POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Informatica (Computer Engineering)

Tesi di Laurea Magistrale

SwarmBike: an IoT Sensing Platform for Air Pollution

Relatore
prof. Fulvio Corno

Candidato
Carmelo Migliore

Luglio 2016

Premessa

Questa tesi di laurea è stata scritta per concludere il corso di Laurea Magistrale in Ingegneria Informatica al Politecnico di Torino, al termine di 9 mesi di lavoro trascorsi presso TIM, in particolare presso il Joint Open Lab SWARM di Torino.

I Joint Open Lab (JOL) sono laboratori di ricerca e innovazione creati all'interno di poli universitari. Nascono da collaborazioni e accordi in specifici campi d'interesse scientifico e tecnologico che vedono coinvolte Telecom Italia e le principali università italiane.

Il JOL SWARM in particolare è un laboratorio che ha l'obiettivo di creare sinergie tra la ricerca accademica e l'innovazione industriale nel campo dei sistemi distribuiti della cosiddetta "Internet of Everything". Il Laboratorio è attivo da febbraio 2014 presso la sede TIM situata all'interno del Politecnico di Torino e persegue un nuovo modello di Open Innovation, che mira a mettere insieme l'università e l'ecosistema tecnologico presente sul territorio. Il laboratorio è coinvolto in diversi progetti di ricerca in ambito ICT che studiano come applicare tecnologie distribuite e pervasive attraverso i paradigmi di collaborazione e cooperazione, dove gli oggetti e/o le persone mediante dispositivi interagiscono tra di loro in sistemi complessi che possono far "emergere" comportamenti collettivi, come avviene per gli sciami di animali (swarm).

Il progetto, un sistema IoT (Internet of Things) per il monitoraggio della qualità dell'aria attraverso dei sensori installati a bordo delle biciclette, è stato sviluppato da me, Carmelo Migliore, con il supporto di TIM, rappresentata dal tutor aziendale Pino Castrogiovanni.

SWARM
Beyond the Internet of Things

JOINTOPENLAB
TIM

Summary

In recent times, there has been the rising of the so called Internet of Things (i.e. many common objects can be now connected to the Internet) and the contemporary enormous success of smart phones (equipped with many sensors). This has led to the idea that users could be exploited to collect data by mean of their devices (crowd sensing), opening new scenarios especially for environmental monitoring.

The objective of this thesis work is to design and prototype an IoT air pollution monitoring system called SwarmBike, based on a sensing device which can be purchased by users and mounted on their bicycles. The main idea is to exploit bikes as mobile sensor nodes which can gather data from the environment. To make the SwarmBike system more appealing for final users, it was decided to enrich the device with some secondary features, which were selected based on the results of a survey. An anti-theft system resulted the most wanted feature by the users.

To decide which pollutants to monitor and which sensors to use, an analysis of the main hazardous substances present in the air of urban areas has been performed. The results of this analysis have led to the choice of a Carbon Monoxide electrochemical sensor for the SwarmBike device, because this type of sensors guarantees a low power consumption and a good reading accuracy at a reasonable cost.

The whole SwarmBike system consists of three major components:

- the device itself
- the smartphone application
- the cloud backend

The device is the component mounted on the bike and is responsible to acquire pollution data and send them to the cloud using the built-in cellular connection. Furthermore it permits to remotely locate the position of the bike and to detect theft attempt using the buit-in accelerometer. It has been implemented using a STM32 Nucleo prototyping board, and ARM mbed OS has been exploited to develop the firmware.

The smartphone application, developed for Android, is used to manage and configure the device, communicating through a Bluetooth Low Energy connection.

It permits to view the position of the bike on a map, it shows the current CO sensor value and it receives push notifications in case of a theft attempt.

The cloud backend consists of two main subcomponents. The first is SiteWhere, an IoT server platform, which is used to collect and store all the data sent by the devices. The other is second one is based on Meteor.js, a cloud platform for developing web and mobile applications, which is responsible to build a map showing the current positions of the device and CO measures collected by cyclists during their daily rides. It also elaborate theft alerts, sending push notifications to the mobile app in case of a theft attempt.

The results obtained by this thesis work demonstrate that monitoring air quality by using bicycles as probes is a perfectly achievable objective. SwarmBike is a valid prototype which could be improved and extended in order to be used in real-life scenarios, although some weak points and issues should be addressed in the future.

Acknowledgements

I want to thank Luca Zanette, who assisted me with IC soldering. Andrea Valenzano, who helped me deal with server side problems. Melissa Giancola, who helped me edit this thesis. Roberta Giannantonio, who conducted the survey about smart bike features. Ilaria Zonda, who realized the artworks for the survey, the maps and the mobile application.

Special thanks also to my corporate tutor Pino Castrogiovanni, who mentored me during all the phases of the project, and to my advisor prof. Fulvio Corno for his support.

Contents

Chapter 1

Introduction

1.1 Crowd Sensing and New IoT Services for Smart Cities

In recent times, the world has seen the rising of the so called *Internet of Things* (IoT). Many common objects, from wristbands to cars, from household appliances to ambient sensors, can be now connected to the cyberspace, offering many new features and development opportunities. The contemporary enormous success of smart phones, which are devices equipped with many sensors (such as GPS, accelerometer, camera, etc.), has lead to the idea that sensor data collected by the users can be exploited in various ways, opening new scenarios especially for environmental monitoring.

Crowd sensing is a new sensing paradigm which aims to exploit the power of the crowd, together with the connectivity and sensing capabilities of smart phones and IoT devices. Leveraging on those two factors, a large sensor network gathering data from the environment can be built, where a crowd of volunteers actively participate in the collection of these data. This is especially true for our cities, where many people can sense environmental data during their every day movements. Crowd sensing can be therefore a key enabler for the "smart city" vision, aiming at better exploiting new ICT technologies in order to increase efficiency, reduce expenses, and improve the overall quality of life in our urban environments.

The crowd-sensing approach is perfectly compatible with the current trends, where the final users are not only content consumers, but they can also provide and share information. "Smart citizens" and their devices can be used to collect many precious information at a relatively low cost. Those people who act as sensors can be referred as *smart sensors*, therefore the resulting sensor network can be defined as a *smart sensor network* [52].

1.2 Context of the SwarmBike Project

Air pollution is blamed to be one of the causes of many dangerous diseases, such as lung cancer, therefore the need for air quality monitoring is being increasingly recognized. Focusing on urban environments, people's mobility is one of the main causes of air pollution, as they often tend to use their private vehicles (such as cars and scooters) for their movements. Motor vehicles releases many hazardous substances in the atmosphere, thus increasing pollution levels and leading to a conditions detrimental to the quality of life of local inhabitants, with adverse effects on public health and the environment. The proposition of the SwarmBike project aims at making people aware of this problem, offering new services which may encourage them to use sustainable way of moving, especially focusing on bicycles. TRACE [54] is an example of a European Union's Horizon 2020 project, whose mission is to better plan and promote walking and cycling in cities. In fact, some studies have calculated that increasing the usage of bikes for personal mobility would lead to a healthier population, to a non negligible reduction of pollution and to a huge energy and resource saving [18].

1.3 SwarmBike Project General Objectives

The objective of this work is to design and prototype an IoT air quality monitoring system, based on sensors that can be easily mounted and used by cyclists on their bikes. The main idea is to exploit bikes as mobile sensor nodes which can gather data from the environment. This way air pollution can be sensed either during trips, enabling the estimation of the quality of the air breathed by the cyclist, or when the bike is parked, permitting to collect better statistical air quality data in a certain zone of the city. One of the ideas was to exploit bike-sharing services, however for the first prototype it was decided to focus on private bicycles.

Leveraging on the collected data, some services could be offered to both cyclists (e.g. suggestions about less polluted paths) and to local public institutions (e.g. better city air quality maps than now, since currently they are based on data collected by few fixed station distributed among the city).

Thesis main objective is the design and prototyping of a device capable of supporting the previously described air pollution monitoring service scenarios. At the same time, given the fact that a high amount of users is needed for the system to work properly, it cannot rely only on volunteers. Therefore, the other objective of this thesis is to study some possible ways to encourage people to use the system. It was decided to add other secondary features to the device, in order to increase users' involvement. A survey has been conducted among some cyclists (mainly Politecnico students and TIM employees) to discover which features are most desired on a smart bike. The results of the survey have been used as a starting point for selecting which features would have been implemented on the device.

1.4 SwarmBike Device Requirements and Features

As stated before, the objective is to build a prototype of an air quality monitor, which can be acquired by cyclists and easily mounted on their private bikes. To achieve that, the following device requirements must be met:

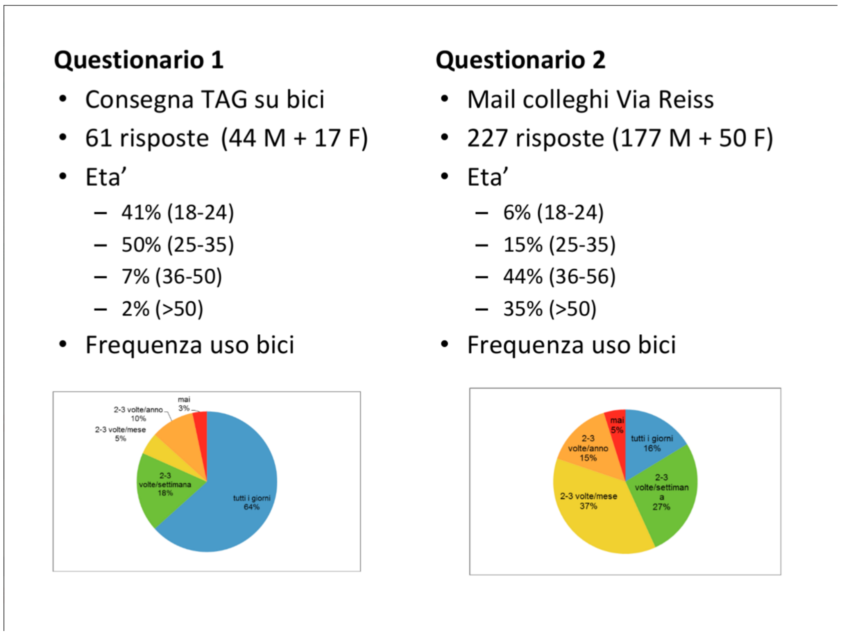
- Its weight and its size must be acceptable for being carried on a bike
- It must run on battery and an acceptable battery life must be achieved, therefore every components should be as much power efficient as possible
- It must have a GPS module to geotag the acquired data
- It must be easily managed and configured through a smartphone app
- It must have a not excessively high cost

To increase users' involvement, it was decided to implement some other features to the SwarmBike device, in order to make it more appealing for cyclists. Those features were selected based on the results of two surveys conducted among some Politecnico students and TIM employees. To deliver the first survey, 500 tags with a printed QR code (which pointed to the survey's link) were attached to the bikes parked in the Politecnico courtyard. The second one, on the other hand, was delivered to TIM employees by e-mail. The results are presented in figure 1.1 and 1.2

It is clear that the results of both surveys are similar, as shown by the first seven positions of the two rankings highlighted in figure 1.2. Remote geolocation of the bike is the most wanted feature, closely followed by an anti-theft system which can send notifications to a smartphone. Direction indicators are a largely requested feature too and so are brake lights. It is worth noticing that receiving information about air pollution levels is also a highly requested feature.

Direction indicators and brake lights are just “hardware” features, therefore it was decided not to implement them yet, but to leave them for future work. Consequently, based on the results of the survey and besides the sensing of hazardous air pollutants, the following features have been selected for the implementation of a first prototype of the device:

- Remote visualization of the device's position on a map in real-time, either through a web interface or on a mobile app. For this, a built-in cellular connectivity is necessary.
- Accelerometer based anti-theft system, with real-time notifications to a smartphone app



Questionario 1

- Consegna TAG su bici
- 61 risposte (44 M + 17 F)
- Eta'
  - 41% (18-24)
  - 50% (25-35)
  - 7% (36-50)
  - 2% (>50)
- Frequenza uso bici

mai
3%
2-3 volte/anno
10%
2-3 volte/mese
5%
2-3
volte/settimana
18%
tutti i giorni
64%

Questionario 2

- Mail colleghi Via Reiss
- 227 risposte (177 M + 50 F)
- Eta'
  - 6% (18-24)
  - 15% (25-35)
  - 44% (36-56)
  - 35% (>50)
- Frequenza uso bici

mai
5%
tutti i giorni
16%
2-3
volte/anno
15%
2-3
volte/settiman
a
27%
2-3 volte/mese
37%

Figure 1.1. People who answered the survey

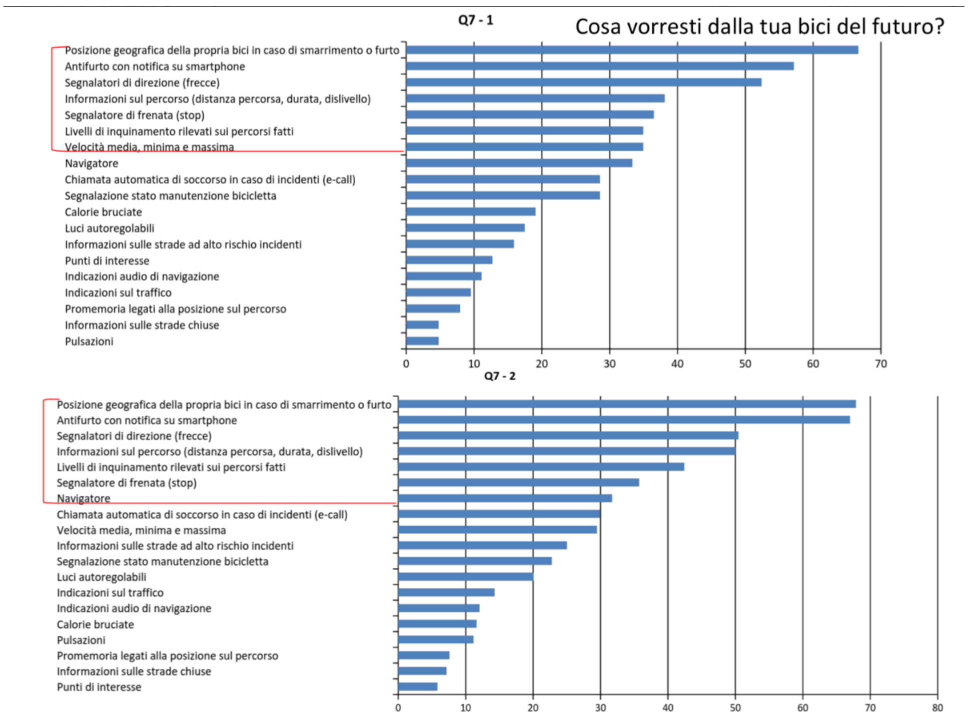


Figure 1.2. Results of the survey

1.5 A Typical Usage Scenario

SwarmBike device scenario is reported as follows:

1. The user takes his bike and turns on the device
2. During the ride, the device acquires data from all its sensors (hazardous gases, temperature, humidity, pressure [1], accelerometer). Information is geotagged using the GPS position and periodically sent to the cloud using the built-in cellular connectivity
3. When the bike is parked, the anti-theft system is activated through the mobile app: if motion is detected by the accelerometer, a notification is sent to the smartphone
4. In case of theft or loss, the position of the device can be viewed remotely either via a web interface or the mobile app
5. Ideally, all the sensed data should be employed to build air pollution maps, which could be used to offer users some services, such as suggestions about less polluted paths and pollution optimized satellite navigation

[1]It is necessary to track also temperature, humidity and pressure because those values can influence the reading of the gas sensor

Chapter 2

Air Quality Sensing

2.1 The Need for Stricter Air Quality Monitoring

The need for stricter air quality monitoring is being increasingly recognized. It is ascertained that a high number of diseases, especially respiratory related ones, are associated to air pollution. Furthermore, recent studies have revealed the importance of micro-level pollution information, including human personal exposure and acute exposure to air pollutants, because of their effects on health [57]. Among the air pollution related illnesses we can remind asthma, a chronic and occasionally debilitating inflammatory disease of the airways, which can be triggered by noxious gases and particulate matter; Chronic Obstructive Pulmonary Diseases (COPD), such as bronchitis and emphysema, which are caused by cigarette smoke and cars exhausts; lung cancer, which is associated to long-term exposure to fine particulate matter [56] [38]. Effects of air pollution have been observed also on the cardiovascular system, on the nervous system and on the urinary system [13].

Consequently, in last times there has been the rising of many projects which aimed at sensing air quality with a high spatiotemporal density. Some of them are: "The Air Quality Egg", an egg-shaped device which track noxious gases [7]; "Smart Citizen", an Arduino based platform to sense environmental data through participatory processes of people in the cities [5]; "SensorBox", a portable sensor node developed by the EveryAware project which can track many hazardous substances [8]; TZOA, a wearable device which tracks particulate matter (PM) [55]. It is clear that the SwarmBike service proposal fits perfectly with this trend, where final users provide air quality information through the devices they own. This is particularly important for cyclists, because they are more exposed to pollutants compared to others. Therefore it would be worthwhile to give them information which could help to reduce the quantity of hazardous substances absorbed (e.g. suggestions on less polluted paths).

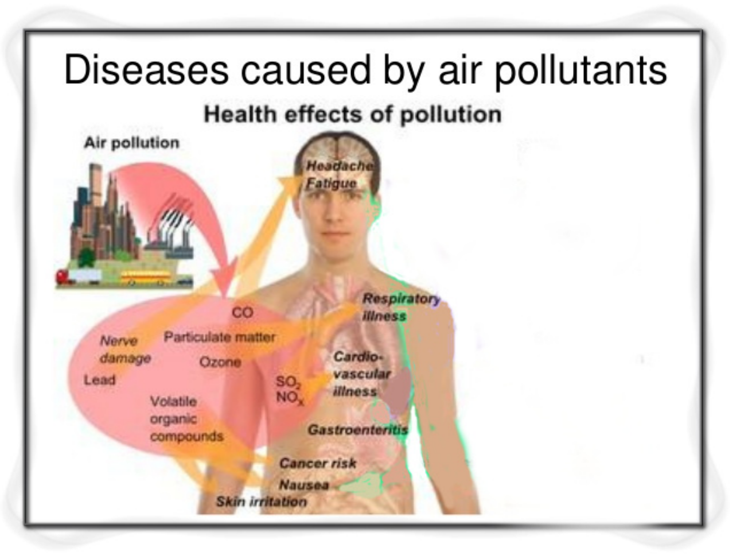


Figure 2.1. Effects of air pollution on human health

2.2 Hazardous air pollutants

There are many dangerous substances which form air pollution. The majority of them are produced by man related activities, such as factories, power plants, car traffic, house heating. However there are also some natural sources like volcanoes and plants. The main air pollutants can be roughly divided as following [13]:

- Gaseous pollutants (e.g. carbon monoxide)
- Particulate matter
- Persistent organic pollutants (e.g. dioxins)
- Heavy metals (e.g. lead, mercury)

2.2.1 Gaseous Pollutants

Gaseous pollutants heavily contribute to air pollution and are mainly produced by the combustion of fossil fuels. The main hazardous substances in gaseous form are:

- Carbon Monoxide
- Nitrogen Dioxide
- Tropospheric Ozone
- Sulfur Dioxide
- Volatile Organic Compounds (VOCs)

Carbon Monoxide

Carbon Monoxide (CO) is a colorless, odorless and tasteless gas. It forms in combustions when there is not enough oxygen to produce carbon dioxide (CO2), such as when operating a stove or an internal combustion engine in an enclosed space [53]. It is mainly produced when hydrocarbon fuels (such as charcoal or diesel) are burned, however it is also emitted by natural sources (e.g. volcanoes).

It is dangerous even at a very low concentration (35ppm). Its toxicity is due to the fact that it binds to the haemoglobin molecule in the blood much faster than oxygen, forming carboxyhaemoglobin. Therefore, the blood becomes less efficient at carrying oxygen. The symptoms of a CO intoxication are usually headache, nausea, dizziness and general weakness. If the level of carboxyhaemoglobin in the blood reaches 50% there are also seizure, syncope, coma and in some cases death. The average level of CO in the air is quite low, usually ranging from 0 to 10ppm, therefore it is not directly dangerous to humans (even if locally, e.g. in traffic jams, the concentration can be higher). However, carbon monoxide level is correlated to other pollutants like nitrogen dioxide [37], consequently it can be used as an "index" of air quality. Carbon monoxide is also involved in the reactions which produce tropospheric ozone.

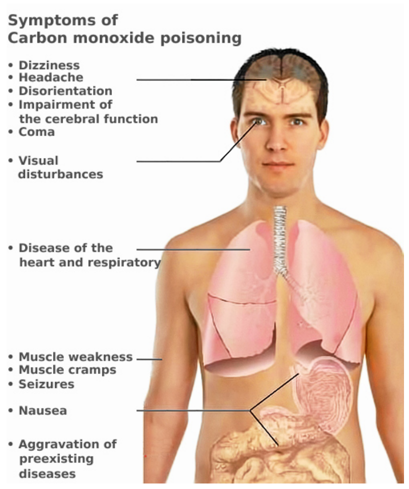

Symptoms of
Carbon monoxide poisoning
• Dizziness
• Headache
• Disorientation
• Impairment of the cerebral function
• Coma
• Visual disturbances
• Disease of the heart and respiratory
• Muscle weakness
• Muscle cramps
• Seizures
• Nausea
• Aggravation of preexisting diseases

Figure 2.2. Symptoms of CO intoxication

Nitrogen Dioxide

Nitrogen Dioxide (NO2) is a reddish and acrid gas. It is mainly produced in internal combustion engines burning fossil fuels, such as cars, power plants and house heaters. Direct exposure to skin or eyes can cause irritation and burns. At a concentration higher than 10ppm it causes irritation to throat and nose, at more than 50ppm it leads to edema and bronchitis, at over than 100ppm it causes death due to asphyxiation from fluid in the lungs. Long term exposure to relatively low levels of nitrogen dioxide is believed to cause bronchitis and asthma, especially in children [32]. NO2 is also one of the precursors of tropospheric ozone.

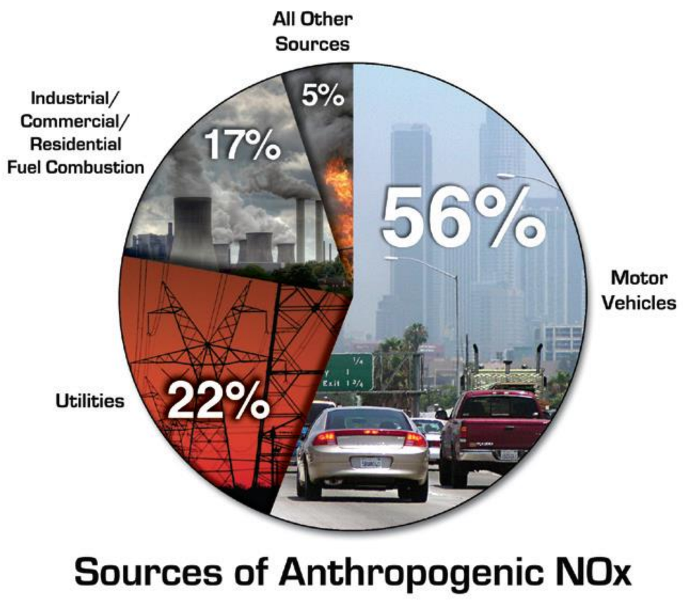


Figure 2.3. Sources of man made Nitrogen Oxides

Tropospheric Ozone

Ozone (O3) is a blueish gas with a typical pungent smell. It is mainly present in the stratosphere, where its action is fundamental to life, because it absorbs a large quantity of ultra-violet rays which would be highly dangerous for biological processes. However, the presence of ozone in the troposphere (i.e. the part of atmosphere between 0 and 10Km of height) is harmful because of its toxicity.

