ALEXANDER TECHNOLOGICAL INSTITUTE OF THESSALONIKI

School of Technological Applications
Department of Computer Science Engineers

Τμήμα Μηχανικών Πληροφορικής ΑΤΕΙΘ

Bachelor of Science Thesis

Building a Testbed for the Internet of Things

Author: **Vasileios Karagiannis**
Registration Number: **07/3273**

Thesis Supervisor : **Dr. Jesus Alonso Zarate**
Head of M2M Department
CTTC, Barcelona, Spain

Thesis Advisor : **MSc Francisco Vazquez Gallego**
Senior Researcher
CTTC, Barcelona, Spain

Academic Supervisor : **Dr. Periklis Chatzimisios**
Associate Professor
ATEITHE, Thessaloniki, Greece

Barcelona, April 2014

Abstract

Over the past few years a technological outbreak has been noticed in the area of electronics and computer networks, granting eligibility to connect these fields in the direction of building intelligent systems. These systems have the ability to increase the growth of Information and Communication Technologies (ICT) in urban environments and to provide services able to significantly upgrade the general well-being of individuals and societies. The whole concept of an Internet of Things (IoT) is reviewed in this thesis, architectures are discussed and some notable commercial solutions are presented. We then build our own development platform by setting up a wireless network of sub-GHz devices and connecting it to a virtual Internet cloud. Moreover, software applications are designed to complete an End-to-End Machine to Machine (M2M) communication network and therefore, emulate an entire Internet of Things environment. The communication framework that is developed, allows us to research a variety of IoT concepts including the upcoming smart grid which is a significant improvement to the current electric power distribution system.

Keywords: Internet of Things, M2M Communications, IoT Cloud Platforms, Wireless Networks, Testbed, Gateway Engineering, Android programming, Electric Power Grid, Smart Grid.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my Erasmus placement supervisors Dr. Jesus Alonso Zarate Head of the M2M Department at CTTC Spain, and Francisco Vazquez Gallego Senior Researcher at CTTC Spain, for welcoming me into the world of research. Without their guidance this dissertation would not have been possible. Special appreciation also to my university supervisor Dr. Periklis Chatzimisios, Associate Professor at the Department of Computer Science Engineering (ATEI of Thessaloniki, Greece) for consulting and aiding me in writing the current thesis. Finally I would like to acknowledge the technical support of the Centre Tecnologic de Telecomunicacions de Catalunya (CTTC) for providing the facilities and the necessary equipment to produce and complete this thesis.

Vasileios Karagiannis

Table of Contents

INTRODUCTION **11**

1.1 Related Work 12
1.2 Thesis Overview 12

RESEARCH ON THE INTERNET OF THINGS **14**

2.1 Three-Layer IoT Model 14
2.1.1 The Perception Layer *14*
2.1.2 The Network Layer *15*
2.1.3 The Application Layer *15*
2.2 Internet of Things Components 15
2.2.1 End-Devices *17*
2.2.2 Communication Protocols for the IoT *17*
2.2.3 Internet of Things Operating Systems *23*
2.2.4 Gateway for the M2M Area Network *25*
2.2.5 Message Queues *26*
2.2.6 Application Layer Protocols for the IoT *29*
2.2.7 IoT Cloud Platforms *34*
2.3 The OpenRemote Platform 38
2.3.1 OpenRemote Controller *39*
2.3.2 OpenRemote Designer *39*
2.3.3 OpenRemote Control Panels *40*
2.4 Suggested Architecture for the Testbed 40

AN IOT TESTBED **42**

3.1 The PanStamp Project 43
3.1.1 PanStamps *43*
3.1.2 Base Boards *45*
3.1.3 PanStick *46*
3.1.4 Lagarto Servers *47*
3.2 Raspberry Pi 54
3.2.1 Raspberry Pi PanStamp Shield *57*
3.2.2 The Raspberry Pi as a Gateway *58*
3.2.3 Practical Implementation *59*
3.3 Tested Cloud Platforms 61
3.3.1 ThingSpeak *61*
3.3.2 OpenSense *62*
3.4 Security Aspect 63
3.5 The public IP approach 64

3.6 Alternative Components for the Testbed 65

IMPLEMENTED REAL LIFE SERVICES 68

4.1 Android Programing 68

4.1.1Google Maps Android API v2 69

4.2 Smart Parking Application 70

4.3 Geo Fencing Application 72

APPROACHING THE ELECTRIC POWER GRID 75

5.1 The Current Power Grid 75

5.1.1 Generation 76

5.1.2 Transmission 76

5.1.3 Substations 77

5.1.4 Distribution 78

5.1.5 Supervisory Control and Data Acquisition 79

5.2 The Need for Improvement 80

5.3 The IoT Perspective of a Smart Power Grid 81

5.3.1 Smart Grid 82

5.3.2 IEC 61850 and GOOSE 83

5.3.3 Phasor Measurement Units and GOOSE generation 83

5.3.4 Gateway for the GOOSE 84

CONCLUSIONS – FUTURE WORK 87

REFERENCES 89

List of Figures

Figure 1: Three-layer IoT model [24] 14
Figure 2: IoT components 16
Figure 3: IoT end-device 17
Figure 4: Zigbee representation on the OSI model 19
Figure 5: 6LowPAN representation on the OSI model 21
Figure 6: M-Bus representation on the OSI model 22
Figure 7: Modbus representation on the OSI model 23
Figure 8: Internet connected end-device network 26
Figure 9: Publish-Subscribe concept 27
Figure 10: Polling mechanism 31
Figure 11: Long-polling 32
Figure 12: WebSocket session 33
Figure 13: ThingSpeak channel [52] 35
Figure 14: OpenSense public Senseboard [31] 37
Figure 15: OpenRemote Designer [29] 39
Figure 16: Suggested testbed architecture 41
Figure 17: PanStamp modules [37] 44
Figure 18: PanStamp base board [32] 46
Figure 19: PanStick [34] 47
Figure 20: Lagarto architecture [39] 48
Figure 21: SWAP packet structure [42] 49
Figure 22: Lagarto SWAP monitor interface 52
Figure 23: Lagarto MAX event interface [38] 54
Figure 24: Raspberry Pi board (model B) [46] 55
Figure 25: Raspberry Pi (model B) design [45] 56
Figure 26: PanStamp shield for the Raspberry Pi [35] 57
Figure 27: PanStamp shield attached to the Raspberry Pi [35] 58
Figure 28: Connection among servers 60
Figure 29: Implemented end-device network with Internet connectivity 61
Figure 30: Temperature/humidity monitoring from ThingSpeak channel 62
Figure 31: OpenSense monitor/control interface 63

Figure 32: The public IP approach of an IoT testbed 65
Figure 33: Arduino Uno, MSP-EXP430G2, Nanode, STM32VLDISCOVERY (left to right) 66
Figure 34: BeagleBone Black, UDOO, MinnowBoard Max and HummingBoard (left to right) 67
Figure 35: Android connectivity to the end-devices 69
Figure 36: Android HTTP GET request 71
Figure 37: Smart Parking Android application 72
Figure 38: GeoFencing Android application 73
Figure 39: Power grid overview [51] 76
Figure 40: High-voltage transmission lines [51] 77
Figure 41: Distribution system [51] 79
Figure 42: PMU device 84
Figure 43: Mbed, Wiznet (left to right) 85
Figure 44: Gateway for the GOOSE 85

List of Acronyms

3G	3rd Generation
6LoWPAN	IPv6 Low Wireless Personal Area Network
AES	Advanced Encryption Standard
AGPL	Affero General Public License
API	Application Programming Interface
CoAP	Constrained Application Protocol
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CSV	Comma Separated Values
CTP	Collection Tree Protocol
DDR	Double Data Rate
EEPROM	Electrically Erasable Programmable Read Only Memory
FTSP	Flooding Time Synchronization Protocol
GOOSE	Generic Object Oriented Substation Events
GPIO	General Purpose Input Output
GPS	Geographical Positioning System
GPU	Graphics Processing Unit
GW	Gateway
HDMI	High Definition Multimedia Interface
HTTP	Hyper Text Transfer Protocol
ICT	Information Communication Technology
ID	Identity Document
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
IED	Intelligent Electronic Device
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
IPv4	Internet Protocol version 4

IPV6	Internet Protocol version 6
ISM	Industrial Scientific Medical
JSON	JavaScript Object Notation
JSONP	JavaScript Object Notation with Padding
LAN	Local Area Network
LED	Light Emitting Diode
LGPL	Lesser General Public License
LLC	Logical Link Control
LTE	Long Tern Evolution
M-Bus	Meter Bus
M2M	Machine to Machine
MAC	Media Access Control
MCU	Micro Controller Unit
MQTT	Message Queue Telemetry Transport
NAT	Network Address Translation
NIST	National Institute of Standards and Technology
NOOBS	New Out Of the Box Software
OS	Operating System
OSI	Open System Interconnection
PMU	Phasor Measurement Unit
RAM	Random Access Memory
RCA	Radio Corporation of America
REST	Representational State Transfer
RF	Radio Frequency
RFC	Request For Comments
ROM	Read Only Memory
RPL	Routing Protocol Low power
RTC	Real Time Clock
RTU	Remote Terminal Unit
Rx	Receive
SCADA	Supervisory Control and Data Acquisition
SMA	SubMiniature version A
SPI	Serial Peripheral Interface

SWAP	Simple Wireless Abstract Protocol
TCP	Transmission Control Protocol
Tx	Transmit
UART	Universal Asynchronous Receiver Transmitter
UDP	User Datagram Protocol
UI	User Interface
uIP	Micro Internet Protocol
URL	Uniform Recourse Locator
USB	Universal Serial Bus
Wi-Fi	Wireless Fidelity
WLAN	Wireless Local Area Network
WSN	Wireless Sensor Network
XML	Extensive Markup Language
ZMQ	Zero Message Queue
µC	Micro Controller

Chapter 1

Introduction

The expression Internet of Things, also referred to as IoT, was first put together by Kevin Ashton in a presentation in 1999 in order to characterize an Internet-based information architecture [21]. The term became popular and is widely used ever since, but coming up with an accurate and precise definition is not a simple task. IoT is the interconnection and Internet-connection of all the things that surround us and comprises the sciences of telecommunications and electronics. With any possible innovation in these fields, IoT grows even more powerful so its definition has to be flexible enough to follow. It can be described as the need emerging from all the entities of a developed environment to communicate with each other. Not only things but living organisms as well can be part of it. For this reason, Cisco has decided to coin the term Internet of Everything. Each element associated with the IoT has the ability to make decisions based not only on its own surroundings but on anything affecting its living, regardless distance or connectivity concerns. Connecting thousands of devices and giving them enough intelligence to act at will, is a way to get a glance at the scenario of an IoT.

The main objective of this thesis is to deploy an IoT testbed comprising the following elements:

1. A **wireless M2M area network** to connect sensors and actuators. In particular, sub-GHz transmissions have been chosen due to their suitability for the IoT, as it will be discussed later.
2. A **gateway** providing the wireless M2M area network with connectivity to the public Internet.
3. A **cloud platform** to store data gathered from the real world and properly formatted to be used by end-users facilitating the so-called smart applications.
4. An **application** on a smartphone (or tablet) to get access to the cloud and retrieve information gathered by the wireless network and interact with the actual sensors and actuators.

1.1 Related Work

There are several real-world experimentation platforms that have been developed over the last decade. Some representative examples are:

- The **Oulou testbed** is deployed in the city of Oulou of Northern Finland. It comprises a Wireless Local Area Network (WLAN) and a Wireless Sensor Network (WSN). The former is a public access network consisting of access points that use IEEE 802.11a/b to provide wireless broadband Internet access. The latter is a wireless sensor network that uses the IEEE 802.15.4 standard and implements the 6LoWPAN protocol stack in the 868MHz frequency band [48].
- The **IoT-LAB testbed** consists of devices deployed in six different locations in France. It comprises mobile and fixed wireless sensor devices that use the physical layer of the IEEE 802.15.4 standard and operate in the frequencies of 2,4GHZ or 800MHZ [47].
- The **W-iLab.t testbed** is deployed in city of Zwijnaarde in Belgium. It consists of mobile and fixed devices that communicate wirelessly using the IEEE 802.11a/b/g/n standards in the frequencies of 2.4GHz and 5GHz [49].

While all these testbeds have been developed for particular applications, the testbed created within the context of this thesis aims at being heterogeneous and flexible, so that it could be used for any final application in mind.

1.2 Thesis Overview

The rest of the thesis is organized in the following 5 chapters:

Chapter 2 includes general information about the IoT, a three-layer model suitable for understanding the concepts of this work and a detailed description of the IoT components. Additionally, a review of some relevant existing commercial IoT platforms is presented and a complete End-to-End architecture is proposed and analyzed.

Chapter 3 describes the implementation of the testbed deployed within this thesis. Moreover another approach to the IoT is discussed including Network Address Translation (NAT) configuration. It is implemented and conclusions are presented.

Chapter 4 demonstrates two Android applications designed to complete the testbed, Smart Parking and GeoFencing. The former works as a modern way to give a solution to the parking problem in big cities. The latter is a state-of-the-art application for remote automation in urban environments.

Chapter 5 describes the connection of the testbed to the electric power distribution system. It covers the current power grid, its need for improvement and how connecting it to the testbed, leads to the emerging smart grid.

Finally, **Chapter 6** concludes the thesis by providing overall conclusions and stating future lines for further research in the area.

Chapter 2

Research on the Internet of Things

This chapter presents an overview of the Internet of Things and describes its components based on a three-layer model. Some remarkable commercial tools are also mentioned and explained. Among them, we highlight the relevance of OpenRemote, an open source automation platform, and some IoT cloud platforms, virtual clouds that store information from thousands of devices. Finally, after introducing and explaining all the components, they are all combined to create an architecture to emulate an IoT environment.

2.1 Three-Layer IoT Model

Even though the concept of the Internet of Things has been under research for over a decade now, still many aspects are not clearly defined. For example, today there is no standardized and specific architecture for the IoT. Despite this lack of common agreement, there is a well-known three-layer architecture that is generally accepted which consists of the Perception Layer, the Network Layer and the Application Layer [24].

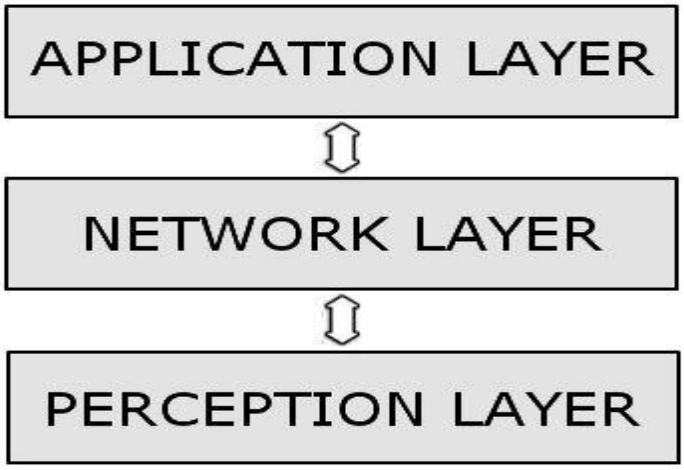

Figure 1: Three-layer IoT model [24]

2.1.1 The Perception Layer

The main task of the Perception Layer is to perceive physical properties such as temperature, location, speed, etc., by various sensing devices and convert this information into digital signals which can be easily transmitted through digital

communication networks and stored. The objects of this layer can have sensing abilities and/or actuating abilities. An actuator is a device which can receive programmed commands and perform tasks at specific times [24].

2.1.2 The Network Layer

The Network Layer is responsible for transmitting data received from the Perception Layer to a data base, server, or processing center. The main technologies used to realize this layer include cellular technologies 2G/3G/LTE, Wi-Fi, Bluetooth, or Zigbee, among others. Despite the wide variety of technologies for the radio access, IPv6 at the transport layer enables the interconnection of all of them besides the possibility to address the expected billions of things that will be connected in the near future [24].

The Internet of Things will be an immense network, which not only connects billions of things, but also encompasses heterogeneous networks.

2.1.3 The Application Layer

The Application Layer stores, processes, and analyses the information received from the Network Layer. These facilitate end-user applications such as building automation, location based services, identity authentication, safety etc. This layer provides applications for all kind of technological challenges. These applications promote the Internet of Things, which is why this layer plays an important role in the spread of the IoT [24].

2.2 Internet of Things Components

All the components required to build an IoT testbed are categorized below based on the three-layer model.

The Perception layer consists of:

- **Sensors**, sense physical properties and convert them into digital signals.
- **Actuators**, receive commands to perform actions at specific moments.
- **End-devices**, are small boards with an integrated microcontroller used to provide processing and communication abilities to sensors and actuators.

The Network layer comprises:

- **Communication protocols** used for end-devices.

- **M2M servers**, used to translate end-device information in routable form.
- **Gateways** to route traffic from end-devices to the Internet.
- **Operating Systems** for the end-devices.
- **Message Queues** are libraries implemented in servers and applications to provide a messaging system for publishing updates.
- **Application layer protocols** to connect applications to the End-Devices.

The Application layer contains:

- **IoT cloud platforms** are online virtual clouds that store information from end-devices and provide an interface with visualization of data (charts, graphs) for End-users.
- **Software Applications** for smart phones, tablets, Desktops to provide graphical user interfaces (GUI) for monitoring, processing and controlling end-device values.

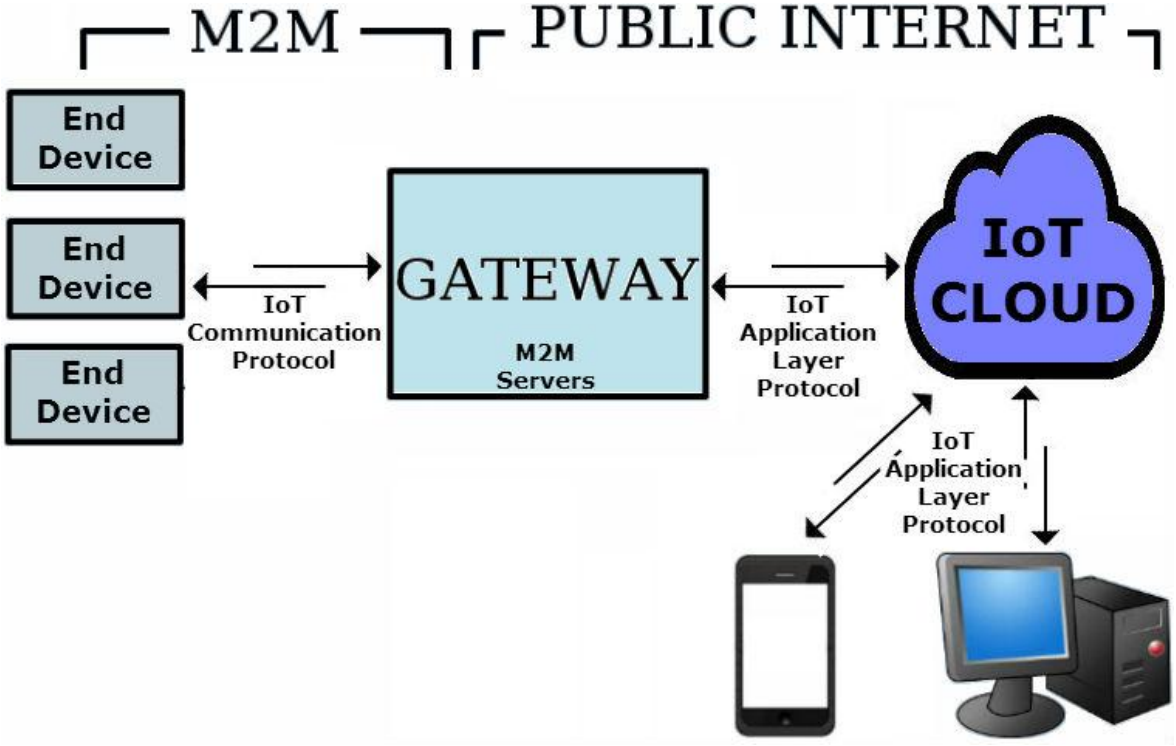

Figure 2: IoT components

Optionally, IoT cloud platforms work as nodes for applications to connect to and extract data, which is a function that belongs in the network layer. Nonetheless their main objective is to store and visualize data for End-users which is why they belong in the Application layer. Similarly, M2M servers might also offer an

application interface. However, this is not their purpose, thus they belong in the network layer.

2.2.1 End-Devices

Recent advances in micro-electro-mechanical systems technology, wireless communications, and digital electronics have enabled the development of low-cost, low-power, multi-functional end-devices which are small in size and communicate untethered in short distances [17]. These end-devices are typically equipped with sensors and/or actuators, microcontrollers (μC), and a radio transceiver which are explained as follows:

1. **Sensors** measure with high accuracy environmental conditions
2. **Actuators** execute an action depending on the power supply they are provided with.
3. **Microcontrollers** are very small computers that accommodate a processor unit and programmable input/output pins.
4. **Radio Transceivers** in order to transmit and receive data wirelessly. There are many alternatives for the RF transceiver and related communication protocols, and some of the most relevant alternatives are summarized in the next section.

