A Self-organized Wireless Sensor Network (WSN) for a Home-event Managed System

Design of a cost efficient 6LoWPAN-USB Gateway with RFID security

Sergio A. Floriano Sanchez
sergiofloriano@gmail.com

June 29, 2015

Supervisor: Mark Smith.

School of Information and Communication Technology
Royal Institute of Technology

Stockholm, Sweden

Abstract

Wireless Sensor Networks (WSN) have existed for many years in industry applications for different purposes but their use has not been fully extended to the global consumers. Sensor networks have lately resulted to be greatly helpful to people in everyday life, specially in home automation applications for monitoring events, security, and control of devices and different elements in the house by using actuators.

One of the main barriers to overcome in order to increase their popularity and achieve an worldwide deployment are costs and integration within other networks.

This Thesis investigates the most appropriate choices to avoid those impediments from a hardware and software design perspective, trying to find a cost-efficient solution for the implementation of a simple and scalable wireless sensor network. The present work studies the elements that form part of a constrained network and focuses on the design by analysing several network protocol alternatives, radio transmission mechanisms, different hardware devices and software implementations. Once an optimal solution is found, the construction of a gateway board that starts and coordinates a sensor network will be the main target of this document including the development of an application that manages the sensors.

The network is designed to be compliant with the TCP/IP stack by means of 6LoWPAN, an adaptation layer protocol used for comprising IPv6 headers over IEEE 802.15.4 radio links in constrained networks. In addition, a small implementation of CoAP (Constrained Application Protocol) is developed that allows interoperability with the sensor nodes on the application layer, similarly as HTTP does in IP networks. The controller device (gateway) acts as a client for the remote sensor devices (nodes) that behave as servers in the CoAP application. The gateway exchange data and is managed from outside the WSN through a USB interface that can be connected to a computer.

Security mechanisms are also considered by providing packet encryption and a method for identification of nodes. The authorization of new nodes entering the network is performed by an RFID reader connected to the gateway. An RFID tag is attached to the sensor nodes with authentication information stored in it. The gateway reads that information through the RFID modules and handle it internally to give access to that node.

As a result of this, it is proven from the conclusions of the study the implementation of the gateway that inexpensive, self-managed, scalable WSNs provided with a robust security mechanism can be achieved and easily deployed .

The work presented in this document is part of a larger project that also includes the design of sensor boards and the acquisition and analysis of sensor data. These works are mentioned and referenced in the related parts in this text.

Sammanfattning

Trådlösa sensornätverk har funnits i många år inom industrin för olika ändamål, men dess användning har inte helt och hållet nått ut till de globala konsumenterna. Sensornätverk har på senare tid visat sig vara mycket hjälpsamma för människor i deras vardagsliv, särskilt inom automatiseringsapplikationer för säkerhet, övervakning och kontroll av apparater och olika delar i huset, genom användning av manöverdon. Ett av de huvudsakliga hindren att ta sig förbi för att kunna öka dess popularitet och skapa en världsomfattande spridning är kostnader, integration inom andra nätverk och en enkel hantering.

I den här avhandlingen undersöks vilka som är de lämpligaste alternativen för att undvika hinder ur ett hårdvaru- och mjukvarudesigns-perspektiv, genom att försöka hitta kostnadseffektiva lösningar för implementering av ett trådlöst sensornätverk. Arbetet undersöker de beståndsdelar ett begränsat nätverk består av, samt fokuserar på designen genom att analysera flera olika nätverksprotokollsalternativ, radiosändningsmekanismer, olika hårdvaror och implementering av mjukvara. När väl den optimala lösningen hittats, kommer huvudmålet för detta dokument att vara en gateways konstruktion, vilken sätter igång och koordinerar ett sensornätverk, samt utvecklingen av en applikation som sköter sensorerna.

Nätverket är designat för att vara medgörligt med TCP/IP-stacken med hjälp via 6LoWPAN, ett anpassat lagerprotokoll vilket används för att komprimera IPv6-headern i begränsade nätverk över IEEE 802.15.4 radionätverk. Dessutom har en liten implementering av CoAP (Constrained Application Protocol) utvecklats vilket tillåter interoperabilitet med sensornoderna i applikationslagret, liknande HTTP i IP-nätverk. Gatewayen fungerar som en klient för sensornoderna, vilka beter sig som servrar i CoAP-applikationen. Gatewayen utbyter data och styrs utifrån det trådlösa sensornätverket genom ett USB-interface som kan kopplas till datorn. Säkerhetskonstruktioner tas också i akt genom att tillhandahålla kryptering och en metod för att identifiera noder. Behörighet för nya noder i nätverket utförs av en RFID-läsare som är kopplad till gatewayen. En RFID-bricka bifogar sensornoderna med lagrad verifieringsinformation. Porten läser den informationen genom RFID-moduler och hanterar den internt för att ge behörighet till noden. I och med detta är det bevisat, med den implementerade gatewayen och slutsatser från studien, att mycket effektiva, billiga och hanterbara trådlösa sensornätverk med kraftiga säkerhetskonstruktioner kan uppnås och enkelt distribueras.

Arbetet som presenteras i det här dokumentet är en del av ett större projekt som också inkluderar uppbyggnaden av sensornoderna samt anskaffning och analys av sensordata.

Dessa arbeten nämns och refereras till i de berörda delarna av texten.

Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

6LoWPAN	IPv6 over low-power wireless area networks
ACG	Automatic gain control
API	Application Programming Interface
ASK	Amplitude Shift Keying
BPSK	Binary Phase Shift Keying
DSSS	Direct-sequence Spread Spectrum
EMC	Electromagnetic Compatibility
EPC	Electronic Product Code
ESD	Electrostatic Discharge
FHSS	Frequency-hopping Spread Spectrum
FSK	Frequency Shift Keying
FSK	Frequency-shift Keying
GSM	Global System for Mobile Communication
HTTP	Hypertext Transfer Protocol
I^{2}C	Inter-Integrated Circuit
IANA	Internet Assigned Numbers Authority
IEEE	Institute of Electrical and Electronic Engineers
IETF	Internet Engineering Task Force
ISM	Industrial, Scientific and Medical

ISO	International Organization for Standardization
NFC	Near Field Communication
NRZ	Non Return to Zero
OOK	On-Off keying
O-QPSK	Offset Quadrature Phase-shift Keying
PIE	Pulse-Interval Encoding
PSK	Phase Shift Keying
PSSS	Parallel-sequence Spread Spectrum
RFID	Radio Frequency Identification
SPI	Serial Port Interface
TI	Texas Instruments
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
WSN	Wireless Sensor Network

Chapter 1

Introduction

In the last few decades humanity has experimented an incredible change. The way people communicate with their environment has been completely transformed. The direction that this change is taking leads to a world where people not only interact with each other but also with devices and machines. We are all integrated, taking part of networks. Devices are continuously getting connected as well to networks. New needs are emerging and Wireless Sensor Networks (WSN) are located in a very remarkable place [21].

Sensors are in contact to the real world, they are embedded, sensing and detecting a huge range of events and variables. Physical phenomena are measured and translated into digital information. Sensors also collaborate with actuators that provide actions depending in many cases on the values coming from those sensors. Information does not have to be sent as raw data, devices could also perform some kind of processing and interpretation of data and send notifications when interesting events occur. In a home-event managed system this data is collected from sensors located in different rooms, delivered by some self-managed mechanisms to a central unit and eventually processed and interpreted. This information is presented to the user as a set of events occurring at a certain time with very interaction and maintenance needed from the user.

1.1. Structure of this thesis

The contents of this section should serve as a guide through the rest of the document. It states the aim of each chapter so the reader can directly access to the most interesting ones for him.

Chapter 1 Is an introduction to the main topic discussed in this thesis. It briefly describes the purposes and benefits of Wireless Sensor Networks and the need of it.

Chapter 2 Defines the goal and proposes a detailed solution describing the requirements in the end. It gives an insight on how to proceed in the design and selection of materials

Chapter 3 Describes different RFID technologies and alternatives. It also shows the characteristics of several network protocols and discussed about the applicability on constrained sensor networks.

Chapter 4 goes through different topologies, network protocols and implementation alternatives in order to find the most adequate solution for the proposed requirements. Additionally it discusses and arguments about the integration of security in the platform through RFID technology.

Chapter 5 is dedicated to the physical construction of the gateway. It shows the results of a research carried through different processors and radio, USB and RFID modules in order to find the most cost-efficient and feasible solution fulfilling the requirements.

Chapter 6 describes the software implementation of the systems based on Contiki OS. The reader will find a description of the internal functionality of Contiki and its requirements, a short description of the firmware for the different hardware modules, the implementation of the network functionality and a wider description of the application and the way the different modules interact with each other through Contiki processes.

Chapter 7 discusses about the results obtained with our design to verify if the main goal and the requirements has been achieved. It states a conclusion in the end.

Chapter 8 proposes different ideas and ways to continue the work started in this thesis.

1.2. Motivation

The seed of the present Thesis was originally provided by Ericsson Research. A department within Ericsson interested in a Wireless Sensor Networks capable of obtaining data (sensors, state of devices, presences...) from different sources that should be interpreted and inform the users. Wireless Sensor Networks are continuously becoming widely popular since the last few years. People security needs have been increasing in parallel as well. Residents want to have homes under control, specially during absence for long periods. Furthermore, simple data values from sensors might be useless without a proper interpretation and users are not usually willing to do that type of task. They typically prefer instant and valuable information, an answer to a question rather than a numeric value or a change of state of a certain device.

The interest is not that a user knows the exact temperature inside a room or the power consumption of a certain device. Moreover, it was required that, depending on the captured data, the system would be able to answer questions such as: Is there someone at home? Everything is going all right in the kitchen? Or, additionally, been able to capture event triggered by different situations (your daughter has arrived home, the fridge has unexpectedly switched down...) Therefore, different applications were thought to provide 'assurance' related to energy consumption control, safety against fire or theft, presence control. Applications might be varied and the system might provide flexibility to increase the number of possible new applications.

From a technical point of view, the system was conceived originally to have multiple and different kinds of sensors connected to a self-managed wireless network, meaning that the network does not need to be controlled by the user. The network configures itself, connect the devices and recover from possible faults. No need for user to interact and manage the network.

Specifically three types of streams were considered from the beginning: temperature, power consumption of devices in the house and door lock event detection. Finally some more sensors were added to measure humidity and light, and an accelerometer. The data generated is delivered to a base station (gateway) connected to the WASA board (a prototyping board developed by Prof. Mark Smith [2]) or any other board. An external system must be responsible of collecting the necessary information from the gateway. The external system or client have to deal with data parsing and processing in order to feed the different applications offered to the end user.

The global project, as a whole, was too big for just one Master Thesis; it was therefore divided into four smaller projects responsible each for a specific role:

1. Sensors [3]
2. Self-organized Wireless Sensor Network (WSN) [In this document]
3. Applications <-> Events, associated actions
4. Statistical analysis (normalizing)

The team was originally composed by a group of four people collaborating respectively with a Master Thesis each. The present Thesis will be in charge of the design and implementation of the Self-organizing Wireless Sensor Network (WSN). Compared to the OSI model, the work will be made on several of the layers, taking part from the data link layer to the application layer that should stablish a way of communication from a client to a server and vice versa.

The official common name of the project was agreed to be "Home-event Management System" particularizing each Thesis with a description: "Design and Implementation of a Wireless Sensor Network for a Home-event Management System".

1.3. Problem statement

From the first moment when the idea of the project arose, the trajectory has moved through different stages until finding out the final objective and real applicability of it. Not only globally as a whole body, but also as individual Theses. It needs now to be made clear enough the purpose of the present work.

The target to achieve consists of designing a self-managed Wireless Sensor Network. Other required network features include low-power, low data transfer, low-cost and easy maintenance. The WSN will be implemented starting with the design and building of devices and later focusing on the networking protocol and the application that send data from the sensors. The network should find a way to establish a bidirectional communication from outside to the sensor network and vice versa. It should allow requests to be sent to sensors and replies to the origin of the request, performing some kind of security and quality of service (e.g. retransmission mechanisms). Sensors could also decide if they should inform when events occur by sending messages. The information exchanged from the WSN will be received in a transparent manner to a database or to an external client through a gateway. Data will be interpreted and treated by the receiver.

Additionally a method for identification of new nodes appearing in the network should be added by means of different technologies such as RFID or NFC.

1.4. Wireless Networks

WSNs commonly consists of several sensor devices usually called sensor boards, nodes, motes or hosts [23]. Sensor devices are connected to other nodes by a radio link at a certain frequency and using a protocol according to the requirements of the application. Nodes perform simple actions like sending raw data, or might provide more complex features such as routing to allow multi hop between nodes out of the range. Gateways, border routers or similar are also commonly found when interacting with sensor networks.

Transmissions from nodes are generally done hop-by-hop through other nodes following a defined route (using routing mechanisms) or not defined route (flooding, retransmissions) until finding the destination, often a gateway to external networks. There are many different mechanisms to achieve a successful and optimal communication, all dependent of the implemented protocol.

There is a huge large number of protocols out in the market that provide connectivity and management capability to WSNs. They usually offer in addition security and reliability functionalities such as ciphering, encryption keys management, acknowledge messaging and retransmission. Some protocols are also built over other well known standards at low layers such as the IEEE 802.15.4 [32] for radio links that implements very useful mechanisms to effectively adapt to the restrictions found in WSNs.

Network protocols allow different network topologies that could match into a WSN. Among them there are three popular network schemes [23]:

- Start network: there is a central device (gateway, base station) where all nodes are connected through. It is responsible of coordinating and managing the network. Nodes communicate exclusively with the coordinator device. This kind of network offers simplicity and fast data exchange but it is totally dependant of one device and poorly flexible.

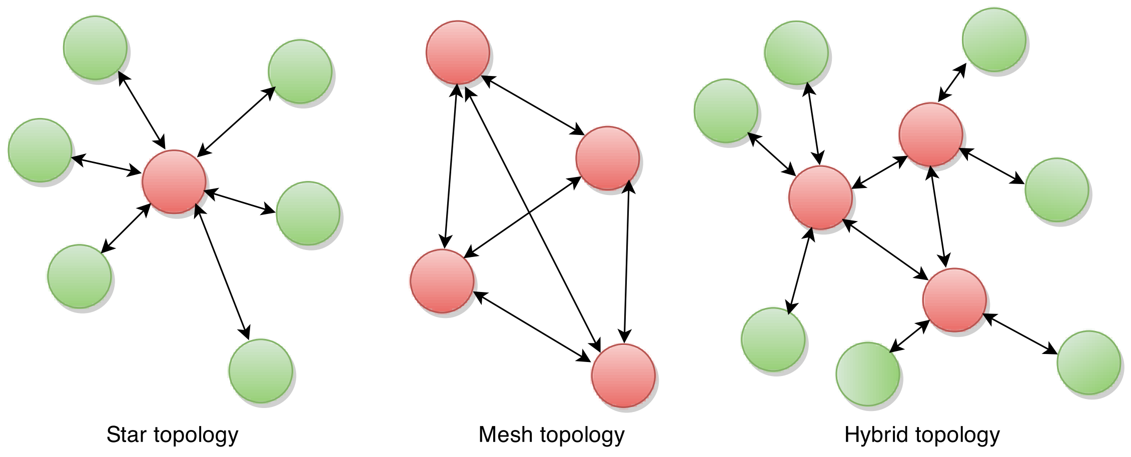


Figure 1.1: Different network topologies. In red those nodes with more with extra functionality acting as controllers or gateway. In green simple nodes reading sensor data

- Mesh network: All nodes are connected together and are able to transmit to others within the radio coverage range. It is offers great redundancy and scalability but management could become very complex affecting device processing what increases latency of messages and power consumption.
- Hybrid network: This a combination of the topologies above. There are more than one node with managing and multi hoping capabilities. Other nodes are simple and limited to transmission of messages saving power consumption. Hybrid networks share advantages and disadvantages making it more flexible offering a trade-off of features.