Tropospheric ozone is produced by the reaction of nitrogen oxides, carbon monoxide and volatile organic compounds (VOCs) with ultraviolet rays. Those chemicals are in fact called ozone precursors. At the concentration level common in some urban zones (0.2 ppm), ozone can cause irritation of the respiratory system, asthma, harm of lung function. Its presence is also linked to premature deaths [32], and it is also known to be harmful for vegetables and agriculture.

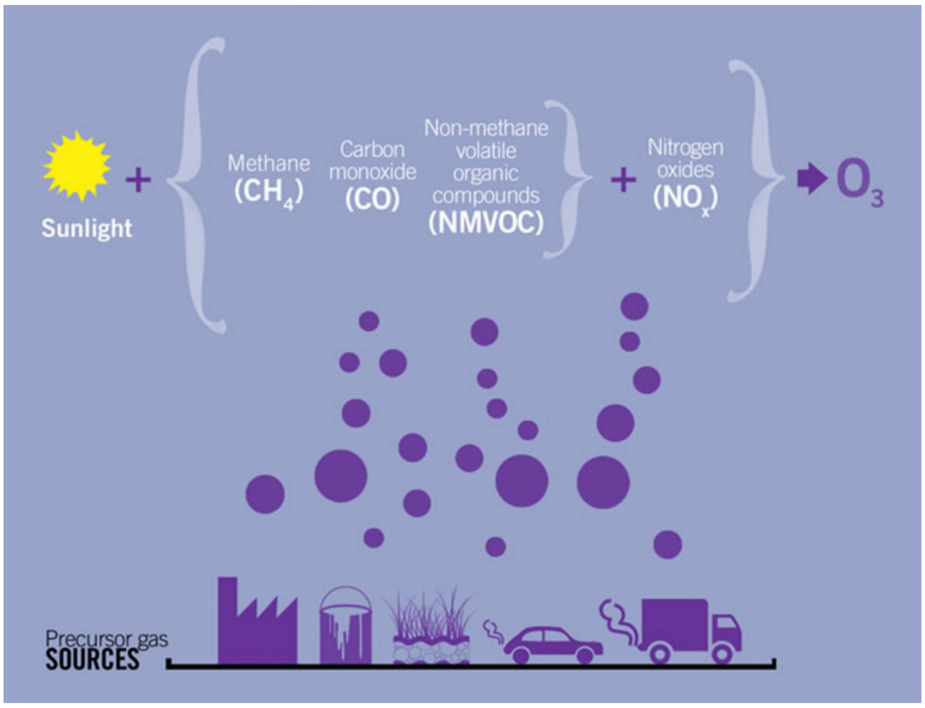


Figure 2.4. Process which leads to the forming of tropospheric ozone

Volatile Organic Compounds

Volatile Organic Compounds (VOCs) are organic chemicals which are volatile (i.e. with a high tendency to vaporize) at room temperature. The principal sources of VOCs are biological (mainly plants). However, a large part of them is anthropogenic, such as acetone, chlorofluorocarbons, fossil fuels (e.g. methane), benzene, formaldehyde. Anthropogenic VOCs can be toxic: symptoms include allergies, irritation of airways, headache, nausea, damage to the liver. Some of them are suspected to cause cancer. VOCs are also involved, with carbon monoxide and nitrogen dioxide, in the reactions which form tropospheric ozone.

Sulfur Dioxide

Sulfur Dioxide (SO2) is a toxic gas with a pungent and irritating smell. It is mainly produced by industrial processes (e.g. energy production in power plants), but it is also released naturally in volcanic eruptions. It is a highly dangerous pollutant and has a big impact on human health. It becomes dangerous to human in concentrations higher than 5pppm. Inhaling sulfur dioxide is associated with increased respiratory symptoms and disease, difficulty in breathing, and premature death. It is also known for being the precursor of acid rains and atmosphere particulate.

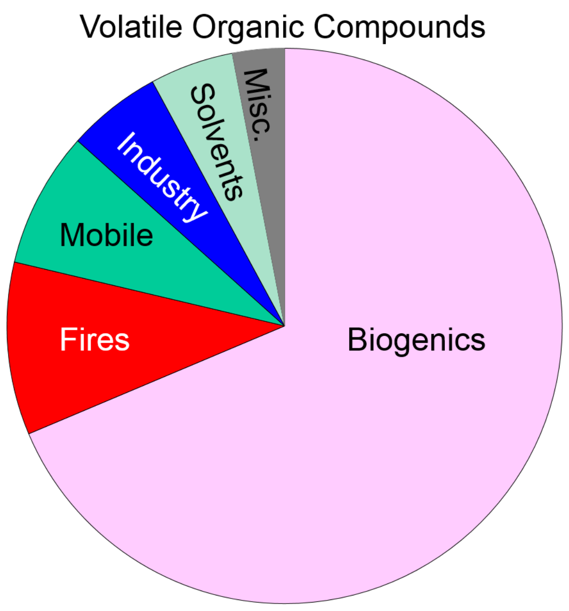


Figure 2.5. Sources of Volatile Organic Compounds

2.2.2 Particulate Matter

Particulate Matter (PM) refers to a type of air pollutants which consist of a complex mixture of particles suspended in the air, with various size and composition. They are produced by both natural and anthropogenic activities. The main sources of particulate pollution are industrial activities, power plants, motor vehicles, construction activity, fires, and natural windblown dust. The size of particles varies and they have been divided into three categories:

- *Ultrafine*: particles with a diameter smaller than 0.1μm
- *Fine*: particles with a diameter smaller than 1μm
- *Coarse*:particles with a diameter larger than 1μm

Additionally, particles with a diameter smaller than 2.5μm are called PM2.5, and the ones with a diameter smaller than 10μm are called PM10.

The composition of PM varies, as they can absorb and transfer a multitude of pollutants. However, their major components are metals, organic compounds, material of biologic origin, ions, reactive gases, and the particle carbon core [13].

Depending on their size, particles will deposit on a different respiratory tract: PM10 will usually deposit on the upper tract, while finer particles will reach lungs and alveoli, therefore they are more hazardous than larger ones, in terms of mortality and cardiovascular and respiratory effects [13]. In 2013, a European study has stated that there is no safe level of PM and that for every increase of 10 $\mu g/m3$ in PM10, the lung cancer rate rose 22%. The finer PM2.5 are particularly deadly, with a 36% increase in lung cancer per 10 $\mu g/m3$ probably because they can penetrate deeper into the lungs [36]. Ultra-fine particles are also able to provoke alveolar inflammation, with release of mediators capable, in susceptible individuals, of causing exacerbations of lung disease and of increasing blood coagulability, which can lead to cardiovascular deaths [38].

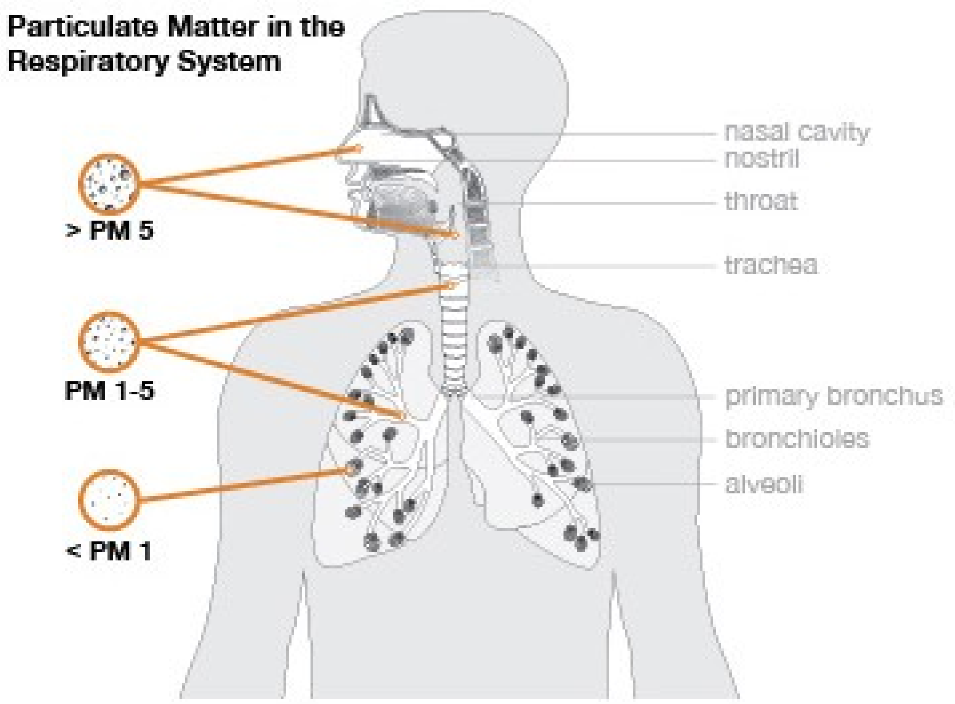


Figure 2.6. Where particulate matter deposit depending on size

2.2.3 Persistent Organic Pollutants

Persistent Organic Pollutants (POPs) are a group of toxic chemicals which persist in the environment for long periods of time. Their effects have a big impact on vegetable and animal health because they move up through the food chain (biomagnification). Often they are referred as *dioxins* or *dioxins-like compounds*. Dioxins are formed during incomplete combustion and whenever materials containing chlorine, such as plastics, are burned. Emitted in the atmosphere, dioxins tend to deposit on soil and water. Most dioxins in plants come from air and dust or pesticides and enter the food chain where they bio-accumulate [13].

Dioxins are highly toxic to humans (especially children) and can cause reproductive and developmental problems, damage the immune system, interfere with hormones and also cause cancer [31].

2.2.4 Heavy Metals

Heavy metals include basic metal elements such as lead, arsenic, mercury, cadmium, silver, nickel, vanadium, chromium and manganese. They are natural components of the earth's crust, which cannot be degraded or destroyed and can be transported by air, entering water and human food supply. In addition, they reach the environment through a wide variety of sources, including combustion, waste water discharges and industrial processes. Most heavy metals are dangerous because they tend to bio-accumulate in the human body [13]. They can cause many health problems, such as vomit, diarrhea, skin-irritation, cardiovascular diseases, neurological illnesses and respiratory diseases.

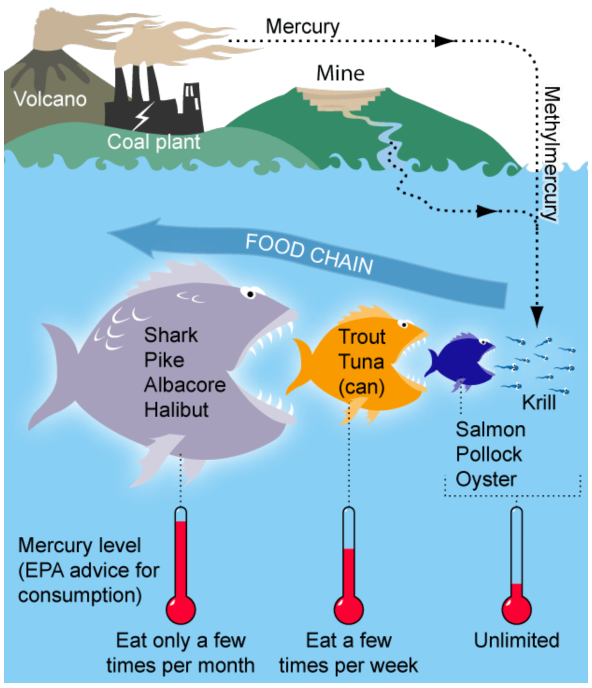


Figure 2.7. Pollution due to mercury, a heavy metal. It is illustrated how it moves up the food chain

2.3 Air Quality Sensors

The objective is to track pollution in urban areas, therefore the focus was only on air pollutants which are related to urban activities, such as traffic or house heating. Consequently only sensors which can track Carbon Monoxide, Nitrogen Dioxide, Ozone and Particulate Matter have been analyzed. There are various types of sensors for monitoring those air pollutants, ranging in prices, technology and accuracy. The sensors have been evaluated based on the requirements of the system:

- They must be portable, therefore limited in size and weight
- They must be low cost, but still their accuracy should be acceptable
- They must be battery operable, therefore they should have a low power consumption

2.3.1 Gas Sensors

For gas sensing, two technologies have been evaluated: semi-conductor (also known as metal-oxide or solid-state) sensors and electrochemical sensors.

Semi-conductor Gas Sensors

The working principle of this type of sensors is based on the P-N junctions of semiconductors, which are sensitive to environmental gases. A semiconductor sensor is formed by one or more metal-oxides, such as tin oxide or aluminum oxide (depending on which gas has to be sensed) and a heating element.

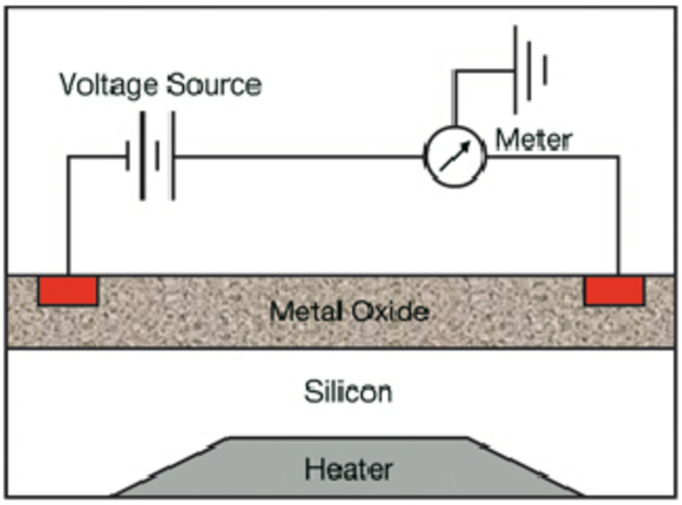


Figure 2.8. Schema of a semiconductor gas sensor

When the metal oxides are exposed to the target gases, they will dissociate into charged ions or complexes, consequently the electrons will accumulate on the surface of the oxides. This accumulation of electrons will change the conductivity of the metal oxides, thus the concentration of the gases can be estimated measuring the resistance of the oxides, applying a voltage between two electrodes. The estimation can be made computing the ratio between the measured resistance in presence of the gas and the resistance measured in clean air. In order to facilitate the reactions and strengthen the output signal, a heating element is used. The heater is also used to regulate the temperature, because the conductivity change of the oxides due to the gases can range depending on the temperature [57].

The main advantage of semi-conductor sensors is their cost: usually a metal-oxide sensor has a price not higher than 10 Euro. Being inexpensive they can be deployed in very large quantities. They also need only few external components (only some resistors), therefore they are easy to configure and to use. On the other hand, they have some big disadvantages too. One of these is the heating element: many times the sensors need to reach a temperature near to 300°C to work properly, consequently the power consumption can be high, making them unfit to be battery operated. Another disadvantage is their accuracy: the relationship between output signal and gas concentration is not linear and much influenced by relative humidity and temperature, thus it is difficult to estimate accurately the pollutant levels. Furthermore many sensors exhibit cross-sensitivity to various gases, making still harder to compute a precise estimation. Finally, metal-oxide sensors need to be calibrated, because every specimen shows a different sensibility compared to the others. Calibration consists in exposing the sensor to a known concentration of polluting gas, adjusting the parameters of the sensor in order to minimize the difference between the known concentration and the output signal.

Electrochemical Sensors

An electrochemical gas sensor (also known as galvanic cell or amperometric sensor) is composed by a cell which contains three electrodes in contact with an electrolyte. Those three electrodes are named the working electrode (WE), the counter electrode (CE) and the reference electrode (RE). The gas enters the cell through a gas-permeable membrane. When it reaches the WE, a chemical reaction occurs: it can be either oxidation (loss of electrons) or reduction (gain of electrons). The reaction type depends on what the target gas of the sensor is. If oxidation occurs at the working electrode, the complimentary reaction (reduction) occurs at the counter electrode and vice-versa.

For instance, in the case of a carbon monoxide sensor, the following oxidation reaction occurs at the working electrode:

$$CO + H_2O \rightarrow CO_2 + 2H^+ + 2e^- \tag{2.1}$$

while at the counter electrode the reduction reaction is:

$$\frac{1}{2}O_2 + 2H^+ + 2e^- \rightarrow H_2O \tag{2.2}$$

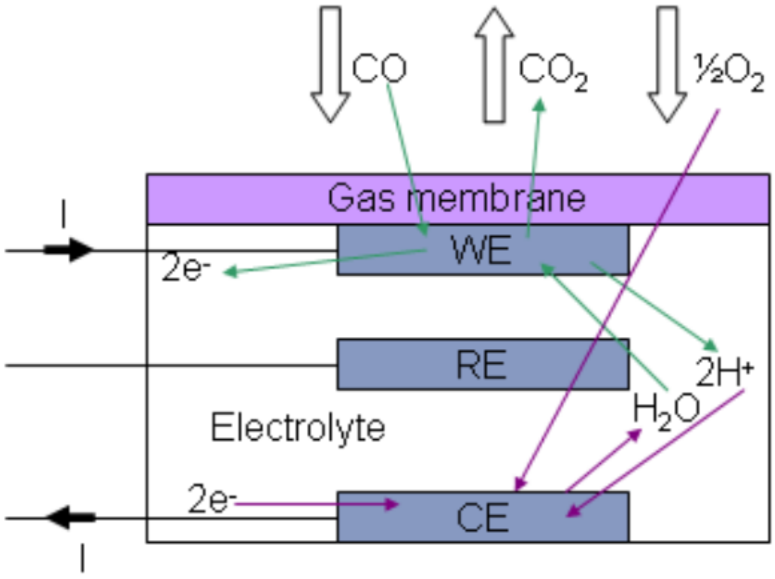


Figure 2.9. Schema of a carbon monoxide electrochemical sensor

Some electrical charges travel through the electrolyte. To keep the total charge constant in the electrolyte, some electrons flow through the circuitry connected to the sensor between the working electrode and the counter electrode. In the case of a CO sensor, H^+ moves in the electrolyte from the WE to the CE , while in the external circuitry e^- moves from the WE to the CE. Effectively, the sensor generates a current flowing from the CE to the WE, which is proportional to the gas concentration. During operation, the working electrode is maintained at a fixed potential while the potential of the counter electrode is allowed to float. In clean air, the counter electrode has the same potential as the working electrode, but this changes as a current between the electrodes is generated in the presence of the sensed gas. To ensure linearity and sensitivity, the working electrode potential is set and maintained relative to the reference electrode, which is set at a constant and stable potential. At this electrode no chemical reactions occur and no current flows [50] [20].

To operate an electrochemical sensor, a voltage has to be applied between the WE and the RE according to the specification of the sensor. If the required voltage is 0 Volt, the sensor is *unbiased*, otherwise it is *biased*. The current generated on the WE has to be balanced by the electronics on the CE. An operational amplifier in trans-impedance configuration must be used to convert the current generated by the sensor into a voltage that can be read by the Analog-to-Digital converter of a microcontroller. A circuit which provides all these features is called **potentiostat**.

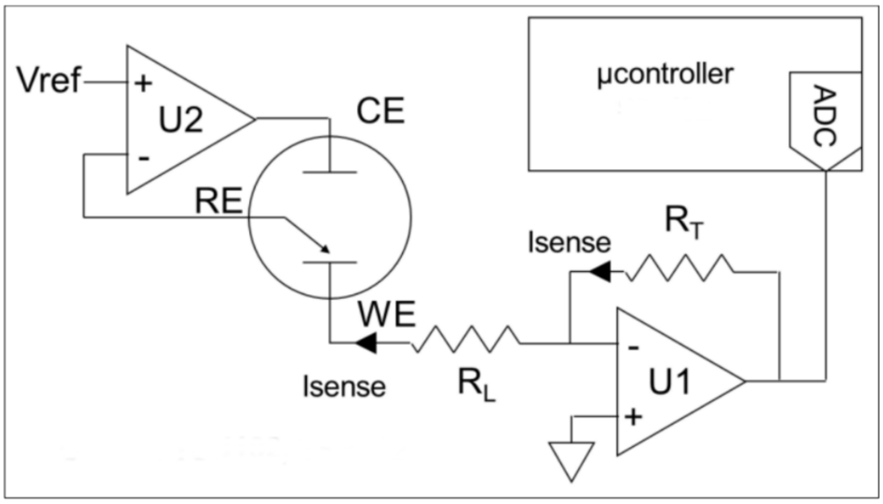


Figure 2.10. Schema of a potentiostat circuit

Figure 2.10 shows how a potentiostat circuit works: the U1 operational amplifier in trans-impedance configuration converts the current generated by the sensor into a voltage thanks to R_T. The output voltage sensed by the ADC is therefore $R_T \times$Isense. Depending on the sensed gas, the current can be either positive or negative because of the different chemical reactions (oxidation or reduction) which occur at the WE. U1 also ensures that the sensor sees the specified load resistor (R_L), whose value depends on the type of sensor, and keeps the working electrode at a fixed potential. The U2 op-amp fixes the specified voltage between the WE and the RE ($V_{WE} - V_{RE} = -V_{ref}$). It also ensures that no current flows through the RE while providing the right amount of current to the CE to compensate the one on the WE. The concentration of the polluting gas can be computed by the microcontroller on the basis of the sensed output voltage, which will be proportional to Isense and thus to the pollutant level [50].

The principal advantage of an electrochemical sensor is power consumption: the sensor itself requires no power, the output current is produced by the chemical reactions which happen at its electrodes. Power is only required by the potentiostat circuit, however there are many low-power operational amplifiers on the market which can be chosen to build the circuit. Furthermore there are some commercial potentiostats which are integrated on a single chip ready to be used, whose current consumption is in the range of microAmperes. Consequently, electrochemical sensors are highly fit to be battery operated.

They also have a good accuracy: they can detect gas concentration even in the parts-per-billion range, similarly to professional gas sensors, and the output current increases linearly with pollutant levels therefore it is easy to estimate the concentration. Cross-sensitivity to other gases is usually limited and can be neglected [20].

Electrochemical sensors are normally calibrated at the factory: a sensitivity code, which indicates what is the output current increasing for every ppm (or ppb) of gas concentration, is usually printed on each specimen therefore they are easy and immediately ready to be used. The influence of temperature and relative humidity is normally indicated on the datasheet, consequently the estimation can be easily compensated by the software.

A disadvantage of electrochemical sensors was their cost, until some years ago their price was no lower than \$100. However, with the progress of technology new low cost electrochemical sensors have appeared on the market, with prices floating at around \$20, making possible to build an electrochemical sensing node at a cost not excessively higher than a semi-conductor one.