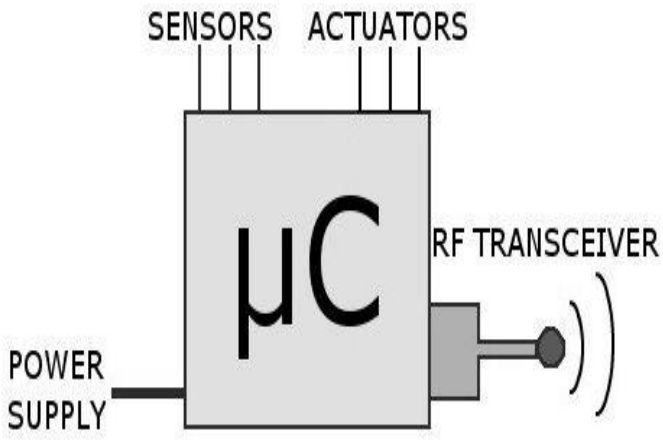

Figure 3: IoT end-device

2.2.2 Communication Protocols for the IoT

The end-devices can be stand-alone and independent able to connect to the Internet on their own or smaller low-power microcontroller boards with constrained communication capabilities. Small boards are preferred for being low-cost devices

able to work constantly with minimal power consumption. However they rely on a gateway to route their data to the Internet. The following protocols are used to connect constrained-communication end-devices to the gateway and are explained in detail in the next sections.

Wired connectivity:

- M-bus
- Modbus

Wireless connectivity:

- Zigbee
- 6LoWPAN
- Wi-Fi
- SWAP

Wireless end-device networks do not need high data-rates for transmissions since most devices do not transmit values continuously and spend most time in sleep mode for power consumption reasons. In addition, considering that sensors must be deployed in broad areas, range is very crucial. Taking everything into account, low frequencies and specifically sub-GHz transmissions are more suitable for wireless end-device networks due to the fact that they need less power and achieve longer ranges than higher frequencies. Low data rates of up to 600Kbit/s is the main disadvantage of sub-GHz frequencies but is considered enough for sensor networks.

2.2.2.1 Zigbee

The ZigBee alliance defines a specification used to create wireless low-range and low-power area networks in the 868/915MHz and 2.4GHz frequency bands. It is based on the IEEE 802.15.4 Standard, which defines the PHY and MAC layers. ZigBee devices can transmit data over long distances by exploiting multi-hop transmissions. The standard defines a maximum data rate of 250Kbit/s in the 2.4GHz frequency, which is suited for periodic or intermittent data or a single signal transmission from a sensor or input device. It is used in applications that require low data rate and long battery life [63]. Zigbee includes the Physical Layer and the Media Access Control (MAC) Layer defined by the IEEE 802.15.4

standard and it defines its own set of rules for the Network Layer and the Application Layer. It was created to target wireless automation and remote control applications. Some of its features are:

- **Collision avoidance**.
- **Timeslots managing.**
- **Integrated transmission security**.

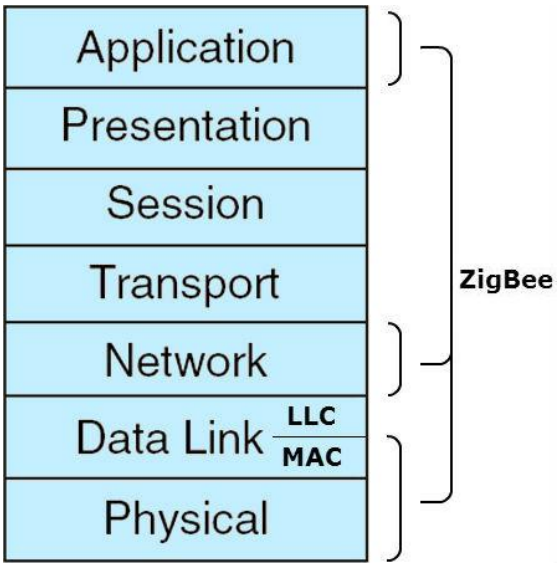

Figure 4: Zigbee representation on the OSI model

Zigbee building automation features include wireless range up to 70m indoors and 400m outdoors with full control of transmitted output power, enough to cover entire campuses. The networks are secured by the use of AES 128 encryption, keys, and device authentication [62].

The Zigbee Standards are the following [61]:

- Building Automation
- Remote Control
- Smart Energy
- Energy Profile
- Health Care
- Home Automation
- Input Device
- Light Link
- Retail Services

- Telecom Services

To get the Zigbee certification a device has to follow one of these standards and meet all the requirements.

ZigBee devices are of three types [63]:

1. **ZigBee Coordinator (ZC)**: The most capable device, it forms the root of the network tree and might bridge to other networks. There is exactly one ZigBee Coordinator in each network since it is the device that started the network originally.
2. **ZigBee Router (ZR)**: As well as running an application function, a Router can act as an intermediate router, passing on data from other devices.
3. **ZigBee End Device (ZED)**: Contains just enough functionality to talk to the parent node (either the Coordinator or a Router). It cannot relay data from other devices. This relationship allows the node to be asleep a significant amount of the time thereby giving long battery life. A ZED requires the least amount of memory, and therefore can be less expensive to manufacture than a ZR or ZC.

2.2.2.2 6LoWPAN

6LoWPAN is another communication protocol that includes the Physical and the MAC layer defined by IEEE 802.15.4 standard and operate in 868/915MHz and 2.4GHz frequency bands, same as Zigbee. It stands for IPv6 over Low Power Wireless Personal Area Networks which is the name of a concluded working group in the Internet area of the Internet Engineering Task Force (IETF). 6loWPAN adds an adaptation layer that works over IEEE 802.15.4 to allow IPv6 communication [2]. It has defined encapsulation and header compression mechanisms that allow IPv6 packets to be sent and received over IEEE 802.15.4 based networks. The base specification developed by the 6LoWPAN IETF group is called RFC 6282 [64].

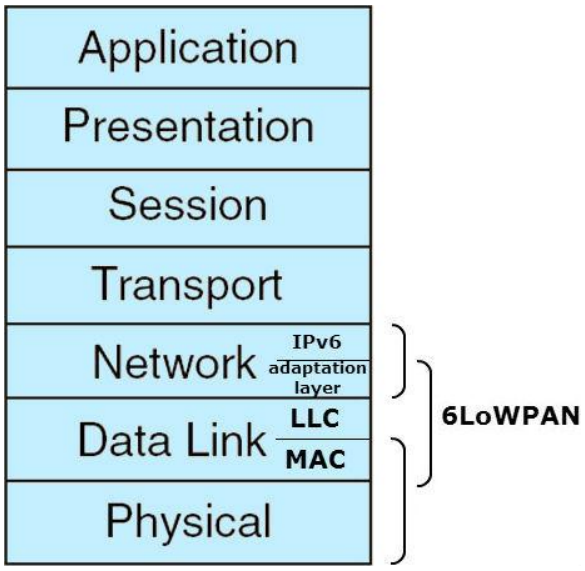

Figure 5: 6LowPAN representation on the OSI model

The 6LoWPAN concept originated from the idea that the Internet protocol should be applied even to the smallest devices [27] and that low power devices with limited processing capabilities should be able to participate in the Internet of Things [58]. The target of IP networking for low power radio communication is the applications that need wireless internet connectivity at lower data rates. Examples include automation and entertainment applications in home, office and factory environments. IPv6 is also in use on the smart grid, enabling smart meters and other devices to build networks before sending the data back to the billing system using the IPv6 backbone [64].

2.2.2.3 Wi-Fi

Wi-Fi is a local area wireless technology that operates in the 2.4GHz and 5GHz frequency bands. It is based on the IEEE 802.11 standards that define the Physical and MAC layers. Typical Wi-Fi devices using 802.11g have maximum data rate of 54Mbit/s and devices using 802.11n have data rate of up to 600Mbit/s. Many devices use Wi-Fi (e.g. personal computers, smart phones, tablets, digital cameras) to connect to a network such as the Internet via an access point. However Wi-Fi is not ideal for sensor networks due to the high frequency band that it uses.

2.2.2.4 SWAP

SWAP or Simple Wireless Abstract Protocol is a lightweight application layer protocol designed for constrained wireless M2M networks. Its characteristic is that

it is data type agnostic which simplifies a lot the implementation. SWAP is created by panStamp SLU which also created an open source M2M server called Lagarto that provides a user interface to monitor and control SWAP end-devices. The SWAP along with Lagarto are of grave importance to this thesis and will be discussed extensively in the next Chapter.

2.2.2.5 M-bus

The M-Bus or Meter-Bus is a European standard for remote reading of heat meters and it is also usable for all other types of consumption meters as well as for connecting various sensors and actuators. After its standardization it gained great importance for the energy industry because it can be used for low-cost real time metering.

Remote reading of meters can take place by collecting all the meter values for a complete housing unit, using just a single two wire cable which connects all the consumptions meters to a building controller. It is a protocol for wired connections which is very cost effective since it only uses one cable in which all meters are individually addressable. M-Bus includes Physical layer and Data Link layer. It could be used for the Internet of things in networks that require no wireless communication [22].

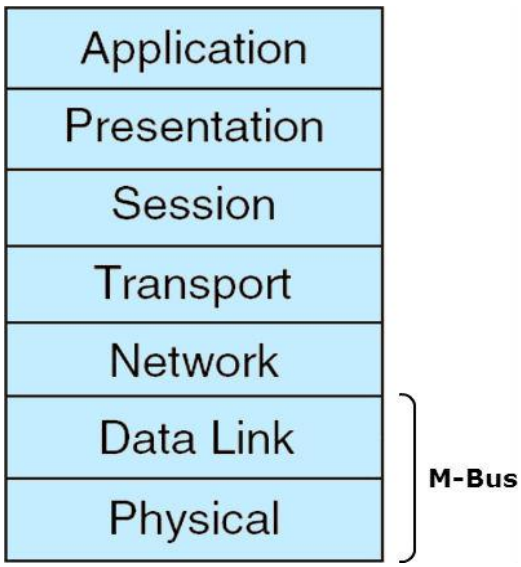

Figure 6: M-Bus representation on the OSI model

2.2.2.6 Modbus

The Modbus Protocol is a messaging structure developed by Modicon in 1979 for wired networks. It is used to establish master-slave/client-server communication between devices. Modbus is used in multiple master-slave applications to monitor and program devices and to connect them to various sensors.

A Modbus TCP/IP specification was developed in 1999. It can be implemented to any device that supports TCP/IP sockets. Modbus TCP/IP has become ubiquitous because of being low-cost and requiring minimum hardware to support it. It is used to exchange information between devices, monitor and program them. Modbus TCP/IP is an Internet protocol which means that a Modbus TCP/IP device can be addressed over the Internet from anywhere. Modbus TCP/IP includes the Physical, Data Link, Network, Transport and the Application layers [23].

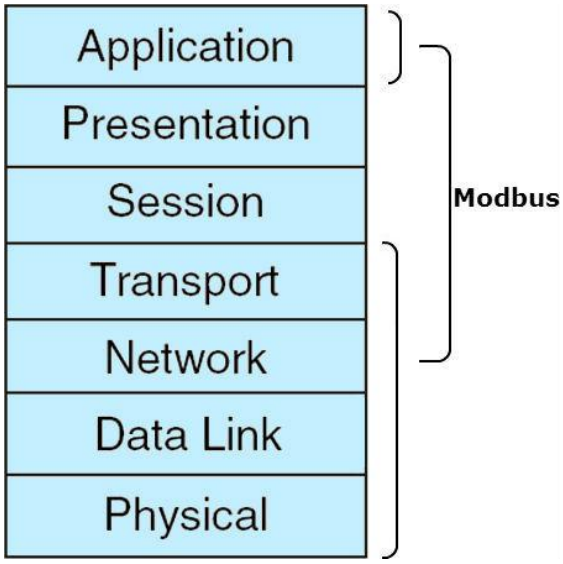

Figure 7: Modbus representation on the OSI model

Modbus is another communication protocol that could help in connecting wired networks to the Internet of Things.

2.2.3 Internet of Things Operating Systems

This section describes two operating systems specifically designed for the IoT: Contiki and TinyOS. Both of them are development tools that include many libraries and provide an IDE for writing operating systems for microcontrollers to be used by the end-devices. Contiki uses the C programming language and TinyOS uses nesC which is a variation of C.

2.2.3.1 Contiki

Contiki is a development platform for uploading operating systems to microcontrollers. It comprises Instant Contiki and the Cooja simulator. Instant Contiki is an open source Ubuntu-based OS that runs over a virtual machine and is used to write Operation Systems in the C programming language and upload them to the microcontrollers. Cooja is an application of Instant Contiki that uploads the OS to emulated end-devices for simulations. Contiki is designed for memory-constrained systems with particular focus on low-power wireless Internet of Things devices. It was created by Adam Dunkels in 2002 and has been further developed by a world-wide team of developers from Atmel, Cisco, Redwire, Oxford University and many others [11]. Contiki is designed to run on hardware devices that are severely constrained in terms of memory, processing power and communication bandwidth. A typical Contiki system only needs about 10 kilobytes of RAM and 30 kilobytes of ROM [12].

Contiki provides three network protocol stacks [10]:

1. **uIP TCP/IP stack** provides IPv4 networking
2. **uIPv6 stack** provides IPv6 networking
3. **Rime stack** is an alternative network stack that is used when the overhead of the IPv4 or IPv6 stacks is prohibitive.

The IPv6 stack was contributed by Cisco [7] and includes the 6LoWPAN header compression and adaptation layer for IEEE 802.15.4 links.

Many Contiki systems are severely power-constrained. Battery operated wireless sensors may need to provide years of unattended operation and with little means to recharge or replace its batteries. Contiki's default mechanism for attaining low power operation of the radio is called ContikiMAC [1]. ContikiMAC nodes can be running in low-power mode and still be able to receive and relay radio messages.

2.2.3.2 TinyOS

TinyOS is an open source development environment for embedded programming. It uses nesC which is a dialect of C to create abstract operating systems that can be used in many different supported microcontrollers [55]. It is designed for

wireless low-power resource-constrained end-devices, such as microcontrollers with a few kB of RAM and a few tens of kB of code space.

TinyOS has extensive networking support mostly because it's used by many low-power wireless research groups, who have then released their code for general use. In particular, TinyOS supports [54]:

- A complete 6LoWPAN/RPL **IPv6 stack**.
- **Multi-hop communications.**
- **Network-wide sub-millisecond time synchronization** through the Flooding Time Synchronization Protocol (FTSP).
- **Data collection** to a designated root or gateway through the Collection Tree Protocol (CTP).
- **Reliable data dissemination** to every node in a network through the Trickle algorithm.
- **Installing new OS over the wireless network** using Deluge.

2.2.4 Gateway for the M2M Area Network

A gateway is device that collects the measurements from all the end-devices and sends commands to the actuators. The gateway is responsible for connecting the M2M area network to other computer networks. Typically, this is the case of the Internet through an Ethernet, Wi-Fi or cellular connection.

End-devices are commonly small devices with constrained communication capabilities unable to produce UDP packets. This means that the gateway must host a server to translate the messages to routable packets. The server is monitoring the network constantly. It uploads end-device information to the virtual cloud and is responsible for receiving commands from the Internet and for translating them to the actuators. This server is called M2M server and is a key component to the system because it coordinates the end-devices network's interconnectivity.

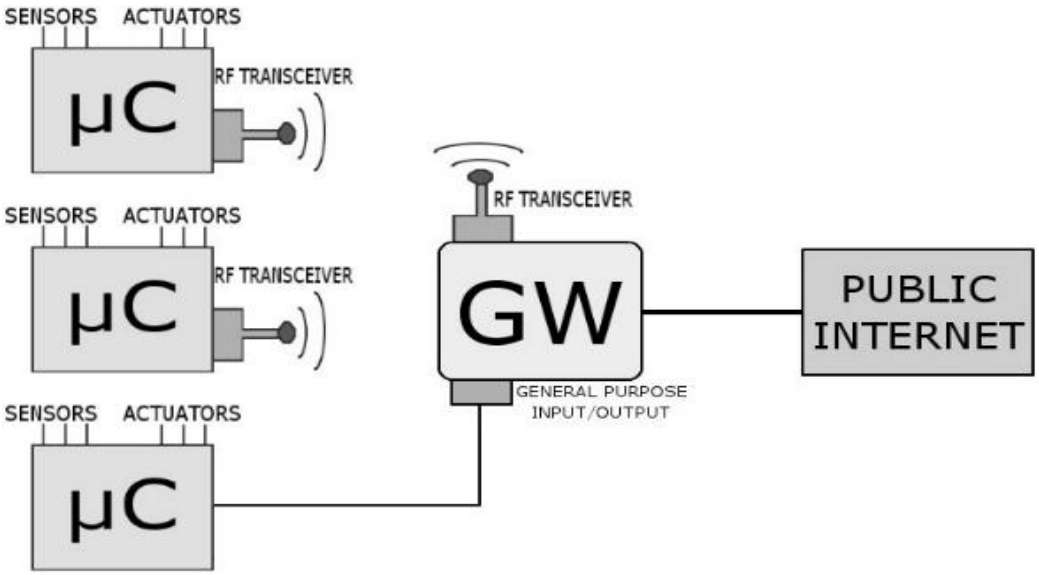

Figure 8: Internet connected end-device network

2.2.5 Message Queues

Message queues are libraries that can be implemented in the server's source code to provide a messaging system for end-device data. They are used when M2M servers need to communicate with each other and exchange information. Message queues release the servers from having to deal with delivery status, bottlenecks, compatibility among protocols, latency, retransmissions. They can provide a messaging system for servers on the same computer, on different computers inside the LAN or in distant computers through the Internet. There are many commercial message queues available such as:

- MQTT
- ZeroMQ
- RabbitMQ
- IronMQ
- ActiveMQ

Among them we highlight MQTT and ZeroMQ. ZeroMQ is not a message queue itself, but a library to build message queues programmatically, which makes it more flexible than other message queues. MQTT, which is also used by Facebook Messenger, is suited for end-devices because it uses bandwidth and batteries sparingly.

2.2.5.1 ZeroMQ

ZeroMQ also called ØMQ or ZMQ is an asynchronous messaging library aimed to be used in distributed applications [60]. It comprises a series of multiple queues that communicate with each other over TCP/IP. The queues can be on two different machines in a network, on two different processes in one machine, on two different threads in one process or any combination of the above. ZeroMQ is based on a publish/subscribe architecture. There are publishers that publish information and the subscribers that receive them. A subscriber can also be publisher to other subscribers and each publisher can be enrolled to receive information from other publishers [59].

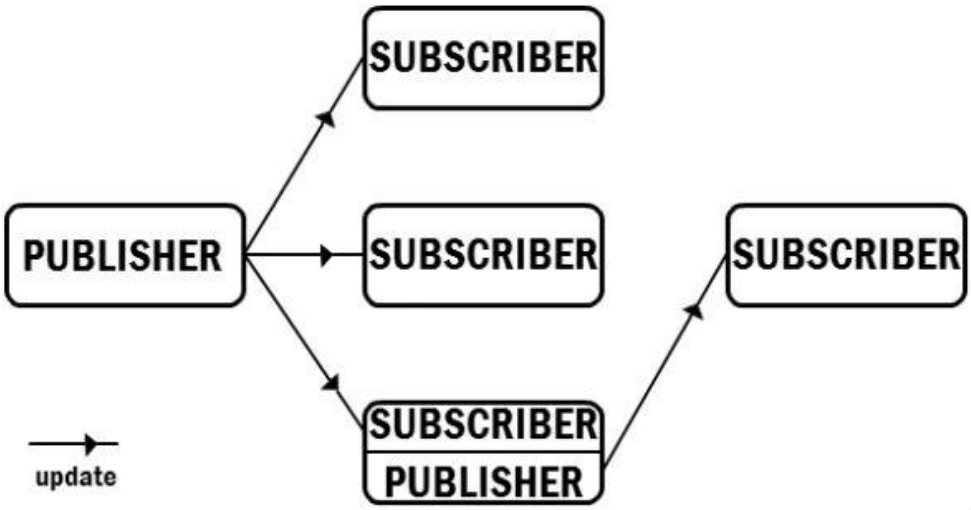

Figure 9: Publish-Subscribe concept

This architecture can be implemented to the IoT in order to connect M2M servers with each other. A main server can be subscriber to multiple M2M servers that monitor different end-device networks. Thus, it receives real-time updates and can monitor all the networks providing an overall picture of the end-devices. In addition the main server can be publisher, in order to send commands to the end-device actuators.

ZeroMQ is licensed under the LGPL. This license gives the explicit right to link ZeroMQ with closed-source applications, as well as open source applications.

