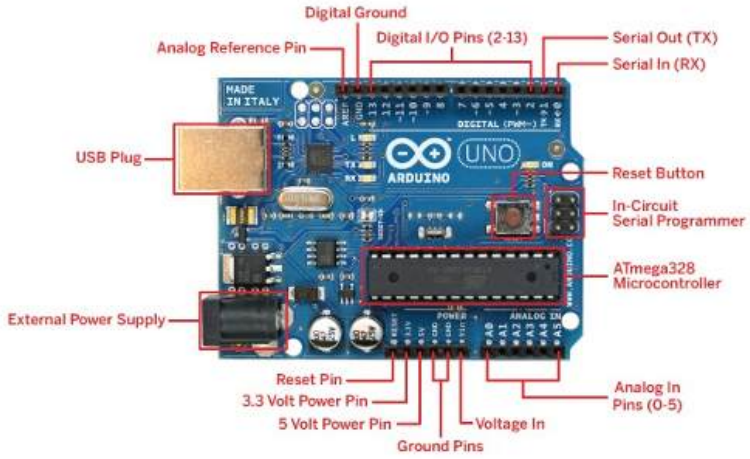

Figure 2.5: Arduino UNO board. On the left, the plug for the external power supply and the USB plug that can also provide power, which is additionally used to send the program code to the ATmega328 microcontroller, located on the right. On the top there are digital I/O, and on the bottom analog inputs are presented. In these ports we can connect modules and shields, which can also be powered using the 3.3 or 5 volt power pins and ground pins, located on the bottom. The communication between modules/shields or other devices, can be performed using the Serial Out (TX) and Serial In (RX), located on the top right.

2.2 The Arduino Platform

2.2.1 Hardware Aspects

The Arduino team claims that "*Arduino is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software*" [Ard13], and there are, in fact, many interesting Do It Yourself (DIY) projects under development based on it.

Physically, an Arduino is a Printed Circuit Board (PCB), as depicted in Figure [Ard13], consisting of an Atmel microcontroller and ports for input and output signals. Being open-source, the Arduino schematics are available and were therefore used to develop our own PCB, as described in the future work section .

According to Maik Schmidt, with a regular computer it is very difficult to control sensors and actuators, "*but using an Arduino, it's a piece of cake to control sophisticated and sometimes even weird devices*" [Sch11]. Another author, Michael Margolis, claims in his book that "*These components can be sensors, which convert some aspect of the physical world to electricity so that the board can sense it, or actuators, which get electricity from the board and convert it into something that changes the world*" [Mar11].

Microcontroller

The Arduino platform provides an Atmel microcontroller, which can be programmed in the C++ language, using software that will be described later in section 2.2.2. Depending on the used microcontroller, an Arduino model can vary on:

- **Port numbers** are used to sense or control the environment. There can be input/output and

digital/analog ports. This will be detailed in section 2.2.1.

- **Serial ports** are used for communication. This will also be further detailed in section 2.2.1.
- **Flash memory** is used to store the application code. Similar to a computer's hard disk drive.
- **SRAM memory** is used to store the application variables on the fly, similar to the RAM in a computer. This memory is volatile, being reset every time the Arduino boots.
- **EEPROM memory** can also be used to store the application variables on the fly. The main difference from the SRAM is that this is a non-volatile memory, meaning that the stored values are always available for the application, even when the Arduino reboots or the software is updated.
- **Clock speed** is the maximum microcontroller's working frequency. Similar to the computer processor speed.

Selecting the microcontroller is a major choice when prototyping, since each one has different characteristics, and depending on the dimension of the project, a simpler Arduino microcontroller may not be enough. A simple example of this is choosing an Arduino that doesn't have a clock speed high enough for a given task, or doesn't have enough flash memory to store all the application code. Table 2.2 presents some examples, for a better understanding of this problem.

Model	Uno	Leonardo	Mega
Microcontroller	ATmega328	ATmega32u4	ATmega1280
Serial Ports	1	2	4
Digital I/O Ports	14 (6 PWM)	20 (7 PWM)	54 (15 PWM)
Analog input Ports	6	12	16
Flash Memory	32K	32K	128K
SRAM	2K	2.5K	8K
EEPROM	1K	1K	4K
Clock Speed	16MHz	16MHz	16MHz

Table 2.2: Main differences between Arduino Models. Each one has a different microprocessor that provides different resources.

Arduino ports

As mentioned before, the Arduino offers the possibility to sense and control in a very simple way. This is possible by using the Arduino ports. The main difference between them is that some are digital (D), while others are analog (A). Regarding the digital ports, they can be input or output ports and some of them can also produce an analog value output, using a technique called Pulse Width Modulation (PWM) with an 8 bit resolution, simulating an analog value with quick variations of the digital output port. Regarding the analog pins, they are simple input pins with 10 bit resolution, allowing to sense the environment. In this thesis, some of these analog pins are used to measure the electrical consumption, as will be explained in Section 4.1.3.

Some of these ports also provide communication, by using the Serial Peripheral Interface (SPI) protocol in pins D11, D12 and D13, or the Inter-Integrated Circuit (I2C) protocol in pins A4 and A5.

These protocols can be used to connect expansion modules, which will be detailed in the next section. The SPI allows full-duplex communication between the expansion modules and the Arduino, being the Arduino master of the communication bus that can support multiple expansion modules. For each one, a digital pin, called “chip-select" (CS), must be used in order to enable that specific module to listen to the Arduino requests. For this reason, only a limited number of expansion modules can be used. Regarding this thesis, this bus was used to connect the Ethernet expansion module, and also the Radio Frequency module, further detailed Section 4.1.3. In the I2C, this limitation is overcome by using only the A4 and A5 pins. The I2C is a half-duplex communication bus, that specifies the address of the expansion module along with the request. In this thesis, this bus was not needed, however it can still be useful in the future, as detailed in section 6.3, for future work.

Some digital ports can also be used as interrupt ports, meaning that can trigger an action is the voltage level change during the execution. The Arduino Uno uses the D2 as interrupt 0 (INT0) and the D3 as interrupt 1 (INT1). An usage example is an expansion module requiring the Arduino microcontroller intervention, therefore also connected to an interrupt pin. Since the Arduino Uno only has two interrupt ports, only two expansion modules using interrupt pins can be connected at same time.

The Arduino also offers serial port communication (in pins D0 for Rx, D1 for Tx), similar to a computer serial port, which can be used by legacy devices that do not communicate using SPI or I2C.

Expansion Modules

Being a open-source platform, the Arduino has many different sensors and actuators developed especially for it. This gives us a big advantage and an incentive for prototyping. Some of the modules can even be attached directly to the Arduino PCB, by using the provided ports to fix both parts, called *shields*. “*Shields ... enhance an Arduino’s capabilities ... adding Ethernet, sound, displays*" [Sch11]. These modules and shields can be combined, creating a tailored product with all the features we need. In order to use an expansion module that requires a specific microcontroller, a library containing software code to communicate with that microcontroller must be used. This will be detailed in section 2.2.2, but for now consider that each expansion module has their specific microcontroller, as Arduino does, and in order to communicate with it, a specific library is needed.

There are many different modules for Arduino, and some of them produce the same features, however they do it in different ways. One of the modules used in our solution is the RS-485 Breakout board. This expansion module converts the serial communication signals produced by the Arduino Serial port, into RS-485 signals used in the RS-485 bus, thus allowing communication between the Arduino and other products that can communicate using a RS-485 connection. This will be detailed in Section 4.1.3.

Another expansion module example is the Ethernet module, which provides Internet connection, allowing remote access to the Arduino. The official Arduino store sells a shield which uses the microcontroller W5100. In order to use it, the Arduino IDE already provides the Ethernet library, making the process easier. However, there are also low-cost alternatives to this official shield. As you may understand by now, the microcontroller is the main key in the Ethernet shield, therefore, other expansion modules that allow Ethernet connection using this microcontroller can be used. Despite of not being

provided by the official Arduino store, they work exactly like the official one, since they have the same microcontroller. An even lower-cost solution is not using this microcontroller, but instead using a cheaper one that also provides Ethernet connection, for instance, the ENC28J60. Being a different microcontroller, it also uses a different library, and its connection to the Arduino board will likely be different. This module is used in our solution and will be detailed in Section 4.1.3.

Another very important expansion module presented in Section 4.1.3 is the RFM12b, also used in our solution. This low-cost module provides wireless communication between different Arduino boards. In order to provide all the features we needed, we developed a software library, further detailed in Section 4.3.3.

2.2.2 Software Aspects

The Arduino platform also provides a software to program the Arduino microcontroller, as depicted in Figure 2.6. These instructions are written using the C++ programming language, which helps the development of programs (officially called sketches). The libraries provided by the Arduino IDE, simplify the creation of simple features such as reading or writing on a port, and it is also possible to import other libraries created by Arduino users for their projects, which helps the development of other users' projects. When buying a new Arduino expansion module for a project, the retailer provides the library containing the source-code, as well as some sketches to demonstrate how it works. However, being an open-source code, users can create their own libraries with some special features, or remove unused features to save flash memory, and then share it with the Arduino community. This open-source culture enables and encourages the creation of simple new products with the potential of truly improving people's lives.

Besides the official software provided by Arduino, there are other alternatives that can improve productivity and also create a final product. Some of those were used in this thesis and are presented next.

Arduino development environment

The major problem with the Arduino IDE is the lack of several features that programmers are used to. In the official editor, it is not possible to change the background, text or highlights colors, and it is not the best option to edit or create library files. A solution for this problem was using another text editor that provides the upload feature as the official IDE does. The Sublime Text 2 [Sub08] with the Arduino plugin [Ard12] offers all the common features that programmers are used to, including viewing the same document in multiple windows, renaming a variable in the whole document at once, and uploading the code to the Arduino board, with better debug information at compilation time. This was our first choice editor, as presented in Figure 2.7.

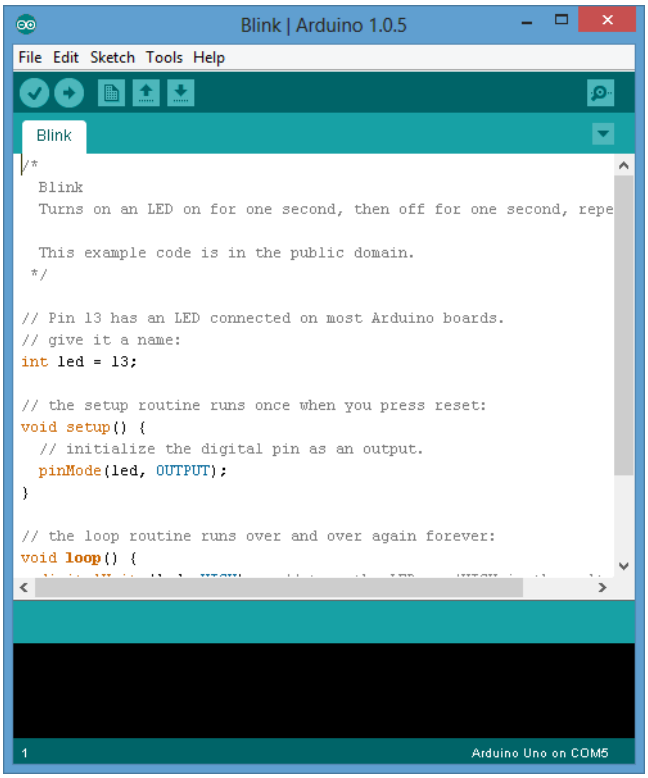

Figure 2.6: Official Arduino IDE screenshot. On the top-left corner, several tools are available to validate and upload the developed code to the Arduino board. On the top-righ corner a button shows up the serial console for debugging purposes. In the middle the sketch code is presented, and the bottom presents a console to feedback about error messages and upload status.

Terminal Viewer

Besides the official Arduino IDE being fully functional for a single Arduino, when working with several, the terminal window only shows one at the time, and to change it, we have to navigate in the menu bar. As this is not very practical, a solution could be to run multiple instances of the program, but it does not allow it. A possible solution is to use the Sublime Text 2 with the Arduino plugin, which can show different terminal windows. However, it can only send commands to a single one, which makes it good for viewing only. The solution for this problem, is running a portable version of the Arduino IDE software, which allows multiple instances and in each one we can have a different terminal window to view and send commands, as presented in Figure 2.8. This could also be done by using other terminal software such as Putty, but this option was not adequate, as it cannot send a string as an input command, but only single characters.

Creating circuits

Prototyping a product often involves combining several expansion modules and electrical circuits connected to a Arduino. These connections are not represented in the sketch code, but they may also be shown when documenting or presenting it. An example of such is presented in this document, where the system connections are detailed in Section 4.1.3. The Fritzing software, presented in Figure 2.9, is ideal

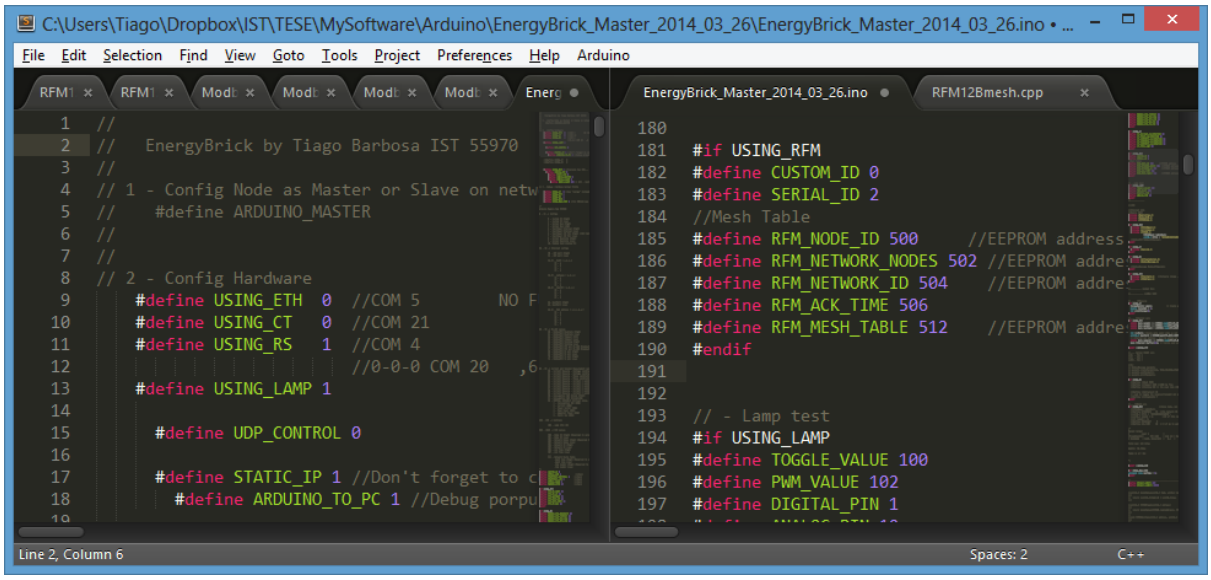

Figure 2.7: Sublime Text 2 with the Arduino plugin was the mainly used text editor during this thesis. It provides better features than the official Arduino IDE. In this image, part of the EnergyBrick code is presented, using the highlight feature, and divided in two windows. The implemented libraries are also open in other tabs.

to present a real representation of connections as you see them in reality. This way, it is easy to understand and rebuild the prototype from scratch. This software can even present the electrical schematic, produce the PCB board and order it. This way, a simple prototype can be made in even less time.

Creating parts

When prototyping with Fritzing, sometimes the expansion module that we are using is not available in the default part list. To solve this problem, the design can be obtained from the vendors page, or even in the Fritzing webpage [Fri13]. However, the extension module may not be designed yet for Fritzing, but it is possible to do it ourselves. The Inkscape [Ink03], presented in Figure 2.10, is a software that can be used to design those missing parts, and then import them to Fritzing. While developing this thesis, we had to design the ENC28J60 Ethernet module and the RS485 Breakout board.

2.3 Energy Metering Protocols

Protocols define a set of message formats and rules for their exchange, with the aim of facilitating communication. In energy metering, communication between meters and data aggregators is achieved through protocols, for obtaining energy information in an automated way. After being automatically gathered to a single point, this information can also be used to create reports, useful to improve decision making. These reports give managers a complete view of the energy use in an very quick and efficient way.

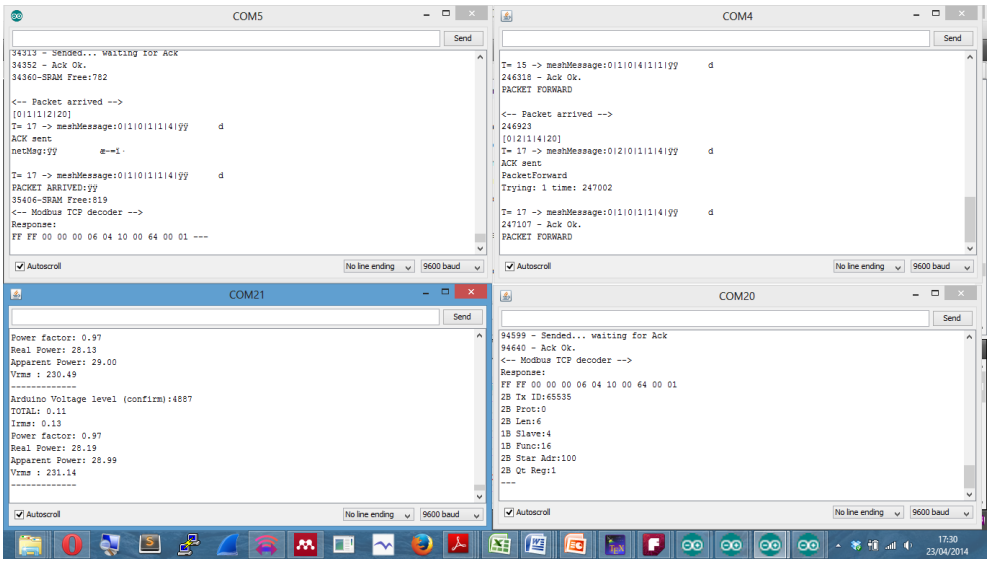

Figure 2.8: Illustration of multiple terminals using the portable edition of the Arduino IDE. With this edition, it is possible to have multiple instances opened at same time, each one showing a different terminal.

2.3.1 Modbus

Modbus [Mod05] is a serial communication protocol developed by Modicon (now called Schneider Automation [Ele81]), for industrial control systems. It was invented in 1979 for communication with Programmable Logic Controller (PLC) systems, aiming at controlling machinery in factories and automating processes with software [Mod12]. Even today, it is one of the most used protocols, and has been maintained since April 2004 by an independent group called Modbus IDA [Mod04, GP07].

This protocol defines the application layer of the OSI model (layer 7), providing Client-Server communication between devices connected on different types of buses or networks. It also standardizes a specific layer 2 protocol on serial line to exchange Modbus requests between a master and one or several slaves [Mod06]. Since the protocol defines the Protocol Data Unit (PDU) messages at the application layer, any network architecture can be used, as long as it provides communication between devices, e.g., connection usins a serial line RS-485 (Modbus RTU) or an Transmission Control Protocol (TCP) Ethernet (Modbus TCP), for which port 502 is reserved [MI06, Mod12]. Both RTU and TCP versions of the protocol have the same PDU, which makes the conversion between both easier, as presented in Figure 2.11.

The Modbus is a master-slave half-duplex message control protocol. A single master sends commands to the slaves, using the function codes defined in this protocol. Whenever the master asks for something and a slave replies to the master, this communication is half-duplex, therefore the communication channel can also be half-duplex. In the beginning, this communication channel (Layer 1 on OSI model) was provided by serial line, and at that time, the most used protocols for serial line were RS-232 and RS-485. In this case, only the RS-485 was appropriate for the physical layer protocol in the Modbus architecture, because it is point-multipoint half-duplex for long distances (up to 1200 m), while RS-232 is point-to-point full-duplex for short distances (up to 15 meters). However, since Modbus de-

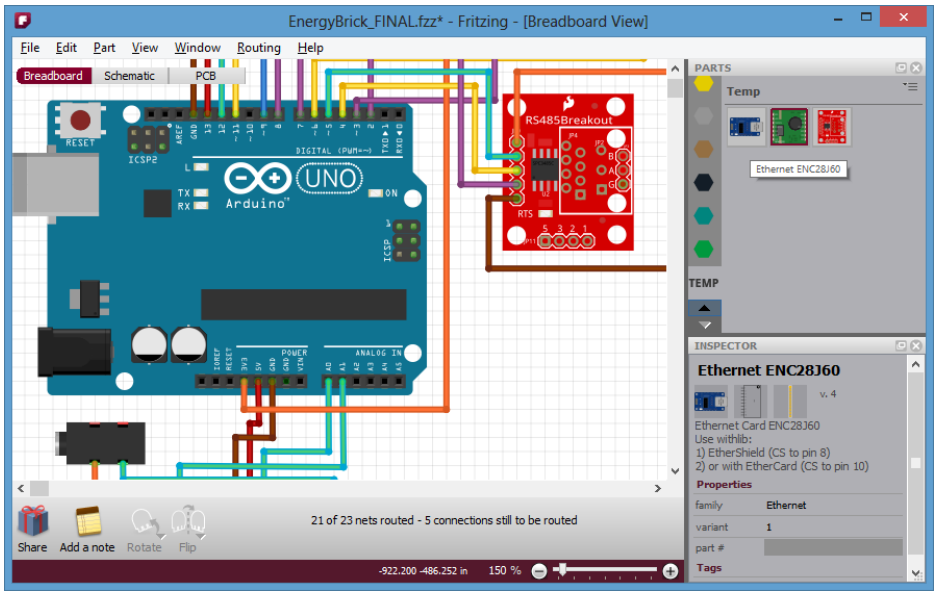

Figure 2.9: Fritzing software, used to develop the EnergyBrick connections. This screenshot presents part of the connections used in the EnergyBricks.

fines an application protocol, independent of the communication channel, any protocol that can provide communication between devices is a valid option, and therefore, TCP/IP can also be used.

One limitation of the Modbus protocol is that due to its master-slave architecture, it only supports 247 device IDs for slaves [Mod06]. For small installations, the accuracy provided can be enough, but for larger ones this solution is not adequately scalable.

In a communication channel using RS-485, the maximum number of devices is not defined, instead, the standard defines the concept of *unit loads*. In the RS-485 protocol, a maximum of 32 unit loads is allowed on the bus, due to the expected voltage drops. It is also possible to use devices that consume only $1/2$, $1/4$ or $1/8$, and in this case the number of maximum devices will be 64, 128 and 256, respectively [Gin04]. To overcome this limitation in serial line communications, parallel networks are required, which implies the need of merging their data.

Another possible solution to overcome the problem is using Ethernet communication with the TCP/IP protocol, called Modbus TPC. In this protocol, the presented previous devices (which have their own Modbus IDs), are connected to a bridge, router or gateway, that has an IP address[MI06] and can be accessed through the 502 port. As presented before, both RTU and TCP have the same PDU, allowing the conversion between both protocols, however there are some differences, as the Modbus TCP packet adds a MBAP header, containing some extra control fields:

- **Transaction Identifier** uses a 2 byte field and identifies the Modbus transaction. This value used to correlating requests and responses.
- **Protocol Identifier** uses a 2 byte value that identifies the used protocol. In the Modbus case, the value is 0.
- **Length** is a 2 byte value that defines the number of following bytes after this field.

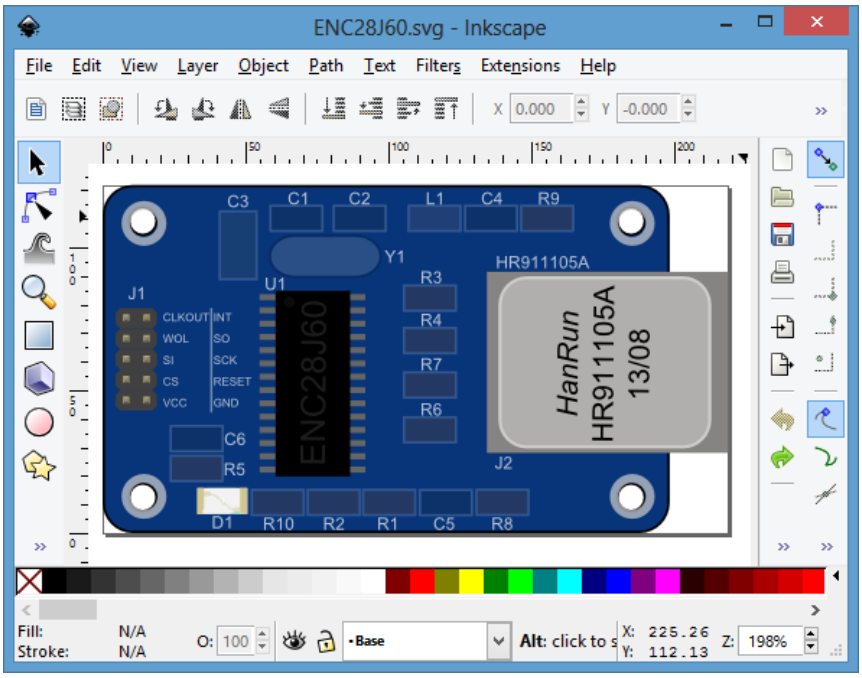

Figure 2.10: Inkscape software used to design the missing parts for the Fritzing software. In this image, the Ethernet ENC28J60 module is presented.

- **Unit Identifier** is a single byte value that identifies the remote slave in the serial line or the other bus.

Despite of the Error Check in the RTU not being present in the TCP version, this does not mean that data integrity is not being taken into account. Since a TCP channel is being used, it already has that control in lower OSI layers.

Using this protocol, the limitation for the maximum number of slaves is overcome using extra TCP gateways that forward the Modbus TCP packet into their own serial line network using the Modbus RTU protocol.

Another Modbus issue is that, being a protocol for general communication purposes between different device types, it simply defines the communication exchange messages, but provides no guidelines regarding the nodes' internal information storage. According to Modbus "*Each device can have its own organization of the data according to its application*" [Mod12]. As such, even if two devices do exactly the same, but were made by different manufacturers, their internal data organization can be different, saving the same data in different memory addresses, therefore the communication messages may need to vary in order to obtain the same data. This can also be a major problem if the administrator wants to include a novel device on the network. In the worst case scenario, this may imply updating the network master to allow a correct communication with this new device, due to its differences in internal data organization.

Usually, the internal data information is organized in memory addresses called coils and registers, as presented in Table 2.3.

A coil is a boolean bit variable, and a register represents a word. These coils and registers are used

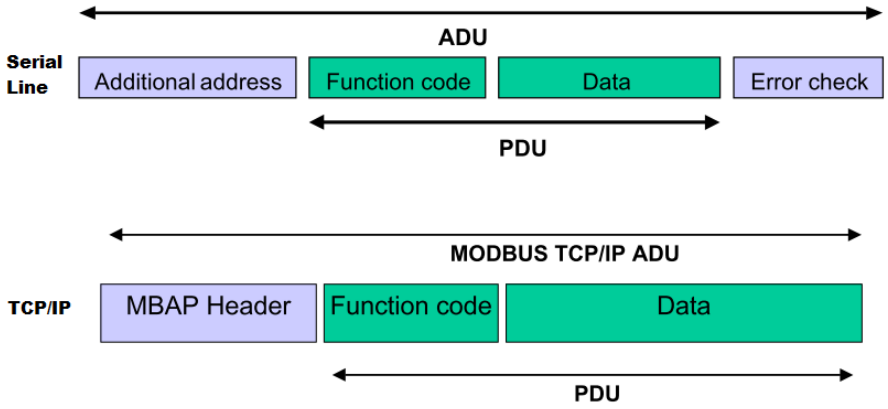

Figure 2.11: Modbus frames [MI06]. On the top, the serial line frame. On the bottom, the TCP/IP frame.