2.3.2 Particulate Matter Sensors

The measurement of particulate matter (PM) is not straightforward and there are many techniques available for measuring the concentrations of PM. Due to the complex nature of PM, different measurement techniques could give different results. The available techniques for measuring the concentration of PM can be grouped into two categories: the first consists in directly reading instrument which provides continuous measurements of the concentration of PM in ambient air. The other one is filters-based gravimetric sampler, which collects the PM onto a filter that needs to be weighted periodically in lab [57]. It is obvious that for a device like SwarmBike weighting periodically a filter is not possible, therefore only the first type of sensors have been evaluated. Those ones are mainly based on optical analysis.

Optical PM Analyzers

The optical analyzers exploit the interaction between environmental particulate matter and light. They can be grouped into three categories: sensors based on light scattering, sensors based on direct imaging and nephelometers.

Light scattering sensors uses a high-energy laser as light source. When a particle enters the detection chamber, the laser light is scattered and this event can be measured by a photo detector. By analyzing the intensity of the scattered light, the size and the number of the particles can be estimated.

On direct imaging sensors, a halogen light is used as source. The shadow of the particles in the detection chamber is projected on a high resolution camera which records a video. The video is then analyzed by a software which can detect the size, the count, the weight, the shape and also the color of the particles.

Nephelometers are similar to light scattering sensors and exploits the same principle. The difference is that they uses an infrared LED as light source. With nephelometers both size and count of the particles can be estimated [57].

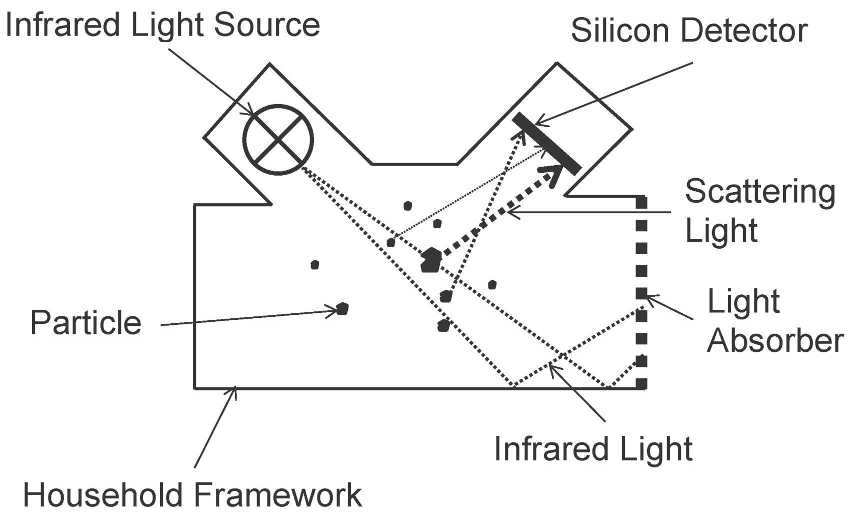


Figure 2.11. A basic nephelometer

It is clear that, needing a constant light source, these types of sensors can have a high power consumption. Especially direct imaging analyzers, having a camera which constantly records and a software which constantly analyze the video, need much power. However, instruments like portable nephelometers, which use infrared LEDs as source are not much power-hungry, with a current consumption in the order of milliAmperes. That type of consumption can nonetheless be problematic for battery operated devices. The accuracy of optical PM analyzers is however not excellent (with the exception of direct imaging sensors, which are obviously not suitable for low-cost portable devices as SwarmBike). Nevertheless, light scattering sensors and nephelometers are widely used in hand-held monitoring devices because of their small size, light weight, low cost and instantaneous measuring capabilities [57].

2.4 Wireless Sensor Network for Air Pollution Monitoring

Recent studies have revealed the importance of micro-level pollution information, including human personal exposure and acute exposure to air pollutants. That level of information is hardly achievable with conventional air pollution monitoring systems, such as public air quality monitors. Therefore, new ways of collecting air pollution data have appeared. These are called "the Next Generation Air Pollution Monitoring System" (TNGAPMS) and have achieved significant breakthroughs by utilizing the advance sensing technologies and Wireless Sensor Network (WSN). Sensor networks used to monitor air quality in urban areas can be classified into three major categories [57]:

- Static Sensor Network (SSN)
- Community Sensor Network (CSN)
- Vehicle Sensor Network (VSN)

2.4.1 Static Sensor Network

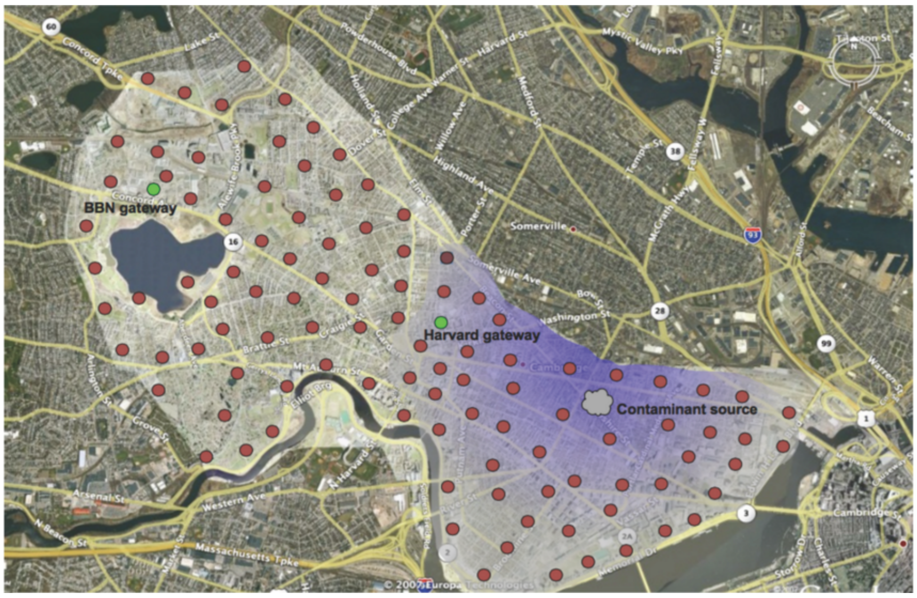


Figure 2.12. Example of SSN deployed in Cambridge (MA)

In SSN systems, the sensor nodes are typically mounted on fixed locations, such as streetlight or traffic light poles. By utilizing the low-cost ambient sensors, the number of sensor nodes in SSN systems is much larger than that in the conventional monitoring systems. Air pollution information with high spatio-temporal resolution is then achievable.

SSN have many advantages: firstly, being mounted on fixed locations, they can run on large battery or receive energy from power lines, thus they do not have strict energy constraints. They also have loose constraints about weight and size because they are not mobile and there is no need for GPS and geolocation. Having a fixed topology, SSN have no problems of connectivity, and the accuracy of the acquired data is high, because sensors can be easily and periodically maintained by professionals.

On the other hand, the placement of sensor nodes must be selected accurately, because air pollution is highly dependent on the location. To obtain information with a high spatio-temporal resolution, a large number of sensor nodes is needed and this can increase the cost. Since nodes are not mobile, some resources are wasted because there is no point in continuously sampling pollutants at a fixed location, therefore the nodes spend the majority of their time in a sleep state [57].

2.4.2 Community Sensor Network

In Community (or crowd-sensing) Sensor Network systems, the sensor nodes are typically carried by the users (usually volunteers or air quality enthusiasts). By using low-cost portable ambient sensors and the smart phones, users are able to acquire, analyze and share the local air pollution information.

CSN are cost-efficient: the connectivity and the computational power of smartphones are typically used, and a part of the cost of the sensing node could be apportioned to users. Being mobile, information with a high spatio-temporal resolution can be collected and a large geographic coverage is possible. Furthermore, data about public behavior can be acquired, making it possible to study the relationship between public behavior and pollution.

CSN however have also some drawbacks. Primarily, cheap sensor nodes are typically used in CSN, thus all the acquired information have a low accuracy. Users spend also much of their time indoors, lowering the quality of the sensed data. Moreover, the users are not typically able to calibrate and maintain the sensor nodes and this worsen the accuracy of the information. CSN also have privacy issues: the users may not want to share sensible information like their own location. Since they must be carried around, CSN nodes must also be lightweight, and because they run on battery they have to respect strict power constraints [57].

2.4.3 Vehicle Sensor Network

In Vehicle Sensor Network, the nodes are usually carried by public means of transportation, such as taxi, buses, tram or car sharing. Exploiting the mobility of vehicles and low-cost sensor nodes, VSN can acquire information with a high spatio-temporal resolution.

VSN nodes do not have strict power consumption constraints, because they can use the power provided by the carrier. Weight and size constraints are loose too, given the fact that the carrier is a vehicle. Consequently many assistance tools such as secondary sensors can be installed on the node, increasing data accuracy. The mobility is also high, therefore a good geographic coverage and a high spatial resolution can be achieved. Finally, the maintenance of the nodes is easily doable, because the vehicles can be driven to specific locations where professionals can maintain the nodes.

On the other hand, the mobility of VSN nodes can be controlled or semi-controlled, because public transportation vehicles typically follow pre-determined paths, thus some locations may be oversampled and some other locations may never be reached. Furthermore, another oversampling issue can happen if the vehicle gets trapped in a traffic jam or in a queue. Vehicles which carry the nodes must also be specially equipped, especially for providing the connectivity which is often cellular, therefore the system may be expensive [57].

2.5 Air Quality Sensing on SwarmBike

The analysis on air pollutants and on sensing technologies has been deeply dissected in order to choose the best solution for the device. It is clear that particulate matter is one of the most dangerous pollutant in urban areas and therefore it was one of the first choices when it was decided which sensors would have been put on board. The only affordable PM sensor on the market was Shinyei PPD42NS, a low cost nephelometer. However, that optical particulate sensor had some major drawbacks: according to multiple sources [35] [51], the sensor was noisy and the accuracy provided was low. Measurements could have been improved by mean of calibration and of an additional fan, but still its size and weight was not negligible and its current consumption, which according to the datasheet is 90mA [39], would have had an impact on the battery life. Consequently, the price-performance ratio of the sensor was not considered worthwhile and it was discarded.

For this reason, it was decided to track a polluting gas. Several semi-conductor gas sensors were evaluated, mainly because their cost was low. However their impact on battery life would have been non negligible and their accuracy would have been poor, therefore metal-oxide sensors were discarded too. Consequently, it was decided to use an electrochemical sensor to track air pollution. Electrochemical sensors were chosen because:

- They have an extremely low power consumption
- Their accuracy is satisfactory
- Their cost, although being higher than semi-conductor ones, is still acceptable (around 20 Euro)

At the time of the choice (December 2015), the easiest to acquire and most inexpensive electrochemical sensors which tracked polluting gases were carbon monoxide ones. Therefore, for the first SwarmBike prototype it was decided to monitor that particular hazardous gas. The chosen CO sensor was Nemoto NAP-505 [16], which was bought on Aliexpress at a price of $18. No calibration was needed, since this sensor is pre-calibrated at the factory and its sensitivity is printed on its case. Furthermore, the use of an electrochemical sensor fits perfectly with the current trends, which tend to prefer this type of pollution sensors over other technologies [20].

As stated before, electrochemical sensors need a potentiostat circuit to be operated. It was decided to use the Texas Instruments LMP91002 AFE potentiostat [12], a potentiostat integrated on a single chip, which guaranteed ultra low power consumption and an easy configuration and usage.

Being a prototype, the SwarmBike system cannot be considered a Wireless Sensor Network yet. However, ideally when the system will grow, its nodes might be considered as part of a Community Sensor Network because they would be carried by users. Nevertheless, they would not use the connectivity and the GPS provided by the smartphone but the built-in ones (as in Vehicle Sensor Network), therefore the nodes would continue to operate even when the user is indoors and the bike is parked, which is the opposite of CSN nodes.

Chapter 3

SwarmBike System Architecture

The architecture is presented in figure 3.1. The whole SwarmBike system has been designed to have three major components:

- The device itself
- A smartphone application
- A cloud backend

This thesis work has been focused particularly on the design and the implementation of a prototype of the device. The cloud backend, on the other hand, has been designed and implemented leveraging on some platforms and technologies already employed by TIM in previous projects, which have been re-adapted to be used with SwarmBike.

This chapter describes the communication technologies and protocols used by the SwarmBike system and provides a functional overview of each major component.

3.1 Communication Protocols

The three major components communicate with one another. The communication between the device and the cloud is done through the Internet, with the connectivity provided by a GPRS cellular modem, while the smartphone app uses the connectivity provided by the smartphone itself. The device and the mobile app on the other hand communicate through the Bluetooth Smart (also known as Bluetooth Low Energy) technology.

The protocol which is used by the device to send and receive information from the cloud is MQTT, an IoT specific publish/subscribe protocol. The communication between the mobile app and the cloud is done by mean of HTTP, using a REST interface and a websocket based protocol (DDP) to communicate with the cloud

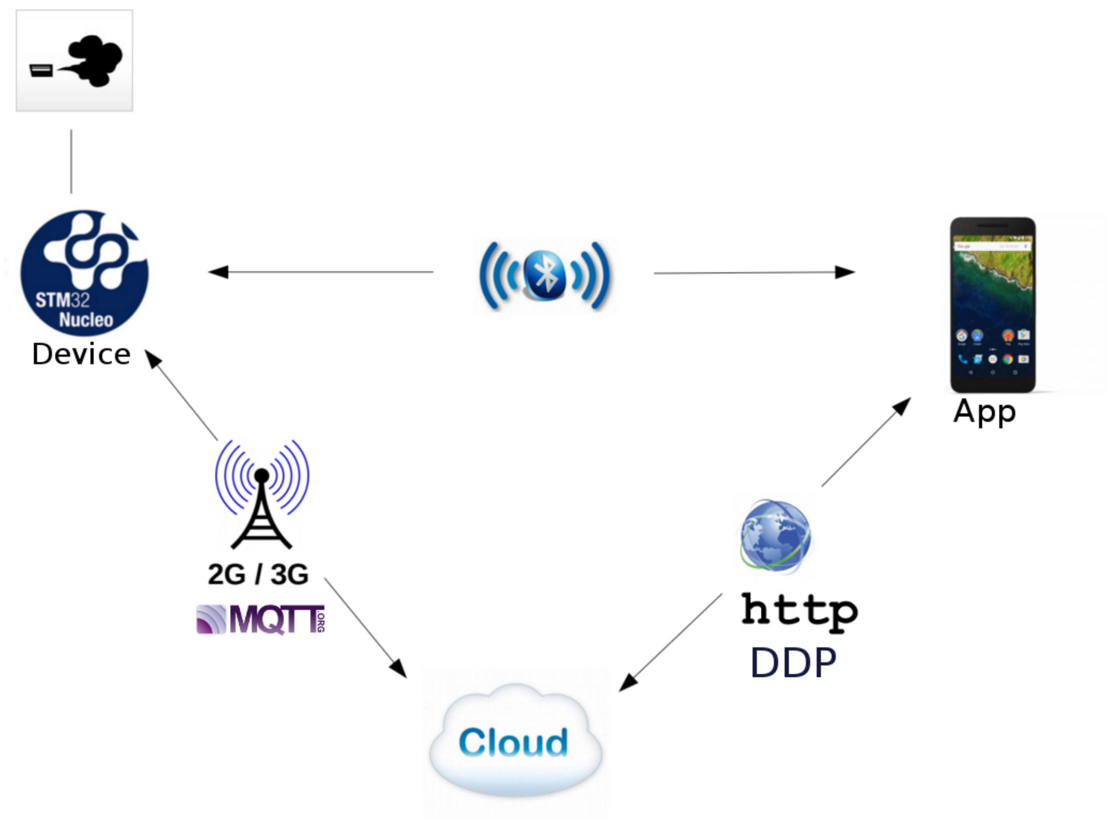


Figure 3.1. Architecture of SwarmBike system

backend platform, while the device and the mobile app uses a custom Bluetooth Low Energy service to exchange data.

3.1.1 MQ Telemetry Transport (MQTT)

The MQ Telemetry Transport (MQTT) is a lightweight broker-based publish/subscribe messaging protocol designed to be open, simple, lightweight and easy to implement. It was introduced in 1999 by IBM. It is a connection-oriented application protocol built on top of the TCP/IP stack. Its characteristics make it ideal to be used with systems whose connectivity has a limited bandwidth or is unreliable and for systems which are embedded and with limited computational power or memory resources. Therefore it fits perfectly the requirements of SwarmBike.

Every MQTT client can publish messages to a specific *topic*, which will be received by all the clients which are subscribed to that topic. The broker, which can be considered as the “MQTT server”, has the task of routing all the messages between publishers and subscribers.

MQTT offers three quality of service (QoS) levels for messages delivery:

- QoS 0: it is also called "at most once" and it offers the same reliability of the underlying TCP protocol. With this level messages can be lost or duplicated. It is particularly fit for context where the loss of a message is not crucial, such as in environmental sensors networks.
- QoS 1: it is also called "at least once", and it ensures that every message is always delivered. However message duplication can occur.
- QoS 2: this level is also known as "exactly once" and it ensures that every message is delivered without duplication. Ideal for billing systems, where the loss or the duplication of a message can lead to errors.

As for the security, MQTT supports Transport Layer Security (TLS), also known as Secure Socket Layer (SSL). If TLS is used, the communication can be encrypted and authenticated. For the client authentication, both plain text password and X.509 certificate authentication can be used [44] [10].

MQTT Control Packets

The MQTT protocol works by exchanging a series of MQTT Control Packets in a defined way. The format of a MQTT control packet is illustrated in figure 3.2.

Fixed header, present in all MQTT Control Packets
Variable header, present in some MQTT Control Packets
Payload, present in some MQTT Control Packets

Figure 3.2. Format of a MQTT control packet

Every control packet has a 2 bytes long fixed header, which contains some information about the message. The format of the MQTT fixed header is presented in figure 3.3. Depending on the type of packets, these may also have a variable header and a payload.

bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
byte 1	Message Type				DUP flag	QoS level		RETAIN
byte 2	Remaining Length							

Figure 3.3. Format of the MQTT fixed header

Message Type, presented as a 4 bit unsigned number, indicates the type of control packet. The most used types are resumed in table 3.1.

Name	Value	Flow	Description
CONNECT	1	client-to-broker	Connection request to the broker
CONNACK	2	broker-to-client	Connect acknowledgment
PUBLISH	3	both	Publish message
PUBACK	4	both	Publish acknowledgement
SUBSCRIBE	8	client-to-broker	Subscription request
SUBACK	9	broker-to-client	Subscription acknowledgement
UNSUBSCRIBE	10	sclient-to-broker	Unsubscription request
UNSUBACK	11	broker-to-client	Unsubscription acknowledgement
DISCONNECT	14	client-to-broker	Client is disconnecting

Table 3.1. Main MQTT control message types

The DUP flag is used in certain control packets (such as PUBLISH or SUBSCRIBE) to signal when the broker tries to deliver a duplicate message.

The QoS bits are used to indicate the quality of service level (0, 1 or 2).

The RETAIN flag is only used on PUBLISH messages. When a client sends a PUBLISH to a server, if the Retain flag is set to 1, the server holds on to the message after it has been delivered to the current subscribers. When a new subscription is established on a topic, the last retained message on that topic is sent to the subscriber with the Retain flag set. If there is no retained message, nothing is sent. This allows new subscribers to instantly receive data with the retained (or last known good) value.

The REMAINING LENGTH field indicates the number of bytes remaining within the current message, including data in the variable header and the payload [44].

Typical MQTT Message Flow

Figure 3.4 represents a typical MQTT message flow for publishing and subscribing with QoS 0.

1. Client 1 wants to connect, thus it sends a CONNECT packet to the MQTT broker

2. The broker sends the CONNACK packet to Client 1 to confirm the connection

3. Client 2 establishes a connection too, sending CONNECT to the broker

4. The broker responds with a CONNACK packet to Client 2 confirming the connection

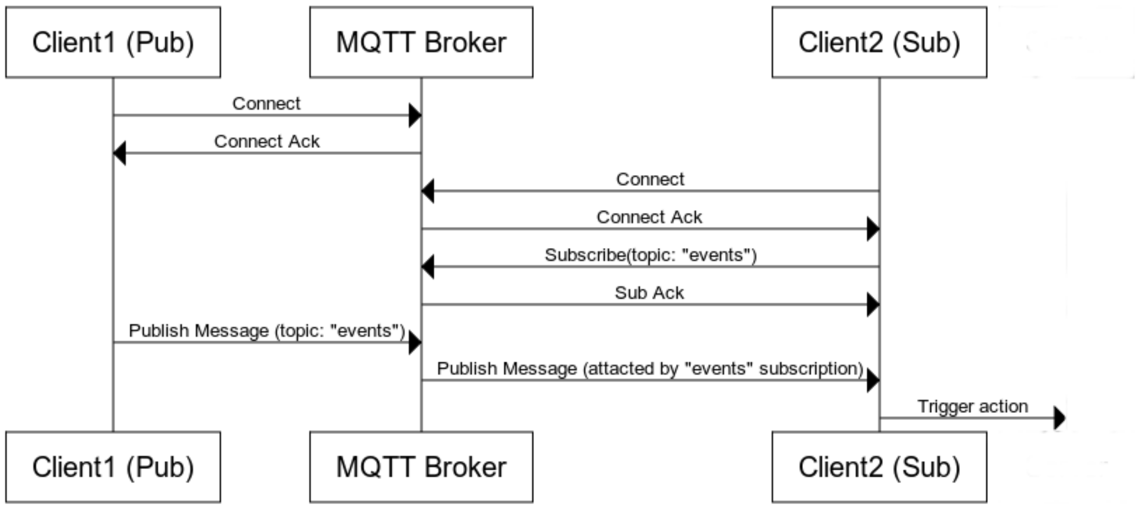


Figure 3.4. Typical message flow in MQTT (QoS 0)

5. Client 2 wants to subscribe to the topic "events", therefore it sends a SUBSCRIBE packet to the broker. The name of the topic is written in the payload of the packet

6. The broker answers with a SUBACK packet to confirm the subscription

7. Client 1 publish something on the topic "events", thus it sends a PUBLISH packet to the broker. The name of the topic and the content of the message are both placed in the payload of the packet

8. The broker delivers the message to all the clients subscribed to the topic "events" and consequently also to Client 2

9. Client 2 receives the message, and in this example it triggers an action

3.1.2 Bluetooth Smart

Bluetooth is the name of a wireless technology for data exchange in a Wireless Personal Area Network (WPAN) through UHF radio waves in the ISM band between 2.4 and 2.485 GHz. It was created in 1994 by the Swedish company Ericsson as a wireless alternative to RS-232 wired connections. The demand for devices with a more efficient power consumption lead to the presentation in 2010 of a new standard called Bluetooth Low Energy (BLE, later commercially rebranded as Bluetooth Smart) which was included in the 4.0 version of the Bluetooth protocol. With many changes to all the protocol stack levels, BLE can reduce power consumption down to 0.01 mW, making it ideal for small, low-cost and battery operated devices, such as sensor nodes.

BLE Modes

BLE devices can assume two roles: master/central and slave/peripheral.