A detailed specification of the application is required in order to optimize the network for its future use. That will also help in the selection of the most appropriate communication protocol, hardware needs. There are many different choices for protocols and platforms offering very good features, but an the optimal will be the one that fixes perfectly with the requirements.

1.4.1. Common features

Some features are commonly required in sensors networks but they imply some constraints that need to be overcome to get the desired properties. Those are **low cost**, **low power consumption** and **minimum maintenance**.

In a WSN it is likely to have tens or even hundreds of devices. It is therefore crucial that cost is reduced to the lowest possible otherwise networks would be unaffordable for the regular customers. Batteries should last periods of years to reduce the cost and also to avoid the necessity of changing the batteries of the devices too often. This is a matter of maintenance and also sustainability since consumed batteries create disposal and toxic waste problems. Sensor networks should be also easy to be deployed and easy to be maintained to avoid the consumer to concern about technical issues making WSNs accessible to anyone. Robust protocols, adaptability and smartness are intended.

Due to the restrictive nature of sensor networks they typically have in common a series of characteristics [24] that lead to achieve these three requirements.

- Slow processing capability: no big data needs to be processed so usually microprocessors with 8-bits or 16-bits addressing are built in the devices with clock frequency around 10 MHz. Slow processors should also allow for a better power consumption efficiency since they consume less. Normally prices are economic and affordable but it is not necessarily a rule.
- Reduced bandwidth: since small data information packets are to be transmitted there is no need of high data rates. Typically 10 Kbps - 60 Kbps throughput in Contiki[42]. So the energy consumption is usually quite efficient but this of course depends on the protocol in use. For instance Bluetooth is low bandwidth and not very low power.
- Efficient energy management: radio transceiver and microprocessor may be put on sleep mode at some intervals of time depending on the application needs. That simple action increases batteries life enormously.
- Small data storage capacity: rarely more than 70 Kb of flash and 16 Kb of RAM [42] [23]. An efficient and small code will get better consumption behaviour but small memories can be quite expensive if there is not big demand for them.
- Routing protocol: the implemented protocol should let the network to be self-configured and self-managed, to get long distances by using routing paths and to be easily scalable.
- Security and Reliability: typically the protocol implementation cares but it. Some mechanisms are usually required to set the packet loss to acceptable ratios and avoid eavesdropping from third part listeners or hacking.

1.4.2. Advantages and disadvantages

The benefits of deploying a wireless network instead of a wired network are mainly based on low installation costs, minimum maintenance and easier scalability and location flexibility [21].

The nowadays increase of use of microprocessors, wireless devices and mass-production of electronic devices have helped also to optimize cost. The trend is to keep lowering prices in the future. That will boost scalability of networks providing extensive deployments.

Sensor networks are not just targeted to provide numerical data and raw info to users. The goal is to perform some kind of useful high level information from raw data already interpreted. Users are therefore aware the interpretations of what is going on rather than values or measurements.

Sensor networks are very useful when registering local events or obtain physical measurements such as weather conditions. Usually the measured values are limited to a reduced range, what improves the SNR (signal-to-noise ratio). Moreover, WSNs generally consist of a large collection of devices so a better resolution and description of local areas can be achieved.

It is also important to notice that due to the short distance range that is achieved with sensor networks (10 meters), typically a great amount of sensors has to be deployed in order to extend the range (this is valid for mesh and hibrid networks) [21]. Unfortunately, it complicates the networking protocols. Self-configuration capability on networks is essential, specially to avoid the necessity of manual maintenance.

Many sensor networks work at high frequencies inside the 800 MHz - 900 Mhz bands or at 2.4 GHz. High frequency links are more affected to disturbances than low frequencies. Considering that they are part of a low power network, the communication channel could become very sensitive and unreliable [21]. Objects, people, antenna orientation or weather could easily affect to connectivity causing momentary loss of link and routing easier. In those cases networks should adapt to the new conditions by self-organizing itself and looking for new routes, adapting dynamically.

A great amount of extra advantages and disadvantages can be found apart from those explained above. There is no need to enumerate all of them but this is just a guide of those considered the most remarkable ones.

1.4.3. Uses

WSNs take part in many diverse fields and applications. The fact that they have been present for years, nowadays its uses are increasing exponentially due mainly to cost decreases and appearance of new protocols specially designed for this kind of networks.

A generic list of applications where WSNs can be helpful would include: home automation, environmental monitoring, disasters detection, security applications (surveillance, alarms, presence detection), healthcare, emergencies, products tracking. As it can be seen, not only home environments are targeted by sensor networks. Uses are only limited by distance ranges, data rates and imagination since it is an extremely versatile type of communication network.

1.4.4. Devices

Sensor networks usually comprise a central node that may be a gateway or border router, several sensor devices or nodes and nodes with routing capabilities to permit scalability by performing multi hoping [23]. All those devices are designed using embedded electronics and embedded software development with specific constraints an limitations. Power consumption must be optimize avoiding leakage of current from components or tracks. Code should be concise and efficient for optimization of resources with special attention to interrupts handling and communication ports.

Devices in a WSN digital electronics share a great deal of components and features due to their embedded and restrictive nature. Control, intelligence and resources managing is provided by 8-bit microprocessors or 16-bit when more complex processing is required (gateways, routers). Radio link communication is carried out using low power radio chips and cheap antennas (monopoles, dipoles, micro-strip). Other communication interfaces might be UART, SPI, I^{2}C or USB depending on the type of communication protocol between other subsystems or modules and according to specifications. Sensor boards are commonly built with low cost sensors, both digital or analogue technology. Analogue data capturing is performed from external or integrated ADC converters into the microprocessor. Some digital sensors may require communication to be established through SPI, UART or I^2C ports, usually available in micro-controllers. Another essential components are batteries that might be of different types: standard alkaline, litium-ion, rechargeable. The choice depends on the desired use and cost.

1.4.5. RFID security

WSN and RFID are two wireless technologies with a huge potential for developing application. They are being deployed almost in parallel but the integration of both in the same network is not attracting much attention from the research community [62]. The advantages of putting both technologies together are numerous and the number of applications that can be developed is unlimited [22].

A possible application of the RFID technology in a WSN is to provided a mechanism to securely spread sensitive information such as secret keys or identification numbers (e.g. unique MAC addresses) belonging to nodes. In this case, a tag with id information can be attached to a node that wants to get access to a network. RFID readers can read tags from nodes from a few centimetres to meters. In a secure environment the nodes should not spread their information far from them. It is only required to be read once by approaching the tag close to the RFID reader.

It is specially important in the case of Home-Event WSNs to provide certain type of security [22]. It is necessary to separate the user network from other networks in the vicinity and avoid a malicious user to send fake data to the gateway and provoke fake events and avoid eavesdropping from third part listeners or hacking. Those actions can lead to breaking the home security.

There is a lack of devices in the market integrating both technologies and getting the benefit of security on WSNs. One of the target of this Thesis is to prove that RFID and WSN technologies can coexist together in the same application getting advantages of each other by means of an affordable device.

Chapter 2

System definition and goals

2.1. First approach

This chapter describes the functionality of the sensor network and what is expected from the application. It also tries to identify some of the requirements for the network and the devices that are part of it.

This Thesis is a part of a bigger project originally conceived by Ericsson as mentioned in the previous introduction. The main target is the design and deployment of a wireless network for a home-event management system. The essential elements to carry this out are common to most wireless sensor networks: a gateway, sensor boards (nodes), an appropriate networking protocol ruling the communication and an application managing the data.

When the project was suggested by Ericsson, the wireless network was intended to communicate with nodes externally through an Ericsson proprietary gateway. Hence, the network would consist of several sensor nodes and a gateway in charge of starting and commissioning the network (the network topology was not decided yet). This gateway would act as a intermediary link to an Ericsson external gateway in charge of delivering messages to outside networks under a common understandable format or protocol. The requirements where not explicitly specified more than an overall concept so we got a great degree of freedom in the design.

2.2. Main goal

The main purpose of this Master Thesis is the design of a platform that allows the deployment of a low-cost Self-Managed Wireless Sensor Network provided with RFID-based security. The goal is to prove that such design is feasible, useful and affordable for the final consumer. In order to achieve this objective, this report goes through different stages accomplishing the tasks shown in section 2.3 below.

2.3. Procedure

It is important to define a strategy in order to get a better understanding and solve the proposed problem. Following a scheduled plan optimizes the work by dividing it in tasks and steps that can be solved orderly and in parallel when possible.

1. Definition of the objectives of the Thesis and requirements.
2. Research on existing networking protocols and documentation.
3. Exploring Wireless Sensor Network designs.
4. Selection and study the protocol: specification, API, code examples, evaluation boards.
5. Study of the hardware platforms: components, costs, energy consumption.
6. Selection of developing tools.
7. Hardware:
   a) Circuit, PCB and antenna design.
   b) Milling and soldering of parts.
   c) Testing modules, ports and peripherals.
8. Software:
   a) Driver development for communication ports and HW peripherals.
   b) Research on available stack implementations.
   c) Study of available Operating Systems for embedded devices.
   d) Porting and integration of code to own platform.
   e) Application design and development.
9. Testing of the system, results and conclusion.

2.4. Deployment of the Network

One of the most important concerns consists of how to start managing the network. This is done typically by a central *authority* or *coordinator* device, in this case the gateway. Some of the procedures for bootstrapping the network are usually specified by protocols, but in some cases they are not [21]. Therefore, those mechanisms should be designed, implemented or configured to comply for the specific tasks independently of the standard.

When a coordinator starts for the first time the network, there is a list of common tasks to deal with: scan the radio environment to detect busy channels and other networks, choose appropriate channel, select a network identifier and an own identifier or node address, start the encryption mechanisms, key sharing, communicate with nodes...

Different mechanisms may occur when nodes want to join the network. The aim is to find a way to first identify and register the node as a member of the network (neighbour). Then, according to the specific protocol, find a way to bind the node to the coordinator (or router), similarly to neighbour discovery from IP or binding mechanism from ZigBee. In some cases, nodes are aware of the existence of a network by scanning the environment expecting a *beacon* message. Beacons are periodic frames sent by coordinators to advertise about their presence and are used for synchronization purposes. In other occasions, coordinators and nodes need to be manually set up to a "joining mode" state by, for instance, pressing a button. Running in parallel, there should be a system for secure exchange of encryption keys if a high level of security is desired.

A discussion about different mechanisms is held in chapter 3 and 4. In the proposed design, sensor nodes join the network by means of RFID technology. A tag is attached to the nodes with some identification information about them or security information. An RFID reader installed in the gateway reads that information when the user get the new node closer to it. A security application should handle this data and authorize that node for future use in the network.

Protocol

Protocols stablish the rules that control the behaviour of a network from the initial state. In order to find the best match for the present design, a set of requirements should be carefully defined. In Chapter 4 there is a discussion aiming at finding the optimal choice.

Many different descriptions of networking protocols can be found in Chapter 3. It gives the reader an idea of which features and constrains WSN protocols have associated and is a good help for the later discussion.

The protocol should be able to provide a way of communication between the nodes not only internally but also externally with other networks. In the specific case of WSNs it is required that the protocol complies with a set of constraints as exposed in Chapter 1. The designer has to decide also the minimum requirements that the network should be able to offer and choose the most suitable protocol in accordance.

The type of network topology, data rate, number of nodes, complexity, stack code size, available existing implementations are typical features that should be considered.

Other attractive features to have in mind are, for instance, the possibility of designing hardware or developing code using free tools, consulting communities of developers, availability of open protocol stack implementations, adaptability of code to different hardware platforms (porting). Those things can greatly facilitate the work.

Characteristics and requirements

- Self-organized: The network should start working, authorise nodes and handle communication without interaction from the user other than the RFID detection. It should also be able to heal itself and restart after shut-down.

- Topology: Only a simpler star topology is considered to start with. It should be enough to prove the main goal and it will allow a simple and faster design of the gateway prototype compared to more complex networks.

- Scalability: The network protocol should allow for scalability and be able to upgrade to hybrid and mesh topologies in future implementations.

- Coverage range: The main scope of the WSN is indoors environment. No large distances are considered in this version, just typically indoor distances no longer that 10 m. By providing scalability it is expected that larger ranges can be achieved in the future.

- Size: In an average home environment the number of rooms rarely exceed 10 rooms, counting a living room, kitchen, 4 bedrooms, 2 bathrooms, basement or garage. Typically on sensor node will be needed per room, depending on the use of that space. But in other to be safe, we can set an average number of 20 nodes per network as a requirement. At least in the first version of the prototype.

- Data rates: Since it is an event-driven network, high constant data rates are not expected, but the network should be able to handle punctual burst of information if several events occurs a the same time or continuous requests are send to the nodes. We cannot provide accurate quantitative requirement at this point because it is highly dependant on the radio technology and protocol, but based on studies of typical WSNs today, data rates are found for different radios between maximum values from 20 to 250 kbps, depending on the radio frequency and the protocol. These are theoretical values that are reduced due to hardware constraints, packet header or interferences, the valuable throughput is usually the half of this values [25]. Typical WSNs cannot reach more than 120 Kbps of throughput (250 Kpbs data rate). It is more interesting then to calculate the throughput that the devices should be able to handle (see gateway requirements below).

- Wireless communication: The communication will be held by radio-frequency at a frequency that allows achieving the data rates and throughput mentioned in this chapter. A discussion about different radio technologies, frequencies and protocols can be found in chapters 3 and 4.

- Installation and maintenance: The installation of the network should be automatically done when the user connects the gateway to the computer. Additionally, more functionality can be provided for the user to interact with the gateway and nodes through the computer. However, the communication within the network should be self-managed by the protocol.

2.5. Application

The problem of developing a functional and useful application is bound to the application layer or API that is provided over the network protocol. The implementation of an application layer is also part of the work of the present document and it has to be agreed with the design of the sensor nodes [3].

From a functional point of view, in a typical scenario the user should be able to add new sensors to the network by approaching the node to the RFID reader installed in the network that will read specific information of that node. The security application should handle the authorization and secure the node link from that information obtained from the RFID (It can be an ID, MAC address or security KEY depending on the protocol). No more installation should be required from the user, the network protocol should manage the identification and communication with the node automatically.

From the USB interface it is expected some information printed on the computer side any time a node with a tag is detected. This will inform the user of the id and type of node that has appear on the network. Additionally, the user could use the USB interface to authorise the node by himself.

The user should have control of the nodes from the USB interface in order to determine what type of data is required, in what format, with what periodicity or give control to the node so the node itself determines when to send some information (event-driven after value passing a threshold). It should also be possible for the user to request status reports from the sensors at any time. All this information should be delivered to the computer through the USB interface.

Characteristics and requirements

- Data handling: Single requests and periodic requests should be possible to do by the user from the USB interface. Nodes should be able to independently send data to the gateway every time events occur. The user should have the possibility to configure such kind of behaviour on the node from the USB command line.

- Information data should be also reported from different sensors every a certain interval preferably minimum 1 second. The periodicity can be changed and sensors can send data independently when events occur.

- The application should be able to control and keep track of temperature, light intensity and humidity in the rooms. It should be possible to perform actions on the nodes as well by, for example, turning on/off a switch.