Primary tables	Object type	Permissions	Comments
Discrete Inputs	Single bit	Read-only	Provided by an I/O system.
Coils	Single bit	Read-Write	Can be altered by an application program.
Input Registers	16-bit word	Read-Only	Provided by an I/O system.
Holding Registers	16-bit word	Read-write	Can be altered by an application program.

Table 2.3: Modbus primary tables than can be used in devices and remotely accessed by software.

with different permissions, thus having specific terms associated: discrete inputs, which are read-only booleans; coils, which are read-write booleans; input registers, which are read-only integers; and holding registers, which are read-write integers [Mod12]. Depending on the application, these coils and registers can be used to store data, such as energy meter measurement in the case of energy metering, and then report it when needed. These are the bases of memory devices, but the implementation depends on the manufacturer, as Modbus only defines their communication protocol and not their functioning or integration. Therefore, the master must know how the information is stored in each device in order to use the correct address argument in the functions called by the Modbus protocol.

2.3.2 M-Bus

The Meter Bus (M-Bus) protocol was created in 1997 as an European standard (EN 13757-2 physical and link layer, EN 13757-3 application layer) for energy meter purposes, to measure not only electricity, but also water and gas [Gro11, M-B97]. According to the M-Bus User-group, "*the most important requirement is the interconnection of many devices (several hundred) over long distances - up to several kilometres*" [M-B97], and at the time, no bus system was available with those requirements.

Also according to the M-Bus User-group, the M-bus is not a network, and therefore does not need a transport or session layer. Regarding the OSI model, the M-bus only defines a physical layer (layer 1), a data link layer (layer 2), an optional network layer (layer 3) and an application layer (layer 7). The other layers are not used in this type of system, as this is only a bus and not a common Internet network [M-B97].

In this protocol, the network topology used is also a bus, as in the Modbus. This topology is con-

sidered reliable and cost-effective [M-B97], being a suitable low-cost solution for a wired network. Also according to M-Bus user-group, the best way to achieve a low-cost solution using cable, is to use cables that do not require shielding, such as telephone cable, ensuring that are installed without electromagnetic interference sources nearby [M-B97]. This bus solution uses *power-over-line* to power up the slave meters, which reduces the hardware needed for an energy meter, therefore reducing the cost of each, and consequently allowing a complete low-cost solution.

In order to power the slaves, this protocol implements its own bus solution. Only two wires are required, as in Modbus RTU when using RS-485, but the way of representing the bits during the transmission is different. The '1' is transmitted in the bus by the master with a voltage of 36 V, and the '0' is transmitted by reducing the voltage (in 12 V) to 24 V [M-B97]. The relevant value in this case is the voltage drop, which the slave must detect in order to decode the signal sent by the master. In the slave-master communication, the slave transmits the information by manipulating the current that it needs to be powered on. The '1's are represented by a constant current of 1.5 mA and the '0's are represented by requiring additional 11 to 20 mA [M-B97], as represented in Figure 2.12. When no one is transmitting, the slaves' current is a constant of 1.5 mA, and the output master's voltage is 36 V. As the system grows, and the distance between slaves grows with it, this voltage will be less that 36 V. However, this is not a problem, since the slaves only need to detect a 12 V drop in voltage.

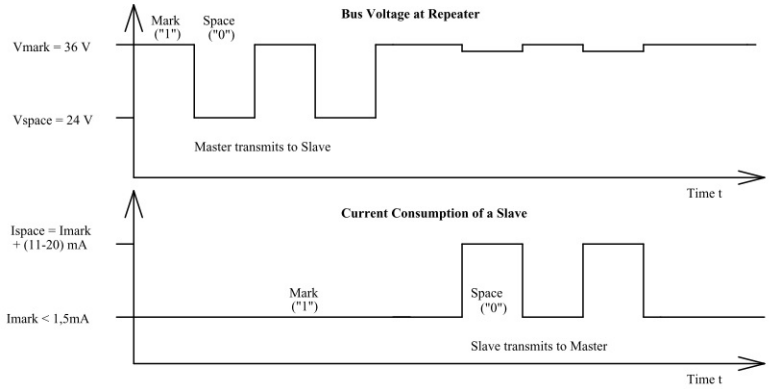

Figure 2.12: Representation of bits on the M-Bus, adapted from [M-B97].

The bus topology used in the M-bus system is slightly different from the one used in Modbus. In Modbus, the master is connected directly to the slaves, using RS-485 protocol for communication, but in M-bus topology, the connection is made through a *repeater*, typically connected via RS-232, and then the repeater is connected to all slaves, transmitting information using the technique detailed before. This repeater is responsible for powering up the meters (slaves) and converting the messages from the master (usually a PC) to the slaves.

The cable suggested for using as a transmission medium in this system is the 2-wire standard telephone cable. Its characteristics allow a maximum distance of 350 m between a slave and a repeater. This distance depends on the cable's resistance, which in this case is typically 29 ohms. The maximum distance can be increased by reducing the number of slaves (which is at most 250) or using a lower

Baud rate (which has a maximum of 9600, varying between 300 and 9600 Baud), but it cannot be more than 1000 m due to the cable requirements to power up all slaves. This encoding brings a problem: if one slave has a short circuit, all the meters can fail due to lack of power. In order to minimize this problem, it is common for meters to have batteries that enable them to continue working normally.

The representation of bits in the previously described line, allows the transmission of 8 bit characters, which are the base for the messages used in this protocol, called *telegrams* [M-B97]. The transmission of 8 bit characters in the cable is similar to RS-485. For each byte sent, it has a start bit, followed by 8 bits that represent the byte, then the even parity bit and finally a stop bit. The telegram messages exchanged between master and slaves and can have 4 different sizes, depending on the type of message that needs to be transmitted: single character, short frame, long frame, and control frame. Depending on the complexity of the instruction that needs to be sent, a certain message type is used, e.g., a simple acknowledgement is represented by a single character (8 bits or 1 byte). These messages also have an associated class that specifies their priority. There are two classes: class 1, which are priority messages, and class 2, which are normal messages. Class 1 messages must be transmitted as soon as possible. The slaves may have one message to transmit to the master, but cannot send it unless the master requests it. As such, to enable a slave to inform the master of an existing message, there is a field in the answer telegram that indicates if the slave has Class 1 data to deliver, and if so, the master should request the Class 1 data next.

Instructions from the master, regarding data requests to the meters, are sent as telegrams to the slaves, as happened with function codes in Modbus. As with Modbus, there are also many types of requests in this protocol, but with M-bus it is possible to request all the information from one meter in a single request, which eliminates the complexity problem for the master device.

2.3.3 Protocol Comparison

Modbus and M-bus are two different protocols, but both can be used to reach the same goal of communication in energy metering. Both protocols aim at providing a reliable and cost-effective way for energy metering using a wired network, supporting around 250 devices. Another common aspect is the transmission mode communication in half-duplex, meaning that both protocols use a client-server protocol. The byte transmission in the bus is also similar in both protocols. In order to send a byte, or character, both start by sending a start bit, followed by 8 bits, representing the byte or character, followed by a parity bit (optional in Modbus) and finally the stop bit (or two in the Modbus case, where if the parity bit is not usied, it is replaced by a stop bit). However, their physical communication is different. Modbus RTU (which uses RS-485 at layer 1) uses a simple representation of positive and negative voltage values to represent ‘0’ and ‘1’, while in M-bus a voltage drop and current variation technique is used. Both protocols can be better compared using the OSI model representation, presented in Table 2.4.

According to the Modbus Organization, the Modbus provides client-server communication between devices using the Modbus application protocol [Mod06], meaning that this protocol also defines the communication messages and function codes, using only the RS-485 to transmit the bits over the bus. In the

Layer	OSI Model	ModBus	M-bus
7	Application	Modbus Application Protocol	EN1434-3
6	Presentation	Empty	Empty
5	Session	Empty	Empty
4	Transport	Empty	Empty
3	Network	Empty	Optional
2	Data Link	Modbus Serial Line Protocol	IEC-870
1	Physical	RS-485 Protocol	M-Bus

Table 2.4: OSI model comparation between Modbus and M-Bus

case of M-Bus, the protocol defines how the bits are transmitted over the bus, as detailed in the previous Section, but, according to the M-Bus User-group, the messages exchanged in the bus are defined in layer 2 by IEC-870, which defines the transmission parameters, telegram formats, addressing and data integrity; and also supported in layer 7 by EN1434-3, which defines data structures, data types and actions. In the case of M-bus, it has the advantage of being able to request all the information from a device, thus simplifying the data collection process [M-B97].

Besides these similarities and differences, both protocols provide communication solutions in energy metering, but M-bus was created with a specific goal in mind, and therefore it's natural for it to have a special concern with the physical layer, being also able to power up the energy meters with the communication channel; and Modbus is a more general purpose protocol to provide communication between different kind of devices, therefore, as expectable, offering functionalities in higher OSI layers.

The network topology used in both protocols brings limitations to the number of devices that can be present in these networks (around 250 devices). To overcome this limitation, additional parallel networks can be used, but it is not an elegant solution. Additional networks also means the need to install additional cable, consequently raising the final price of this solution. In the next section, the mesh topology will be presented as a wireless and scalable solution for this problem.

2.4 Wireless Mesh Networking

The communication channel presented in the previous section was provided by a communication bus. It is usually a long cable that has to be spread throughout the places where devices are intended, making them hard to install. In this section, a better approach will be presented, using wireless technology with a mesh network topology. Wireless Mesh Networks (WMN) have emerged as a promising design paradigm for self-organizing and auto-configurable wireless networks [HL07].

In a mesh network, a node not only captures packets addressed for itself, but also participates on forwarding packets in the network that have other destination addresses. This feature is called *multi-hopping*.

A mesh network is an autonomous decentralized network that searches for the optimal routes in a multi-hop architecture. Each element in a mesh network provides the multi-hop feature, composing the backbone of the network (also called *backhaul*), and can additionally serve as an Access Point (AP) for

other clients. This new topology has several advantages, namely:

- **Easy deployment** since the network's backbone is provided wirelessly, no network cables are required to install it. Only a power source is needed, which is also cheaper to install.
- **Better coverage area** due to the easy deployment of new mesh nodes in a mesh network.
- **Failure proof** due to the ability to perform scans to find optimal routes. If a node fails, it does not break the network, since the communication can still be performed via another working route. In addition to this, each node is connected to several other nodes instead of just one, making this network more robust and less prone to failures. These features are key elements in mesh networks, since the nodes' mobility is the main motivation for this network topology, especially over WiFi.
- **Spectral efficiency** using multiple antennas. Radio antennas are cheap, but the radio spectrum is very limited for transmissions and must be carefully managed. In the case of WiFi, a mesh node can have more than one antenna (usually directive antennas), using also more than one frequency, meaning that the interference can be minimized inside the network itself, to maintain its performance as it grows. The maximum performance can be achieved using one antenna (pointed at another mesh node) for uploading in a given frequency and another antenna (pointed at a third mesh node) for downloading in another frequency, and additionally having a third antenna for legacy clients in 802.11, also working as an AP. In the case of ZigBee, devices usually have just one antenna, but use specific techniques to achieve spectral efficiency, as we will detail later on this document.
- **Low power consumption** while communicating in a mesh network. The clients do not need to increase power in order to maintain the wireless connection, as happens with 802.11, when only one AP is available. In the case of a moving device, e.g., a laptop, as the distance to an AP increases, the transmission power will also increase, thus spending more energy to maintain the connection. In the case of a mesh network, the client can automatically change to a closer AP, therefore saving energy. This feature is especially important in ZigBee, due to its low-power goal.

This concept of mesh network is relatively new in industry, and has as main objectives a flexible and extensible standard based on IEEE 802.11 [Bah06]. In the case of Local Area Networks (LAN), the standard 802.11s was developed, and for Personal Area Network (PAN), there is also the ZigBee standard developed with the aim of providing low-power and low-rate wireless communication for Machine to Machine (M2M).

Mesh topologies have military origins. In the battlefield, wireless communications are an essential tactical tool. Moving soldiers need to communicate with each other, and they cannot rely on a single point of failure, such as a single access-point (AP) used in traditional wireless (802.11) networks. With a mesh topology, as used in mesh networks, the installation of a large antenna is avoided. Such installation on hostile territory could reveal their positions and also be easily shut-down by the enemy. Each soldier radio is a mesh node, including the radios on vehicles. As they move, a tactical movement can force them

to separate from each other, forming sub-groups. In this case, the network automatically restructures itself, forming separated networks to maintain communications inside each sub-group, and restoring the original network when the soldiers get back together. This system was also designed to increase tactical situation awareness, by adding support for real-time data and video connectivity to individual soldiers and battlefield commanders [Mot08].

2.4.1 Mesh networking over WiFi

In the case of WiFi, an amendment to the original 802.11 emerged [IEE11], called 802.11s. The availability of such standard allows the mass production of interoperable mesh networks, i.e., with nodes from different vendors that can interoperate.

The main distinctive factor of WiFi networks is that AP nodes are not necessarily connected to a cable. “Mesh Dynamics" [Mes02] and “Rajant" [Raj13] both have a solution for video surveillance, with specific hardware and software, using a WiFi mesh network. There are also implemented and deployed solutions using mesh networks to provide public Internet access. Working examples of this are Motorola [Mot13] with Motomesh Duo already deployed in some countries, such as Japan, Brasil, Spain and USA; and also Cisco in Brazil, with a mesh solution for a municipality in Minas Gerais [Cis07].

2.4.2 Mesh networking over ZigBee

ZigBee is a worldwide protocol standard developed by ZigBee Alliance [Zig02] for low-power and low-rate wireless PAN, aiming at low-cost M2M communication between different devices from different manufacturers. Besides the common aspects of mesh networks introduced before, ZigBee was designed with the following aspects in mind: the use of unlicensed radio bands, which avoids costs in frequency licensing, and a mesh topology, which allows the network to grow just by installing nodes on the network.

Typically in M2M the required bandwidth is low, and the devices do not need to be constantly communicating, thus enabling the device to enter a sleeping mode to save power.

Profiles work as a common language for applications, just as the ZigBee protocol works as a common language in the network. Several products from different manufacturers that follow a standard can interoperate, allowing users to choose the combination that best suits their needs. A typical example of interoperability using ZigBee is the installation of a light bowl produced by a manufacturer, which can be turned on and off with a switch produced by another. The communication is possible due to this standard.

Regarding energy, a M2M solution that is especially interesting in this document is called “*ZigBee Smart Energy*" and is defined as a profile by the ZigBee Alliance. According to the ZigBee Alliance, “*this specification provides standard interfaces and device definitions to allow interoperability among ZigBee devices produced by various manufacturers of electrical equipment, meters, and Smart Energy enabling products*" [Zig11]. This standard was completed in 2013 and requires the use of the ZigBee specification.

To guarantee interoperability in ZigBee, products have to be “ZigBee compliant" and “ZigBee certified". According to the ZigBee Alliance, the first one is related to the radio itself, meaning that it is possible to acquire just a radio module to provide communication to a device or to replace cables, while the second is related to out of the box products that provide well known functionalities [Zig13a], following a given profile, and can also join in ZigBee networks using their own ZigBee compliant radio. These devices can interoperate with one another. In the case of ZigBee Smart Energy, there are already many ZigBee certified products [Zig13b].

ZigBee is a transport (layer 3) protocol in the OSI model, that operates over the IEEE 802.15.4 standard protocol, which defines the physical layer (layer 1) and media access control (layer 2), allowing the creation of mesh networks, as presented in Figure 2.13.

There are three different nodes types in a ZigBee network:

- **Coordinator (ZC)** is the first in the network, when the network is formed. There must exist one per PAN to form the network, it has to coordinate the entrance of new elements in the network by storing the network information, and it can work as a bridge to other networks. After the network is formed, it acts like a normal router, as will be explained in the next point. It must always be on (cannot sleep) in order to ensure a correct network functioning.
- **Routers (ZR)** are responsible not only for sending and receiving their own packets, but also for transmitting packets from other elements in the network, as normal routers do. They connect to the network through the coordinator or another router. They are the core of the network and can also store packets to deliver to nodes that may be sleeping by the time the packets arrive. This type of node must also be always on, to ensure the network’s communications.
- **End devices (ZED)** are the nodes who “sense" the environment. They are the only nodes that can sleep in a ZigBee network, and they connect to a router or to the coordinator. These sensors cannot communicate to each other directly, instead, they must use a router or coordinator if they need to. These sensors are usually battery-powered and installed in places that do not have electric power, which makes them more flexible, easier to install, and significantly cheaper.

Existing ZigBee device classes can be classified into Full Functional Device (FFD) and Reduced Functional Device (RFD). The difference between them is that the FFD can be configured as any type of node presented before, and the RFDs can only be end devices (ZED), therefore they can only communicate with the coordinator. This allows manufacturers to create simple RFDs with a specific function at a very low-cost. The ZC and ZED allow elements proposed in the 802.15.4, however the point-to-point nature of 802.15.4 only allows creating networks with a star or a tree topology. In contrast, the ZR was introduced by ZigBee enabling networks with a mesh topology, since routers can communicate between them and also with the coordinator. However, by doing this, the nodes in the mesh network can never sleep. The DigiMesh allows all nodes to sleep at the same time, with a time synchronization provided by an elected coordinator (since in DigiMesh only one node type exists) [Dig08].

ZigBee networks are also secure and reliable. Only authorised devices can join a mesh network. Each device has an unique ID that can be associated to a network, and only if the coordinator recog-

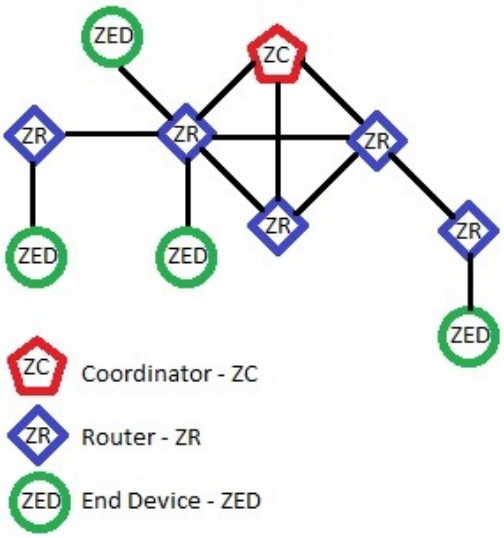

Figure 2.13: ZigBee Mesh Network representation, consisting of 3 different node types: coordinator (ZC), routers (ZR) and end devices (ZED). The coordinator controls the entrance of new nodes. The routers forward packets in the network. The end devices sense the environment and report the measurements.

nizes that node as belonging to its network, it will allow that node to join the network. The protocol offers security features such as encrypting messages with AES 128, and using three password levels in the network to exchange messages: one password between two devices (data link layer), another password for the entire network (network layer), and the last one for applications profiles (application layer). Another provided security feature is a message freshness validation mechanism to avoid replay attacks. This is an important feature, since these wireless networks can be used for automation proposes, and if someone with bad intentions captured a message and replied it, actions from the past, e.g., an action such as opening a garage gate could be reproduced by attackers.

There are ZigBee compliant module kits that provide communication to any device we may want to connect, or even to develop by ourselves, using development kits. An example of such a development kit is Arduino, which will be explained later in this document.

IEEE 802.15.4

The IEEE 802.15.4 protocol is used as a physical layer (layer 1) and media access control (layer 2) by ZigBee, as mentioned before. It defines point-to-point communication between a pair of devices, and was developed as a common platform for wireless communication. The standard describes the wireless transmission of bits, using a modulation technique known as Direct-Sequence Spread Spectrum (DSSS), which spreads the signal of a single frequency over several frequencies with less power, being more reliable in noisy environments. For instance, using this modulation technique, if the central frequency is jammed due to noise, the communication will still be possible, since the receiver is listening in several adjacent frequencies and not just in the jammed one. These frequencies depend on where the system is being used, since the radio spectrum is regulated, as shown in Table 2.5.

Frequency	No. of Channels	Region	Bit-Rate [kb/s]
868.0 - 868.6 MHz	1	Europe	20/100/250
902.0 - 928.0 MHz	10	USA	40/250
2.40 - 2.48 GHz	16	Worldwide	250

Table 2.5: IEEE 802.15.4 frequencies and corresponding radio spectrum allocation

To avoid communication collision between the nodes that compose the network, this protocol uses two different methods: Carrier Sense Multiple Access / Collision Avoidance (CSMA/CA) and Granted Time Slots (GTS). The CSMA/CA is a network multiple access method in which nodes avoid collisions with others that are already transmitting, by sensing the channel before performing the transmission. This method is used in best-effort communication, where the bit-rate is allowed to change over the time. GTS is a method to grant a Quality of Service (QoS) level to communications that need a guaranteed constant bit-rate. When using this mode, each time slot is reserved in the network for a specific node.

The IEEE 802.15.4 is a low-consumption protocol, which allows the devices to sleep 99% of the time (having low-duty cycles) and only transmit information when really needed. This feature is explored by the ZigBee and can extend the lifetime of a single AA battery for years.

The IEEE 802.15.4 is a Media Access Control (MAC) layer protocol, used by many other higher level protocols, being ZigBee the most famous one. Protocols such as Wireless HART and ISA-SP100, used in industrial automation, or IETF IPv6, to make devices available on the Internet, also use 802.15.4. There are also mesh protocols under development by companies which use 802.15.4 as a base to implement communication between their products, e.g., DigiMesh [Dig13b], a proprietary protocol solution provided by Digi [Dig13a].

Chapter 3

Related Work

There are many solutions available for energy metering. As presented previously, there are EMS that use a wired solution for communication, using the protocols presented in Section 2.3. However, other solutions are being developed offering a wireless communications that can be easily installed in a building. Some of this wireless solutions are already available in the market, using the ZigBee Smart Energy profile defined by ZigBee Alliance, as the ones presented in Section 2.4, while others are still in a prototype stage to be commercialized. Meanwhile, the appearance of open-electronic platforms ease up the development of low-cost hardware solutions, that can be shared as open-software projects, such as projects for measuring energy consumption.

The presented solutions can be classified in the following types:

- **Commercial solutions** that provides an energy metering service to the client, where a payment is applied. Some solutions are presented in Section 3.1.
- **Open-source solutions** which, is an alternative to the commercial solutions, where the details can be accessed by anyone, as a free knowledge. Some solutions and implementation details are presented in Section 3.2.

3.1 Commercial solutions

The energy metering concerns created an opportunity to implement solutions to save energy costs. Usually in this type of solutions, an external entity perform an energy audit in the company who hires the service, and presents a report of energy savings that is possible tocachieved wit their solution plan. In this section, some commercial solutions are presented to save energy costs.

Wi-J&D Wireless Energy Meter

The J&D Electronics [Ele94a] is an South-Korean company who produces tailor made solutions using high-precision current sensors. This company also has standard products and services such as the Wi-J&D Wireless Energy Meter [Ele94b] presented in Figure 3.1.

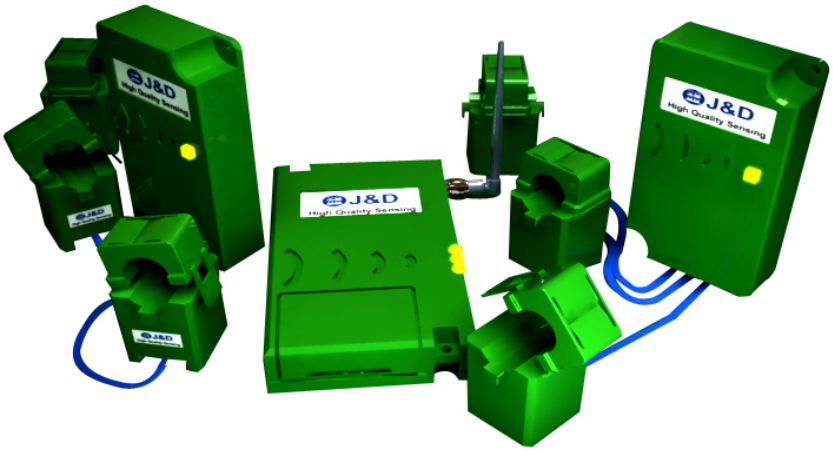

Figure 3.1: Wi-J&D Wireless Energy Meter. A small size and easy to install wireless solution.

This metering solution is easy to install since it uses 2.4 GHz Zigbee modules for wireless mesh communications, and split core CT to measure energy consumption, without interrupting the service. There are different devices in this solution, presented in Figure 3.2, were each one performs a specific task: a) The Wireless Energy Meter Node, performs the energy meaturing measurements that are reported to the b) Wireless Mesh Node, that forwards the packets in the mesh network until it reaches the c) Wireless Mesh Gate that using a d) RS-232 to TCP/IP converter sends the energy readings to the e) end-user interface.

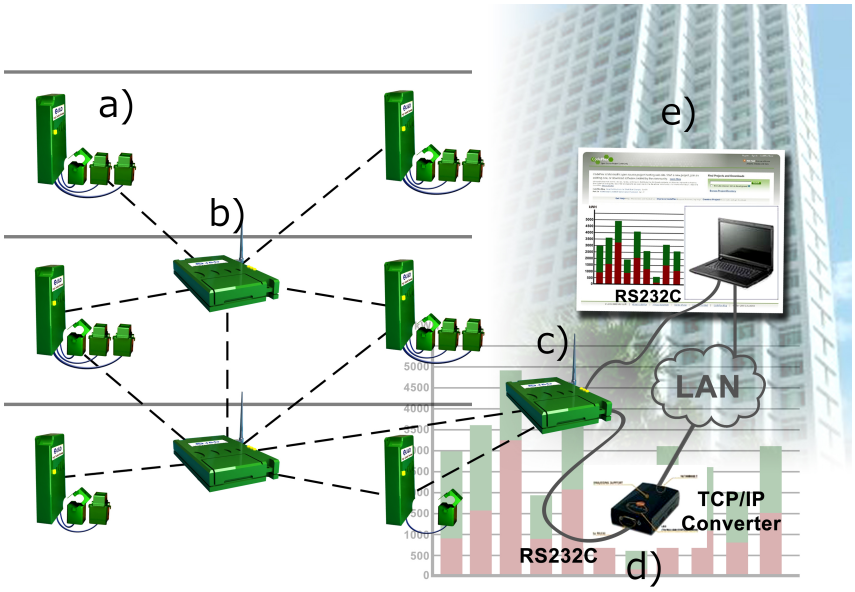

Figure 3.2: Wi-J&D Architecture composed by a) Wireless Energy Meter Node, b) Wireless Mesh Node, c) Wireless Mesh Gate, d) RS-232 to TCP/IP converter and e) Interface to the end-user

VerifEye

The VerifEye is Its an *easy to install and maintenance-free sub-metering solution* [Lev12] presented by Leviton [Lev89]. This solution includes metering of several energy sources: electric, water, gas and steam. The energy meters communicates using the ModbusRTU application protocol and reports the measured values using a radio frequency mesh network to a data collector who forwardsthem to the central server using a Ethernet connection, allowing the end-user to access the consumption data.