- Master (or central): usually a device with more computational and energetic resources, such as smartphone or a PC
- Slave (or peripheral): usually a device with low computational power and energetic resources, such as a sensor node

Additionally, BLE uses two other terms to describe two connected devices:

- Server: the device which has some information to share, typically a peripheral device
- Client: the device which needs information or services, typically a central device

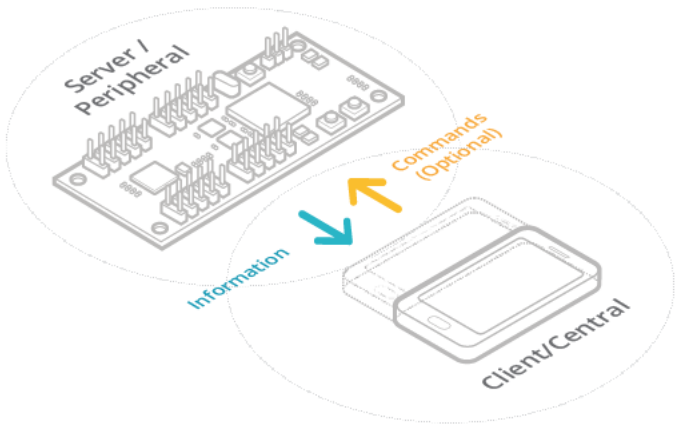


Figure 3.5. BLE roles

Bluetooth Smart can work in two modes:

- Advertising mode: the peripheral device sends small data blocks which all the near central devices can receive
- Connected mode: a peripheral and a central device are one-to-one connected and they can exchange complex data

Advertising mode is therefore one-to-many and unidirectional, while the connected mode is one-to-one and bidirectional. A master device must know that a slave device is near to establish a connection. Consequently the peripheral has to announce its presence using the advertising mode. In this mode the device uses *GAP (Generic Access Profile)* to send small data blocks at a constant rate to signal its presence to all the near BLE master devices. The devices which send the data are called *advertisers*, while the ones which receive the data are called *scanners*. For some applications, like when a peripheral needs to send only a small quantity of data and there is no need for authentication, the advertisement mode is sufficient. The so called "beacons" are a clear example.

In some other cases there is the necessity of a more complex interaction and to exchange data between devices. In those cases it is necessary to establish a connection between a master and a slave device, therefore the connected mode is used. In this mode they use the *GATT (Generic Attribute Profile)*. GATT defines the way two Bluetooth Low Energy devices transfer data back and forth using concepts called *Services* and *Characteristics*.

Characteristics are state variables maintained in a peripheral devices to which a client can access (e.g. they could be temperature, battery level, heartbeat rate, etc.). A characteristic is defined by three fields:

- Declaration: it contains some information on the characteristic, such as its UUID
- Value: the value of the characteristic
- Descriptor: a non mandatory field which can be used to provide additional information on the characteristic or to control its behavior

Furthermore, characteristics can be *static* (such as the name of the device) if their value does not change, or *dynamic*, if the device can change their value when needed.

Characteristics are grouped in Services, which in turn can be grouped in *Profiles* [19].

BLE Packet

In BLE there is only one format which describes the packet for both advertising and data exchange (figure 3.7). Its maximum size is 47 bytes and it has 4 fields:

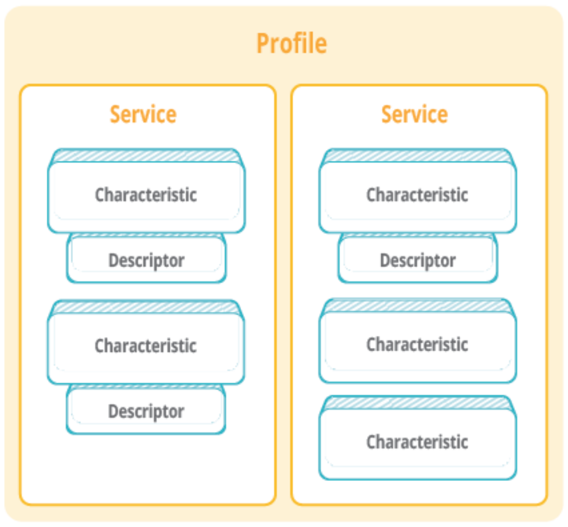


Figure 3.6. Hierarchy of BLE profiles, services and characteristics

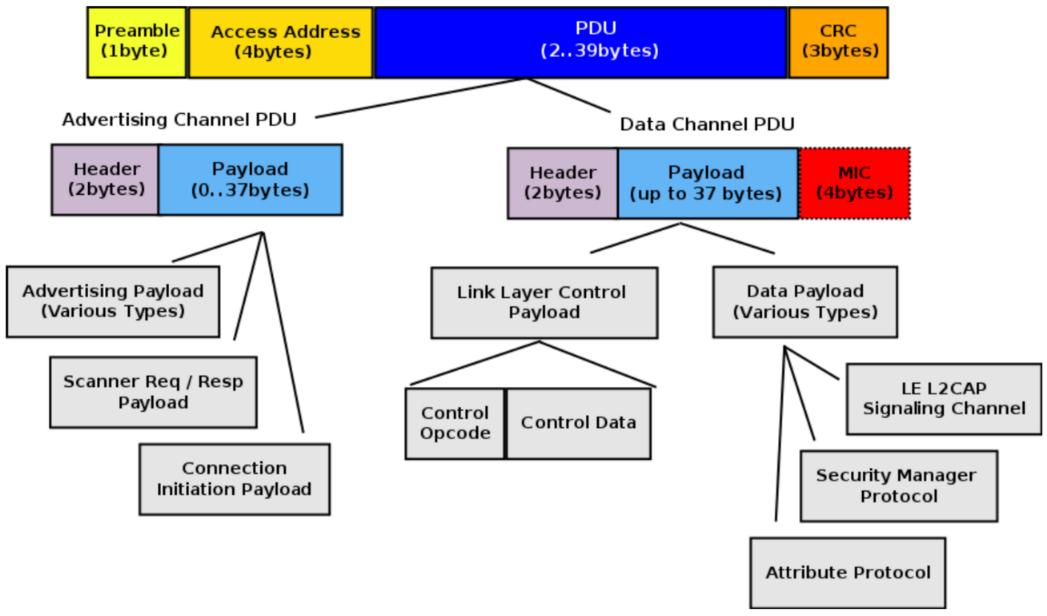


Figure 3.7. BLE packet

Preamble (1 byte), Access Address (4 bytes), PDU (Protocol Data Unit - 0 to 39 bytes), CRC (Cyclic Redundancy Control - 3 bytes).

There are two PDU types, one for advertising channel and one for data channel:

- For the advertising channel, PDU consists of a PDU header (2 bytes) and, depending on the packet, of the device address and up to 31 bytes of information. A scanner can request to the advertiser up to 31 bytes of additional information using the Scan Request and Scan Response packets. Therefore some data can be received by the advertiser even without establishing a connection
- For the data channel, PDU consists of a PDU header (2 bytes) followed by up to 37 bytes of information and a MIC (Message Integrity Check - 4 bytes). MIC is used if the connection is encrypted. The payload of the data channel can carry, depending on the packet type, link layer control information or effective data of higher level protocols (such as characteristics). [40]

3.1.3 Distributed Data Protocol

The Distributed Data Protocol (DDP) is a simple and lightweight data-on-wire (without overhead) protocol, useful for real-time communication. It is based on JSON and it is implemented on top of websockets. DDP is currently at the heart of the Meteor [21] framework.

The two main features of the DDP protocol are:

- Handling Remote Procedure Calls (RPC)
- Managing data in real-time

Remote Procedure Calls

A RPC permits to invoke a method on the server and get something back in return. Furthermore, DDP also notifies the caller after all the write operations in the method have been reflected to all the other connected clients (that is why it is called "distributed").

1. In the example of figure 3.8, the DDP client (arunoda) invokes the method transferMoney with three parameters: 1000USD, arunoda and sacha.
2. After the transfer has been accepted, the DDP server (bank) sends a message with an updated balance to the client. The balance is in the result field. If there was an error, there would be an error field in place of the result.
3. In a future moment, the DDP server sends another message (called "updated"), notifying the client that the transfer has been sent to "sacha" successfully and he has accepted it. [22]

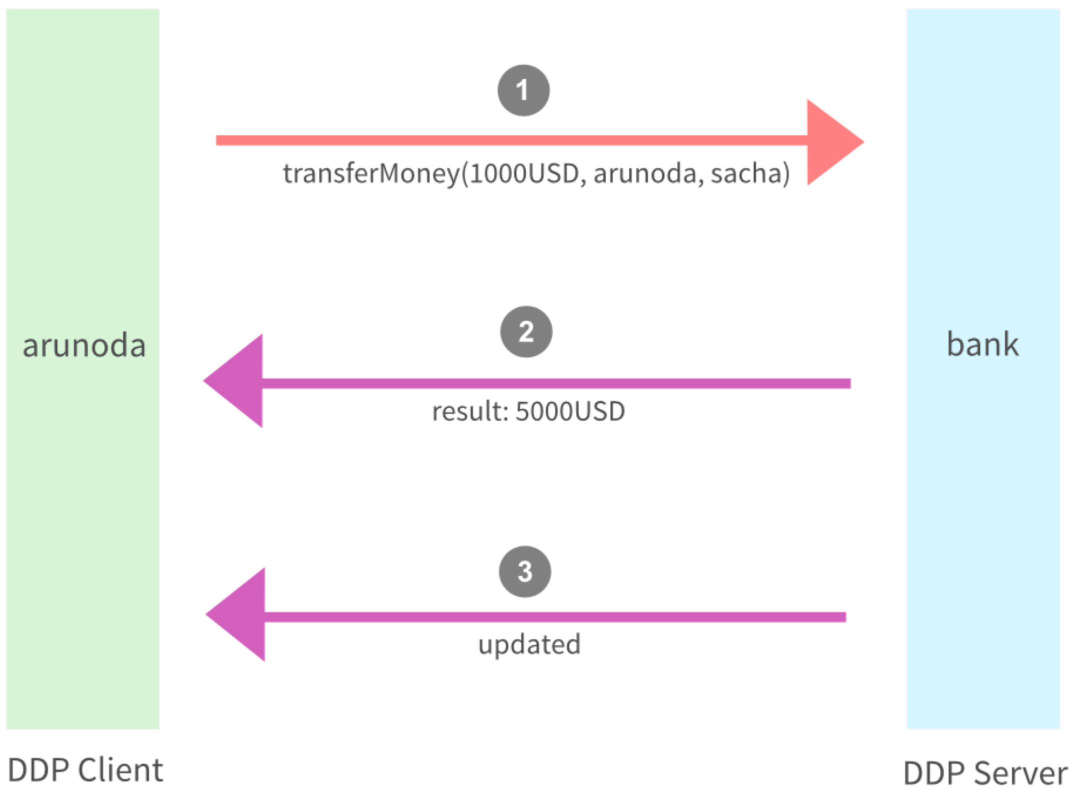


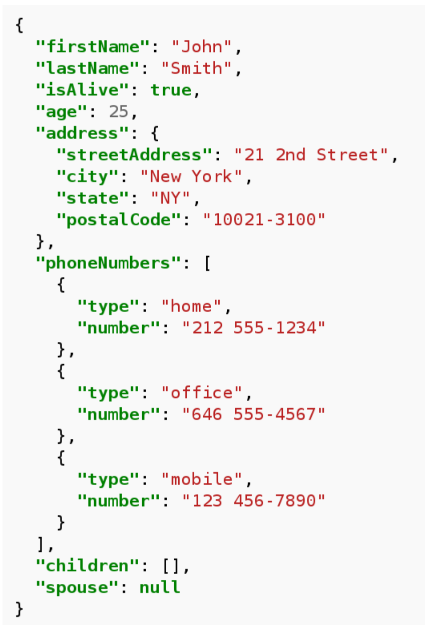
Figure 3.8. Example of DDP RPC call

Real-time Data Management

Using DDP, the server can offer the so called "real time data sources", to which a client can subscribe to get notifications. When something in a real time data source changes, all the subscribed clients are notified. The notifications can be of three types: added, changed and removed, depending on the event that has occurred. Real time data sources are usually implemented as MongoDB collections [22].

3.1.4 JSON

The media used to communicate between the components is JSON. It is an open-standard format that uses human-readable text to transmit data objects consisting of attribute–value pairs. JSON is a language-independent data format. It derives from JavaScript, but code to generate and parse JSON-format data is available in many programming languages. The official Internet media type for JSON is application/json.



```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    },
    {
      "type": "mobile",
      "number": "123 456-7890"
    }
  ],
  "children": [],
  "spouse": null
}
```

Figure 3.9. Example of JSON representing a person

3.2 System Functional Overview

3.2.1 Bike Device

The device is the component which is mounted on the bike. It is equipped with:

- A microcontroller
- A GSM/GPRS modem
- A GPS antenna
- A Bluetooth Smart transceiver
- An accelerometer
- Some environmental sensors: temperature, relative humidity, barometric pressure)
- An electrochemical CO sensor
- A LiPo battery

The main features of the bike device are the following:

- Measurement of pollution levels through the CO sensor

- Anti-theft and remote tracking of the bike
- Provision of status information to the smartphone app
- Acquisition of accelerometer data during bike rides

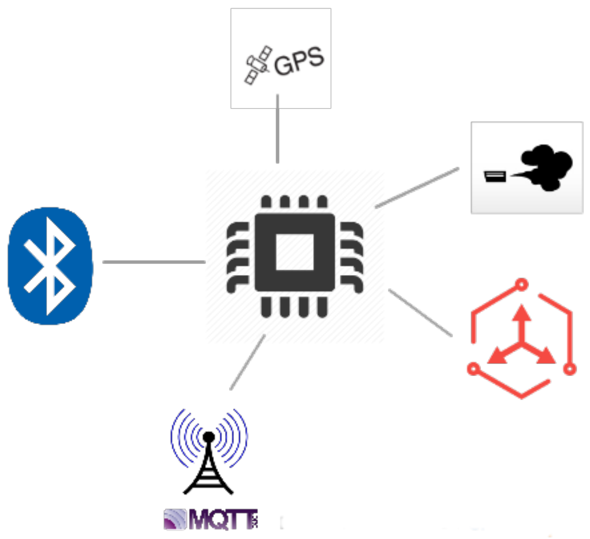


Figure 3.10. Schema of the SwarmBike device

Pollution Sensing

The main purpose of the bike device is to gather information from the environment, in particular to collect data about the concentration of carbon monoxide. Taking into account that the CO sensor can be influenced by temperature, relative humidity and barometric pressure, those data are also acquired. All the information are geotagged using the latitude and longitude values provided by the GPS module and periodically sent to the server through the built-in cellular connectivity. As stated before data are encoded in JSON format and periodically transmitted using the MQTT protocol. To identify the device, the MAC address of its BLE module is used. The MQTT broker requires a username and a password for the authentication, which are provided by the user through the mobile application.

Anti Theft and Remote Tracking

When the bike is parked, the user can enable the anti theft mode on the device using the mobile application. When the anti theft is enabled, the accelerometer is used to detect motion. If a movement is detected, the device sends an alert to the server, which in turns sends a push notification to the smartphone app. Therefore the user is alerted if there is a theft attempt. Furthermore, when the anti-theft is enabled the device periodically sends its position to the backend application, therefore it is always possible to remotely visualize the position either through a web interface or in the smartphone application. Consequently, in case of theft or loss the bike can be always located.

Provision of Status Information to the Smartphone Application

The device provides, through a custom Bluetooth Low Energy service, some status information to the mobile app which in turn presents them to the user. This information includes: anti theft state, gas level value, temperature, humidity, battery charge. However, in the prototype power is supplied via the USB port through a powerbank therefore battery charge is not available.

Acquisition of Accelerometer Data During Bike Rides

Given the fact that the device is equipped with an accelerometer and it is jointly liable with the bike, it was decided to gather also the value of the acceleration on the three axis, which could be used to estimate the quality of road's pavement or to identify some dangerous road tracts.

3.3 Smartphone Application

To let the user manage and configure the device, it was decided to develop a smartphone application. For the first prototype, it was chosen to develop the application for Android, however in the future an iOS app can be implemented too. The main features of the mobile application are:

- Remotely visualize the position of the device on a map
- Present the status information of the device
- Enable/disable anti theft and send authentication data to the device
- Visualize theft alert notifications

Device Status Information

The application establishes a connection with the device using Bluetooth Low Energy. When connected, the app periodically polls the device requesting the values of gas concentration, temperature, relative humidity, barometric pressure and battery charge. These values are then presented graphically to the user.

Anti Theft Status Control and Authentication Information Transmission

Exploiting the Bluetooth Smart connection, the app can send some information or commands to the device. For instance, it is possible to enable or disable the anti-theft mode through a button. It can also be used to transmit authentication information to the device, such as the username and password which will be used to connect to the MQTT broker.

Remote Visualization of Device Position

When the anti-theft mode is enabled, the mobile application can visualize the position of the device on a map. The map is constantly updated, therefore the bike's movement can be tracked in real-time. To achieve a real-time visualization, the app exploits the DDP protocol on top of a websocket to communicate with the cloud backend application. It is also possible to display maps with daily bike paths and associated CO collected measures.

Alert Notifications Visualization

When the device sends a theft alert, the cloud application sends a push notification to the mobile app. When such a notification is received, the smartphone vibrates and a sound is played, therefore the user is informed of every theft attempt.

3.4 Cloud Backend

3.4.1 IoT Platform

An IoT platform is a suite of software components which enables remote data collection and control over connected devices. It offers an easy management of the devices themselves (provisioning and assignment) and convenient collateral functions regarding user and device policy. It can be referred as an IoT middleware, because it effectively is a mediator between the hardware and the application layers. In fact it is completely independent from underlying hardware and overhanging software [34].

The SwarmBike IoT platform receives the JSON encoded data from the various devices via the MQTT protocol. It collects and stores in a suitable database all the information about devices' geographical position, alerts and sensor data. That information is then made available to the overhanging software through a set of REST API.

3.4.2 Backend Application

The backend application allows the fruition of the data collected by the devices, which it gets either from the IoT platform using the set of REST APIs or directly from the device through MQTT. When the anti-theft is enabled on the device, it takes care of building a real-time map on which the geographic position can be visualized. Ideally, when the system will be more mature, it will also have to construct interpolated pollution maps, which could be used to offer users some further services, such as suggestions on less polluted paths.

To construct the map which shows the position of the device, the backend application receives the geographic coordinates directly from the device via MQTT and exploits the DDP protocol for the real-time updates. The map can be viewed via a web interface exposed by the backend application itself, or alternatively it can be accessed from the mobile app.

The backend application is also responsible of elaborating theft alerts sent by the devices: if needed, it sends a push notification to the smartphone application to inform the user of a theft attempt. Push notifications are also sent if, after the anti-theft has been enabled, geographic data are not received for a certain amount of time.

Chapter 4

SwarmBike System Implementation

4.1 Hardware for Device Prototyping

The SwarmBike device is indeed the core part of the whole system. Therefore, the choice of the hardware components played an important role. Many parts were carefully analyzed, and the final decision on which ones should have been used on the device has been taken based on the requirements of the system, especially considering power consumption. The chosen hardware is indicated as follows:

- Prototyping board: STM32 Nucleo L476RG
- GSM/GPRS and GPS module: Adafruit FONA808
- Bluetooth Low Energy 4.1 shield: ST X-Nucleo-IDB05A1
- Environmental and motion sensor shield: ST X-Nucleo-IKS01A1
- Electrochemical CO sensor: Nemoto NAP-505 with Texas Instruments LMP91002 potentiostat

4.1.1 Prototyping Board and Microcontroller Evaluation

The microcontroller is the heart of the device. It contains the CPU, the memory and all the peripherals (such as serial ports and analog-to-digital converters) which are needed to interface the device with the external world. Usually, to simplify the prototyping phase, the so called "prototyping boards" are exploited. A prototyping board is a particular board that makes available all the features and functions of a microcontroller through an easy pin configuration, for a simpler and faster development. Two different prototyping boards were evaluated for SwarmBike: Arduino

Leonardo and STM32 Nucleo L476RG [3] [46]. The principal differences between these two platforms are resumed in table 4.1.

	Arduino Leonardo	STM32 Nucleo L476RG
Microcontroller	ATMega 32u4	STM32L476RGT6
Architecture	AVR 8-Bit	ARM Cortex-M4 32-bit
CPU Frequency	Up to 16Mhz	Up to 80MHz
SRAM	2.5KB	128KB (96+32)
FLASH	32KB (28KB available)	1MB
Operating Voltage	5V	3.3V
UART	Yes	Yes
I2C	Yes	Yes
SPI	Yes	Yes
ADC	Yes (10 bit)	Yes (12 bit)
DAC	No	Yes (12 bit)
Arduino Shield compatibility	Yes	Yes
ARM mbed enabled	No	Yes
Power Consumption	Low (down to μA)	Low (down to μA)
Price	$\sim$ 25 €	$\sim$ 10 €

Table 4.1. Comparison between Arduino Leonardo and ST-Nucleo L476RG

As far as power consumption is concerned, they both satisfy the low-power requirement. However, it is clear that the ST-Nucleo L476RG is much more powerful than Arduino Leonardo. The difference is particularly noticeable for the CPU frequency, which is five times higher on the Nucleo board (80MHz vs 16MHz) and for SRAM memory, with the Nucleo board having an amount more than 50 times larger (128KB vs 2.5KB). CPU computational power and SRAM memory are important to SwarmBike, because the device should be ready for a future SSL/TLS security implementation, not possible on devices with a slow CPU and a tiny memory. The difference in flash memory space is also evident: the L476RG has 1MB available, while the Arduino Leonardo only 32KB. ST-Nucleo is also compatible with ARM mbed (more on this in section 4.2.1), while the Arduino is not. Last, but not least important, although being more powerful the ST-Nucleo board is less expensive, with a price tag of about 10€ compared to the 25€ needed for Arduino.

For all these reasons, the chosen prototyping board for the SwarmBike device was the STM32 Nucleo L476RG.

4.1.2 STM32 Nucleo L476RG

STM32 Nucleo are highly affordable boards which allow to quickly create prototypes with any STM32 microcontroller unit (MCU). Based on the number of available pins and size, there are three different types of boards: Nucleo-32 (32 pins), Nucleo-64 (64 pins) and Nucleo-144 (144 pins). The boards can easily be extended with a large number of specialized application hardware add-ons and shields (Nucleo-64 include Arduino Uno rev3 and ST morpho connectors, Nucleo-32 include Arduino Nano connectors). They also integrate an ST-Link debugger/programmer, consequently there is no need for a separate probe. To upload a firmware to the MCU it is sufficient to copy the binary file produced by the compiler to the virtual mass storage device exposed by the board when it is connected to the PC via a USB cable [45].