2.2.5.2 MQTT

MQTT or Message Queue Telemetry Transport was invented by Dr. Andy Stanford-Clark of IBM, and Arlen Nipper of Arcom in 1999. It is a lightweight

publish/subscribe messaging protocol, designed for constrained devices. The design principles are to minimize network bandwidth and device resource requirements whilst also attempting to ensure reliability and some degree of assurance of delivery. These principles also make the protocol ideal for the IoT and for mobile applications where bandwidth and battery power are at a premium [26].

The MQTT protocol is based on the principle of publishing messages and subscribing to topics. Clients subscribe to topics that they are interested in, but they also publish messages to topics. Many clients may subscribe to the same topics and do with the information as they please. MQTT acts as a common interface for everything to connect to over TCP/IP. This means that the clients can add new sensor data to a topic and other clients can store this information to a database [25].

All messages may be set to be retained. This means that the message will be kept even after sending it to all current subscribers. If a new subscription is made that matches the topic of the retained message, then the message will be sent to the client. If a topic is only updated infrequently, then without a retained message, a newly subscribed client may have to wait a long time to receive an update. With a retained message, the client will receive an instant update [25].

There are three qualities of service for message delivery [18]:

1. **At most once** where messages are delivered according to the best efforts of the underlying TCP/IP network. Message loss or duplication can occur. This level could be used, for example, with ambient sensor data where it does not matter if an individual reading is lost as the next one will be published soon after.
2. **At least once** where messages are assured to arrive but duplicates may occur.
3. **Exactly once** where message are assured to arrive exactly once. This level could be used, for example, with billing systems where duplicate or lost messages could lead to incorrect charges being applied.

Encryption across the network can be handled with SSL, independently of the MQTT protocol itself [26].

2.2.6 Application Layer Protocols for the IoT

The gateway is responsible for maintaining direct bidirectional communication with all the end-devices in the M2M area network and to provide them with Internet connectivity. The application layer protocols are implemented in the gateway to provide client–server based Internet communication, where the gateway is the client an IoT cloud platform is the server. They are also implemented in the applications that need to connect to the cloud platform to extract information. In this case, an application is the client and the cloud is the server. The cloud must also support the specific application layer protocol in order to communicate with the clients.

When the client wants to access the server, it creates a request and sends it to the port 80 of the server, which is by default the port for client-server communication in TCP. The server then sends a response to the client using the source port of the request which is called ephemeral port. Ephemeral ports are temporary ports dynamically created, and they are only used for responses. All the client ports are by default firewall protected for security reasons.

Regarding the gateway, it can send all the end-device information to the Internet using requests, but if it is firewall protected which is by default the case, it cannot accept requests for commands. The following protocols address the way end-devices connected to a typical firewall protected gateway, are monitored and controlled through the Internet.

- **REST** services use HTTP commands as a client-server update system.
- **WebSocket** use an HTTP handshake to establish sessions for client-server full duplex communication.
- **CoAP** is a light-weight application layer for constrained-communication devices that runs over UDP.

2.2.6.1 REST Web Services

REST web services use HTTP methods explicitly in a client-server architecture where the client requests and the server responds. The basic REST design

principle establishes a one-to-one mapping between create, read, update, and delete (CRUD) operations and HTTP methods. According to this mapping there are four basic commands [50]:

1. **POST**: To create a resource on the server.
2. **GET**: To retrieve a resource.
3. **PUT**: To change the state of a resource or to update it.
4. **DELETE**: To remove or delete a resource.

REST web services allow formatting the data that the server and client exchange in the request/response payload. To give client applications the ability to request a specific content type that is best suited for them, the services make use of the built-in HTTP Accept header, where the value of the header indicates the type of data that it contains. Some common format types used by REST services are JSON and XML which allows the services to be used by a variety of clients written in different languages. Using the HTTP Accept header is a mechanism known as content negotiation, which lets clients choose which data format is right for them and minimizes data coupling between the service and the applications that use it [19].

REST was first introduced in 2000 by Roy Fielding at the University of California, Irvine, in his academic dissertation, "Architectural Styles and the Design of Network based Software Architectures", where he analyzed a set of software architecture principles that use the Web as a platform for distributed computing [14]. Measured by the number of Web services that use it, REST has emerged in the last few years as the predominant Web service design mode [19].

The REST commands are widely used to exchange information with virtual clouds. The M2M server used by the gateway to translate incoming and outgoing traffic can be configured to send frequent HTTP POST requests to the cloud in order to store the sensor measurements to an Internet platform. Then, applications use HTTP GET requests and retrieve the values from the online platform in order to process, save or display them to the end-user. Most virtual clouds use the JSON format for exchanging information through the HTTP requests.

2.2.6.1.1 HTTP Polling

HTTP Polling uses the HTTP methods defined by the REST services to allow the server to send information to the client in a typical firewall protected system. Normally the server would make use of the HTTP POST method to send data to the client but that is not possible since the client is protected with a firewall that only allows responses. All the HTTP requests to the client are ignored from the firewall and never reach their destination. That is why the polling mechanism uses the response to help the data pass the firewall. Instead of making an HTTP POST request from the server to the client, the polling mechanism makes an HTTP GET request from the client to the server. Then the server encapsulates the information that need to be sent to the client, inside the HTTP GET response that can pass the firewall security.

The M2M server, hosted in the gateway, can send the sensor data to the Internet by making HTTP POST requests. But when it comes to receiving commands from the Internet, the gateway acts as the client and its incoming traffic gets filtered by the firewall. By using the polling mechanism the gateway makes frequent HTTP GET requests that check if there are pending updates on the server. Depending on the frequency of the GET requests there can be delays from the time that the update is ready until the time that the gateway will send a GET request to check for updates.

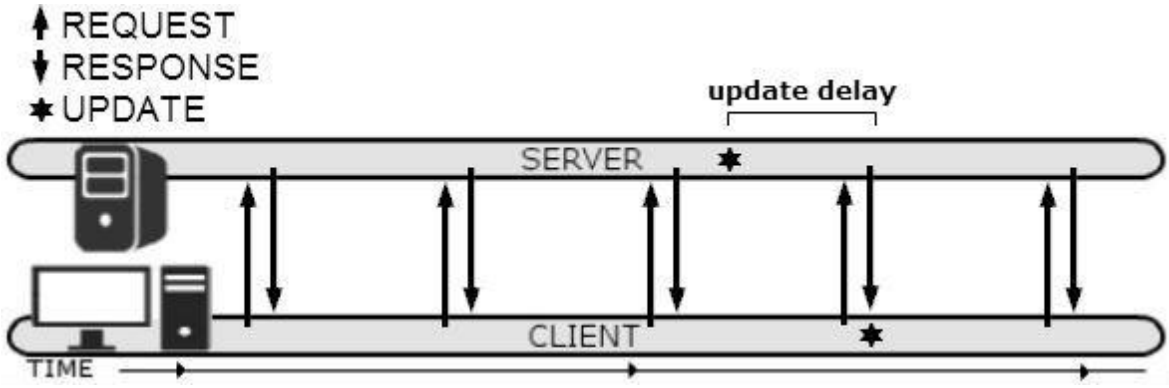

Figure 10: Polling mechanism

2.2.6.1.2 HTTP Long Polling

Long Polling is a variation of the traditional polling. It is addressing the same problem as the regular polling and its principals are exactly the same but the HTTP GET request itself, is different.

The gateway needs to get a command from the Internet and update the sensor network. For the same reasons as in polling, the gateway initializes an HTTP GET request to the online server. The difference in long polling is that the request stays pending over time and there is no response unless there is an update for the end-device network (e.g. a command for an actuator), in which case the HTTP response is sent back carrying the update to the gateway. If a request is initialized but there are no updates, it expires and then another request has to take its place. Long Polling is a mechanism that provides real-time communication between the client and the server and therefore, real-time updates to the end-device network.

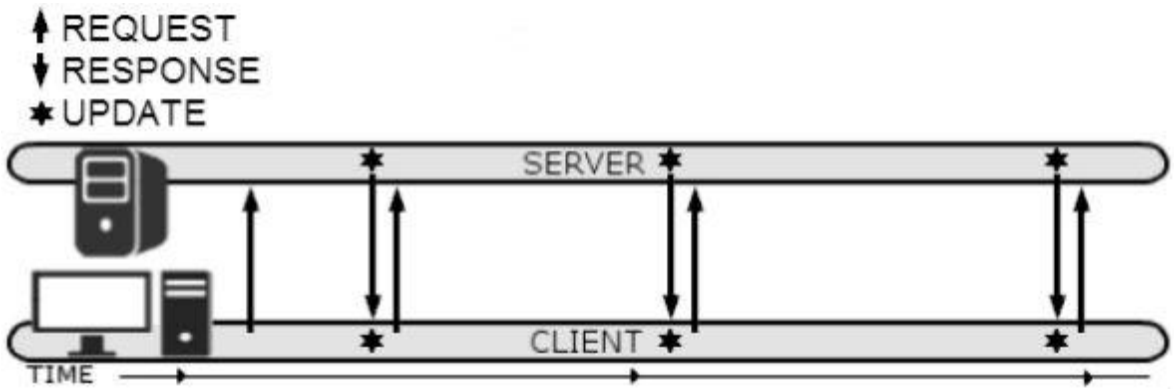

Figure 11: Long-polling

2.2.6.2 WebSocket

WebSocket is a new protocol standardized in 2011 to facilitate various communication channels over a single TCP connection. It relies on sessions to provide full duplex communication. To establish a WebSocket session, the client sends a WebSocket handshake request over HTTP, for which the server returns a WebSocket handshake response. After that, messages are initialized and sent from each side at any time as long as the session remains open. The handshake is over HTTP so that servers can handle HTTP connections as well as WebSocket sessions on the same port. However, what follows after the handshake, do not conform to the HTTP protocol [57].

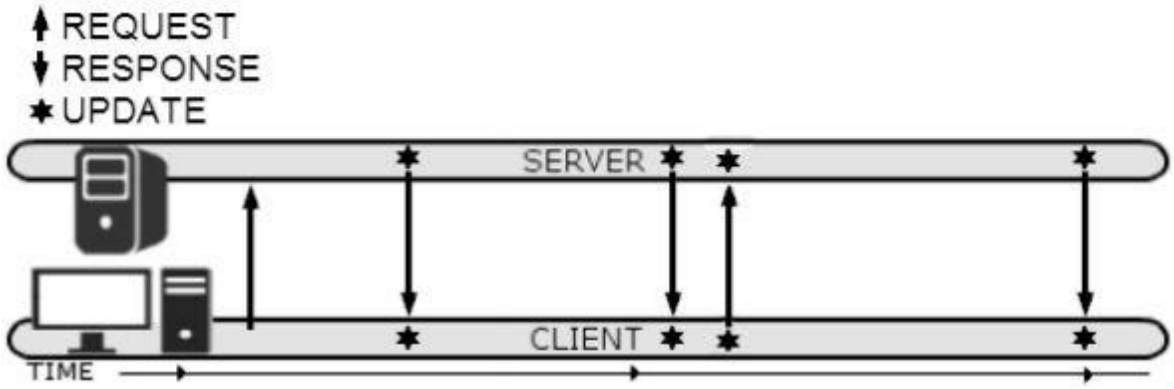

Figure 12: WebSocket session

Implementing WebSocket on the gateway and also on the server-side to an online IoT cloud is the fastest way to monitor and control a sensor network from anywhere in the world. It can be used both for sending end-device data to the cloud and for receiving commands. WebSocket is designed to target demanding real-time applications like online gaming, but unfortunately it is not yet very popular to IoT and M2M applications.

2.2.6.3 CoAP

The Constrained Application Protocol or CoAP is an application layer protocol that is intended for use in resource-constrained internet devices, such as Wireless Sensor Network nodes unable to implement HTTP libraries, thus unable to use REST or WebSocket. It uses the GET, POST, PUT and DELETE primitives to provide resource-oriented interactions. CoAP runs over UDP to keep the overall stack implementation lightweight, in which case it defines a procedure to guarantee reliability of transmission, which is very often a requirement of M2M applications [6]. It is a protocol specifically designed for devices with constrained communications and low computational power that need to connect to the IoT [9].

CoAP is one of the two application protocols currently supported by the ETSI M2M together with the wide spread HTTP.

Contrary to REST and WebSocket, CoAP can be used by independent devices with low computation and communication capabilities that do not rely on the gateway to translate their messages. They can implement CoAP which acts as a UDP library and they can use it to produce routable UDP packets.

2.2.7 IoT Cloud Platforms

A key element in the IoT is the concept of the cloud platform. IoT Cloud Platforms are online virtual clouds that can receive information from any Internet connected device. IoT clouds can be used for monitoring the conditions of a device at any moment. They offer storage for the device measurements and easy access to the values with useful charts and graphs. There are already various commercial virtual clouds in the market including ThingSpeak, OpenSense, Axeda, ThingWorx, Xively, among others. Some of them are listed and outlined below.

- **ThingSpeak** is the only virtual cloud which is open source and its source code is available for download.
- **OpenSense**, which is still a beta version, is a very promising cloud with an intuitive user interface.
- **Axeda** that in addition to web services, it allows messages queue integration.

2.2.7.1 ThingSpeak.com

ThingSpeak is an open-source Internet of Things online application that processes HTTP requests for storing, retrieving and processing information from end-devices. The ThingSpeak interface is Channel-based. Each ThingSpeak Channel supports data entries of up to 8 data fields, plus three standard fields, latitude, longitude, and elevation. Channels support JSON, XML, and CSV formats. In addition to storing and retrieving numeric and alphanumeric data, ThingSpeak allows numeric data processing such as time scaling, averaging, median, summing, and rounding [52].

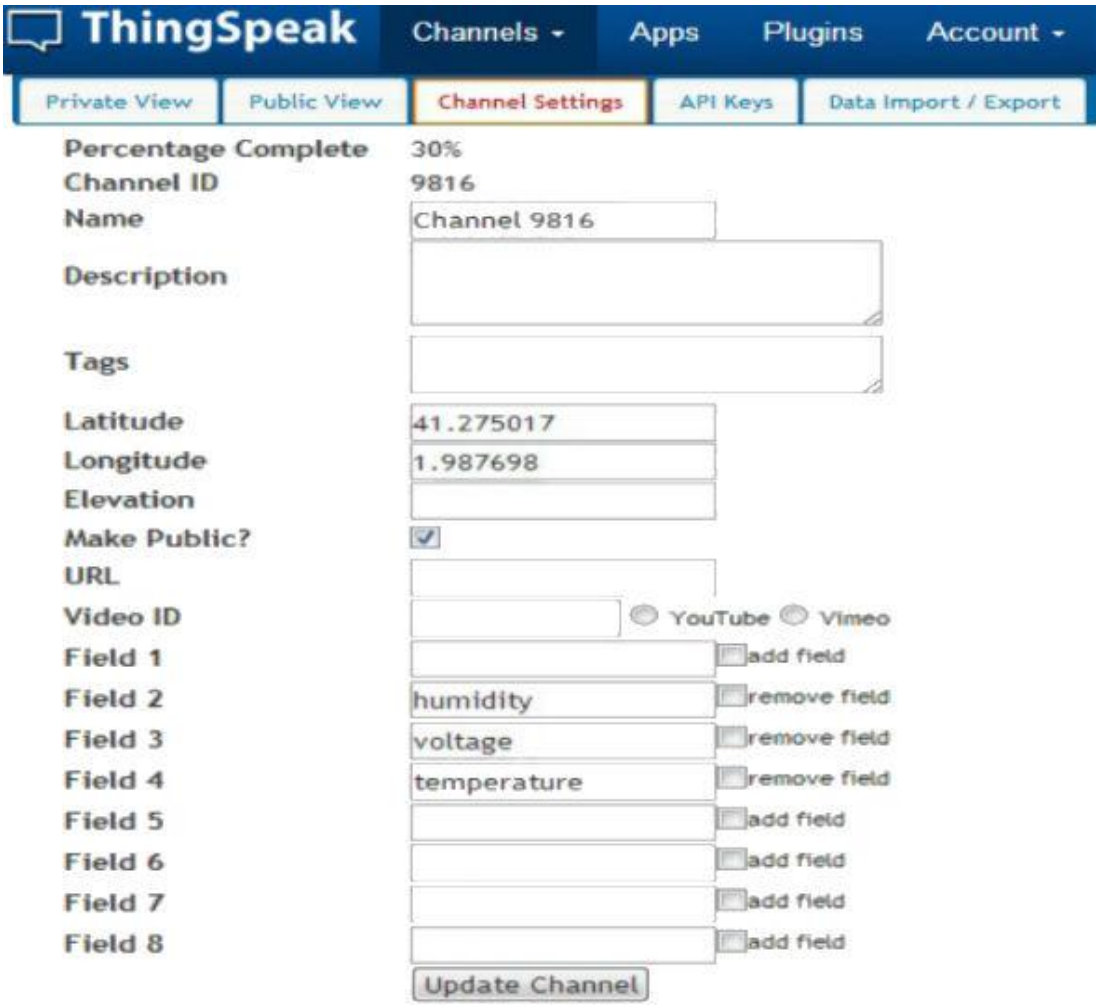

Figure 13: ThingSpeak channel [52]

ThingSpeak's source code is available on GitHub [53], which is an online repository for open source projects, and includes the complete ThingSpeak server written in the Ruby programming language. The only difference is that the online cloud has an update limitation. After each update you have to wait at least 15 seconds before sending the next one, otherwise it is ignored. The rule is not integrated in the available for download source code.

ThingSpeak Features:

- **Geo-location data** and display of end-device on maps.
- **Data processing** like averaging, median, summing, and rounding.
- **Data visualizations** with charts and graphs
- **Plugins** that allow custom encapsulation of HTML, JavaScript and CSS scripts.
- **Social Network** integration with Twitter.

2.2.7.2 Open.Sen.se

Open.Sen.se is an online IoT platform that receives and processes data through HTTP requests, and stores its context so that it can be retrieved at any time. The Open.Sen.se interface is based on a three-option menu which includes Senseboard, Applications and Channels. The Senseboard tab is the main way to visualize the received data. It is a customizable board where graphs, meters and switches can be added for monitoring the end-devices and their surroundings. The Applications tab includes various processes that can be added to the Senseboard for displaying data, send notifications or add computing functions (average, sum, counter etc.). Finally, the Channels tab is where new devices, real or virtual, can be created and configured [31].

Open.Sen.se is used for testing new devices and applications in order to create a globally interconnected and immersive world. It is at the time of writing of this thesis new, and the online virtual cloud is still a beta version. For this reason, it is not open for registration. People eager to start using the services have to fill out an invitation request form.

OpenSense Features:

- **Real-time data collection**
- **Data processing** counters, sum, average
- **Data visualizations** with graphs
- **Virtual switches** that represent end-device actuators.
- **Email notifications**
- **Social Network integration** with Twitter and Facebook.

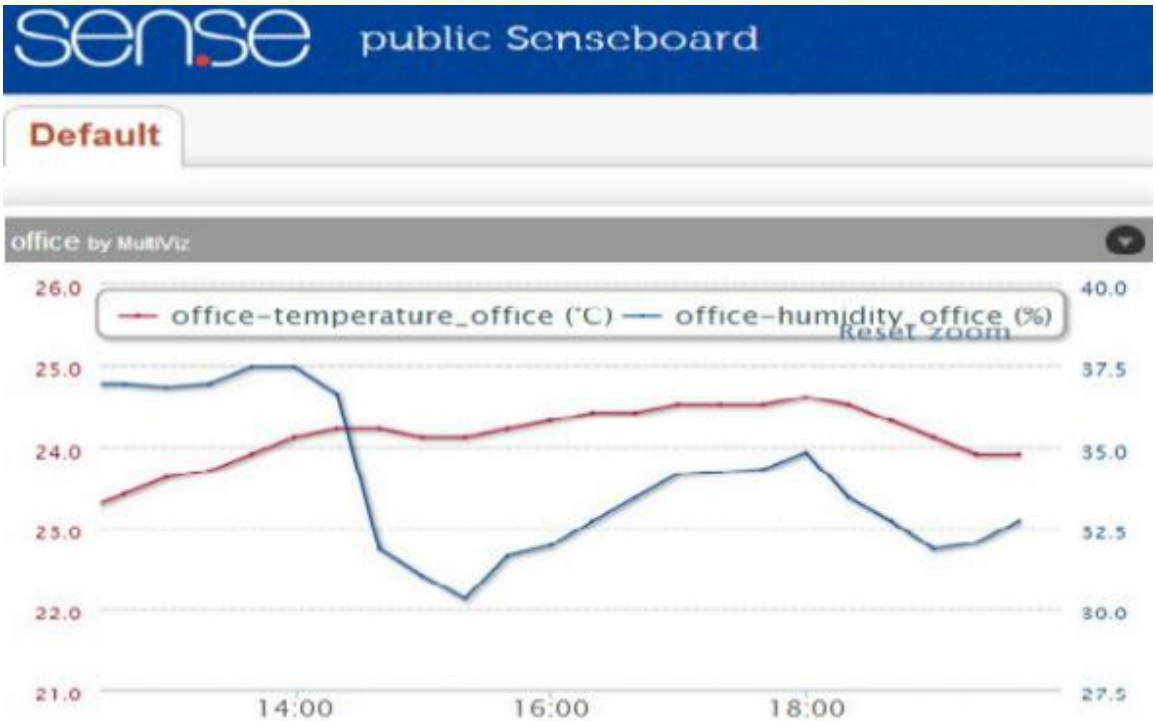

Figure 14: OpenSense public Senseboard [31]

2.3.3 Axeda.com

Axeda provides a cloud-based service and software for managing connected products and machines and implementing Machine-to-Machine and Internet of Things applications. The Axeda Machine Cloud can be used to turn machine data into valuable information, to build and run M2M and IoT applications and to optimize business processes by integrating machine data.

