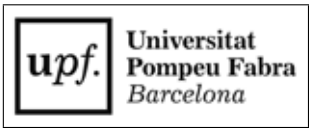
Open Sensor Network

Alejandro Andreu Isábal

TFG UPF / YEAR 2013

TFG SUPERVISOR:
Jaume Barceló Vicens
DEPARTMENT:
Departament de Tecnologies de la Informació i les Comunicacions



Dedicated to my parents, who always believed in me, and who never hesitated on answering my infancy *"why does this work?"* questions.

Acknowledgments

Foremost, I would like to thank my project supervisor, Jaume Barceló for his patience, motivation and guidelines. His pointers always motivated me to explore unknown horizons and I ended up learning a huge amount of knowledge.

I also strongly thank the open source community, which has been as well so motivating and helpful, and never has stopped to help. Without it, I probably could never have finished my work.

Thanks to the other fellows of BuB4EU Jorge Beltrán, Nacho Justel and Fernando Gros for accompanying me in this adventure.

Finally, my sincere gratitude as well to the European Commission through the project Commons for Europe (CIP-ICT-PSP-2011-5-297191) which was the initial seed of this work.

Abstract

Sensor networks are increasingly being used for environmental monitoring to anticipate or react to certain events. Despite many initiatives seemingly encouraging and being based on the use of sensor networks, only a few are really open. In this thesis we design, prototype and test a wireless sensor network using open hardware and open source software. Arduino and ZigBee are the main bases for the sensor nodes, and a Raspberry Pi is used as a data sink. Gathered information is then uploaded to cloud platforms, making it available to developers which can bring new ideas to life. The proposed solution is an economical, flexible and do-it-yourself alternative to those willing to collect and share sensory data.

Resum

Les xarxes de sensors són cada cop més utilitzades per la monitorització mediambiental per tal d'anticipar-se o reaccionar davant determinats esdeveniments. Malgrat que moltes iniciatives estan basades en l'ús de les xarxes de sensors, només unes poques són realment obertes. En aquesta tesis es realitza el disseny, el prototip i el testing d'una xarxa de sensors sense fils utilitzant hardware obert i programari de codi obert. L'Arduino i ZigBee són les bases principals pels nodes de sensors, i una Raspberry Pi s'utilitza com a receptor de dades. La informació recopilada es puja a les plataformes en el núvol, de manera que està disponible pels desenvolupadors que poden aportar noves idees a la vida. La solució proposada és una alternativa econòmica, flexible i "do-it-yourself" per a aquells que estan disposats a recollir i compartir dades de sensors.

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

During the last years we, the Internet users, just had one chance to know how things are managed, from top to bottom. That is, telecom operators reserve some resources (optic fiber, certain bandwidth, etc.) for each one of their clients and charge them for this service. In a top-down approach the consumer remains completely passive and has to reluctantly accept what the telco dictates. Now, a new model is willing to turn this trend upside down.

This new way to do things is called *bottom-up broadband*, and is also how this project is posed. The very same users that were before passive will become very active, helping not only by designing the network but also by deploying it and maintaining it, thus participating in every step of the system lifecycle. Hence, without a central authority the usufructuaries are the only ones that conform this kind of networks.

Bottom-up broadband —BuB from now on— schemes have several important advantages over those that follow a conventional top-down approach, such as: easier and faster setup due to the lack of a central authority (as it happens in peer-to-peer networks), it can be adapted to anyone's needs since they are the caretakers of the system, and could also become the solution to those that live in an area that is not economically attractive to regular ISPs[Oliver et al., 2010].

As for disadvantages, a BuB network creation can be very time consuming, since users participate in every single step of this endeavor[Barcelo et al.,].

Sensor networks are very important nowadays and its objective is to gather data. "*Data itself isn't good nor bad. Data just represents the surrounding reality. The more data we may access, the more accurate model we may create of the reality, thereby also define our actions in ways that are maximally beneficial to our aims*"[paraZite, 2013].

But, does it make sense building such a system under a BuB model? The answer is yes, it does. As it happened with traditional telecom operators, the information that is obtained through sensor networks is usually kept by the agencies

that own them without even making a public API[1] to "play" with this data.

Therefore the main goal of this project is to design and deploy a sensor network that gathers real-time information and that enables developers to create applications that will ultimatelly help the citizenship improve their daily lives. Intrinsically this can be divided in more specific objectives, such as:

- Allow citizens, individuals belonging an organization or even enterprises to connect scattered sensor nodes.
- Collect different kinds of information and transmit it to the Internet.
- Samples[2] must be gathered often enough to be almost real-time.
- Use of open technologies to allow easier replication and modification as well as reducing final costs.
- The project shall become a tool so anyone that needs or wants to deploy a sensor network can do it as soon as possible.

1.1 Structure of the thesis

Chapter 2 takes a look at the state of the art. That is, why are sensor network important nowadays and what has been done until now regarding commercial and open solutions.

Chapter 3 addresses what technologies have been used to perform this endeavor. A brief description about each element is attached so the reader can replicate more easily the network and have some details at first glance.

Chapter 4 focuses on the way this pilot has been completed. That is, the methodology that has been followed, as well as how problems have been confronted.

In Chapter 5 we can see how does each type of node work along with schematics and also how programs and scripts work (this last part through flowcharts).

Chapter 6 shows the results on having worked on this project.

Chapter 7 presents what challenges could be confronted in the future as well as some conclusions regarding the obtained results.

[1] An API defines how different pieces of software interact with each other.

[2] Values obtained by sensors.

Chapter 2

STATE OF THE ART

Sensor networks, as many other technological advances, see their origin on military research. They date back to the early 60's during the Cold War, when the United States deployed an underwater system to detect Soviet submarines called SOSUS (sound surveillance system). However, it is not until the beginnings of the 21st century that more applications were found beyond warfare. The main causes for that to happen is that the cost and the size of the components have drastically decreased.

Another crucial factor for this to happen is that new sets of wireless standards did see the light. On one hand we have IEEE 802.15 that allows to create low bitrate wireless networks called WPANs[1] which incredibly extend battery lifetimes. On the other hand IEEE 802.11 enables wireless communications to experiment similar bitrates to those obtained in a wired network.

One of the motivations to develop an open sensor network is that normally those who are the owners of the information keep it to themselves, a situation on which nothing is given back to society. Therefore, it is important to share all gathered information.

Deploying sensor networks significantly contributes to the growth of smart cities ICT[2] structures which, at the same time make social, cultural and urban development thrive[Caragliu et al., 2009][Hollands, 2008].

There are already some initiatives that are based on wireless sensor networks. The word "initiative" is not intended to refer just to companies but also to organizations and individuals, from city halls that want to improve the daily lives of their citizens to people that want to share the environmental conditions from their balconies.

At the time of writing, we can distinguish between two main kinds of sensor

[1] Wireless personal area networks, defined in IEEE 802.15, refer to wireless networks where devices are just a few meters away from each other.

[2] Information and communications technology.

networks: company-driven and community-driven networks, depending on who shapes the system.

These networks can generate a big amount of data over time, creating the necessity of storing this information somewhere. This information has to be always available for further usage. There are already some websites with the only objective of storing this information and providing useful visualization tools.

2.1 Company-driven sensor networks

These kind of systems work normally in an opaque or translucent way but with the advantage that they have very clear objectives. Also, they usually have more resources, as making money is the main motivation.

Telcos have recently shown an interest for this market. Namely, in May 2013 the IMC[3] has seen the light, where Deutsche Telekom is the protagonist member. This organization aims to help companies adopt M2M technologies as well as making new policies concerning this branch of knowledge.

2.1.1 Libelium

Having its headquarters in Zaragoza (Spain), this is one of the biggest companies in the world built around wireless sensor networks. Libelium[4] offers the mechanisms and tools to deploy and build systems around the Internet of Things, smart cities and M2M[5] communications.

Figure 2.1: Libelium logo.

The majority of the products they sell are focused on one specific application, such as waste management, structural health, etc. These systems are intended to be bought by system integrators for end users. However, they also offer the so-called "Waspmote", which is a sensor device for developers that can be freely

[3] International M2M Council. More information is available at `http://www.im2mc.org`.
[4] `http://libelium.com`
[5] Machine-to-machine communications are established between two autonomous devices.

customized and reprogrammed, since it is an open hardware product. For their non-open products they provide a very complete API along with an excellent documentation.

Their products are being widely used across more than 75 countries and they are definitely one of the leaders of the wireless sensor network industry.

2.2 Community-led sensor networks

Projects that are driven by the community give all the decision making power as well as resources to the community. The individuals that conform the community are very passionate about what they do and the workflow is highly transparent.

It is worth saying that the initiatives presented in this section are not sensor networks per se, but *sensing networks*. This is because communication does not take place between sensor nodes but between a sensor node and a central server that processes and represents data.

2.2.1 Air Quality Egg (AQE)

This is a sensing network that aims, as its own name indicates, to measure the air quality. This is done through NO_2 and CO levels.

The users are supposed to connect their "eggs" (or base stations) to their local network via an Ethernet inferface. Then, a bunch of outdoor sensors are placed outside and communicate their readings to the base station —as it can be seen in figure 2.2— wirelessly through a radio frequency transmitter. Finally the data is sent in real time to Xively.

It is worth mentioning that the AQE project is completely open, hence anyone can improve the platform as well as building his own egg from scratch without having to actually buy one.

All the information related to the hardware, software and sensor calibration can be found in their wiki[6].

2.2.2 Smart Citizen

Originally designed in Barcelona, Smart Citizen[7] is a very young project (still in beta stage) that intends to create the biggest community around social sensing.

[6] `http://airqualityegg.wikispaces.org`
[7] `http://smartcitizen.me`

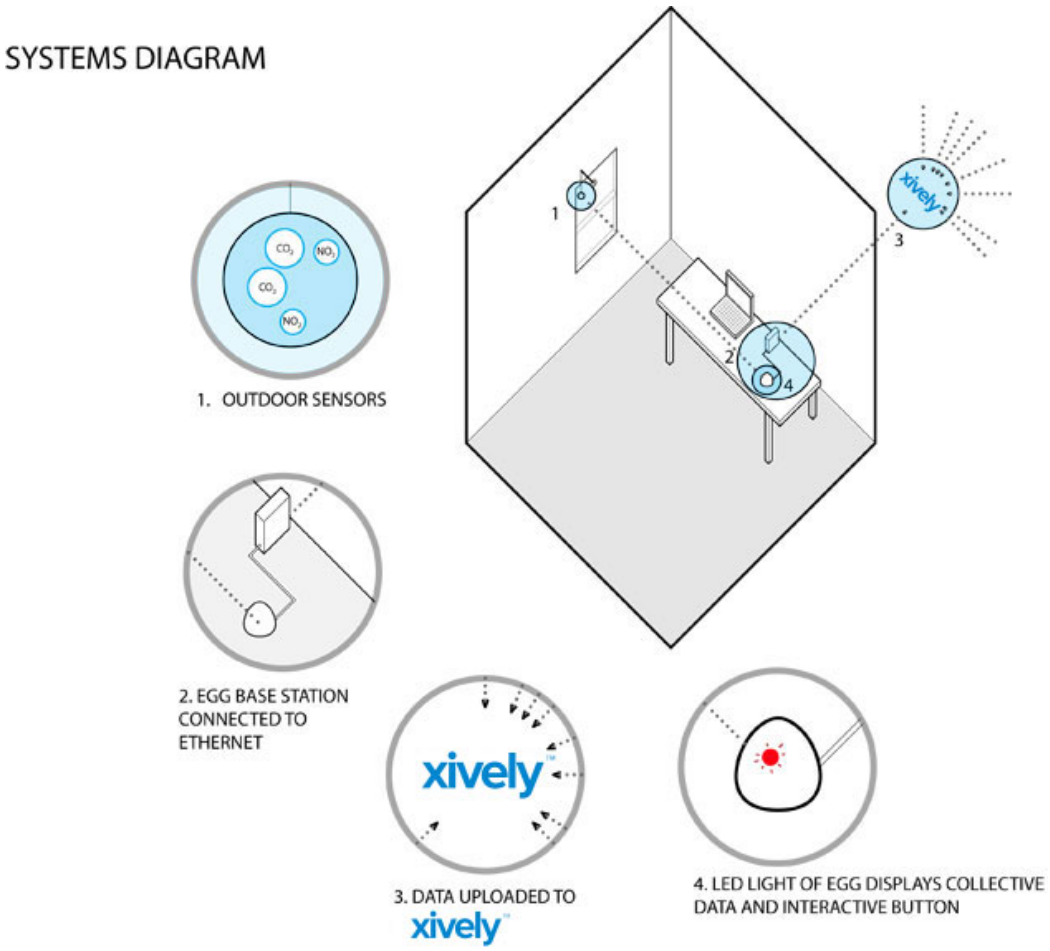


Figure 2.2: AQE typical scenario. Picture taken from the AQE official website (`http://airqualityegg.com`)

It was initially crowdfounded in 2012 through a Goteo campaign[8] and they have succesfully launched another crowdfunding campaign on Kickstarter.