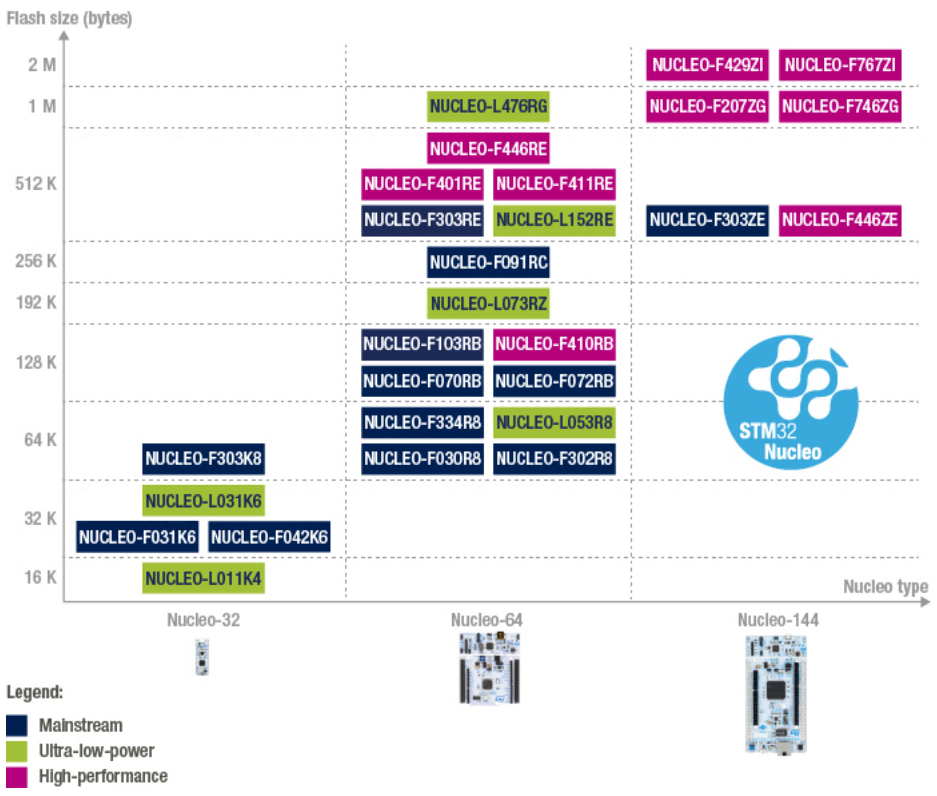


Figure 4.1. Overview of ST Nucleo boards

The STM32 Nucleo L476RG [46] is a Nucleo-64 prototyping board based on the STM32L476RGT6 microcontroller. It is the first Nucleo board of the STM32L4 MCU family, which has been launched by ST in 2015. The particularity of this MCU family is to be both high-performance and ultra-low power. STM32L4 MCUs are in fact based on the ARM Cortex-M4 architecture which provides high performances, delivering 100 DMIPS at 80MHz. However they can have a power consumption as low as a few nanoAmperes, in fact STM32L4 MCUs have ranked high in the standardized EEMBC ULPBench tests which compare the efficiency of ultra-low-power microcontrollers [47] [6].

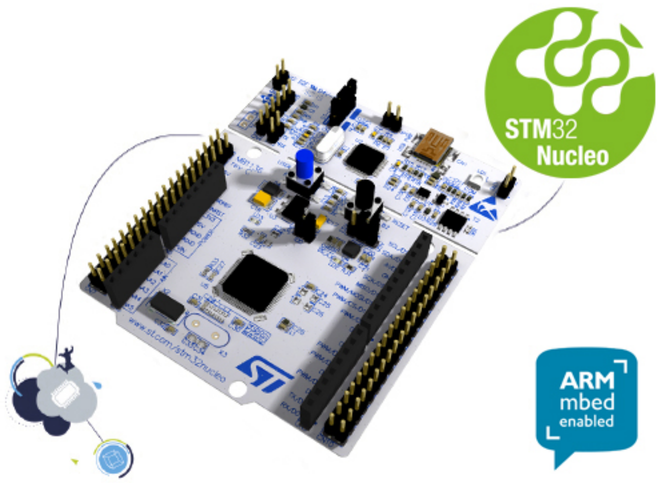


Figure 4.2. STM32 Nucleo L476RG

4.1.3 Adafruit FONA808 GSM and GPS Module

The Adafruit FONA808 is a small board produced by Adafruit Industries, based on the Simcom SIM808 chip, with a price tag of about 50€. It provides a quadband (850/900/1800/1900MHz) GSM module with GPRS/2G connectivity and GPS localization (-165 dBm tracking sensitivity). The module needs to be connected to the microcontroller through a serial port and thanks to the "auto-baud" feature it is able to automatically recognize the baud rate of the UART.

It needs two external antennas, one for GSM and one for GPS, to be able to operate correctly. Additionally, it also needs a dedicated LiPo battery for the power supply, because, especially during transmission, it requires a high current which cannot be entirely provided by the microcontroller. On the board, there is a battery recharger circuit which can be used to recharge the LiPo through a microUSB port.

The module is configured through a set of commands which are sent via the serial port. These commands are known as "Hayes commands", because they were first developed by Dennis Hayes for the Hayes Smartmodem 300 baud modem in 1981. They consist of a series of short text strings which can be combined to produce commands for operations such as dialing, hanging up, changing the parameters of the connection (APN, username, password, etc.), request of GPS coordinates and so on. The commands are also referred to as "AT commands", because every instruction starts with the "AT" string, which stands for "ATTENTION".

The SIM808 can be used to make and receive phone calls, using adequate headphones, and to send and receive SMS. It also has a tiny built-in TCP/IP stack which can be used for basic operations and which can be exploited through the AT commands [11].

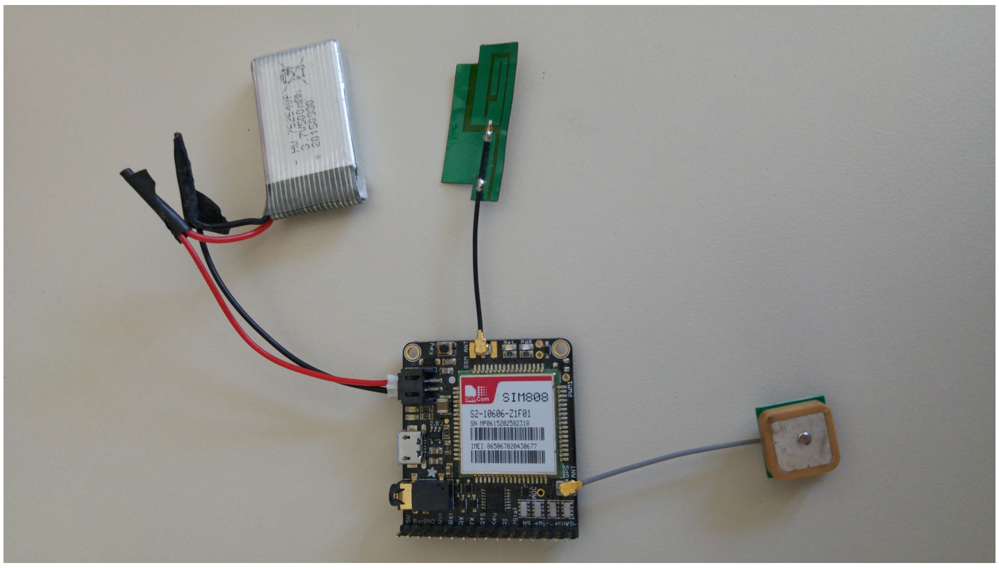


Figure 4.3. Adafruit Fona 808 with GPS and GSM antennas and a LiPo battery

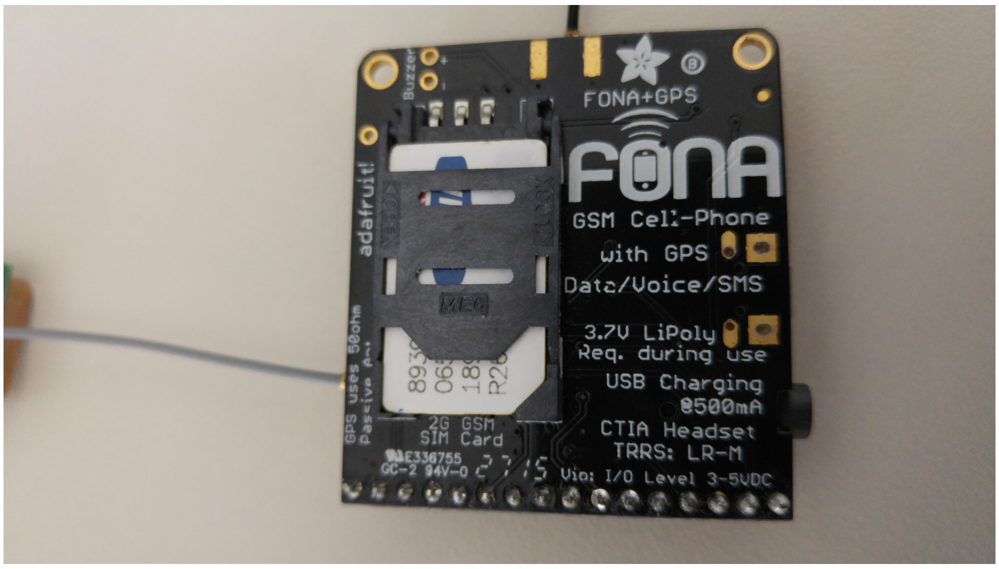


Figure 4.4. Back of the Adafruit Fona 808, the SIM slot is clearly visible

4.1.4 ST X-Nucleo IDB05A1 Bluetooth Smart Shield

The ST X-NUCLEO-IDB05A1 is a Bluetooth Low Energy evaluation board (shield) based on the SPBTLE-RF module to allow expansion of the STM32 Nucleo boards. The SPBTLE-RF is a FCC and IC certified module based on the Bluetooth SMART 4.1 network processor BlueNRG-MS. The X-NUCLEO-IDB05A1 is compatible with the ST Morpho and Arduino UNO R3 connector layout, therefore it can be mounted on top of the Nucleo board and it is immediately operable. It interfaces with the STM32 microcontroller via the Serial Peripheral Interface (SPI) pin [48]. The price of this Bluetooth shield floats around 10€.

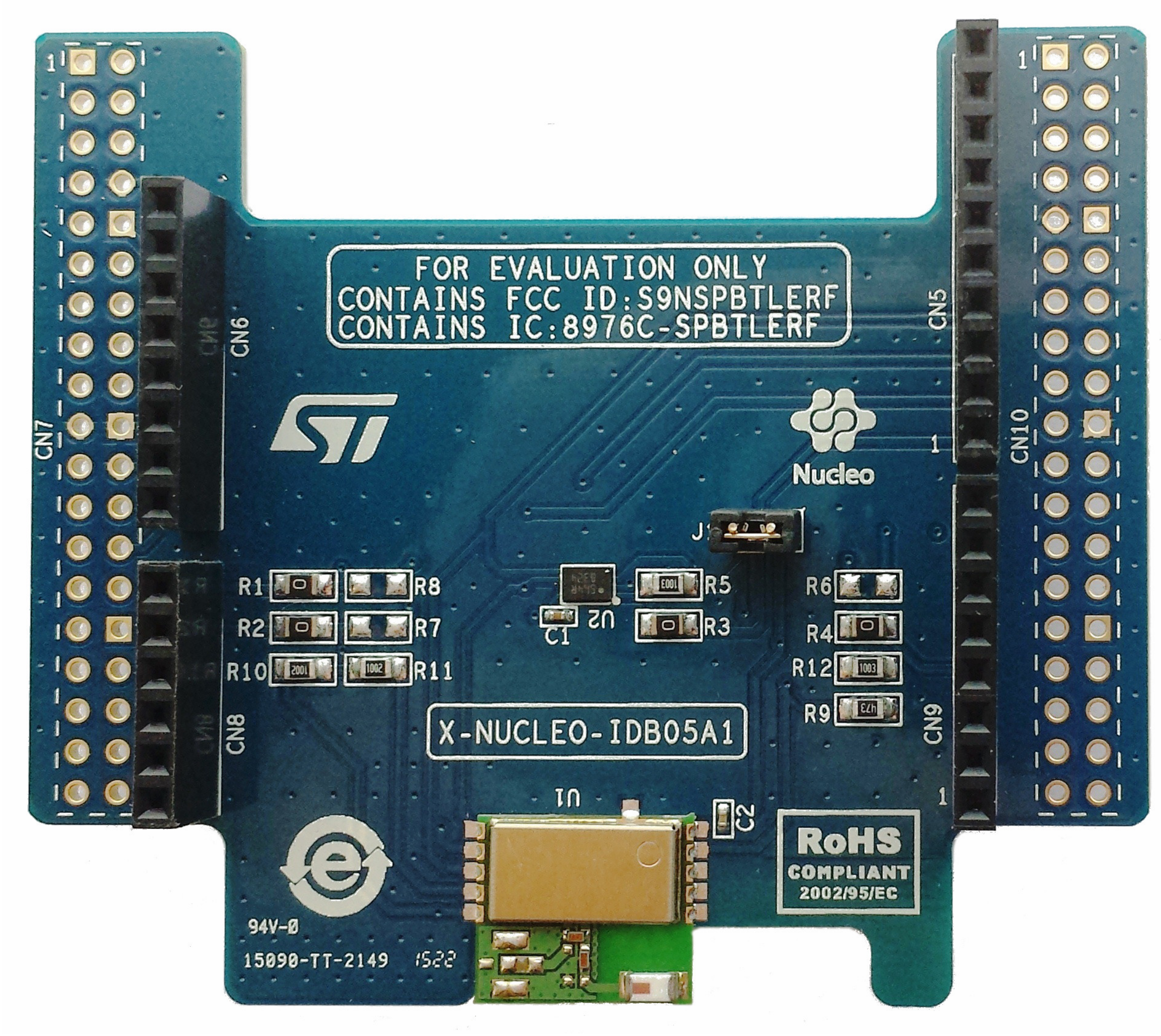

FOR EVALUATION ONLY
CONTAINS FCC ID:S9NSPBTLERF
CONTAINS IC:8976C-SPBTLERF
X-NUCLEO-IDB05A1
Nucleo
RoHS COMPLIANT 2002/95/EC
94V-0
15090-TT-2149

Figure 4.5. X-Nucleo IDB05A1 Bluetooth Low Energy shield

4.1.5 ST X-NUCLEO-IKS01A1 Motion MEMS and Environmental Sensor Shield

The X-NUCLEO-IKS01A1 is a motion MEMS (Micro Electro-Mechanical Systems) and environmental sensor prototyping shield, with a price tag of about 10€. It is able to track motion, temperature, relative humidity, barometric pressure and it interfaces with the microcontroller using the Inter-Integrated Circuit port (I2C). Being compatible with the Arduino Uno R3 shield layout it can be mounted on top of the Nucleo board and it is immediately operable. The board contains the following sensors [49]:

- ST LSM6DS0: MEMS 3D accelerometer (±2/±4/±8 g) + 3D gyroscope (±245/±500/±2000 dps)
- ST LIS3MDL: MEMS 3D magnetometer (±4/ ±8/ ±12/ 16 gauss)
- ST LPS25HB: MEMS pressure sensor, 260-1260 hPa absolute digital output barometer
- ST HTS221: capacitive digital relative humidity and temperature

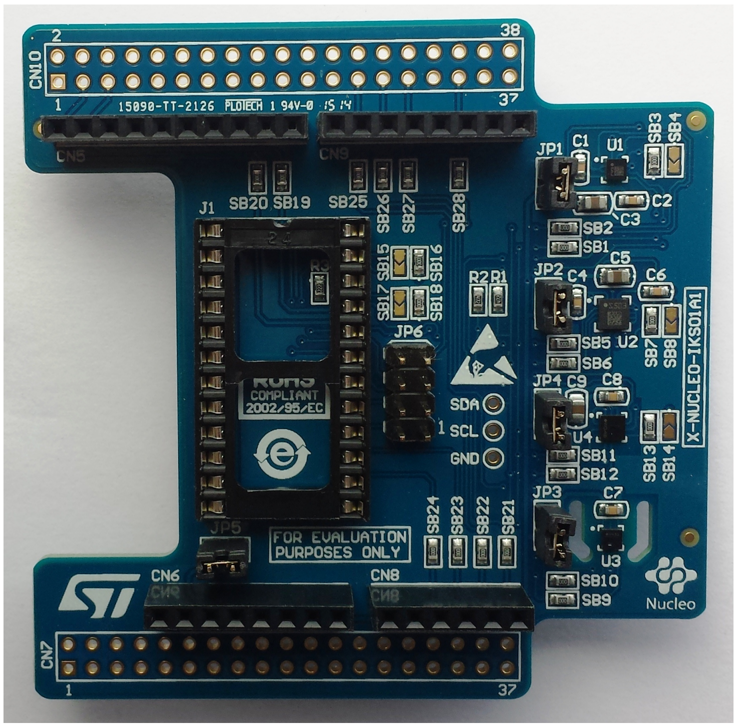


Figure 4.6. X-Nucleo IKS01A1 motion and environmental sensor shield

4.1.6 Nemoto NAP-505 CO Sensor and Texas Instruments LMP91002 Potentiostat

NAP-505 CO Sensor

As stated before, the chosen electrochemical sensor for the SwarmBike device was Nemoto NAP-505, which has been bought on Aliexpress at a price of $18. The NAP-505 Gas Sensor is a new, low–cost 3-electrode electrochemical gas sensor designed for the detection and measurement of carbon monoxide in the range 0-1000ppm. No calibration is needed for this sensor, since it is pre-calibrated at the factory and its sensitivity (i.e. the increase of the output current when the CO concentration rises by 1 part per million) is printed on its case [16].

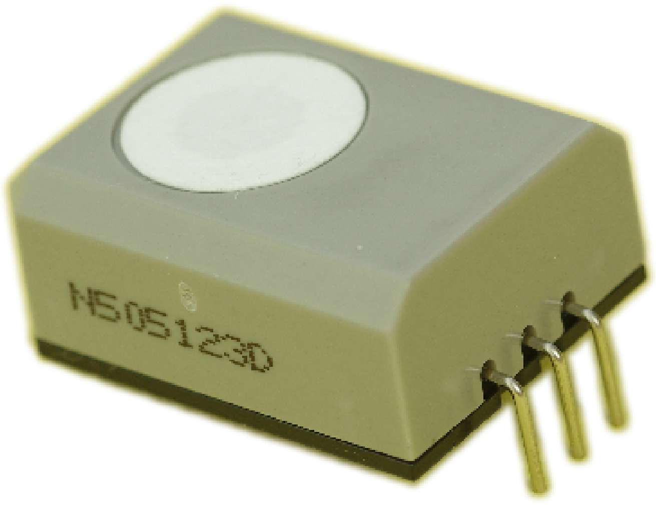


Figure 4.7. Nemoto NAP-505 CO gas sensor

The specifications of this sensor are resumed as follows:

- Detection range: 0-1000 ppm
- Output current: 40±15 nA/ppm
- Reproducibility: ±2%
- Response time (90%): $<$ 30s
- Temperature dependency in air: $<$ 10ppm (0-40 °C)
- Humidity dependency: unaffected by humidity
- Bias voltage: not required [16]

Texas Instruments LMP91002 AFE Potentiostat

An electrochemical gas sensor needs a potentiostat circuit to be operated. It was decided to use the Texas Instruments LMP91002 potentiostat, which has been bought on Mouser for 3.80€ a piece. The LMP91002 device is a programmable Analog Front End (AFE) for use in micro-power electrochemical-sensing applications (the total current consumption can be less than 10 µA). It provides a complete signal path solution between a not biased gas sensor and a microcontroller generating an output voltage proportional to the cell current. It supports gas sensitivities over a range of 0.5 nA/ppm to 9500 nA/ppm. Its transimpedance amplifier (TIA) gain is programmable through the I2C interface [12].

The specifications are resumed as follows:

- Supply Voltage: 2.7V to 3.6V
- Supply Current (Average Over Time): < 10 µA
- Programmable TIA Gain: 2.75 kΩ to 350 kΩ
- Complete potentiostat circuit to interface to most unbiased electrochemical gas sensors
- I2C-Compatible Digital Interface
- 14-pin WSON package

As can be seen, the LMP91002 comes in a 14 pin WSON (acronym of Very Very Thin Small Outline No Lead) package. This means that it cannot be directly operated or connected to the microcontroller or to the sensor, because it needs to be first soldered to a printed circuit board. Therefore, it was necessary to design and realize a PCB which could host both the potentiostat and the sensor, and which could be easily connected to the prototyping board through some jumper wires.

PCB for the Sensing Node

To make the design and the routing of the PCB, it was decided to use the EAGLE PCB Design Software by CadSoft. The schematic of the PCB can be seen in figure 4.8.

The rectangle at the center of the schematic which is labeled “U1” is the LMP91002. The pins numbered 14, 13 and 12 are respectively the connections for the counter electrode, the reference electrode and the working electrode. They are connected to three holes, on which the NAP-505 has to be soldered. The circuit needs few external components:

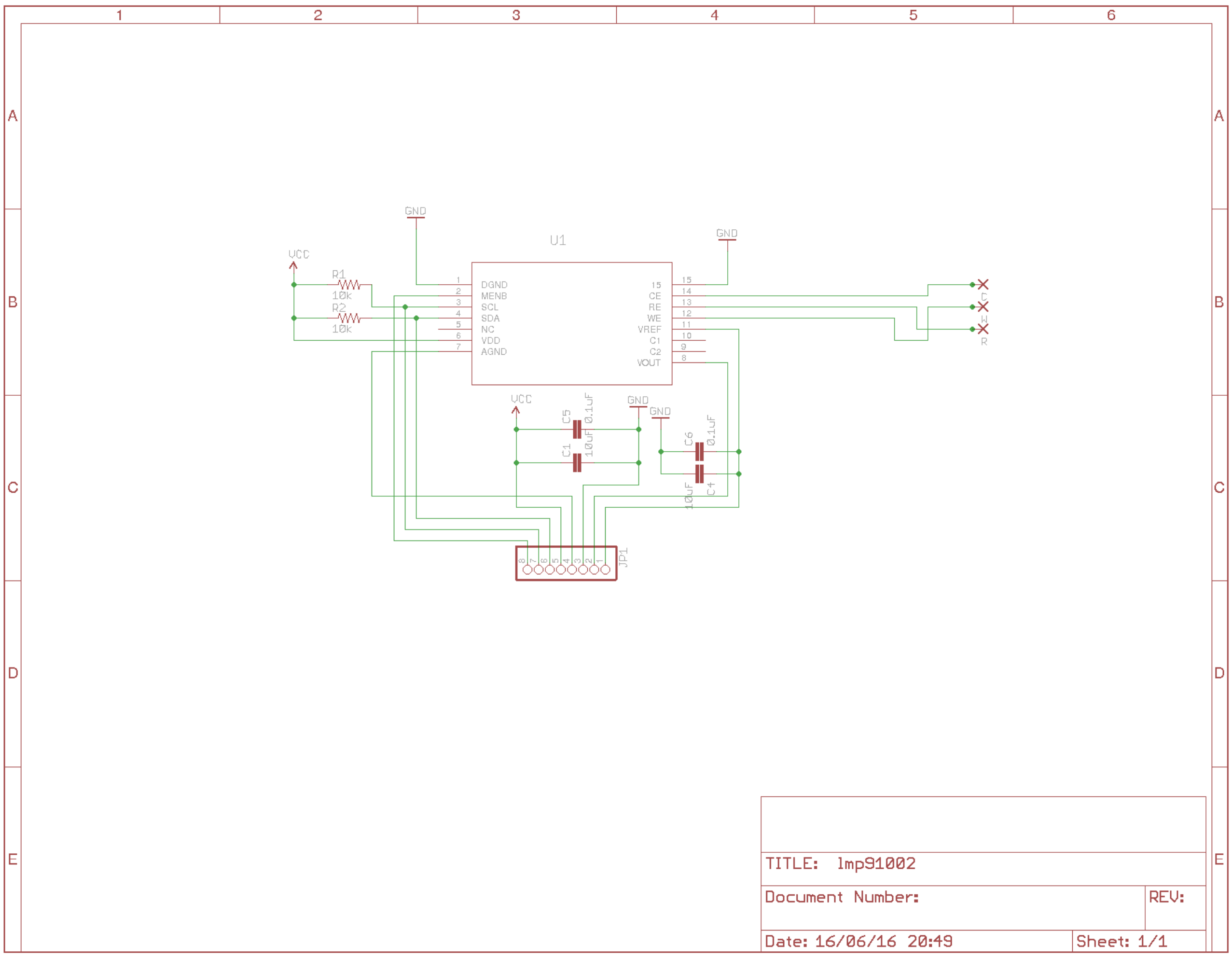


Figure 4.8. Schematic of the PCB for the sensing node

- Two 10 kΩ resistors (R1 and R2 in the schematic), which are needed to pull-up the SDA and SCL lines of the I2C interface
- Two 100 nF capacitors (C5 and C6), which act as a noise filter
- A 8-Pin header (JP1 in the scematic), to which all the potentiostat pins are joined, that is needed to easily connect the PCB to the prototyping board through some jumper wires

The schematic also contemplates two other 10 µF capacitors (C1 and C4) which were not needed on the final design.