The Leviton System Architecture is presented in Figure 3.3

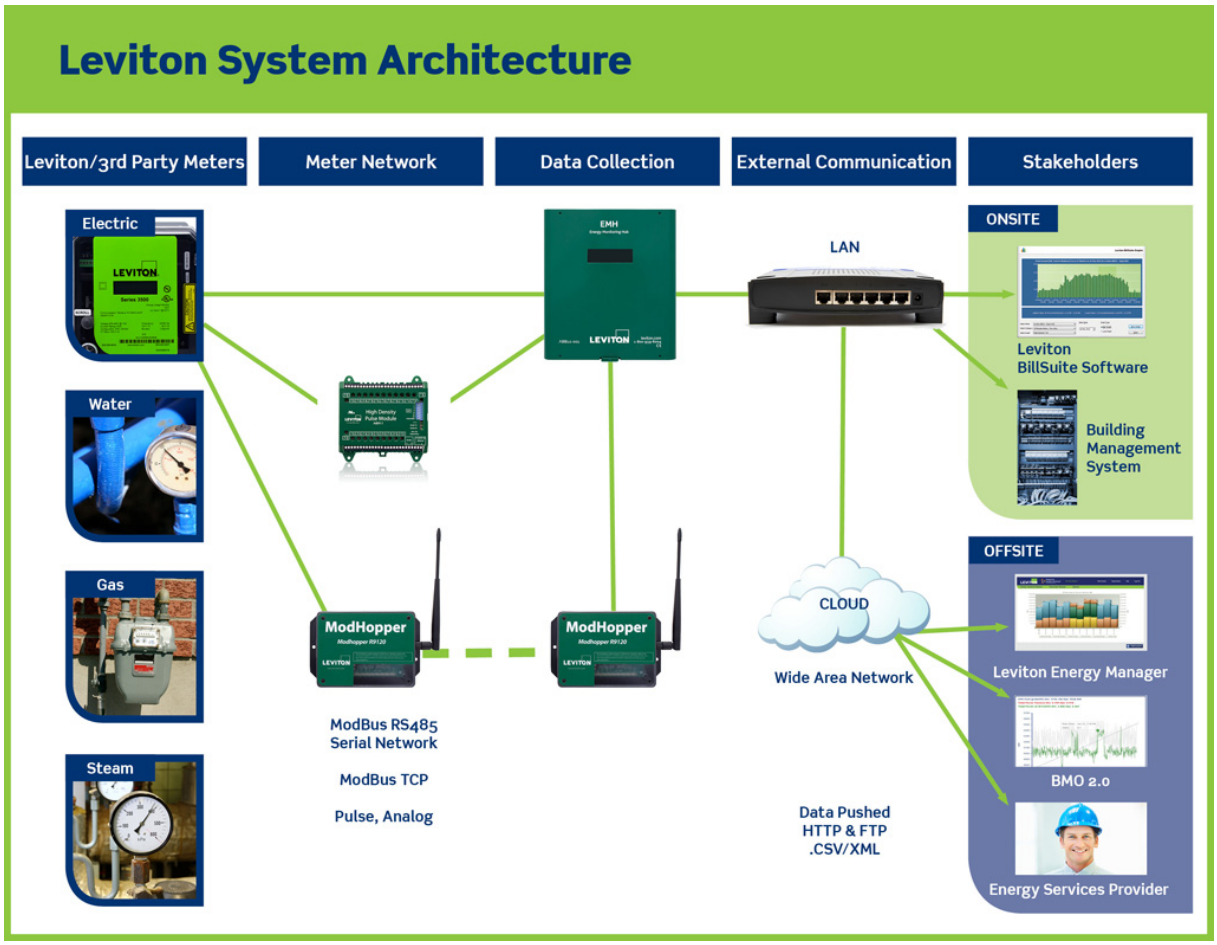

Figure 3.3: Leviton System Architecture for submetering of electricity, water, gas and steam.

To access the collected data, the Leviton provides a hosted meter monitoring software application: Building Manager Online 2.0. This online platform allows the end-user to access the consumption data without installing additional software in the computer. However, this online platform do not produce reports or notifications to the building manager. To enable those features, Leviton also provides a software solution that can produce several consumption reports *in just three clicks* [Lev12] called: Energy Manager. Besides being a more complete solution, this software has to be installed in the building manager's computer.

3.2 Open-source solutions

There are several Energy Management Systems available in the market, but they are expensive. In order to solve this problem, some open-source projects are being developed, increasing the pressure towards low-cost solutions. The open-electronics platforms enable the possibility of creating low-cost hardware to implement a energy metering solution which can be compared to commercial ones.

3.2.1 Hardware Overview

To better understand those solutions, a brief explanation about the hardware used to develop the solutions is presented in this section. The Figure 3.4 presents the multiple ways to develop a solution.

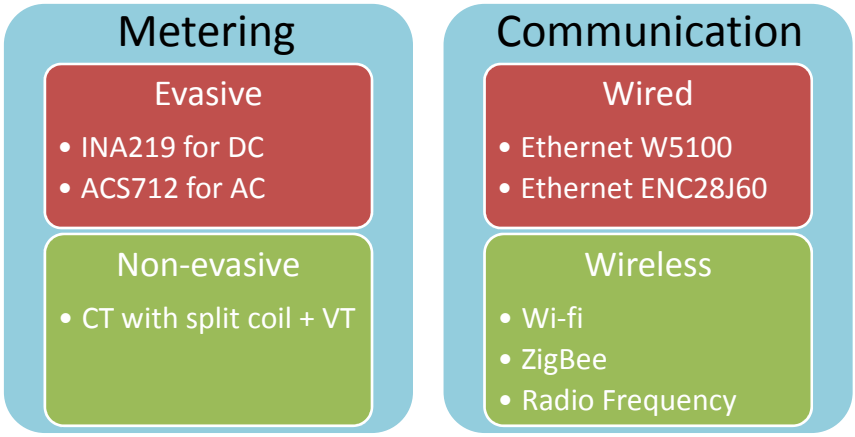

Figure 3.4: Hardware Overview to develop open-source energy projects

Metering

In order to develop an energy meter, the metering measure technique adopted in the following projects depends on the chosen hardware. There are two possible ways to perform energy metering:

- **Evasive** which implies breaking the power circuit to install the energy meter, being a more dangerous technique. To perform energy readings using this technique there are two possible hardware solutions:
  - **INA219**: This module is presented in Figure 3.5, where in a) the module is presented and in b) is connected to the Arduino. This module can be used for measuring DC such as solar panels, or other appliances that uses DC power through the usage of power supply. In the figure, a LED is connected to exemplify the usage of this module. This module reports the consumption to the Arduino using the I2C bus.
  - **ACS712**: This module is presented in Figure 3.6, where in a) the module is presented and in b) is connected to the Arduino. This module can be used for measuring AC and DC, being

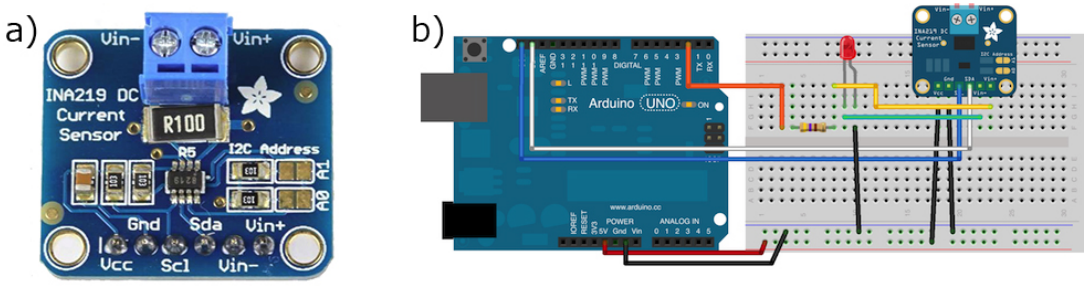

Figure 3.5: Current Sensor INA219 used to measure DC. In a) the module is depicted and in b) the connections to the Arduino are presented to measure the current consumption of a LED.

able to measure any appliance that needs to connect to the main power. This module reports the consumption to the Arduino using the analog ports.

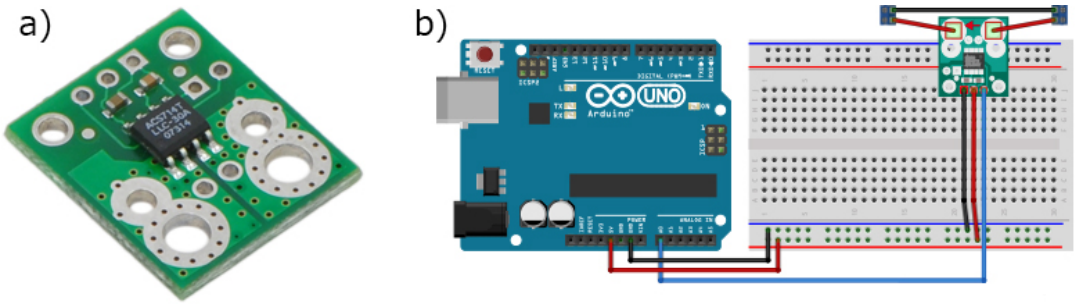

Figure 3.6: Current sensor ACS712 used to measure AC. In a) the module is depicted and in b) the connections to the Arduino are presented to measure an appliance connected it

- **Non-evasive** which does not implies to break the power circuit to install the energy meter, thus beafer. To perform energy readings using this technique a Current Transformer (CT) and a Voltage Transformer (VT) are used jointly with an electric circuit to perform measurements. In our solution this technique is used, ands will be etailed in Section 4.1.3.

Communication

In order to report the metering data to a central server, a communication channel must be established, and there are also several ways to perform it:

- **Wired** which implies installing an Ethernet cable, which can be a difficult task depending on the location where the metering will be performed. There are two hardware solutions to implement this type of communication:

  – **W5100**: This Ethernet microcontroller is used in the official Arduino Ethernet shield depicted in Figure 3.7. This shield also has a Secure Digital (SD) slot to store data, such as energy measurements. here are several hardware alternatives to the official shield using the same microcontroller, which the Ethernet works exactly the same way, but does not thetSD slot, and are not a shield,s, thuing less expensive.

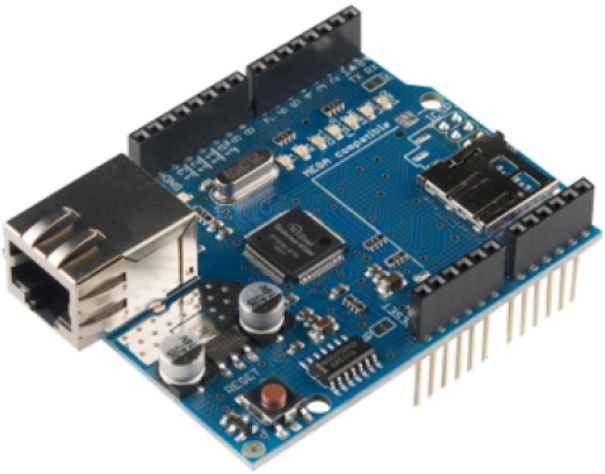

Figure 3.7: Official Ethernet Shield for Arduino. It uses the W5100 microcontroller. It also has a slot for a SD card to store data.

  - **ENC28J60**: This Ethernet microcontroller can also allow the Arduino to report data to a central server. It is less expensive than the W5100, and for that reason it will be used in our solution, being detailed in Section 4.1.3.

- **Wireless** is an alternative solution to communicate with the central server. It is also simpler to install an energy meter due not being necessary installing cable to perform such communication. There are also several ways to communicate using this solution:

  - **Wi-Fi**: The official Arduino WiFi shield, presented in Figure 3.8 is the simplest solution to communicate with a central server. With this solution, the eergy meter acts like a laptop, accessing to the Internet using an AP. having direct access to the Internet as a,it did using the Ethernet connection. It also has an SD slot o store store data, such as enery measurements.

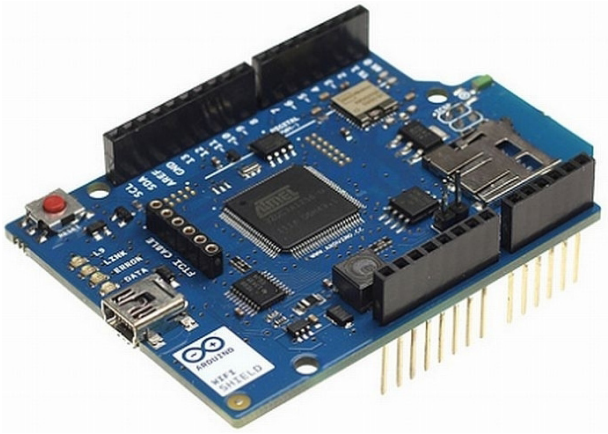

Figure 3.8: Arduino Wifi shield. This shield also has a SD slot to store data.

  - **ZigBee**: With this solution, is possible to create a wireless mesh network. However, if the central server is not present in this mesh network, additional communication modules may be necessary. The ZigBee module is depicted in Figure 3.9.

Figure 3.9: ZigBee module for Arduino. This module allows the creation of wireless mesh networks in a simple way.

– **Radio Frequency**: This solution offers wireless communication at a very low price compared with the previous ones. Usually this module is used to communicate to a single device that works as an AP, not offering a wireless mesh network solution, as this thesis aims to, as described in Section 4.1.3 and detailed in Section 4.3.3.

3.2.2 Projects

In this section, some open-source projects composed by the hardware introduced in the previous section are presented and detailed. All of them have a metering module to perform energy measurements, however, not all have a communication system to report the data to a central server, reporting the measures locally only to the end-user.

OpenEnergyMonitor

"*OpenEnergyMonitor is a project to develop open-source energy monitoring tools to help us relate to our use of energy, our energy systems and the challenge of sustainable energy.*" [Ope13a]. The goal of this project is to create an open-source end-to-end energy monitoring and control system, suitable for both domestic and industrial applications.

The solution architecture is presented in the Figure 3.10 where several devices, which are called modules, were developed.

The emonTx module is very similar to the goal of this thesis. It is a Arduino based device who uses CT and VT to measure energy and the radio frequency RFM12b module to report energy data. However, in their solution the radio frequency module communicates in a tree network topology instead of a mesh network topology. In this solution, multiple emonTx modules can be installed in a building, and the energy consumption data is reported to the enomBase module. This module supports 3 CT to measure the current, an AC voltage transformer for real power measurement, pulse counting, and can also report temperature measurements. It is based on the Atmega328 8-bit micro-controller and is fully compatible with the Arduino IDE. This module is presented in Figure 3.11.

As mentioned previously, the emonTx data is reported to the emonBase module. This module is responsible to forward the reported data to the central server, called emonCMS, using the RFM12b module communicate with the emonTx, and a wired Ethernet port to communicate with the central server. The emonBase, presented in Figure 3.12 can be implemented in two different ways: *a)* using an

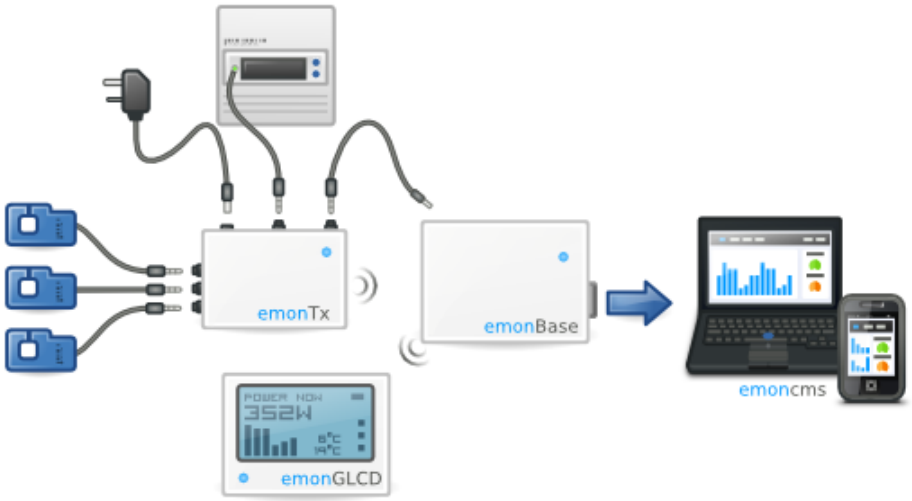

Figure 3.10: Block diagram of the Open Energy Monitor [Ope13a], a wireless solution that measures the energy consumption using the emonTx module, which reports the measurements to the emonBase gateway module, which in turn delivers the data to the emonCMS central server. An optional emonGLCD can be used for presenting data to users.

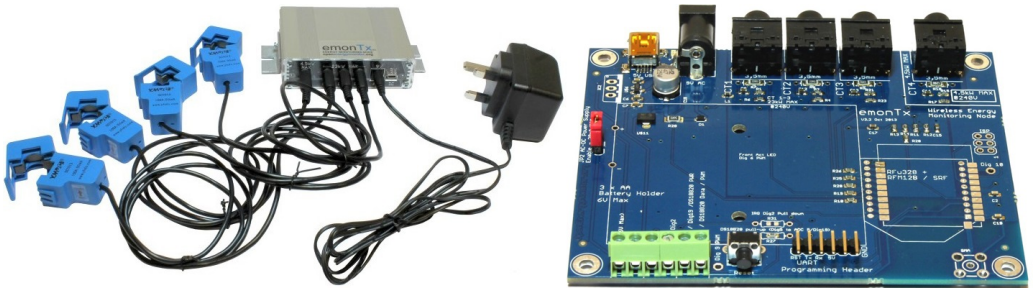

Figure 3.11: emonTx: The energy meter device provided by the OpenEnergyMonitor project.

Arduino-based solution called NanodeRF, or b) using a Raspberi Pi.

Besides this module can be implemented using both open-source platforms, and the NanodeRF has been in use for longest, the OpenEnergyMonitor team suggests using the Raspberry Pi platform claiming that it *is best future development potential* [Ope13a]. The Raspberry Pi is a low-cost computer with the size of a credit card that runs Linux and includes a graphic card with a High-Definition Multimedia Interface (HDMI) output interface, a SD card slot that can be used to store the measurements, an Ethernet port to communicate with the central server, and can also support extension modules, such as the Radio frequency modules to communicate with the energy meters (emonTx).

The central server in this solution, is a software called emonCMS, presented in Figure 3.13. This open-source software is a web application responsible for processing, logging and presenting the data regarding energy, temperature and other environmental data. Besides the OpenEnergyMonitor provides an online emonCMS platform [Ope14a] for energy consumption, it can also be installed on Windows or Linux, and therefore, the Raspberry Pi can also be used, thus having the emonBase and the emonCMS in a single device. The emonCMS interface is completely customizable by the end-user, to present only the information that really matters for each case scenario.

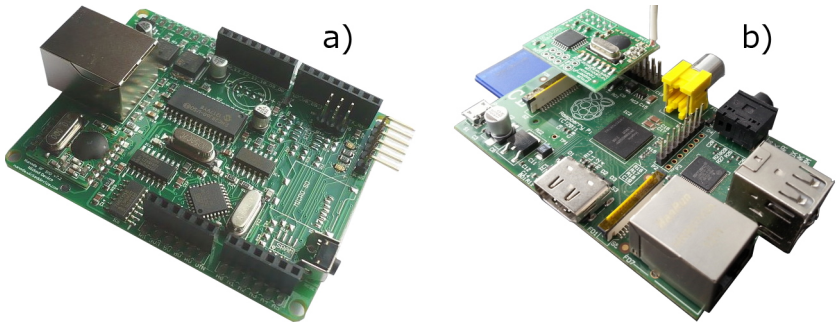

Figure 3.12: EmonBase is a gateway provided by OpenEnergyMonitor to report the energy data to the central server.

Figure 3.13: emonCMS provided by OpenEnergyMonitor to present energy consumption data

Besides of being an online platform to present consumption data on a web browser, the enomCMS team have a public GitHub repository [Ope14b] with all the source code of their solution, as well other software such as Android application development kits,compatible with their energy monitoring solution.

In this project, an optional Liquid-Crystal Display (LCD) can be installed to report consumption data, called emonGLCD presented in Figure 3.14. This Arduino based module receives data from the emonTx modules or from emonBase. It uses an Atmega328 microcontroller to process the data and the LCD to display the results. Besides presenting numeric data, this module is fully customizable and can also present graphics to report energy consumption to the end-user.

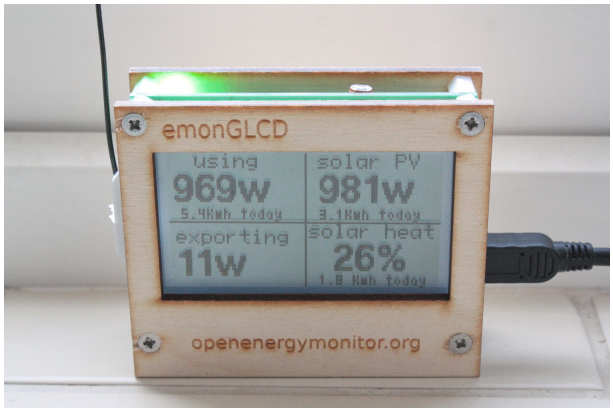

Figure 3.14: emonGLCD provided by OpenEnergyMonitor to present the consumption data

Besides being a complete open-source solution provided by the OpenEnergyMonitor, all modules implies hardware usage, than can be acquired in any electronics store. However, the OpenEnergyMonitor team provides an online strore [Ope14c] to ease up the process.

SEGmeter

The SEGmeter is other Arduino base solution initiative aimed at energy metering, provided by Smart Energy Groups [Sma12], which the architect solutionurs presented in the Figure 3.15.

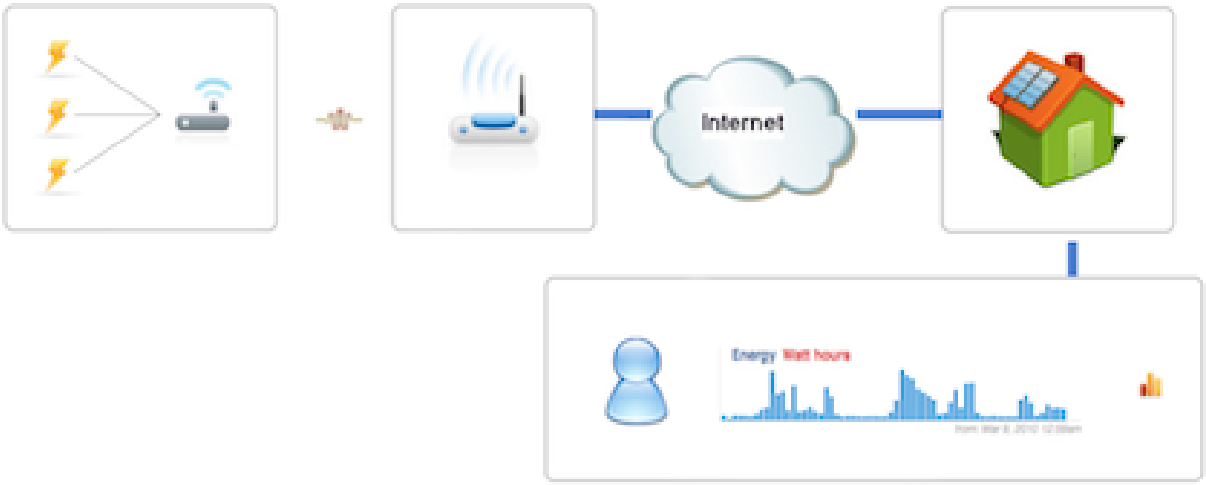

Figure 3.15: Architecture solution provided by the Smart Energy Groups for energy metering.

The SEGmeter supports up to 8 CT and reports the measurements every minute to the Smart Energy Groups servers, using an Ethernet or WiFi connection. Besides being a energy meter, the SEGmeter can also use relays to control electric usage, supporting up to three relays. The SEG meter is presented in Figure 3.16

Figure 3.16: SEGmeter provided by Smart Energy Groups for energy metering

In this solution, the measurement data is accessed using the Smart Energy Groups website, in the private area reserved for each client, as presented in Figure 3.17.

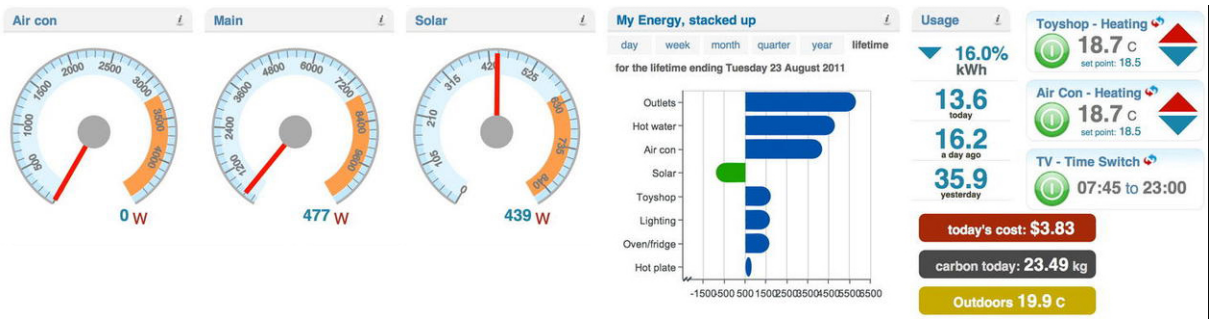

Figure 3.17: Smart Energy Groups Web Interface for energy consumption

Besides being an open-source project, the Smart Energy Groups sells their equipment in an online store [Gro12] to ease up the adoption of energy savings solutions.

Open Home Automation project

The Open Home Automation project [Sch13a] presents a solution for energy metering [Sch13b] using the INA219 [ada13] sensor presented in Figure 3.5.

This Arduino based project only explains the principles of using the INA219 sensor to measure DC current in a circuit, and so no communications with the central server are taken into account. It is all locally to the Arduino itself, presenting the energy metering information in the Arduino serial console.

In this open-source project, this principle is explained measuring the consumption of a LED but it can be applied to any DC device. Besides being a invasive method, meaning that the circuit has to be break to measure the current consumption, it is a very simple to measure the current consumption of a DC circuit.

Arduino home energy monitor shield

The Arduino home energy monitor shield [jme12], presented in Figure 3.18 is an open-source project that presents an Arduino shield for energy monitoring, using a non-evading technique to measure the power consumption, and report the data using the official Ethernet shield.

Besides the presented shield does not have a professional finishing, the open-source eagle files can be used to produce a PCB using an online PCB service for a better final look.

In this project, the VT used to measure the AC voltage is also used to provide energy for the Arduino and Ethernet Shield, avoiding the usage of a power supply. Since that it uses the Ethernet shield for communication, the energy data can be reported to the emonCMN (OpenEnergyMonitor online platform), or similar to present the data to the end-user.

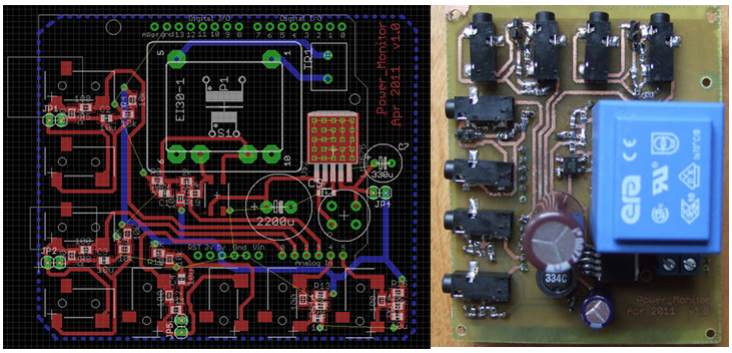

Figure 3.18: Arduino home energy monitor shield. On the left, the schematic of the PCB and on the right the PCB already mounted.

Desert home

The Desert Home[Dav10a] solution for power monitor is other interesting Arduino based project[Dav10b]. This project was based on the OpenEnergyMonitor detailed before to measure the power consumption, using CT and VT, and present the results to the end-user, using the emonCMS web-platform. However, the difference in this project is the communication to the central server, using ZigBee modules to report data in a wireless mesh network. The Figure 3.19 depicts the energy meter module.