The Smart Citizen platform allows its users to precisely geolocate their data and see other users' information. There is also a very big emphasis in data sociability, since every value or datastream[9] can be shared through any social networking site or inside the same web application.

Openness is as well one of their main values, since every piece of code —including the very own website— and hardware schematics is open source licensed.

2.3 Open data services

All gathered information must be stored in some place, and this is where open data portals —also called Internet of Things clouds— come into play.

These websites provide users with an open API so they can upload new values, create new feeds[10], retrieve the data and even create customized triggers, such as sending a push notification to a smartphone or even "tweeting" something. This way, we cannot only sense but *react* to certain kinds of events.

Because data by itself is usually worthless, one of their most important features is the availability of data visualization tools. They allow us to easily detect patterns and also correlate certain factors.

Good examples of these sites are, as mentioned before, Xively and Sen.se[11]. Pompeu Fabra University has its own cloud to store sensory data as well, called Open Sensor Network Web Application[12]. All these options are free to use and very easy to interact with, since they provide us with a RESTful API[13].

In case we want to host our own cloud for the Internet of Things, there is also a great solution called Nimbits[14]. It is open source software and anyone can install it in its own server.

[8] Goteo is a Spanish social network that helps crowdfund open projects that result in a society improvement. This campaign can still be visited at `http://goteo.org/project/smart-citizen-sensores-ciudadanos`

[9] Set of values that represent an individual sensor over time.

[10] Representation of an environment. A feed can be, for instance, a museum hall where presence and noise levels are measured.

[11] `http://open.sen.se/`

[12] Available at `http://opencities.net/`.

[13] Web API that works with the regular HTTP methods. That is, `GET`, `PUT`, `POST` and `DELETE`.

[14] `http://nimbits.com`

Chapter 3

TECHNOLOGIES

Three essential blocks form a sensor network. Namely sensors, processors and communication devices[Chong and Kumar, 2003]. In the next sections all of them will be explained and in Chapter 5 I will show how these are related.

3.1 Active sensors

We now live in a world where we hear a lot the word "sensor", but what is exactly a sensor? A sensor transforms physical measurements into eletromagnetic signals that can be understood by an external device. This definition might seem (and is) quite simple, but complexity resides on calibration and coming up with actual useful applications.

Since every sensor from the same family is equal in terms of design but different in reality (due to small random variations during the fabrication process) output has to be adjusted to agree with a given standard. When it comes to applications, only imagination poses a limit. RFID tags can be used to determine whether a book is on the right spot in a library or not, or with a very intense light beam we can detect how the blood flows through a vein therefore successfully sensing heart rate. These are just two imaginative uses for nowadays sensors.

As for types of sensors, we have simple/complex and passive/active sensors. Simple ones just provide us with very basic information —a binary state, 1 or 0—. Cameras, on the other hand, are a perfect example of complex sensors.

Active sensors emit a signal to the environment and then measure the response, while passive just measure a factor from the surroundings.

Also, the output of a sensor can be expressed in two formats. Digital and analog, depending on how data transmission is done. Digital means that the output is represented in the form of bits, thus requiring some amount of computational power to "interpret" the results. Analog sensors usually act as a variable resistance

depending on some factor, so very little processing power is needed.

To complete this project, only environmental factors have been measured since those have been tested over and over for the last years and they serve as a proof of concept for this network. Also, sensors are not the main focus of the pilot but the tools to deploy such a network.

3.1.1 Aosong DHT22

The DHT22 is a low cost, humidity and temperature measuring digital sensor designed by Aosong Electronics, a Chinese corporation[1].

The output —although digital— uses a special single-wire protocol[2] format that is precisely described in the datasheet. Luckily there already are some library implementations to work with the DHT22. Then, this device will only work with platforms that allow digital input, such Arduino.

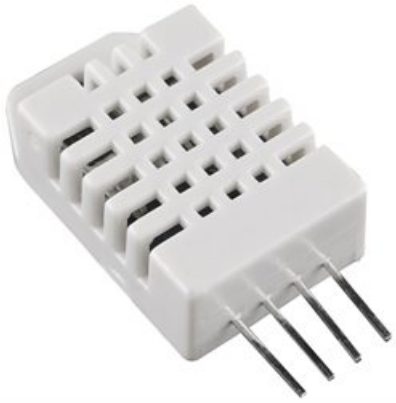


Figure 3.1: DHT22 humidity and temperature sensor.

3.1.2 Emartee Mini Sound Sensor

Manufactured by Emartee, can also be found by the name of "Emartee part number 4021", and can be used to measure noise levels among other uses. Essentially, it consists on a microphone with a built-in amplifier onto a breakout board. This last feature can be useful to work directly with perfboards or breadboards[3].

[1] Its datasheet can be found at: `http://www.adafruit.com/datasheets/DHT22.pdf`

[2] All useful information is transmitted through just one of its pins.

[3] Both are construction bases for rapid prototyping of electronic circuits.

The output signal is analog and is increased (amplified) by some factor that allows an Arduino or any device with analog input pins to detect this signal easily[Gertz and Di Justo, 2012]. Its operating voltage is 5V.

3.1.3 Sharp GP2Y1010AU0F

This is an inexpensive optical dust sensor, used to measure air quality. It is made out of an infrared emitting diode which, with a well positioned phototransistor can measure the reflected IR rays thus detecting dust levels in the air[Sharp, 2006]. This device, which can be powered with up to 7V, gives an output voltage proportional to dust density in the air. Some of its applications are air monitoring and air conditioning.

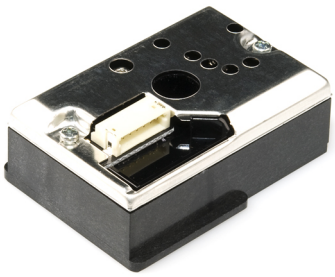


Figure 3.2: Sharp GP2Y1010AU0F optical dust sensor.

Surprisingly, this detector, which is priced at the time of writing about $12, gives very precise results, similar to those offered by an expensive laser particle counter[Nafis, 2012].

3.1.4 LM35

The LM35 is an analog sensor designed by Texas Instruments that precisely measures temperature[4]. Its output is linear to temperature —in Celsius degrees— which means that this sensory value will not have to be processed later.

[4]Datasheet: `http://www.ti.com/lit/ds/symlink/lm35.pdf`

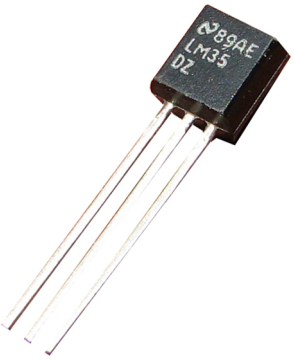


Figure 3.3: TI LM35 temperature sensor.

3.2 Digi XBee® Wireless RF Module

These radio modules are based on the IEEE 802.15.4 standard and provide inexpensive, low power, low rate communication. They mainly use ISM[5] bands.

Figure 3.4: Digi XBee® Wireless RF Module

There are two versions of these modules, chronologically named “Series 1”

[5]ISM refers to industrial, scientific and medical radio frequency bands. These are internationally reserved slices of the spectrum which are intended to be used not for communications but with industrial, scientific or medical purposes in mind. Examples of these bands are 868MHz, 2.4GHz, 5.725GHz, etc.

and “Series 2”. The older version implements the previously mentioned IEEE 802.15.4 standard which enables the network to be configured in point to point topologies. This standard specifies the physical and media access control layers for WPANs.

On the other hand however, the latter implements a standard specification called *ZigBee*[6]. It specifies the upper layer of the protocol stack. That is, network and application layers. Although more complex, ZigBee provides mesh networking capabilities which can be a key feature in some sensor networks.

Despite their size these devices offer us many interesting features, such as 128-bit encryption, over-the-air configuration and several pins that enable the XBee to read analog values as well as working with digital input and output[International, 2007].

This is why for the sake of this project “Series 2” has been chosen. It is worth saying that each of the two versions can transmit with different power levels thus varying the effective communication range[Faludi, 2010]. More detailed information can be found in table 3.1.

Also, any XBee® device has two operational modes, namely transparent and API. In transparent mode —also known as AT mode—, the device only relays serial data until it reaches the network sink. This mode is simple and “universal”, since a connection can be established with every device that speaks and understands serial. However, neither data reception or integrity are assured (specially when working with the popular 2.4GHz ISM band, which is highly saturated) and eavesdropping can be a real problem since encryption cannot be used.

On the other hand, API mode is slightly more complex than transparent mode, since information is encapsulated in packets. These are the main features of this operational mode:

- When the sink receives a packet, it immediately transmits an ACK[7] packet. If such packet is not received then transmitting radio will retry sending it.
- Radios can be re-configured dinamically over the air.
- Checksums that verify that no errors have been introduced. In other words, it checks if the received information is the same than the one that was originally transmitted.
- Encryption (using a symmetric key algorithm) can be enabled, either using one pre-established key or getting one from the network sink.
- I/O samples, which enable us to use all the DIO[8] pins that the module has.

Version	Power	Indoor range	Line of sight range
Series 1	1 mW	30 m	100 m
Series 1 PRO	63 mW	90 m	1600 m
Series 2	2 mW	40 m	120 m
Series 2 PRO	63 mW	90 m	1500 m

Table 3.1: Comparison of different versions of XBee®.

Also, if extra range is needed —up to 40 Km in line of sight— there are also XBee® devices that transmit in lower ISM bands —900 and 868 MHz—. However, when transmitting in these frequencies neither ZigBee nor IEEE 802.15.4 can be used. DigiMesh™ networking protocol is the only option and it is property of Digi International Inc, which has other features.

3.3 Arduino

Arduino is the leading protopying platform nowadays. It is completely open source including the schematics of the hardware itself, which is a single-board microcontroller. Anyone can program the board through a programming language very similar to C/C++ and based on Wiring[9]. To upload a sketch —a program— to the microcontroller they also have developed an Arduino IDE based on the Processing IDE[10].

The amount of projects related to this platform is incredibly big, and it has gained huge popularity amongst designers, hackers, programmers and hobbyists these past years. It offers several advantages over similar devices, because it is really cheap, cross-platform and has every benefit inherent to the open source initiative. Also, like other open projects Arduino comes in many "flavors" depending on the characteristics of the project.

As it can be seen in figure 3.5, the board has many input/output pins that are compatible with analog and digital values. It's not just that but also it can establish a serial communication with a computer so interaction between programs and the platform can take place.

Arduino UNO is the "flavour" that has been chosen to perform this project,

[6]Protocol stack developed and maintained by the ZigBee Alliance. More information can be found at `http://zigbee.org`

[7]Acknowledgement packets are used to indicate that the transmission was successful.

[8]Although DIO stands for digital input/output, some of them can handle continous values through the ADC.

[9]`http://wiring.org.co`

[10]More information about Processing and its IDE can be found at `htpp://processing.org`

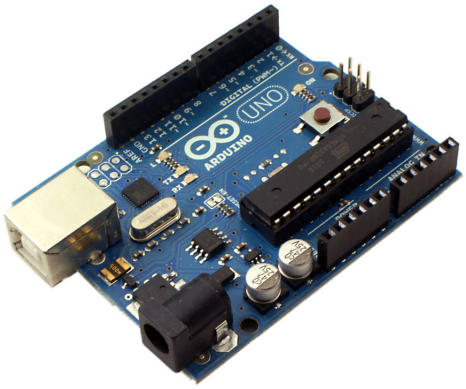


Figure 3.5: Arduino UNO prototyping platform.

since it is cheap and also the most common. That means all shields[11] work by default on it and also has the biggest community. In particular, this model has the following features (table 3.2):

3.3.1 External libraries

In order to interact with the XBee module I used the `xbee-arduino` library, which is a C/C++ library that enables an Arduino to send and receive information via an XBee® radio. It was originally developed by Andrew Rapp and is currently hosted on Google Code[12].

3.4 Raspberry Pi

This device is an inexpensive GNU/Linux box that follows an ARM architecture and acts as the sink of the network. There are two models of this device: one that has 256MB of RAM (known as "model A"), and another one that has 512MB of such memory ("model B"). As for the processor it utilizes a 700MHz Broadcom SoC[13]. The operating system is directly loaded from an SD.