The Axeda Platform includes [5]:

- **IoT Application Services** allow developers to extend and customize the core platform via the embedded scripting engine that uses the Groovy programming language and via the REST web services.
- **IoT Integration Framework** accelerates integration with the Axeda Platform with a standard-based message queue technology called ActiveMQ.
- **IoT Data Management** processes and stores incoming M2M and IoT data, manages device types, data items, locations, and files and includes built-in security for managing users, user groups, and device groups.

2.3 The OpenRemote Platform

OpenRemote is a platform for residential and commercial building automation. It is not an online cloud-based service like ThingSpeak and Axeda, but an open source M2M server that runs on the gateway and connects to the end-device network. In addition it includes a mobile application that needs to be inside the gateway's LAN in order to monitor and control the network. The user interface does not offer storing and processing of the data, it is purely an application for end-users to monitor and control the end-devices in real time. The application is available for iOS and Android devices, and for devices with modern web browsers. The user interface design and configuration can be handled with OpenRemote online design tools.

OpenRemote consists of the following three main components [29]:

1. **OpenRemote Controller** is a server that connects to the sensor network and manages runtime integration between your devices.
2. **OpenRemote Designer** is an online tool that helps the end-user design the application's layout and deploys on the Controller all the information it needs to connect to the sensor network.
3. **OpenRemote Control Panel** is the layout of the application, created from the Designer and allow the user to monitor the end-devices and control them in parallel.

Figure 15: OpenRemote Designer [29]

The software is distributed under AGPL. The source code is currently hosted on Source Forge [30] which is an online repository for open source projects.

2.3.1 OpenRemote Controller

The Controller relays the commands from the panels to the target protocols and acts as a translator between devices and the OpenRemote application. It is a server that runs non-stop and connects to a mobile application through which users can achieve real-time automation. It handles the overall runtime performance after initial design has been created on the Designer.

The Controller receives commands from iPhone/Android/Web panels and routes the commands to the appropriate devices. It provides the control panels with user interface definition files from the OpenRemote online Designer. When multiple wall panels, tablets and phones are used at the same time, OpenRemote Controller can be used to maintain device status to keep all user control devices in sync with current state of devices.

2.3.2 OpenRemote Designer

The OpenRemote Designer is an online application to rapidly and easily create touch-driven control panels. It is device-independent and can support Android

phones and tablets, Apple iPhone, iPod Touch and iPad and also create designs for stand-alone web browsers.

OpenRemote Designer can be used remotely from any web browser without additional install. The OpenRemote Controller can automatically sync with designs created and stored online. Users can make small tweaks and changes to the user interface layouts as the preferences and needs evolve over time. Multiple profile interfaces can be created, where the degree of control and technical information exposed in the UI can vary per user.

2.3.3 OpenRemote Control Panels

Panels are a client of the OpenRemote Controller. This client is a native iPhone/iPod Touch application, Android application or a web application. The user interface and layout can be edited from the OpenRemote Designer.

So the panel renders user interfaces defined in OpenRemote Designer. They can display various user interface elements such as buttons, labels, images and sliders and render status updates from devices.

2.4 Suggested Architecture for the Testbed

A testbed for the IoT has to be an open system able to host many different technologies. It also has to be flexible so that all the hosted technologies can be modified to work together in order to provide a functional development platform. Many emerging IoT technologies are analyzed in this thesis. An IoT testbed should be able to implement and test most of these technologies in order to examine and evaluate multiple IoT concepts. All the implementations have to be accomplished in such way, that other stacks can be easily implemented on top, without further effort.

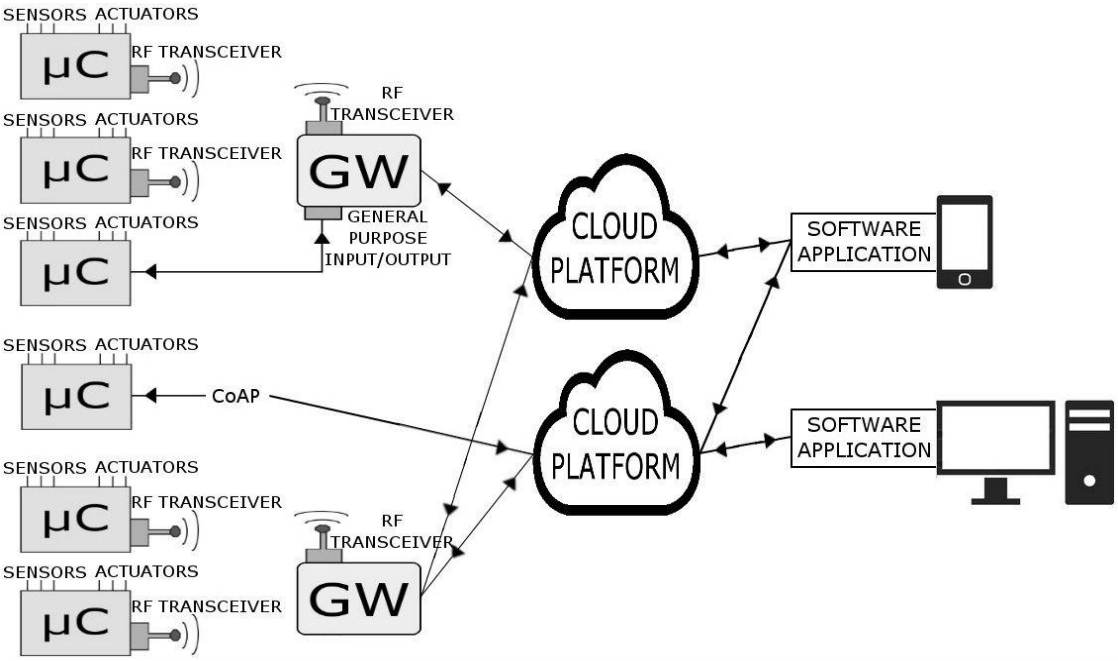

Figure 16: Suggested testbed architecture

Figure 17 demonstrates a complete End-to-End IoT architecture. End-devices connected to sensors and actuators are connected to gateways through wired or wireless means. The gateways host M2M servers that run non-stop and translate data to/from the end-devices and connect the sensor networks to the online virtual clouds. Independent recourse-constrained devices can produce routable packets using CoAP and connect to the clouds using 2G/3G/LTE modules without the need of an extra gateway. The IoT clouds collect data from various devices and display them in a homogeneous way for end-users. Software applications running on smart phones, tablets, laptops can also connect to the IoT clouds, extract information and add their own computational power to process and combine the data and control the end-devices.

Chapter 3

An IoT Testbed

This chapter describes the practical implementation of the IoT testbed deployed in this thesis. To build the testbed we used the following devices and technologies:

1. The **Perception layer** includes the following hardware devices:
   - **PanStamps** are small wireless boards with an integrated microcontroller that can connect to sensors and actuators.
   - **Base boards** are small boards with a battery case and integrated sensors and actuators.
2. The **Network layer** includes:
   - **Raspberry Pi** is a small computer. A hardware device that is the gateway for the end-device networks and is also used to host the M2M servers.
   - **Lagarto SWAP** is an M2M server. A software process that monitors and controls end-devices.
   - **OpenRemote Controller** is another M2M server that monitors and controls end-devices but also connects to the OpenRemote out-of-the-box Android application.
   - **Lagarto MAX** is a server for the overall system. A software process that uses a messaging queue to connect to other M2M servers and present diverse end-device data in a homogeneous way.
3. The **Application layer** includes the following software components:
   - **OpenRemote application** is a software application for iOS and Android devices that provides a monitor/control interface.
   - **ThingSpeak** is an online IoT cloud that is connected to the gateway for being the only open source cloud.
   - **OpenSense** is another online virtual cloud that is connected to the gateway due to its simple and intuitive interface.

Each of these components is described in detail in the next sections.

3.1 The PanStamp Project

The panStamp project is designed for automation using small low-power wireless motes transmitting in the sub-GHz frequency band and achieving ranges of more than 200 meters. It includes the following components which are explained in detail in the next sections:

- **PanStamps** are PCB boards with integrated microcontrollers and wireless communication capabilities at 868/915MHz frequency band.
- **SWAP** which is an application layer protocol for the wireless devices.
- **Lagarto** an M2M server for monitoring and controlling.
- **Base boards** are PCB boards with a battery case and integrated sensors and actuators. They are designed to connect to end-devices in order to provide them with power supply and uncomplicated connection to sensors.

The PanStamp project is aimed at creating ICT solutions for measuring and controlling things wirelessly. The central element of the project is the PanStamp, which is a small wireless board designed to fit in low-power applications. PanStamps are suitable for any kind of automation including energy metering, weather monitoring, home automation and robot control [33].

The project is kept open-source and all parts of the code, the protocol stack and the hardware designs are released and available for download from the online repository [36].

3.1.1 PanStamps

PanStamps are small wireless boards designed for monitoring and controlling things wirelessly which contains:

1. An Atmega328p microprocessor, which makes them reprogrammable and configurable.
2. An integrated CC1101 IC RF transceiver from Texas Instruments which provides communication ranges of more than 200 meters in open spaces with transmissions at sub-GHz frequency bands [37].

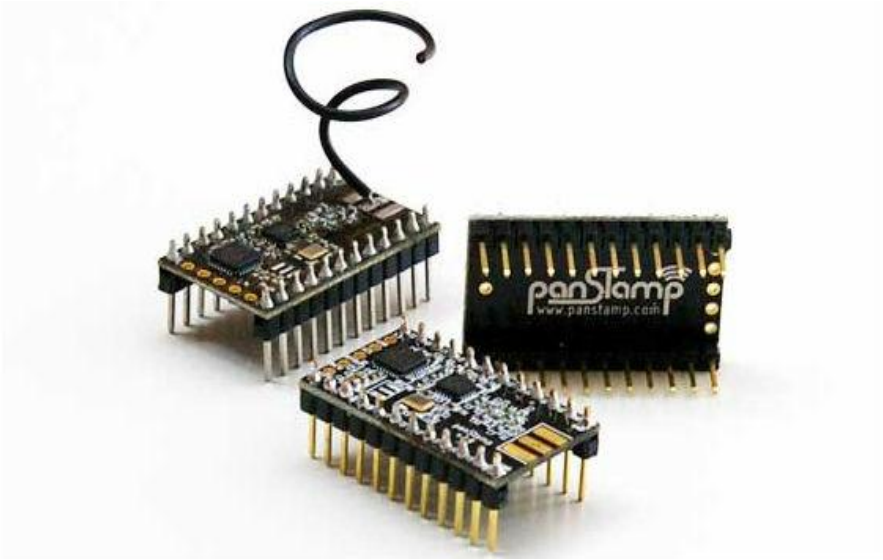

Figure 17: PanStamp modules [37]

Product Specifications [41]:

- Dimensions: 0.7 x 1.2 in (17.7 x 30.5 mm)
- MCU: Atmega328p
- Radio chip: CC1101 (Texas Instruments)
- Bus Speed: 8MHz
- Flash Memory: 32KB
- RAM: 2KB
- EEPROM: 1KB
- Voltage range: from 2.5VDC to 3.6VDC
- Rx current: 24 mA max
- Tx current: 36 mA max
- Sleep current: 1 uA
- Maximum Tx power: +12 dBm
- RF bands: 868/905/915/918 MHz ISM bands
- Modulation: GFSK
- Data rate: 38 Kbit/s
- Communication length: 200m in open spaces

PanStamps use the CRC (Cyclic Redundancy Check) mechanism to detect errors in messages and use CCA (Clear Channel Assessment) in order to switch from Receive mode to Transmit mode to avoid collisions. Both CRC and CCA are

integrated in the TI CC1101 RF transceiver. The panStamp itself, does not support collision detection. If two panStamps begin transmitting at the same, there will be a collision. However, they do have a retransmission mechanism if CCA detects the carrier busy, in which case the panStamp will try again three times delayed by the number of the panStamp's ID multiplied by 2 in milliseconds. This guarantees that if two panStamps sense the carrier simultaneously and both detect it busy, they will not try again at the same time.

PanStamps are programmed through the Arduino IDE. The Arduino IDE is an open source environment for programming microcontroller boards based on processing, another open source programming language built to teach computer programming fundamentals [43]. The Arduino IDE programming language is merely a set of C and C++ functions that are called from the code. The Software scripts written using the Arduino IDE are called sketches. Sketches are written in the text editor and are saved with the file extension .ino [4]. Before uploading the sketch form the IDE to the panStamp, the correct board has to be configured. Tools **>** Board **>** Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega328 is the correct selection for programming panStamps. The IDE also offers a serial monitor for direct bidirectional serial communication with the board.

3.1.2 Base Boards

Base Boards are PCB boards that can accommodate panStamps and easily connect them to sensors, actuators and power supply. Base boards take the power from a single AA battery and can host multiple sensors, including temperature, humidity and pressure. Using the proper programming techniques, this board can run for months from a single battery, thus being suitable for building automation and environmental monitoring applications [32].

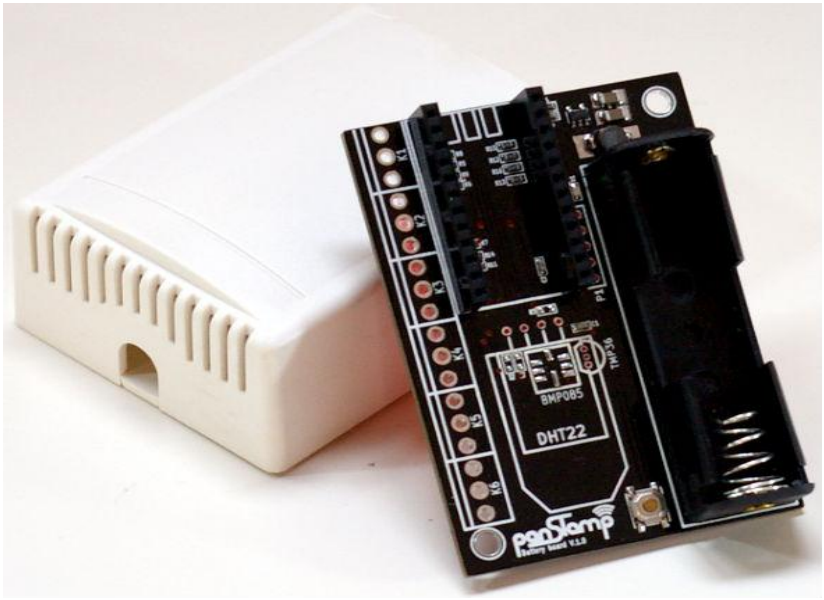

Figure 18: PanStamp base board [32]

Base boards generally remain simple since most of the electronics are contained in panStamps. The core of the board is a step-up voltage regulator (MAX1724) which transforms 0.8 - 1.5 volts supplied by the AA battery into 3.3V, a voltage accepted by most sensors nowadays [32]. They are perfect for prototyping since most pins are available so that users can solder external sensors or mount 3.5 mm terminals for further comfort. Base boards can be purchased as simple battery boards or with integrated sensors.

3.1.3 PanStick

PanStick is a USB mother board for panStamps. It is used to provide a USB connection to the panStamp in order to plug it in a computer and use the Arduino IDE to program it. It also acts as a serial gateway to the wireless network by placing a panStamp on the panStick, and then to the computer. The panStick, being attached to a functional panStamp, can also be connected to sensors and actuators and send data directly to the USB port [34].

Figure 19: PanStick [34]

The panStick includes an on-board voltage regulator which takes the power directly from the USB bus. Thanks to this regulator, panStick is able to power sensors up to 250 mA at 3.3 VDC [34]. The USB board also includes a reset button that allows instant restart without having to unplug the board.

3.1.4 Lagarto Servers

Lagarto is an open source software server written in the Python programming language. It is designed to be used as an M2M server that runs on the gateway to translate end-point messages to routable form and automate physical tasks by controlling actuators. Lagarto servers provides an HTTP GET/POST interface used to control values from clients or any other application with HTTP client capabilities (Web browser, javascript, Flash). They use a common ZeroMQ Publishing socket over TCP to notify events to clients so that they do not have to continuously request updates. This dual communication mechanism (ZeroMQ + HTTP) results in a functional solution for automation [39]. There are two types of Lagarto servers, both explained in detail in the next sections:

1. **Lagarto SWAP** is a software M2M server designed to monitor and control end-device networks.
2. **Lagarto MAX** is a software server designed to communicate with multiple M2M servers over TCP using the ZeroMQ message queue.

Lagarto processes provide a web interface for configuration and basic monitoring purposes. Every process binds to a different port number, allowing the existence of multiple servers on the same computer. The HTTP port number is set manually from an XML file [39].

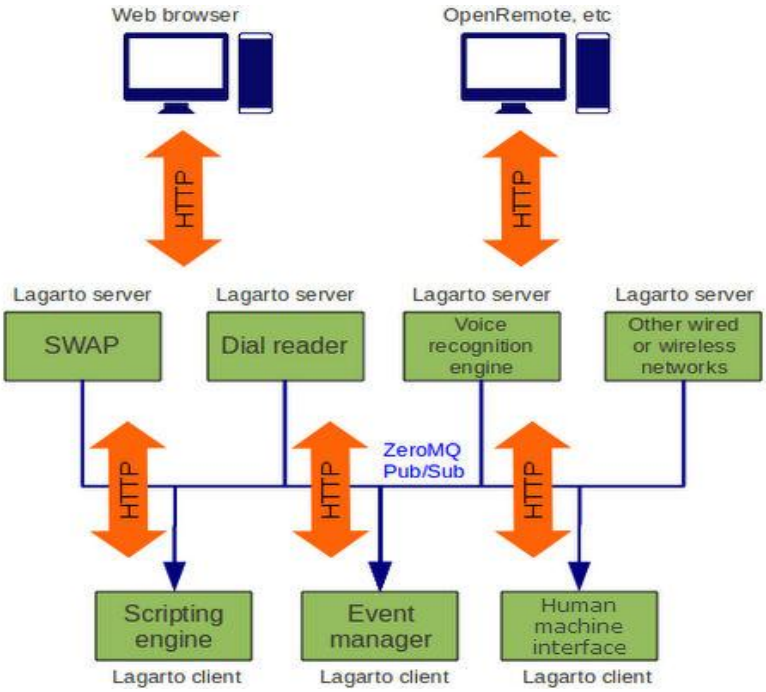

Figure 20: Lagarto architecture [39]

Lagarto is created for automation in buildings and industrial plants. It provides communication tools and computing infrastructure to build complex networking applications, capable to integrate resources from very different technologies and present them in a homogeneous way.

3.1.4.1 Simple Wireless Abstract Protocol (SWAP)

SWAP or Simple Wireless Abstract Protocol is the protocol used by the Lagarto servers. It is an application layer lightweight protocol for the wireless communication of panStamps. SWAP is designed to be used on Texas Instruments CC11XX-based radios and it relies on their packet structure which is based on abstract registers and was inspired by Modbus [42].

SWAP provides mechanisms for sending, requesting and controlling abstract data registers. Freeing the protocol from data types simplifies a lot the implementation. Instead, end-applications are accessing the definition folders. The definition folders are central repositories containing the registers' specifications in form

of XML files. Each register used in SWAP devices is described in detail in an XML file on the Lagarto SWAP's directory and not in the OS of the end-device. This way SWAP end-devices remain lightweight and only maintain a small amount of mandatory registers containing basic information like communication channel, device address and network id [42].

The Packet structure can be used under two different addressing schemes: single-byte addresses and 2-byte addresses. The function byte will tell the application which schema is being used in each frame [42].

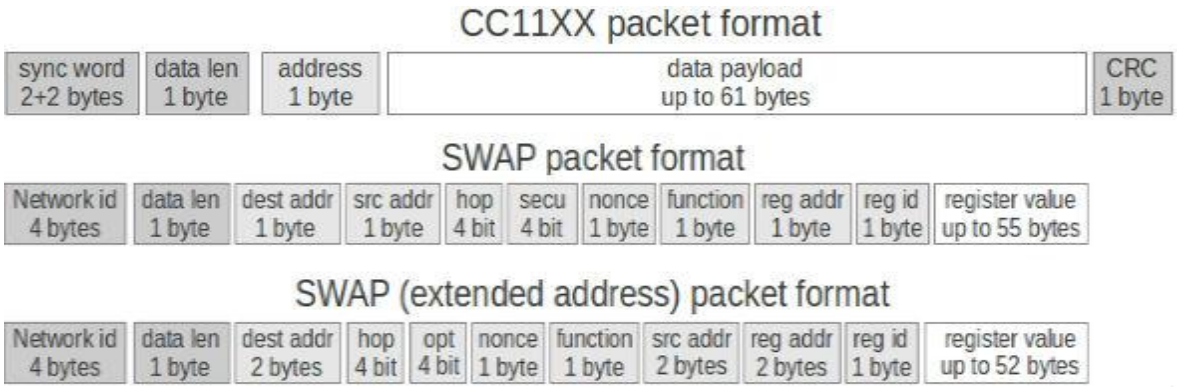

Figure 21: SWAP packet structure [42]

Network ID

SWAP uses CC1101's Synchronization Word to identify the wireless network. Defining different network ID's, multiple wireless networks can co-exist without interaction between them. Length of the Network ID: 2 bytes.