- The format of messages and data should be plain text or xml so it can be easily interpreted and represented. The average payload size should fit in the packet payload field to avoid more complex treatment such as fragmentation. A size of 50 bytes is considered enough to represent sensor data, periodic reports or events information and it shouldn't be exceeded. Depending on the selection of the protocol it might

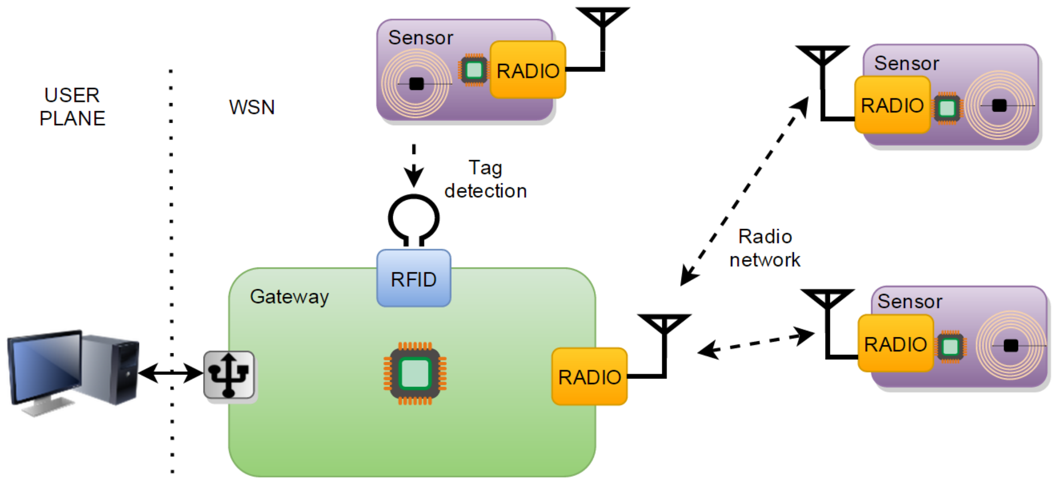


Figure 2.1: Wireless Sensor Network overview. It shows the different elements that form part of the WSN and their connectivity. The solid arrow means wired connection while dashed arrows define wireless link.

actually need to be reduced (e.g. in the 2.4GHz radio standard IEEE 802.15.4 the packet size is 128 bytes and the payload variable depending on the header)

- Security: should be provided by the application by means of authentication of nodes through the RFID application and encryption of packets using secure private keys.

2.6. Devices

It is fundamental to have a good description of the devices and how they interact together. Once the devices have been clearly described it will be possible to start selecting the components and designing the board and the software implementation. That makes possible to separate the work into different tasks and stages.

2.6.1. Gateway

The target of this Master Thesis is to prove that the proposed self-managed system is achievable at a low-cost with security provided by RFID technology. The main element that makes the WSN possible is the gateway. The aim is the design of a gateway prototype with no commercial purposes or strict performance requirements at first.

The gateway is the central point of contact between the WSN and the exterior environment. It concentrates the information obtained from different nodes via radio and then delivers the information outside to the user via a wired interface. The platform is equipped with two interfaces providing connectivity. On one side a radio transceiver is required to allow nodes and gateway to communicate wirelessly. But neither the frequency of operation nor the specific hardware for this task will be defined at this point. This and other hardware matters are studied in Chapter 5. On the other side, a wired communication link will connect the gateway externally to a computer by means of a USB port as required from the original project proposed by Ericsson.

The gateway should be built with different modules in order to fulfil the requirements of the application and the network: a low power radio, a USB interface and an RFID reader.

Other functions performed by the gateway involve: starting and commissioning the WSN, management of identifiers and/or addresses, delivery of encryption keys and registration of new nodes into the network.

Characteristics and requirements

- Power requirements: since the gateway is connected to a USB standard port on a computer, it can be directly powered from it in order to simplify the design. A USB port on a computer is typically limited to provided 500 mA at 5 V so the gateway should not consume more than that, this is a power limitation more than a requirement, but it certainly fulfils the need of power for the modules.
- Speed: It depends on the clock source. For typical low power micro-controllers it could be 8, 12, 16, 24 or 32 MHz, rarely more. All these clock frequencies might be enough to run the system. However, in order to avoid future problems, it is preferably to give some flexibility to the board by providing it with a faster clock source. Then it will be possible to test other frequencies by means of the internal divisors of the micro-controller in order to find the most optimal. The requirement at this point is a frequency between 16-32 MHz.
- Memory size: It depends on the running OS, the drivers implementation, the network stack and the complexity of the application. Commonly an OS targeting an WSN has a small code footprint in the order of tens of kB (for example Contiki requires 60 kBytes for the OS [42], extra size is incremented by the application and the implementation of drivers in approximately 30 kB). We can set a requirement of 100 kB for the flash memory size.
- Wireless communication: Apart from the maximum data rates mentioned for the radio link, we can estimate the throughput of the application giving some average general values. Considering that each node is transmitting 4 packets per second, one for each type of sensor another for other requests, if the requested payload with sensor information is estimated to occupy 50 bytes in text/xml format. The throughput can be calculated from the formula below:

  $$Throughput = 20\ nodes \times \frac{4\ packets}{second} \times \frac{50\ Bytes\ payload}{packet} + \frac{8\ bits}{Byte} = 32\ kbps$$

  In this situation, the gateway should handle a throughput of 32 kbps in this network. This is coherent with studies stating that typical throughput is in the range of 35 to 40 kbps for such networks [25].

- USB connectivity: A standard USB connector should be available in the gateway to connect to the computer. There is no requirement for the operating system to use as long as its drivers are compliant with the USB specifications, but in order to prove this, testing is required either on Windows or Linux systems. The gateway will act as a slave (or client) of the USB interface, the computer should recognise the device and act as a host.

- USB commands: A set of commands or message in a specific format should be defined and should be available on the USB port so the user can configure the nodes or send requests.

- USB requirements: High data transfers are not required in average, but the interface in the gateway should be able to handle burst of information at a certain time. Considering 20 nodes with maximum *throughput* of approximately 1.5 Kbps it should handle at least 30 Kbps. The USB standard specifies different speeds (from 1.5 Mbps to 480 Mbps) on different versions but it should be considered that the maximum achievable rates are usually much lower due to physical constraints and overhead.

- RFID tags: should be small to fit the size of a sensor board and cheap. Tags should be able to store an amount of data similar to a encryption key (e.g. 128 bits) or MAC address (e.g. 64 bits) or both. These values might vary depending on the protocol, it is better to have a high requirement than the sum of them, at least 256 bits (32 bytes).

- RFID reader: should be able to read at short distances near 10 cm. No need for much larger distances. There is no need for high data transfers since the reader is only used when new nodes are installed in the network (if one per second 256 bps). A simple protocol might be enough.

2.6.2. Sensor nodes

The study of the sensor boards is not within the scope of this work [3], neither the capture and interpretation of their data [4] [5]. This report focuses on providing the mechanism to access to the sensor data and to give them the possibility of sending useful information.

The communication protocol will be common for all the participants of the network but their role will vary. Physical and link layer parameters related to the wireless communication will be similar for all devices: band frequency, radio channel, modulation and protocol. However, hardware modules and circuitry might be different to find the best trade-off between cost and functionality. The boards should be as cheap, durable and simple as possible.

The application has to control the sensor boards and the data over the networking protocol. There are different stack implementations and embedded Operating Systems that offer specific, proprietary APIs or application layer protocols that can be very useful for the task.

Characteristics and requirements

- Power requirements: There is no need to specify in detail hardware requirements, but functional selection such as radio duty cycle length and sleeping modes might affect the throughput.
- Battery duration: No less than 1 year, optimal 2 years similar to other products in the market.
- Throughput: (see network requirements above) Considering only one radio link for a single node transmitting 3 packets/second on average (temperature, luminosity and humidity) on a 50 bytes payload, the minimum required throughput can be calculated from the formula:

$$Throughput = 1\ node \times \frac{3\ packets}{second} \times \frac{50\ Bytes\ payload}{packet} + \frac{8\ bits}{Byte} = 1,2\ kbps$$

- Application: An application should be capable of retrieving data from a temperature, a light intensity and a humidity sensor. There should be also able to modify a switch, a LED or similar component as a replace for an actuator. The application should be also able to respond to single data requests, send periodic reports, or send event-triggered responses.
- The sensor boards should be provided with and RFID tag and a MAC address or Secure key stored in it and in the flash memory of the micro-controller so it can be used to

The design of the sensor boards from hardware, firmware development and low level communication layer is the topic of another Master Thesis on progress at the time of writing [3].

Chapter 3

Background

3.1. Wireless Sensor Networks

Wireless Sensor Networks are typically designed to handle low data rates and being low-power consuming [24]. The devices are therefore resource constrained and limited in capabilities such as processing speed, memory space or data rate. They have to be very well designed to accomplish with the constrains, specially regarding the low power consumption since batteries are aimed to last for years. This kind of networks are thought to be working locally in small spaces usually transmitting low rates of data. Theses limitations must be taken into account when designing applications and deploying the network. An ideal WSN should be scalable, have very little power consumption, be smart and programmable, reliable, cheap and easy to install with just a little of maintenance needed [21].

Choosing the right hardware parts is fundamental to achieve the features above when designing a network. Once the cost of components has been reduced to the optimal, there is still left the concern about how resources and communication will be managed. Here is where communication protocols take part. The aim is to make nodes to communicate in the most efficient manner. Energy efficiency is the most important issue at this level, followed closely by network auto-configuration possibility, scalability, reliability and security.

3.1.1. Protocols

There are a large number of protocols in the market specifically destined for WSNs deployment. There are also numerous variables to considered in order to find out the most suitable protocol for a particular application. So to enumerate some features affecting not only the behaviour of the network, but also the hardware requirements:

- Band frequency
- Stack code size
- Data rate

- Network architecture
- Number of nodes
- Multi-hop or routing
- Node-to-node range

All the needs and purposes of a sensor network application should be studied carefully before selecting the protocol that will manage the network. That would help to find out the most optimal protocol that fits the requirements.

Some of the WSN protocols that are going to be exposed here rely on the **IEEE 802.15.4** [32] radio standard that specifies the Physical and MAC layers for low-range wireless personal area networks (WPAN) being a good candidate for building WSNs. IEEE 802.15.4 offers different mechanism that make networks very robust. It can operate at 868 MHz, 915 MHz and 2.45 GHz ISM bands specifying different data rates that can be also chosen according to them (20 kbps, 40 kbps, optionally 100 kbps and a maximum of 250 kbps). More bands are contemplated in the standard later revisions (years 2007 and 2009). As it is stated in the last specification from 2006, the last standard is backward-compatible to the 2003 edition.

At the PHY level IEEE 802.15.4 different techniques and modulations are specified: direct/parallel sequence spread spectrum (DSSS or PSSS respectively) techniques and different modulations (BPSK, O-QPSK or BPSK+ASK) in several combinations (refer to the specification for more info [32]). New transmission techniques and modulations has been added to the latest specifications in 2007 and 2009 revisions.

Different numbers of radio channels are available in the frequency bands: 3 channel in the 868 MHz band, 30 channels in the 915 MHz band and 16 in the 2.45 GHz band. This is an excellent feature to avoid interference in highly saturated channels. Networks might dynamically be able to change communication to a less busy or interference-free channel. Network radios can do the task by scanning the spectrum and registering those channels where lowest activity is detected.

Many of the transceivers found today in the market today offer hardware pre-built functionalities compliant with the IEEE 802.15.4 specification. A transceiver will modulate/demodulate the signal performing additional amplification, filtering and some kind of processing. A good IEEE 802.15.4 compliant transceiver can also service the MAC layer or upper layers with some extra functionality according to the specification requirements.

To achieve a higher degree of reliability, IEEE 802.15.4 includes frame checksums (FCS) that use a 16-bits Cyclic Redundancy Check (CRC) and are appended at the end of packets.

At the MAC level the standard uses Carrier Sense Multiple Access Collision Avoidance (CSMA-CA) another mechanism to avoid interference increasing reliability. When the most optimal channel with less interference is found, the protocol looks starts the transmission when the channel appears to be clean.

MAC layer can provide beaconless or beacon-enable mode. Beacons are framed used by the network to advertise its presence to other networks and devices or for synchronisation (see specification [32]). Any mode can be chosen but complexity grows in beacon-enabled network where superframe structures and beacon frames are used and time slots needs to be managed carefully. Beaconless networks use simple CSMA channel access.

There are four frame structures defined in the standard each of them with an specific purpose and different header fields and sizes. If the network is beacon-enabled, *beacon frames* are transmitted periodically acting as beacons from the coordinator of the network; *data frames* transmit usable data payloads; *acknowledgement frames* confirm reception of packets to the sender and finally, MAC *command frame*s handle MAC peer entity control transfer.

Most used frames are typically the data and the acknowledgement frames when beacons are disabled. That is normally the common situation due to the complexity that beacon-enabled networks introduce. There is no need to detail all frame types but it can be interesting and helpful to understand the network upper layers to show the structure of a data frame.

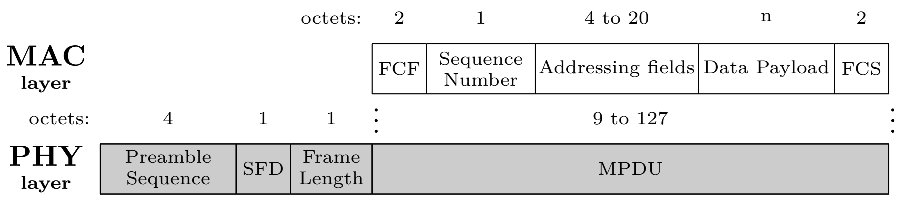


Figure 3.1: IEEE 802.15.4 data frame

Figure 3.1.1 represents the physical (PHY) and media access control (MAC) headers of a regular packet. The PHY layer is used for detection of the frame and synchronization and it is typically detected, parsed and processed by hardware.

The MAC layer header is a sub layer of the OSI link layer and is allocated after the PHY header. According to the IEEE 802.25.4 specification it consists of three different fields: FCF stands for Frame Control Field and it is a 2 bytes long field. Bit values here give MAC control information related to compression, acknowledgement requests, security, frame type, addressing mode or standard version. The following Sequence Number in needed to differentiate consecutive packets. In the Addressing field it is possible to find the PAN identifier of the network and/or the Source and Destination MAC addresses. Those addresses can be 16-bit or 64-bit long and may appear or not depending on the compression mode.

According to the IEEE 802.15.4 specification, we can classify the devices as FFD (Full Function Device) or RFD (Reduced Function Device) depending on the amount of tasks that they can carry out. Gateways and routers are usually FFDs, while sensor boards are RFDs.

IEEE 802.15.4 standard is able to manage addresses, routes and build star or mesh networks (see Chapter 1) by its own, providing multi-hoping capability and use of acknowledgements. Many other functionalities are exposed in the specification document [32].

Zigbee [7] [27] is probably the most known and extended protocol for Wireless Sensor Networks. It is an open standard created by the ZigBee Alliance, formed by a group of companies in the electronic field (Philips, Analog Devices, Texas Instruments, Motorola, Freescale and many more). It is one of the reasons why ZigBee has showed that spectacular growth in the last years. ZigBee is built over the IEEE 802.15.4 specification taking advantage of the features that it offers as those explained above. It is important to remark that some features from different standard editions might not be compatible with other revisions.

There are several properties that characterize ZigBee. It is low cost: since many companies are involved in its standardization they produce cheap products off-the-shelf for easy development. ZigBee is very reliable: it is built over IEEE 802.15.4 standard the offers many capabilities to improve reliability. In addition, ZigBee provides mesh networking, multi-hop, automatic routing and moreover encryption and authentication mechanisms for security. ZigBee is also a low power consuming protocol: It manages energy consumption by making nodes to go to sleep mode during some time or avoiding unnecessary transmissions of non relevant data.

The ZigBee specification describes a networking protocol (NWK), however it also provides a powerful and multipurpose application layer by means of *application profiles* that define and describe devices of a network. Then the devices decide on a common profile to interact with each other. Those profiles define the functionalities and actions between ZigBee Device Objects (ZDO). Applications Profiles are described in additional specifications to extend the standard. That lets devices from different manufacturers to be able to communicate using the common profile. Manufacturers usually provide their own ZigBee stack and proprietary APIs to manage easily the functionalities that ZigBee offer. In a ZigBee network three kind of devices may coexist: A Coordinator, Routers and End Devices.