[11] A shield is another board plugged on top of the Arduino to extend its functionalities (for instance Ethernet, Wi-Fi, SD cards, etc.).

[12] More information can be found at `https://code.google.com/p/xbee-arduino/`

[13] System on a chip are integrated circuits that collect all the necessary modules of a traditional computer in one single chip.

Feature	Value
Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz

Table 3.2: Characteristics of the Arduino UNO. Taken from the official Arduino website (`http://arduino.cc`).

Currently it supports many popular distributions, such as: Arch Linux, Raspbian (a fork of Debian specifically designed to run on the Raspberry Pi), Gentoo, NetBSD, etc.

This is an optional part of the network since an XBee can be attached to any device that speaks serial. Nonetheless it is small and low-power, providing the necessary versatility for this kind of systems.

3.5 D-Link DWA-123

The Raspberry Pi, as stated before, has an ethernet interface, but in case such option is not available, a Wi-Fi dongle is the best solution. This model in particular works out-of-the box with some Raspberry Pi GNU/Linux distributions. Also, it is cheap and supports IEEE 802.11b/g/n. This device will be the bridge between the XBee network and the Internet.

3.6 Arch Linux ARM

The project originally started as a port of the popular distribution Arch Linux[14], compiled for ARM devices. The main feature of this "distro" is *simplicity*.

Arch Linux ARM follows, as its predecessor does, the KISS ("keep it simple, stupid") principle, which means it just provides a working and minimalist system

[14] `http://archlinux.org`

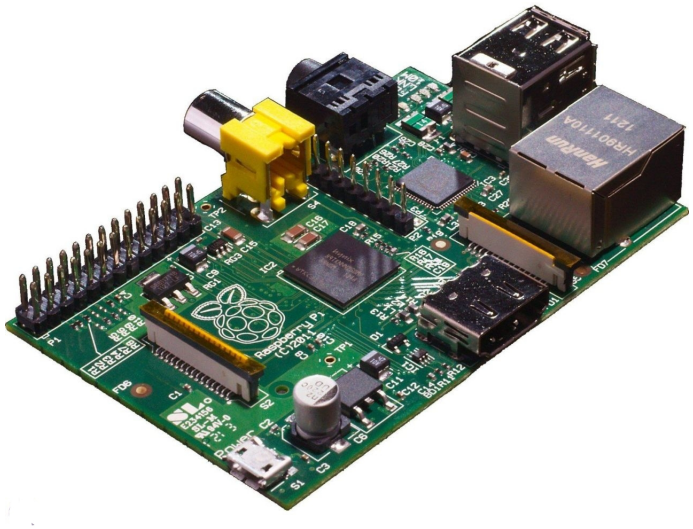


Figure 3.6: A Raspberry Pi.

Figure 3.7: Arch Linux ARM logo.

to work with. That is, no unnecessary packages are installed by default, giving the user complete control of its operative system. For instance, no GUI is available by default and everything must be done through the terminal (at least in the first place). This means the Raspberry Pi will be running a lightweight operative system, therefore leaving more resources to process all the data.

3.7 Python

Python is a general-purpose, high-level scripting programming language that first saw the light of day in 1991, originally designed by Guido van Rossum. It is an open source language that has more than one implementation. The default implementation —and used in this project— is CPython, which is written in C.

There are two main versions of this programming language, 2.x and 3.x which are not compatible. Version 2.x will be the one used to complete the project since more libraries and modules use Python 2.x.

The reasons why I chose Python instead of other object-oriented programming languages are simple. Python is simple and it just gets things done. Also, with all its built-in functions is very suitable for rapid prototyping, which is very similar to the approach taken in the completion of this project.

Figure 3.8: Python logo.

This project does not utilize solely the standard libraries but also some additional ones, such as:

- `pyserial`[15] — This library enables Python to establish serial connections (more specifically, following the RS-232 standard) with all kinds of devices. In this case, it allows us to receive data from an XBee® module.
- `python-xbee`[16] — Written by Paul Malmsten, this library allows Python to work with XBee API serial information. It has two main operational

[15] `http://pyserial.sourceforge.net/`
[16] `https://code.google.com/p/python-xbee/`

modes, synchronous and asynchronous. That is, continuously receiving information from an XBee® (and blocking the whole program until it finishes) or spawning a new background thread every time a new packet arrives, respectively.

- `python-requests`[17] — This is an HTTP library that, according to its creator, is written "for human beings". In other words, HTTP requests made easy. This module will be used to upload sensory data to the Internet

[17] `http://python-requests.org/`

Chapter 4

METHODOLOGY

The first step I took to complete this project was having a deep look at the state of the art. There are a lot of sensor network designs and some are as well open sourced, but the majority of them require either advanced knowledge on PCB fabrication or are focused on just one particular area (they aim to solve just one problem). Thus developing a system which is multipurpose and uses well-known technologies for rapid deployment are some of the key requirements that this network should meet (apart from the initial objectives).

Once all initial requirements are identified I had to choose one appropriate life cycle for the project. The pilot scope was not strictly constrained thus changes shall be handled in some way. Consequently, I chose an *agile* approach.

Agile management is a special case of iterative management. More specifically, it driven by changes[Institute, 2011]. Development of small modules is the usual thing, with deadlines every two or four weeks. Also, stakeholders are highly involved which is very related to the approach we followed all the components of BuB4EU. Each month there was a scheduled workshop where every participant informed the rest of the team about his last advancements. This method is very useful for getting constant feedback hence improving the overall quality of the project.

When problems emerged we had an available mailing list[1]. There, each participant can propose whatever he/she wants, but also ask questions and await for answers. This way, we have achieved a solid level of collaboration.

At the same time, the pilot followed an open development model, since it was —and still is— available on GitHub from the beginning[2]. I decided to go with a complete open model because this way I give back something to the community.

[1]Hosted in Guifi.net servers, accessible by entering `https://llistes.guifi.net/sympa/arc/bub`.

[2]The repository can be found at `https://github.com/aandreuisabal/OSN`. To see the latest changes, check out the "develop" branch.

Chapter 5

OPEN SENSOR NETWORK

The big picture objective of this project was to create a sensor network that allowed developers to gather real-time information from the Internet to create new solutions[Barcelo et al.,]. This chapter addresses how this main objective has been completed and dives into step-by-step explanations, from some ZigBee basic concepts (which are necessary to understand how the system works), to very detailed aspects related to the code, together with some flowcharts to visually interpret underlying features.

My contribution to the sensor network ecosystem is a set of tools to rapidly deploy such a system. More exactly:

- XBee® configuration files. They are ready to be used and loaded into these RF modules.
- Fritzing[1] schematics, in order to replicate the nodes I worked with.
- Sink daemon code, used to receive all the information and then upload it to the Internet.
- Code of the Arduino program that will be running on some of the nodes.
- This document, which will guide everyone that wants to replicate or expand my work.

The GitHub repository (`https://github.com/aandreuisabal/OSN`) is organized as depicted in figure 5.1.

Technologies, as described in Chapter 3 are well-tested and mature solutions, thus inferring the project the following two main features:

[1] Fritzing is open source software that allows to design Arduino-based prototypes. The same software can be used to design final PCB boards from the initial prototyping view.

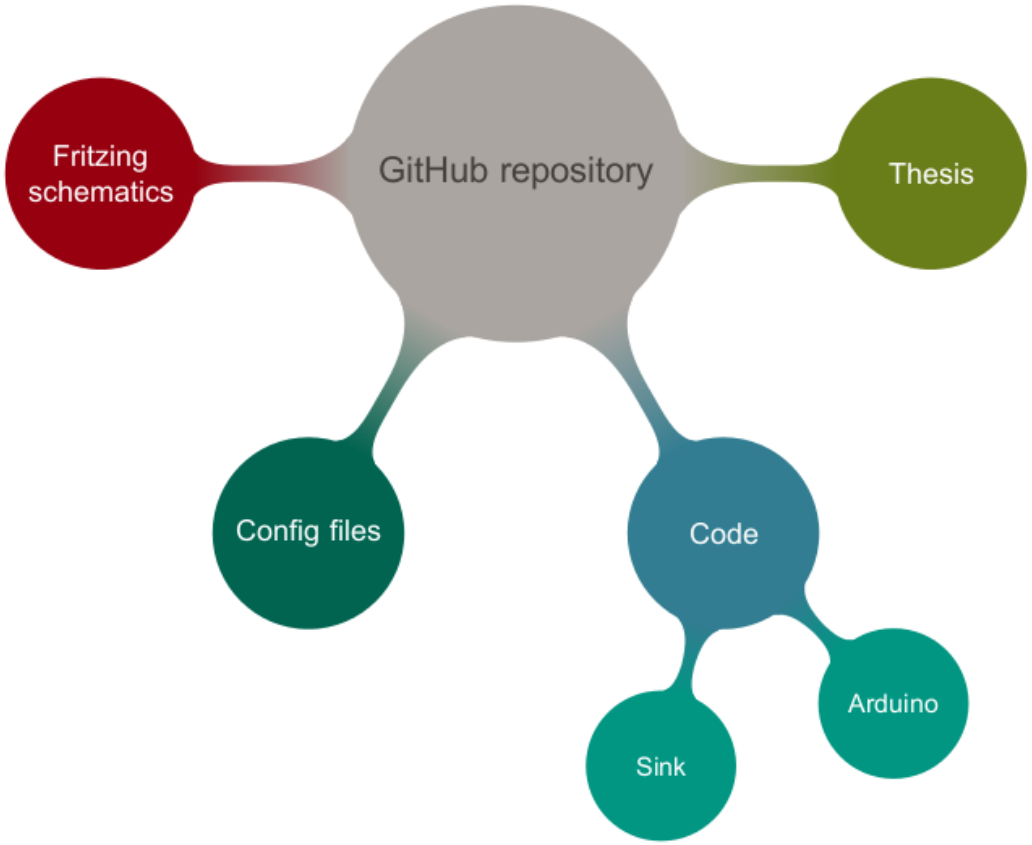


Figure 5.1: Structure of the GitHub repository.

- Flexibility — The system is prepared to transmit heterogeneous information. Each node is able to transmit different information and the sink will decode it anyway. This allows a community to gather what each individual wants, or to achieve better granularity where needed. That is, someone might be interested in measuring temperature every two blocks and humidity every four blocks.

- Velocity of deployment — To deploy this network, just the RF modules must be properly configured, as well as some small code tweaking. In other words, just sensor nodes have to be adapted to particular needs, the sink will work anyway.

5.1 Network topology

Network topology refers to the way nodes that conform the system are arranged, thus it is clear that this factor will determine very important components about the network, such as reliability, modularity, fault occurence, etc. The use of Digi Xbee® RF modules enables the network to be configured in any kind of topology, from a simple star to a complex mesh[2]. An example of such networks is depicted in figure 5.2, where each circle represents a node and the lines the wireless links between them.

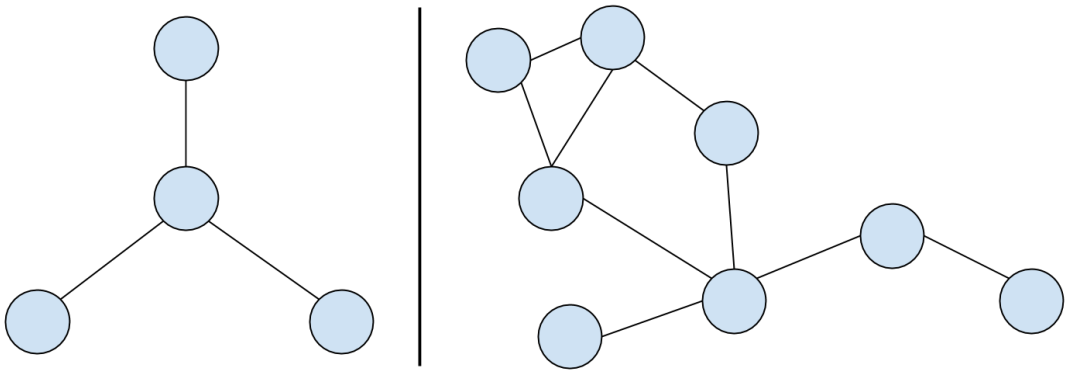


Figure 5.2: (Left) Star network topology; (Right) Mesh network topology.