Function Codes

There are only three function codes:

1. **Status packet** report actual register values.
2. **Query packet** request information from remote devices.
3. **Command packet** are used to control register values on any remote device.

All SWAP packets share the same structure except query packets that don't contain a data field. This makes parsing wireless packets simpler.

In order to guarantee interoperability between devices, SWAP uses the following techniques:

- The reception of a query packet must be followed by the transmission of a status message. Thus, querying a register must be answered by a status packet containing the register value being queried.
- After receiving a command packet, a status message has to be sent with the register being modified.
- Developers may decide whether transmit periodic status message or not and the transmission interval in each case.

The SWAP packet structure is analyzed below.

Device address

SWAP device addresses are 1-byte or 2-byte length, depending on the addressing schema being used. Broadcast address is 0 so wireless devices must take an addresses between 1 and 255 (0xFF) for the simple addressing schema or between 1 and 65535 (0xFFFF) for the extended schema.

Each SWAP packet contains three different device addresses:

1. **Destination address** is the address of the device targeted by the SWAP packet.
2. **Source address** is the address of the device that sends the packet.
3. **Register address** is the address of the device that actually owns the register being queried, controlled or reported.

Register ID

Each register is uniquely identified within a device by this 1-byte digit. Combined with the register address, any register is uniquely identified within a wireless network. Length of the register ID: 1 byte.

Wireless hop

This 4-bit field counts the amount of times that a wireless packet is repeated. When a packet is originally generated, the hop counter is 0.

Security option

This 4-bit value specifies the type of security defense used to protect data fields and avoid external attacks.

- security = 0 -> No data encryption, security nonce disabled
- security = 1 -> No data encryption, security nonce enabled
- security = 2 -> Smart encryption and security nonce enabled

Smart encryption is implemented from firmware and, as such, it has been designed to be efficient and effective. It runs a simple XOR-based encryption using a 12-byte password and the nonce.

Security nonce

When enabled, security nonce provides a way of combating against play-back attacks. Every wireless device maintains its own single nonce counter. Whenever a device sends a status packet, the nonce is incremented by one. Status packets that do not contain the correct nonce have to be discarded by any other device listening the media. Later, any device wanting to send a command has to include the current nonce of the targeted device.

Query packets do not have to include the correct nonce. Thus, nonce = 0 in all query messages.

Cyclic Redundancy Check

This is a simple 16-bit CRC field, automatically calculated and appended by the CC11XX IC, based on the contents of the cc11XX data payload. It is an error detecting mechanism.

3.1.4.2 Lagarto SWAP

Lagarto SWAP is the M2M server that communicates directly to the panStamps. This process is written in Python and relies on pySWAP, a library than contains the implementation of the SWAP protocol for Python applications. At the time of writing of this thesis, there are no libraries to support other programming languages [40].

Lagarto SWAP runs on the gateway and requires a panStamp to be connected to the serial port and act as a modem for the network traffic. The server is bound to listen to the serial port for SWAP traffic and is able to send a command to the serial port that will be transferred to the panStamp for wireless transmission. This way all the SWAP messages transmitted from the panStamp network are received from the panStamp which is connected to the gateway, sent to the serial port and captured by the Lagarto SWAP server.

The SWAP protocol defines that the first transmission of any SWAP device includes the Product ID. This is how Lagarto SWAP auto-detects new devices. When a device is auto detected from Lagarto for the first time, the server uses the product ID to find the device in the definition folders and uses the device's XML file to add its registers in the monitor interface.

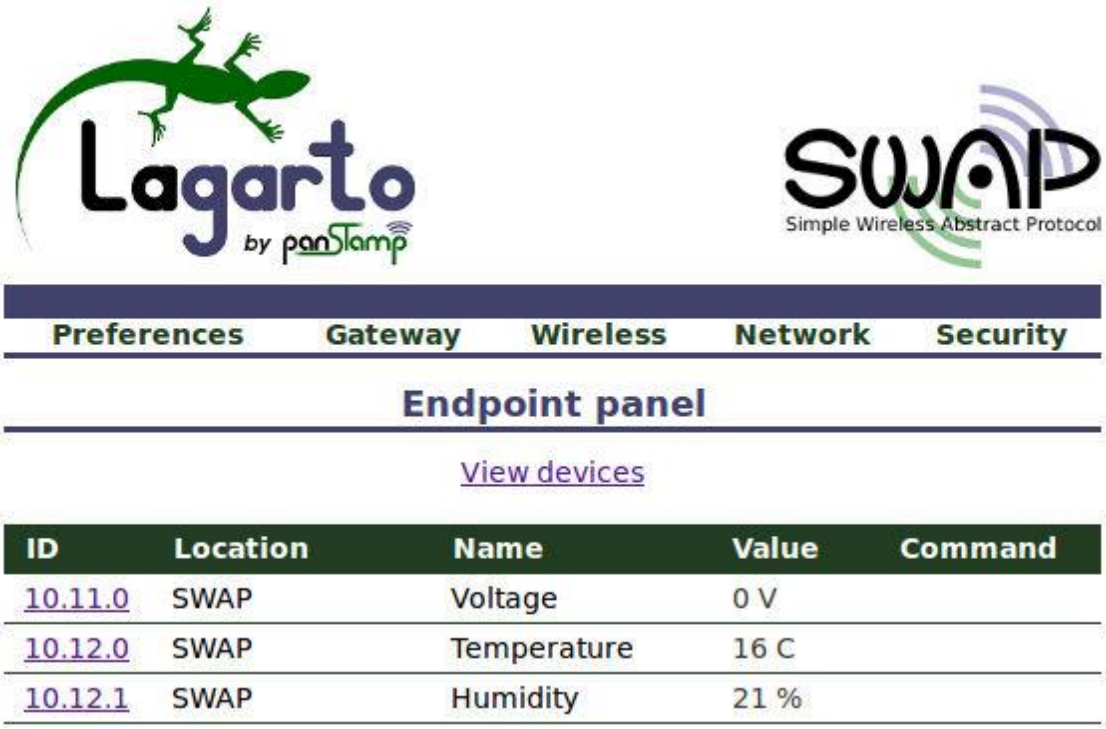

Figure 22: Lagarto SWAP monitor interface

The server is configured from a web interface. It provides a simple interface for viewing and controlling SWAP values and for setting up the network with options like serial port connection speed, Broadcast TCP/IP channel and HTTP server port. The default HTTP port for the interface is 8001 and can be accessed from the computer running the server or any other computer inside the LAN. The URL for the default port is: http:\\ip_address:8001.

3.1.4.3 Lagarto MAX

Lagarto MAX is a software process that runs on the gateway and communicates with one or more M2M servers running on the same network using the ZeroMQ message queue. It is a basic component to the Lagarto architecture because it's able to collect end-device values belonging to different networks and present them to the user in a homogeneous way. Moreover, being user-programmable, Lagarto MAX can send to the end-devices automated commands based on time or sensor values [38].

The Lagarto MAX server has a web interface for monitoring and controlling any Lagarto endpoint on the same network and it offers an event manager which can be programmed from the web interface to interact with the end-devices. The event manager can also be programmed to push end-device values to the supported virtual clouds (ThingSpeak, OpenSense, Xively, Twitter, GroveStreams). The default HTTP port for Lagarto MAX is 8002 so the URL for accessing the web interface is: http:\\ip_address:8002.

There are three sections in the web interface regarding programing events:

1. **Triggers** are initial conditions required to start an action. Multiple triggers can be added on a single event but only one of them has to be fulfilled in order to run the subsequent actions. There are two different types: network endpoint conditions and time conditions.
2. **Additional conditions** don't trigger the event by themselves but all of them must be fulfilled. There are two different types: network endpoint conditions and time conditions.
3. **Actions** can be of two types: endpoint commands and pushing network values to a virtual cloud platform.

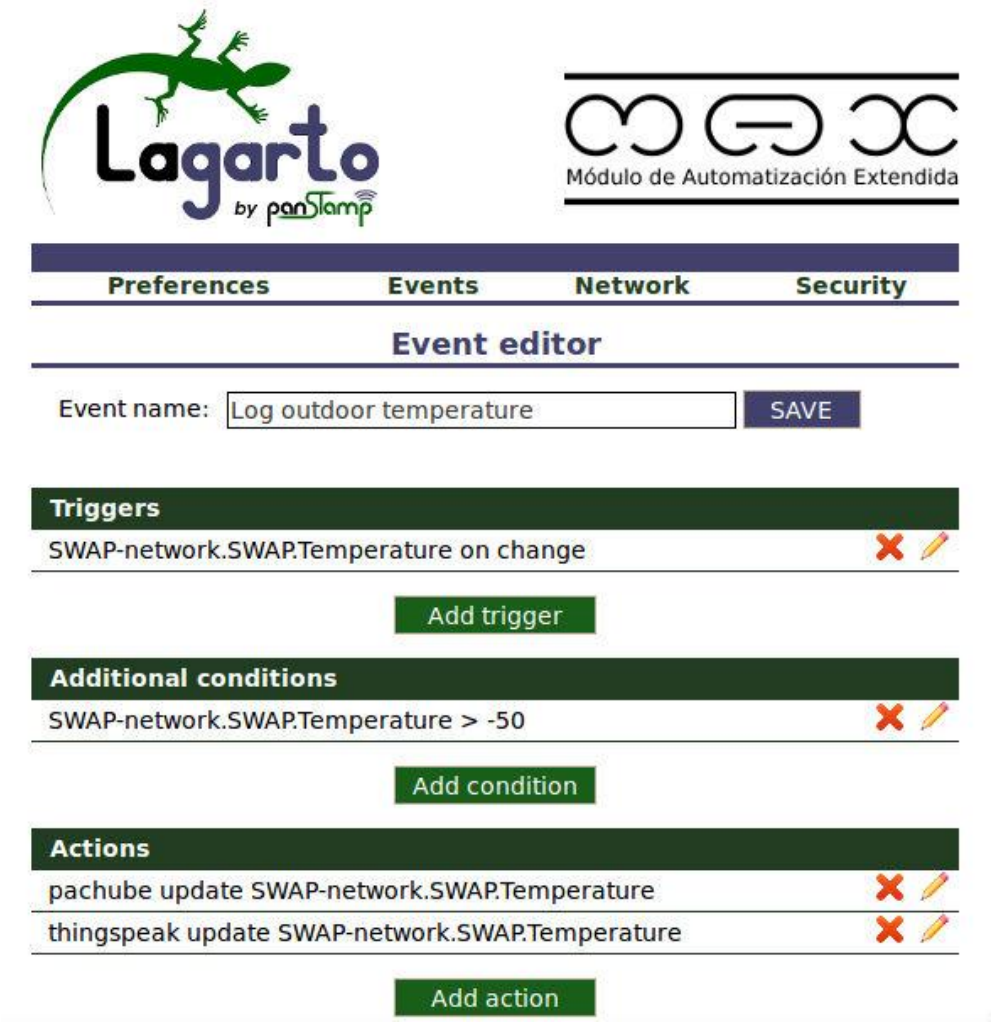

Figure 23: Lagarto MAX event interface [38]

More complicated events are also possible by altering and adding more code to the file where the events are programmed (webscripts.py), to the file that handles the HTTP requests (clouding.py) and to the file that hosts the network configuration (api.py). All these files are inside the Lagarto MAX directory and can also be found in the available online release of the panStamp project [36]. The webscripts.py file contains all the programmed events and is initially empty.

3.2 Raspberry Pi

The Raspberry Pi, as it will be discussed later, makes an ideal gateway for end-devices. It is a low-cost**,** credit card-sized computer that plugs into a regular monitor and uses a standard keyboard and mouse. It's capable of doing everything a normal computer does. It is created by the Raspberry Pi Foundation which is registered as educational charity and is based in the UK [45].

Figure 24: Raspberry Pi board (model B) [46]

There are currently two models available. The main differences are that Model A has 256MB RAM, one USB port and no Ethernet port. The Model B has 512MB RAM, 2 USB ports and an Ethernet port.

Product Specifications for Raspberry Pi (Model B) [46]:

- CPU: 700 MHz ARM1176JZF-S core
- GPU: Broadcom VideoCore IV @ 250 MHz
- Memory (SDRAM): 512 MB (shared with GPU)
- USB 2.0 ports: 2
- Video outputs: RCA ,HDMI
- Audio outputs: 3.5 mm jack
- Onboard network: 10/100 Mbit/s Ethernet
- Power source: 5 V via MicroUSB or GPIO header
- Size: 85.60 mm × 56 mm (3.370 in × 2.205 in)
- Weight: 45 g (1.6 oz)

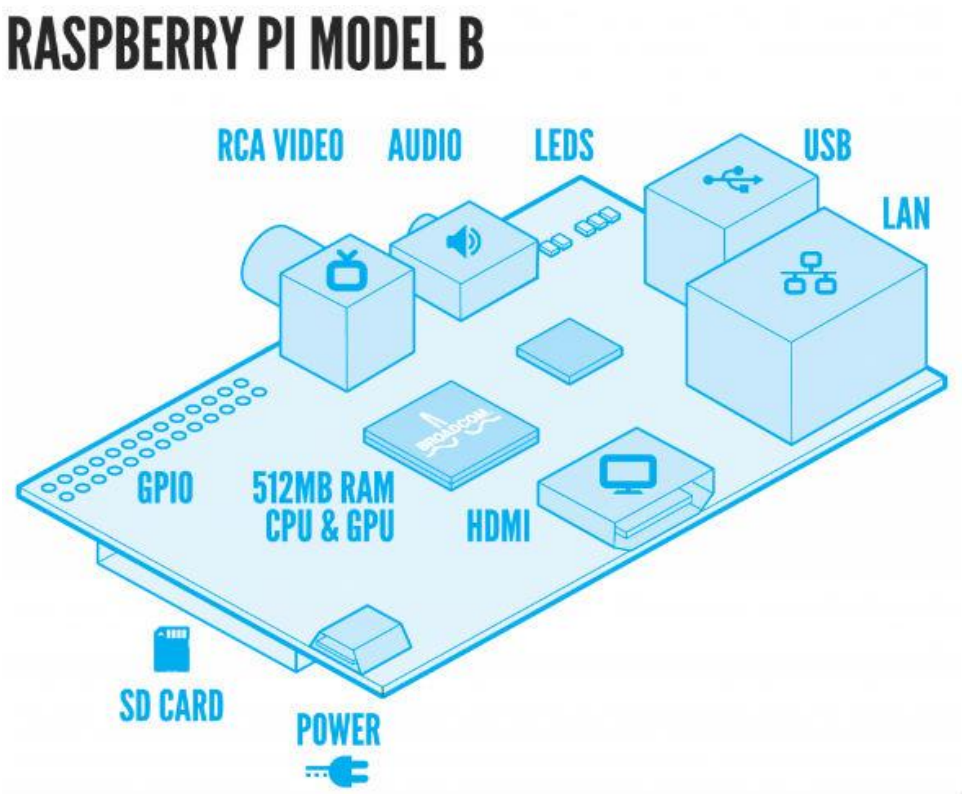

Figure 25: Raspberry Pi (model B) design [45]

The Raspberry Pi community has released an operating system manager for easy installation of an OS, called NOOBS. NOOBS (New Out Of the Box Software) is an install manager that lets the user select out of a list with several operating systems and handles all the installation process. The recommended OS is called Raspbian and it's the most commonly used. Raspbian is a Debian-based OS optimized for the Raspberry Pi hardware. [46].

The Raspberry Pi itself has no integrated hard drives for storage, it uses a regular SD card (8GB or more recommended). To get started with the Raspberry Pi the user has to load the NOOBS image or the image of a specific operating system to the SD card, plug the card to the Raspberry Pi and power it on [28].

The Raspberry Pi comes with a General Purpose Input Output (GPIO). The GPIO can be used for UART (Universal Asynchronous Receiver Transmitter) or SPI (Serial Peripheral Interface) communication and can provide an interface to connect to various modules that allow interacting with the physical world such as a GPS sensor, a camera or a wireless transceiver.

3.2.1 Raspberry Pi PanStamp Shield

The panStamp shield is a module that plugs into the Raspberry Pi's GPIO and connects it to the panStamp sensor network. The shield has an on-board panStamp that receives CC1101 messages and transfers them to the Raspberry Pi.

Figure 26: PanStamp shield for the Raspberry Pi [35]

It also includes a real-time Integrated Circuit (IC) with battery backup so that the Raspberry Pi does not depend on Internet connection to get the current time, even after an outage.

Features: [35]

- A panStamp managing all the low-power wireless communications on the 868/915MHz ISM band.
- SMA connector for external antenna
- DS1338 IC for the Real Time Clock (RTC) function with battery
- Size: 44 x 34 mm

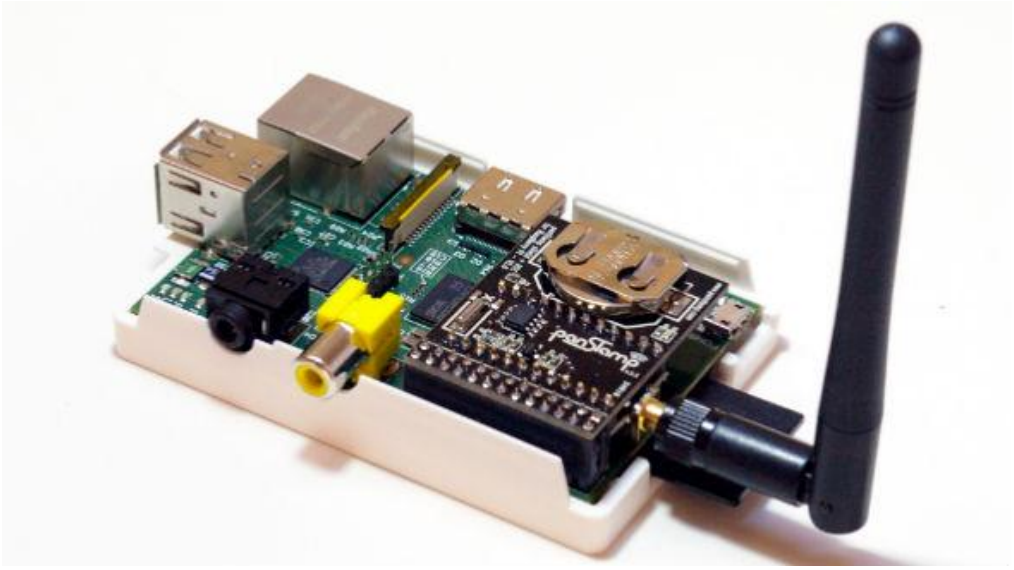

Figure 27: PanStamp shield attached to the Raspberry Pi [35]

The Raspbian OS has the GPIO pre-configured to be used as a serial terminal. In order to use the GPIO with the shield, it has to be configured to stop working as a serial console. There are many online tutorials to do that, one is mentioned in the references [56].

3.2.2 The Raspberry Pi as a Gateway

Small boards with integrated microcontrollers like panStamps are low-power and low-cost, suitable to be deployed to a broad area. However they are not able to produce packets that can be routed to the Internet. Thus a regular gateway that only routes incoming and outgoing traffic is not appropriate. Constrained resource end-point networks require a gateway to receive sensor data, translate them to routable packets and then route them to the Internet. This means that the OS running on the gateway, has to allow the implementation of servers that include network libraries and run constantly in order to translate messages to routable packets in real time.

The Raspberry Pi is a very low-power computer with many capabilities that create ideal conditions to make it the gateway for end-device networks. Raspbian the OS running on the Raspberry Pi is Debian-based and supports a plethora of programming languages and libraries to implement network protocols [44]. In addition, the Raspberry Pi has an Ethernet interface and can also connect to Wi-Fi and cellular networks by using extra modules. For this reason, it can upload information to the Internet with various ways such as 2G/3G/LTE/Wi-Fi/Ethernet. It

is extremely low power and only draws few watts of electricity. It has a huge community supporting it and there are many forums that help new developers accomplish their tasks. Another important reason that makes the Raspberry Pi a great gateway is that there are many modules that have been developed to fit in Raspberry Pi projects such as Camera module, GPS module, cellular Internet shields etc.