SimpliciTI [10] is a very simple and proprietary network protocol from Texas Instruments aimed at low power and low cost wireless networks. It does not rely on IEEE 802.15.4 but a simpler and minimal RF interface (MRFI). More information and the code is available at the website of Texas Instruments [10]. They claim that it uses $<$ 8Kb of flash memory and 1Kb of RAM. The simple provided API let users to create star, peer-to-peer and start-extended networks (see [10]) and applications with only 5 commands. Devices may be of three types: Access Points, Range Extenders or End Devices. This type of protocol builds networks focus on home automation applications, alarm and security, detectors, sensors, meters or remote control. This is a very basic protocol for small networks and little reliability.

MiWi [11] is another simple WSN Protocol developed by Microchip. It is based on IEEE 802.15.4 standard. Features are not as advanced as other more complex protocol such as ZigBee, but it can work with a very small code size having a favourable impact on energy consumption and cost. Application Note from Microchip [11] claims that

MiWi protocol can tolerate up to 1024 nodes in the network. MiWi is able to create star, mesh and cluster-tree networks.

The star network has a unique coordinator that configures and manages the network. End devices can communicate to each other through the coordinator. A cluster-tree network is similar to a hybrid networks having more than one coordinator/router in the network. Devices are linked to only one coordinator them and only one of the coordinators can configure the network. In a mesh network devices and coordinators can communicate directly with each other with no restriction. One coordinator is limited to a maximum number of 127 children, and there must be a maximum of 8 coordinators at the same time operating in network. Packets can travel up to 4 hops within the network or 2 hops from the main coordinator. MiWi offers the possibility of using beacons and acknowledgements.

6LoWPAN [31] [34] is a set of standards still under development by the IETF 6LoWPAN working group. It has been created to enable IPv6 packets to run efficiently on low power wireless networks and embedded devices. It was originally conceived to be built over IEEE 802.15.4 standard but later generalized for any other similar link layer standard for constrained networks.

6LoWPAN can be seen as an intermediary adapting layer between the link and IPv6 layers. It is mainly designed to compress huge IPv6 headers (40 octets) and addresses (128 bites) into smaller and more efficient ones for low power wireless networks (best case down to 4 bytes). In addition, the 6LoWPAN working group is making good efforts to make specific 6LoWPAN mechanisms such as Neighbour Discovery where IPv6 ND results to be too inefficient and consumes too much power.

Best advantage of 6LoWPAN is that it actually makes use of an international standard protocol IP instead of being another networking protocol with different mechanism and addressing. It helps gateways to be much more straightforward avoiding difficult translations of protocols from the Internet to the WSN. The applications might be built on any other protocol from the IP suite such as UDP, TCP or management with ICMP or SNMP. The fact that 6LoWPAN is being developed by the IETF will lead it to become an open international standard making integration of low data rate wireless networks to be seamless with existing Internet network.

Despite 6LoWPAN development and standardization is not finished by the time this Thesis is being written, there are several open implementations and IPv6 stacks available such as those integrated into Contiki OS for embedded devices by the SICS (Swedish Instute of Computer Science) [39] or in Tiny OS [12] supported by an open community.

OneNet [8] is an standard protocol designed for low power and low cost WSNs and developed by an Open-source community. At the physical layer side, One-Net works at 868 MHz and 915 MHz bands mainly with 25 channel available, but other frequency bands may be optional too. It uses FSK modulation instead of FHSS or DSSS. Base data rate may vary from 38.4 Kbps up to 230 Kbps per node dynamic data rate configuration as they state. One-Net is able to build Star, Peer-to-Peer and Mesh in a range over 500 m outdoors and from 60 to 100 m indoors, extendible to kilometres when using mesh networking. It provides advanced security functions adding XTEA2 as the default encryption scheme. API is claimed to be helpful and easy to use. Porting for different platforms, schematics, reference designs and code is available at the official website [8]. Moreover, One-Net is supported by several companies including Texas Instruments Analog Devices, Freescale, Renesas or Micrel.

The fact that One-Net is an open and free alternative is very interesting since no proprietary tools are need for development. It drives the cost of developing to a minimum.

Dash7 [13] is another low-power networking protocol that differs from the protocols above for being based on ISO 18000-7, an open standard for low power RF that operates in the 433 MHz band. This band is an ISM available for free use at Region 1 (Europe, Africa, Middle East and part of Asia) and restricted to low ranges, dedicated to amateur radio and requires permission from the FCC amateur regulations [82] in the USA. It is much less saturated than the 2.4 GHz band. There are numerous advantages when working at 433 MHz band, less power is needed, wider ranges can be covered and it offers deeper penetration into materials. However,the bandwidth is more limited. Therefore, DASH7 is attractive for those applications where high-medium data rates are not required.

DASH7 is developed by an Alliance and it is also supported by several companies offering evaluation boards or compliant hardware products such as ST Microelectronics or Analog Devices.

In order to not to extend unnecessarily this section it has been preferred to mention other protocols instead of describing them with detail. The most popular ones have been already showed, but a few more interesting protocols can be found such as: Z-wave [15], Synkro-RF [16], Insteon [17], WirelessHART [18], PopNet [19], ULP Bluetooth [20], and so on.

In Table 3.1 [1] there is a comparison of several features between various protocols. All the information showed will be very useful at the time of deciding the most adequate protocol considering the requirements of the network. The reader must be aware that some of the data presented could be dependent of the hardware and stack implementation itself, but that variation should not be very noticeable. Code size and covering range are the most sensitive parameters to that dependency

3.1.2. Bootstrapping and Security

A wireless sensors network is very likely to be sharing radio space with other different networks in a building. There is a need of isolation between those networks. The nodes

[1] Source: several different resources [7] [27] [31] [34] [10] [11] [8] [13]

	Zigbee	**SimpliciTI**	**MiWI**	**6LoWPAN**	**OneNet**	**Dash7**
Standard based	Yes (802.15.4)	No	Yes (802.15.4)	Yes (802.15.4)	No	Yes (ISO 18000-7)
Frequency band	868-915, 2400 MHz	868-915, 2400 MHz	2.4 GHz	868-915, 2400 MHz	868-915 MHz	433 MHz
Channels	16	-	16	16	25 (USA)	up to 5
Data Rate	20 to 250 Kbps	100 Kbps	250 Kbps	20 to 250 Kbps	38.4 - 230 Kbps	28kbps to 200kbps
Max range	75 m	-	-	75 m	60 - 100 m	250 m
Nodes	2 to 100s	2 to 30	Max 1024	Up to 2^{64}	Max 4096	-
Network types	Star, P2P, Mesh	Star, Extended, P2P	Star, P2P	Star, P2P, Mesh	Star, P2P, Mesh	-
Code Size	32 Kb to 100 Kb	< 8Kb	3K-17K	22 Kb	16 Kb	-
RAM	8 Kb	1 Kb	-	4 Kb	-	-

Table 3.1: Wireless Sensor Network protocols

must be aware to which network they belong to in a similar way as coordinators need to know what sensors are associated to it.

Before starting communication within the network, the nodes need to know some information to successfully join to it. The collection of mechanisms that allow that integration of the new node is known as Bootstrapping.

In the bootstrapping stage there are a few common procedures usually performed. As an example, scanning of the radio space by the coordinator or by nodes to locate coordinators, association mechanisms, sharing of secure keys for encryption and assignment of network addresses are some of those procedures.

Some of the mechanisms are defined by the protocol specifications over the network layers, but for a first recognition and identification of nodes there is not a fixed or standard method. Some guiding and advice are generally given by authors but finally the mechanism employed will depend on the designer of the network, the implementation or the needs of the network. Also, key exchanging is a delicate matter that requires a good engineering implementation to keep security under the necessary margins when new nodes access to the network.

Users interfaces might be very useful for recognition and authorization of new nodes. Sensitive information such as key material or identification numbers can be easily exchanged using keyboards, pads and displays. Unfortunately, this is not the case typically useful for low cost embedded devices. The way that users interact with constrained devices is typically be means of buttons and LEDs. So, it is preferred an automatic, self-managed and straightforward process to perform bootstrapping tasks.

In the last few years a new term has been coined known as Auto-ID that refers to any automatic identification method for items, products, animals or people. There are different mechanisms that can be suitable for the purposes of the network that this thesis is focused on.

Some procedures have appeared in the market today as shown below [29]:

- Barcodes
- Smart cards
- Magnetic cards
- RFID

All the mechanisms above might be adequate for identifying new nodes appearing in the network, hence it would be interesting to briefly describe them in order to find a proper choice for this project. Later it will be possible to investigate more in deep, finding new features, properties and ways of implementation in the system.

Barcodes are composed by painted black bars and white spaces in parallel codifying a symbol. It is commonly read by capturing the reflection from a laser beam (or from a camera e.g.), then processed and interpreted as a number or alphanumeric character. As a disadvantage, a barcode cannot keep more information than a codified value, hence it is very limited. In addition, this technology needs specific advanced hardware to read the barcode, making the design more complex and expensive.

Smart cards can be considered like memory cards or microprocessor cards. These cards increase hugely the capacity to store information or security, and their applications go usually one step beyond than just allowing identification of an item. Reader system is also much more complex than the needs of a wireless sensor network. Smart cards can be found in GSM systems, photographic cameras or music players [29].

Magnetic cards, as those used as bank cards, could be also a valid option. However, despite cards are reasonably cheap, the reader is big in size and it would add an extra complexity and costs to the gateway compared to other technologies. Moreover, adding a magnetic stripe to each node would be an inconvenient to the aesthetic design of sensor boards.

RFID stands for Radio Frequency Identification. Each item to be identified has a tag or label attached where information is stored in a chip and powered by induced or radiated energy obtained from a reader in the environment. Tags are becoming cheaper every day as this technology is gaining in importance as an expanding business. Readers are also simple enough and cheap. They consist of an antenna or coil, a matching circuit and a chip that acts as a data frame reader. RFID tags are able to store between 16 to 64k typically [29] and they can be rewritable. This feature extends enormously the possibilities of the systems, since rewriting allows tracking of items by having a report of the conditions suffered during a supply chain. It is even possible to attach sensors directly to tags. Communication is usually set wirelessly in the 13.56 MHz band (not exclusively, but most common band for RFID), what is seen as an advantage since it provides more versatility to the design of nodes, not carrying about alignment of objects.

There is another very similar Auto-ID technology called *NFC* (Near Field Communication) [30] that can be seen as an extension of RFID. Actually, it is based on the RFID ISO/IEC14443 standard [30] but improving its features by increasing data rate, security, using more effective codification schemes, and adding the possibility of letting two devices to communicate to each other transmitting and receiving data indistinctly of reader or tag concepts.

Auto ID choice

Because of the properties that has been extracted from each auto identification procedure, and considering that the WSN under development in this project needs to be simple, cheap and standardized the best option seems to be RFID.

Barcodes are simple and there is barely any cost associated when attaching a code to devices but the complexity and price of the scanner makes it a very bad choice as it does not accomplish with the requirements of simplicity and low cost that the gateway needs.

Cards suffer from the same inconvenience of complexity and high cost of the reader despite of cards themselves increase information storage and allow manipulation of data.

On the contrary, building an RFID system would be relatively easy since RFID modules and transceivers are easy to find and they are usually well documented also with code examples available. Most tags types allow rewriting of their internal user memory space, what increases functionality and additionally, tags are also getting low-priced and RFID is becoming widespread at everyday life.

NFC is still not broadly extended and the extra capabilities would provide unnecessary complexity to the implementation. RFID features fix perfectly to the requisites of the system under study.

3.2. Radio Frequency Identification (RFID)

3.2.1. Introduction

There is no intention of explaining in detail the mechanisms that makes RFID technology possible. It is important, however, to approach the reader to the principles underneath this technology before describing the implementation for the system.

RFID is a RF technology that is aimed at identification and tracking of things as the main purpose. In order to operate, there are two essential elements: an interrogator, more extensively known as reader and on the other side, a transponder, typically called *tag* or label.

Tag It includes a microchip, a simple electronic circuit and an antenna or coil that acts as an antenna. Sometimes it is also possible to find a battery taking part of a tag. It is usually used as label and is stuck to the products to track.

Reader It is an electronic device consisting of a micro-controller and a transceiver that takes the action of sending and reading information by means of electromagnetic waves, and an analog front-end that comprises an antenna and an RF matching circuit.

The reader initializes the communication by sending electromagnetic pulses via radio at a certain frequency at some time intervals. This can be seen as an act of interrogation to the proximity field. By the time a tag is crossing the near environment it immediately gets powered and replies with some data. This is possible by means of electromagnetic inductive coupling on low-medium frequencies below UHF or radiative coupling at UHF. The data is captured by the reader in the form of a frame of bits carrying an identification number. The tag takes his energy from the field radiated by the reader. The physical mechanism that generally lies behind is called energy coupling.

Concerning tags, a classification can be established depending on the sort of powering mechanism used by the chip.

- Passive tag: It uses the energy radiated by the reader using inductive coupling to power up itself and transmit data back to the reader.
- Semipassive tag: The internal chip is powered by a backup battery usually working at UHF band with radiative coupling.
- Active tag: Are battery powered but additionally they include their own transmitter and do not take advantage of energy coupling to use energy from the reader. Usually also working at UHF band.

In a typical scenario where long detection distances of meters are not crucial, passive tags are much more common. Avoiding the use of batteries gives the advantage of notably reducing costs.

3.2.2. RFID Uses

Tags hold information in an internal memory that will be scanned by the reader afterwards. The internal memory usually contains an identification number that should ideally be unique for that tag. This accomplishes the first main objective of RFID providing identification capability.

Moreover, a tag could also store other useful information such as the place and time of production, other places where it has been passed through, expiry dates, etcetera. This is made possible due to the potential of tags for being rewritable several times. This property allows RFID systems to keep track of as many items as wanted.

RFID has been under development for some decades and it is widely present in many places and different industries but it is not broadly deployed for home automation purposes.

The number of future applications of this technology is immeasurable, as huge as one's imagination can reach. However, cost and security issues might still slow down the growth rate and introduction into the society.

In order to increase the presence of RFID technology worldwide there are a few points that need to be reinforced. Higher investments are crucial to achieve that:

- Reduce number of RFID standards and anti-collision protocols avoiding proprietary solutions and incompatibilities among devices.

- Reducing tag cost to a minimum to make them affordable and profitable.
- Optimal integration of RFID systems into present and future data networks (heading to the Internet of Things).

RFID is used for common applications such as: Items tracking, identification, inventories of products, information management (weather conditions, positioning control), security (users authentication, access control, validation of devices information obtained from tags).

3.2.3. Energy coupling fundamentals

There are different ways for a tag to be supplied of energy [29]:

- Inductive coupling
- Radiative coupling
- Close coupling
- Electrical coupling

From those above, inductive coupling and radiative coupling are the most extensively used, at bands from LF to HF and up to UHF respectively.

Whether the frequency used is high or low, the type of energy coupling can be inductive or radiative. It is actually more appropriate to talk about wavelengths related to antenna sizes instead of absolute frequencies as it is explained below.

Inductive coupling

When an antenna is radiating or receiving a signal with a wavelength much larger than the size of the antenna, there are barely detectable differences of voltage along the antenna. The signal is transmitted by the variation of the magnetic field in the surrounding area. The magnetic variation can be detected with another antenna by means of a physical phenomenon known as inductive coupling. It is more appropriate then to work with coils as antennas instead of electrical antennas.

Coils are the most suitable component to store and couple magnetic energy. The size, number of cable turns or shape of the coil will determine the inductance and limits of detection of the generated magnetic field. The magnetic energy is concentrated around the proximity of the coil to decay drastically as we move away and proportionally to the cube of the distance. It makes the range of detection to be shorter at LF band but gives the possibility of building smaller tags by increasing the number of turns.

It also should be pointed out that the lines of the magnetic field are distributed in a determined way around a coil. Coupling will be optimal when lines from one coil cross the inner area of another coil. It can be inferred that a good inductive coupling is associated with the orientation of coils respect to each other.