As these RF devices only allow one single sink per network, not making use of mesh topologies would be an enormous drawback, since packet delivery would

[2]Networks where a packet can follow more than one path to reach its destination.

be subject to other nodes availability. Luckily, ZigBee specification allows enable some nodes to act as a relay for other nodes, enabling us to build complex networks with redundant paths. This statement, translated to battery-powered systems means that the more relay nodes a network has, the more prolonged the network's lifetime will be[Hou et al., 2005], since relay nodes will have to pass on less messages. This is true except in the case the topology is a chain[3].

A mesh sensor network of this kind can be fully connected —each node is connected to every other one— or partially connected. This topology brings many advantages. They basically enable the system to be:

- Self-healing — Allows the network to operate when a node goes down.
- Self-routing — When a packet is transmitted or forwarded, the route that it follows is created —that is, calculated— locally within every node.
- Self-forming — Nodes that are new to the network create links automatically with the rest of nodes and routes are created dinamically as well.

The mentioned features and constraints imply that the first step to build such a network will begin by placing a sink and then start building from there, progressively reaching more distant places.

5.1.1 Device roles

Inside an ZigBee network there are three roles that a node can assume: coordinator, router and end device.

A *coordinator* is the sink of the network, and as stated before, only one is allowed per network. A coordinator stores vital information about the network, acts as well as the trust center[4] and manages network security in general[Alberto Bielsa, 2010]. In this case, it will be connected to the Raspberry Pi so data can be processed and uploaded to the Internet.

A *router* can be understood as the previously mentioned relay nodes. It can generate and transmit data by itself to other router or to a coordinator but it is also able to forward packets from other nodes. If the network is very redundant it should not be a problem having a battery-powered router. If that is not the case, this option could lead to data outage, since packets from the edge of the network could not be forwarded.

Finally, an *end device* is the least capable device of all. It can only transmit information that will be or will not be forwarded, thus they are always on the edge

[3] In a chain topology (also called linear topology), each node has a two-way link with another node.

[4] Stores the keys if encryption is enabled, deciding who may or may not join the network.

of the network. Since no other node depends on an end device, they can make use of the *sleep mode*. This mode allows an XBee radio to wake up certain amout of time, transmit whatever it has to and then go back to sleep again. This mode is very energy efficient.

An example of such a network can be seen in figure 5.3.

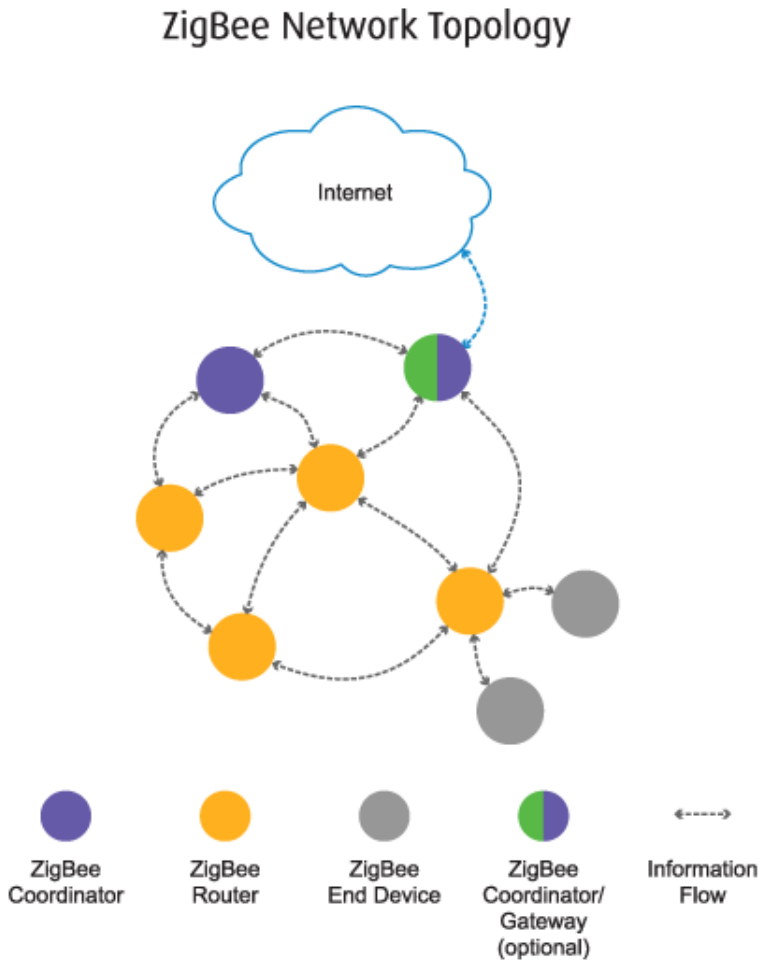


Figure 5.3: An example of an ZigBee network, taken from the ZigBee Alliance website (`http://zigbee.org`).

5.2 Sensor nodes

A sensor node is an element inside a wireless sensor network that is capable of gathering information, has some processing power and can relay information (if needed) to other nodes in the network[Chong and Kumar, 2003].

In the design of this network two feasible scenarios have been considered, depending on which kind of sensors need to be used —whether they are digital or analogic—, and if processing power is required. In case at least one of those features is needed, an Arduino plus an XBee® module are coupled together, with sensors attached to the board. Otherwise a standalone XBee® is used since it has a built-in ADC[5], hence being able to directly read information from analog sensors.

These two operational modes have been taken in consideration because despite one of them can equate the other's characteristics one can think of some applications where the features of an additional microcontroller are just not needed. For instance, a sensor network monitoring temperature in an industrial environment just needs the so-mentioned analog-to-digital converter and a transceiver. Although there are two types of nodes, both can be used at the same time inside a given network.

To configure the radio of a sensor node one must use X-CTU, a piece of software developed by Digi International that, although it is intended to be run on Windows, it works fine on GNU/Linux using Wine[6], as it can be seen in figure 5.4.

In the next subsections it will be presented how these two types of sensor nodes are wired, how do they work and how are they configured.

5.2.1 Standalone XBee

To collect data directly from an XBee®, we need to make use of the analog-to-digital converters mentioned in section 3.2. In figure 5.5 we can observe the pinout of an XBee radio, with all its DIO (digital input output) pins. For analog input, we can only use from `DIO0` (sometimes called as well `AD0`) to `DIO3` (or `AD3`)[Faludi, 2010].

An schematic of this node can be seen below (figure 5.6), where only light level and temperature are measured. This particular setup could be useful for example to prevent fires in forests (although the optimal setup should have a humidity sensor as well).

[5] An analog-to-digital converter takes a continous value –voltage, in our case– as its input and converts it to a digital numeral.

[6] Wine is open source software that helps running Microsoft Windows applications on Unix-

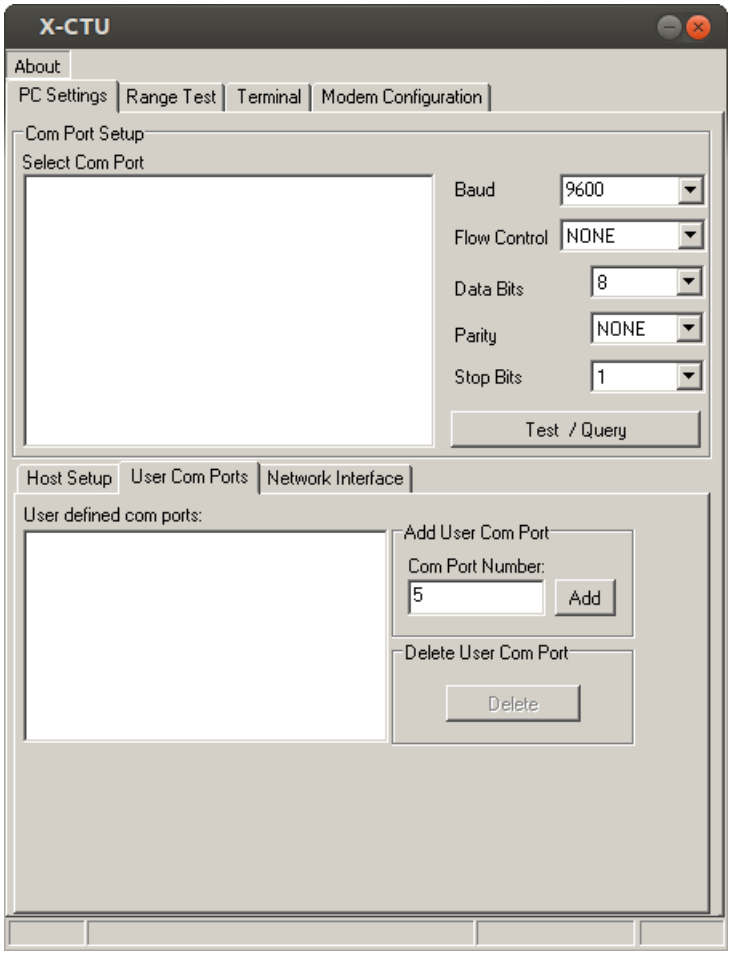


Figure 5.4: A screenshot of X-CTU running on Ubuntu 12.04 under Wine.

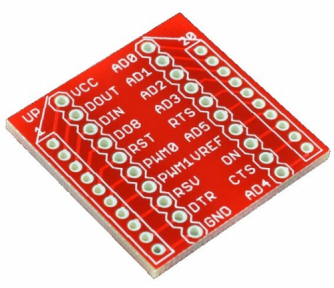


Figure 5.5: XBee pinout, seen from a breakout board.

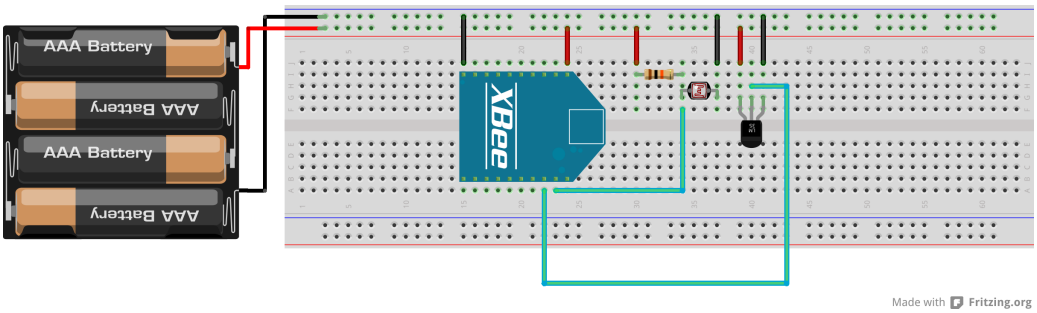


Figure 5.6: A standalone XBee node.

Here, with the help of four AAA batteries we power an XBee® as well as the other sensors placed in the prototyping board. The temperature sensor (subsection 3.1.4) has a dedicated output pin, which is directly attached to an analog input pin. As for light levels, the used LDR resistance[7] does not have an output pin, and this is why we have to collect the sensory value through a voltage divider.

Using a battery and a solar panel to power a sensor node like this one, along with *sleep mode* can result in very long lifetimes.

5.2.1.1 Configuring a standalone XBee®

The first step is to flash the radio module with the correct firmware, that is, with the `XB24-ZB` firmware. This `ZB` firmware implements the ZigBee 2007 specification[8]. Also, a function set (or role) has to be set, as shown in figure 5.7.

To configure a standalone node one must ensure that the RF module has the same `PANID` than the coordinator —that is, the same network—. Also, API mode shall be enabled, thus setting the `AP` parameter to `2`, which means not only that API mode must be used but also *escaping*. This `AP` parameter must be set to `2` because the Arduino based node only works with escaping enabled (as described in the next subsection). Thus, to achieve a certain degree of homogeneity both the nodes and the sink need to have this parameter explicitly enabled.

With escaping mode the system escapes some special characters. In other words, if special characters appear in the packet —for instance `0x7E`, which serves as a start frame delimiter— they are replaced by other sequences so they can be decoded as well by the receiver but without causing any trouble in the interpretation phase. Receiving an arbitrary `0x7E` could pose many problems for the receiver. It would not know when a packet really starts[International, 2013].

---

like operating systems.

[7] A photoresistor whose resistance varies depending on the surrounding light level.

[8] At the time of writing, the latest specification is from 2012.