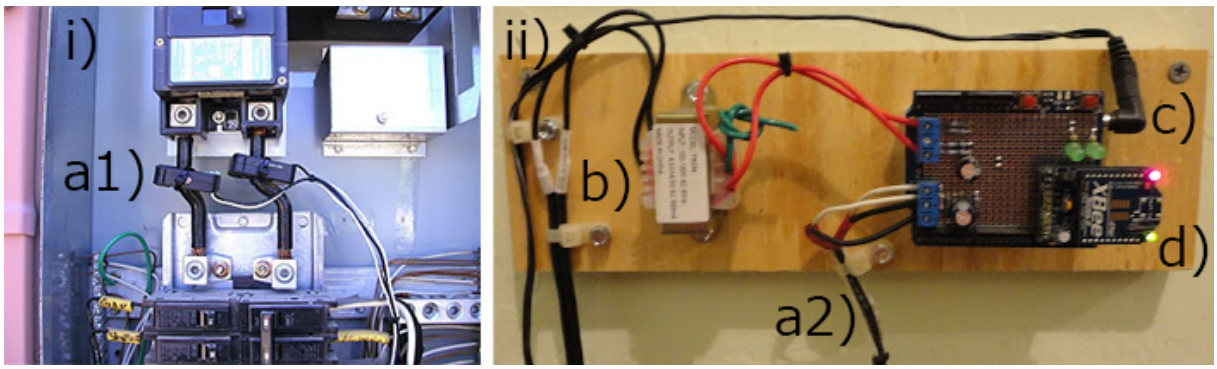

Figure 3.19: Desert Home energy meter. In this solution, the scenario i) represents the main breaker box and the scenario ii) represents the Arduino energy meter. The a1) depicts the CT measuring the current and the a2) the connection to the energy meter. The b) represents the VC, the c) the energy meter power source and d) is the ZigBee module to communicate wirelessly

Besides monitoring energy consumption, the ZigBee mesh network can also be used to communicate with other modules, such as actuators to control appliances, sensors to measure temperature, and LCD which can also be used to report energy data, being a complete solution for metering and controlling.

Arduino Energy Meter

The Arduino Energy Meter [Dut14] presents an alternative way to measure and present energy consumption. In this project, the metering module used is the ACS712, to measure AC.

To report data consumption, this project uses the Ethernet module to connect directly to the central server. In this project, the energy consumption is presented to the end-user in two different ways: i) using the Xively public cloud platform [Xiv03] and ii) using an LCD attached to the Arduino.

Prototype Energy Monitor

Another interesting open-source project regarding energy metering, is presented by Jay Kickliter on his blog [Kic11]. Together with Franklin Lynam, they developed a prototype, depicted in Figure 3.20 that can measure current using CT, and report the measurements to the central server, using ZigBee modules, forming a mesh network.

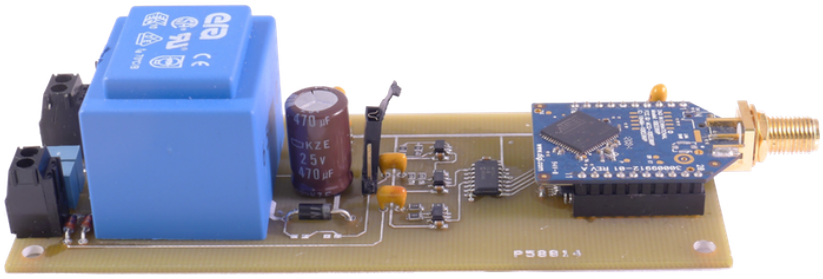

Figure 3.20: Prototype Energy Meter to measure energy consumption

To present the measured data in the central server, they developed a software that shows the information in a graphical manner, depicted in Figure 3.21.

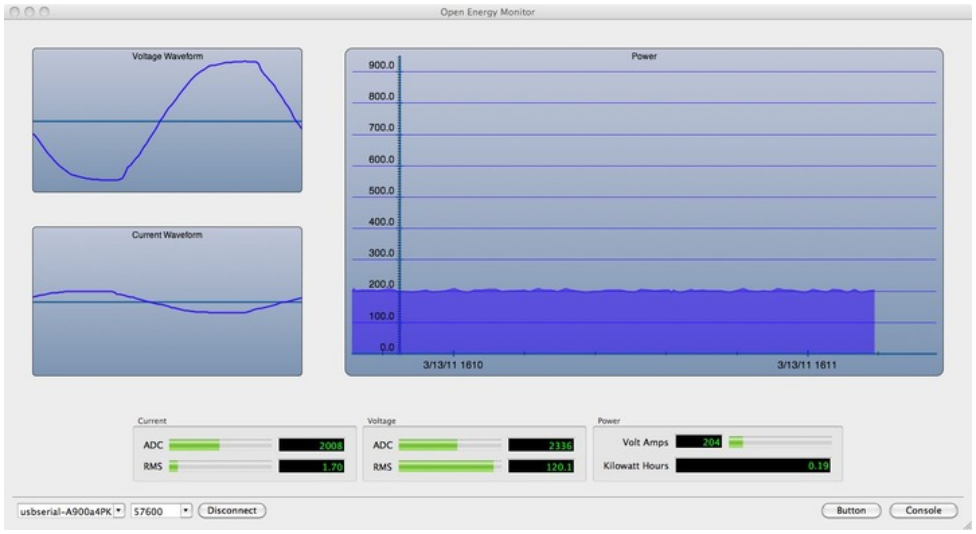

Figure 3.21: Prototype Energy Meter graphical report

In this project an 8 core propeller (P8X32A) micro-controller, provided by Parallax Inc. [Par13], is used. This micro-controller is similar to Arduino micro-controllers, but it can perform 8 different tasks at the same time. It can also be programmed, but uses a different programming language which is object oriented. This micro-controller is also available with a quick start board for prototype development, like Arduino does, but it has 8 LEDs, 8 resistive touch-pad buttons and 32 I/O ports.

The communication with the central server is provided by ZigBee modules, creating a mesh network of this prototypes. The usage of such module provides an out-of-box communication wireless mesh solution, but it is more expensive when compared with simple radio frequency modules.

It is an interesting project, still in a prototype stage, but the final price of each energy meter may be more expensive that the one proposed on this document, due to the usage of more expensive components.

Open Meter protocol

“OPENmeter" [OPE13b], is not a hardware or software project as the previous ones presented. It is a project which aims at building the European Advanced Metering Infrastructure, and enabling the widespread adoption of smart metering. According to the authors, this is possible through the development of a comprehensive set of open and public standards for smart meters, with the cooperation of a wide circle of key European stakeholders in the field, to ensure the acceptance of the project results [OPE13b]. This project sets recommendations in several areas, in order to create a standard based on existing technology and protocols, and includes not only Energy Management Systems for electricity, but also for gas and water. Besides not being a hardware solution, this type of initiatives are also important to save energy costs.

3.3 Comparison

During this chapter, several commercial and open-source solutions for energy metering were presented. In this section, a comparison of their main features is performed in Table 3.1 to better understand their differences.

Project	Metering	Communication	User Interface	Microcontroller
Wi-J&D	CT + VT	ZigBee	proprietary	proprietary
VerifEye	smart-meters	proprietary	proprietary	proprietary
OpenEnergyMonitor	CT + VT	Radio Frequency	EmonCMS	Arduino
SEGmeter	CT + VT	WiFi	proprietary (payed)	Arduino
Open Home Automation	INA219	N/A	N/A	Arduino
Arduino H.E.M. shield	CT + VT	Ethernet	EmonCMS	Arduino
Desert Home	CT + VT	ZigBee	EmonCMS	Arduino
Arduino Energy Meter	ACS712	Ethernet	Xively	Arduino
Prototype Energy Meter	CT + VT	ZigBee	proprietary	Parallax

Table 3.1: Energy metering solutions comparison

Besides existing several solutions, only the OpenEnergyMonitor uses radio frequency modules for wireless communication. The advantage of this technology over the Wifi and the ZigBee is their lower price. However, WiFi offers a direct access to the Internet, when connected to an AP, to report data to a remote server, which the radio frequency module can not offer by itself, and the ZigBee offers an off-the-shelf mesh network, which the radio frequency module does not offers by default, but it can be implemented, as explained in our solution in the next chapter.

Chapter 4

Solution

The creation of a low-cost meter implies the use of low-cost tools that can help solving the energy metering problem, thus reducing the initial investment on an EMS. Some of these tools are widely accepted, not only because of their low price, but also for the well explained and open-source documentation they provide, being the Arduino platform an example of this. This chapter covers the implementation and coverage of this platform, in order to present the solution for the problem solved by this thesis.

4.1 Solution Overview

4.1.1 Requirements Analysis

In order to develop a useful solution to a problem, it is important to understand the user needs and requirements. Not performing a good requirement analysis can lead to a very functional model, but useless to solve the initial problem. Regarding this thesis, several requirements were identified:

- **Energy readings** must be performed with a non-invasive technique. In this case, a current transformer can measure the power usage and a voltage transformer can be used to measure the voltage line variation.
- **Communication to Central server** must use the extension modules that Arduino supports, allowing a communication channel between the central server and a specific solution module.
- **An existing protocol** Modbus must be used to communicate with the central server to report data.
- **Failure proof** maintaining configuration values even after an electrical outage.
- **Low price** using of-the-shelf open-solution available to the public as the Arduino platform, a final product solution can be created without royalty fees.
- **Easy to install** solution modules that only require being powered on to be ready to function.
- **Master-slave topology**, in that communication between the central server and the modules must be performed one at a time, to avoid packet loss in the wireless channel.

4.1.2 Solution Highlights

In this solution, there are several important aspects described in different sections. In this section, all major highlights will be described.

- **Modular Architecture** is used in our low-cost solution. Only the necessary modules are installed in the EnergyBrick for a specific task. If a module is not needed, then it is not installed, thus saving it for someone who really needs it, and thereby saving costs. The extension modules chosen for this solution were also selected based on their price, to reduce even more the final cost.

- **Simple protocol communication** used by all EnergyBricks, communicate using the Modbus TCP protocol in a User Datagram Protocol (UDP) packet, and only functions 3 and 16 are used. This open protocol is very simple to understand, thus allowing an easy and custom made implementation by programmers, which can also be integrated with already installed systems.

- **Network for standalone smart-meters** can be provided using our solution. An isolated smart-meter that communicates with Modbus RTU can be integrated in an EMS using solution. If several are installed in a building, not forming any network, then we can offer a communication solution.

- **Low-cost metering** can be performed using our solution. All decisions made were based on reducing costs. Even for metering energy, the hardware used were selected to be as low-cost as possible. Depending on the already installed system, our solution can be even more affordable.

- **No energy service interruption** is needed to install our meters. The usage of split core CT allows the system installation without shutting down the building's power.

- **Controllability** is also important in a EMS. In our solution, is also possible to include remote control modules allowing to save even more energy. This can also improve people's life quality by increasing their comfort.

- **Wireless solution** where no additional cables are need to be installed in the building, since all data is sent via wirelessly.

- **Local and Remote configuration** of our energy meters is possible in our solution. Such features ease up the commissioning process.

- **Failure proof** is also an import requirement in a system. After a failure, all EnergyBricks boot up with the configurations that were set before the failure, thus completely restoring the wireless mesh network.

- **Safe firmware update** is possible in our solution. In our solution, uploading new firmware to offer new features is possible, without losing any of the previously made configurations.

4.1.3 Block Diagram

Our energy metering solution consists of distinct modules that handle different problem scenarios. Arduino is an open-source platform that can support shields to provide extra features at a low-price.

Using this feature-based idea, we developed some different devices, each one only performing a specific task, as will be described in Section 4.2. The advantage of using a modular architecture is that we can provide only the necessary modules for a specific task instead of having all of them installed in the meters and then only use a few functions. This way it is possible to reduce even more the production costs for each meter in a final implementation.

By developing our own shields, a well customized meter can be created for a specific place where it will be installed. The advantage of this approach is that we can have a complete set of tools and use only the required parts instead of developing a complete and high-price device with too many modules that might not be used in a given place, thus wasting resources that could be used elsewhere. This is an important aspect, because given the goal of developing a low-cost solution, wasting resources would be counter-productive.

In our solution, there are: *i)* metering modules, which perform energy measurements, and *ii)* communication modules that report those measurements. In our architecture, some meters can have more that one communication module, and also more than one way of metering, depending on the situation, e.g., some meters can be also connected to the central-server using a Ethernet shield, or some meters can use a serial RS-485 module to inquire an already installed smart-meter that does not have the ability to communicate its information to the central server.

This idea is depicted in Figure 4.1, and will be detailed next.

Metering modules

In our solution, there are different modules that can be used to perform energy metering:

- **Meter module** which performs energy metering itself, by using a simple electrical circuit and also current and voltage transformers. This module can be used in every part of the circuit's building, measuring from a simple lamp to the entire building. Regarding the measurement modules, this is the most expensive one, due to the price of the current and voltage transformers, presented in Figure 4.2. These transformers are connected to the Arduino, using an electrical circuit that will be detailed in Section 4.2.3.

- **Meter Inquire module** which can perform requests to already installed smart-meters. In order to do this, the Arduino communicates using the Modbus RTU protocol in a RS-485 serial line to inquire the smart-meter about the energy readings. Regarding measurement modules, this one is cheaper than the previous module, only depending on a simple UART - RS-485 converter, depicted in Figure 4.3.

- **Pulse Counting module** which performs energy readings using a simple light sensor. Some already installed meters, may have a LED that blinks every time a specific value of energy is

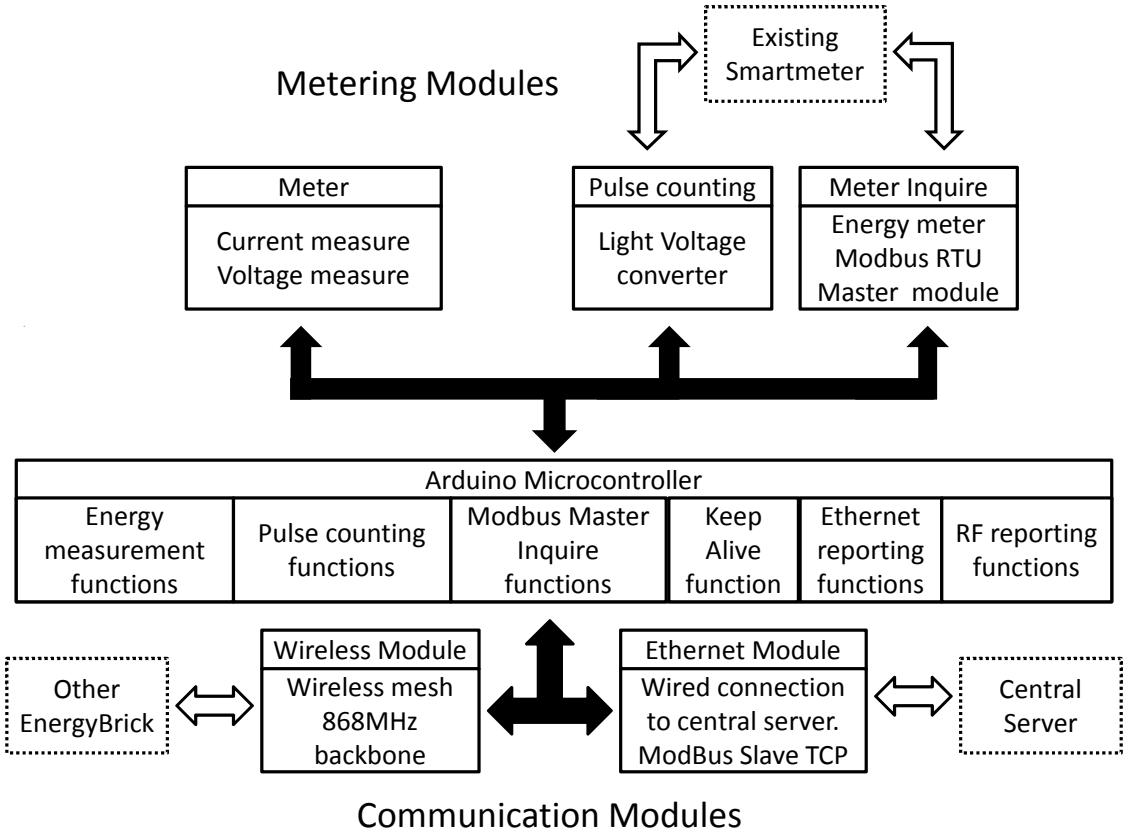

Figure 4.1: EnergyBrick block diagram used to developed the prototype models. On the top are presented the metering modules that can be used to measure electricity directly or through a smart-meter. On the bottom are presented the communication modules that allows communication between Energy-meters and also to the EMS. On the middle is represented the main microcontroller functions of the solution.

consumed. In this case, an Arduino with a simple light sensor attached can detect that pulse and update its energy count, performing energy measurement in a very simple way, thus being this measurement module the cheaper one.

Depending on the already installed energy measurement system, the required investment for our system can be very low. An example of that is using just the pulsing counting modules to perform energy measurements, attached to the already installed smart-meters in a building. On the other hand, if there is no previous installation, the usage of metering modules can still be very low-cost, due to our hardware solution.

Communication modules

In our solution, the EnergyBrick is not a single entity that performs energy measurements. As will be described in Section 4.2, there are several different roles that can be performed in our solution, but all of them communicate with each other. However, in our solution two communication modules are used:

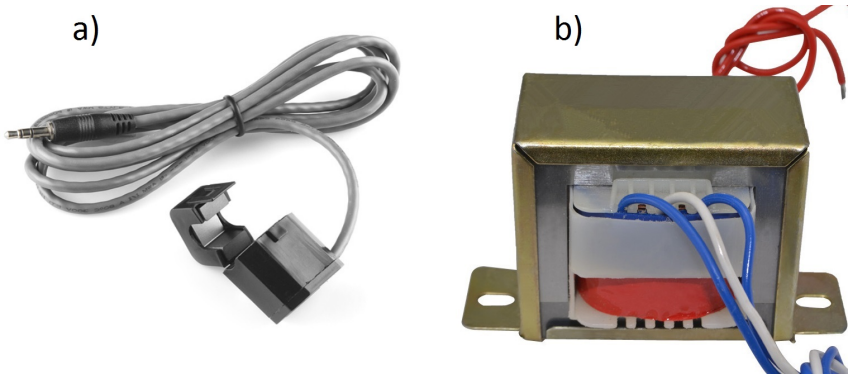

Figure 4.2: Metering Hardware used in our solution. The current transformer, *a)*, performs consumption readings that are supported by the voltage transformer, *b)*, wich provides an adequately accurate consumption metering.

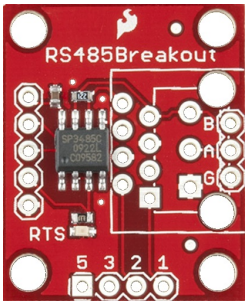

Figure 4.3: RS-485 Breakout Board used in our solution to communicate with smart-meters. This extension module converts the Arduino UART signals into RS-485 signals, allowing communication with devices that use this interface.

- **Ethernet module** provides communication between an Arduino and the central server. This way, it is possible to perform remote requests to the EnergyBrick which is the entry point in our solution. The chosen Ethernet expansion module uses the microcontroller ENC28J60, depicted in Figure 4.4, which currently has the lowest price in the market, allowing this communication without increasing too much the price.

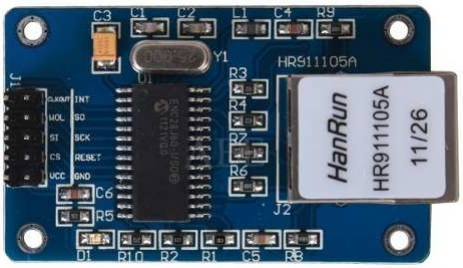

Figure 4.4: Ethernet module ENC28J60. This module provides communication to the central server.

- **Radio Frequency module** provides a wireless solution communication between EnergyBricks forming a mesh network with a frequency of 433 MHz that does not interfere with the Wi-Fi signal that can already be installed in the building. This way, it is possible to have a wireless backbone, which means that no additional cables need to be installed. This can be a major advantage in

case a quick implementation must be performed, or if for historical reasons the building structure must remain untouched. This wireless expansion module uses the RFM12b microcontroller, which is also very cheap, especially when compared with ZigBee solutions, and offers a good coverage signal in 868MHz, as presented in Figure 4.5.

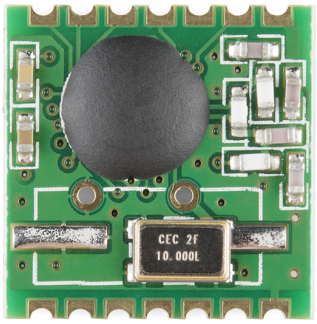

Figure 4.5: RFM12b module used to developed a low-cost wireless mesh network between EnergyBricks.

Connections

To provide extra features to the Arduino, expansion modules must be correctly connected to it. Usually, vendors sell their Arduino-related products along with examples of how to use them with Arduino, to help the user understand how to use them. To do this, they usually provide the schematic to connect the device to the Arduino, an Arduino library containing the library code, and also some example sketches. Connecting a single module is not complicated, since depending on the used library, there is a base schematic of how to do it. But if more than one must be connected, then both modules won't work if both schematics indicate to use the same Arduino pins. This problem is especially relevant in expansion modules that use the SPI bus.

As mentioned before, the bus is shared, so it is normal to connect both modules to it, but then, the chip select pin must be different in both, which sometimes is not, and also the interrupt pin must be different, which almost never happens, due to the Arduino Uno only having 2 interrupt pins, and the modules are usually connected to the first one. The advantage of open-source, is that we can solve these conflicts by examining how the lib is implemented, and changing the pins' definitions. This section describes not only how the connections are made, but also common problems that occur when connecting multiple expansion modules to a single Arduino, and how to solve them. The overview solution schematic presented in Figure 4.6 represents all the modules' connections to a single Arduino Uno.

Metering modules connections

There are three measurement modules in our solution: *i)* metering module, *ii)* meter inquire module and *iii)* pulse counting module. The metering module is plugged in the analog port 0 (A0) to measure the

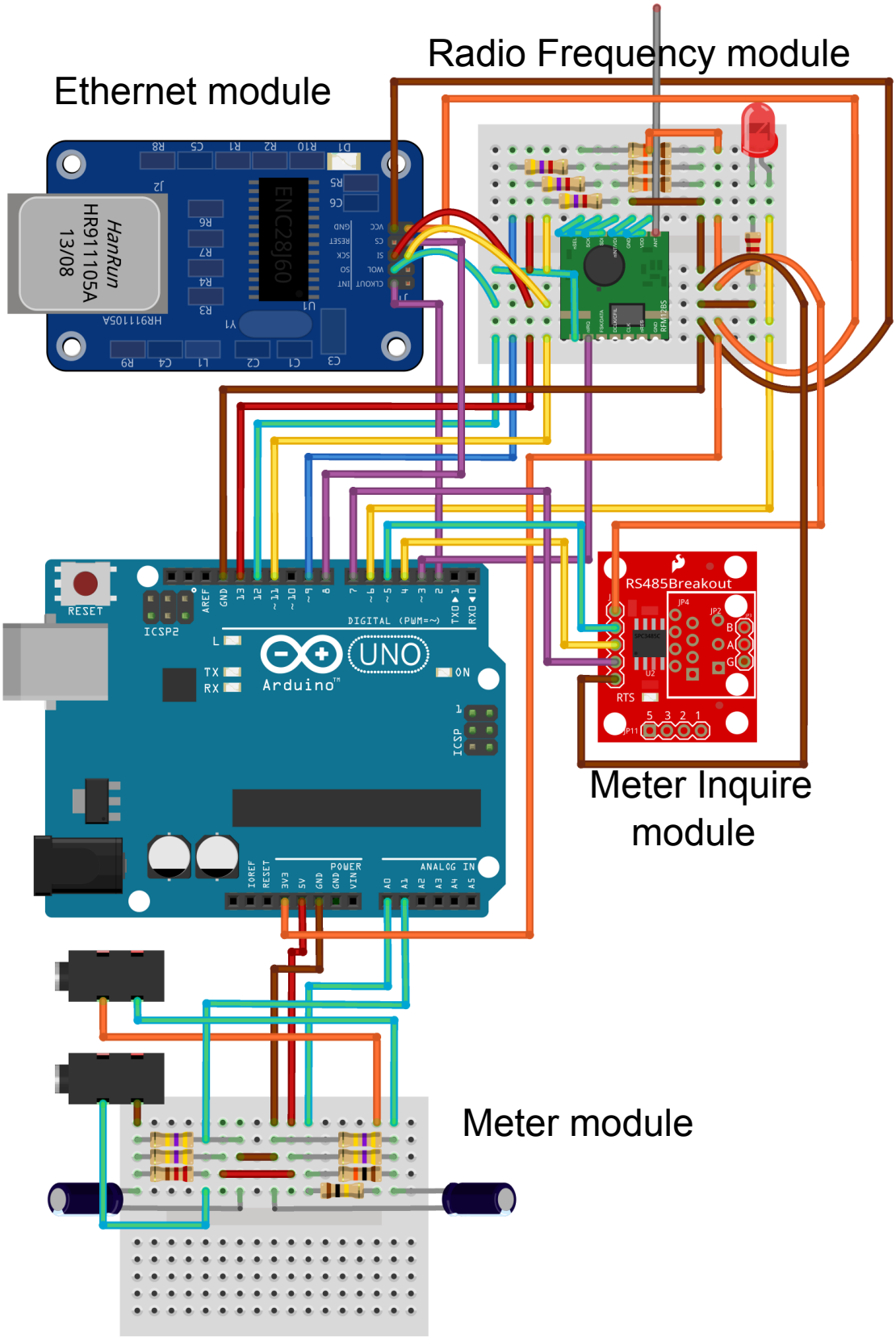

Figure 4.6: Block diagram of the Arduino based Architecture.

voltage and in the A1 to measure the current in a single phase line. To measure a three phase line, additional current transformers are required, as presented in A1, connected to A2 and A3.

The meter inquire module, converts the UART signal provided by Arduino, into a RS-485 signal that can be understood by the smart-meters. With this module, a serial line is created, using the digital pin 4 (D4) as receiver (Rx), the D5 as transmitter (Tx), and D7 as request to send (RTS), which commutes the state of sending and receiving. By default, the serial communication in Arduino is the D0 as Rx and D1 as Tx, but using it, implies that all debug messages also go through the serial communication with the smart-meters, and worst than that, every time an upload has to be performed, the D0 must be disconnected, which is not practical in a final solution. As such, we decided to use the New Software Serial library provided by Arduino to create a virtual communication port.

The pulse counting module is not presented in the figure, due to the usage of pins that are already used by other modules, but since the idea is to choose only the needed modules, as explained in the next section, it is possible to also use it.

Communication modules connections

The EnergyBrick can have two communication modules, however, some library changes in one of them must be performed, since both try to use the same chip select pin, and also the same interrupt pin. Regarding this thesis, we decided to change the RFM12b library to not use the same interrupt pin as Ethernet does, so both can be used simultaneously.

As presented in figure 4.6, both modules are connected to the SPI bus, but the Ethernet module uses the chip select pin D10, and interrupt pin D2 as INT0, while the Radio Frequency module uses the chip select pin D9 and the interrupt pin D3 as INT1. With both modules working simultaneously, it is possible to have an Arduino that can be the entry point to the wireless mesh network, where the requests can arrive via Ethernet, and then forwarded wirelessly, as will be explained in the next section.