Inductive tags are made with a coil, a resonance circuit connecting the coil and a capacitor, a signal rectifier circuit and chip that will provide modulated data. On the other side, the reader constantly transmits a signal at a certain frequency through another coil acting as an antenna. When a tag gets close to a reader, if the resonance circuit is tuned to the same frequency of the signal, it will generate a maximum voltage due to resonance at the ends of the circuit. This signal can be then rectified by a diode or diode bridge and a capacitor. An approximate constant voltage is achieved that supplies energy to the integrated circuit. It will generate a low frequency clock signal internally or dividing the signal captured from the resonance circuit. This signal will be used as a subcarrier, it will be controlled by a data output from the chip and it will feed a switch at the output of the transponder. The reader and tag coils are coupled magnetically when they are close together in the near field (no more than 0.16λ [29]) because energy is mostly concentrated in a form of magnetic field there. They form an inductively coupled system that works as a transformer with a primary and secondary coil. When coupling occurs, the modulated data going out from the switch in the tag makes the impedance of the virtual transformer to change accordingly.

This has been called load modulation and it is another way to transmit data e.g. by changes in the impedance. The reader detects the variation and filters the modulation product created by the load modulation at fc+fs or fc-fs. It subsequently amplifies the signal to proceed to demodulation.

Radiative coupling

Backscattering can be understood in RFID as a reflection of electromagnetic waves. Passive tags are fed with the electromagnetic energy obtained from the transmission performed by a reader. The energy is provided by a radiative coupling between the reader and the tag.

Radiative coupling gains importance when the wavelength of a signal is similar to the size of an antenna. It is possible to achieve longer tag detection distances since power decreases inversely to the squared distance instead of the cube. The method applied for the antenna design is based now on electromagnetic antenna theory for antennas with similar size as the wavelength is use. The size of the antenna will be dependent on the frequency of operation, since the higher it is, the smaller the tag can be. It happens at the microwave frequencies range.

In brief, both, reader and tags antennas must be tuned to the same transmission frequency in order to achieve an effective coupling. Because of the principle of reciprocity, an antenna has identical propagation properties in transmission as in reception of waves. If an antenna is efficient transmitting at a certain frequency, it will keep being effective when receiving at the same frequency. When a voltage feeds an antenna, it radiates energy in a way of electromagnetic waves. Receiving a wave with similar properties will generate the same voltage at the output. This voltage can be used to power up the integrated circuit in the tag. At long distances of several meters, a backup battery is normally used to supply energy.

The reader is not instantly coupled to the tag, there is a delay in the exchange of information, so load modulation is not possible. In this case, systems using radiative coupling commonly have both, a transmitting and a receiving antenna, or one antenna and a directional coupler in order to separate RX from TX signal when processing them.

It is possible to get longer distances than in the previous case since radiative energy in the far field decays much slower than in the near field where inductive coupling takes part. Whether longer distances are desired, a common solution consists of adding batteries to tags, since beyond meters, energy obtained from backscattering becomes not enough to power tags up. Nevertheless, despite of being able to achieve larger detection areas using radiative coupling, there is a disadvantage associated. Due to reflections, irregular propagation patterns may appear since constructive and destructive wave interference occurs. This means that the signal may disappear and appear again as we move away from the reader.

3.2.4. Frequencies

RFID operates within three primary frequency bands from low-frequency (LF) to ultra-high frequency (UHF). More precisely, at LF it is the 125/134 kHz band, at high-frequency (HF), the 13.56 MHz band as the most used band or the 5-7 MHz band. Meanwhile, UHF RFID systems usually operate at 860-960 MHz band or the microwave region at 2.4 GHz. However, it is also possible to find RFID systems working at 433 MHz or from 5.2-5.8 GHz frequencies [28].

In consonance with the frequency band, here is an optimal energy coupling mechanism associated as mentioned before. Specifically, at low and high-frequency bands, inductive coupling is normally used in RFID systems. In case we consider a simple dipole as an antenna, effective electromagnetic coupling at these bands would occur whether the length of the dipole is equal to $\lambda/2$. Being that the frequency of the system is 13.56 MHz, it means that if $\lambda = c/f$, wavelength is 22 meters, so the length should be around 11 meters. That would make tags to be absurdly and disproportionately huge. Instead, coils inductively coupled are used. To achieve a better coupling at a certain frequency, a resonant circuit is normally used. A proper coil design tries to find its inductance and capacitance associated to bring the circuit to resonance at the same frequency for reader and tag.

Up in the radio spectrum, the wavelength starts to be small compared to the dimension of a reasonable antenna for a tag (in a scale of centimetres). For the example of the dipole and frequency of 900 MHz we need a length of 16 cm, in case of the 2.4 GHz band it would be of 6.25 cm. Thus, it is assumed that at UHF band, electromagnetic coupling can be achieved in a more effective manner since smaller tags can be produced.

It is also worth to mention that transmission will be less or more robust regarding the frequency of the system. Electromagnetic waves are able to go through many different conducting materials up to certain distances dependant on the frequency band of operation. It all has to do with a physical parameter known as skin depth. According to it, the depth that an electromagnetic wave is able to penetrate into a material is inversely proportional to the conductivity of the material and to the frequency of the wave. So, the higher the frequency, the smaller the depth waves go into a material. That makes materials to behave like shields blocking the signal.

Government frequency bands regulations

When designing RFID systems we must be aware that radio signals might be present in many different frequency bands so interference might occur over other regulated bands and must be avoided. RF power emissions must not exceed certain values and should be limited within the boundaries of the ISM (Industrial, Scientific and Medical) frequency ranges. ISM ranges are open for any radio service but only under certain regulated limits of the signal strength. For example, it is 60dBμA/m at 10 m from the reader in the 13.56 MHz band $\pm$7kHz bandwidth. Despite there is a close worldwide agreement, at some frequencies regulations can be different depending on the region (America, Europe and Africa, and Asia) or even between countries. For instance, at the 900 MHz ISM band Europe and the US have different regulations. In Europe, the 868MHz band has more transmission limitations than in the US. Exactly the opposite situation occurs at the 915MHz band that is more restricted in the US than in Europe. The 2.4GHz ISM band is worldwide regulated under the same conditions and it might mean a problem due to the saturation at this band due to the different services .

Another critical point to be contemplated is that the frequency band that has been chosen for an RFID system will also determine the coding and data modulation and the communication protocol of the system.

3.2.5. Modulation and Coding

In every digital communication system, information has to be transported according to certain rules. Modulation is a process of encapsulating some kind of information into an analog periodic signal (subcarrier) by means of changing a physical parameter. That could be a variation in its amplitude, frequency or phase in relation to a modulated signal. In digital modulation there are three main mechanisms derived from those parameters. They are known as ASK (Amplitude Shift Keying), FSK (Frequency Shift Keying), and PSK (Phase Shift Keying). All those modulation schemes can be used in RFID systems and are typical at high and ultra-high frequencies. More specifically, when working with inductive coupled RFID systems, the load modulation behaves the same way as the ASK modulation. The simplest ASK application, very extended in RFID systems, is called On-Off keying (OOK), meaning that the subcarrier appears or disappears representing 1's or 0's respectively.

Coding is related to how information units (bits) are represented and should not be confused with modulation. The purpose of coding is to get a proper signal adapted to the transmission channel, error correction, data compression or cryptography.

In RFID systems several codes can be used such as NRZ, Manchester or Miller. A classic one is Pulse-Interval Encoding (PIE). The mechanism bases on representing logic 0's and

1's by a transition from signal off to signal on where the pulse active is longer when coding 1's. That avoids that long chains of 0's leave the tag unpowered.

3.2.6. Protocols

Protocols are used to provide certain rules for communication when exchanging information, like a language, that both parts know. A protocol in RFID has to be simple, since tags do not carry too much intelligence and it helps to power save. There are different standard protocols depending on the frequency range or type of tag. Most of them have been created by several standardization organisms such as EPC global (exclusive for RFID), ISO, IETF or IEEE. In addition, there are other proprietary standards developed by companies to make their products to communicate. All of them cover different modulations, coding or are designed for different detection ranges and purposes.

Standard protocols specified by ISO 14443A and 14443B work in the 13.56MHz frequency, and both are incompatible between each other. The ISO 15693 protocol is very popular but older than the others. Texas Instruments and Philips are two of the most interested companies developing RFID technology. As an example, Texas created Tag-it, a proprietary but simple and easy to deploy protocol based in ISO 15693 [58]. From NXP Semiconductors (Philip's spin-off) we can find MIFARE.

Other protocols that can be found are: ISO 11784/6, ISO 18000-2 at 125/134 kHz band; ISO 14443A,B, ISO 15693, ISO 18000-3 at 13.56MHz and ISO 18000-6A,B,C, EPC class 0 and class 1 at 800-900 MHz band or ISO18000-4 at 2.45 GHz band.

These protocols also support anti-collision mechanisms that help to differentiate from various tags detected at the same time in the same vicinity region. This and other issues concerning protocols will be detailed more extensively in Chapter 4.

Chapter 4

Wireless system design

As it has been stated previously, to build a WSN and fulfil with the requirements exposed in Chapter 2, it is fundamental to find an suitable protocol, an optimal hardware platform and finally develop the application that exchange data within and outside the network. This Chapter goes in more depth to solve those questions.

4.1. Protocol features

A great deal of protocols for constrained networks have been studied in previous sections (see Chapter 3). In order to find an optimal decision it must be clearly defined the purpose and functionality of the network and what constraints the system can afford concerning data rate, memory of devices, battery life, processing capability and cost.

It is an advantage to have a widely extended standard protocol. This will make things easier when finding possible hardware solutions such as radios and micro-controllers specifically designed for compatibility with that protocol. Additionally, it can be helpful for the software development since in most occasions an open source implementation of the stack is available. In addition, it is likely to get support from a software community and have access to free developing tool.

Frequencies of operation within the ISM bands can be freely used all over the world. That would make the product to be worldwide useful. No modification would be needed depending on countries or continents. Regarding frequencies, the higher it is, the more data rate is possible even though coverage distance and energy consumption is worse compared to low frequencies. In the WSN under study distances are usually short so those disadvantages does not affect so much to the design. In any case, if a larger coverage is required in the future, the use of hybrid and mesh topologies will help to extend the network and reach further areas (by adding of course some extra complexity to the system).

In order to be as much energy efficient as possible protocols can put devices into sleep mode, waking up at certain moments, sending information only when relevant events occurs and so on.

Complexity is a feature that usually affects the power consumption of devices since more processing is required. It will be therefore optimal to find a simple and straightforward protocol solution rather than a complex one. Also development time will be affected by complexity unless a proper API is available.

Recalling Table 3.1 from Chapter 3 there is an illustrative comparison of several features between some of the most popular wireless networking protocols.

The WSN that is under development in this Thesis is targeting a home-event management system. The applications will run in a house environment. As a result, the network should be flexible to adapt to many different distributions. Houses can have a large number of rooms, far or close one to another. One, two or more floors are possible, a basement, a cellar or a garden. Star topology might not be sufficient in that cases so mesh networks are preferred despite the increment in complexity and code size.

The number of nodes that the network is able to load might be a valuable feature. Big houses might be filled with tens or even hundreds of sensors all over it. Some nodes (extension routers) will have to deal with higher data rates that the protocol has to support.

Finally, the user should be kept away from network management and maintenance tasks. The protocol should be as simple as possible for the end user, even when adding or extracting nodes to/from the network. Theses mechanisms should be transparent to the user and the network should automatically adapt to new circumstances. This is a task not only by the protocol but also the designer.

To sum up, the desired features of the protocol that match the needs of the present WSN will include:

- Open and standard protocol
- Modifiable coverage by means of mesh network topology
- Low complexity, simple API and possibility of support
- Adaptability to environment
- Easy maintenance
- Security

4.2. WSN protocol selection

The discussion brings a first selection of three candidates fitting the requirements: Zigbee, 6LoWPAN and OneNet.

4.2.1. OneNet

In the beginning of the present Thesis, OneNet was strongly considered since it has an open source implementation available incorporating an API. OneNet's specification for design was distributed royalty-free.

There are two important points regarding what has just been stated. First of all, having porting capability to different chips is crucial since the study of hardware will be done after the selection. By the time OneNet was being considered, March 2010, the implementation of the stack had only been built for an evaluation board using a Renesas R8C/23 micro-controller and an ADF7025 transceiver from Analog Devices working at 915MHz. However, they offered a short guide for porting to other transceivers and micro-controllers.

Besides, the frequency of operation for the evaluation board was 915 MHz that is and ISM frequency in the United Stated but it has more restrictions of use in Europe where the closer ISM frequency is found at 868 MHz. That would make this system to be useful in EEUU but not in other countries. Fortunately, OneNet was designed to operate at 868 MHz band as well but there are no existing designs with transceivers at that band.

A major problem that can be attributed to OneNet is the lack of standardization. That is, this protocol is not worldwide deployed, actually it is under constant development due to bug's fixing. It is not openly supported by vendors so it can affect to the degree of compatibility between different implementations. The community of developers is not big either and it can affect to the fact of getting immediate support when it is required.

By April 2010 a company located in France named Sen.se got in contact with the team in charge of the project that this Thesis is part of. They offered a collaboration to the KTH team since they were enrolled in a similar project, working in the deployment of a Wireless Network of Devices, not uniquely sensors, but with the same features of low-power consumption, low-cost, limited data rate and easy deployment.

Sen.se was very interested in having the team participating in the implementation of their network. It was a strong requirement the use of one of the most known and used protocol standard. First protocol on scene was Zigbee, a widely extended one elaborated by an alliance of the biggest companies in the electronics sector. On the other side a candidate known as 6LoWPAN, a new and promising real standard protocol aimed at constrained networks under development by the IEFT and based on IPv6.

The situation comprised at that moment the performance of two different networks, one for this Thesis under OneNet and a second one with Sen.se using Zigbee or 6LoWPAN. It was agreed that it was useless and unproductive realizing two WSN in two different processes in parallel. Mainly because of a matter of time. Implementing a network protocol can turn into a quite hard and difficult task of unnecessary time consuming. After some discussion it was preferred to continue collaborating mutually with Sen.se getting feedback and important experience in the real market. Since for them it was decisive to use a widely extended protocol, finally, OneNet was dismissed.

4.2.2. Zigbee

Within the first weeks after the collaboration with Sen.se started, around the second quarter of 2010, it was almost decided that Zigbee was going to be the protocol that the devices in the network would understand. The advantages were many apart from the quite obvious concerning low-power, low-cost and easy use. To enumerate just a few:

1. Zigbee is constructed over IEEE 802.15.4 link layer standard for radios. This provides Zigbee with extra features increasing reliability and robustness, getting advantage of mechanisms such as Offset-Quadrature Phase-Shift Keying (O-QPSK) and Direct Sequence Spread Spectrum (DSSS), Carrier Sense Multiple Access Collision Avoidance (CSMA-CA), AES-128 encryption, checksums or retransmission.

2. Zigbee is supported by a huge number of important companies (Freescale, Texas Instruments, Renesas, Philips, Analog Devices...) that also offer off-the-shelf hardware such as system-on-chip solutions that integrate the Zigbee stack and other helpful Zigbee-compliant platforms.

3. ZigBee offers a high degree of versatility allowing deployments of mesh networks, a huge number of nodes or large extensions using routers.

4. The application layer is commonly known as it forms part of the standard and it is distributed in *profiles* and *clusters*. Devices from different vendors utilizing the same profile can interact mutually, offering advantages of interoperability between systems.

However, some of those advantages might also be seen as disadvantages. Despite most companies state their Zigbee stack is open for developers, it is usually hardware dependant so the software implementation works exclusively with certain SoC devices. Moreover, projects should be designed using proprietary programming environment software with the impossibility of developing with an open IDE, or forcing developers to buy a license.