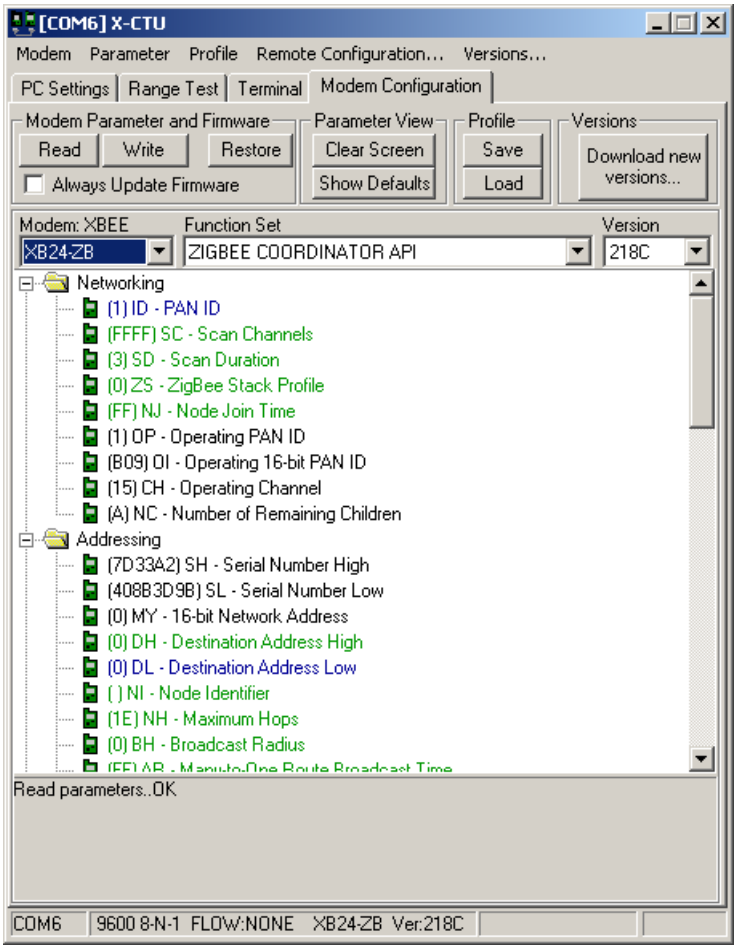


Figure 5.7: XBee® configuration through X-CTU.

Since we want to transmit samples periodically, we must configurate a sampling rate inside X-CTU, with the parameter `IR`. This value must be hexadecimal, so if for instance we want the module to transmit values every second, we must set `IR` to `3E8` —or 1000ms—.

Finally, parameters `D0`, `D1`, `D2` and `D3` can be set to `2`, which will mean they are in ADC mode. In other words, for each sensor wired to one of these pins, one must configure those pins to work in the proper mode.

Example configuration files were exported from X-CTU and can be freely downloaded from the original git repository[9]. More precisely, they can be found in the `Config/X-CTU` folder.

5.2.2 Arduino-based node

Arduino is capable of executing C code, thus being able to process any kind of information no matter how complex it is (always bearing in mind its hardware limits). To transmit all the information, the RF module attached to it will be configured as described in subsection 5.2.2.1.

As a proof of concept, the example setup I have worked with has an analog sensor that measures sound levels (3.1.2), another one that measures air quality in terms of fine particles in the air (3.1.3) and a digital sensor that reads humidity and temperature (3.1.1). A possible setup with these elements is shown in figure 5.8.

Here, the Arduino board can be powered by batteries or by a more stable power supply. It is the very same board that powers the sensors and the XBee®. Thereby, information is gathered and finally transmitted through the RF module (more information in section 5.2.2.2).

This type of node has the same communication features as the standalone XBee. That is, encryption, acknowledgements, etc. Additionally, ACKs are better handled in this case since an Arduino can *react* to them.

5.2.2.1 XBee® configuration

The configuration for this type of node is quite simple. The same firmware and function set that were mentioned in subsection 5.2.1.1 have to be written in the XBee®.`PANID` must be set to the same than the coordinator's so they can interact with each other, and `AP` (that is, API mode) must be set to `2` to enable escaping.

[9] `https://github.com/aandreuisabal/OSN`

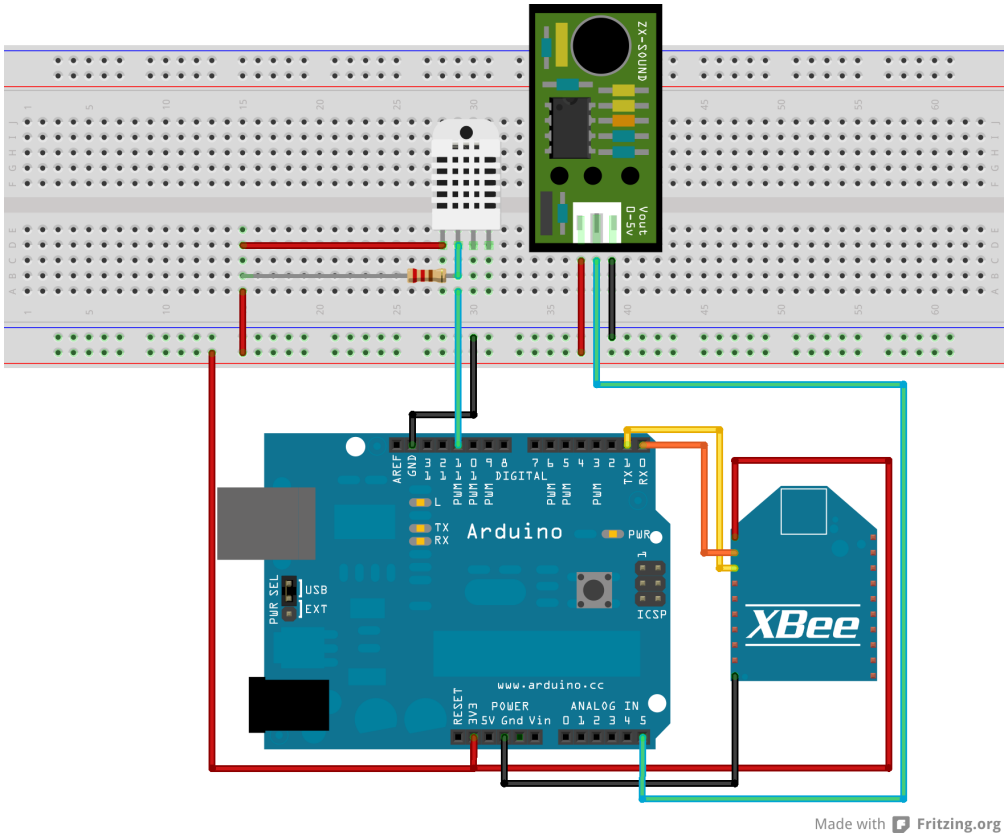


Figure 5.8: Sensor node based on Arduino.

5.2.2.2 Arduino sketch

A sketch is nothing more than the program an Arduino runs. The basic code is written in C/C++, and it consists of a main file —with `.ino` extension— and the XBee® libraries. The code can be browsed and downloaded via GitHub. There are two versions of the sketch:

- Exact same code I used to conduct the experiments, so anyone can verify and/or test the obtained results.
- A skeleton file, that follows a very minimalistic approach in terms of lines of code but fully commented. This way, it can be extended as desired to build a sensor network from scratch with customized sensor nodes.

Arduino IDE is based on Processing IDE[10], and it follows the same structure than the Processing programming language. There are two main functions necessary for every program to work, namely `setup()` and another one called `loop()`. Respectively:

- The `setup()` funcion initializes variables, modules (such as the XBee), libraries and sets pins in specific modes. After this function is successfully executed `loop()` is immediately called.
- `loop()` is a function that as its own name indicates, is executed over and over again. Thus inside this structure is where the action takes place.

In our case, even before `setup()` starts, the following steps take place:

- An object of type `XBee` is created, so serial information can be exchanged between the microcontroller and the RF module.
- Additional information useful for the transmission is set: destination address (by default `0x0000000000000000`[11]), how big the packet will be, etc.
- Also, an array called `payload` is initialized, which will contain all readings as well as some *metadata*.
- The variables that will hold the different readings are also declared outside the two main function so they are recognized in a global scope.

[10] `http://processing.org`

[11] This address is used to simply reach the coordinator. It is possible to write the specific address of the XBee —which is written in the RF module— as well.

- Finally, auxiliary C unions[12] are created. One for every compatible data type.

In our `setup()` step we initialize a serial connection with the XBee® and set a pin to act as digital output. This pin is the number `13`, which is connected to the on-board LED.

Then, in `loop()` all sensory values are recollected and then transmitted via a ZigBee packet. Finally, if the previously mentioned LED blinks once that will mean transmission took place successfully, otherwise it will blink twice. This last feature is especially interesting when debugging.

In figure 5.9 it is depicted how the program works in more detail. This flowchart follows a top-down approach. That is, from general to more specific functions.

As for how values are read from the Arduino, figure 5.10 explains how this process takes place step by step. Basically, digital values are read one time since the readings are more precise, and when reading an analog value the Arduino computes an average of `n` samples to smoothe the values from "jumpy" sensors.

5.2.2.2.1 Working with metadata

An initial requirement was that each sensor node can transmit whatever it needs to, I designed a rudimentary but efficient mechanism. At the start of every packet, an integer is sent. This is the only fixed value that every packet will hold. This integer will be tremendously helpful so the sink knows how to decode the payload —keep in mind that data is sent in binary form—.

That is, the four first bytes of payload of every packet will be decoded as an unsigned integer at the sink — which ranges from 0 to $(2^{16} - 1)$—. Then, each digit that conforms this number will be interpreted (one by one) as a unique data type, using table 5.1. The range of this unsigned integer is bigger when using other flavors of the Arduino, such as the Arduino Due[13].

To come up with an actual way to translate between digits and data types, I had a look at which data types the Arduino IDE could handle, and which of them Python can decode —that is, which data types do they share—. For more information on supported data types one can visit the reference on the official Arduino webpage[14] and Python documentation on data structures[15].

To give an example, let's say a node wants to transmit two floats and a boolean. According to table 5.1, this would yield the numbers `9`, `9` and `1`. Hence this

[12] An union allows us to represent information in more than one way. In our case it helps us convert integers, booleans, etc. into byte-level data.

[13] `http://arduino.cc/en/Guide/ArduinoDue`

[14] `http://arduino.cc/en/Reference/HomePage`

[15] `http://docs.python.org/2/library/struct.html`

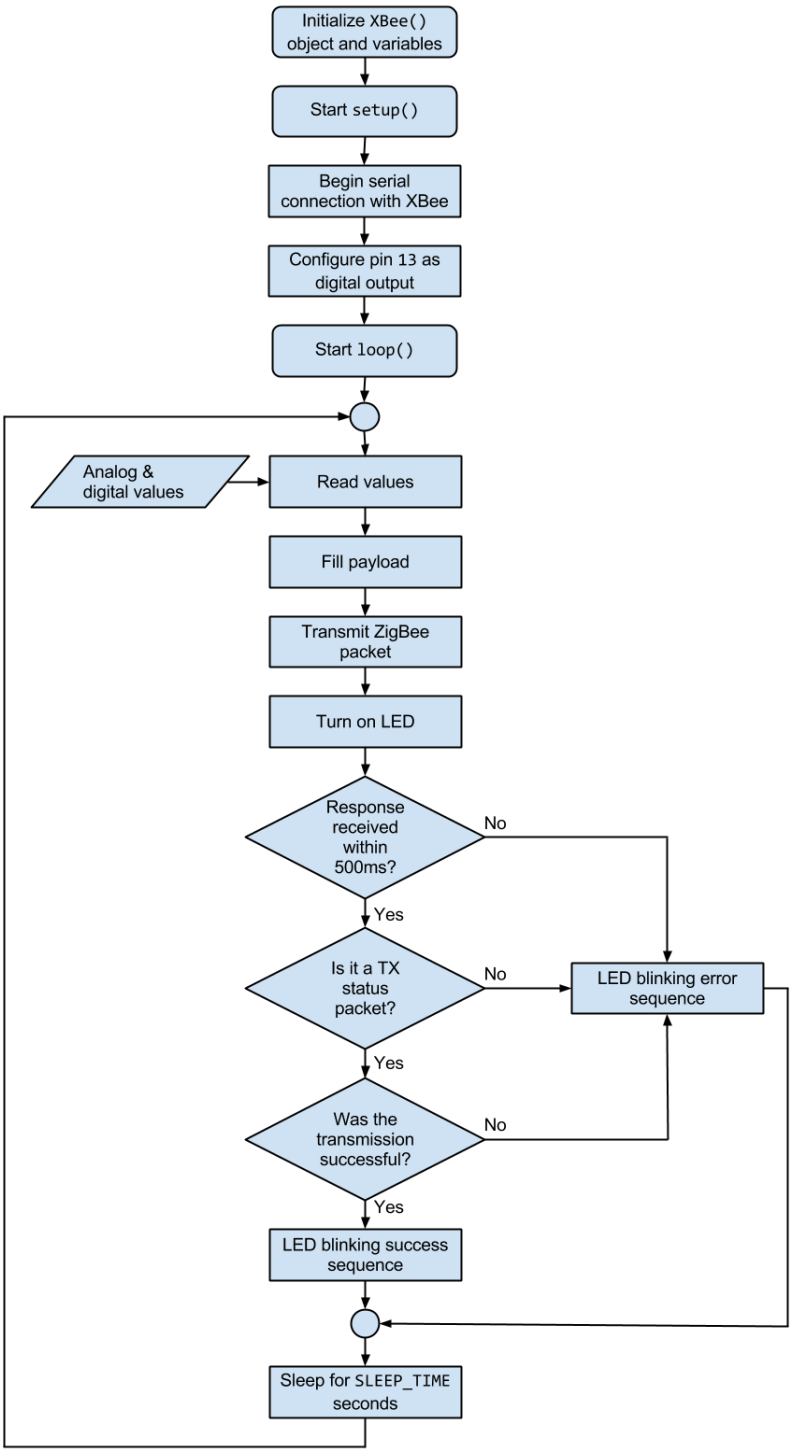


Figure 5.9: Flow diagram of the Arduino program.

