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 (installation 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.

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

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 [4]. 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. 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 [9] standard was created by the International Organization for Standardization (ISO) [8] 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.

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: price. 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 support efficiently the building manager to detect and correct possible problems. Such problems results in energy wast, 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.

Besides existing many EMS solution in the market all of them have common problems that may be solved in order to increase their adoption.
Regarding this thesis, several requirements were identified: a functional model, but useless to solve the initial problem. Performing a good requirement analysis can lead to a very important to understand the user needs and requirements. Not having these interactions with specialized software. All these problems still exist in EMS solutions, 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.

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.

2. 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 [1] an example of this. With Arduino is possible to control sophisticated [15] into something that changes the world [10]. The Fritzing [5] and the Inkscape [7] are design software tools that helps to develop a solution.

2.1 Solution Overview

2.1.1 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 meters price is too high to install hundreds of them in a large building. Not allowing 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.

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.

Energy readings must be performed with a non-invasive technique.

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 [14], developed by Schneider Automation [2] in 1979, and maintained since April 2004 by an independent group called Modbus IDA [11, 6], must be used to communicate with the central server to report data.

Failure proof maintaining configuration values even after an electrical outage.

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.

2.1.2 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 [12] 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 [13] 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 Current Transformer (CT) allows the system installation without shutting down the building’s power.

• Controllability is also important in a EMS. In our solution, it 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 easing 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 offering new features without losing any of the previously made configurations.

2.1.3 Block Diagram

This solution consists of modular architecture of distinct modules that handle different problem scenarios. Only the necessary modules are used in a specific problem reducing production costs for each meter in a final implementation.

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.

In our solution, there are: i) metering modules, which performs energy measurements, and ii) communication modules that report those measurements, as depicted in Figure 1.

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.

• Meter Inquire module which can perform requests to already installed smart-meters. Connecting a simple UART - RS-485 converter, the Arduino can communicate using the Modbus RTU protocol in a RS-485 serial line to inquire the smart-meter about the energy readings.

• 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 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.

Communication modules

In our solution two communication modules are used:

• 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, which currently has the lowest price in the market, allowing this communication without increasing too much the price.

• Radio Frequency module provides a wireless solution communication between EnergyBricks forming a mesh network 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.

3. 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) Gateway, 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 2 uses combinations of the presented modules to perform a single task that will be described in next sections.

Figure 2: 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.

3.1 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 4.1.

- **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 4.2.1.

- **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. Finally, a auto-commissioning feature is provided, which automatically set up the EnergyBrick in the system.

3.2 Gateway Brick

The Gateway Brick is the entry point of our network, as depicted in the Figure 3. 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.

To lower the flash memory footprint an UDP solution with Modbus TCP was developed, which will be detailed in the Section 4.2.1.

3.3 Energy Meter Brick

The Energy Meter Brick, presented in Figure 4, measures consumption using the Metering module, and report those readings to the central server using Radio Frequency module.

Typically to measure the current in a circuit, an amperemeter have to be installed. This installation requires breaking the circuit in an evasive way to force the current to pass through the amperemeter 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" [3]. The induced current is smaller than the current passing through the cable, but it can be calculated using the CT scale. Having the induced 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 induced 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 known. Since that measuring the main voltage directly is dangerous, and could easily break the Arduino, we use an AC/AC Voltage Transformer (VT) to reduce the scale. 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 inputs 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 can produce a voltage output) to the Arduino analog inputs, a simple circuit is 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 -230V and +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 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 measure safely, since the resulting wave, depicted in Figure, is safe to the Arduino board.

![Figure 5: The graphic represents the transformation that must be performed in the main voltage to perform its reading using an Arduino analog input port.](image)

### 3.4 Energy Meter Serial Brick

The main function of the Energy Meter Serial Brick, presented in Figure 6 is to request data to a already installed smart-meter.

![Figure 6: 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.](image)

This Brick uses the Meter Inquire Module presented in Section 2.1.3, 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.

4. 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.

4.1 EEPROM Organization

Providing a system failure mechanism, 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 store all the configuration data as a system failure mechanism.

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, allowing 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.

4.2 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 4.2.1, and the second was the communication between the EnergyBricks detailed in Section 4.3.

4.2.1 Application protocol

The application protocol defines how an external application can interact with the EnergyBricks modules. In this case, we use the Modbus TCP. To save flash memory the Modbus TCP 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.

From all functions defined in Modbus protocol, only two were considered in our case to manipulate the EEPROM configuration data as a Holding Registers table: i) Function code 3, to read multiple values, and ii) Function code 16 to write multiple values.

The Energy Meter Inquire Brick has a special feature that allow it to convert between Modbus TCP and Modbus RTU to request the smart-meters.

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 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 forwarding the request.