As an example, the Zigbee stack offered by Texas Instruments is excellent for developers especially the API since they provide additional functions and event management mechanisms to build quickly and easily a network. However, a critical part of the stack is provided in a binary code format only understood by specific compilers such IAR Embedded Workbench. That constrain makes the Zigbee implementation to be hard to use in a different platform that it has been designed for.

Texas Instruments provides developers with different versions of a Zigbee stack for different SoCs such as CC2530 with an 802.15.4 compliant radio and a micro-controller (5081) or separately a radio (e.g. CC2520) and a micro-controller (e.g MSP430). It was almost certain after long discussions that a MSP430 micro-controller in conjunction with an CC2520 2.4 GHz radio would be used, as it will be analysed in Chapter 5. The Zigbee stack for that system was available at TI website but only compatible with IAR Embedded Workbench. That software is available for free download as a fully functional 30-days evaluation version or a code size limited version without power debug functionality or runtime libraries. For this Thesis and for the project at Sen.se those free editions were not enough for the requirements of development time or code size, so paying for a license turned necessary. That will be considered a determining disadvantage of using Zigbee.

4.2.3. 6LoWPAN

6LoWPAN emerged from the IEFT forum in 2007. Two RFC proposals were published that year [33]: RFC 4944 deals with mechanisms for transmission of IPv6 packets over IEEE 802.15.4 networks and RFC 4919 focuses basically on the benefits and problems to face when integrating IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN).

6LoWPAN is considered nowadays one of the most remarkable Zigbee competitors. The fact that it is developed by the IETF makes 6LoWPAN to be an open and long term standard for constrained data networks.

Many advantages are helping to continuously increase its popularity (see [31]):

1. 6LoWPAN is based on 802.15.4 MAC layer making use of some of its advantages. Nevertheless, it can also run mounted over other link layers.

2. It is based on the Internet Protocol. It offers better and easier integration with other IP systems and many IP native mechanisms can be used such as routing or security (not defined yet). Transport and application protocols from the TCPIP stack such as UDP can be used over 6LoWPAN.

3. It provides a high degree of scalability. Devices will be part of the Internet and they will have a unique IPv6 address that can be reached from different routes.

4. Great interoperability capabilities. 6LoWPAN devices can interoperate with other 6LoWPAN devices, 802.15.4 devices or other IP-based links (IPv4 and IPv6) such as Ethernet.

4.2.4. Decision

Zigbee is not fully a standard since it has been created by an alliance of companies, it does not offer large scalability since it was designed to operate in a local environment. Zigbee devices from different networks can only interoperate if the use the same API profile. On the other hand, an open standard application layer protocol aimed to run over 6LoWPAN is been defined. It is called CoAP (Constrained Application Protocol) [71] that will be in part implemented in this Thesis and will be described later.

The lack of free development tools and the dependency to the vendors hardware are also major disadvantages found to Zigbee. These facts exposed here make the team to decide on 6LoWPAN as the best protocol alternative to run in the network. There are also very good advantages that would have helped Zigbee to be the selection but 6LoWPAN's advantages are very promising specially when facing the future.

4.2.5. 6LoWPAN implementation: Contiki

Since 6LoWPAN is in continuous development (Jun 2010) by the IETF working group, it is more difficult to find a stack implementation compatible with the latest version of the draft. There is no standalone stack available either yet. Currently there are two popular operating systems specially designed for embedded devices with a built-in IPv6-6LoWPAN stack. One is called TinyOS [12], and open source SO developed by an international alliance formed by the University of California and Intel among others. Contiki [40] is another open source embedded SO but developed by the Swedish Institute of Computer Science (SICS) in Kista, Stockholm.

TinyOS and Contiki are specially designed for devices with constrained resources to work efficiently. Both implement some parts of the 6LoWPAN proposed specification and the TCP/IP stack (with both IPv4 and IPv6 protocols), but the development method is slightly different.

TinyOS uses a dialect of the C language called nesC specially meant for memory constrained devices. TinyOS is built over an event-driven architecture and it uses software components that represent hardware abstractions communicating via interfaces. There are a few platform where TinyOS has been ported to, e.g. Mica2 or Telosb

Contiki is built with an event-based kernel on top of which applications can be loaded dynamically. It also offers a programming abstraction layer to write applications based on a threads model called protothreads. In other words, Contiki combines event-driven programming with multi-threading used in an efficient way, allowing low memory usage, simple applications programming and easy porting between platforms, also avoiding difficult event-driven state machines.

Contiki supports several micro-controller architectures such as MSP430, 8051 or AVR and provides drivers for several radios such as CC2420 from TI. It is written in C and it uses a Makefile to easily port it to different hardware platforms (e.g. Raven).Those factors make Contiki to be lightweight, well supported, portable and a very suitable choice for the project.

Contiki also provides different low level MAC layer implementations. It comes with different communication stacks such as a lightweight stack for low-power sensor networks called Rime, the uIP stack (supporting IPv4 and IPv6 protocols) and the mechanisms of header compression and fragmentation included in the 6LoWPAN drafts [REF]. Additionally also a implementation of the IETF standard RPL protocol for routing IPv6 in low power networks known as ContikiRPL [REF], and a provisional implementation of the IETF CoRE CoAP protocol, an application layer designed for constrained devices that is described later in this chapter.

The uIP stack was initially developed by Adam Dunkels at SICS in Sweden, originally under a BSD style license. In the first release it did not support IP version 6, but only version 4. After continued work by other groups of developers and companies, in October 2008, Cisco, Atmel, and SICS released an IPv6 extension for uIP, called uIPv6 [45].

Underlying protocols such as ARP can be implemented by hardware or firmware, or higher level protocols such as HTTP or SMTP by the application on top of uIP [43].

Considering the benefits that **Contiki** provides and the fact that it is developed in the SICS in Stockholm, making easier to get support, makes it the preferred Operating System.

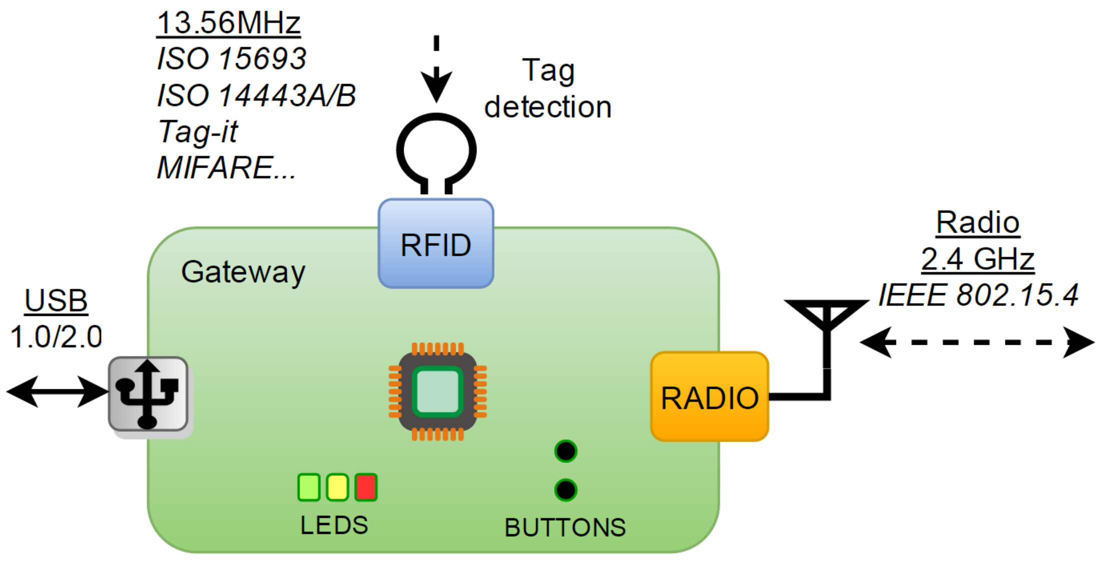


Figure 4.1: Functional diagram of the gateway showing the connectivity options. It is depicted the different communication interfaces, protocols choices and frequency bands in the case of radio communication. It also shows some components such as LEDs and Buttons that can be used for human interaction and testing of the board.

4.3. Gateway

The main objective of the present Thesis is build a gateway that coordinates a wireless sensors network and provide external communication and security based on RFID.

4.3.1. Communication

Communication is achieved in two ways for the gateway. On one side by means of a radio module that establishes links with the sensor boards providing internal communication within the network. On the other side, a USB wired interface that links the gateway to external devices such as a PC, a router or another gateway. Two different wired interfaces were considered for the outer communication: USB and Ethernet. However, the design with the USB interface was chosen, as it was also proposed in first place by Ericsson (see 2) and it is described here.

USB connectivity can be provided by an external adapter chip connected to the microprocessor ports or as an embedded module internally available in some microprocessor models as it will be shown in Chapter 5.

The radio part will be similar for both boards. The most adequate technology for this type of low powered networks is commonly based on the MAC standard IEEE 802.15.4 that 6LoWPAN is based on. Although different frequency ranges can be used by the protocol, the band at 2.4GHz is the most standardized one. The next step would be the selection of radio modules able to work with such protocol at that frequency band, as it is described in 5.

4.3.2. Bootstrapping

At bootstrapping of the network, after all the hardware modules have started, the coordinator will analyse the data traffic in the radio environment to discover other networks. The purpose of this is to find an interference-free channel when possible and extract network identifiers (PAN ids) from packets over-the-air in order to choose a unique one. This is not mandatory in the prototype and those parameters can be previously hard-coded, but it would be required in an automated product that is aimed to be released to the market.

Depending on the application and the needs, the use and handling of encryption might be desired or not. Usually radio chips provide encryption at low layers and that can be taken as an advantage, but other security or authentication methods can be implemented at different layers. More information related to security is provided in section 4.5.

While the coordinating platform is booting up, all the hardware is configured with predefined values, setting memory spaces, clocks frequencies, ports and interruptions, USB module and radio module configurations etc. Once the frequency scanning is performed, the radio parameters are selected. Following, the coordinator will start different processes in specific order starting with the TCP/IP network process in the first place. It will implement mechanisms such as neighbour discovery or forwarding right after the nodes are authenticated. The next process that will run is the RFID process that will continuously read the near field until a valid tag (attached to a node) is detected, authorizing it in the network afterwards. Finally, the CoAP process at the application layer will start. It will rule how the information is requested and delivery for those nodes that are part of the network. This behaviour will be more thoroughly described in 6.

4.4. Sensor boards

The sensor boards design is part of another Master Thesis [3] but part of their design has to be coordinated together with this Thesis and partly done for the application implementation.

The nodes will support the same operating systems and communication protocols. From the hardware point of view, special attention must be put on memory consumption since nodes will be powered with batteries. A good control and management of the radio and processor sleeping cycles is crucial to save energy.

Different types of sensors and actuators will be placed on the prototype board. The collection of sensor data and start of communication between devices is performed by the CoAP application process and then delivered remotely by means of the TCP/IP and 6LoWPAN protocols over the air. The radios have the same characteristics as in the gateway, working at 2.4 GHz and being IEEE 802.15.4 compliant.

Regarding bootstrapping and security, the nodes should use a common key for encryption at link layer if security is desired for the prototype. Other more advanced methods of key sharing can be provided in real world applications. For authentication in the network, the solution using RFID provides the nodes with a tag containing an unique identifier that could be their MAC addresses. A common *neighbor discovery* mechanism from the uIP stack will follow to include the nodes in the network and give them an IP address.

The sensor boards work independently and cannot communicate with other sensor nodes in the first version or the prototype since only star topology is required. Alternatively, some nodes could act as routers to increase the radio range to a more extensive area, but this is a future improvement.

4.5. Security

Security may be integrated in several forms and levels in the system. The mechanisms that provide security must be defined, as well as the requirements. Dealing with wireless sensor networks there are some basic aspects to cover.

Data traffic should be encrypted for privacy and confidentiality, particularly when sensitive data is to be sent.

Access control is another sensitive feature in wireless networks. The devices (sensor boards) must be aware whether the network they belong to is the right one. It may happen that nodes form other networks in the vicinity associate by mistake to our network. Hence, the network might provide some isolation mechanisms from other networks and not authorized nodes to avoid messaging from untrusted participants.

Other security issues involve data integrity, normally performed by CRCs, robustness by the physical radio layer or availability to ensure that the network and nodes are activated and working properly.

Security mechanisms are typically applied at the bootstrapping phase, when the nodes are being detected. It is also common to implement security at different layers. Encryption and redundancy checks can be done at physical layer, e.g. IEEE 802.15.4 protocol also includes AEC/CCM encryption and authorization procedures in the specification. IETF 6LoWPAN working group is trying to integrate security in the specification. There are some discussions about IPsec claiming that it is not suitable to operate with constrained devices [35]. Other mechanisms are being considered such as Secure Neighbor Discovery (SeND). At the application layer IETF core CoAP working group consider IPsec Encapsulating Security Payload (ESP) [77] or Datagram Transport Layer Security (DTLS) [76] over UDP. This is only a proof of the great diversity that can be found.

The network under development will have some optional security mechanisms implemented. Encryption and a particular authorization method for nodes using RFID will be proposed. CRC will be also calculated and added to frames but radio chips today can do this automatically in the hardware. Regarding encryption, AES/CCM as specified by IEEE 802.15.4 can be optionally provided at the link layer by using specific functionality of the radio module. More discussion on this topic can be found in chapters 5 and 6.

The mentioned encryption is based on security keys that are shared between devices. There are a few ways to manage keys. In this case, the network will start up with a common hard-coded key in all devices. This key is shared and known by all nodes and might be assigned during manufacturing and there is no need to transmit it. In order to increase the security, keys can be changed dynamically. Transmission is needed in that case to inform all devices about the new key. To avoid risks of security this information should be sent encrypted to new nodes using the default key that is known by all nodes.

The authorization of new devices in the network will be handled by means of an RFID system. A typical scenario will consist of nodes with tags attached including an id number. Each time the user wants to add a new node he would simply have to pass the sensor over the reader. The reader will capture the identifier number recorded in the tag (e.g. the device's MAC address). That address will be included in a table in the (flash or RAM) memory of the coordinator. The regular traffic in the network uses 6LowPAN with IPv6 addresses that are formed from MAC addresses. Only transmissions to or from the addresses stored in the table will be allowed. This is an optional method but others may apply as well.

4.6. RFID system

There are several parameters that determine the characteristics of an RFID system. A major one is the frequency band that it uses to detect tags. Other parameters such as maximum distance of detection, data rate or transmission power are dependant on the frequency, the type of tag and the specifications of the reader. But there are other issues to consider and select such as the protocol to implement.

All these issues are discussed in this section. The hardware design can be started after their optimal selection.

4.6.1. Frequency band selection

Since the range of frequencies have different implications on a RFID system. The objective here consists of deciding the most appropriate frequency, and therefore the data rate, maximum distance, and other related features.

In order to avoid unnecessary repeated text, it has been preferred to summarize here the details from section 3.2 into table 4.1 for a fast and easy access. This would be helpful when comparing the consequences that different frequency bands have on the development of an RFID system.

As it was stated previously, the mechanisms for energy coupling are different at different bands. In the UHF band, radiative coupling is exploited instead of inductive coupling. The advantages are higher data rates, cheap antennas (e.g. dipoles) large detection distances (increased by adding powered transponders in the tags), and less influence of the antenna orientation at far field. On the contrary, as disadvantages, irregular propagation patterns

	Frequency bands		
	LF	HF	UHF
Coupling	Inductive	Inductive	Radiative
Powering	magnetic field	magnetic field	EM radiation + batteries
Range of detection	cm	cm	meters
Tag capacity	1b-2Kb	16b-8Kb	256b-64Kb
Water penetration	meters	cm - 2m	mm
Antenna type	coil	coil	RF antenna
Antenna size	mm to cm	> 3cm	> 3 cm
Orientation influence	medium	important	medium
Propagation pattern	fixed	fixed	irregular

Table 4.1: RFID frequency bands features

due to reflections might appear, making the detection range to be unpredictable. Moreover frequency band regulations are different at the 900 MHz band in different continents [29]. In addition, the 2.4 GHz ISM band is very saturated by other different services what can be a source of interference.