3.2.3 Practical Implementation

The Lagarto SWAP and the Lagarto Max servers are successfully set up on the Raspbian OS and a panStamp shield is attached to the GPIO of the Raspberry Pi. The panStamp shield relays SWAP messages to the Lagarto SWAP server, enabling the Raspberry Pi to connect to the SWAP network. There are also some panStamps attached to base boards. Each base board have an AA battery to supply power to the panStamps and is connected to a temperature sensor, a humidity sensor and an LED light which is an actuator. The SWAP protocol is implemented in the panStamps which are programmed to send frequent updates of the sensor values to the Lagarto SWAP server and are able to receive commands at any moment for the actuators. The Lagarto Max server can create events that based on the network values can turn on/off an LED light or upload sensor measurements to a virtual cloud for online monitoring. The polling mechanism is implemented on the Lagarto MAX source code in order to add online control capabilities to the end-device network.

At this point, the end-device network is connected to a gateway, the Raspberry Pi, that routes information to and from the Internet. The network is connected to a server, the Lagarto, that monitors and controls the sensor values locally and can also create automated events (e.g. if the temperature is above 20 turn on the LED). Online monitor and control is also achieved through the online virtual clouds that receive values through HTTP requests and send commands using the polling mechanism. Two virtual clouds are connected to the servers ThingSpeak and OpenSense. Both of them, along with their functionality, are reviewed in the next section.

The OpenRemote Controller is set up on the Raspberry Pi and communicates with Lagarto MAX through the ZeroMQ message queue. The OpenRemote Designer is

used for designing an application layout and also to collect all the necessary information about the panStamp network's registers. After that, the controller downloads an XML file with all the register details from the designer and can relay the sensor network to the OpenRemote application. All servers communicate with each other using the ZeroMQ message queue over TCP. The OpenRemote android application is downloaded on a smart phone and provides the first working application for monitor and control over the sensor network.

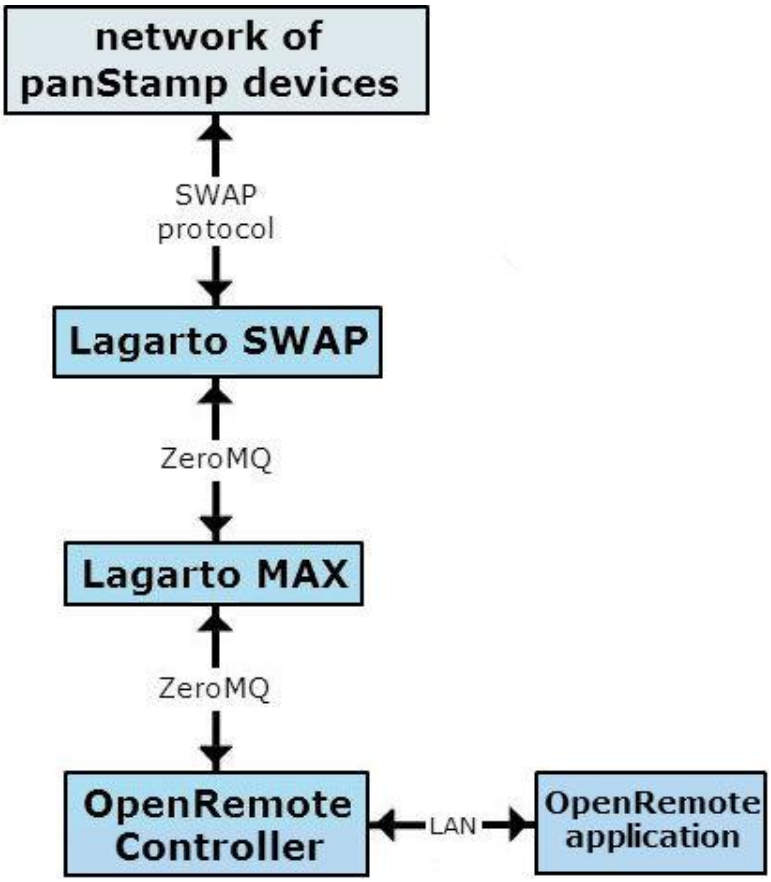

Figure 28: Connection among servers

The OpenRemote application has to be installed on a device inside the Raspberry Pi's LAN. Any computer inside the LAN can access the Lagarto servers and gain access to the end-device network through the Raspberry Pi's IP and the Lagarto's port number (e.g. http:\\ip_address:8001), but OpenRemote is the easiest way to have a working application for smart phones.

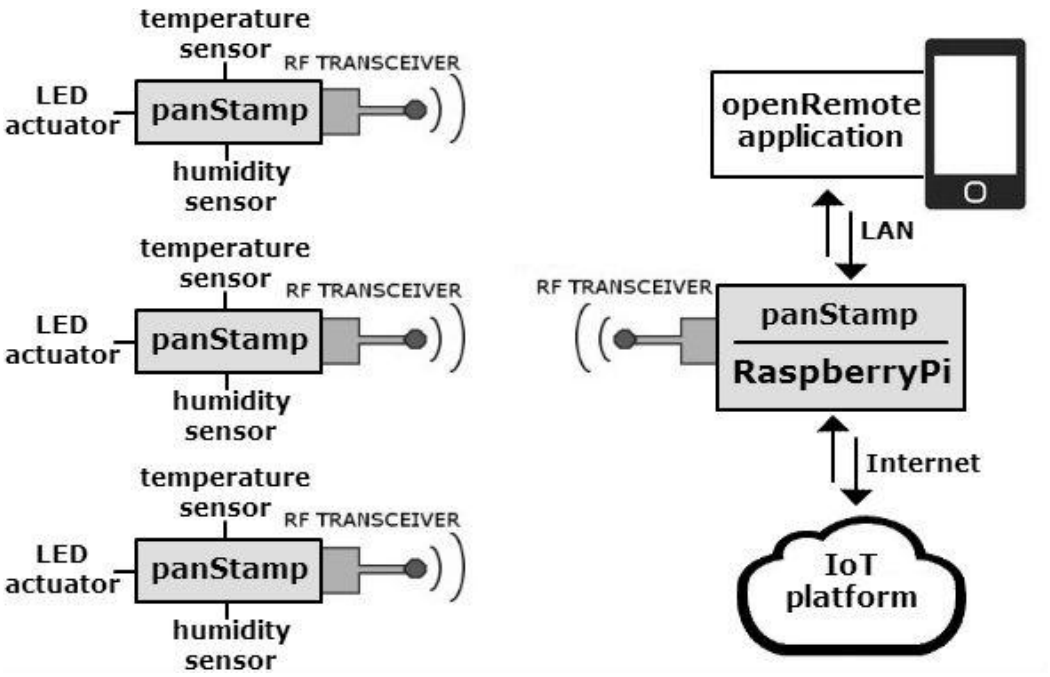

Figure 29: Implemented end-device network with Internet connectivity

3.3 Tested Cloud Platforms

After setting up the end-device network and successfully connecting it to the gateway, the sensors are ready to communicate with the IoT platforms. The IoT platforms provide an interface for the users to monitor/control the network, store the sensor data and visualize them using built-in graphs and charts. They also work as nodes for tablet and smart phones applications to connect to and gain full control over the end-device network through the Internet. Two cloud platforms are connected to the sensor network and are reviewed in the next sections:

1. **ThingSpeak** because it is the only open source IoT platform available.
2. **OpenSense** because of its simplified and intuitive interface.

3.3.1 ThingSpeak

ThingSpeak is the first virtual cloud, selected to communicate with the gateway and thus, with the end-devices. The ThingSpeak architecture is based on channels. Each channel represents an end-device and supports up to 8 data fields. It also has the option to import/export stored data and create multiple authentication keys for accessing the data with different permissions.

A panStamp is programmed to send measurements from the temperature sensor and the humidity sensor to the gateway which is configured to update the

ThingSpeak channel using HTTP POST requests. All the HTTP requests have to include in their header the authentication key obtained from the ThingSpeak channel for security reasons. To lower the power consumption of the panStamp, it is programmed to execute a loop of transmitting values and hibernate for twenty minutes.

The channel receives frequent updates and displays dynamic graphs with the sensor measurements. The sensor is placed inside our office and the graph demonstrates the temperature and humidity changes during a two week period.

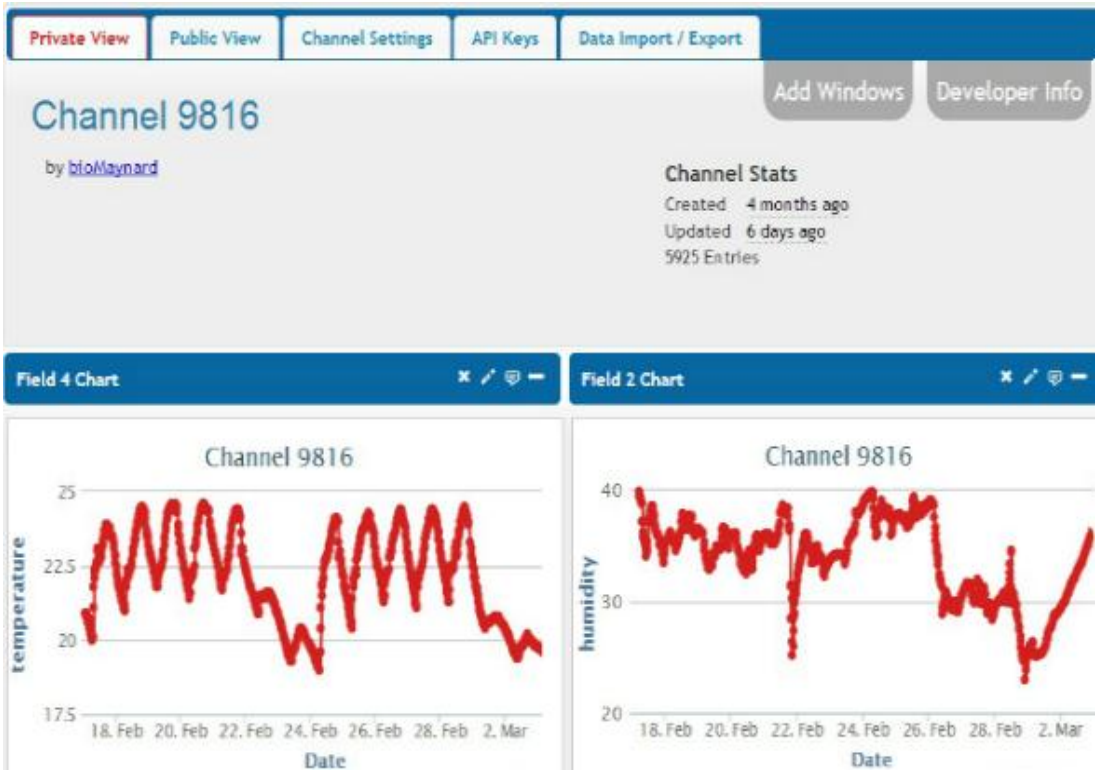

Figure 30: Temperature/humidity monitoring from ThingSpeak channel

3.3.2 OpenSense

The OpenSense cloud is also connected to the gateway. Another panStamp is programmed to sense temperature and humidity and transmit the values to the cloud, where users can monitor the measurements using the user interface through the Internet. All HTTP requests have to include the authentication key in their header for security reasons.

The panStamp is also programmed to receive commands regarding the LED light that represents a room light. The light is controlled using the polling mechanism,

through which, a command is sent from the online cloud to the light. To control the light from the online user interface, a virtual switch provided from the OpenSense cloud is used. The switch only controls a light but using a different actuator, it could be a window, a garage door or the heating system. Using this method it is now possible to achieve complete automation of buildings over the Internet, using different kinds of actuators.

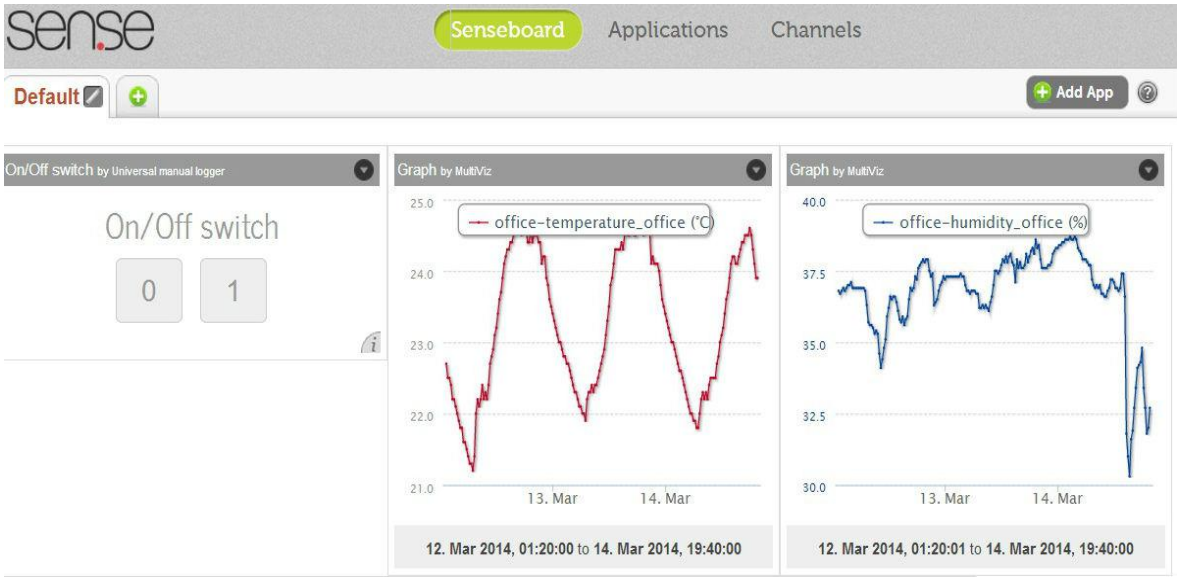

Figure 31: OpenSense monitor/control interface

3.4 Security Aspect

A complete Internet of Things environment is created including an end-device sensor network, a gateway and connection to online virtual clouds. The system is ready to be used for testing purposes but there is one concern. Is it safe to use it in the real world, or there is the risk to jeopardize the user's privacy.

The suggested architecture consists of different components. While implementing the architecture, a lot of attention is paid in connecting the different parts and technologies without raising security concerns. Each connection is established considering third party intruding attempts.

The sensor networks and the gateway are safe behind a firewall that only allows responses to pass. All the HTTP requests for altering the values on the cloud have to include an authentication key in their header otherwise they are being ignored. Tablet and smart phone applications use the cloud as a node and do not have direct access to the gateway. They communicate with the gateway using HTTP

requests that are targeting the virtual clouds and they have to obey the rule of the authentication key.

Taking everything into consideration, it is based on true facts, to say that the system is very well protected.

3.5 The public IP approach

Composing an architecture for an IoT testbed is not straightforward. Alternatives are discussed and some of them are even deployed. Among them, there is a non cloud-centric approach which is also implemented. In this case, the applications target directly the M2M servers that are now treated as web servers. The primary advantages of skipping the cloud platform is that by establishing direct connection to the server, possible cloud limitations are avoided and the cloud traffic is unnecessary which makes the communication faster. The M2M server is responsible for monitoring end-devices and coordinating events. By accessing directly the server's features, applications obtain full control over the end-devices.

To achieve this, Network Address Translation (NAT) configuration is needed in the router that provides the Internet IP. The Internet architecture is such that when a request packet is initialized from a station to reach an Internet IP address, it goes through the router. The router changes the source address and the port of the packet to the public IP address of the router and a dynamic port. It saves both the addresses in the NAT tables so that when the response packet arrives at the router it can forward it to the correct station. This is a technique that is used to provide more IP v4 addresses to the Internet. This way a station’s IP address remains hidden from the Public Internet. The downside is that it cannot be directly addressed from the Internet. If a packet arrives at the router and its destination address in not in the NAT tables, the packet will be rejected.

A solution to this is the Network Address Translation or port forwarding. In order to deliver a request packets to a station inside the LAN from the Internet, the router creates a permanent rule which says that all the traffic that is addressing the router’s public IP and a specific port will be forwarded to the station’s local IP and another specific port. For instance all the incoming Internet traffic that targets the public IP http://88.102.77.215:8001 will be forwarded to the local IP http://192.168.1.125:8005. This way, clients can initialize HTTP GET and POST

requests that reach directly the M2M server without using a virtual cloud. The values are stored and visualized locally and the clients can request any information at any time. This approach is implemented and linked to applications for testing purposes but it is not embraced due to the security concerns that it raises.

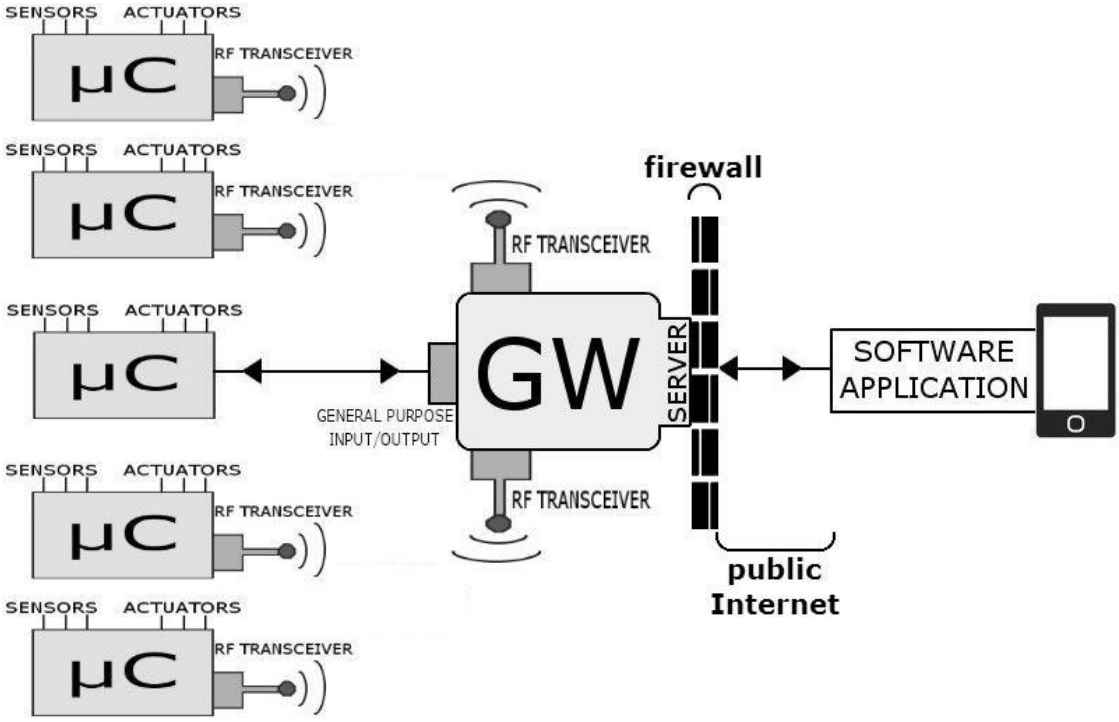

Figure 32: The public IP approach of an IoT testbed

Opening a port to the public, means that anyone can use the port to extract information or control the end-devices. The security issue can be resolved sufficiently with various safety measures like user authentication in the software side of the system and encrypted port knocking in the network side. Port knocking is a mechanism that allows access to a port, only after trying a specific combination of ports first. It works as a complicated password and encrypted port knocking prevents sniffing which makes it even more difficult to intrude. In conclusion, this structure works effectively for automation in a solitary level. However, deploying this architecture to broad areas with thousands of appliances will require complicated password management systems that can be prevented.

3.6 Alternative Components for the Testbed

The testbed is a collection of software processes and hardware devices configured to coexist and communicate with each other in order to provide an End-to-End

communication network. The software involved can be modified, adapted or even replaced at any time in found incompetent. On the contrary, hardware devices require funds and could constrain the entire functionality in found incapable, thus they have to be selected carefully.

We selected the panStamps for being small, low-power and with an integrated RF transceiver which is not common among other similar boards. The Raspberry Pi is selected because it has the necessary interfaces (UART, USB, Ethernet), there are many available modules to add more interfaces (3G, LTE, Wi-Fi) and most importantly since it needs to run non-stop, because it is extremely low-power. Each of the following lists contains four alternatives to our hardware selections.

PanStamp alternatives:

1. **Arduino uno** is a famous board with an 8-bit ATmega328 microcontroller with 2KB of RAM, 16MHz clock speed and a huge community.
2. **MSP-EXP430G2** has a 16-bit MSP430 microcontroller with 512 bytes of RAM and 16MHz clock speed.
3. **Nanode** has an ATmega328 microcontroller, same as Arduino and panStamp but it also has an on-board Ethernet port for Internet connectivity.
4. **STM32VLDISCOVERY** has an STM32F100 microcontroller based on the 32-bit ARM Cortex M3 core running at 24MHz with 8KB of RAM.