4.2 Solution Architecture

Different roles may be performed by the EnergyBrick depending on the installed modules. In our solution, there are three main EnergyBrick models: i) GatewayBrick, which makes the bridge between the central server and the other EnergyBricks, ii) Energy Meter Brick, than can measure power consumption, and iii) Energy Meter Serial Brick, that reports smart-meters data. These models depicted in Figure 4.7 uses combinations of the presented modules to perform a single task that will be described in next sections. Having all modules at once could be to much expensive, so the best approach to a low-cost solution is only use the modules that are really needed to perform a specific task, uploading only the necessary sketch code. This is also important, due the flash memory limitation in the Arduino Uno, where the code of all modules do not fit in, so using this approach also gives the possibility to improve each task, since there are more flash room to use, when uploading just part of the code.

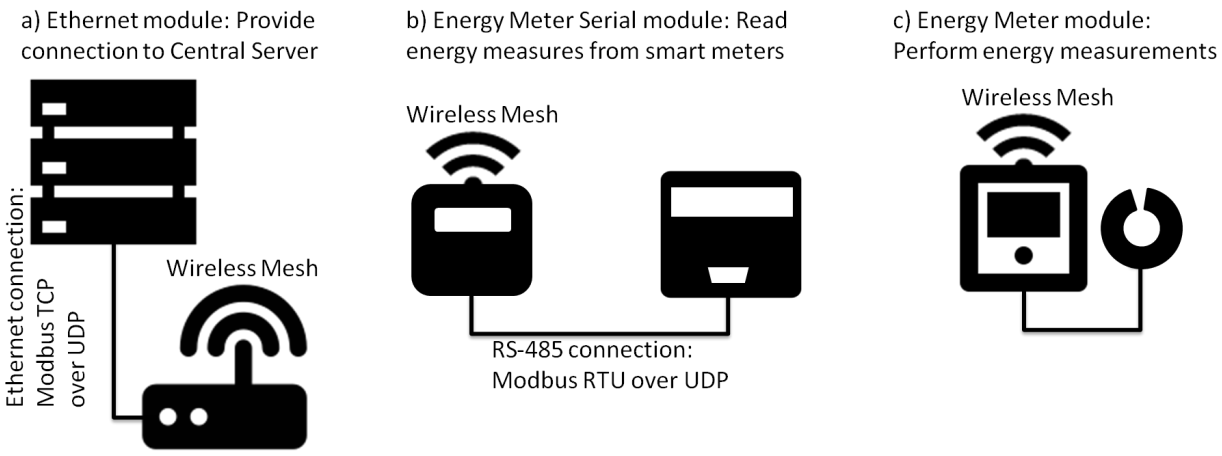

Figure 4.7: Solution overview. On the left, the Ethernet module provides communication between the central server and the other modules using the wireless mesh. On middle, the energy meter serial module provides readings from smart-meters. On the right, the Energy meter module performs energy readings.

4.2.1 Common EnergyBrick Aspects

EnergyBrick modules share a number of common aspects that should be presented together. There are essentially four main common characteristics:

- **Radio Frequency mesh** provided by the RFM12b extension module to communicate in the wireless mesh network, allowing the reception, forwarding, and response to the requests performed by the central server. The mesh network protocol will be described in the Section .
- **Internal memory organization** provided by the Arduino EEPROM memory organization is the same for all the EnergyBrick modules. This memory is used to store the individual configuration and is not lost after a energy black out, allowing a complete restore after failure. The address details will be presented in Section 4.3.1.
- **Modbus TCP protocol** is used as the application protocol message to communicate with the EnergyBrick modules. They understand the function codes for reading and writing in the EEPROM, which will be detailed in 4.3.2.
- **Common Base features** in all EnergyBrick for convenience purposes. Besides those features do not require any expansion, module, they are still useful to understand the system status. Is the case of the heartbeat feature, which with a single LED provides a visual regular blinking status giving to the user a feedback that the EnergyBrick is working. Other visual feedback that the EnergyBrick offers is the possibility to connect it to a Laptop, via USB, and check in real time all the behavior of the EnergyBrick. This connection can also be used to access the options menu and setup the EnergyBrick locally. This changes can also be performed remotely using the wireless mesh network. Finnaly, a auto-commissioning feature is provided, which automatically set up the EnergyBrick in the system.

4.2.2 Gateway Brick

The Gateway Brick is the entry point of our network, as depicted in the Figure 4.8. In this case, the Arduino has the Ethernet expansion module attached which provides communication with the central server, and also the RFM12b module which provides communication with the others EnergyBricks in a mesh network. Since this module uses both communication modules, it works as a gateway between the two networks, and so it has the role of master in the wireless mesh network, performing the requests to the others EnergyBricks slaves, similar to what happens in the Modbus solutions.

As mentioned before, the Ethernet module used in the Gateway Brick has the microcontroller ENC28J60, which is currently the lowest-cost module providing Ethernet connection. The drawback of this choice regards in additional space required by their library, which may be wisely chosen to provide the basic functions required. In our case, we tested several libraries, with different configurations to achieve a desirable balance between flash usage and function. The initial idea for the communication protocol was to establish an TCP connection between the central server and the Gateway to communicate using the Modbus TCP protocol, but only one library was able to do it successfully: UIPEthernet. The drawback of this library, is that it occupies almost all flash memory in the Arduino Uno, not allowing additional require features such as wireless mesh networking which is mandatory in our architecture. We also tested having an Arduino Uno using this library in TCP mode, and communicating with other Arduino using a simple point-to-point wireless communication with the RFM12b, where the other Arduino performs the master role in the mesh network, but the code didn't fit in the Arduino.

Other attempt to interconnect both Arduinos was using the New Software Serial library, but once again it didn't fit in the flash memory, so we tried to use the I2C bus to communicate between them. Besides the successful code's upload in the Arduino with the Ethernet module, it didn't worked as expected due the time needed to perform an request, on a remote device in the mesh network. Since the response didn't arrive immediately, a timeout occurs, and the response fails to be delivered to the central server. Even if this solutions works, it would be more expensive than using an Ethernet module with the W5100 microcontroller which also provides TCP connection using much less flash memory, allowing to also include the wireless mesh library.

Since that all TCP options didn't worked as expected with that low-cost Ethernet module, we developed an UDP solution with Modbus TCP, which will be detailed in the Section 4.3.2, but for now, considerer that instead of using a TCP connection with the where the Modbus TCP request is sent, we just put it in the payload of an UDP datagram and send it to the Gateway Brick. The UDP feature has a very small flash footprint compared to TCP, which also allows the usage of the wireless mesh module in the same Arduino. In this UDP solution, we use the EtherCard lib, since it has a smaller footprint compared to others libs that also provides the UDP feature, such as UIPEthernet or Ethershield. The Dynamic Host Configuration Protocol (DHCP) feature allows the EnergyBrick to automatically acquire an Internet Protocol (IP) address. However, it also consumed a lot of flash memory and so it was removed, forcing to setup the IP address manually.

By default, the EtherCard lib uses the Chip Select pin D10, which is also used by the official Ethernet extension, module, we decided to change it to D8. With this modification, is possible even with a already

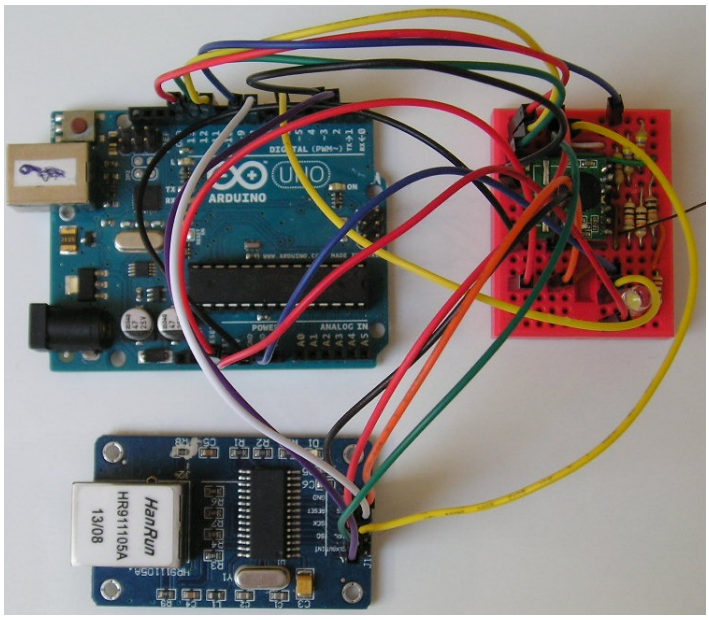

Figure 4.8: Gatweay Brick prototype. This models provides communication between the central server and the EnergyBricks present in the mesh network.

produced breadboard in the future to change the module, or work with both.

4.2.3 Energy Meter Brick

The Energy Meter Brick, presented in Figure 4.9, is able by itself to measure the energy consumption in the installed location. To this end, this brick uses the Metering module to perform readings, and the Radio Frequency module to report them. In order to perform the measurements a CT is used to measure how much current is being used, and it also uses a VT to measure the voltage level in the main line.

As mentioned before, typically to measure the current in a circuit, an ampere-meter have to be installed. This installation requires breaking the circuit in a evasive way to force the current to pass through the ampere-meter to perform the measurement. This installation implies cutting off the power to be performed and is also dangerous specially in high voltage circuits such as the ones used in buildings.

A better and safer approach is using a non-evasive way to measure current using a CT with a split coil. With a split core the conductor can be easily installed, splitting the CT core and putting it around the conductor. The CT has a magnetic that must be installed around the cable to measure the current. When electricity is passing thorough a conductor, a magnetic field is generated along it. “A CT utilizes the strength of the magnetic field around the conductor to form an induced current on its secondary windings" [ELK06]. The induced current is smaller than the current passing through the cable, but it can be calculated using the CT scale. Having the inducted current with a much smaller scale, and isolated from the main circuit, is now possible to measure it in a safer way. The CT output usually is a inducted current but there are also CT with an internal (burden) resistor that converts the current in voltage using the Ohm’s law.

These reading can achieve a better precision if the exact value of the voltage level at that time is

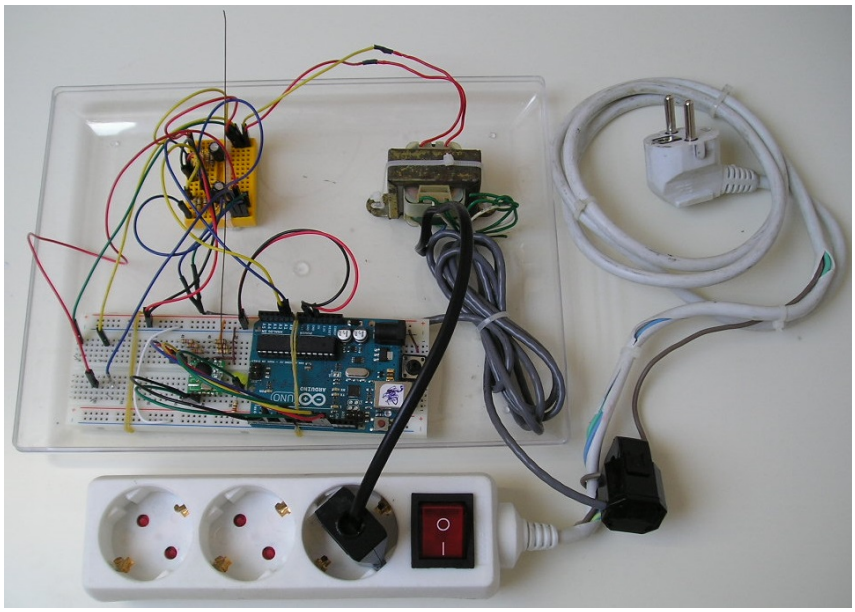

Figure 4.9: The Energy Meter Brick performs consumption readings and is able to report them to the central server.

known. Since that measuring the main voltage directly is dangerous, and could easily break the Arduino, we use an AC/AC VT to reduce the scale. As mentioned before in Section 2.1.2 both current and voltage curves change polarity, at the same time, but this alignment can be lost if the load is not resistive. This polarity reading can be achieved when both CT and VT are used, which also allows the calculation the power factor, apparent and real power for a precise measure.

Having both CT with a voltage output, and also a VT, is now possible to connect then to the Arduino analog input ports, as showed in Figure 4.10, to measures how much energy is being used, and report it to the central server when asked.

In order to connect both VT and CT (which also produces a voltage output) to the Arduino analog inputs, a simple circuit ir required due the nature of the main voltage and current values. There are two main problems that must be solve:

- **Negative values** which are not supported by the Arduino analog input ports. They only supports values between 0 and 5 volts. As mentioned before, the voltage and current waves can be positive and negative, so if we could connect them directly, only positive values could be read. The solution is to implement a simple circuit that puts the Arduino zero value in 2.5V. This way the negatives values are represented between [0 ; 2.5[and the positive values between the]2.5 ; 5], meaning that the maximum input values in the Arduino analog input must be escalate between [-2.5 ; +2.5], as presented in the next item.

- **Scale values** to not wreck the Arduino. When measuring the main voltage in Europe, the result is a variation between negative and posivite values: 230V (RMS). Those values, breaks the Arduino board if connected directly. A possible solution is to scale down using a VT AC/AC. If a 2,5 volts or lower VT is used, then we could connect it directly to the Arduino if combined with the previous item, but in many cases, is more common to find 9V or 12V AC/AC VT which is better than the

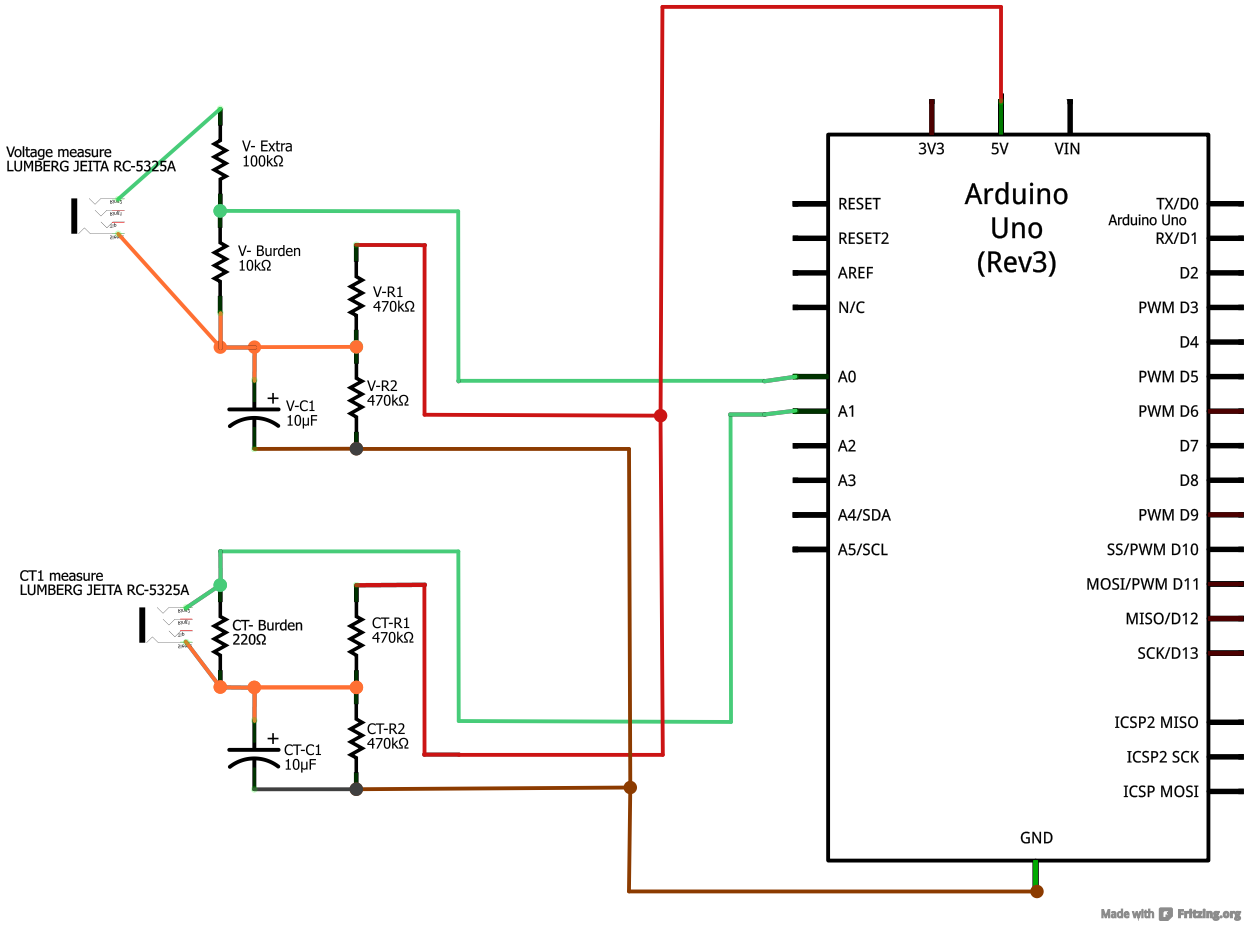

Figure 4.10: Energy Meter Brick schematic.

main voltage levels, but still to high for the Arduino analog input.

In this case, the solution is to perform a voltage divisor to output a maximum value of 2.5V in both positive and negative values. This way, the combined solutions outputs a value that can be read by the Arduino analog inputs safely, allowing it to calculate a correct voltage level in the main line. To read the current consumption, the CT may already have a burden resistor that outputs the reading value in a voltage level. Depending on that value, a voltage divisor may also be used.

When applying this techniques, the energy can be measured safely, since the resulting wave, depicted in Figure, is safe to the Arduino board.

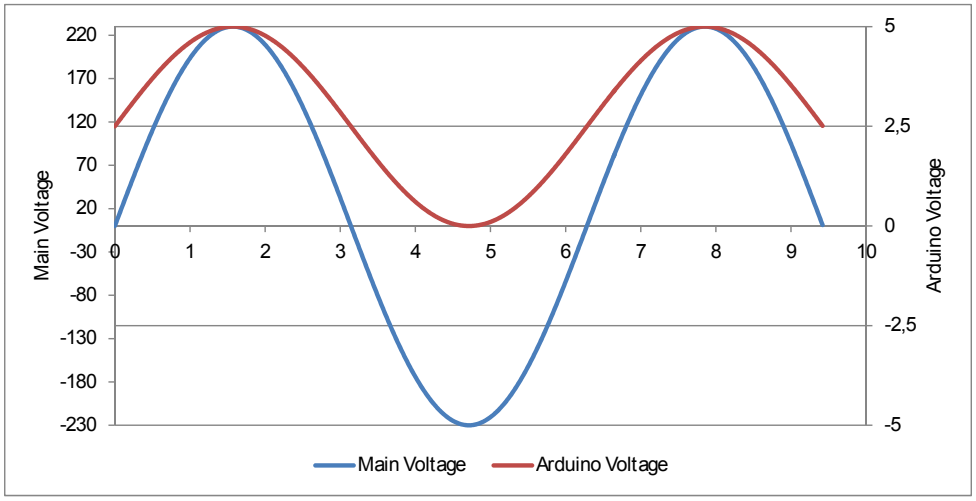

Figure 4.11: The graphic represents the transformation that must be performed in the main voltage to perform its reading using an Arduino analog input port.

Implementation details for current transformer

The hardware specifications used in this implementation, are not mandatory. Any CT or VT can be used, however, its safer to work with low voltage devices and converters to produce the same result. In order to not break any component, some calculations must be performed. When connecting this converters to the Arduino board, we must be sure that the maximum voltage levels that can be achieved will not damage the board.

The CT output voltage, is proportional to the consumed energy. In this case it can be used up to 30A converting this maximum in 0.015A. The data sheet of this device indicates that it has an internal burden resistor of 250Ω, so by the Ohm's Law ($V = R.I$) the maximum voltage level is $V = R.I = 250 \times 0.015 = 3.75V$, which higher than 2.5V. If we want to set 2.5V as the maximum output value for this CT, one of two thing must be performed:

- **Lower the resistor** using a resistors in parallel. Since that the the burden resistor must be lower, a way to achieve it, without removing it is to install an extra resistor between the output connections, being in parallel with the internal burden resistance, which creates an equivalent resistor that can be calculated using the formula $(R_1 \times R_2)/(R_1 + R_2)$.

In this case, for a maximum 2.5V maximum voltage output, and using the Ohm's law, $R = \frac{V}{I} = 2.5/0.015 = 166.6\,(6)$ equivalent resistor must be performed. Since that the internal resistor is 250Ω, representing a fixed value for R_1, the selected resistor for R_2 must be around 500Ω. However, not every values are available in resistors, so, in this case, its is safer to chose a lower one around this value, which will be 470Ω. By using it, the equivalent resistor has the value of $(250 \times 470)/(250 + 470) \approx 163.19\Omega$, which gives a final voltage level of $V = R.I = 163.19 \times 0.015 \approx 2.445V$, being safer for Arduino.

- **Limit the current** when perform metering on a location. Depending on the circuit breaker installed in the circuit, the maximum value can be lower than maximum value that the CT supports. In this situation, the maximum CT voltage level is lower, and can also be used safely in Arduino. Even if is not the case, we can still use it, but we must be aware of that a limited amount of consumption must be known. This solutions is not ideal, but can also be used in several situations.

  For the given hardware, the maximum current that can flow can be calculated using the Ohm's law: $I = \frac{V}{R} = 2,5/250 = 0,01A$, converting this value to the real scale, we get: $Max_I = (30 \times 0,01)/0.015 = 20A$. If the room has a circuit breaker that cuts off the power at 16A, then we can safely install the CT assuring that it will not damage the Arduino board.

Implementation details for voltage transformer

To perform a correct voltage reading without damaging the Arduino board, a AC/AC VT must be used. As mentioned before, if a 12V or 9V is used it is still higher than the 2.5V. The solution is using a voltage divisor to lower the voltage level output, to get a maximum value of 2.5V.

In our case, the maximum voltage output is around 10V for the 230V in the main circuit. Appling the voltage divider, formula $V_{out} = \frac{R_2}{R_1+R_2} \times V_{in}$, with resistors $R_1 = 100k\Omega$ and $R_2 = 33k\Omega$ the output value is $33k/(100k + 33k) = 2.481V$, as presented in which can be connected to the Arduino.

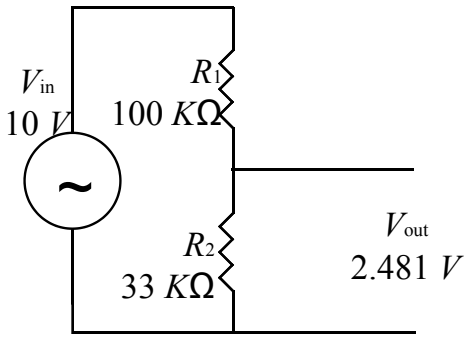

Figure 4.12: Voltage divisor representation to read values from the voltage transformer

Calibration

When accomplishing this calculations, the expected value may not be the exact one as expected. This occurs due the materials not being perfect. To solve this problem, a calibration must be performed

by software to get the expected consumption values. This is only possible if the obtained values are compared to a calibrated EnergyBrick, or other available energy meter.

Safety considerations

In the previous sections a strict approach to the values were performed. However, as also said, the materials are not perfect, and dimensioning the solution for the maximum resolution that Arduino can get, it may be dangerous and wreck it. To avoid this problem, is safer to dimension the circuit to a lower maximum voltage level, as [1 ; 4] to ensure that no problem will occur while performing energy metering.

4.2.4 Energy Meter Serial Brick

The main function of the Energy Meter Serial Brick, presented in Figure 4.13 is to request data to a already installed smart-meter.

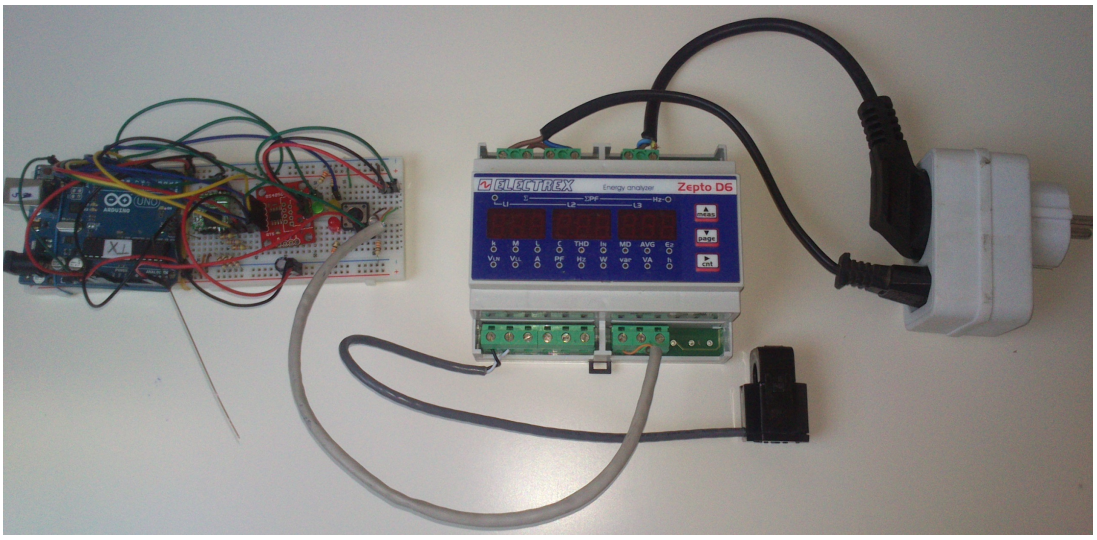

Figure 4.13: The Energy Meter Serial Brick, on the left, reports smart-meters reading values, on the right, to the central server using the meter inquire module connected to an RS-485 energy meter.

This Brick uses the Meter Inquire Module presented in Section 4.1.3, to inquire the smart-meter using the Modbus RTU protocol, and report those values to the central server. The main advantage of this module, is that it can provide a wireless reporting for the smart-meter, and also include it in the EMS.

The only installation requirement for this module is that both meters have the same Modbus slave ID, which virtually creates a single energy meter in the system. This abstraction allows a simple control of both energy meters, as one, allowing an easier management. When communicating with this virtual device, the Modbus TCP request arrives to the Energy Meter Serial Brick which then checks if the request is for itself or to the smart-meter device.

Since that both have the same slave ID, the way that the EnergyBrick decides is using the Function codes and also the memory address in the Modbus request. By looking at the Modbus packet it can respond if the request is for itself, or convert the ModbusTCP in a Modbus RTU packet and inquire the

smart-meter, which will respond, and then the EnergyBrick converts back the Modbus Messege in a response to the Modbus TCP original request, and send it back to the central server.

4.3 Solution Implementation

To implement our solution, several decisions were taken into account to assure the scalability of the system. In this section, the based software features are detailed to better understand how our solution is implemented.

4.3.1 EEPROM Organization

Providing a system failure mechanist, is a feature that may always exists. After a power failure, for instance, is important to restore all settings as they were before the failure, to guarantee a proper system operation. Regarding the EnergyBrick, is important to not lose any configurations previously made to avoid a system malfunction. To store this information the EEPROM memory that Arduino provides is used to stores all the configuration data as a system failure mechanism.

The EEPROM is a memory that has 1024 entries (for Arduino UNO) of 8 bits each, which allows to store a value up to 255. In many circumstances, a 8 bit value could not be enough, so it was decided to merge pairs of this memory, where the higher bits value are stored in EPPROM adress N and the lower bits value in stored in the address N+1, creating a new Arduino library to ease up the programing process. By creating this abstraction layer, the EEPROM virtually has 512 entries of 16 bits each, but is still possible to store data in a single 8 bit entry if needed. All EnergyBrick presented modules has a reserved configuration memory that are detailed in Table 4.1. The presented options can be changed: *i)* locally, connecting the EnergyBrick directly to the computer through a USB connection to setup it using a console to access the options menu, or *ii)* remotely, using where the central server sends configuration requests to the EnergyBrick.