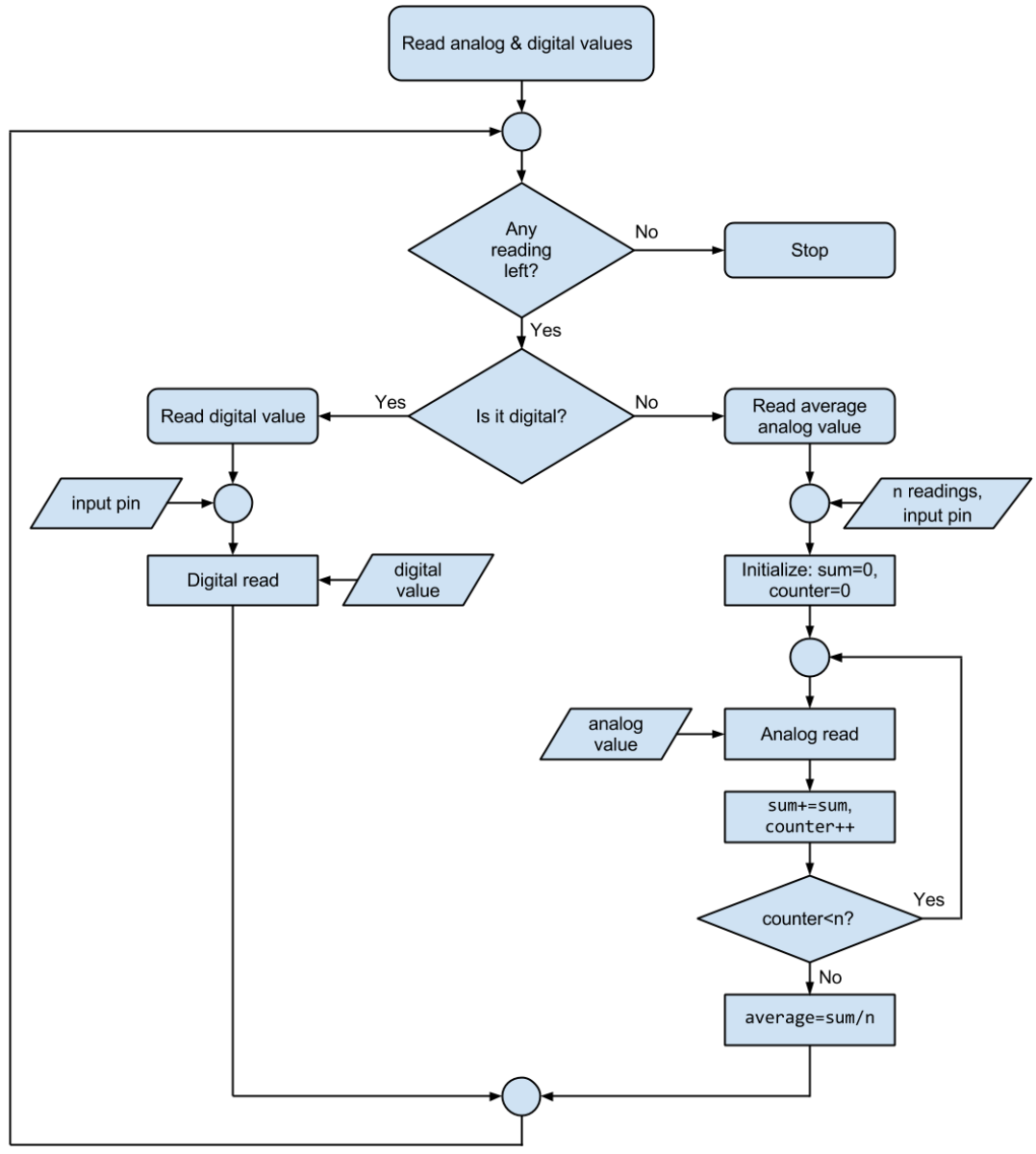


Figure 5.10: Flowchart of how values are read.

Number	Data type
1	`boolean`
2	`char`
3	`unsigned char`
4	`int`
5	`unsined int`
6	`long`
7	`unsigned long`
8	`short`
9	`float`
0	`double`

Table 5.1: Mapping between numbers and data types.

number (`METADATA` in the code) can be any permutation using the following digits: `199`.

Nonetheless, it is recommended to first transmit the lowest values (starting with `1`,`2`...) and finishing with the highest ones (... `9`,`0`). This is because since the range of an `unsigned int` is somewhat small —at least in a 8-bit architecture— transmitting information in this order reduces the probabilities or exceeding that range (again, from 0 to $(2^{16} - 1)$).

In the current state of the code, this "magic" number has to be hardcoded. Once this number has been written in the `METADATA` variable, the information shall be sent in the same order. Following the previous example, this number would be `199`, thereby sending the boolean first and then the two remaining floats.

5.2.2.2.2 Packing information

Information is packed with the help of unions, as stated before. A union is declared as follows:

```
union u_boolean {
    uint8_t b[1];
    boolean boolean;
} boolean_union;
```

This means that the union named `boolean_union` will translate indifferently between a boolean and a byte. Later on, the `payload` variable —which as its own name indicates, holds the payload— shall be filled with the data (including the metadata):

```
boolean_union.boolean = sample_boolean;
```

```
for (int i=0;i<BOOLEAN_SIZE;i++){
    payload[i]=boolean_union.b[i];
}
```

Although these lines of code seem "messy" at first glance its functionality is quite simple. It just loads the payload with byte information.

5.3 Network sink

The sink is where all the information is headed. As stated in the previous section, it is composed of a Raspberry Pi with an XBee module connected to it. The schematic, although simple can be seen in the figure 5.11. Note that in the figure there is not an XBee but a breakout board for it that has a miniUSB interface, very useful for our purposes. On top of it there is the actual XBee®.

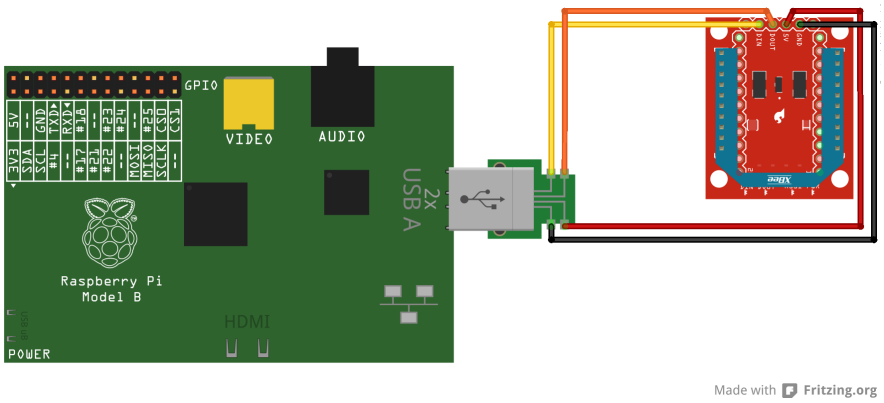


Figure 5.11: Schematic of the sink.

The GNU/Linux distribution that has been chosen to operate in this device is Arch Linux ARM, as stated in section 3.6. Its repositories are huge because regular users contribute to a non-official repository, called "Arch User Repository" (also known as AUR). There, all the packages mentioned in chapter 3 are available without compiling from source —in the form of binaries—.

The script, called `server.py` is then run as a daemon which will be always receiving information. It needs two arguments:

- Device file that interfaces with the XBee® (e.g. `/dev/ttyUSB0`). Depending on the devices already attached to the computer the last number might change.

- Baud rate the XBee® is working at (e.g. `9600`). Can be customized via X-CTU.

What the script basically does is run an asynchronous `XBee` dispatcher, which will create a new background thread for every new packet that arrives, thus enabling the sink to process many packets at the same time without blocking the whole script. The program is quite modular, since it allows to upload the information to the website the user wants by just uncommenting certain lines. A more detailed view on how it generally works can be seen in figure 5.12.

However, in figure 5.12 we cannot appreciate how packets are actually handled, and what happens to them afterwards. For that matter, figures 5.13 and 5.14 explain these two phases more precisely.

At the moment of writing, two uploaders have been created. One for Xively[16] and one for any Nimbits cloud[17]. The results can be seen in chapter 6.

5.3.1 Setting up the sink

The XBee® settings that have to be set are the same that those on an Arduino node. That is, a certain `PANID` and the `AP` parameter set to `2`.

These are the steps related to the Raspberry Pi that have to be followed to get a functioning sink:

1. Get a compatible SD card[18], download the latest ISO image from the Arch Linux ARM website[19] and transfer it to the card. Depending on what operative system you are using you will want to use one tool or another (`dd` tool for *NIX-based operating systems, Win32DiskImager for Windows). If the operating system is correctly loaded in the SD card the Raspberry Pi shall boot properly and its LEDs will start blinking.

2. To actually interact with the device, there are two options:

   - Through a display that accepts HDMI input and a USB keyboard.
   - Arch Linux ARM has the SSH[20] daemon enabled by default.

3. Arch Linux makes use of `pacman`, a wonderful package manager. From a terminal, issue the following command (requires root access):

[16] `http://xively.com`
[17] `http://nimbits.com`
[18] `http://elinux.org/RPi_SD_cards`
[19] `http://archlinuxarm.org/platforms/armv6/raspberry-pi`
[20] Secure Shell allows to remotely access another machine and remotely execute commands.

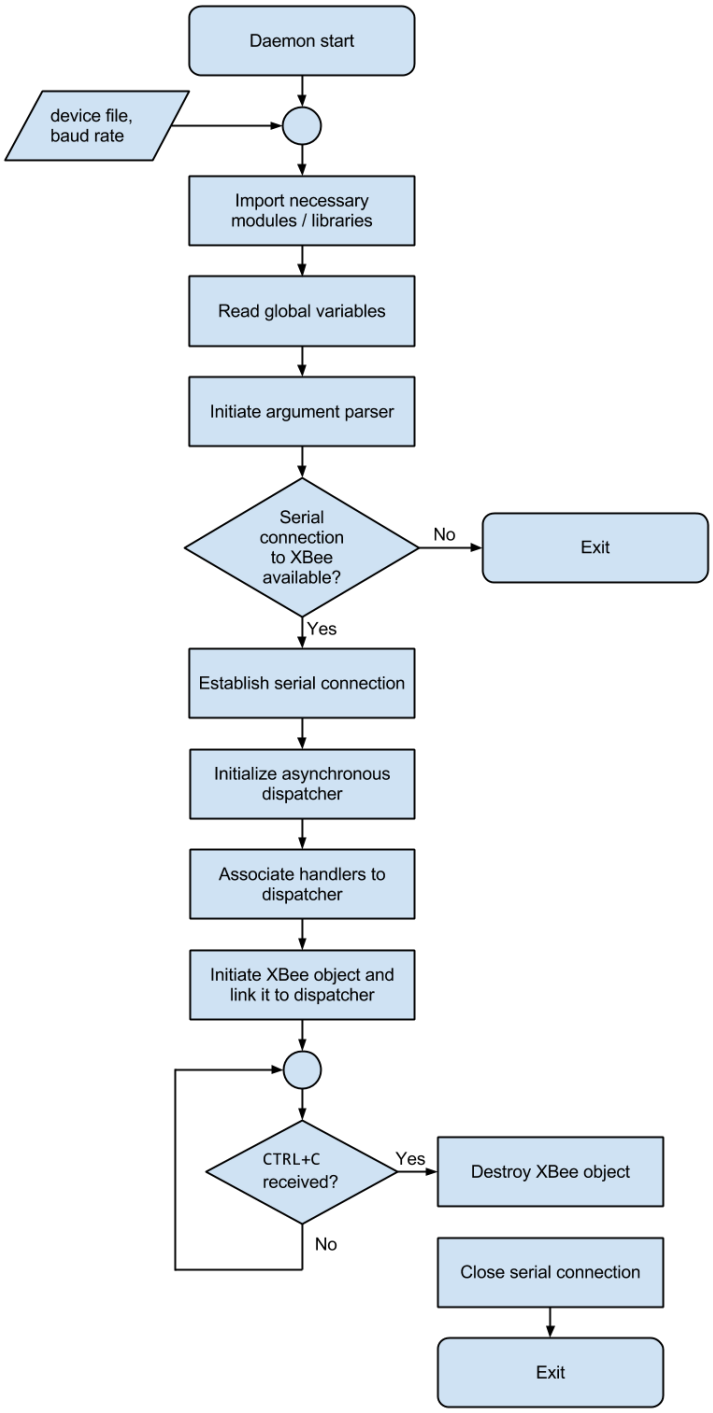


Figure 5.12: Flow diagram of the sink script.