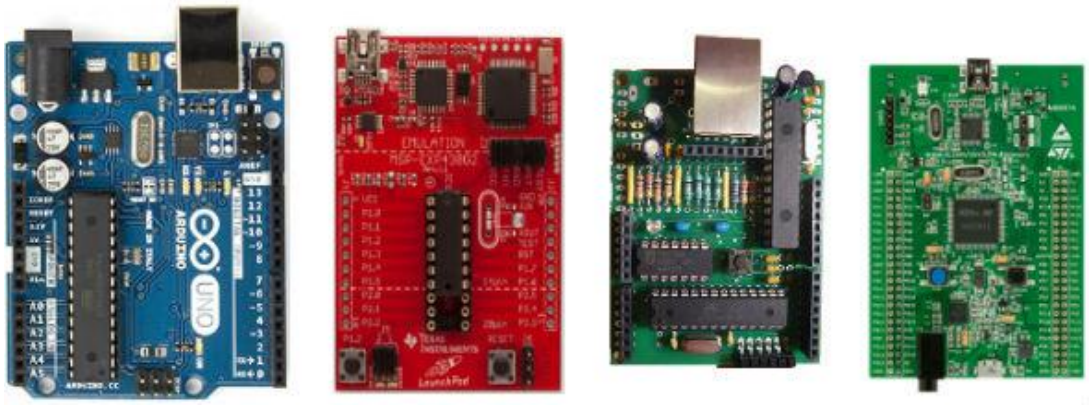

Figure 33: Arduino Uno, MSP-EXP430G2, Nanode, STM32VLDISCOVERY (left to right)

Raspberry Pi alternatives:

1. **Intel MinnowBoard Max** has a 64-bit Intel Atom E3825 dual core processor at 1.33GHz and 2GB DDR3 RAM. It supports Debian/Linux and Android OS.
2. **BeagleBone Black** has an ARM Cortex-A8 processor running at 1GHz and 512 DDR3 RAM. It supports Debian, Ubuntu and Android OS.
3. **UDOO** has an ARM Cortex-A9 Dual/Quad core 1GHz CPU and 1GB of DDR3 RAM. It also has an on-board Wi-Fi interface.
4. **HummingBoard** is almost identical to the Raspberry Pi but with an ARM Cortex-A9 Quad core 1GHz processor and 2GB of DDR3 RAM.

Figure 34: BeagleBone Black, UDOO, MinnowBoard Max and HummingBoard (left to right)

Chapter 4

Implemented Real Life Services

After implementing a complete End-to-End Machine to Machine communication network, next step is the integration of representative applications of the Internet of Things. This chapter describes Android, the targeted OS for the applications along with two smart phone applications, designed to improve the quality of life in modern urban environments.

The applications are Smart Parking and Geo Fencing. Smart Parking provides an information system regarding the status of common parking spaces. Geo Fencing is a process that allows remote automation based on the geographical position of the user. Both applications are implemented targeting the Android OS so that users can carry them in their mobile phones and use their services at any moment.

4.1 Android Programing

The design of useful and functional applications is an important part of this thesis. Applications that accompany people at any place, and offer their services whenever an opportunity presents itself. To this end, the created applications are targeting the world's most famous mobile operating system, Android. GOOGLE being the company that released Android, has established a huge community for android developers. There are available development tools and libraries to aid in the spread of the OS. In addition, tutorials and detailed guides have been released to encourage amateur android developers. Android is open source and is supported by thousands of machines including mobile devices and tablets [3].

To connect an Android application to an online IoT platform and thus, to the end-device network, the application uses HTTP GET requests to extract sensor data and HTTP POST requests to send commands for the actuators. This way the mobile devices obtain full control over the sensor network and can provide user-friendly and functional applications.

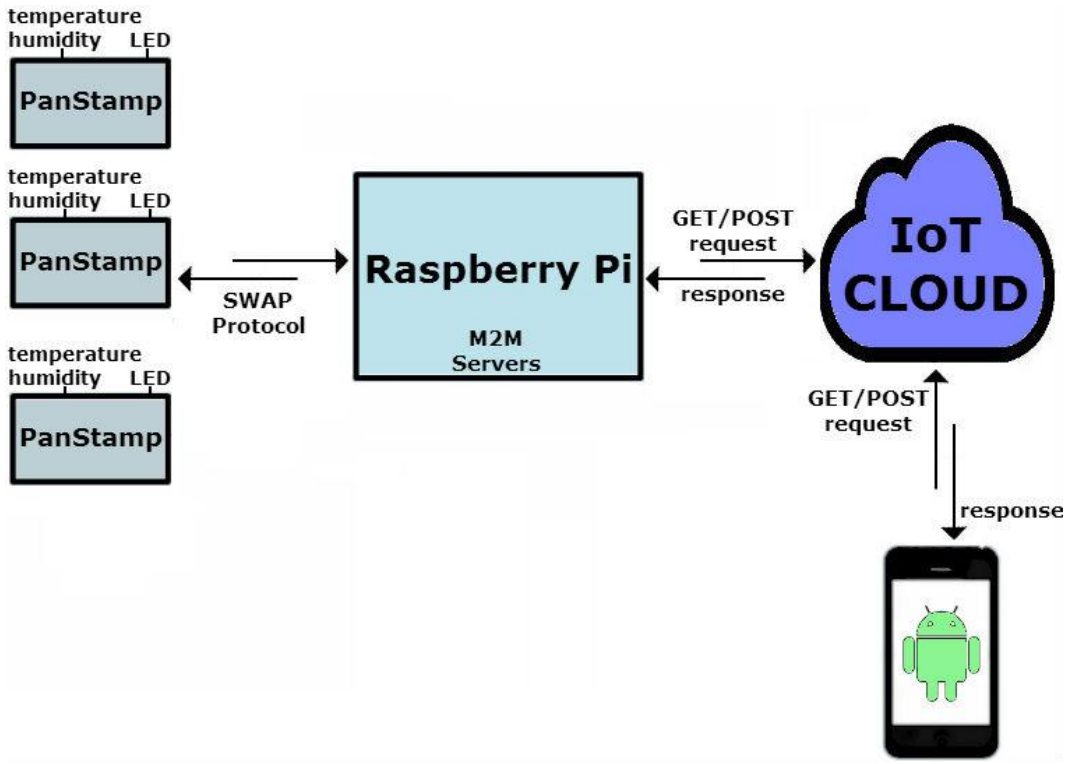

Figure 35: Android connectivity to the end-devices

4.1.1Google Maps Android API v2

One of the reasons that Android is the targeted OS for the applications is the huge community that supports it and the huge amount of libraries and APIs that help Android developers. A very interesting and useful API that will be widely used in IoT applications is the Google Maps API that provides map visualization of users, sensors, actuators and generally all the points or areas of interest.

With the Google Maps Android API, maps based on Google Maps can be added to applications. The API automatically handles access to Google Maps servers, data downloading, map display, and response to map gestures. The API can also be used to add markers, polygons, and overlays to a basic map, and to change the user's view of a particular map area. These objects provide additional information for map locations, and allow user interaction with the map. The API allows these graphics to be added to a map [16]:

- **Markers** are icons anchored to specific positions on the map.
- **Polylines** are sets of line segments.
- **Polygons** are enclosed segments.

- **Ground Overlays** are bitmap graphics anchored to specific positions on the map.
- **Tile Overlays** are sets of images displayed on top of the base map tiles.

The official guide with information about the use and the implementation of the Google Maps API can be found in the references [15].

4.2 Smart Parking Application

There are currently monstrous amounts of vehicles swarming the cities and parking has become a rare commodity in most urban centers. Traffic congestion, environment pollution and waste of energy recourses are a few downsides of the phenomenon when vehicles are wandering around searching for available spots. A lot of research is being conducted all over the world in order to implement better parking management techniques, but the problem still persists.

To provide an IoT solution, we connect an application to the testbed. Initially, panStamps have to be connected to sensors that can examine the parking space and successfully decide if it is occupied by a car or not. Many kinds of sensors can be used like magnetic sensors that detect metal, ultra-sonic wave sensors that detect objects in the line of sight or pressure sensors to calculate weight. The sensors we identify as the most appropriate are the magnetic sensors because they can be hidden away or even buried under the parking space so they can be protected from theft or weather conditions. No real sensors were involved during the development of this application. Instead they were simulated with a digital input on the microcontroller where digital 1 means that there is a car and digital 0 means that the space is available. The microcontroller can also be connected to a GPS sensor in order to know its location or its exact coordinated can be integrated in the firmware.

PanStamp are programmed to transmit to the M2M server the values of the sensors that decide if there are cars in the parking spots and the coordinates of their exact location. Every time a value changes, the panStamps send the new value to the M2M server so it is kept updated. The M2M server has a configured automated event and every time there is an update from the panStamps that represent parking spaces, it uploads all the information to a virtual cloud. The OpenSense platform is used for online storing of the data.

The Android application is used mostly for translating the data obtained from the IoT cloud into a simplified and user-friendly interface. It implements an HTTP library so it can send HTTP GET requests to the virtual cloud and acquire information about the coordinates and states of the parking spaces. The application also implements the Android Google Maps API v2 which enables the display of a map based on Google Maps. The specific API is selected because Android users are already familiar with the Google Maps interface so using the Smart Parking application is simple. The application processes the HTTP GET response and creates markers on the map with different colors and explanatory labels depending on the sensor values.

```
class Showallspots extends AsyncTask<String, String, String>

protected String doInBackground(String... uri) {
HttpClient httpclient = new DefaultHttpClient();                    //new http client object
HttpResponse response;                                  //new variable to save the http response
String responseString = null;                             //String variable to save the response
try {
response = httpclient.execute(new HttpGet(uri[0]));  //execute http request, save response
StatusLine statusLine = response.getStatusLine();                   //save conection status
if(statusLine.getStatusCode() == HttpStatus.SC_OK) {                      //if status is OK
ByteArrayOutputStream out = new ByteArrayOutputStream();  //create a Byte output array
response.getEntity().writeTo(out);                          //save response in the array
out.close();                                                      //close output stream
responseString = out.toString();}                          //convert response to String
else{
response.getEntity().getContent().close();                          //close http connection
throw new IOException(statusLine.getReasonPhrase()); } }
catch (ClientProtocolException e) {//TODO Handle problems}
catch (IOException e) { //TODO Handle problems }
return responseString;}}                                  //return http response in String format
```

Figure 36: Android HTTP GET request

A premium membership feature is also implemented by adding an output register on the panStamps. The register is configured as output so that it can be turned on and off dynamically. This way the premium membership status is not integrated in the firmware and it can be controlled from the M2M server. It is a virtual output that corresponds to no physical pin of the microcontroller. The M2M server updates the cloud about the state of the switch and the application translates it to premium membership if the switch is on and normal membership if the switch is off. Other features of the application is to show only the available spots or display the user's current location on the map.

Figure 37: Smart Parking Android application

Any new Smart Parking end-device that is powered on, will be auto-detected by the M2M server and its values will be automatically sent to the virtual cloud. After that, the application will request the data and a new parking space will be added to the map. This means that due to the architecture of the system, new Smart Parking end-devices can be added without further effort.

4.3 Geo Fencing Application

Geo fencing is a technology that uses the Geographical Positioning System (GPS) to define virtual borders around a specific region. The passing of the border can trigger actions and events programmed by the administrator. This means that by setting up virtual fences across the territory, points of interest can be created. This application aims in improving the quality of life in urban environments by using methods of automation and remote automaton based on the geographical position of the user.

For the Geo Fencing application a panStamp is programmed to control an actuator. The actuator used for the needs of the application is an LED light, but in real life the actuator could be integrated in the heating system, the garage door or the coffee maker. The M2M server communicates with the panStamp and can send commands to the actuator at any time. The polling mechanism is used so

that the server receives status updates regarding the state of the actuator from a virtual switch on the OpenSense cloud. The rest is handled by the Android application.

Initially the application's user interface requires some fields to be filled.

- **Latitude** defines the latitude of the center of the fenced area.
- **Longitude** defines the longitude of the center of the fenced area.
- **Feed ID** is the ID number of the virtual switch that represents the actuator.
- **Authentication key** is the key that needs to be in the header of the HTTP requests for security.

All the information is saved in a private file on the device so that they don't get erased when the application terminates. The user can create multiple virtual fences with different coordinates, ID's and authentication keys. The application implements the Android Google Maps API so the user can visualize his position and the location of the fences on a map. Finally, there is the option to enable the background service.

Figure 38: GeoFencing Android application

The background service is an autonomous process that finds the current location of the user using the Google Maps library and checks if his location is inside the virtual fence. If it is, it uses the HTTP library to send an HTTP POST request to the

cloud and turn on the virtual switch. The M2M server using the polling mechanism detects the change on the switch and turns on the actuator. This way the Geo Fencing application can perform automated tasks that require no human intervention.

The application is implemented using a background service so that the user can enable the service and use his device as a regular mobile phone at the same time. The background process and the M2M server handle the rest.

The application can be used in various scenarios. Could be to turn on the heating system when the user is approaching his home so that the house is already warm when he arrives, or to open the outside door automatically every time the user is outside his house. It is designed to be able to save many events with different actuators so it can automate many processes.

Chapter 5

Approaching the Electric Power Grid

This chapter describes the connection of the testbed to the electric power distribution system. It is divided in three main sections:

1. **The Current Power Grid** reviews the operation of the electric power grid. Generation, transmission, substations and distribution are described along with control centers and how they manage to coordinate the entire operation.
2. **The Need for Improvement** presents flaws and inabilities of the power grid and also failures and outages that happened over the past years.
3. **The IoT Perspective of a Smart Power Grid** describes the concept of smart grid, IEC 61850 which is a standard for automating the power distribution, and how the combination of the IoT testbed and IEC 61850 leads in the emerging smart grid.

5.1 The Current Power Grid

The power grid is a real-time energy delivery system, which means that power is generated, transported, and supplied the moment a consumer turns on a light switch. The system starts with generation, by which electrical energy is produced in the power plant and then transferred over long distances over high-voltage transmission lines to the substations. Substations transform this high-voltage electrical energy into lower-voltage energy that is transmitted over distribution power lines that are more suitable for the distribution of electrical energy to its destination, where it is again transformed for residential, commercial, and industrial consumption [51].

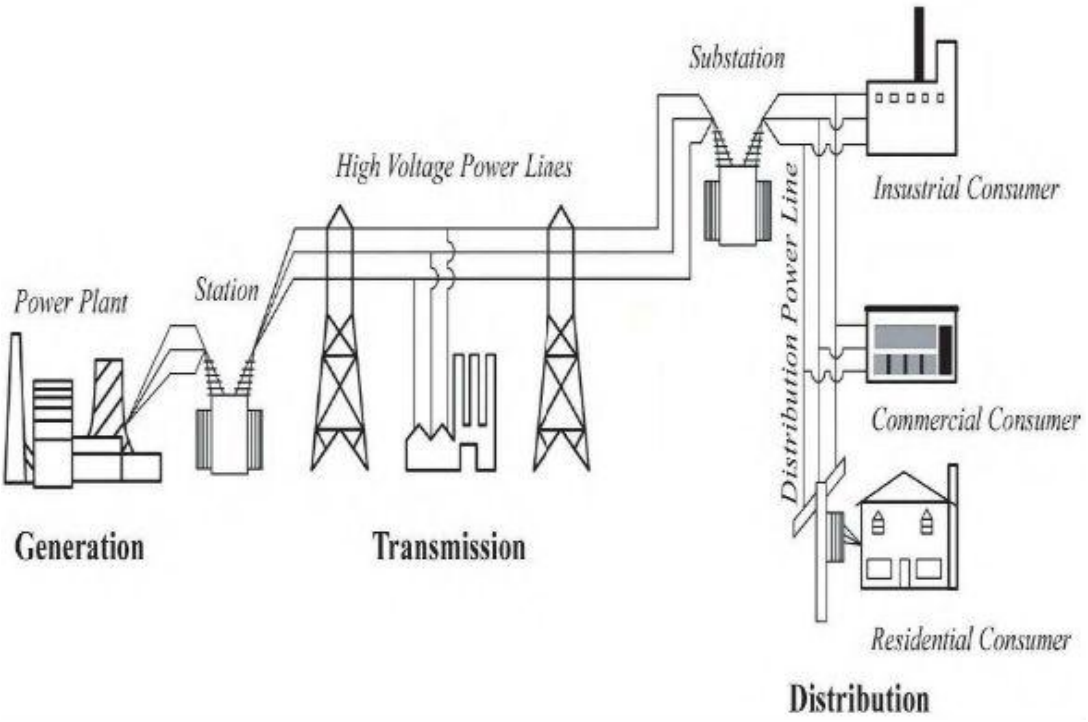

Figure 39: Power grid overview [51]

A full-scale actual interconnected electric power grid is much more Complex. However the basic principles, concepts and theories are all the same.

5.1.1 Generation

Power plants produce electrical energy on a real-time basis. Electric power systems do not store energy thus when an appliance is switched on and drawing electrical energy from the system, the associated generating plants immediately see this as new load and increase productivity to balance the demand on the system. There must always be enough generation online to maintain the balance during light and heavy load conditions [51]. Power plants produce electricity with a voltage of few thousands volts. A volt is a measurement of electromotive force in electricity. The electricity first goes to a transformer at the power plant that boosts the voltage to hundreds of thousands of volts and then it is sent to the transmission lines.

5.1.2 Transmission

The transmission system uses long thick cables for transmission lines made of copper or aluminum because they have low resistance to transfer the electricity over long distances. They use high-voltage transmission lines to minimize transportation losses. The lines usually have static wires on the very top to shield

itself from lightning. They are directly connected to the metal towers so that lightning strikes are immediately grounded to earth [51].

Figure 40: High-voltage transmission lines [51]

5.1.3 Substations

The power lines go into substations near businesses, factories and homes. Transformers change the very high-voltage electricity back into lower-voltage. From these substations electricity in different power levels is delivered to distribution or further transmission. Substations use a lot of equipment to operate, the most important types are the following [51].

- **Power Transformers** are used to convert high-voltage power to low-voltage power and vice versa. Generation plants use large step-up transformers to raise the voltage for efficient transfer. Then step-down transformers convert the power to sub-transmission.
- **Regulators** provide regulated and steady voltage all the time, otherwise several undesirable conditions might occur. Low voltage can cause motors to overheat and burn out. High voltages can cause light bulbs to burn out too often or cause other appliance issues.
- **Circuit breakers** interrupt current flowing in the line, transformer, bus, or other equipment when a problem occurs and the power has to be turned

off. A breaker accomplishes this by mechanically moving electrical contacts apart.

- **Disconnect switches** are used mainly to isolate equipment for maintenance purposes. Disconnect switches usually have low current interrupting ratings compared to circuit breakers.
- **Lighting arresters** are designed to limit the line-to-ground voltage in the event of lightning or other excessive transient voltage conditions.
- **Electrical bus** is a conductor, or group of conductors, that serves as a common connection between two or more circuits. It is meant to connect equipment together.
- **Capacitor banks** keep the transmission system voltage stable during disturbances. They are used to cancel out the lagging current effects from motors and transformers.
- **Control buildings** are commonly found in the larger substations. They are used to house the equipment associated with monitoring, control, and protection of the substation.

5.1.4 Distribution

Distribution systems are responsible for delivering electrical energy from the distribution substation to the service-entrance equipment located at residential, commercial, and industrial consumer facilities. They use transformers to convert the primary voltage to different consumer voltages for industrial or residential use [51].

Figure 41: Distribution system [51]

5.1.5 Supervisory Control and Data Acquisition

Control centers operate constantly to make sure the electric power system within their control area is operating properly. System operators are looking for signs of possible problems and taking immediate action to avoid major system disturbances. Operators are tasked with the responsibility to maintain system reliability, stability, and continuous service. They are also responsible for coordinating field crew work activities and making sure crews are safely reported on high-voltage lines and equipment [51].

The main tool control centers use is the Supervisory Control and Data Acquisition (SCADA) system. This system allows control operators to monitor, control and dispatch generation, and obtain written reports of all parameters about the power system. The SCADA system is made up of a centrally located master computer and several Remote Terminal Units (RTU) located throughout the system. Up until the late 1940s, many utilities had personnel stationed at substations. In some cases, these were residents who remained on call 24 hours a day. With the advent of SCADA, it was no longer necessary for utilities to maintain manned operation of

substations. SCADA gives operators the ability to remotely monitor analog electrical quantities in real time. Also, operators are alerted to problems as they occur through alarm and indication points [51].

5.2 The Need for Improvement

Sometime in the 1960s, the industry initiated the use of computers to monitor and offer some control of the power system. This, coupled with a modest use of sensors, has increased over time. It still remains less than ideal. Power system area operators can, at best, see the condition of the power system with a 20-second delay. Industry suppliers refer to this as “real time.” However, 20 seconds is still not real time when one considers that the electromagnetic pulse moves at nearly the speed of light. Actually, the electric power delivery system is almost entirely a mechanical system, with only minimal use of electronic communication, sensors and almost no electronic control [8].

The current power grid is not only outdated but, as history shows, it is also unstable. The following list contains unexpected blackout incidents that happened in the last 50 years [13].