The PCB was physically realized by a Belgian company called Eurocircuits [1]. The total cost was of 70€ for 12 PCBs.

The SMD components (capacitors, resistors and LMP91002) were soldered in a lab of the Electronic Engineering Department of Polito, applying by hand some solder paste on the copper pads and using a SMD oven. The NAP-505 sensor, on the other hand, was soldered to the PCB using a normal soldering iron.

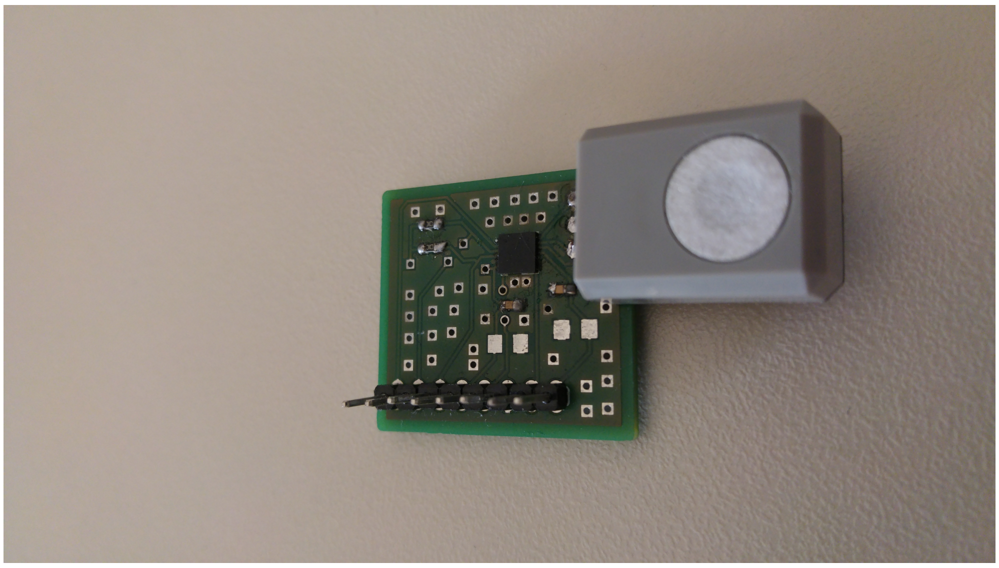


Figure 4.9. The sensing node: the black square at the center of the PCB is the LMP91002

4.2 Firmware Implementation for the SwarmBike Device

4.2.1 ARM mbed OS (mbed 3.0)

As stated before, the STM32 Nucleo MCUs can use the ARM mbed operating system. The available mbed versions at the time of the firmware development were two: mbed 2.0 (also known as mbed Classic) and mbed 3.0 (also known as mbed OS). The chosen version was mbed 3.0, because it assured many advantages over the 2.0 version for an IoT device like SwarmBike.

ARM mbed OS is in fact an open source embedded operating system designed specifically for the Internet of Things devices. It includes all the features that are needed to develop a connected product based on an ARM Cortex-M microcontroller, therefore it is compatible with many ARM based products distributed by different vendors (such as ST, Freescale, NXP, Atmel, Texas Instruments, etc.), and it fits perfectly for applications including smart cities, smart homes and wearables.

Traditional embedded operating systems were designed before embedded devices were connected in vast networks. They consequently don't address the emerging requirements for IoT devices. By contrast, mbed OS is built from the ground up specifically for IoT devices. mbed OS follows by default a single-threaded, event driven architecture, rather than a multi-threaded (real time operating system) environment. This ensures that it scales down to the lowest cost and lowest power IoT devices. The operating system includes a simple event driven scheduler, called "Minar", that provides services for user and system events. mbed OS also includes a low level hardware abstraction layer (HAL) as well as higher level abstract drivers for common hardware peripherals like SPI and I2C ports, ADC and timers. Hardware abstraction and drivers in mbed OS provide a deeply integrated support for power management, which together with energy-awareness in the scheduler helps the OS manage applications where power efficiency is critical [15].

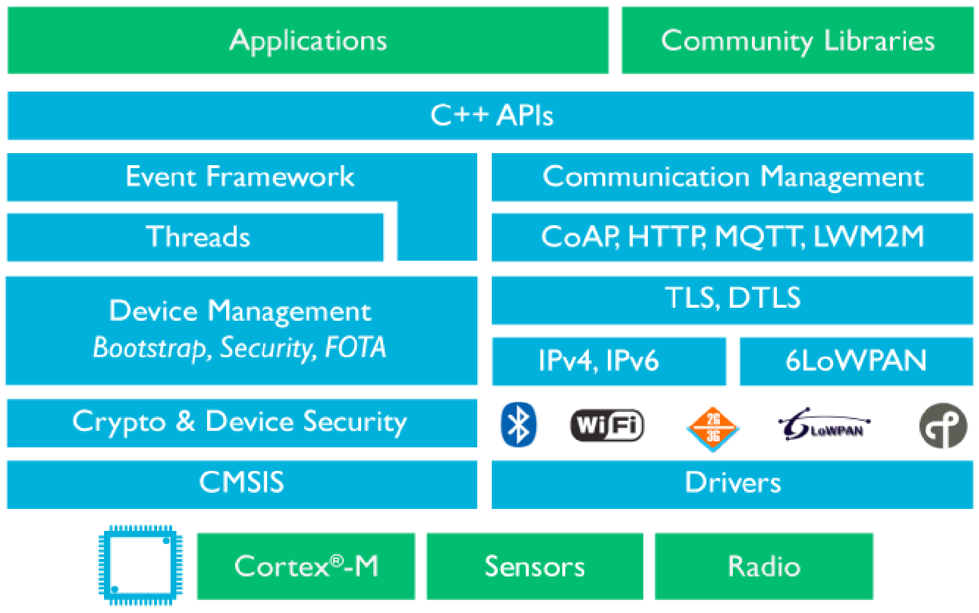


Figure 4.10. mbed OS Architecture

CMSIS

At the fundamentals of mbed OS there is the Cortex Microcontroller Software Interface Standard (CMSIS). It is a vendor-independent hardware abstraction layer for the Cortex-M processor series and it defines generic tool interfaces. The CMSIS enables consistent device support and simple software interfaces to the processor and the peripherals, simplifying software re-use. CMSIS in fact ensures that low-level access to the Cortex-M core functions (such as CPU registers, interrupts, etc.) is consistent between different microcontrollers. Leveraging on the CMSIS, mbed OS is able to run on a great variety of Cortex-M MCU based devices from many different vendors. The CMSIS, however, has a small piece called "CMSIS core" which is target dependent and therefore must be implemented for every different microcontroller.

mbed Hardware Abstraction Layer (drivers)

The access to the MCU peripherals (such as UART, I2C, SPI, ADC, etc.) varies depending on the microcontroller. Consequently a target dependant drivers layer is needed in order to abstract the underlying hardware and allow mbed OS to use those peripherals. This layer is called "mbed-hal" (mbed Hardware Abstraction Layer).

Network Stacks

The networking in mbed OS is based on SAL (Socket Abstraction Layer), which offers a common Unix-like non-blocking socket interface to access the underlying network stacks. Supported network stacks include IPv4, IPv6, 6LoWPAN. The IPv4 stack is based on lwIP (lightweight IP), an open source TCP/IP stack optimized for embedded devices. The system also supports TLS/SSL security using a dedicated TLS implementation called "mbed-tls".

mbed OS natively supports Bluetooth Low Energy too.

Minar Scheduler

Minar is the event-driven scheduler of mbed OS. The user or the system itself can submit tasks to Minar which will be then executed. A task can be scheduled to be executed as soon as possible, with a delay or even periodically. When there are no tasks to be executed, Minar automatically put the microcontroller in a sleep state to save power. It is important to remark that Minar is not a preemptive scheduler, therefore if a task takes too much time to be executed, all the other tasks will be delayed.

mbed C++ APIs

mbed OS offers a set of C++ APIs (called "mbed-drivers"), which present an unified interface to access the MCU peripherals and all the OS features, therefore software written exploiting those APIs will run on any device compatible with mbed OS.

4.2.2 Needed Collateral Software Developed for the Firmware

mbed OS HAL and CMSIS Modules Porting for STM32 Nucleo L476RG

As stated before, the chosen mbed version for the development of the firmware was mbed 3.0, also known as mbed OS. At the time of the implementation of the firmware (January 2016), mbed OS was still in the "Technology Preview" development stage and it was available only for a few prototyping boards. Unfortunately, the STM32 Nucleo L476RG was not in that number. However, given the fact that the advantages offered by the 3.0 version of mbed were highly valuable, it was decided to make a porting of the needed low-level modules in order to use mbed OS on that board. Therefore, a 3.0 compatible version of the "cmsis-core" and "mbed-hal" modules for the Nucleo L476RG has been developed. The porting work consisted in the adaptation of the CMSIS and HAL modules for the Nucleo L476RG which already existed for mbed 2.0.

The ported modules have been released under the Apache License and are all available on github at [26] [25] [27] [23] [24] [30].

Adafruit FONA 808 Driver

The built-in TCP/IP stack of the Adafruit FONA808 was not considered appropriate to be used on SwarmBike, because it was too limited and had some flaws. Therefore, it was decided to exploit the mbed OS own lwIP TCP/IP stack. In order to make it possible, a driver has been developed. The driver configures the FONA808 sending the appropriate AT commands (e.g. it sets the correct APN) on the serial port to which the modem is connected, and establishes a connection via the PPP (Point-to-Point) protocol, exploiting a PPP client implementation offered by the lwIP stack. The driver also offers an interface to access the FONA808's GPS module.

This driver has also been released under the Apache license and it is available on github at [29].

Eclipse Paho Embedded MQTT Client

At the time of the implementation (January 2016), no MQTT clients were available for mbed OS yet. Consequently, it was necessary to adapt an existing version of the Eclipse Paho embedded MQTT client [33], which was available for mbed 2.0. The existing client has been modified to use the Minar scheduler for handling events and to exploit the new Socket Abstraction Layer offered by mbed OS.

Even this MQTT client has been released under the Apache license and is available on github at [28].

Texas Instruments LMP91002 Driver

It was necessary to develop a driver for the Texas Instruments LMP91002 potentiostat. It was necessary a porting from an existing Arduino library. The porting work consisted in the adaptation of the I2C interface of the driver to the mbed OS one.

4.2.3 Firmware Architecture

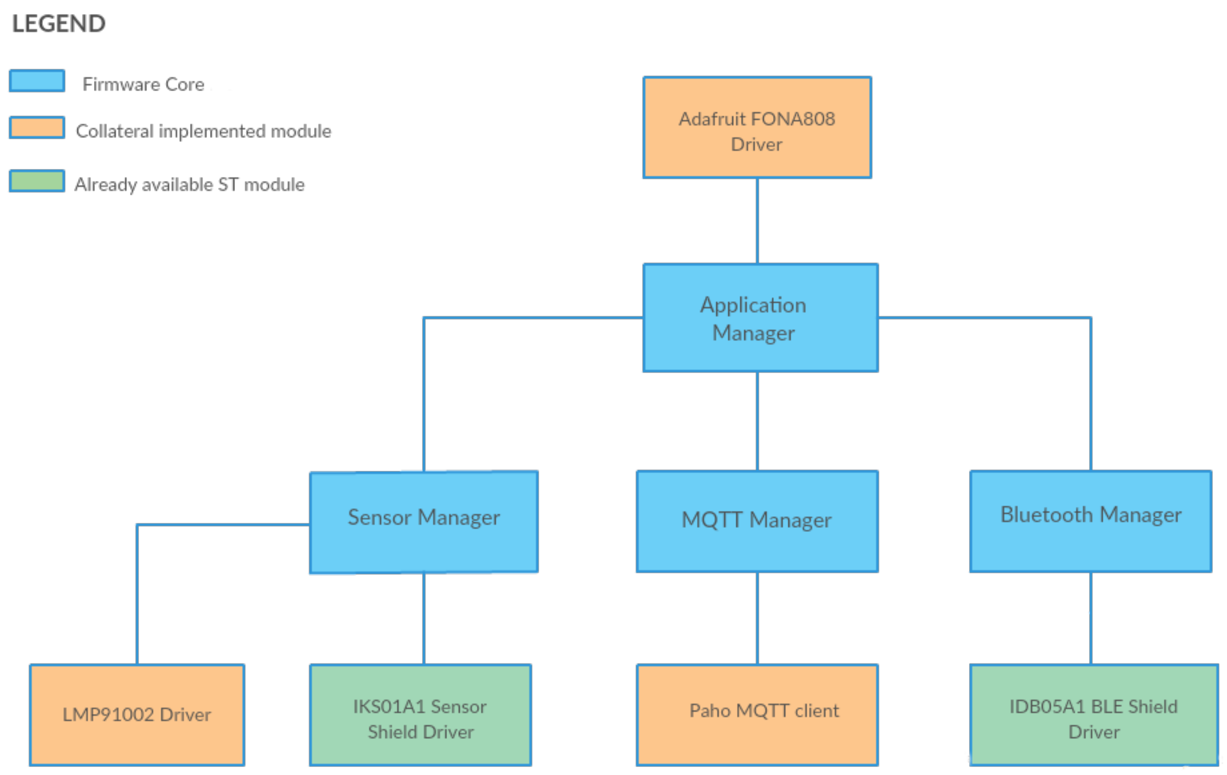


Figure 4.11. Architecture of the firmware

Figure 4.11 illustrates the architecture of the firmware. The blue colored components are the ones which form the core of the firmware, the piece of software which implements the logic behind the SwarmBike device. The orange colored components are the developed collateral drivers which are needed for the device to work, but which are not part of the core. The green colored ones are drivers provided by STMicroelectronics.

Application Manager

Application Manager is the main component of the firmware. Its purpose is to coordinate all the other components, ensuring that everything works correctly. For instance, it takes care of starting the MQTT connection process only after the GPRS connection has been established by the FONA808, or it ensures that the sensed data are transmitted only once the GPS has made the fix. ApplicationManager is the component which holds the state of the anti-theft (enabled/disabled) firmware. It is also the entry point of the firmware (i.e. it is the component which is first called when the system starts).

The typical flow of the actions executed by the ApplicationManager when the anti-theft is not enabled is indicated as follows:

1. If the Internet connection is not active, it establishes the GPRS connection using the FONA808 driver
2. If the MQTT client is not connected, it starts the MQTT connection process through the MQTT Manager
3. It gets the sensed data from the Sensor Manager
4. It gets the GPS coordinates using the FONA808 driver
5. It makes available the sensed data through a custom Bluetooth Low Energy service, using the Bluetooth Manager
6. It publishes the coordinates and the measurements (encoded in JSON format) on a MQTT topic through the MQTT Manager
7. Go back to step 1

Sensor Manager

Sensor Manager is the component which is used to read sensor values. It interfaces with the IKS01A1 library to access the temperature sensor, the humidity sensor, the barometric pressure sensor and the accelerometer. It also configures the potentiostat using the LMP91002 driver and reads the value of the CO sensor. Sensor Manager is also used to detect jolts through the accelerometer, which can trigger an alert if the device has the anti-theft enabled. All the sensed values are exposed to the Application Manager through a set of functions.

Bluetooth Manager

Bluetooth Manager is responsible of managing the Bluetooth connection with the mobile app. It initializes the mbed BLE stack and interfaces with the ST BLE shield using the IDB05A1 driver provided by STMicroelectronics. It contains the implementation of the custom Bluetooth Low Energy GATT service which is used for the communication with the smartphone. This component also takes care of sending/receiving data to/from the mobile application (e.g. anti-theft commands, sensed values, authentication information) and it notifies the Application Manager through a callback when a BLE event occurs.

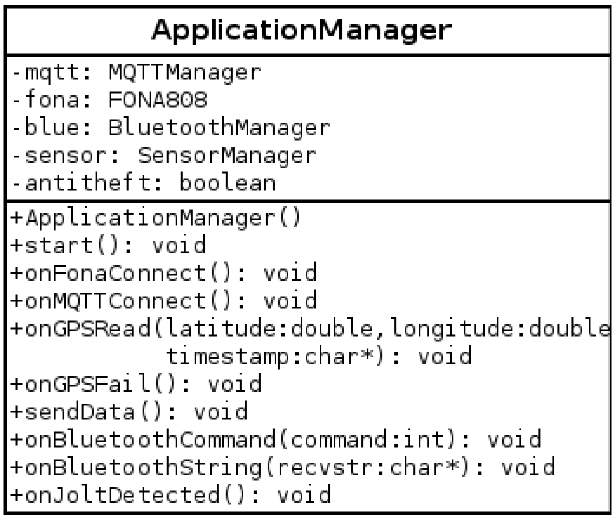
MQTT Manager

MQTT Manager is the component which manages the MQTT connection. It provides to the Application Manager a simplified interface of the Paho MQTT client. It manages the connection and disconnection of the client and it offers suitable functions for publishing a message and subscribing/unsubscribing to a topic. It also notifies the Application Manager if a message arrives on a subscripted topic.

4.2.4 Firmware Core Implementation

The firmware has been implemented using the C++ programming language and exploiting the APIs provided by mbed OS. Given the fact that the software has to run on an embedded system, it has been written to be as much lightweight as possible (e.g. avoiding dynamic run-time memory allocation). Every core component of the firmware is therefore implemented as a C++ class. Yotta, a python script developed by ARM for mbed OS, has been used to manage the compiling and the assembling of the code and of all the needed mbed OS modules. The firmware has been developed on GNU/Linux (Debian Jessie). The code has been written with the aid of a simple text editor, Gedit.

Implementation of the Application Manager Component



```
ApplicationManager
-mqtt: MQTTManager
-fona: FONA808
-blue: BluetoothManager
-sensor: SensorManager
-antitheft: boolean
+ApplicationManager()
+start(): void
+onFonaConnect(): void
+onMQTTConnect(): void
+onGPSRead(latitude:double,longitude:double
           timestamp:char*): void
+onGPSFail(): void
+sendData(): void
+onBluetoothCommand(command:int): void
+onBluetoothString(recvstr:char*): void
+onJoltDetected(): void
```

Figure 4.12. ApplicationManager class

The ApplicationManager class has five main **members**:

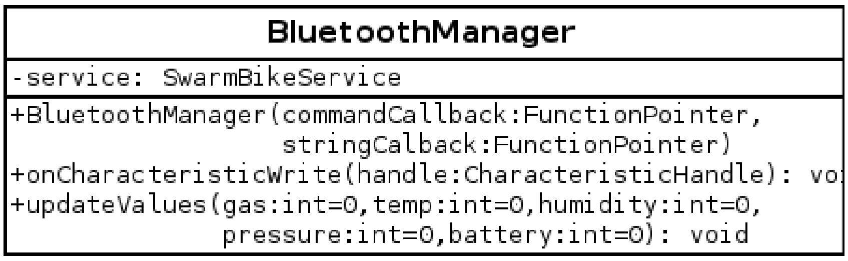
- mqtt: an instance of the MQTTManager component
- fona: an instance of the FONA808 class, provided by the FONA808 driver
- blue: an instance of the BluetoothManager component
- sensor: an instance of the SensorManager component
- antitheft: a boolean holding the anti-theft state

On the other hand, ApplicationManager has ten main **methods**:

- the constructor: it simply initializes all the members, instantiating the relative objects
- start: when the system starts, this is the first method that is called. It starts the application main routine, beginning the FONA connect process and initializing the bluetooth subsystem
- onFonaConnect: callback method called by the FONA driver when the connection process has finished
- onMQTTConnect: callback method called by the MQTTManager when the MQTT connection has been established
- onGPSRead: callback called by the FONA driver when the GPS reading process has ended. The arguments will contain the latitude, the longitude and the timestamp obtained from the GPS
- onGPSFail: callback called by the FONA driver when the GPS reading process has failed because the GPS has not made the FIX yet
- sendData: this method gets the sensor data from the SensorManager, encodes them in JSON format and publishes the resulting JSON string on a MQTT topic (to which the IoT server platform is subscribed) using the MQTTManager. If the anti-theft is enabled, geographic data are also published onto a second MQTT topic (to which the cloud backend application is subscribed) in order to permit the construction of a map showing the position of the device in real-time. This method is called every 12 seconds when the anti-theft is disabled and every 30 seconds when the anti-theft is enabled.
- onBluetoothCommand: callback called by the BluetoothManager when a command is received via Bluetooth (e.g. enabling or disabling the anti-theft). The argument contains the received command

- onBluetoothString: callback called by the BluetoothManager when a string is received via Bluetooth. The string (passed as an argument to the function) usually contains authentication information
- onJoltDetected: callback called by the sensor manager when a jolt is detected by the accelerometer. If the anti-theft is enabled, it sends an alert via MQTT using the MQTTManager

BluetoothManager Implementation



```
BluetoothManager
-service: SwarmBikeService
+BluetoothManager(commandCallback:FunctionPointer,
                  stringCalback:FunctionPointer)
+onCharacteristicWrite(handle:CharacteristicHandle): vo
+updateValues(gas:int=0,temp:int=0,humidity:int=0,
              pressure:int=0,battery:int=0): void
```

Figure 4.13. The BluetoothManager class

The BluetoothManager class has one main **attribute**, service, which contains an instance of the custom Bluetooth Low Energy service used for the communication with the smartphone application.