Auto-commissioning

When booting for the first time, the auto-commissioning feature checks the EEPROM table. If all table is cleared, all entries has the default Arduino value of 255. In this situation, the auto-commissioning feature sets the EnergyBrick default values in the EEPROM, as presented in Table 4.1, which allows a correct configuration for all installed EnergyBrick modules. If a specific module is not present, its auto-configuration will not be performed. This way, a simple remote EEPROM check is enough to understand which modules are installed in a specific EnergyBrick. The Serial number is the only random value generated at run time during the first boot. Regarding the mesh network auto-join, it is described in the Section 4.3.3.

Module	EEPROM Address (H,L))	Configuration	Default Value
Common configurations	0,1	Custom ID	65535
	2,3	Serial No.	65535
	4,5	Heartbeat Interval	1000
	6,7	Heartbeat Digital Port	6
	8,9	Serial Baud rate	9600
	10	Random value generator pin	4
	11-29	Reserved	65535
Ethermet module for Central Server communication	30,31	UDP port	5002
	32-35	IP address	10.0.0.2
	36-39	Gateway address	10.0.0.1
	40-43	Central Server address	10.0.0.1
	44, 45	Destination Port	55970
	46-51	MAC address	65535
	52-60	Reserved	65535
RS-485 module for Modbus RTU	60,61	Baud rate	9600
	62,63	Timeout	1000
	64,65	Polling	300
	66,67	RTS-pin	7
	68,69	Rx pin	4
	70,71	Tx pin	5
	72-79	Reserved	65535
Metering module	80,81	Voltage Calibration	24298
	82,83	Phase Calibration	170
	84,85	CT 1 Calibration	933
	86,87	CT 2 Calibration	933
	88,89	CT 3 Calibration	933
	90,91	Consumption Kwh (stored)	0
	92-98	*Access runtime memory*	*65535*
	92	Consumption Kwh	65535
	93	Current	65535
	94	Real Power	65535
	95	Apparent Power	65535
	96	Real Power	65535
	97-99	Reserved	65535
Control module	100	ON/OFF	255
	102	PWM value	255
	103-199	Reserved	65535
Not used	200-499	Not used yet	65535
Wireless mesh Configuration	500, 501	Node ID	65535
	502, 503	Network Nodes	0
	504, 505	Network ID	170
	506, 507	Ack Time	2000
	508, 511	Reserved	65535
	512 - 1023	*Forwarding Table*	*65535*
	512+N, 513+N	Next Node ID to N	65535
	514+N, 515+N	Hop number to N	65535

Table 4.1: EEPROM memory map. This persistent memory provided by the Arduino, stores all configuration data for all modules used by EnergyBricks. This configurations are automatically set during the first boot, and can be remotely changed latter.

4.3.2 Communication Protocols

In this section the protocols used in the EnergyBricks are presented. As mentioned before, there are two main choices that must be made. The first one was the communication between the central-server and the wireless mesh network gateway, which will be detailed in Section 4.3.2, and the second was the communication between the EnergyBricks detailed in Section 4.3.3.

Application protocol

The application protocol defines how an external application can interact with the EnergyBricks modules. To avoid defining a new energy protocol, we chose to implement a common used protocol already known worldwide. By using a already known protocol, a already made application can interact with any EnergyBrick, right away without problems. In this case, we adopted the Modbus TCP. As explained in Section 4.2.2, the lack of memory flash space in the Arduino won't allow us to implement a TCP connection, and so, a UDP connection must be performed instead. The only difference, is that the Modbus TCP format packet is encapsulated in a UDP datagram, and then sent to the Gateway Brick, which forward the Modbus TCP request to the corresponding EnergyBrick checking the Slave ID. In the case of the mesh network, the Slave ID corresponds to the Node ID of the Mesh Network, as it will be better detailed in Section 4.3.3.

The Modbus protocol defines several function to read and write values in the different defined tables described in Table 2.3, whoever, since that all configuration values are stored in EEPROM, in 16-bit registers, as mentioned in Section 4.3.1, which is stores the configuration values that can be customized by the central-server, only the Holding Registers primary Modbus table is considered in this thesis. From all functions defined in Modbus protocol, only two were considered in our case to manipulate data: *i)* Function code 3, to read multiple values from the EEPROM, and *ii)* Function code 16 to write multiple values in the EEPROM.

The Energy Meter Inquire Brick has a special feature that allow it to convert between Modbus TCP and Modbus RTU to request info from the smart-meters. To implement it, an additional library were developed. Since both EnergyBrick and smart-meter have the same slave ID, the Slave ID in the Modbus request is not enough to inform to who is the request. To solve this problem, a packet inspection must be performed. The function code, and start address fields are the ones who defines the destination in this case. If the function is not 3 or 16, so it is a request for the smart-meter since that functions are not implemented in EnergyBricks. Since that the EnergyBricks only have 1024 address memories, in the case of the function code is 3 or 16, then it inspects the start address field. If this value is bigger than the EPPROM memory, then the request is for the smart-meter. When converting it to a RTU request, the start address is also converted to start from 0 ($x - 1024$) allowing to access all smart-meter memory addresses.

The EEPROM organization used, allows with a simple request get configurations from a specific module, or set the configurations in it, being very easy to remotely configure any EnergyBrick.

4.3.3 Wireless Mesh Network

In our solution architecture the central server communicates with all EnergyBricks sending the Modbus request to the Gateway Brick, that has a wireless mesh module that allows the request forwarding. In this Section, the wireless network topology, protocol and features, and the will be better detailed.

By default, this RF module can send and receive data between two nodes with a range of 100 meters fitting well on our needs. It also provides acknowledgments by software, using its library. Besides working well, it does provide a mesh network solution by itself, whoever it can still be very helpful to provide communication between nodes in a mesh network.

Since that a mesh network library has to be implemented, the main idea were to implement a transport library that uses the link library layer provided by the hardware manufacture. This abstraction, grants a functional communication between nodes provided by the hardware library, and also an easier way to migrate to a even lower-cost hardware if needed.

The next sections presents the mesh network details regarding the frame format, network architecture and commissioning protocol.

Frame Format

To implement the mesh network, additional fields has to be added in the link packet payload. In this earlier stage a very simple solution were implemented, but yet powerful enough to provide a mesh network. The original RFM12b library already provides the source and destination nodes in the packet and also the acknowledgment fields for communication between two nodes. To provide the mesh solution, similar fields were added to extend this link-to-link communication, as presented in Figure 4.14

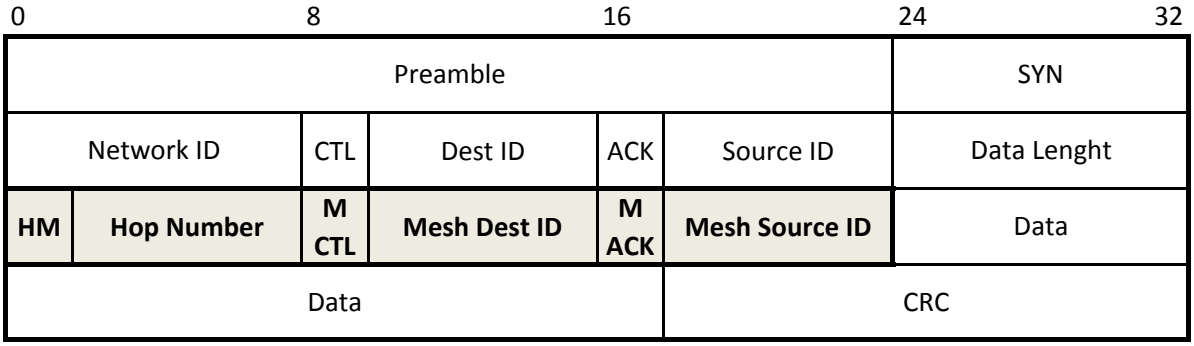

Figure 4.14: RFM12b packet used in the mesh network. The darker section represents the changes made to implement our mesh solution.

To ensure communication between two nodes in a mesh network, the RFM12b provides in the packet the fields:

- **Preamble and SYN** are used to initiate a packet transmission and synchronization

- **Network ID** is also used for synchronization, but it also identifies the Network channel.
- **CTL** is a bit flag that indicates a acknowledgment to the packet received.
- **Dest ID** identifies the address of the destination node.
- **Ack** is a bit flag that indicates if a packet requires an acknowledgment.
- **SourceID** identifies the address of the packet sender.
- **DataLenght** specifies the size of payload data.
- **Data** is the payload of the message.
- **CRC** is a Integrity check value to ensure a correct packet.

In the original packet data, some fields were added in the Data field to ensure end-to-end communication:

- **Have Message (HM)** is a bit flag which indicates if the EnergyBrick has some message to transmit to the Central Server, when responding to a request. This can be used in situations that allowing the EnergyBricks start a communication with the central server can cause problems. To avoid them, the master-slave topology can be used with this single bit which informs the central server that it has a message to deliver. By doing it, the central server request that message, and the EnergyMeter responds.
- **Hop Number** identifies the number of hops between EnergyNodes nodes to reach the destination node. Every time that the packet is forward, the hop number is decreased. In a well formed network, the hop number is 0 when it gets to the destination. If the value gets negative the packet is no more forward in the mesh network to avoid flood.
- **Mesh CTL (M CTL)** is the same as the CTL, but between end-to-end communication.
- **Mesh DestID** is the address of the destination node in the mesh network.
- **Mesh Ack (M ACK)** is a bit flag that indicates if a packet requires an acknowledgment.
- **Mesh SourceID** is the address of the mesh packet sender.

Mesh Routing

In the mesh network, the packet must be forward over the network until it reaches the destination. For every given hop, an acknowledgement must be performed back in order to the sender knows that the message was successfully delivered to that next node. For a given node perform the packet forwarding, firstly it checks if the packet passes in the CRC verification, avoiding invalid packets to be forward, and then send an acknowledgement informing that it successfully received that packet. Then the node check if the packet hop number is not negative, avoiding lost packets looping in the network. After those verifications it consults a mesh forwarding table to know which is the next node in the path, decreases the

packet hop number and forward the packet to the next node. An example of a mesh table is presented in 4.2.

To Node ID	Next Node ID	Number of Hops	Observations
0	0	255	Broadcast Packet
1	65535	65535	This entry is never used. It's itself
2	2	1	The node ID 2 is connected directly to itself
3	2	2	The node ID 3 is not connected directly, but the next hop is 2.
4	4	1	The node ID 4 is connected directly to itself

Table 4.2: Example of the master forwarding mesh network table. Each EnergyBrick has is own table definitions that were automatically stored during the entrance on new nodes. This table can remotely be edited if needed allowing a customization of the mesh network.

This algorithm is performed until the packet reaches its destination, allowing the receiver performs the given request and sends back a response. Using this method, a Modbus TCP request can be sent by the central server, and the EnergyBrick can send the Modbus TCP response. This algorithm is depicted in the Figure 4.15.

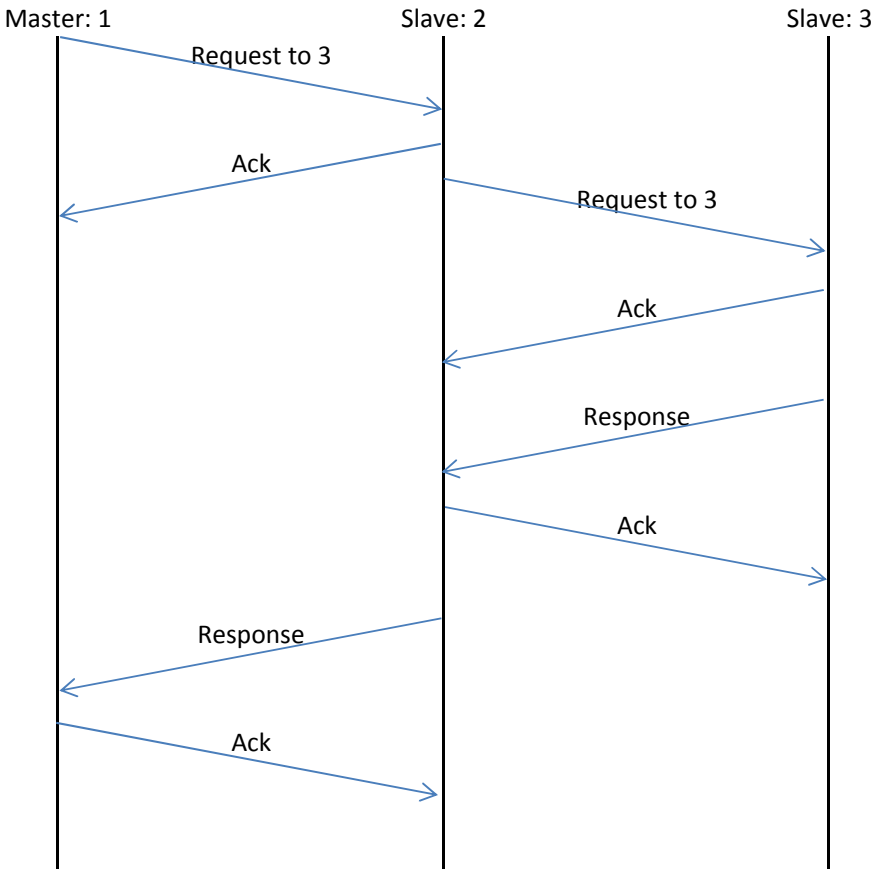

Figure 4.15: The Request and Response in the Wireless Mesh Network is represented in this message flow diagram where the packets are being forward by slave 2.

Mesh Commissioning

One of the functional requirements of this thesis is the easy out-of-the-box equipment installation. The solution for this problem, is allowing a self-configurable energy meter in our network without previous configuration. When booting up for the first time, the energy meter performs some defaults verifications described in Section 4.1, and one of them is to check if it already has an network ID (nodeID). If it has, then it means that he just rebooted due some reason, such as an energy blackout. If not, then it means that is it first time booting, and so, he must request a new ID to be part of the mesh network. In the case of the Gateway Brick, its node ID is already defined as 1, being the mesh network master.

To better understand the join process, the process is depicted in Figure 4.16. and presented next in several steps:

1. **Scanning** is the first stage when a node is joining the mesh network, the EnergyBrick assumes the temporary node ID 127, and sends periodically a Broadcast "Hello?" message with his Custom ID and Serial number attached to it. Only register mesh nodes will respond with an acknowledgment to this broadcast packet, but not immediately. By being a broadcast packet, if all nodes acknowledge without a back-off mechanism, a air collision occurs and the acknowledge will not be received, to solve this problem all nodes waits a random time, shorter than the Ack Timout. This way, an acknowledgment is always received by the joining node. The first who replies, is then selected to forward his request to master to enter in the network.

2. **Requesting ID** is performed to the selected node in which it will be connected in the mesh network all the time to communicate with the central server. The EnergyBrick send a "ReqID" with its Custom ID and Serial ID attached to that node, who creates a new packet with the received request and send it with the wireless mesh network master as destination.

3. **Responding ID** is performed after the master receive the packet. It creates a "RspID" packet with the node ID value to the joining node, which is the total network nodes + 1, adds the entry to this new node in his forwarding table: (to: new node ; next node: packet sender ; hop number: same as packet sender + 1), and send the response packet. As this packet is being forward in the network, all nodes adds in their tables this new entry

   After arriving to the slave node that requested this new entry, it also adds this new ID to his table and creates a new packet to the joining node with the "RspID" and adds the hop number to the central server and send it to the new node.

4. **Confirmation ID** is performed when the joining node receives the packet, it stores his new ID, adds the entry to the master in his table, reboots the RFM12b with the new ID associated to it, and sends a confirmation "NewIDok" that the process was well performed.

5. **Updating Network** parameters is performed when the master receives this confirmation, it updates the new network nodes in the EEPROM, finishing the process. Additionally, a message can be sent to the EMS informing the presence of a new node and where it is connected.

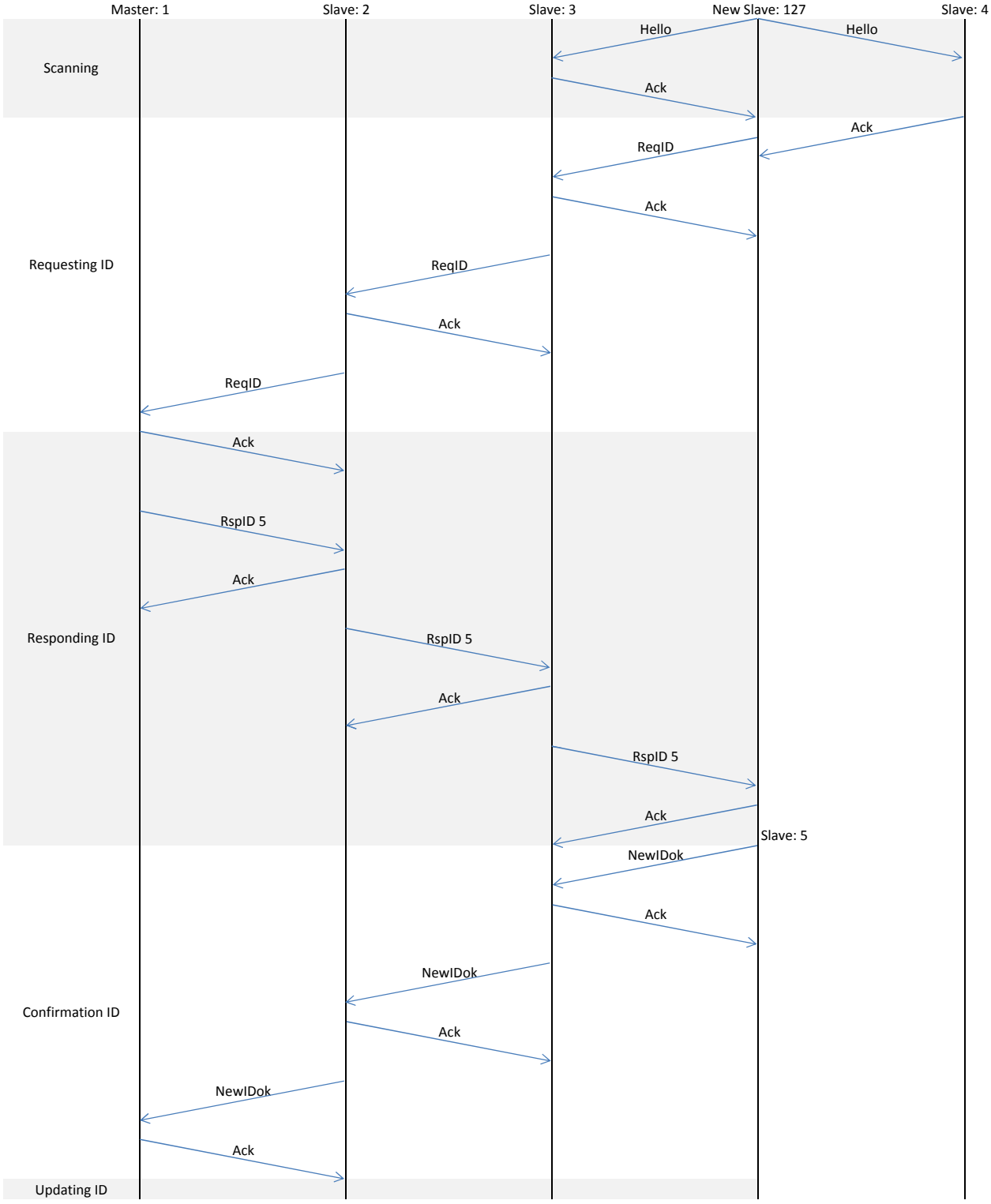

Figure 4.16: The Auto-commissioning in the wireless mesh network is presented in this message flow where the new node enters and request a new ID. The forwarding request results in a response that contains its new ID 5, and causes a new entry in the forwarding table of each node that participates in this process.

Chapter 5

Evaluation

To validate the easiness of installation, several tests were developed and will be described. Since there different EnergyBricks, each one have a different way to be installed, however, all of them must be performed in a very simple way to reduce commissioning costs. This tests are also presented in this chapter.

The developed solution resulted in multiple EnergyBricks modules, each one with its own features. This chapter conducts field tests to validate the solution.

5.1 Methodology

The low-cost solution must be very easy to install and to integrate in the system. It must also be able to perform and report energy measurements in different ways, depending on the used module.

Ideally, those measurements are reported to an EMS, which processes and presents them in a report to the building manager. Since it is out of the thesis scope, an EMS was not used to produce a energy report using this solution, however, we were able to produce a graphical view based on the reported values to validate our work. The energy measurements are reported through a low-cost wireless mesh network developed especially for this thesis, to ensure a fully functional low-cost product.

To perform the solution evaluation, a real usage scenario was implemented using the developed EnergyBricks, as presented in Figure 5.1.

In this scenario, all EnergyBrick modules developed are presented, and each one performs its own tasks:

- **The central server (1)** is responsible for requesting consumption measurements and performing reports. In this setup scenario, a laptop was used to perform that specific task, and it was connected to the Energy Gateway Brick (2) using an Ethernet connection. A consumption graphic was produced in real time, by requesting data through a UDP datagram that transports the Modbus TCP request.
- **The Energy Gateway Brick (2)** has both communication modules installed, thus merging both networks. Every request and response between the central server and an EnergyBrick, passes

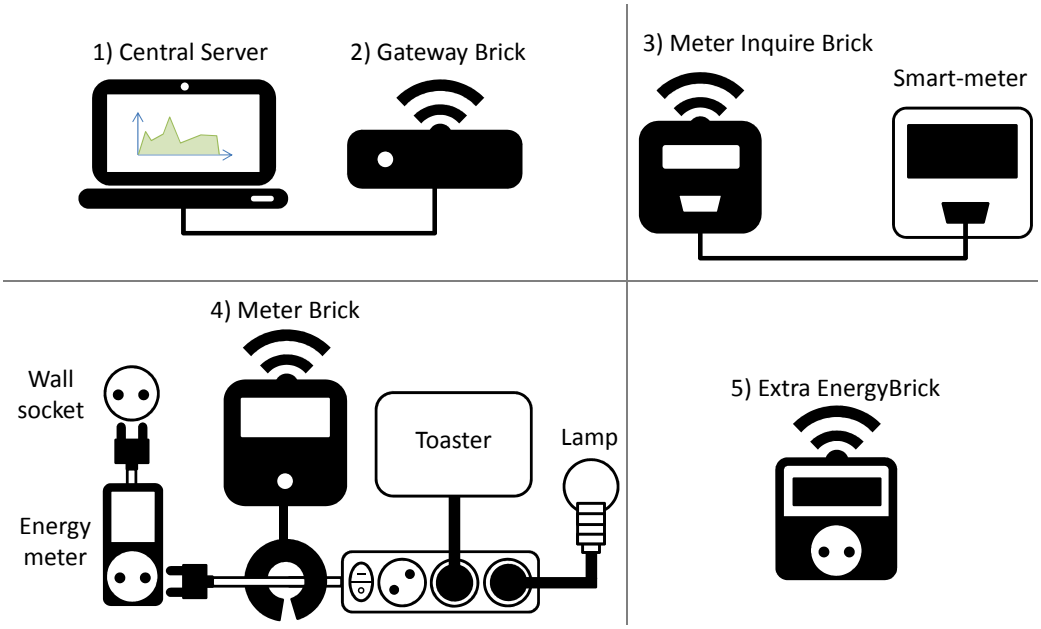

Figure 5.1: Setup scenario used to evaluate the solution. The central server (1) uses an Ethernet connection to the Gateway Brick (2), which communicates with the other EnergyBricks using the developed Wireless Mesh Network. The Meter Inquire Brick (3) requests energy meter data to the smart-meter, while the Meter Brick directly measures the energy consumption. The extra EnergyBrick (5) has no measurement modules and is used to test the network formation.

through this gateway. This module is also responsible for generating wireless node IDs when a new EnergyBrick connects to the network for the first time.

- **The Energy Meter Inquire Brick (3)** is responsible for converting Modbus TCP into Modbus RTU messages and forwarding them to the smart-meter. The smart-meter has a RS-485 communication port and communicates using Modbus RTU.

- **The Energy Meter Brick (4)** performs real time energy consumption measurements. It was connected to an extension cord to read from the connected electronic equipment. In this scenario, several equipments were measured, including different lamps and a toaster. This EnergyBrick was calibrated using an energy brick already available in the market to compare the results.

- **The EnergyBrick (5)** was created just to test the mesh network. It has no meter modules, however it can still be remotely accessed for configuration, which validates the network tests.

To test the wireless mesh network, a base network was established, and is presented in Figure 5.2. In this network, 5 nodes are already defined and an extra node is used to perform commissioning tests. Regarding the presented EnergyMeters, node 1 is the Energy Gateway Brick, node 2 is the Energy Meter Inquire Brick, node 3 is the Energy Meter Brick. Nodes 4, 5 and 6 are EnergyBricks that only have the wireless communication node, of which the 6th node is used to perform the commissioning tests.

Having this setup scenario presented, the next sections detail the tests that were performed in order to validate the solution. Those tests include:

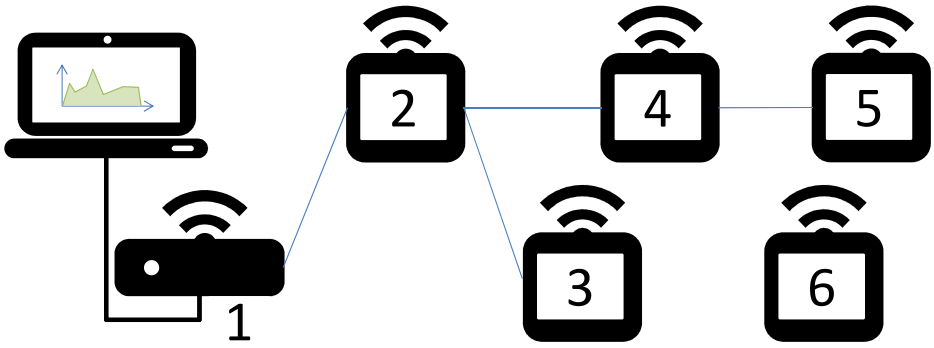

Figure 5.2: Wireless Mesh Network tested to validate the solution. Nodes 1 to 5 are fixed and node 6 was used to perform commissioning tests.

- Maximum distance between two EnergyBricks
- Message forwarding between EnergyBricks
- Auto-commissioning tests
- Resetting an EnergyBrick
- Measuring energy directly
- Message using Ethernet
- Remote configuration tests
- Requesting energy measurements to smart-meters

5.2 Tests

5.2.1 Distance between nodes

These tests allow us to conclude the maximum distance allowed between two nodes. When installing the proposed solution, it is important to know this metric to successfully perform it. In the presented solution, the EnergyBricks communicate with at least one neighbour node. The datasheet of the RFM12b expansion modules claims to have an open air range above 100 meters. To test the modules, a simple *ping* test was performed between two Arduinos. The Arduino that sends the packet is fixed, while the receiver is moving in the scenario.

To test the maximum distance between two nodes, three locations were selected: i) Residential area, ii) IST Tagus Park campus and iii) IST Alameda campus. The residential area, represents a scenario without interference, were the signal has to pass through many walls. The IST Tagus Park and IST Alameda campus represent scenarios where the interference may be a decisive factor for the range. However, the IST Alameda building has many walls and the IST Tagus Park building is more open spaced.

In the first scenario, the test was performed in a three-story building, being the sender placed in the third floor and the receiver moving at the street level. The result was quite impressive, as the signal almost crossed the entire building (around 50 meters, crossing many ceilings), losing signal only in 1 meter to the ground.