- 2012 India
- 2011 Southwest United States Blackout
- 2008 Brazil
- 2007 Victoria, Australia
- 2007 Colombia
- 2006 large areas in Europe
- 2006 Tokyo, Japan
- 2005 Australia
- 2005 Moscow, Russia
- 2004 Greece
- 2003 Denmark and Sweden
- 2003 London, England
- 2003 Northeast and Midwest United states
- 2003 Italy
- 1999 Southern Brazil Blackout
- 1997 California

- 1997 New Zealand
- 1987 Tokyo, Japan
- 1978 France
- 1977 New York City
- 1967 Mid-Atlantic USA
- 1965 Northeast United States and Southeast Canada

The general impacts of power grid failures are numerous, including the loss of life, mainly due to accidents. Long-term large-scale outages affect many crucial everyday services like:

- **Transportation** (traffic lights),
- **Public services** (hospitals)
- **Communications** (telephone lines, TV and radio stations).

Many of these services have their own backup electrical power supplies, but the backup systems only last for short periods of time [13].

5.3 The IoT Perspective of a Smart Power Grid

The power grid can be upgraded using the components of the IoT. Sensors, communications, computational ability and control can optimize the overall functionality of the electric power delivery system. Even the improved power grids that use SCADA and achieve some kind of automation, rely on human operators. In order to optimize the process and accomplish actual real-time synchronization between demand and response, the human factor has to be replaced by M2M communications with negligible delays.

This section is divided in four sub-sections:

1. **Smart Grid**, which is an advanced electric power grid that uses modern technology to improve the quality of service.
2. **IEC 61850 and GOOSE**. This sub-section describes the IEC 61850 which is a standard for electric power substation automation that can turn the power grid into a smart grid. It defines the GOOSE message as a communication protocol for Intelligent Electronic Devices (IED).

3. **Phasor Measurement Units (PMU) and GOOSE generation** describes the PMU which is a device that can generate GOOSE messages and why it is a key component of the Smart Grid.
4. Finally, **Gateway for the GOOSE**, includes the practical implementation of connecting a PMU to the Raspberry Pi gateway and use the testbed to emulate a Smart Grid environment.

5.3.1 Smart Grid

Smart grid is an advanced system that will increase the productivity resulting from the use of electricity. The design of the smart grid addresses five main functionalities which should be part of the power system of tomorrow [8].

1. **Visualizing the Power System in Real Time.** This attribute would deploy advanced sensors more broadly, throughout the system on all critical components. These sensors would be integrated with a real-time communications system through an electric and communications system architecture.
2. **Increasing System Capacity.** This functionality embodies a generally straight-forward effort to build or reinforce capacity particularly in the high-voltage system. This would include building more transmission circuits, making improvements on data infrastructure, upgrading control centers, and updating protection schemes and relays.
3. **Relieving Bottlenecks.** This functionality includes increasing power flow, enhanced voltage support, providing and allowing the operation of the electrical system on a dynamic basis.
4. **Enabling a Self-healing System.** This functionality requires wide-scale deployment of electronic devices such as electronic circuit breakers so that new paths can be created dynamically in the grid in the event of an error. This, will then provide the integration of an advanced control architecture to enable a self-healing system.
5. **Enabling Enhanced Connectivity to Consumers.** Integration of a communications system that allows connectivity to the ultimate consumers. This enhancement creates new areas of functionalities that relate directly to electricity services (e.g. added billing information, real-time pricing, home security, appliance monitoring).

5.3.2 IEC 61850 and GOOSE

IEC 61850 is a standard for the design of electrical substation automation. It defines standardized communication between IEDs located not only within electric utility facilities, such as power plants and substations, but also outside these facilities such as storage systems and meters. IEC 61850 has been identified by the National Institute of Standards and Technology (NIST) as a cornerstone technology for field device communications and general device object data modeling [20].

IEC 61850 defines peer-to-peer communication mechanisms between IEDs using GOOSE messages (Generic Object Oriented System Event) over Ethernet. The use of sampled values from IEDs using GOOSE minimizes the use of copper wiring throughout the substation leading to significant benefits in cost savings, more compact substation designs, and advanced and more flexible automation systems [20].

The GOOSE is one of the most beneficial pieces of the IEC 61850 standard. It is a model to transmit sampled values in a fast way to multiple subscriber devices using multicast. GOOSE messages are designed to utilize high-speed Ethernet networks, achieving speeds equal to or faster than hardwired [20]. They are ideal for the messaging system of the smart grid because they are extremely fast, they define a data set that can contain electricity information and they can be encapsulated inside the current Internet application layer protocols (Chapter 2.2.6).

5.3.3 Phasor Measurement Units and GOOSE generation

Phasor measurement units (PMUs) are electronic devices that produce GOOSE messages with electricity measurements from the grid and publish them to a subscriber using a synchrophasor-based protocol such as IEC 61850-90-5 [20]. PMU data are truly synchronized which enables the monitoring of value changes in real time. Placing the PMUs in critical positions such as power substations can help in obtaining information about the system conditions of the power grid.

Figure 42: PMU device

PMUs are fundamental in the smart grid architecture because they can measure the electricity's frequency. Changes in the frequency can create a balancing system for the generation-demand equation. In Europe the line frequency of the electricity is 50Hz. The frequency is increasing when more electric power is generated than the power that is being consumed which results in electric energy wasting. If the frequency is dropping, it means that there is more demand than generation which results in power failures and blackouts. A PMU produces 30-60 updates per second which provides real-time monitoring of the grid.

5.3.4 Gateway for the GOOSE

The goal is to use the Raspberry Pi, which is the testbed's gateway, as a gateway for GOOSE messages. A PMU that produces GOOSE messages is connected via Ethernet to a microcontroller (mbed board is used) which receives the GOOSE message and removes unnecessary headers. Then, it sends the lightweight message to a Wiznet W5100 via SPI. The Wiznet is a small board with an Ethernet port that includes UDP and TCP libraries. It creates a UDP packet and adds the GOOSE message in the payload. After that, it provides the final GOOSE message encapsulated in a UPD packet available through an output Ethernet port.

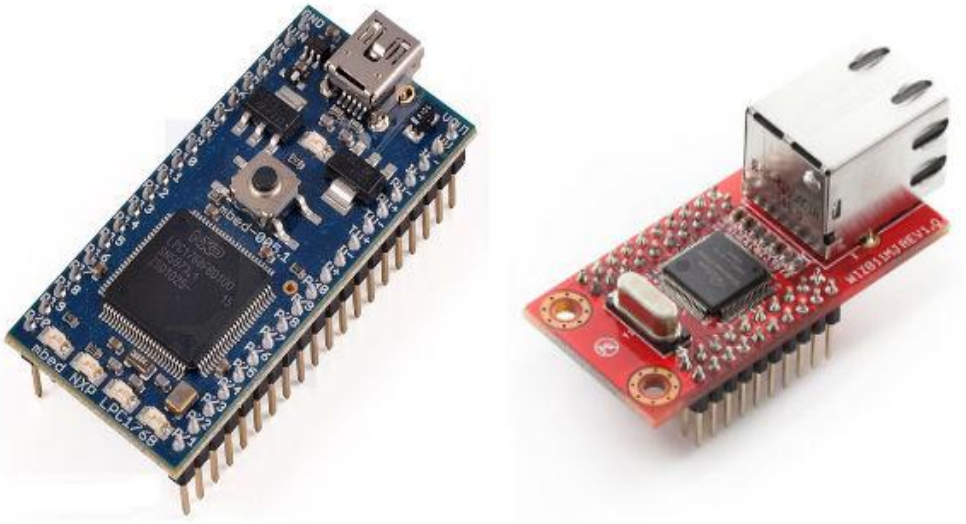

Figure 43: Mbed, Wiznet (left to right)

The process takes place in hardware microcontrollers because they handle calculations faster than software and the objective is to achieve real-time monitoring of GOOSE messages. The overall goal is to connect this Ethernet communication to the Raspberry Pi gateway and provide real-time monitoring, storing, visualizing and routing of GOOSE messages and thus of the power grid.

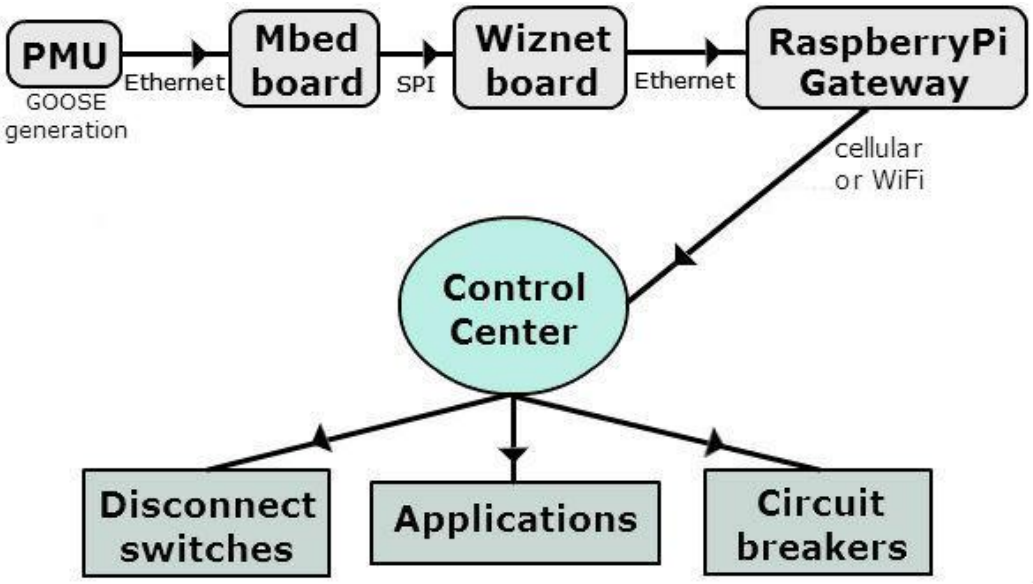

Figure 44: Gateway for the GOOSE

UDP is the preferred protocol to transfer GOOSE messages to the Internet because it is lightweight, fast and lacks retransmission mechanisms. To achieve real-time monitoring, new updated messages are more crucial than retransmitted old ones, thus retransmission mechanisms are avoided.

Use of public internet for exchanging GOOSE messages for the power grid is highly unlikely for security reasons. Instead The IoT approach of the smart grid suggests the use of private networks that exchange information over the Internet. This way the framework is safe from Internet threats, but at the same time uses the same technologies. Consequently even if the grid never goes over public Internet, the testbed can be used for simulations.

Chapter 6

Conclusions – Future Work

An emerging tendency for automation lies in the recent future. All the necessary tools to begin improving the quality of life in urban environments are currently available. After exploring the literature but also by conducting independent research on the Internet of things, we design a testbed that can cope with the demanding requirements of modern societies.

The panStamp project is selected for implementing the automation part and it proved to be very stable and surprisingly open to modifications. Gateway and host of the M2M servers is a Raspberry Pi which is ideal for a device that needs to run constantly due to its extremely low-power consumption. The Raspberry Pi also hosts the OpenRemote controller which is an excellent platform for those who want to use a mobile application for automation without writing a single line of code. The gateway connects the sensor network to the IoT clouds that provide an online monitor/control user interface available through the Internet and also serve as nodes between M2M servers and end-user applications.

Two representative applications of the IoT are designed for smart phones to complete an End-to-End Machine to Machine communication network, Smart Parking and Geo Fencing. The former is an information system that uses sensors to detect cars in parking spots and informs the user of their availability. The latter is a background process capable of performing automated tasks (e.g. opening a door) based on the geographical position of the user. Both applications, even if different to each other, work surprisingly well, which proves the flexibility and robustness of the testbed.

In addition the current electric power grid is described, flaws and inabilities are presented along with a way to connect the power distribution system to the implemented IoT testbed for further research on the Smart Grid.

We named the testbed SMARTWORLD because it is an innovative platform that can be used for any final application in mind and face technological and societal challenges.

Next step would be to deploy the SMARTWORLD testbed to a physical area and obtain a realistic picture of its services. The end-device network currently uses an application layer protocol called SWAP which can be implemented on top of other protocols for wired or wireless communication in order to present diverse data in a homogeneous way. In addition to REST services, the gateway can use CoAP and WebSocket to exchange information with the IoT platforms and the same protocols can also be used to route GOOSE messages for further research on the emerging smart grid. There are already available modules for connecting the Raspberry Pi to cellular Internet and convert the fixed end-device network to a mobile network. The cellular modules enable even LTE Internet connection which is preferred for the real-time updates of the smart grid.

The online IoT clouds that are currently connected to the testbed are ThingSpeak and OpenSense. The source code of ThingSpeak can be downloaded and deployed in a private server in order to free the testbed from commercial cloud dependency and create a fully autonomous development environment.

References

[1] Adam Dunkels, (2011), The ContikiMAC Radio Duty Cycling Protocol, SICS Technical Report T2011:13, ISSN 1100-3154, December 2011.

[2] Andreas Kamilaris, (2012), Enabling smart homes using web technologies, Doctroral thesis, University of Cyprus, December 2012.

[3] Android official website, https://source.android.com/, retrieved 28 April 2014.

[4] Arduino official website, http://arduino.cc/en/Guide/Environment, retrieved 28 April 2014.

[5] Axeda official website, http://www.axeda.com/, retrieved 28 April 2014.

[6] Buonaccorsi N, (2012), Cicconetti C,Mambrini R, Podias N, Russell P, ETSI M2M release 1 demonstration, World of Wireless Mobile and Multimedia Networks (WoWMoM), 2012 IEEE International Symposium, 25-28 June 2012.

[7] Cisco press release, (2008), Cisco, Atmel and the Swedish Institute of Computer Science (SICS) Collaborate to Support a Future Where Any Device Can Be Connected to the Internet, http://newsroom.cisco.com/dlls/2008/prod_101408e.html, retrieved 28 April 2014.

[8] Clark W. Gellings, (2009), The Smart Grid: Enabling Energy Efficiency and Demand Response, Fairmont Press, 2009.

[9] CoAP, Wikipedia site, http://en.wikipedia.org/wiki/Constrained_Application_Protocol, retrieved 28 April 2014.

[10] Contiki, Wikipedia website, http://en.wikipedia.org/wiki/Contiki, retrieved 28 April 2014.

[11] Contiki official website, http://www.contiki-os.org/community.html, retrieved 28 April 2014.

[12] Contiki official website, http://www.contiki-os.org/, retrieved 28 April 2014.

[13] Eli T. Iceman, (2012), Power grid operations, Dog Ear Publishing, September 2012.

[14] Fielding Roy Thomas, (2000), Architectural Styles and the Design of Network-based Software Architectures, Doctoral dissertation, Irvine University of California.

[15] GOOGLE Maps Android API, official site, get started, https://developers.google.com/maps/documentation/android/start#getting_the_google_maps_android_api_v2, retrieved 28 April 2014.

[16] GOOGLE Maps Android API, official site, intro, https://developers.google.com/maps/documentation/android/intro, retrieved 28 April 2014.

[17] Ian F. Akyildiz, (2002) W. Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. Wireless sensor networks: a survey. Computer Networks, 38(4):393–422.

[18] IBM official website, MQ Telemetry Transport (MQTT) V3.1 Protocol Specification, http://www.ibm.com/developerworks/webservices/library/ws-mqtt/index.html, retrieved 28 April 2014.

[19] IBM official website, RESTful Web services: The basics, http://www.ibm.com/developerworks/webservices/library/ws-restful/, retrieved 28 April 2014.

[20] John D. McDonald, (2012), Electric power substations engineering, CRC Press.

[21] Kevin Ashton, (2009), That 'Internet of Things' Thing, RFID Journal website, http://www.rfidjournal.com/articles/view?4986, retrieved 28 April 2014.

[22] M-bus official website, http://www.m-bus.com/info/mbuse.php, retrieved 28 April 2014.

[23] Modbus official website, http://www.modbus.org/faq.php, retrieved 28 April 2014.

[24] Miao Wu, (2010), Ting-Jie, Lu Fei-Yang, Ling Jing, Sun Hui-Ying Du. Research on the architecture of Internet of Things, Advanced Computer Theory and Engineering (ICACTE), V5-484, August 2010.

[25] MQTT manual, http://mosquitto.org/man/mqtt-7.html, retrieved 28 April 2014.

[26] MQTT official website, http://mqtt.org/faq, retrieved 28 April 2014.

[27] Mulligan Geoff, (2007), "The 6LoWPAN architecture", EmNets '07: Proceedings of the 4th workshop on Embedded networked sensors, ACM.

[28] NOOBS, Raspberry Pi official website, http://www.raspberrypi.org/help/noobs-setup/, retrieved 28 April 2014.

[29] OpenRemote official website, http://www.openremote.org/, retrieved 28 April 2014.

[30] OpenRemote source code, http://sourceforge.net/projects/openremote/, retrieved 28 April 2014.

[31] OpenSense official website, http://open.sen.se/, retrieved 28 April 2014.

[32] PanStamp official website, base board, http://www.panstamp.com/products/battery-board, retrieved 28 April 2014.

[33] PanStamp official website, http://www.panstamp.com/home, retrieved 28 April 2014.

[34] PanStamp official website, panStick, http://www.panstamp.com/products/panstick-1-2, retrieved 28 April 2014.

[35] PanStamp official website, raspberry Pi shield, http://www.panstamp.com/products/rpishield, retrieved 28 April 2014.

[36] PanStamp source code, http://code.google.com/p/panstamp/source/checkout, retrieved 28 April 2014.

[37] PanStamp technical details, http://code.google.com/p/panstamp/, retrieved 28 April 2014.

[38] PanStamp technical details, lagarto MAX, https://code.google.com/p/panstamp/wiki/LagartoMAX, retrieved 28 April 2014.

[39] PanStamp technical details, lagarto servers, http://code.google.com/p/panstamp/wiki/lagarto, retrieved 28 April 2014.

[40] PanStamp technical details, lagarto SWAP, https://code.google.com/p/panstamp/wiki/LagartoSWAP,retrieved 28 April 2014.

[41] PanStamp technical details, specifications, http://code.google.com/p/panstamp/wiki/panStamp, retrieved 28 April 2014.

[42] PanStamp technical details, SWAP, http://code.google.com/p/panstamp/wiki/SWAP, retrieved 28 April 2014.

[43] Processing official website, http://www.processing.org/, retrieved 28 April 2014.

[44] Raspbian OS official website, http://www.raspbian.org/RaspbianFAQ, retrieved 28 April 2014.

[45] Raspberry Pi official website, help and FAQ http://www.raspberrypi.org/help/what-is-a-raspberry-pi/, http://www.raspberrypi.org/help/faqs/ retrieved 28 April 2014.

[46] Raspberry Pi, Wikipedia, http://en.wikipedia.org/wiki/Raspberry_Pi, retrieved 28 April 2014.

[47] Related work, IoT-Lab testbed, https://www.iot-lab.info/, retrieved 28 April 2014.

[48] Related work, Oulou testbed, http://www.panoulu.net/, retrieved 28 April 2014.

[49] Related work, W-iLab.t testbed, http://www.iminds.be/en/succeed-with-digital-research/technical-testing, retrieved 28 April 2014.

[50] REST, Wikipedia website, http://en.wikipedia.org/wiki/Representational_state_transfer, retrieved 28 April 2014.

[51] Steven W. Blume, (2007), Electric power system basics, IEEE Press.

[52] ThingSpeak official website, https://thingspeak.com/, retrieved 28 April 2014.

[53] ThingSpeak source code, https://github.com/iobridge/ThingSpeak, retrieved 28 April 2014.

[54] TinyOS official website, http://tinyos.stanford.edu/tinyos-wiki/index.php/FAQ, retrieved 28 April 2014.

[55] TinyOS official website, http://tinyos.stanford.edu/tinyos-wiki/index.php/Platform_Hardware, retrieved 28 April 2014.

[56] UART tutorial, http://www.raspberry-projects.com/pi/pi-operating-systems/raspbian/io-pins-raspbian/uart-pins, retrieved 28 April 2014.

[57] WebSocket, Wikipedia website, http://en.wikipedia.org/wiki/WebSocket, retrieved 28 April 2014.

[58] Zach Shelby, (2011), Carsten Bormann, "6LoWPAN: The wireless embedded Internet - Part 1: Why 6LoWPAN?" EE Times, 23 May 2011.

[59] ZeroMQ official presentation, ZeroMQ is the answer, http://www.youtube.com/watch?v=v6AGUeZOVSU, retrieved 28 April 2014.

[60] ZeroMQ, Wikipedia website, http://en.wikipedia.org/wiki/%C3%98MQ, retrieved 28 April 2014.

[61] Zigbee official website, http://www.zigbee.org/Standards/Overview.aspx, retrieved 28 April 2014.

[62] Zigbee official website, http://www.zigbee.org/Standards/ZigBeeBuildingAutomation/Features.aspx, retrieved 28 April 2014.

[63] Zigbee, Wikipedia website, http://en.wikipedia.org/wiki/Zigbee, retrieved 28 April 2014.

[64] 6loWPAN, Wikipedia website, http://en.wikipedia.org/wiki/6LoWPAN, retrieved 28 April 2014.