The main **methods** are three:

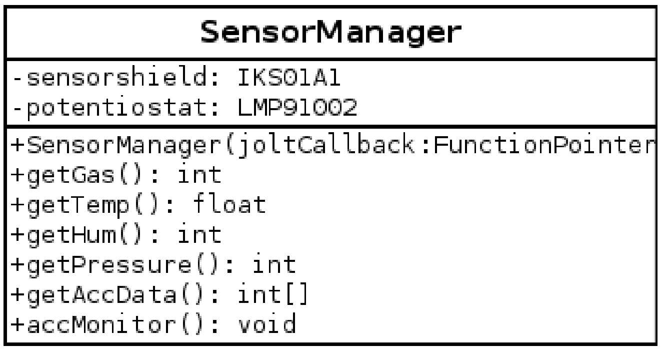
- the constructor: it instantiates the BLE service and initializes the mbed OS BLE subsystem and the ST BLE shield. The two arguments represent the function pointers of the callbacks which have to be called when a command or a string is received via Bluetooth
- onCharacteristicWrite: this is a callback method which is called by the BLE subsystem when something is received via Bluetooth. It distinguishes whether the received data is a command or a string and calls the appropriate callback
- updateValues: it updates the characteristics of the BLE service using the values passed as arguments

The BluetoothManager component contains the implementation of the **custom BLE service** used by the SwarmBike device. The service is made by seven different characteristics:

- gas (int): it holds the sensed CO value. It is read-only for the mobile application

- temp (int): it holds the sensed temperature value. It is read-only for the mobile application
- humidity (int): it holds the sensed humidity value. It is read-only for the mobile application
- pressure (int): it holds the sensed pressure value. It is read-only for the mobile application
- battery (int): it holds the charge level of the battery. It is read-only for the mobile application. Not used in the first prototype of SwarmBike
- command (int): the mobile application can write this characteristic. It is used to send commands to the device
- auth (char[]): the mobile application can write this characteristic. It is used to send authentication information to the device

SensorManager Implementation



```
SensorManager
-sensorshield: IKS01A1
-potentiostat: LMP91002
+SensorManager(joltCallback:FunctionPointer
+getGas(): int
+getTemp(): float
+getHum(): int
+getPressure(): int
+getAccData(): int[]
+accMonitor(): void
```

Figure 4.14. The SensorManager class

The SensorManager class has two main **members**:

- sensorshield: an instance of the IKS01A1 class, provided by the ST sensor shield driver
- potentiostat: an instance which represents the potentiostat, provided by the LMP91002 driver

The main **methods** are:

- the constructor: it initializes the two members. It takes care also to configure the potentiostat: the TIA gain is set to 350 kΩ and the internal zero offset is set to 20% of V_{DD}. The argument of the constructor is the function pointer of the callback which has to be called if a jolt is detected.

- getGas: it reads the value of the CO sensor and returns it to the caller. It is worthwhile to illustrate how the CO concentration value is calculated: firstly, the output voltage of the sensor is read ten times through the analog-to-digital converter, and the average of the readings is calculated. Secondly, the value of the zero offset is subtracted from the computed average. The resulting value is the voltage increase due do the CO in the air. The corresponding current value is calculated using Ohm's law and the TIA gain. Finally, the equivalent parts-per-million concentration is computed, dividing the current value by the sensitivity of the CO sensor (40 nA/ppm in the case of Nemoto NAP-505)
- getTemp: it reads the temperature value from the ST sensor shield and returns it to the caller
- getHum: it reads the relative humidity value from the ST sensor shield and returns it to the caller
- getPressure: it reads the barometric pressure value from the ST sensor shield and returns it to the caller
- accMonitor: it is executed every 250ms and it has two purposes: the first is to constantly analyze the acceleration values to detect jolts. If a jolt is detected, the joltCallback is called. The second is to collect acceleration data. In fact it was decided to acquire the acceleration values on the three axis, because those data could be used to analyze road's pavement quality or identify dangerous road tracts
- getAccData: returns to the caller the collected accelerometer data

MQTTManager Implementation

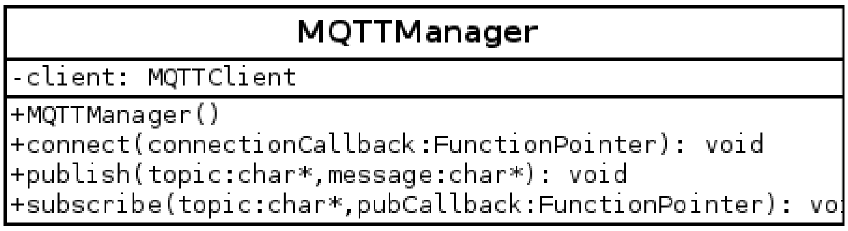

MQTTManager
-client: MQTTClient
+MQTTManager()
+connect(connectionCallback:FunctionPointer): void
+publish(topic:char*,message:char*): void
+subscribe(topic:char*,pubCallback:FunctionPointer): vo

Figure 4.15. MQTTManager class

This class has one main **member**, client, which is an instance of the Eclipse Paho MQTT client.

The main **methods** are:

- the constructor: it initializes the client member

- connect: it starts the connection process of the Paho client. The argument is the function pointer of the callback which has to be called when the connection has been established
- publish: it publishes a message to a MQTT topic, using QoS 0. The topic and the message are both passed as arguments
- subscribe: it subscribes to the topic passed as argument. The other argument is the function pointer of the callback which has to be called when a message is published on the subscribed topic

Typical Actions Flow

In figures 4.16 and 4.17 the flow of two typical actions, measurements transmission and alert sending, is shown:

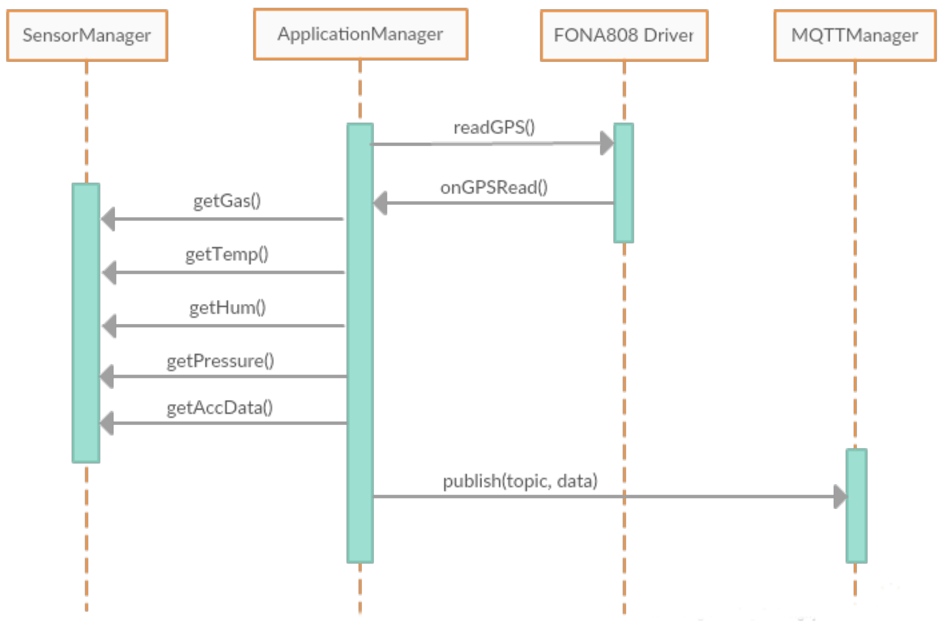


Figure 4.16. Flow of measurements transmission

Security in the SwarmBike Firmware

Given the fact that SwarmBike is still a prototype, security was not one of the key point during the implementation of the firmware. However, some potentially sensible data, like user's position or authentication data are exchanged through the components of the SwarmBike system. Furthermore the device also implements an

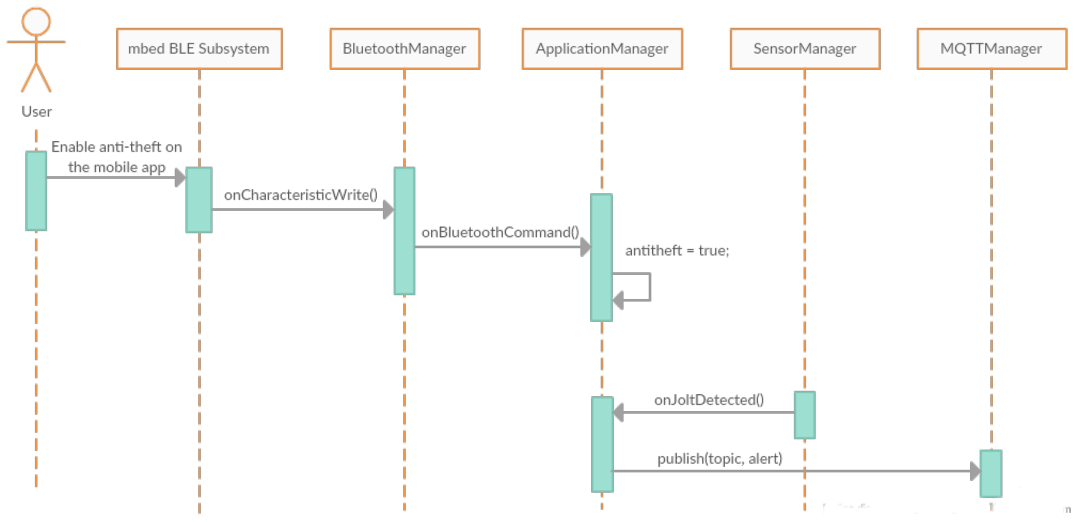


Figure 4.17. Flow of anti-theft enabling and alert transmission

anti-theft system, therefore it would need secure communication. Consequently, the device was designed to be "security ready". The two main security concerns are:

- Encryption and authentication of data sent and received through the Internet via MQTT
- Encryption and authentication of data sent and received via Bluetooth Smart

Regarding the first point, the microcontroller is powerful enough to support SSL/TLS, and mbed OS has a native implementation of the TLS stack. The MQTT client should be modified in order to use secure sockets in place of regular sockets.

In relation to the second point, the chosen bluetooth transceiver supports all the BLE security features, but the driver provided by STMicroelectronics does not support them yet. However, security support is expected to be implemented in next releases.

Therefore, in a future work, it should be easy to address all the security concerns currently present in SwarmBike.

4.3 Mobile Application for Android

As stated before, it was decided to develop a smartphone application to let the user manage and configure the SwarmBike device. The app has been developed for Android, using the last version "Marshmallow" as target, however it works on any Android device equipped with the Android 5.0 "Lollipop" version or newer. The development has been done using Android Studio as IDE.

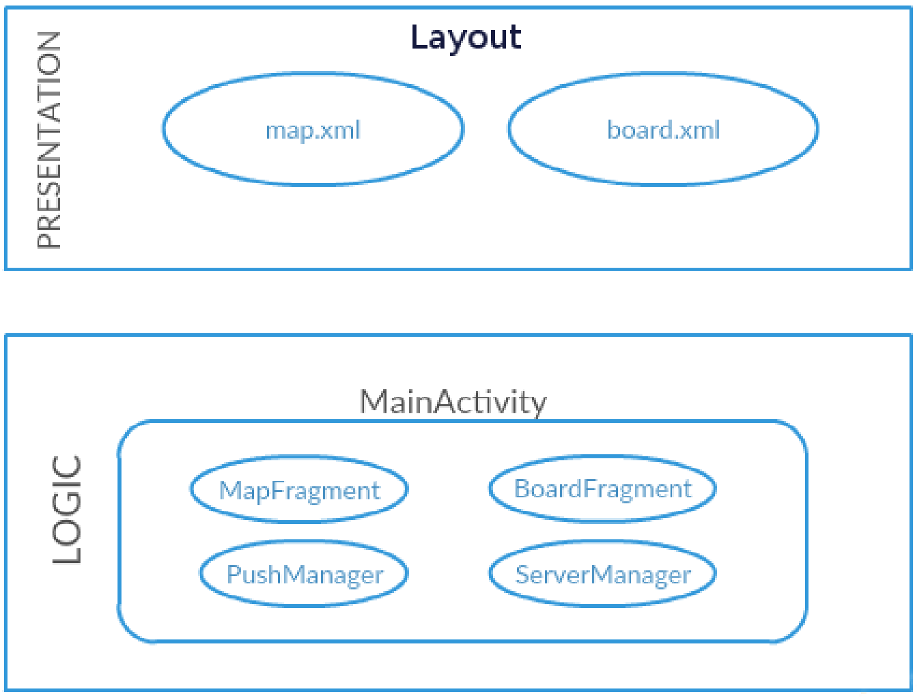


Figure 4.18. Android application architecture

As can be seen in figure 4.18, the application is organized into two levels: the logic level and the presentation level. The logic level contains the application logic, the presentation level contains the graphical part, implemented using XML. Dividing the graphical level from the logical level is a best practice in Android app development.

The components of the presentation level are:

- map.xml: the xml that implements the graphic of the MapFragment
- board.xml: the xml that implements the graphical part of the BoardFragment

The components of the logic level are:

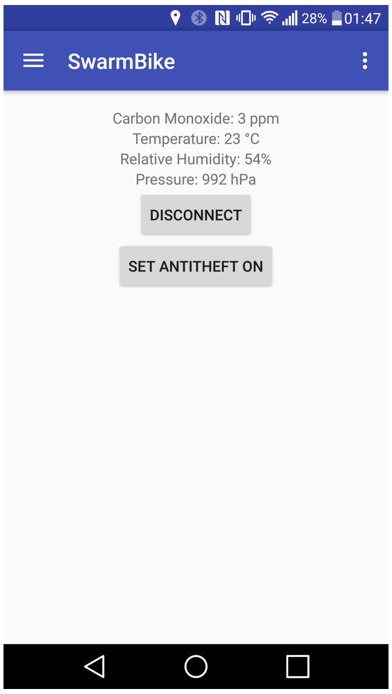
- MainActivity: this is the Android Activity which contains all the other fragments. It takes care to ask the user all the permissions necessary for the app to work properly. It also manages all the bluetooth related logic: it executes the BLE scanning, it establishes the connection to the device and it reads and writes the characteristics of the SwarmBike service
- MapFragment: it shows (using the map.xml layout) a map on which is possible to visualize the current position of the SwarmBike device. The map is shown

through a WebView object. The WebView connects to the cloud backend application which is responsible to build the map. In order to permit real-time updates of the device's position, the DDP protocol is exploited.

- BoardFragment: it lets the user connect to the device using a button. Once the connection has been made, the current gas, temperature, pressure and humidity sensed values are shown in this fragment. It permits the management of the status of the anti-theft through a button. The graphical part of this fragment is implemented in board.xml

- PushManager: it allows the mobile application to receive the push notifications, exploiting the Google Cloud Messaging (GCM) system. In case a theft alert is triggered by the device, a push notification is received and a heads-up notification is shown to alert the user.

- ServerManager: it manages the connection with the cloud backend application. The connection is exploited to transmit to the server (on the first use of the app) the GCM notification ID which will be used for sending push notifications. The notificationID is sent by mean of a DDP remote procedure call.



Carbon Monoxide: 3 ppm
Temperature: 23 °C
Relative Humidity: 54%
Pressure: 992 hPa

Figure 4.19. Screenshot of the BoardFragment from the SwarmBike Android app

4.4 Cloud Backend

As stated before, some platforms and technologies already employed by TIM in previous projects have been used to implement the SwarmBike cloud backend. In particular, SiteWhere has been chosen as IoT platform, while a Meteor application previously developed by TIM SWARM JOL has been re-adapted to work with the SwarmBike system.

For the prototyping stage, it was used a free Amazon EC2 instance to host SiteWhere, the Meteor App and all the other needed components (such as MongoDB). Mosquitto has been used as MQTT broker.

4.4.1 SiteWhere IoT Server Platform

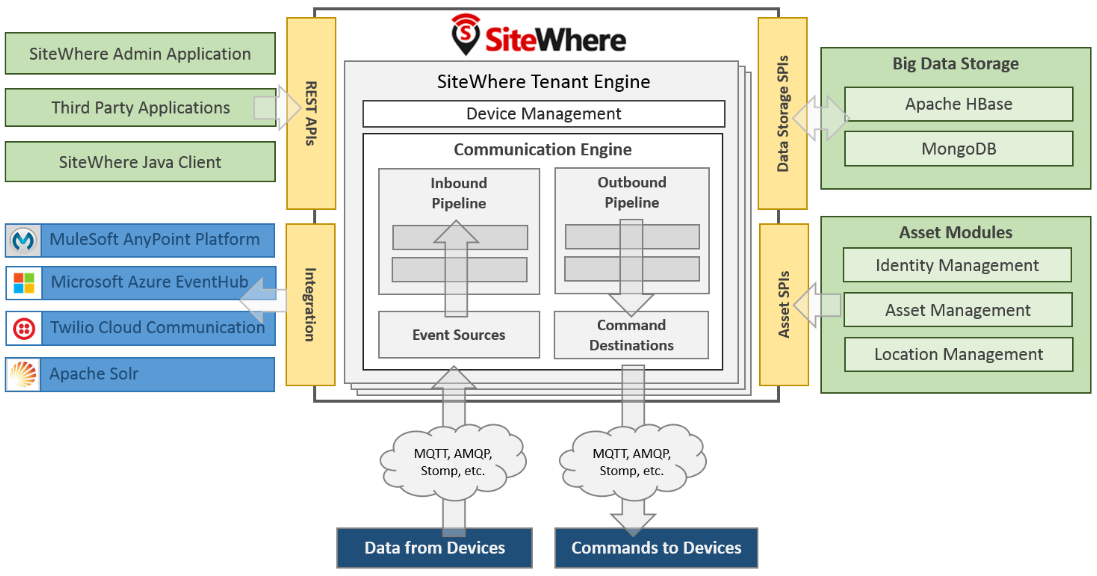


Figure 4.20. SiteWhere architecture

SiteWhere is the IoT server platform chosen for the SwarmBike system. It is an open source IoT platform which provides a system that facilitates the ingestion, storage, processing, and integration of device data.

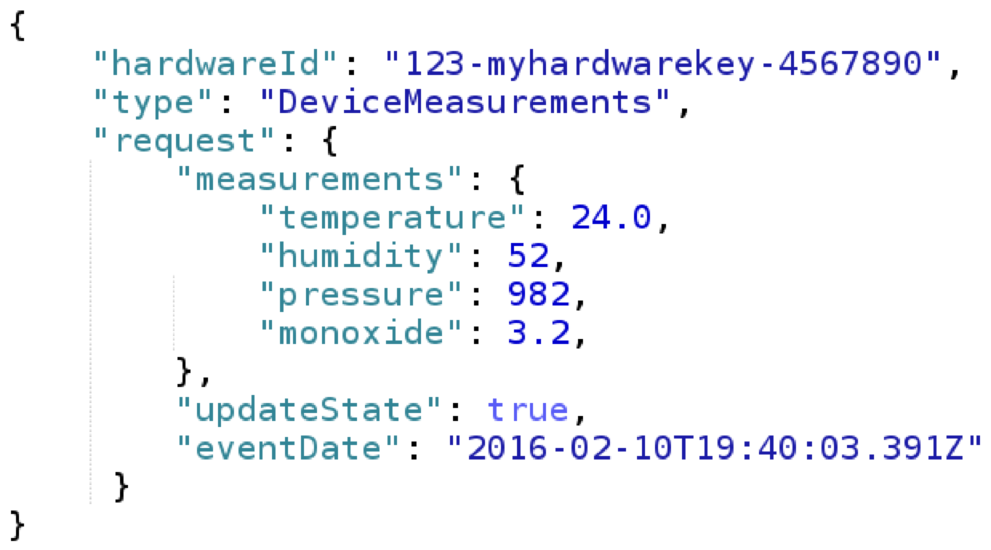
SiteWhere provides a complete devices management system. Devices are created based on specifications along with unique information such as a hardware id and device-specific metadata. Device assignments allow devices to be associated with physical assets, such as people or physical items. For instance, a badge may be associated with the person wearing it. A tracking device can be associated with the physical resource it is attached to.

SiteWhere receives data from the devices through MQTT, AQMP, Stomp or HTTP. The received data have to be encoded in JSON format and can be of three types:

- Location: the geographical coordinates of a device
- Measurements: sensor measurements
- Alerts: alerts sent by the devices

All the data are stored in a suitable non-relational database and can be accessed through a set of REST APIs. SiteWhere permits also to send commands to the devices [41].

SiteWhere receives data from the SwarmBike device through the MQTT protocol, subscribing to the topic “SiteWhere/input”. All the three types of data can be sent by the device. An example of received JSON can be seen in figure 4.21.



```
{
    "hardwareId": "123-myhardwarekey-4567890",
    "type": "DeviceMeasurements",
    "request": {
        "measurements": {
            "temperature": 24.0,
            "humidity": 52,
            "pressure": 982,
            "monoxide": 3.2,
        },
        "updateState": true,
        "eventDate": "2016-02-10T19:40:03.391Z"
    }
}
```

Figure 4.21. JSON with measurements sent by the device

The hardwareId field contains a key which uniquely identifies every device. Currently, the MAC address of the bluetooth module of the device is used as identifier. The other fields are self-explaining.

4.4.2 Cloud Backend Application Implementation

The cloud backend application, is based on the Meteor framework. Meteor [21] is a full-stack JavaScript platform for developing modern web and mobile applications. It uses MongoDB as data storage.

Meteor permits to develop, using only JavaScript, in all environments: application server, web browser (client), and mobile device. The applications developed with Meteor are reactive: when the server DB is modified, the modifications are synchronized among all the clients connected to that server. Furthermore Meteor applications compensate for the latency: in fact when a modification is done on the client-side, the UI immediately reflects it, without waiting for server's confirmation. Lastly, the communication between Meteor apps is done using the DDP protocol (more on this in chapter 3.1.3).

SwarmBike Meteor Application

The SwarmBike Meteor application has three main purposes:

- building a map on which the position of the device can be seen
- building a map which shows for each user the CO measurements acquired during the daily bike rides
- sending push notifications to the mobile application in case of a theft alert
- sending push notifications if, after the anti-theft has been enabled, no data are received via MQTT for a certain amount of time (5 minutes in this implementation)

The Meteor application subscribes to a MQTT topic on which the device publishes all the theft alerts and, when the anti-theft is enabled, also its geographical coordinates. The map is built using the Leaflet.js library, to permit a real-time update of the map the DDP protocol and websockets are exploited. The map which shows the CO measurements for each user, on the other hand, is built by the Meteor application leveraging on the data collected by SiteWhere, which are accessed through the REST APIs. Leaflet.js is used to build this map too.

Regarding push notifications for theft attempts and data timeout, the SwarmBike Meteor application works in two phases: firstly, it stores the notificationID sent by the mobile app on the first use. Then, when a theft alert is received via MQTT, it sends a request to the Google Cloud Messaging server (using HTTP POST), indicating the previously stored notificationID. The GCM server will then take care of delivering the notification to the smartphone associated with the notificationID.