In the second scenario, the sender was placed in room 1-42 and the receiver was moving around the IST Tagus Park building. Being a very open space building, the covered area was about 40 meters, almost reaching the outdoors. This result shows us that the signal can easily cross a building's width.

In the third scenario, the sender was placed inside a room on the third floor, and then the receiver was moving at street level, as performed in the first test. In this scenario, only one floor was successfully crossed, however, it is still enough to cover the entire building.

The results are presented in Table 5.1.

Scenario	Residential	IST Tagus	IST Alameda
Interference	No	Yes	Yes
Walls	Yes (4)	No	Yes (1)
Distance (m)	50	40	20

Table 5.1: EnergyBrick test range results using different scenarios.

The obtained results allow us to conclude that these low-cost modules have a very good coverage area. Since the main goal is spreading hundreds of them in a large building as IST Tagus Park to measure the entire consumption, we can conclude that this is a viable solution.

5.2.2 Message Forwarding

In our solution, a low-cost mesh network was developed, and this test presents the results of the packet forwarding mechanism. To perform the message forwarding test, the mesh library has to be developed first, since the provided library only supplies communication between two nodes. To perform this test, the Arduinos were uploaded with the solution code, and the messages were sent between three nodes, which are depicted in Figure 5.3.

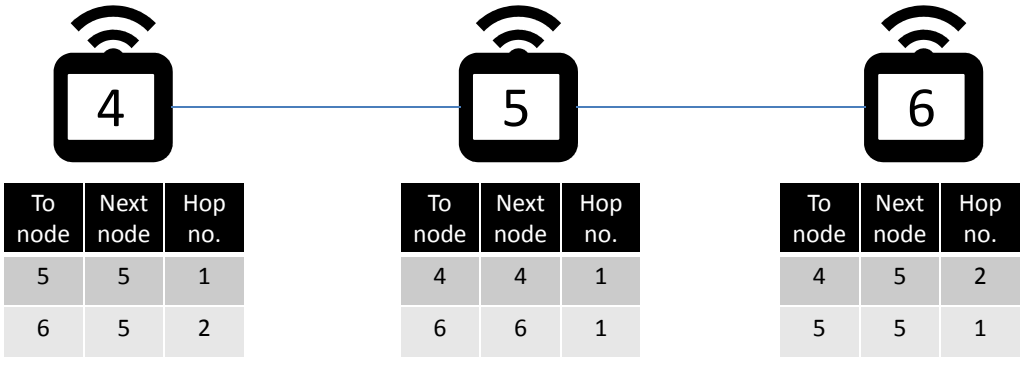

Figure 5.3: Forwarding setup to perform the test. Three nodes and their respective forwarding tables are presented.

In this test, two dummy requests were sent: i) Between direct links and ii) between indirect links, using a node to forward the message. An output example is presented in Figure 5.4, successfully validating the forwarding feature.

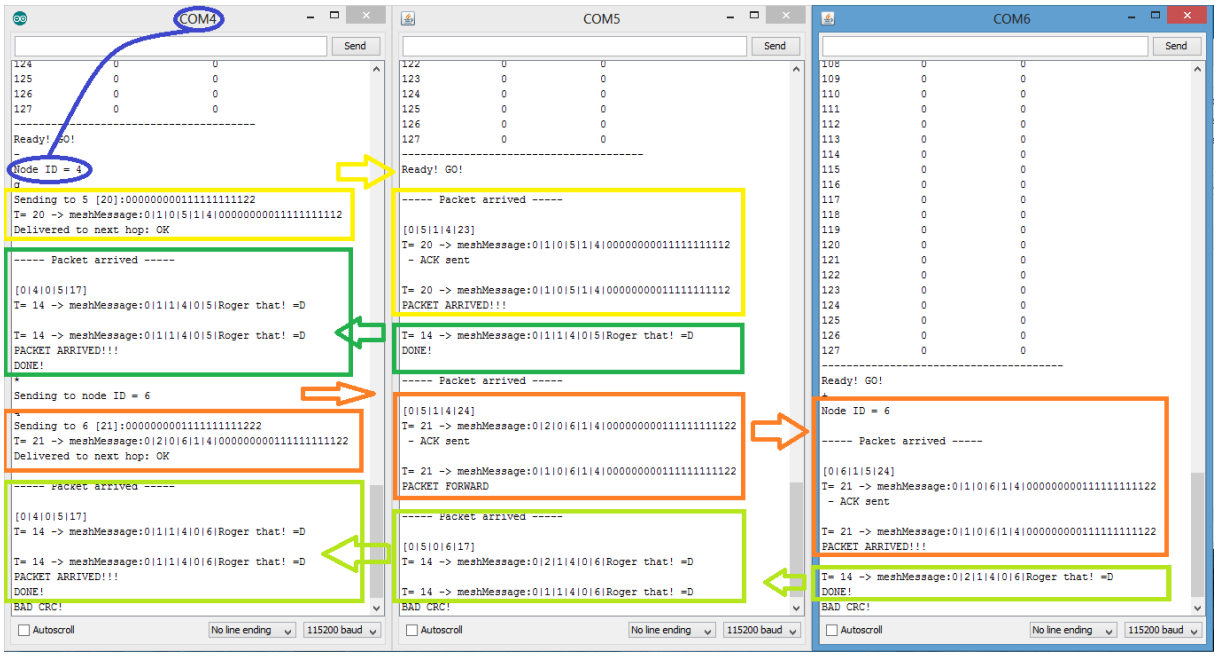

Figure 5.4: Forwarding test output between three nodes. Two requests are sent. The first is sent to a direct link, and the second to a forward link.

In the test, two messages were sent. Node 4 sent the first test message to a direct link (node 5), which successfully replied, as expected. The second message was sent to node 6, which was connected to node 5 at that time, thus forwarding the request and reply messages.

The success of this test represented a major achievement in this project, allowing a complete wireless building coverage using low-cost modules.

5.2.3 Auto-commissioning

One of the main goals of our solution is an easy installation. To achieve this, an EnergyBrick must automatically join the mesh network without any previous configuration. To perform the auto-commissioning test, an out-of the box Arduino was connected to the wireless module and uploaded with our solution. The results are depicted in Figure 5.5.

In the presented test, the EnergyBrick connected itself directly to the master node, however, it is still possible to connect through any other node, as the messages are forwarded in the network. In both situations, the auto-commissioning only takes a few seconds.

The success of this test represents a major achievement in this solution, as it will allow a very fast solution deployment of the final product.

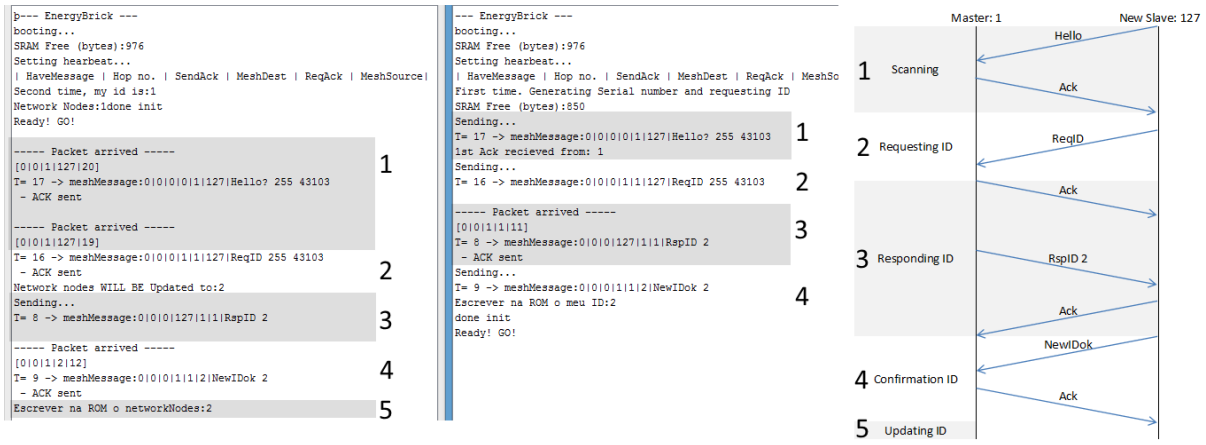

Figure 5.5: Auto-commissioning test. On the left, the output from the master. On the middle, the output of the joining slave. On the right, a flow of the exchanged messages. When a wireless mesh node is entering the system for the first time, it performs a scan, to check available nodes (1) and then selects one to request an ID to the master (2), which replies with a new response ID to this node (3). To finalize the process, the entering node sends a confirmation (4) and the master node stores this new network change(5).

5.2.4 Reverting to default settings

When installing our solution, the auto-commissioning feature is automatically executed during the first boot, storing permanently the resulting configuration in the EEPROM for further boots. However, due to the simple installation, the client may desire to perform sub-metering in another building area using the available EnergyBricks. In such situation, the EnergyBricks are no longer in the same positions and the configuration may not be valid, especially the mesh network, where some nodes may be out of range in the forwarding table.

To solve this problem, a reset feature was implemented, to enable clearing all EEPROM configurations. Such action will put the EnergyBrick in a initial state, and the auto-commissioning feature takes place, forming new valid configurations. To perform this action, the client only has to push a button for 2 seconds when powering up the EnergyBrick. The status LED will become always on, until it connects to the mesh network and starts blinking regularly again. This button is connected to the Analog 4 pin in the Arduino and when pressed it drops the analog value to 0. When booting, this value is checked, and if it is 0, then the reset is performed.

This feature allows us to easily perform the auto-commissioning test, thus being very useful for situations where the nodes have to be installed elsewhere.

5.2.5 Message using Ethernet

To connect an EMS to the metering devices, a link must be established. In our solution, an Ethernet connection is used to link both the central server and the Energy Gateway Brick. To ensure the correct functioning of this module, several tests were performed:

- **Hardware connections tests** were made to ensure a proper operation. The test consists of sending a simple ping message to the Arduino and getting a reply from it.

- **UDP tests** were also performed to verify that a custom message can be sent and received. In this test, the Arduino worked as an echo server.
- **Modbus UDP tests** ensure that the Modbus TCP stream can be sent over an UDP datagram and replied back with the Modbus response.

5.2.6 Measuring Energy

The main focus of this work is to perform energy metering reports to the central server. In our solution, there are two different ways to perform the measurements: *i)* direct metering and *ii)* requesting measurements to a smart-meter. For those different measurement methods, different tests were performed.

Requesting the Energy Meter Brick

To perform this test, a Modbus request were sent to the Energy Meter Brick who were performing consumption readings. The Modbus response includes: *i)* total consumption, *ii)* instantaneous current, *iii)* instantaneous power factor, *iv)* instantaneous real power, *v)* instantaneous apparent power and *vi)* instantaneous main voltage.

To calibrate our meter for the energy measurements, an external energy meter was used. With a correct calibration, several requests were performed, obtaining the graphical consumption representation presented in Figure 5.6.

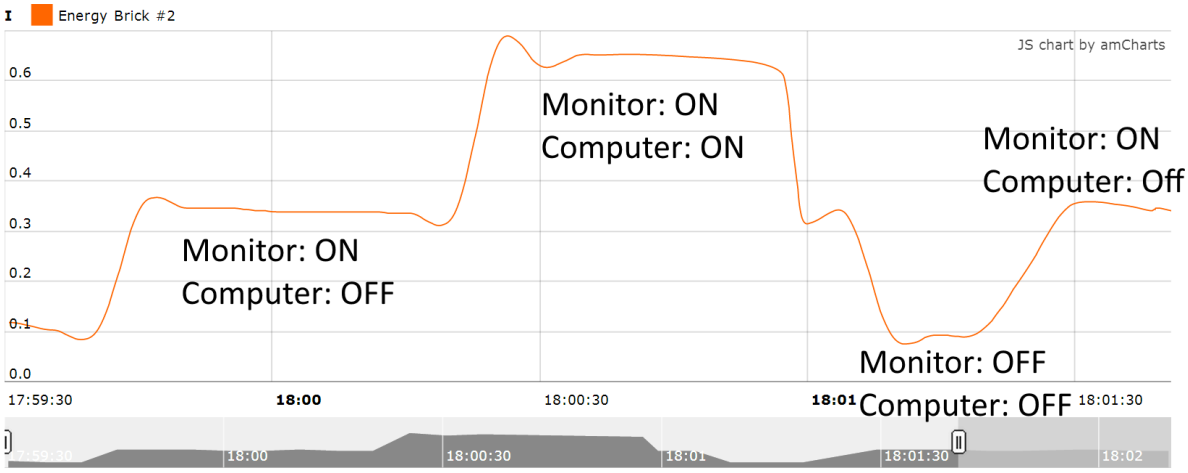

Figure 5.6: Energy Meter Brick graphical consumption representation. In this graphic, a consumption representation of a computer and a monitor is presented.

Requesting the Energy Meter Inquire Brick

To include a smart-meter in our metering solution, the Energy Meter Inquire Brick must be able to communicate with it and request its energy data. In our solution, the Modbus TCP request is converted to a Modbus RTU request and sent over the serial line to the smart-meter.

To test the correctness of this functionality, two tests were performed: *i)* connecting to a Modbus RTU slave installed in the laptop, and *ii)* connecting to a smart-meter that acts as a Modbus RTU slave.

When connected to the laptop, as presented in Figure 5.7, the laptop acts not only as a central server but also as a Modbus RTU slave. The Modbus UDP request goes from the laptop Ethernet port to the Energy Gateway Meter Brick, which forwards the request to the Energy Meter Inquire Brick. In turn, the Inquire Brick creates a Modbus RTU request based on the original request and sends it to the Modbus RTU slave, which is an application running in the laptop, using a serial RS-485 communication. The application successfully receives the request and produces a response that travels back to the central server application.

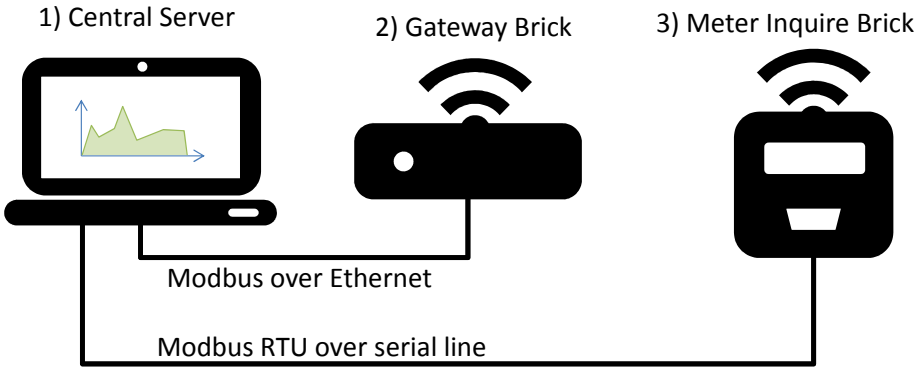

Figure 5.7: Modbus RTU test 1. The Energy Meter Inquire Brick was connected to the laptop, which acted as a central server, and also as a Modbus RTU slave. The Modbus UDP request goes from the laptop (acting a central server) through the Ethernet port, reaching the Energy Gateway Brick (in the middle) who forwards the request in the wireless mesh network to the Energy Meter Inquire (in the right). After receiving the Modbus UDP request, the Energy Meter Inquire converts it to a Modbus RTU request and sends it to the Modbus RTU slave application (in the laptop) through a serial line, which then replies to the Energy Meter Inquire. The response is then converted to Modbus UDP and sent to the Energy Gateway Brick who forwards it to the central server.

To test the EnergyBrick with a smart-meter, an Electrex Zepto D6 was used as a smart-meter. The Figure 5.8 depicts the setup scenario.

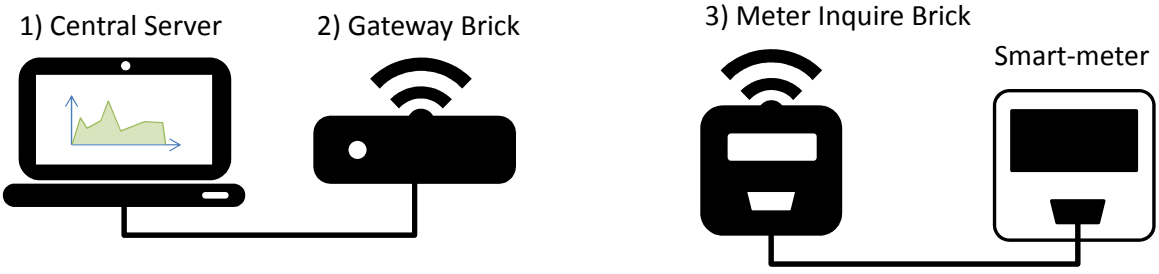

Figure 5.8: Modbus RTU test 2. The EnergyBrick was connected to a smart-meter to test the conversions between Modbus messages. In this test, the Energy Meter Inquire was connected to the Electrex Zepto D6 smart-meter to request energy measurements.

In this scenario, the obtained results were valid Modbus exceptions messages, indicating that the tested function codes were not developed in the smart-meter. To validate this result, the laptop was connected directly to the smart-meter, as depicted in Figure 5.9, using a serial line, and the same

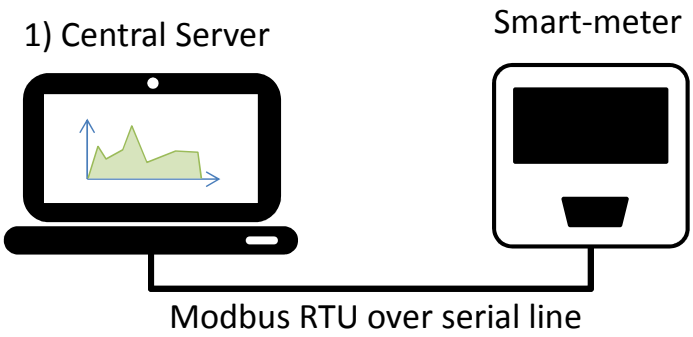

Figure 5.9: Modbus RTU test 3. The laptop was connected directly to the Electrex Zepto D6 smart-meter to confirm the previous results in test 2.

requests were made. The resulting responses were exactly the same, thus confirming the results.

5.2.7 Configuration tests

To ensure a completely customizable solution, all EnergyBrick settings can be changed if needed. In our solution, it is possible to perform these changes in two ways: *i)* locally, using a USB connection to access the option menu, or *ii)* remotely, by sending Modbus packets through the wireless mesh network.

To perform all the previous tests, several configurations were performed to prepare them. The developed option menu presented in Figure 5.10 makes the set up process easier. It was the main configuration feature used due to most of the EnergyBricks being connected directly to the laptop. However, several configurations were also performed remotely, using the Modbus protocol to more distant EnergyBricks that were not connected to the laptop during the performed tests.

5.3 Low-cost solution

The main objective in this thesis is to present a low-cost energy meter. There are a lot of energy meters in the market, yet they are too expensive to be spread all over a large building. Using open-hardware solutions such as Arduino, we developed several low-cost EnergyBricks that are detailed in this section. The Arduino was chosen for this project not only for its advantages of being programmable in C++ language, and enabling the use of extension modules, but also for their low-price. The Uno model was used due to its unique feature of easily removing the Atmel microcontroller to include it in a custom PCB. This feature can speed up the production while maintaining a low-cost product.

The table 5.2 presents all the different components used in our solution and their respective prices.

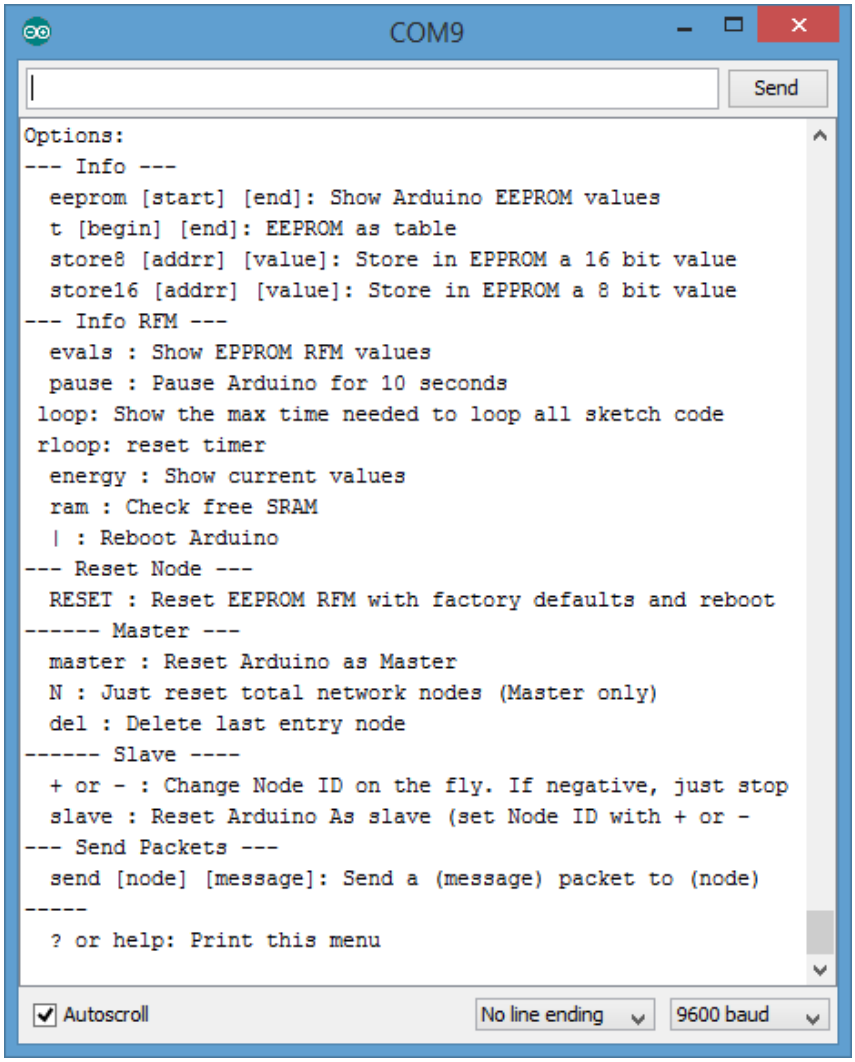

Figure 5.10: Options Menu that allows to locally and directly perform changes in the EnergyBrick using the input text box at the top.

Component	Module	Hardware	Price (€)
Processor	Microcontroller	Atmega328	4,5
Metering	Meter	Current Transformer	10
		Voltage Transformer	10
	Meter Inquire	RS485 Breakout Board	10
Communication	Ethernet	ENC28J60	10
	Wireless	RRM12b	7

Table 5.2: Expansion models component pricing of the EnergyBrick. This table presents all expansion modules and prices used in the EnergyBrick to developed a low-cost solution.

Chapter 6

Conclusions

With the herein presented work, we have developed a wireless mesh networked energy metering solution that enables a fast installation and a tailored configuration of the metering area.

Clients can define the meters' locations, thus enabling sub-metering, and can later adjust these locations at will, in order to achieve a more accurate sub-metering in areas that suggest potential consumption problems or deviations.

A striking aspect of our solution is that it successfully stacks Modbus protocol atop a mesh network RF protocol, which has been developed and tested over the low-cost RFM12b chip.

The assembled solution allows metering energy consumption at a very low-cost. This small investment allows significant savings, as it can provide users with feedback on where energy is being used and wasted. Consumption savings combined with low-cost enable a return of investment faster than other similar solutions.

This solution is also very flexible, especially when compared to cabled solutions, where rearrangements are difficult, if not impossible, without the acquisition of additional equipment. By using low-cost meters, the installation cost can be neglected and the installation of more energy meters is more easily considered, with a better sub-metering detail. This allows offering a less expensive solution for the client than similar available solutions, as well as a more adequate and accurate solution, therefore a better service.

A major advantage of our solution is that depending on the already installed metering solution in the client, our solution can be even more affordable. The Energy Meter Serial Brick has a lower production cost than the Energy Meter Brick, therefore, installing this model in places where a smart-meter is already installed can reduce the final solution cost for the client. Such advantages can be targeted at home metering, or at the small and medium enterprises market.

Open-hardware platforms and open-source software bring new possibilities to develop products that otherwise could not be performed by a single person. These new possibilities allowed us to develop a system that can be compared to several existing solutions in the market, and to easily add new features by adding new modules, which can bring a competitive advantage. Besides being a metering solution, consumption control can also be easily implemented by adding a control module that allows to turn any

kind of equipment on or off, such as a lamp or heat radiators, or controlling the power consumption of equipments that can be regulated, such as electronic ballasts that allow light dimming. Such a remote control would allow full automation scenarios and increase occupants' comfort, while enabling a more proactive and effective energy saving.

6.1 Achievements

The developed work using open solutions allowed us to achieve interesting results. In greater detail, the achievements of this work are as follows:

- **EnergyBrick prototypes** are developed in this thesis and successfully tested in a mesh network. With this scenario setup, a starting point for further development was achieved, proving that it is possible to control energy consumption with open-hardware and electronics.
- **The combination of expansion modules** in a single Arduino, enabled us to develop an EnergyBrick capable of using all modules at once. However, due to flash memory limitations, it was not possible to upload all the code needed to operate with all the modules at once, therefore, several models were developed, where each one uses combinations of this low-cost hardware.
- **The understanding of the Modbus protocol** allows a simple application protocol implementation that can work in already existing systems, or implement a software solution to communicate with the EnergyBricks to perform remote energy metering, that can be performed by direct consumption reading or by inquiring already installed smart-meters. Using this protocol, it is also possible to remotely configure all EnergyBricks, being those settings persistently stored in each one.
- **The wireless mesh network** was also a major achievement in this thesis. Without it the system communication would have been very limited. In this solution, an auto-commissioning mechanism was also implemented, allowing an EnergyBrick to be easily installed in the network. Regarding this mechanism, all default configurations are performed for the installed modules, which can later be remotely customized by the central server.

6.2 Lessons Learned

During this thesis, several problems occurred. However, all of them were overcome, having as end result the creation of several EnergyBrick prototypes, as well as some valuable lessons:

- **Nothing is as simple as it seems.** In the beginning, we participated in a 2-week hacker event in Instituto Superior Técnico called LXReactor [Pub13] , during which a simple energy meter able to measure current, was developed. During this time, the used hardware caused a major problem on performing measurements, which was overcome by adapting the resistors to calibrate the measurements. The event resulted in a first prototype[Hac13], depicted in appendix B.1.

- **Connecting two expansions modules is not a straightforward task.** After the LXReactor event, communication features were added to the Arduino. During this time, another problem occurred. When connecting more than one expansion module into a single Arduino board, the used pins could be the same, thus causing problems, e.g., if the interrupt pins and the chip-select pins are the same, then both modules may not work. This problem was solved by editing the hardware libraries to change the pins.

- **The microcontroller storage memory may not be enough.** When developing the solution, the initial idea was to upload the entire code to the Arduino, allowing to change the modules as needed without any code update. However, we faced serious problems when including communication modules: as these modules use a high amount of flash memory, when combining both, almost all the memory was used, and the entire code did not fit in a Arduino Uno microcontroller.

- **RAM is also a serious problem in microcontrollers.** Running out of RAM happened many times while testing, when using too many variables, or large vectors. Usually, to perform debug in these microprocessors using the Arduino IDE, a lot of *print lines* are used to find out the problem's source. In the Arduino microprocessors, those *print lines* are also stored in the SRAM, and their usage can actually cause the exact "out of RAM" problem that we were trying to solve. When this happens, the Arduino just reboots while performing a task and keeps rebooting until the problem is solved in the sketch code. A solution for this problem is storing the *print lines* in the flash memory using the *F command* provided by Arduino. Besides being a better solution, it can lead to the first memory related problem of out of flash memory when debugging.