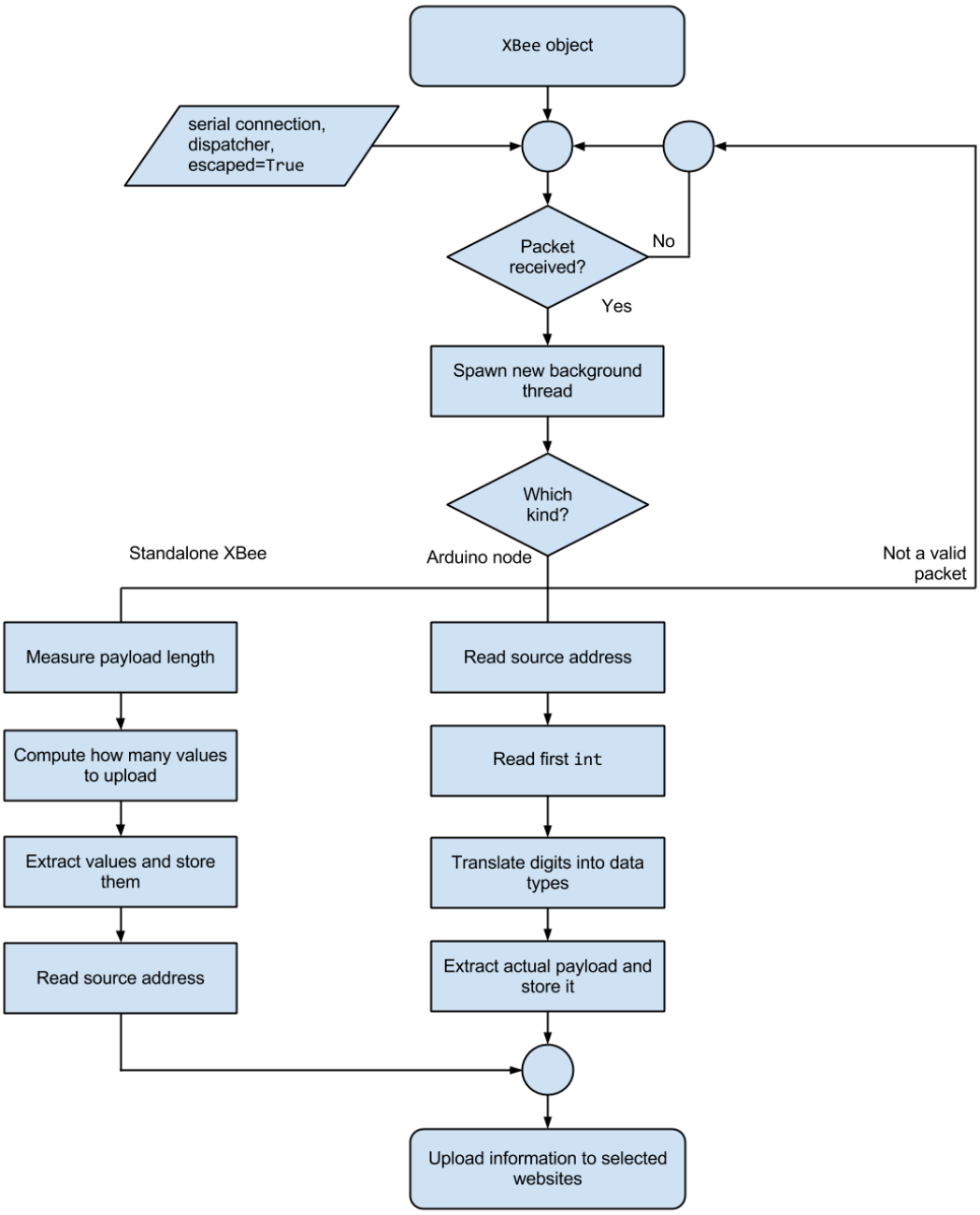


Figure 5.13: Flowchart of how packets are dispatched.

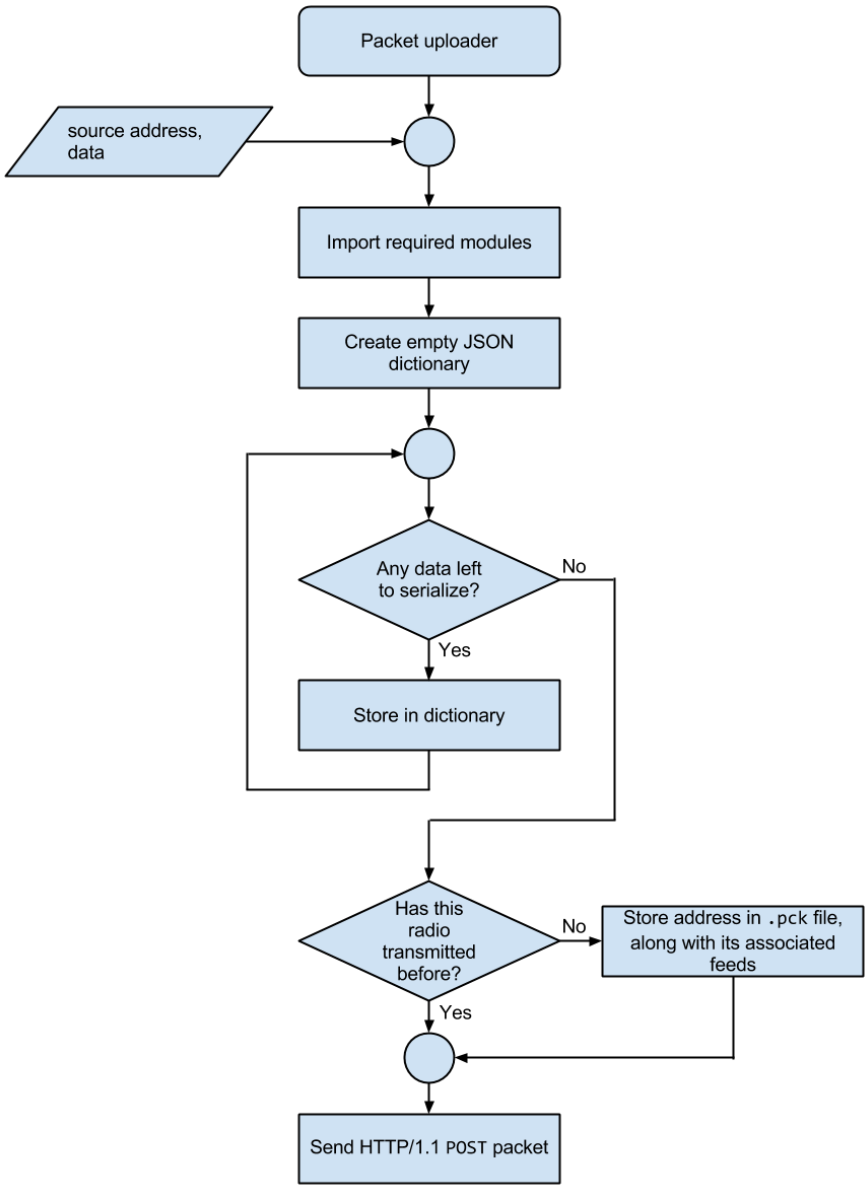


Figure 5.14: How a packet uploader works.

```
pacman -Syu \
python2 python2-requests \
python2-pyserial
```

This will upgrade all the operating system packages and will install as well the necessary ones for the script to work.

4. Clone the GitHub repository by issuing:

```
git clone \
git@github.com:aandreuisabal/OSN.git
```

This will clone the GitHub repository inside a folder called `OSN` in the current directory. There, inside `Code/server` folder there are all the necessary files to run the sink daemon.

5. Modify the file `osn.service` so the path to the script is correctly set. Finally, move this file to `/etc/systemd/system/` and issue this command to start the script at boot time (requires root privileges):

```
systemctl enable osn
```

5.4 Deploying the network

To deploy this network one must first configure the coordinator as explained before, and start placing sensor nodes —configured properly— around until reaching the desired coverage. As for the script, one must first fill the necessary constants, such as API keys —necessary to upload information to data clouds—, usernames and passwords. This last step is done through the modification of the uploader files (located in `Code/server/uploaders/*.py`), where variables denoted by capital letters hold this information. Each file refers to a data cloud where data shall be uploaded. So for instance, if you are willing to upload data to Xively, the user shall modify the file `uploaders/xively.py`.

Chapter 6

RESULTS AND DISCUSSION

In this chapter I will present the obtained results along with little reflexions. This chapter will also be strongly correlated with the initial objectives and how they have been carried out.

The tests were done at Tànger building that belongs to the Pompeu Fabra University. In this edifice, there are a lot of Wi-Fi networks which also operate in the 2.4 GHz band, apart from wireless mesh network nodes and concrete walls (which are known to pose serious problems to this particular ISM band).

Under this environmental conditions, I set up a network whose nodes had a maximum range of 25m, half of the range indicated by the datasheet provided by Digi[1], probably caused by spectrum saturation.

The tests were carried out with the two possible sensor nodes, which transmitted some environmental factors. A picture of one deployed node is displayed below (figure 6.1). Despite the messy appearance, it took ten minutes to load the sketch on the Arduino, configure the XBee® and wire cables and sensors. More precisely, they were measuring temperature, relative humidity and noise levels.

Then, two other nodes were transmitting at the same moment. Those were two standalone XBee® nodes, depicted in figure 6.2. With the help of the Emartee mini sound sensor (subsection 3.1.2) and the TI LM35 (3.1.4) they measured noise levels and temperature. Although measured factors are somehow redundant, they served as a proof of concept of the system.

Sadly, due to lack of opportunities, no external organizations were able to test this network.

Putting aside the medium restraints of the physical medium the protocol uses, the main result of this project can be appreciated on figure 6.3. There, gathered data can be seen in the visualization tools that Xively provides.

When the sink script is running, the following output is presented through

[1] `ftp://ftp1.digi.com/support/documentation/90000866_A.pdf`

Figure 6.1: An actual Arduino node measuring environmental factors.