In this project long detection distances are not needed at all. The first aim of using RFID in the system is for authentication in the network. Sensor boards will have a tag attached that need to be read so it can be identified. In this situation, long distances are even an inconvenient since security gets poor if someone is able to detect sensor data from outside the building. Radiative coupling can be discarded as the powering mechanism of tags.

The RFID system under design is focused on security. The requirements ask for short interrogation area and a medium data rate. It is fundamental to have the capacity of sending an identification number of the node that could be its MAC address. That means that at least 64 bits need to be sent, since it is the maximum size for Extended Unique Addresses (EUI-64) and it can perfectly act as a unique global identifier. It would be also convenient to leave freedom for extra memory capacity to use in future applications. We could also want to exchange encryption keys or similar information. In conclusion, although some amount of data exchange is required, high data rates are not really needed for our purposes. That can be easily achieved working in the HF band.

Regarding the internal memory of the tags, HF systems and inductive coupling are usually provided with EEPROM memories from 16 bytes to 8 Kbytes as it can be inferred from table 4.1.

Antenna size should be suitable for tags to be attached to sensor boards of some centimetres side. Systems working in the LF or HF ranges with inductive coupling allow smaller coils since inductance can be varied by just increasing or decreasing the number of wire turns. Micro-tags could be expensive due to the mechanical process of production since hundred of turns are required to produce enough voltage to power the IC [28].

So far, it seems that the HF band is the most suitable to the RFID system it is being designed. Typical frequency ranges are the ISM bands: 6.78 MHz, 13.56 MHz, 27.125 MHz, 40.680 MHz or 433.920 MHz.

The band centered at **13.56 MHz** is one of the most used in RFID systems from vendors all over the world. A great deal of different RFID chips and solutions can be found working at this band. Most of its features fulfil the requirements except having a detection range dependant on the coil orientation. But in conclusion, all the presented advantages make this band the best option for the system.

4.6.2. Protocol selection

There are several standard and proprietary physical layer protocols for RFID systems (see subsection 3.2.6). In the RF band at 13.56 MHz there are some popular standards such as: ISO 14443A/B, ISO 15693, ISO 18000-3 at 13.56MHz. Considering also proprietary protocols Texas Instruments created Tag-it (simple protocol based on the ISO 15693 [58]) and MIFARE prtocol from NXP Semiconductors (Philip).

Avoiding the physical differences of the protocols such as modulation, ASK, FSK, etc. From the functional point of view, we can set a classification based on:

- Detection distance: proximity (10 cm) or vicinity (50 cm)
- Power consumption
- Data rate

Typically ISO 15693 protocol is considered to reach a vicinity range and perform on low power, ISO 14443 protocols are more complex but can achieve higher transaction speeds within a shorter range in the proximity, but the power consumption required is higher. Unfortunately we can not provide quantitative data to support this discussion more accurately, but these differences are well known within the RFID community.

From this discussion it seems that simpler ISO 15693 protocol is a good candidate since the application does not require much data to be transfer (only max 64 bytes every time a node is added to the network). However, it might be interesting to support different protocols in the prototype. It would be helpful to choose a protocol that is well known, well supported, easy to implement and cost effective. This selection depends closely on the hardware (refer to section 5 where the RFID chip features and implementation details are studied in details). The ideal hardware should support most of them so the reader can detect different tags.

One of the requirements for the selection is imposed by the type of tags available in the lab, those are **Tag-it** and **ISO 14443A/B** in this case. In the case of Tag-it type of tags [60] it provides up to 256 bits of configurable user data and a unique id number of 64 bits. According to the requirements 256 usable bits it is exactly what is needed.

But for a real product another study need to be done to find the most cost effective tag type and hence the protocol.

4.7. Constrained Application Protocol (CoAP)

According to the OSI model, the application layer provides the interaction interface on top of a communication system. Networking protocols such as IPv6 or 6LoWPAN for embedded devices build local networks easily adaptable to connect to the Internet. After nodes establish their connections to other nodes or networks a common application protocol is needed so that both sides understand and interact with each other (e.g. HTTP protocol).

HTTP is worldwide protocol for web communication but not suitable for LoWPANs. Constrained networks are usually found in home automation, energy management or other machine-to-machine (M2M) applications as extensions of the Internet of the Things concept. The designers of such networks have to be aware of the high probability of packet losses, small throughput, power limitations, and memory restrictions. A small code size is crucial for embedded devices using 8-bits micro-controllers with limited memory. An HTTP implementation would be too heavy, not only considering code size but also the length of packet headers, and the unnecessary options that it provides, leading to undesired complexity.

Since the mid of 2010, the Constrained RESTful Environments (CoRE) working group from the IETF [70] has been focusing on the realization of a REST (Representational State Transfer)[1] architecture of the web adapted for constrained nodes and networks. The goal is to achieve a subset of HTTP features to apply to M2M applications. It does not mean to simply compress the HTTP header but rather to map straightforwardly features from both protocols. It takes advantage of some of the HTTP functionalities and ignores others in such a way that optimization for constrained M2M networks is achieved.

Specifically, the CoRE group is aimed at defining a framework to develop applications that control constrained devices and resources such as sensors or actuators. To do that, a new protocol is being written known as Constrained Application Protocol (CoAP) [71], targetting 6LoWPAN networks but also traditional IPv4-IPv6 networks.

CoAP is implemented on constrained devices based on a client-server model. It allows manipulation of *resources* (create, read, update and delete) on a device with sensors or actuators.

There are some features that are listed below:

- Short packet header and simple parsing.
- Asynchronous messaging.
- Operation over UDP with unicast and multi-cast support.
- Optional resource discovery method.
- Optional proxy and packet fragmentation.

[1]REST is a software architecture style for systems such as the World Wide Web that uses client-server models and describes the elements that interact in the communication.

It is convenient to inform that CoAP is subject to vary over time. The specification is currently under development, therefore, the text is found in form of an IETF draft [71]. Different drafts related to CoAP definition have been being studied for the present Thesis. It is very important to mention that the present work is based on version 05 of the CoAP draft. It can be found at [71] as it was mentioned before. The implementation is also based on this document.

4.7.1. CoAP Packet Format

The link layer radio specification IEEE 802.15.4 specifies a maximum size of 127 bytes for a single frame. After subtracting the bytes consumed by the link layer header, the network layer (6LoWPAN/IPv6 in this case) and transport UDP header, there is not much space available for data, not even after 6LoWPAN header compression is applied. As it will be studied in chapter 6 there is a maximum number of 56 bytes available for the application with the configuration we are working with in the present project. CoAP tries to be a high efficient HTTP version that overcomes this constrained situation. The header it adds is as simple as it can be. Messages are codified into binary format, varying sizes from 4 bytes to the maximum frame size, depending on the number of options requested by the protocol and the payload size carried by the message.

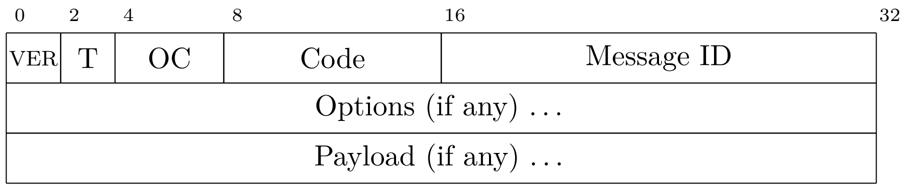


Figure 4.2: CoAP Message Format

As it is shown by Figure 4.2, a CoAP packet contains several fields that are described below:

VER This is a 2-bits field that shows the version of the CoAP protocol. For implementations of draft [71] this value must be 1. Other values might appears with future versions.

Type (T) A 2-bits integer that designates the type of the message. Four values can be possible: Confirmable (0), Non-confirmable (1), Acknowledge (2) or Reset (3), different types of messages with different purposes in the client/server model of the CoAP application. More information in Section 4.7.2.

Option Count (OC) CoAP provides a set of well-defined options that can be used in requests as well as responses. The OC 4-bits number indicates the number of options included in the packet after the Message ID field.

Code A 8-bits field that can be empty or filled with a number announcing a request or with a response code from a reply. See Section 4.7.2 for extended details.

Message ID A 16-bits field used in the interchange of messages between clients and servers to identify unique links and to avoid duplicates.

4.7.2. CoAP Message Types and Codes

The 2-bit header field in the CoAP packet labelled as *Type* accepts four values that refer to different possible message types. Those types define a certain purpose for that packet that implies likewise a particular reply or behaviour from the other part (client or server). Both, requests and replies can be of any type.

As it has been stated previously, CoAP is constructed over UDP, therefore, the reliability that TCP offers is not found. Fortunately, CoAP present a mechanism to provide that reliability using Confirmation and Acknowledgement message types in the same manner as HTTP. Thus, when a message is labelled as Confirmable (CON), the sender will expect an Acknowledgement (ACK) packet carrying a response or being empty. If no confirmation is desired, the message is sent marked as Non-confirmable (NON) meaning that there will be no awareness whether that packet has been received or not. This is a useful way to avoid congestion of the network especially when multicast messages or when sending periodic data from a resource. The fourth message type is the Reset (RST) that appears when a CON message is received but the context needed to deal with it and process it is missing (e.g. because the device has been restarted). In this case, instead of and ACK in response to that CON message, an RST marked and empty packet is sent back to the original source. Furthermore, CoAP helps to reliability by additionally performing CON retransmissions if the first packet was not replied. It also can be done using other control mechanisms at other layers.

Whether a message is a request or response is defined by the number inside the *Code* field of the header. The represented code is a 8-bit unsigned integer with reserved values for requests (1-31), responses (64-191) or empty (0). Other values are reserved for future uses.

Defined codes for requests are GET (1), POST (2), PUT (3) and DELETE (4). Having similar meanings as their HTTP correspondences. Those codes individually imply different actions in the server and their descriptions are well detailed in Draft [71]. In brief, GET ask for a representation value of a resource. POST can modify a resource, its behaviour or even create a new resource. PUT specifically create or update resources with some particular data. DELETE requests a certain resource to be deleted from the server.

Response codes have a different meaning depending on the action that a request produces on a server. These codes that can be mapped to those with similar meaning from HTTP, representing the results of requests on a server. The set of codes has changed from the first CoAP draft. This must be considered if compatibility with older implementations is desired, adapting codes where necessary.

Response codes can be classified in three different categories or classes. They are codified in a way that the first 3 bits represents the class of the response and the other 5 bits give precise information about the result. According to this, classes can get the value: 2 for succeeded requests (received, understood and accepted); 4 for client errors due to bad syntax or impossible actions; and 5 for server error when it cannot complete valid requests.

Some examples of codes that will be used in the implementation of the system are: 2.01 (Created), 2.02 (Deleted), 2.04 (Changed), 2.05 (Content), 4.00 (Bad Request), 4.02 (Bad Option), 4.04 (Not Found), 4.05 (Method Not Allowed) or 5.00 (Internal Server Error).

4.7.3. CoAP Options

CoAP defines up to 15 different options. They are represented by a unsigned integer from 0 to 15. When adding options to a message header it must be done in ascending option number order.

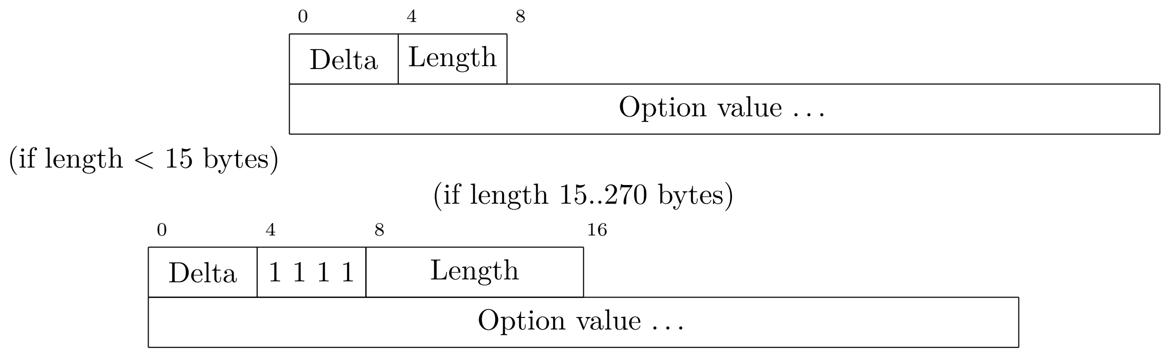


Figure 4.3: CoAP Options Format

Options are carried in a sub-header of variable size that depends on the length of its value. An option value which length is in between 0 - 14 bytes needs only a 4 bits size header field. An option value of 15 bytes long or larger requires a 16 bits header with the first 4 bits set to 1 as it can be seen in Figure 4.3.

Delta A 4-bits unsigned integer used to determine which option is next in the header if any. It is calculated using a delta mechanism of codification between options. It is the result of subtracting to the present Option Number the Option Number of the precedent option in the message or zero if none. The Option Number is then calculated in reception by summing the delta of this option and previous options before it [71].

Length A 4 or 16 bits unsigned integer that represents the length of the value of the current option.

Some options and features from CoAP have been modified or removed since the first version of the draft was released, e.g. Subscription Option from a device to another device in order to receive published value or events has been changed several times.

The OC field in the CoAP header forces a maximum of 15 options that are allowed to be appended to a single packet.

Options can be classified as Critical or Elective. It implies a different action by the destination endpoint when the option is recognized. In order to increase efficiency, some default values are defined and implicitly assumed for some options in the absence of these options.

Table 4.3 enumerates the collection of CoAP options that has been defined at version 05 of the draft. It represents their names, respective numbers, critical or elective nature and default value. All the options except number 10 and 13 are defined in draft [71]. Option 10 is named Observe and it is described in Draft [73]. Option 13, Block, is detailed in Draft [72].

Number	Name	C/E	Default
1	Content-Type	Critical	0 (text/plain)
2	Max-Age	Elective	60 seconds
3	Proxy-Uri	Critical	(none)
4	ETag	Elective	(none)
5	Uri-Host	Critical	Text representing IP address
6	Location-Path	Elective	(none)
7	Uri-Port	Critical	(none)
8	Location-Query	Elective	UDP port
9	Uri-Path	Critical	(none)
10	Observe	Elective	(none)
11	Token	Critical	empty
13	Block	Critical	0
15	Uri-Query	Critical	(none)

Table 4.2: CoAP Options

Content-Type indicates the format of the data in the payload. Typically, default text/-plain value is applicable.

Max-Age is used when caching messages into a memory. It gives the maximum time that a resource representation can be considered valid or fresh.

Proxy-Uri is an absolute URI (Uniform Resource Identifier) and is used when requests are performed to a proxy instead of directly to a server.

ETag (Entity Tag) is an option that, similarly to HTTP, informs that a resource has changed and the information stored in the cache is not fresh. ETag represents the code of the new version of the resource.

Uri-Host, Uri-Port and Uri-Path options identify resources in a CoAP server. These options include the different parts of an URI, helping to parse and reconstruct the path to the resource. URI-Host contains the IP address in literal format, URI-Port carries a port number if the default port [IANA_TBD_PORT] is not used. The URI-path determines the route to access a resource. Commonly numerical IP addresses and the default port are used in the implementation for this Thesis.

The Location-Path and Location-Query options have same meaning as Uri-Path and Uri-Query but they inform in responses about paths for new created resources.

The Observe option is a substitute to the former Subscribe Option. It provides the client with a mechanism to observe changes in resources of remote servers during a defined interval of time. The client has to register with a server so it can send a notification to the client every time a resource changes.

A Token Option is originated by the client to match a request to one or more similar responses one or more serves (e.g. multicast requests). By this, a client can differentiate requests-responses pairs.