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.

4.3.1 Frame Format

To implement the mesh network, additional fields has to be added in the link packet payload. To provide the mesh solution, similar fields as already provided by the RFM12b library were added to extend this link-to-link communication, as presented in Figure 7

![Figure 7: RFM12b packet used in the mesh network. The darker section represents the changes made to implement our mesh solution.](image)

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.
- **DataLength** 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.

### 4.3.2 Mesh Routing

In the mesh network, the packet must be forward over the network until it reaches the destination. For every given hop, an acknowledgment must be performed back in order to the sender knows that the message was successful 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 acknowledgment 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 who 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 1.

<table>
<thead>
<tr>
<th>To Node ID</th>
<th>Next Node ID</th>
<th>Number of Hops</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>255</td>
</tr>
<tr>
<td>1</td>
<td>65535</td>
<td>65535</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 1: Example of the master forwarding mesh network table. Each EnergyBrick has its 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 8.

![Figure 8: 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.](image)

### 4.3.3 Mesh Commissioning

One of the functional requirements is the easy out-of-the-box equipment installation, allowing a self-install energy meter in the network without. When booting up it requests 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 9 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. 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 sends 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.

If the selected node is the GatewayBrick node, then the communication between the joining node and gateway is direct, and no forward mechanisms are used.

5. **EVALUATION**

The developed solution resulted in multiple EnergyBricks modules, each one with its own features. This section 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. To perform the solution evaluation, a real usage scenario was implemented using the developed EnergyBricks, as presented in Figure 10.

![Figure 10: 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.](image)

To test the wireless mesh network, a base network was established, and is presented in Figure 11. 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.

![Figure 11: Wireless Mesh Network tested to validate the solution. Nodes 1 to 5 are fixed and node 6 was used to perform commissioning tests.](image)

5.2 **Tests**

5.2.1 **Distance between nodes**

These tests allow us to conclude the maximum distance allowed between two nodes. The datasheet of the RFM12b
expansion modules claims to have an open air range above 100 meters.

To test the maximum distance between two nodes, a ping test were performed in 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 2.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Residential</th>
<th>IST Tagus</th>
<th>IST Alameda</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interference</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Walls</td>
<td>Yes (4)</td>
<td>No</td>
<td>Yes (1)</td>
</tr>
<tr>
<td>Distance (m)</td>
<td>50</td>
<td>40</td>
<td>20</td>
</tr>
</tbody>
</table>

Table 2: 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 12.

In this test, two dummy requests were sent: i) Between direct links (4 to 5) and ii) between indirect links (4 to 6), using a node to forward the message. 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 13.

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.

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 situations, 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 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 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.

5.2.6 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 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 14.

![Figure 14: Energy Meter Brick graphical consumption representation. In this graphic, a consumption representation of a computer and a monitor is presented.](image)

5.2.7 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 15, 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.

To test the EnergyBrick with a smart-meter, an Electrex Zepto D6 was used as a smart-meter. The Figure 16 depicts the setup scenario.

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 17, using a serial line, and the same requests were made. The resulting responses were exactly the same, thus confirming the results.

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++ lan-

![Figure 15: 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.](image)

![Figure 16: 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.](image)

![Figure 17: Modbus RTU test 3. The laptop was connected directly to the Electrex Zepto D6 smart-meter to confirm the previous results in test 2.](image)
guage, 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 3 presents all the different components used in our solution and their respective prices.

<table>
<thead>
<tr>
<th>Module</th>
<th>Hardware</th>
<th>Price (€)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microcontroller</td>
<td>Atmega328</td>
<td>4.5</td>
</tr>
<tr>
<td>Meter</td>
<td>Current Transformer</td>
<td>10</td>
</tr>
<tr>
<td>Meter Inquire</td>
<td>Voltage Transformer</td>
<td>10</td>
</tr>
<tr>
<td>Ethernet</td>
<td>ENC28J60</td>
<td>10</td>
</tr>
<tr>
<td>Wireless</td>
<td>RRM12b</td>
<td>7</td>
</tr>
</tbody>
</table>

Table 3: Expansion modules component pricing of the EnergyBrick. This table presents all expansion modules and prices used in the EnergyBrick to developed a low-cost solution.

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 such 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, and the need to reduce energy costs until 2020 can be a market advantage to us in time to market issues.

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

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.
- **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 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.

6.2 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. Other feature that can be implemented is the Dynamic Host Configuration Protocol (DHCP) for a complete auto-configuration of the Energy Gateway Brick. It was fully tested, but was not included in the final release due flash memory limitation.

- **Create 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 increasing to much 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 reformatting 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 18 an example of an Arduino with the Ethernet and RFM12b module is depicted, which could be a first EnergyBrick Gateway PCB.

7. REFERENCES