- **Memory address violations are not trivial to detect.** Usually, when programming in C language, this problem might occur and the application stops with a *Segmentation Fault*. In the case of Arduino, it just writes on the SRAM without any special concerns, overwriting previous stored values, which in turn causes system instability and stops the microprocessor from working. A problem that occurred during the development stage, was overfilling a buffer that was used to send a message. The previous memory access was in a completely different part of the code, to store the booting parameters of the RFM12b module. As a result, the RFM12b boot process failed with no feedback, and the microprocessor just blocked when trying to send a message in another completely different part of the code, due to the RFM12b not being properly ready to send or receive messages.

- **Uploading code to multiple Arduino is not simple, and is time consuming.** Several Arduino boards are used, causing a problem in this earlier stage, where the code upload must be performed to all, and each one uses a different COM port. Since there were several EnergyBricks, each one with its own modules installed, and a single sketch to be uploaded, every time that an upload had to be performed, the sketch definitions had to be changed, and the COM port also had to be changed in order to upload the sketch to the specific Arduino board.

Besides all these problems, in the end a fully functional EnergyBrick was successfully developed. The big lesson is that in this new age of open electronics platforms, everything is possible. Even if we do not have specific hardware to replicate a solution, a similar hardware can still be used, adapting its behaviour to match a desired function. In this new age, where the Internet in the palm of our hands, this solution can also be remotely controlled, thus giving us control not only of our virtual world, but also of our real world, merging both to make life better.

6.3 Future Work

In a short term, a final product could be created based on these prototypes, however, several aspects must be overcome first.

- **Flash memory** stores the sketch code, nevertheless the Arduino Uno microcontroller is not able to store all the developed code. This limitation do not allow hardware changes without new code upload. To solve this problem, there are two possibilities: *i)* using other Atmel microprocessor with a bigger flash memory, such as the ATmega 2560 used in Arduino Mega2560, which will increase the final product value, but also allows more complex functions, or *ii)* reduce the code size to fit into the Arduino Uno microprocessor Atmel ATmega328.

- **Alternative Atmel microcontroller** can also be used in this project. Since the code size reduction was always present during the development stage, and some features such as path discovery in the mesh network were not implemented due to this limitation, a new microcontroller may be needed. Besides the Arduino platform, which only provides core libraries to some Atmel microcontrollers, the Atmel company sells a much wider variety than can be used in this project. The Arduino platform does not support the core libs for these microcontrollers, however it is possible to develop additional libraries to support those new microcontrollers as needed, to be able to compile the code without any modification.

- **Adding new features** may be possible after solving the flash memory limitation. As mentioned before, the path discovery in the mesh network was not implemented due to the flash memory limitation. With this feature, the nodes could find an alternative way to communicate with the central server if a node failed due to some problem, such as running out of battery. Regarding this thesis, it is not a problem, since we are measuring energy, the EnergyBrick can be powered up with main electricity. However, problems can still occur, and this feature must be implemented. Another feature that can be implemented is the DHCP for a complete auto-configuration of the Energy Gateway Brick. It was fully tested, but was not included in the final release due to flash memory limitation.

- **Creating new modules** might be available in the future. Being a modular architecture can improve the final solution. An example is using a temperature and humidity sensor, to correlate the electrical consumption with those factors, thus allowing a much more accurate prediction. Since the I2C bus

was not used, it is still possible to include expansion modules that communicate in that bus, to improve the final product. To control power consumption costs, it is important not only to measure and find out possible consumption wastes, but also to act on those problems. To overcome this situation, a remote controller can be designed, with some extra modules. To test this idea, an EnergyBrick controller module was built and is presented in Appendix A. These potential new modules would add value to the final solution without significantly increasing the price.

- **Expanding the wireless mesh network** is also an objective for a near future. The objective is to increase the number of nodes to 65535 by using 2 bytes instead of 7 bits in the mesh frame. The EEPROM organization was implemented taking this idea into account, and for that reason the mesh forwarding table entries are on the last EEPROM addresses. This way, the scalability of the mesh network only depends on re-formatting the mesh frame. The implemented libraries to store and read 2 bytes from the EEPROM will also allow a quick development of this solution.

- **Creating a final product** will be possible after all limitations are solved and all major functions implemented. The tools presented in this document are enough to produce a PCB board that can the starting point for a commercial product. In Figure 6.1 an example of an Arduino with the Ethernet and RFM12b module is depicted, which could be a first EnergyBrick Gateway PCB.

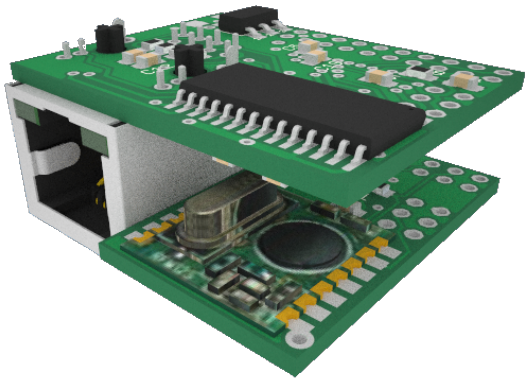

Figure 6.1: A possible EnergyBrick PCB representation. In this case, the Energy Gateway Brick having both communication modules presented in a compact format, that can be produced in large scale.

Bibliography

[ada13] adafruit. Adafruit online store. http://www.adafruit.com/products/904 (Last accessed on May 1st, 2014), 2013.

[Ard12] Arduino plugin for Sublime Text. https://github.com/Robot-Will/Stino, 2012.

[Ard13] Arduino Team. http://www.arduino.cc (Last accessed on May 7th, 2014), 2013.

[Bah06] M. Bahr. Proposed Routing for IEEE 802 . 11s WLAN Mesh Networks. In *Proceedings of the 2nd annual international workshop on Wireless internet*, WICON '06. ACM, 2006.

[BS06] C. Batini and M. Scannapieca. *Data Quality: Concepts , Methodologies and Techniques*. Springer, 2006.

[Cis07] Cisco. Brazil Extends Broadband Technology to Tiny Municipality to Promote Benefits of a Digital City, 2007.

[Dav10a] Dave. Desert home blog. http://www.desert-home.com (Last accessed on May 1st, 2014), 2010.

[Dav10b] Dave. Desert home: How i monitor power. http://www.desert-home.com/p/test-html-code.html (Last accessed on May 1st, 2014), 2010.

[Dig08] Digi. Wireless Mesh Networking: ZigBee vs. DigiMesh. Technical report, Digi International, 2008.

[Dig13a] Digi. http://www.digi.com/ (Last accessed on May 5th, 2014), 2013.

[Dig13b] Digi. The DigiMesh Networking Protocol. http://www.digi.com/technology/digimesh/ (Last accessed on May 1th, 2014), 2013.

[Dut14] Debasish Dutta. Arduino energy meter. http://www.instructables.com/id/arduino-energy-meter/ (Last accessed on May 1st, 2014), 2014.

[Ele81] Schneider Electric. Schneider-electric: Make the most of your energy. http://schneider-electric.com/ (Last accessed on May 6th, 2014), 1981.

[Ele94a] J&D Electronics. J&d electronics: The key for smart grid. http://www.hqmeter.com/ (Last accessed on May 1st, 2014), 1994.

[Ele94b] J&D Electronics. Wi-j&d: Wireless energy meter. http://www.hqmeter.com/wireless-energy-meter.html (Last accessed on May 1st, 2014), 1994.

[ELK06] ELKOR Technologies Inc. Introduction to Current Transformers. Application Note - AN0305, May 2006.

[Eur11] European Commission. *Energy 2020 - A strategy for competitive, sustainable and secure energy*. European Comission, 2011.

[Fri13] Fritzing. Fritzing: eletronics made easy. http://fritzing.org/ (Last accessed on May 6th, 2014), 2013.

[Gin04] K. Gingerich. The RS-485 unit load and maximum number of a bus connections. *Analog Applications Journal*, pages 21–24, 2004.

[GP07] J. Gonzalez and M. Papa. Passive Scanning in Modbus Networks. In *Critical Infrastructure Protection*, volume 253 of *IFIP International Federation for Information Processing*, pages 175–187. Springer US, 2007.

[Gro11] OMS Group. Open Metering System Specification. Technical report, January 2011.

[Gro12] Smart Energy Groups. Smart energy groups online store. https://shop.smartenergygroups.com (Last accessed on May 1st, 2014), 2012.

[Hac13] Hacker School. http://hackerschool.ist.utl.pt/lxr-projects-energybrick/ (Last accessed on May 10th, 2014), 2013.

[HL07] E. Hossain and K. Leung. *Preface*. Springer US, January 2007.

[IEE11] IEEE P802.11 - Task Group S. Status of Project IEEE 802.11s: Meetings Update. http://grouper.ieee.org/groups/802/11/Reports/tgs_update.htm (Last accessed on May 5th, 2014), 2011.

[Ink03] Inkscape. Inkscape: Draw freelly. http://inkscape.org/ (Last accessed on May 6th, 2014), 2003.

[Int47] International Organization for Standardization. http://www.iso.org/iso/home.html, 1947.

[ISO11] ISO 50001 - Energy management. http://www.iso.org/iso/home/standards/management-standards/iso50001.htm, 2011.

[jme12] jmengel. Arduino home energy monitor shield. http://www.instructables.com/id/Arduino-home-energy-monitor-shield (Last accessed on May 1st, 2014), 2012.

[Kic11] J. Kickliter. Chasing Trons. http://chasingtrons.com/main/2011/3/24/energy-meter.html (Last accessed on May 1st, 2014), 2011.

[Lev89] Leviton. Leviton homepage. http://www.leviton.com/ (Last accessed on May 1st, 2014), 1989.

[Lev12] Leviton. Verifeye: Submetering solutions. http://www.leviton.com/verifeye (Last accessed on May 1st, 2014), 2012.

[M-B97] M-Bus Usergroup. The M-Bus: A Documentation, Revision 4.8. Technical report, M-Bus Usergroup, 1997.

[Mar11] M. Margolis. *Arduino Cookbook*. O'Reilly, March 2011.

[MCS+10] X. Ma, R. Cui, Y. Sun, C. Peng, and Z. Wu. Supervisory and Energy Management System of large public buildings. In *Mechatronics and Automation (ICMA), 2010 International Conference on*, pages 928–933, 2010.

[Mes02] Mesh Dynamics. Mesh Dynamics: Wireless for the Outdoor Enterprise. http://www.meshdynamics.com/ (Last accessed on May 5th, 2014), 2002.

[MI06] Modbus-IDA. Modbus Messaging on TCP/IP Implementation Guide. Technical report, Modbus Organization, October 2006.

[Mod04] Modbus-IDA. Modbus protocol transferred to modbus-ida. Technical report, Modbus Organization, April 2004.

[Mod05] Modbus homepage. http://www.modbus.org, 2005.

[Mod06] Modbus Organization. MODBUS over serial line-Specification and Implementation guide, 2006.

[Mod12] Modbus Organization. Modbus Application Protocol Specification V1.1b3. Technical report, Modbus Organization, April 2012.

[Mot08] Motorola Inc. Mesh Networking: A Revolution in Anywhere, Anytime Wireless Connectivity. Technical report, Motorola, 2008.

[Mot13] Motorola. Motorola Mesh Wide Area Networks, 2013.

[Nat12] National Energy Education Department. *Secondary Energy Infobook*. National Energy Education Department, 2012.

[Ope13a] OpenEnergyMonitor. http://openenergymonitor.org (Last accessed on May 17th, 2013), 2013.

[OPE13b] OPENmeter. http://openmeter.com/ (Last accessed on May 17th, 2013), 2013.

[Ope14a] OpenEnergyMonitor. Emoncms: Open-source energy visualisation. http://emoncms.org (Last accessed on May 1st, 2014), 2014.

[Ope14b] OpenEnergyMonitor. Github emoncms. https://github.com/emoncms/ (Last accessed on May 1st, 2014), 2014.

[Ope14c] OpenEnergyMonitor. Openenergymonitor online store. http://shop.openenergymonitor.com (Last accessed on May 1st, 2014), 2014.

[Par13] Parallax Inc. Propeller: General Information. http://www.parallax.com/ (Last accessed on May 1st, 2014), 2013.

[Pio10] N. Piotrowski. True Power vs. Apparent Power: Understanding the Difference. Technical report, Associated Power Technologies, 2010.

[Pub13] Publico. http://p3.publico.pt/vicios/hightech/8656/lx-reactor-e-um-campo-de-ferias-para-geeks (Last accessed on May 10th, 2014), 2013.

[Raj13] Rajant. http://www.rajant.com/ (Last accessed on May 10th, 2014), 2013.

[RGK02] K. Roth, F. Goldstein, and J. Kleinman. *Energy Consumption by Office and Telecommunications Equipment in Commercial Buildings*, volume I. Arthur D. Little, January 2002.

[Sch11] M. Schmidt. *Arduino: A Quick-Start Guide*. Pragmatic Bookshelf Series. O'reilly and Associates Incorporated, 2011.

[Sch13a] Marco Schwartz. Open home automation blog. http://www.openhomeautomation.net (Last accessed on May 1st, 2014), 2013.

[Sch13b] Marco Schwartz. Power monitoring with an arduino board and the ina219 sensor. http://www.openhomeautomation.net/power-monitoring-arduino-ina219 (Last accessed on May 1st, 2014), 2013.

[Sma12] Smart Energy Groups. http://smartenergygroups.com/ (Last accessed on May 1st, 2014), 2012.

[Sub08] Sublime Text. http://www.sublimetext.com, 2008.

[Xiv03] Xively. Xively by logmeln. https://xively.com (Last accessed on May 1st, 2014), 2003.

[Zig02] ZigBee Alliance. ZigBee Alliance: Controll your world. http://www.zigbee.org/ (Last accessed on May 6th, 2014), 2002.

[Zig11] Zigbee Alliance. ZigBee Smart Energy Profile Specification, 2011.

[Zig13a] Zigbee Alliance. ZigBee Certified Frequently Asked Questions, 2013.

[Zig13b] Zigbee Alliance. ZigBee Smart Energy Certified Products. http://www.zigbee.org/Products/ByStandard/ZigBeeSmartEnergy.aspx (Last accessed on May 6th, 2014), 2013.

Appendix A

Actuating using the EnergyBrick

A.1 Control module

In order to save energy, being able to control its usage is crucial. Our EnergyBrick module enables a remote control of eletrical equipments. In this module, the control can be performed in two ways:

- **On and Off** feature uses a digital impulse to control the electric flow, allowing it or not to pass through. It works as a local circuit breaker, allowing to remotely turn any kind of equipment on or off, including lamps, socket walls, computers, printers, or even an entire room or building, depending on where it is installed. This control is possible by using an Arduino digital port connected to the relay, as depicted in Figure A.1.

Figure A.1: This mechanical relay can be used to control the electrical flow in a circuit using low-voltage levels. This can be used to allow an Arduino to control lamps or other electrical devices.

- **Dimming** feature uses an analog value to control more sophisticated devices, such as an electrical ballast for fluorescent lamps, which can dim the output light level instead of just be turned on or off. To do this, a ballast has an extra input sensor, where a voltage level can be applied to control it. Since this dimming level is controlled by voltage, and the Arduino can output an analog voltage level, it is possible to control the dimming.

With this control module, an update of the block diagram can be performed, as presented in Figure A.2.

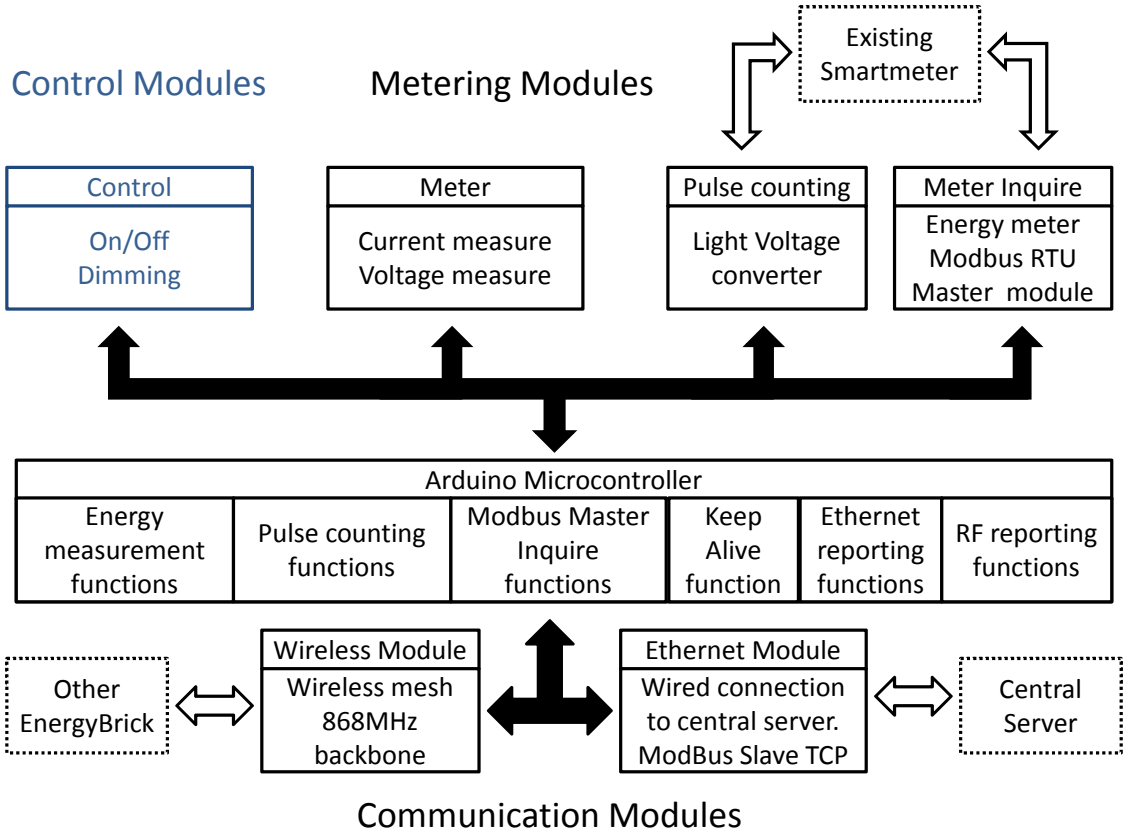

Figure A.2: EnergyBrick block diagram, updated with the control module.

A.2 Energy Controller Brick

When performing energy metering, the main goal is to acquire enough data, which will produce useful information, in order to realize where the energy is being wasted. However, to start saving energy, actions must be taken, and with this Energy Controller Brick, presented in Figure A.3, it is possible.

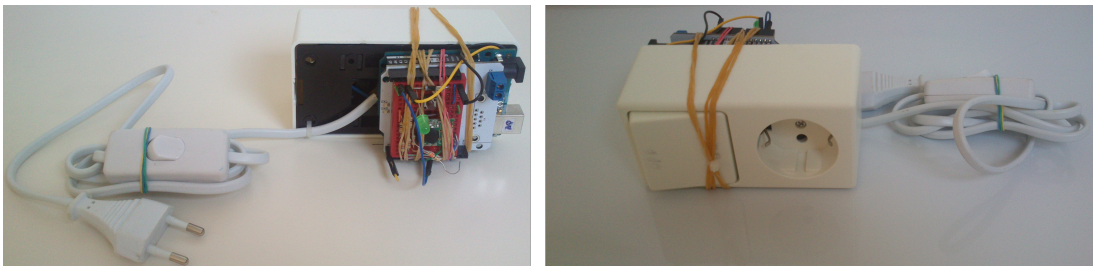

Figure A.3: This Energy Controler Brick prototype uses the controller module to remotely turn on and off a socket wall.

Since the Arduino has several digital ports, they can be connected to a relay and turn any device on or off. Additionally, it can also produce output voltage levels which can control any device that can be controlled with voltage levels, such as an electrical ballast, dimming the light.

Imagining that it is connected to a socket wall, controlling it, every electrical device connected to this socket can be controlled at the same time, to be turned on or off. Another example is connecting this EnergyBrick to a light interrupter, allowing a remote light control of a entire room. Better yet, it is imagining that we have a room with 5 lamps, and each one can be controlled independently, instead of having the control with a single interrupter. This allows the creation of automation mechanisms that can be better explored by users' needs, increasing their comfort while saving energy.

Since with the EnergyBrick it is possible to control every single device connected to the main power, or several at once, and the combination of some EnergyBricks can allow a customized solution for everyone, then it is also possible to create personalized usage scenarios, meaning that it is possible to orchestrate several actions with a single command. Imagine a teacher that enters the amphitheatre to give a lesson. While he is still entering, the stairs' lights can be the only ones turned on, which can be called the "entering/exiting room scenario" since the procedure is the same when leaving the room. When starting the lesson, those lights can turn off, and the ceiling and board lights turn on, which can be called "lesson scenario". In the meantime, if the teacher decides to turn on the projector, then the "projector scenario" is activated, meaning that the celling lights are dimmed, the board lights are turned off, the projector turned on, and the projection screen goes down, all in a single command performed by the teacher.

In the meantime, all this activity is being measured and taken into account by the central server for energy consumption purposes, allowing to understand how we can save energy while performing everyday tasks.

Implementation details

The Arduino by itself, only produces a voltage level of 5V in the digital output pin. However, if a 230V relay controlled by 5V is connected into the Arduino, it is possible to change the status when commuting the Arduino digital output level, as detailed in Figure A.4. When connecting the main cables to the relay, it is recommended to use the normally closed (NC) pin, to ensure that even if the EnergyBrick fails, the socket can still be used normally.

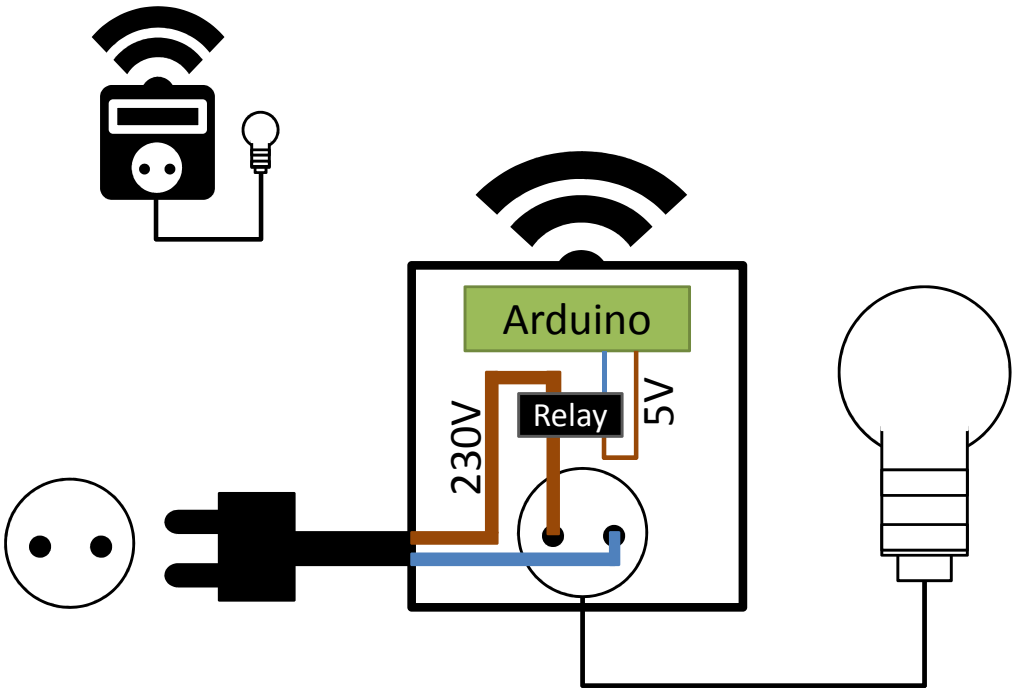

Figure A.4: EnergyBrick Controller Internal schematic

Appendix B

First EnergyBrick

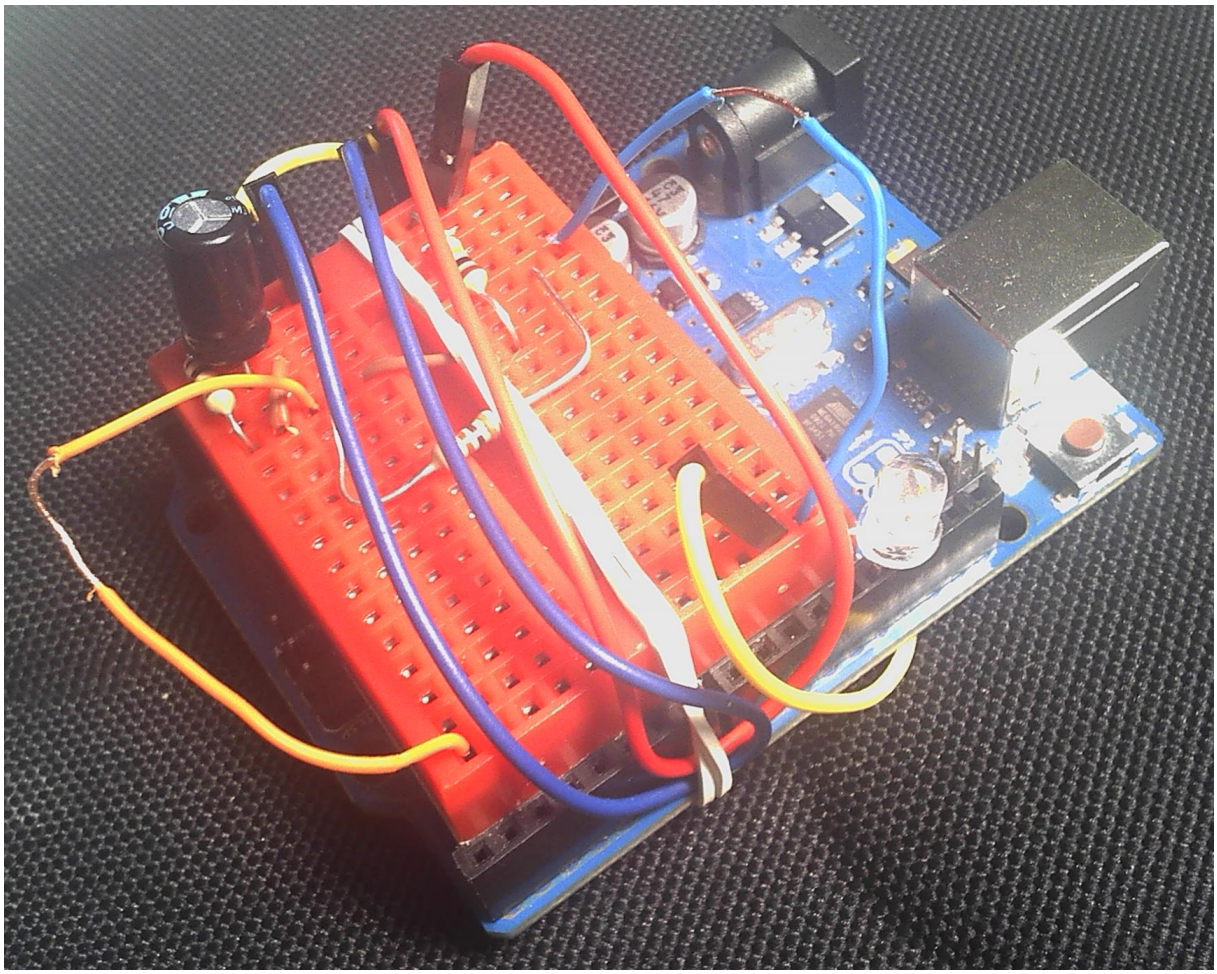

Figure B.1: First Energy Meter Brick prototype, developed in LXReactor during 2 weeks, to measure current consumption.