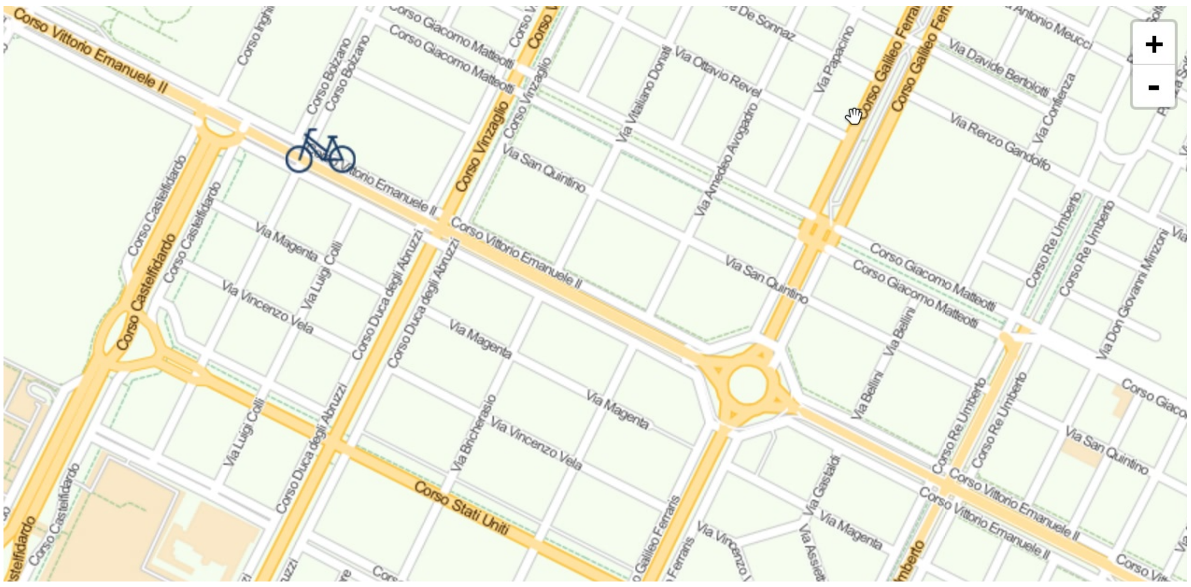


Figure 4.22. The position of the bike can be remotely visualized on a map

Chapter 5

Testing and Validation

5.1 Testing

At the end of the implementation phase, the whole system has been subjected to a test phase. Two pieces of every hardware component have been bought, permitting to build two different specimens of the SwarmBike device. The two prototypes have been used and tested by different people, who reported all the problems and bugs they found. The prototypes have been placed on two bikes (inside the bikes' basket) and the testers simply took some rides to let the device gather data from the environment. The anti-theft feature has been tested by trying to move the parked bike to simulate a theft attempt.

The Android application has been tested on different models of smartphone with different Android versions (5.1 Lollipop and 6.0 Marshmallow).

This testing stage has permitted to verify that:

- the SwarmBike system worked correctly for the three main use cases (acquisition of pollution data, remote localization of the device and accelerometer based anti-theft)
- the SwarmBike system satisfied all the requirements identified during the design and implementation stages
- the SwarmBike system responded correctly to all the inputs
- the SwarmBike system performed all the activities in an acceptable time
- the SwarmBike system was stable and usable enough

5.2 Size and weight of the Device

As stated, the realized device is just a prototype. However, it was decided to build a cardboard case to help carrying and protect the prototype during the tests and to make it look better for demo reasons. The dimensions of the case are 20.5 x 12.5 x 9 cm, consequently it can be easily put in a bike's basket. Including a 10000mAh powerbank, the device has a weight of about 650 grams, which is perfectly acceptable to carry on a bike. Therefore, the weight and size requirements have been satisfied.

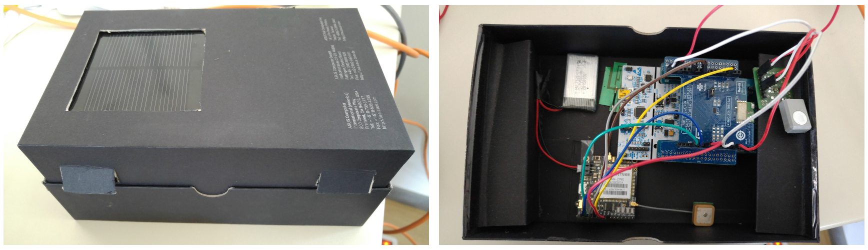


Figure 5.1. The cardboard case. On the right, the SwarmBike device inside the case

5.3 Device Battery Life

To measure the devices' battery life, it was decided to perform the following test:

1. A 10000mAh powerbank with two output ports has been full charged and so has been the additional 500mAh which supplies power to the FONA808
2. The prototyping board has been connected to the first port of the powerbank, while the other port has been connected to the FONA's microUSB, to constantly recharge the additional 500mAh battery.
3. The device has been configured to send a sample every 12 seconds, which is an acceptable rate for tracking the position and sensing the pollution. From time to time, a BLE connection has been established using the mobile app to simulate a real usage.
4. The device has been let run until the powerbank was fully discharged.

The device ran continuously for 2670 minutes (44.6 hours), with an average current consumption of 224 mA. This test has been conducted with the anti-theft mode disabled, thus increasing the amount of sent data. Given the fact that SwarmBike is a prototype, this result is acceptable (considering a usage of 8 hours per day, the device would permit 5 days without a recharge). Furthermore it is worthwhile to underline that the device is currently implemented using a STM32 Nucleo prototyping board, which has some additional components (such as a built-in microcontroller programmer or some status LEDs) that increase remarkably the overall power consumption. Therefore, if in the future the device will be implemented with a custom PCB design with the removal of all the unneeded components, the battery life could improve sensibly (see paragraph 6.1).

5.4 Device Cost

Overall, the hardware pieces to build a single prototype had a total cost slightly higher than 100€, which is an acceptable price for a prototype. If in future the device will be produced in series, a massive drop in the cost of a single unit is expectable.

5.5 Samples Acquired by the CO Sensor

As stated, in the testing phase the device has been placed in a bike's basket and carried around. Figure 5.2 shows a map with the CO samples acquired during one of these rides in the city center of Turin.

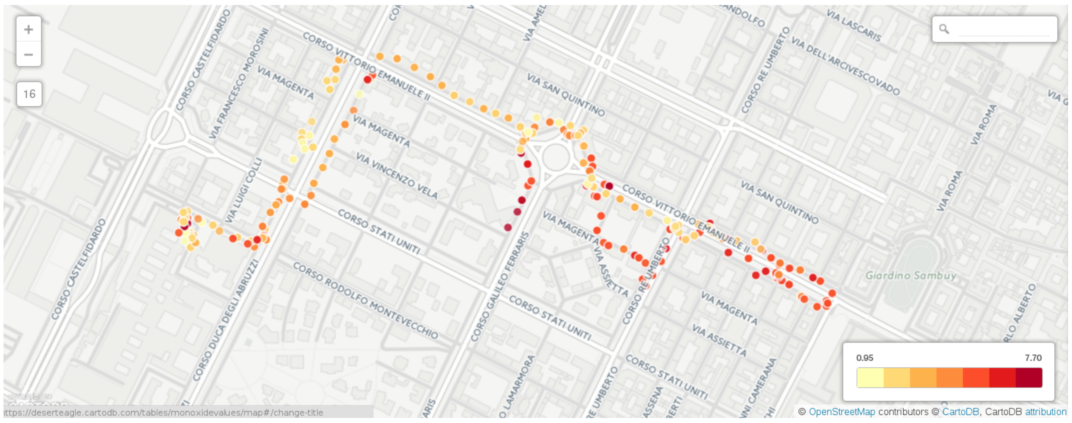


Figure 5.2. Map of CO samples acquired during a ride

Given the fact that the sensor does not need calibration, it was possible to estimate the gas concentration straightaway. The gathered data were perfectly plausible and compatible with the expected carbon monoxide concentration in a city like Turin (ranging from 1 to 8 ppm). However, a cross validation of those data with a professional high cost CO sensor would have been needed to measure the accuracy of the sensed values. Unfortunately, it was not possible to access such a professional instrumentation, therefore the data could not be cross validated. In order to remedy this flaw, it will be done a cross validation in collaboration with ARPA (see paragraph 6.1).

Chapter 6

Conclusion and future works

This thesis work has been focused on the design and the prototyping of SwarmBike, an IoT mobile platform which aims at exploiting bicycles for air pollution sensing. The SwarmBike system is based on three major components.

The first component is the device, which is the one that has to be mounted on the bike and is responsible of sensing air pollution. Many types of pollutant substances were analyzed, as well as the various sensor technologies to detect them. In the end it was decided to monitor Carbon Monoxide, by mean of a low-cost and low-power electrochemical gas sensor, which permits to gather data about pollution with an acceptable accuracy. Such a system would need many participants to be effective and it cannot rely only on volunteers, therefore it was decided to enrich the device with other features to increase users' involvement. Those features were selected based on a survey conducted among some Politecnico students and TIM employees. The chosen additional features were: the possibility to remotely view the position of the device on a map and an accelerometer based anti-theft system with smartphone notifications. The device has been realized using a STM32 Nucleo L476RG prototyping board, which ensured high performance and low power consumption. The chosen electrochemical CO gas sensor was the Nemoto NAP-505, driven by a Texas Instruments LMP91002 potentiostat. The firmware which drives the device has been implemented exploiting ARM mbed OS, an operating system developed by ARM specifically for IoT devices.

The second component is the smartphone application, which has been implemented for the Android operating system. The purpose of the mobile app is to let the user manage and configure the device, by mean of a Bluetooth Low Energy connection. For instance, the user can enable and disable the anti-theft system. Furthermore, the mobile app shows the map (provided by the cloud backend application) on which the position of the device can be remotely viewed, and it is also responsible to to alert the user when a theft-attempt notification is sent by the cloud backend.

The third component is the cloud backend, which is in turn divided into two subcomponents: the IoT server platform and the backend application. The IoT server platform is responsible for collecting and storing all the information gathered and transmitted by every device. The backend application, on the other hand, constructs the map on which the position of the device can be remotely viewed. Furthermore, when the device sends a theft alert the backend application takes care of sending the push notification to the mobile app.

The results obtained by this thesis work demonstrate that monitoring air quality by using bicycles as probes is a perfectly achievable objective. SwarmBike is a valid prototype which could be improved and extended in order to be used in production scenarios. However, some weak points and issues should be addressed in future works.

6.1 Future Works

This paragraph describes what could be done to improve the system and address the main open issues of the current prototype, including some experimental activities that have been already planned in order to better validate the SwarmBike device.

Implementation of the device on a single custom PCB

The prototype of the device is currently using a prototyping board (the STM32 Nucleo L476RG). The prototype board has some built-in components, such as the STM32 programmer or some status LEDs, which help in the prototyping stage but are not needed for the device to work. These components make the device far bigger and heavier than necessary. Furthermore, the use of prototyping shields (such as the sensor shield and the BLE shield) increases the size and the weight even more. If all the needed components would all be mounted on a single custom PCB, the size and the weight of the device would be much lower than now. Moreover, the elimination of some unneeded components would also reduce the power consumption, thus increasing battery life. Therefore, a future work should investigate the possibility of implementing the whole device on a single PCB.

Tamper-proof Mounting

Given the fact that one of the main features of SwarmBike is the anti-theft system, the device should have a tamper-proof mounting, such that it would be difficult for a theft to steal or damage it. There are some projects, such as the SmartHalo one [42], which have achieved this goal. A future work should address this problem.

Security Implementation

As stated before, some sensible information are exchanged between the SwarmBike components, such as the position of the user. Furthermore, secure communication are essential for the anti-theft feature provided by the SwarmBike system. Therefore, it is necessary to ensure that all the communication between the components are confidential, integer and authenticated. The communication between the mobile app and the cloud application is done on top of HTTPS, therefore it is already secure. Consequently, a future work should focus on making secure the MQTT connection between the device and the server and the BLE connection between the smartphone application and the device (more on this in chapter 4.2.3, paragraph "Security in the SwarmBike firmware").

Development of a Model for Building Pollution Maps and Offer Additional Services

Pollution maps could be built leveraging on the collected air quality data. However, as proved by [9] and [17], it is essential to develop a mathematical model for building pollution maps with a high spatio-temporal resolution, which is needed to offer additional services (e.g. suggestion on less polluted paths). A large quantity of samples is needed for the development of such a model (tens of millions), which should be acquired in different pollution, seasonal and meteorological conditions. Therefore a future activity should:

1. Involve as many users as possible, in order to speed up the development and acquire enough samples in different condition

2. Identify which services could be implemented leveraging on the acquired samples (e.g., air pollution forecast for most used bike paths) and develop an appropriate model (e.g., a forecast model leveraging both on the historical data collected by the SwarmBike system and latest data provided by the existing fixed stations)

Gamification to Increase Users' Involvement

As stated before, the higher is the amount of users using SwarmBike, the better is the monitoring of air quality. Consequently, it would be necessary to involve as many users as possible. A gamification mechanism, in which users receive a prize depending on the quantity of gathered data, could help reaching this goal (for instance, a user could receive a phone recharge proportional to the time he has spent riding around with his bike). A future work should hence focus on the development of such a strategy [2].

Experimentation with NO2 Monitoring, in Collaboration with ARPA

In collaboration with Arpa (Agenzia Regionale per la Protezione Ambientale) Piemonte [4], two specimens of the SwarmBike device will be placed next to a NO2 monitoring station, in order to cross-validate the values measured by the sensor. The experimentation will last for 6 months. Given the fact that NO2 will be monitored in place of Carbon Monoxide, the Nemoto NAP-505 sensor will be substituted by a 3SP_NO2_20, an electrochemical NO2 sensor produced by SPECSensors [43].

Experimentation in Quartiere Campidoglio in Turin

SwarmBike has been selected for an experimentation which will be done in Quartiere Campidoglio in Turin, in the context of the Torino Living Lab experimentation, an initiative launched by the city council [14]. Some SwarmBike devices will be mounted on the bicycles of several pony express which will carry them around. This experimentation will help testing the whole SwarmBike system with a higher number of users.

Bibliography

[1] Eurocircuits. http://www.eurocircuits.com. Accessed: 2016-06-22.

[2] Evangelia Anagnostopoulou, Efthimios Bothos, Babis Magoutas, Johann Schrammel, and Gregoris Mentzas. Persuasive technologies for sustainable urban mobility. *arXiv preprint arXiv:1604.05957*, 2016.

[3] Arduino. Arduino leonardo. https://www.arduino.cc/en/Main/ArduinoBoardLeonardo. Accessed: 2016-06-22.

[4] ARPA. Agenzia regionale per la protezione ambientale. https://www.arpa.piemonte.gov.it/. Accessed: 2016-06-27.

[5] Smart Citizen. Smart citizen kit. https://smartcitizen.me/. Accessed: 2016-06-27.

[6] EEMBC. Eembc ulpbench. http://www.eembc.org/ulpbench/. Accessed: 2016-06-22.

[7] Air Quality Egg. Air quality egg. http://airqualityegg.com/. Accessed: 2016-06-27.

[8] EveryAware. Sensorbox. http://www.everyaware.eu/activities/case-studies/air-quality/. Accessed: 2016-06-27.

[9] David Hasenfratz, Olga Saukh, Christoph Walser, Christoph Hueglin, Martin Fierz, Tabita Arn, Jan Beutel, and Lothar Thiele. Deriving high-resolution urban air pollution maps using mobile sensor nodes. *Pervasive and Mobile Computing*, 16:268–285, 2015.

[10] HiveMQ. Mqtt security fundamentals: Tls / ssl. http://www.hivemq.com/blog/mqtt-security-fundamentals-tls-ssl. Accessed: 2016-06-20.

[11] Adafruit Industries. Adafruit fona808 overview. https://learn.adafruit.com/adafruit-fona-808-cellular-plus-gps-breakout/overview. Accessed: 2016-06-22.

[12] Texas Instruments. Lmp91002 sensor afe system: Configurable afe potentiostat for low-power chemical sensing applications. http://www.ti.com/lit/ds/symlink/lmp91002.pdf, 2015.

[13] Marilena Kampa and Elias Castanas. Human health effects of air pollution. *Environmental pollution*, 151(2):362–367, 2008.

[14] Torino Living Lab. Torino living lab. http://torinolivinglab.it/. Accessed: 2016-06-27.

[15] ARM Ltd. mbed os. https://www.mbed.com/en/development/software/mbed-os/. Accessed: 2016-06-22.

[16] Nemoto Sensor Engineering Company Ltd. Operating characteristics and handling manual for the nap-505 carbon monoxide gas sensor. http://www.nemoto.eu/nap-505-manual.pdf, 2016.

[17] Ali Marjovi, Adrian Arfire, and Alcherio Martinoli. High resolution air pollution maps in urban environments using mobile sensor networks. In *2015 International Conference on Distributed Computing in Sensor Systems*, pages 11–20. IEEE, 2015.

[18] Jacob Mason, Lew Fulton, and Zane McDonald. A global high shift cycling scenario. 2015.

[19] ARM mbed. Ble modes and profiles. https://docs.mbed.com/docs/ble-intros/en/latest/Introduction/BLEInDepth/. Accessed: 2016-06-20.

[20] MI Mead, OAM Popoola, GB Stewart, P Landshoff, M Calleja, M Hayes, JJ Baldovi, MW McLeod, TF Hodgson, J Dicks, et al. The use of electrochemical sensors for monitoring urban air quality in low-cost, high-density networks. *Atmospheric Environment*, 70:186–203, 2013.

[21] Meteor. Meteor, the fastest way to build javascript apps. https://www.meteor.com/. Accessed: 2016-06-27.

[22] Meteorhacks. Introduction to ddp. https://meteorhacks.com/introduction-to-ddp/. Accessed: 2016-06-27.

[23] Carmelo Migliore. cmsis-core-stm32l4. https://github.com/carmelomigliore/cmsis-core-stm32l4. Accessed: 2016-06-22.

[24] Carmelo Migliore. cmsis-core-stm32l476rg. https://github.com/carmelomigliore/cmsis-core-stm32l476rg. Accessed: 2016-06-22.

[25] Carmelo Migliore. mbed-hal-st-stm32cubel4. https://github.com/carmelomigliore/mbed-hal-st-stm32cubel4. Accessed: 2016-06-22.

[26] Carmelo Migliore. mbed-hal-st-stm32l4. https://github.com/carmelomigliore/mbed-hal-st-stm32l4. Accessed: 2016-06-22.

[27] Carmelo Migliore. mbed-hal-st-stm32l476rg. https://github.com/carmelomigliore/mbed-hal-st-stm32l476rg. Accessed: 2016-06-22.

[28] Carmelo Migliore. mqttclient. https://github.com/carmelomigliore/mqttclient. Accessed: 2016-06-22.

[29] Carmelo Migliore. sal-iface-serialmodem. https://github.com/carmelomigliore/sal-iface-serialmodem. Accessed: 2016-06-22.

[30] Carmelo Migliore. st-nucleo-l476rg-gcc. https://github.com/carmelomigliore/st-nucleo-l476rg-gcc. Accessed: 2016-06-22.

[31] World Health Organization. Dioxins and their effects on human health. http://www.who.int/mediacentre/factsheets/fs225/en/. Accessed: 2016-06-13.

[32] World Health Organization et al. Who air quality guidelines for particulate

matter, ozone, nitrogen dioxide and sulfur dioxide: global update 2005: summary of risk assessment. 2006.

[33] Eclipse Paho. Embedded mqtt c/c++ client libraries. https://eclipse.org/paho/clients/c/embedded/. Accessed: 2016-06-27.

[34] Kaa Project. What is the internet of things platform? http://www.kaaproject.org/iot-101-what-is-an-iot-platform/. Accessed: 2016-06-20.

[35] India Air Quality. Measuring the pickle jr. – a modified ppd42 with an attached fan. https://indiaairquality.com/2014/12/14/measuring-the-pickle-jr-a-modified-ppd42-with-an-attached-fan/. Accessed: 2016-06-13.

[36] Ole Raaschou-Nielsen, Zorana J Andersen, Rob Beelen, Evangelia Samoli, Massimo Stafoggia, Gudrun Weinmayr, Barbara Hoffmann, Paul Fischer, Mark J Nieuwenhuijsen, Bert Brunekreef, et al. Air pollution and lung cancer incidence in 17 european cohorts: prospective analyses from the european study of cohorts for air pollution effects (escape). *The lancet oncology*, 14(9):813–822, 2013.

[37] Guido Schattanek, P Wan, A Kasprak, and Helen Ginzburg. Carbon monoxide and nitrogen oxides relationships measured inside a roadway tunnel and a comparison with the mobile 6.2 emission model predictions. In *Presented paper at the AWMA Annual Meeting. Minneapolis, Minnesota*, 2005.

[38] Anthony Seaton, D Godden, W MacNee, and K Donaldson. Particulate air pollution and acute health effects. *The lancet*, 345(8943):176–178, 1995.

[39] Shinyei. Ppd42ns specification sheet. http://www.seeedstudio.com/wiki/images/4/4c/Grove_-_Dust_sensor.pdf, 2010.

[40] Bluetooth SIG. Bluetooth sig adopted specifications. https://www.bluetooth.com/specifications/adopted-specifications. Accessed: 2016-06-20.

[41] SiteWhere. System overview. http://documentation.sitewhere.org/overview.html. Accessed: 2016-06-22.

[42] SmartHalo. Smarthalo. https://www.smarthalo.bike/. Accessed: 2016-06-22.

[43] SPECSensors. No2 nitrogen dioxide sensor 20 ppm. http://www.spec-sensors.com/product/no2-sensor/. Accessed: 2016-06-27.

[44] OASIS Standard. Mqtt version 3.1. 2014.

[45] STMicroelectronics. Stm32 mcu nucleo. http://www.st.com/content/st_com/en/products/evaluation-tools/product-evaluation-tools/mcu-eval-tools/stm32-mcu-eval-tools/stm32-mcu-nucleo.html?querycriteria=productId=LN1847. Accessed: 2016-06-22.

[46] STMicroelectronics. Stm32 nucleo l476rg. http://www.st.com/content/st_com/en/products/evaluation-tools/product-evaluation-tools/

mcu-eval-tools/stm32-mcu-eval-tools/stm32-mcu-nucleo/nucleo-l476rg.html?icmp=pf261636_pron_pr_sep2015&sc=nucleo-l476rg. Accessed: 2016-06-22.

[47] STMicroelectronics. Stm32l4 series. http://www.st.com/content/st_com/en/products/microcontrollers/stm32-32-bit-arm-cortex-mcus/stm32l4-series.html?querycriteria=productId=SS1580. Accessed: 2016-06-22.

[48] STMicroelectronics. X-nucleo-idb05a1. http://www.st.com/content/st_com/en/products/ecosystems/stm32-open-development-environment/stm32-nucleo-expansion-boards/stm32-ode-connect-hw/x-nucleo-idb05a1.html. Accessed: 2016-06-22.

[49] STMicroelectronics. X-nucleo-iks01a1. http://www.st.com/content/st_com/en/products/ecosystems/stm32-open-development-environment/stm32-nucleo-expansion-boards/stm32-ode-sense-hw/x-nucleo-iks01a1.html. Accessed: 2016-06-22.

[50] STMicroelectronics - Pierre Sennequier. *Signal conditioning for electrochemical sensors*. Application Note AN4348. November 2013.

[51] Darell Tan. Testing the shinyei ppd42ns. http://irq5.io/2013/07/24/testing-the-shinyei-ppd42ns/. Accessed: 2016-06-13.

[52] Cristian Tanas and Jordi Herrera-Joancomartí. When users become sensors: can we trust their readings? *International Journal of Communication Systems*, 28(4):601–614, 2015.

[53] Dr. Mike Thompson. Carbon monoxide. http://www.chm.bris.ac.uk/motm/co/coh.htm. Accessed: 2016-06-13.

[54] TRACE. Trace: Walking and cycling tracking services. http://h2020-trace.eu/. Accessed: 2016-06-20.

[55] TZOA. Tzoa. http://www.tzoa.com/. Accessed: 2016-06-27.

[56] Alice S Whittemore. Air pollution and respiratory disease. *Annual review of public health*, 2(1):397–429, 1981.

[57] Wei Ying Yi, Kin Ming Lo, Terrence Mak, Kwong Sak Leung, Yee Leung, and Mei Ling Meng. A survey of wireless sensor network based air pollution monitoring systems. *Sensors*, 15(12):31392–31427, 2015.