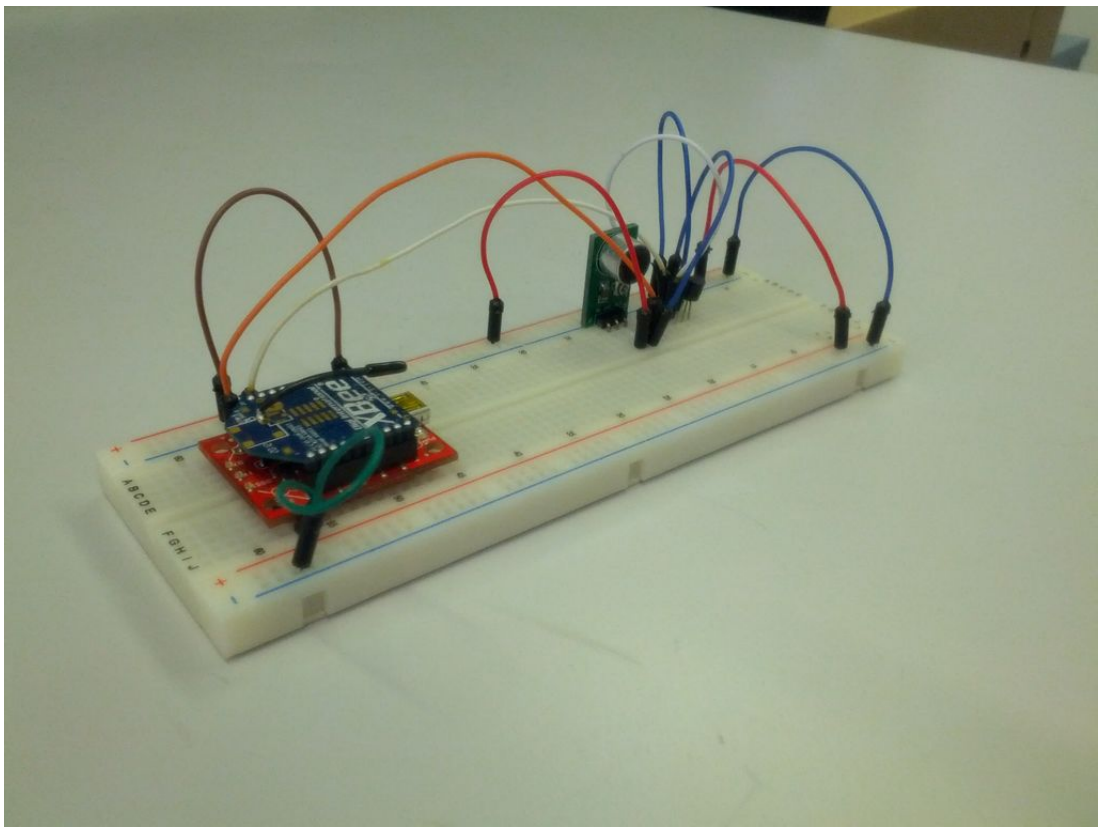


Figure 6.2: Standalone XBee® measuring temperature and noise levels.

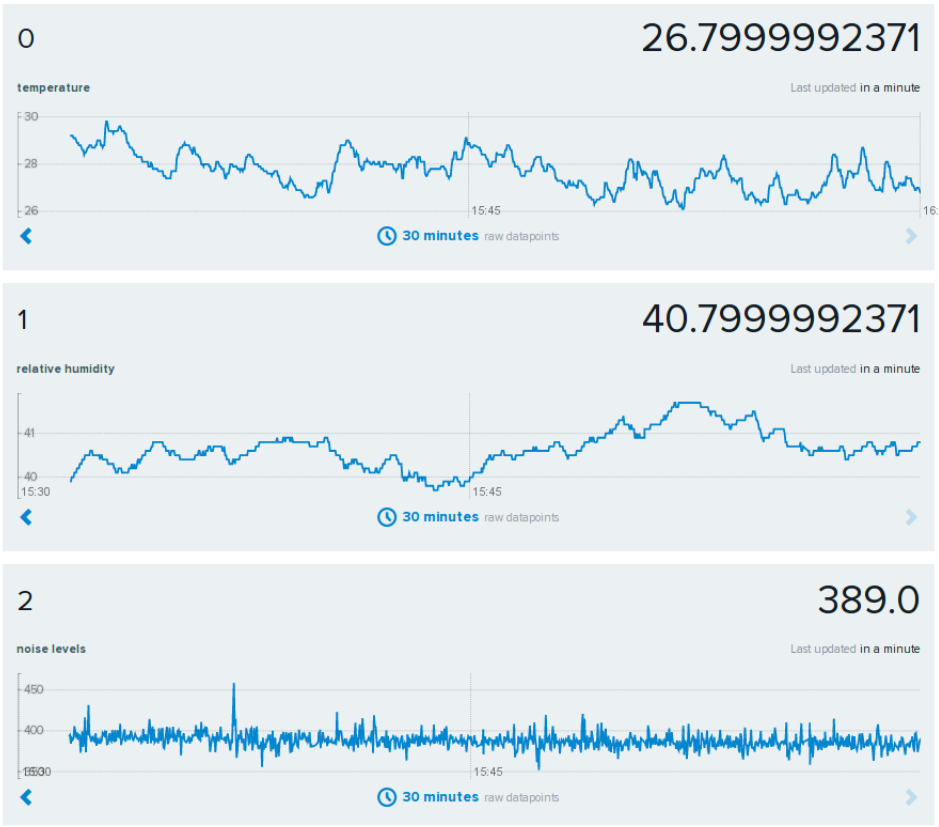


Figure 6.3: How data is viewed in Xively (`http://xively.com`)

`stdout` (figure 6.4). That is, the original sender of the packet as well as its content.

```
p312z ~/Dropbox/Code/server $ python2 server.py /dev/ttyUSB0 9600
Listening on /dev/ttyUSB0 ...

PACKET RECEIVED FROM: 408b3de1
CONTENT: [28.299999237060547, 39.400001525878906, 381.6000061035156
]

PACKET RECEIVED FROM: 408b3de1
CONTENT: [28.299999237060547, 39.5, 379.70001220703125]

PACKET RECEIVED FROM: bf3e
```

Figure 6.4: Output of the script running.

Chapter 7

CONCLUSIONS AND SUGGESTIONS FOR FUTURE WORK

This chapter addresses the conclusions extracted from the obtained results as well as some pointers which could be useful when future working on this type of networks.

7.1 Conclusions

This project demonstrated that it is possible to build an entire sensor network through prototyping, thus reducing development time and development costs. Although it is true that the final users will require a little bit of background in sensor networks and programming to adapt the system to his/her needs, this grade of involvement is inherent to every BuB system.

This solution can be considered then a good solution to those seeking a cheap and flexible alternative to the presented services in chapter 2. Any kind of data can be now automatically uploaded to Xively and a Nimbits cloud.

7.2 Future work

In the following bulleted list I present some ideas that are offered in order to improve this work in future iterations.

- The current program for the Arduino consumes a reasonable amount of power. To solve this problem, one could:

  - Using the `narcoleptic` library[1], we can put the whole Arduino in sleep mode and wake it up when desired. This would be the perfect solution for end devices, since they would not need to forward packets for other devices.
  - Another option however, is to manually tweak the `sleep` library from the AVR libraries (available from the Arduino IDE). According to the official documentation from Atmel[2], the microcontroller can sleep while keeping some clocks and interfaces awake (i.e. powering pins). Hence, although the Arduino can sleep, the XBee can still be working relaying messages. This solution, although more complex, can serve for end devices and routers.

- Create more uploaders so data can be uploaded to even more data storage clouds, since at the time of writing data can just be uploaded to Xively and Nimbits.

- Metadata generation, as explained in chapter 5, could be done automatically.

- Although the system works pretty well with ZigBee, the implementation is not fully open[3]. Thus testing open standards such as DASH7[4] would make this network completely open.

[1] `https://code.google.com/p/narcoleptic/`

[2] Documentation can be found at `http://www.atmel.com/Images/doc8161.pdf`, at page 39.

[3] Although there are some open implementations like Open-ZB or ZBOSS.

[4] `http://dash7.org`

Bibliography

[Alberto Bielsa, 2010] Alberto Bielsa, D. G. (2010). Wireless sensor networks research group. [Online; accessed 20-May-2013].

[Barcelo et al.,] Barcelo, J., Roca, R., Holding, A., Spensley, P., Domingo, A., Calcerano, G., Goretti, M., Dalmau, L., Oliver, M., Cano, C., et al. Bottom-up broadband pilot proposals in europe (c4eu 5.1. 1: Report on selection of opportunities and projects-a).

[Caragliu et al., 2009] Caragliu, A., Del Bo, C., Nijkamp, P., et al. (2009). *Smart cities in Europe*. Vrije Universiteit, Faculty of Economics and Business Administration.

[Chong and Kumar, 2003] Chong, C.-Y. and Kumar, S. P. (2003). Sensor networks: evolution, opportunities, and challenges. *Proceedings of the IEEE*, 91(8):1247–1256.

[Faludi, 2010] Faludi, R. (2010). *Building Wireless Sensor Networks: with ZigBee, XBee, Arduino, and Processing*. O'Reilly Media, Incorporated.

[Gertz and Di Justo, 2012] Gertz, E. and Di Justo, P. (2012). *Environmental Monitoring with Arduino: Building Simple Devices to Collect Data about the World Around Us*. Make.

[Hollands, 2008] Hollands, R. G. (2008). Will the real smart city please stand up? intelligent, progressive or entrepreneurial? *City*, 12(3):303–320.

[Hou et al., 2005] Hou, Y. T., Shi, Y., Sherali, H. D., and Midkiff, S. F. (2005). On energy provisioning and relay node placement for wireless sensor networks. *Wireless Communications, IEEE Transactions on*, 4(5):2579–2590.

[Institute, 2011] Institute, P. M. (2011). *A Guide to the Project Management Body of Knowledge*. Project Management Institute, fourth edition.

[International, 2007] International, D. (2007). Xbee series 2 oem rf modules. [Online; accessed 6-April-2013].

[International, 2013] International, D. (2013). Escaped characters and api mode 2. [Online; accessed 4-June-2013].

[Nafis, 2012] Nafis, C. (2012). Monitoring your air quality. [Online; accessed 4-April-2013].

[Oliver et al., 2010] Oliver, M., Zuidweg, J., and Batikas, M. (2010). Wireless commons against the digital divide. In *Technology and Society (ISTAS), 2010 IEEE International Symposium on*, pages 457–465. IEEE.

[paraZite, 2013] paraZite (2013). main.parazite # anarchy files and underground links — sun apr 7 05:23:56 eest 2013. [Online; accessed 11-June-2013].

[Sharp, 2006] Sharp (2006). Gp2y1010au0f compact optical dust sensor. [Online; accessed 6-April-2013].

Appendix A

PILOT CHARTER

Pilot Charter — OSN

Alejandro Andreu Isábal

December 7, 2012

1 Introduction

The opportunity of making this project possible is born through the initiative Commons for Europe, which at the same time was brought by one of our tutors. More precisely this pilot –open sensor network– belongs to the bottom-up broadband branch, which pursues the utilization of several technologies in networks that follow this approach. That is, a network that gets bigger and better thanks to the active participation of its users.

The technologies used in OSN are –as its name implies– sensors and and the kind of networks that can be created with them.

2 Description

The project consists on the deployment of a sensor network which shall gather real-time environmental data. This information would be then uploaded to an open data portal so developers can create new applications that help improve the citizens daily lives.

3 Goal

Deploying such a bottom-up network with not too many nodes to prove that indeed it meets the requirements and test the utility of this type of networks.

4 Objectives

- Recollecting data from several types of sensors before december 2012.
- Point-to-point communication between two nodes before the end of the present year.
- Being able to upload information to an open data portal before the end of 2012.
- Communicating mesh network before the first trimester of 2013 reaches its end

5 Scope

The correct choices have to be made in order to prove the efficiency of this kind of network inside a sensor network. That is, protocols, hierarchy, etc.

6 Shareholders

- Governmental regulation.
- Commons for Europe.
- Universitat Pompeu Fabra.

• Material and technology suppliers.

• Public organizations.

• Users.

7 Costs

The cost of this project by C4EU/UPF raise to 6,000€ approximately. These costs can be split into the scolarship that those pilots give to the students, amortization of the equipment, two european trips, the time that the director spends as well as fungible assets.

7.1 Investment return

This return isn’t monetary since the pilot objective does not consist on selling a product. That is, thanks to the accomplishment of the pilot there will be an impact in future deployments of this kind of networks.

Appendix B

SENSOR BOARD CHOICE

This document was part of one of the BuB4EU workshops we attended, as mentioned in Chapter 4. More precisely, this workshop helped me chose Arduino over other platforms.

Initially we had lots of Crossbow TelosB rev. B nodes[1] ready to use in the laboratory so this was quite a good option to consider. They run over an embedded Unix-based operating system —TinyOS—, which brings several advantages such as default multithreading, advanced customization options, etc. However this board lacks the community and good documentation that other platforms such Arduino and its derivatives do have. Also, complexity when using this platform is high.

Hence contemplating another options such as the one mentioned before is a must. Arduino, while maintaining simplicity, is as well a very powerful platform used by many hackers and hobbyists today. This, along with its open source philosophy has created a large community that has been very active for the past years. Moreover, many companies create modules for this prototyping platform to extend its original functionalities. It does support ZigBee —which is a prerequisite— through an external module called XBee.

Therefore, being complexity my main criteria I chose Arduino as my sensing board to complete this project.

[1] `http://bullseye.xbow.com:81/Products/productdetails.aspx?sid=252`