The Block Option is a very useful feature that can be implemented as an alternative to IPv6 fragmentation when the size of a payload is too large to be transported on a 802.15.4 packet (max 127 bytes). The mechanism consists of dividing the payload of the CoAP packet in several pieces that will be sent in different responses. Each response will be identified by a number and the client will reconstruct the payload.

The URI-Query will be used with POST requests to make specific queries to a certain resource.

Applicable options for this Project are the URI-Path and URI-Query or Token Option when having high traffic. Observe and Block are also quite valuable options that could be considered, but taking care of compatibility issues with older CoAP version. Options related to proxies or caches will not be considered at this stage but tey might be interesting for future versions.

4.7.4. CoAP URIs

How to access to resources is specified by means of the Uniform Resource Identifier (URI). They define routes and uses of resources in a remote LoWPAN, they must be parsed and constructed following a set of rules specified at [71] (prone to future modifications). Parsing of URI paths usually results in a translation of some fields into options in the CoAP message.

The syntax of the URI scheme in CoAP is generally described as:

```
coap-URI = "coap:" "//" host [":" port] path-abempty ["?" query]
```

The host must be present indicating the IP where the server can be reached. In case a port is not specified in the URI, a default CoAP port assigned by the IANA should be used. Port number 5683 for CoAP service has been requested. Anyway, CoAP should also accept compressed UDP ports ranging from 61616 to 61631 when using 6LoWPAN for compression in the network layer [34]. The path might comprise several members that must be separated by a slash "/". The query is usually filled with a key and value in the form "key=value".

It is possible, and it has been proved, that an external web based CoAP client can communicate with internal nodes of the sensor network through a gateway /citeluis. Taking advantage of an Ethernet 6LoWPAN gateway build in other Master Thesis /citeluis, we could temporarily install a CoAP server in a sensor node [3] access to it remotely from an IPv6 real network. This was done in order to prove the versatility of 6LoWPAN networks and REST applications using CoAP. The next is a practical and typical example that was used for testing operations with an add-on for Firefox browser called Copper [79]

```
coap://[2002:82e5:828b:1234:0207:62ff:fe81:1108]:61616/sensors/led?red=on
```

The example line above shows a common POST request in CoAP format to the host reachable at the IPv6 written between the square brackets to the UDP port 61616. The target is the resource identified by the path "/sensors/led". The query asks for turning the red led on in the in the host as it can be inferred from the pair key and value "red=on".

According to the example, the complete CoAP packet will have its fields filled as:

VER:	1
Type:	Confirmable (0) or Not-confirmable (1). It will depend on the application whether ACK is needed or not.
Op. Count:	2 (URI-path and URI-query)
Code:	POST (2)
Message ID:	16 bits set by the application program.
Options:	
URI-path:	"/sensors/led"
URI-query:	"red=on"
Payload:	""

In Chapter 6, the CoAP implementation that is included in Contiki will be analysed and used as a base. All the needed features for the network will be added where they are not implemented.

There are several more characteristics described in the draft. See [71] for extended details.

Chapter 5

Electronic design of platform

This chapter explores a few of the available hardware platforms in the market to try to find the most optimal hardware, in terms of price, performance and power consumption for our purposes. It details the hardware requirements for the network and describes the process of designing and construction of the platform.

Most of the chip solutions analysed to integrate into the devices are compliant with the studied protocols. For instance, Texas Instruments manufactures chips specifically designed to optimally implement IEEE 802.15.4 and Zigbee. Some of these platforms are showed and commented below.

5.1. Gateway main board

The design of the gateway is the main target of the present Thesis. First of all, a dissertation of what is actually needed is fundamental. The proposed gateway will be equipped with three different modules: a transceiver to perform communication over an IEEE 802.15.4 compliant radio link, a USB module that provides wired communication and an RFID reader. Other desired devices include a low capacity microprocessor, a set of LEDs and buttons for testing, sensors in the case of sensor boards, capacitors and inductors for signal filtering and a micro-controller.

Figure 5.1 depicts the connectivity of the selected electrical components of the platform. The gateway is provided with an USB port to interact with external devices. It is proposed for this first version of the prototype to support connectivity towards a standard PC running Windows or Linux SO. Radio transceivers and modules are normally connected to micro-controllers through a SPI or UART port. The same case applies for RFID readers. The USB can be implemented using additional chips such as FTDI that typically uses an UART connection to the micro-controller. But also some micro-controllers are built with and internal USB module, avoiding the extra connectivity between chips that can slower the performance.

There are many factors that have to be considered to avoid future problems. For instance, compliance with different version protocols or the interaction between chips from different vendors might be a possible source of troubles (different voltages, clock requirements...). Performance, power consumption and price need to be optimized. Radio design, including the radio module, antenna design, transmission power or sensitivity should also be looked with special attention to avoid interferences and malfunctioning.

It is very useful to classify all these devices by vendor, protocols supported, specifications, connectivity, prices and so on. Tables 5.1, 5.2, 5.3, 5.4 and 5.5 below give a better view of the huge collections of devices (SoC) available in the market today. In general, the values shown for data rates and power consumption are considered typical values. Prices are obtained from vendors websites or, if not available there, on the sites `www.mouser.com`, `www.digikey.com`, `www.farnell.com` and `www.aliexpress.com`. Usually prices are given per unit for a minimum certain number or units. Typically 1000, 2000 or 5000 units.

5.1.1. micro-controllers

The selection of the components will be highly dependant on the central micro-controller. But first, there are several features that are crucial for the selection of the controller:

- **Connectivity**: to connect to an USB interface, 2.4GHz radio and an RFID module.
- **Performance**: fast to be able to support different communication ports and manage a WSN.
- **Memory**: enough capacity for the Contiki SO, drivers, network and RFID protocols.
- **Price**: as cheap as possible.
- **Support**: from vendor or community of developers.
- **Consumption**: specially needed in battery powered nodes.

The micro-controller is the central piece of the platform. It must provide at least one port for each device. That is three different communication channels of a certain type. There are different serial and parallel communication standards used in embedded systems. Commonly known are the UART (Universal asynchronous receiver/transmitter), SPI (Serial Peripheral Interface Bus), I^2C (Inter-Integrated Circuit Bus) or Parallel GPIO Bus (General-purpose input/output).

The micro-controller will coordinate several applications running inside requiring a great amount processing. It is necessary then to have a fast processing unit at least in the coordinator board. Modern microprocessors are usually designed to work from 8 MHz up to 32 Mhz. Since we do not have a specific requirement for this, the preference is to have a wide range so we can test the performance and different velocities. For example, selecting a 32 MHz based micro-controller would allow us to test the performance at different speeds (32, 24, 16, 8...) in case a lower speed is not enough for the gateway. It has to run several processes and handle different communication interfaces what consumes many hardware resources and processing, as it will be shown in Chapter 6. On the other side, this speed

Company	Model	Existing protocol implementation[a]	Consumption[b] (mA)	Price ($)
Renesas	R8C-1A/1B	ONE-NET	9	1.50-2 [1ku]
Renesas	R8C-26/27	ONE-NET	10	1.88-2.38 [1ku]
Freescale	68HC08 (HC08)	ONE-NET	12	0.90 [10ku]
Texas Ins.	MSP430F2xx 16MHz	Zigbee, 6LoWPAN	N/A	2.70-5.80 [1ku]
Microchip	PIC18	MiWi	N/A	1.37-4.13 [5ku]
Microchip	PIC24	MiWi	N/A	1.16-4.83 [5ku]

[a]An open source implementation of such protocols exists and it has been adapted to the specific micro-controller.

[b]Consumption is given as maximum values.

Table 5.1: micro-controllers

requirement can be softened for the sensor boards since they only implement one application, one interface and there is no need of constant transmission.

According to the documentation from different vendors and the protocols specifications, low-cost 8-bit and 16-bit processors are well-suited for low capacity sensor networks. In most cases, having 16K ROM and 1K RAM is usually enough for a simple application. For the platform in this project at least 60 kBytes are needed for Contiki [42] and still the code size for the application and drivers need to be added. Assuming this extra size in 30 kB, considering vendors stimations, in conclusion, aproximately a minimum space of 90 kB will be needed.

Different 8-bit processors from different vendors were considered for the project. The list was reduced to the most popular ones in the market at the time of the design. The hardware recommendations included in the protocols documentation were additionally taken into account as well as the their popularity. That increases the possibility to get access to previously tested open source code and support from software communities. Additionally, in most cases, source code is provided with an API to develop your own application.

The classification of some popular micro-controllers can be seen in table 5.1. Prices vary within a range due to the differences in capacity and features of the product family. Those appearing here are typical values for the micro-controllers that could fit in the platform.

Texas Instrument's micro-controllers are very appropriate since 6LoWPAN implementation in Contiki has already been ported and tested. Moreover, they are attractive in terms of cost, features and open source code availability. There is also good support from TI vendor, making easier the integration with some of their products such as radios or RFID chips. TI's MSP430 families have a good diversity regarding CPU speed, memory, ports, etc. MSP430F2xx family is shown here just as example, but others will be considered. Due to the advantages, **MSP430** micro-controller architecture is was chosen to start the work of the platform. Texas Instrument offers a great variety of MSP430 models in their catalogue [48].

Renesas and Freescale chips are officially supported by ONE-NET protocol community [9]. PIC micro-controllers have been tested for the MiWi protocol. MSP430 processors have been tested with different protocols implementations based on IEEE 802.15.4 radio standard as well as ONE-NET, what makes it a very versatile alternative.

Porting based on hardware such as and Microchip PIC or Atmel AVR is under development. In addition, other platforms supporting the IEEE 802.15.4 radio standard would be well suited to carry 6LoWPAN or Zigbee on top, since both use the same radio standard.

In subsections 5.1.2 and 5.1.3 there are some SoC including a radio or USB module already built-in. That can be a great advantage in terms of board space, communication speed and price, since there is no need for an extra chip and a link to the controller.

5.1.2. Radio module

In wireless networks there are two possible hardware configurations: a micro-controller connected to a external **radio transceiver** or an independent **radio module** or System-on-Chip (SoC) integrating a micro-controller. The first case is more flexible and better performance can be achieved. The second is usually more optimal in terms of size, power consumption, easiness of development or price.

From the protocol perspective there are specific alternatives to implement the network.*6LoWPAN* requires that radio modules are compliant with the IEEE 802.15.4 specification. Most popular 6LoWPAN implementations are available in Contiki SO and Tiny OS. Contiki has ported its code in platforms such as: cc2420dbk, cooja, esb, ethernut, netsim or sky using radio modules such as CC1100 or a transceiver CC2420 from Texas Instruments with an MSP430 processor.

Transceivers that have been tested as working with ONE-NET include: RF Monolithics TRC102, Semtech XE1203F and XE1205, Analog Devices ADF7025, Integretion Associates IA4421, Texas Instruments CC1100 and Micrel MICRF505. In case of microprocessosr, models that have been tested are the TI MSP430, C8051, Renesas R8C and Freescale 68HC08.

Texas Instruments provides supported code for different platforms and some of its proprietary networking protocols. This information can be consulted at their RF producst catalogue [47]. For instance:

SimpliciTi is available for platforms based on: CC1101 or CC111x radio with an MSP430, CC2500 or CC2520 radio with an MSP430, or integrated System-on-Chip such as CC2430, CC251x, CC2530

TI-MAC and *Z-Stack* (Zigbee) is available with platforms: CC2420 or CC2520 radios with an MSP430, and CC2430 or CC2530 SoC modules.

MiWi protocol, developed by Microchip, can be also easily implemented in their MRF24J40 2.4GHz transceiver that supports IEEE 802.15.4 and ported across PIC16, PIC18, PIC24 and PIC33 devices.

Company	Model	Protocols	Freq. (GHz)	Peripherals	RX [a] (mA)	TX (mA)	Price($)
Texas Ins.	CC2520	Zigbee, SimpliciTi	2.4	6 GPIO, SPI	18.5	33	1.95-2.15 [1ku]
Texas Ins.	CC2420	Zigbee, 6LoWPAN, SimpliciTi	2.4	SPI	18.8	11-17	N/A
Texas Ins.	CC1101	SimpliciTi	<1	SPI	14.7-16.3	12-27	1.85-2.05 [1ku]
Texas Ins.	CC1100	ONE-NET, Zigbee, 6LoWPAN	<1	SPI	16	15-30	2.30 [1ku]
Micrel	MICRF505	ONE-NET	<1	3-wire SPI	13	28	4.5 [1ku]
Semtech	XE1205	ONE-NET	<1	SPI	14	62	N/A
Microchip	MRF24J40	Zigbee, MiWi	2.4	6 GPIO, SPI	19	23	2.36 [5ku]
Microchip	MRF24J40MA	Zigbee, MiWi	2.4	SPI	19	23	4.86 [100u]
Freescale	MC13202	Zigbee	2.4	7 GPIO, SPI	30	37	3.52 [1ku]

[a] Power consumption in sleep (standby mode) is $< 1\ \mu A$ for all the chips except the MRF24J40 and MRF24J40MA ($2\ \mu A$)

Table 5.2: Radio transceivers

For *Dash7*, working at 433 MHz, ST Microelectronics and Analog Devices offer different solutions and Texas Instrument recommends their CC430 sub-1 GHz RF SoC (CC1101 RF transceiver with a MSP430).

Table 5.2 shows a list of the most popular radio transceivers in the market. Some of them mentioned previously. Most of these transceivers are well known and used in many embedded devices. For some of them there is a good support provided by vendors and open source communities. Protocol stacks are provided free of charge when used with a Microchip's PIC® micro-controllers and the MRF24J40 transceiver. Texas Instruments offers their own Zigbee with IEEE 802.15.4 implementation and their proprietary SimpliciTi protocol and Microchip do the same with MiWi Protocol.

As a general characteristic, these transceivers communicate with the MCU through a serial SPI in all cases or a parallel port. Prices are quite similar with just bigger differences regarding the MICRF505 from Micrel or MRF24J40MA from Microchip. Among the most economic ones the CC2420 from TI is not recommeded for new designs and the CC1100 and CC1101 don't operate at 2.4GHz, what leaves only two candidates, the MRF24J40 and the CC2520 transceivers. Finally the **CC2520** have the advantages of being cheaper and produced by TI like the MSP430 micro-controller chosen previously, providing support for their interaction and avoiding compatibility issues. Moreover the CC2520 is very similar to the CC2420 that has already been ported in Contiki, what could be helpful in the development of drivers.

Table 5.3 shows a list of some popular radio modules that integrate an micro-controller that could fit the requirements of the gateway.

The main advantage of modules is the reduced space that they require and the low power consumption. AS a disadvantage these micro-controllers usually have limited memory and performance compared to the versatility of standalone controllers. Regarding connectivity, one communication port is internally occupied by the radio transceiver. Table 5.3 shows different connection ports available in the radio modules. But it has to be considered that the physical pins are usually shared by the different communication buses and selected by means of a multiplexor, meaning that these interfaces are not all available at the same time. This becomes a problem when still two more connections are needed for the USB module and the RFID reader.

Selecting exclusively the devices working at 2.4 GHz and ignoring of the most expensive ones reduces the selection to the CC251x, CC2430, CC2530 and MC13213 modules. More in detail, in the specific case of the CC251x TI chip, this module is very versatile and combines a radio transceiver, a micro-controller and a USB interface. However, the flash memory to store the code is still too small for the size of the application and Contiki. Being 32 KBytes while at least 60 kBytes are needed for Contiki [42]. The same thing happens with the MC13213 from freescale. In this case it has some more memory up to 60 kBytes, but still too limited if we want to give some margin for other applications. The decision has to be made between the CC2430 and the CC2530 modules from TI, but considering that the CC2430 is older and more expensive, that gives only place for the **CC2530**.

5.1.3. USB module

The aim is to provide the gateway with a standardized cable connectivity to other devices such as computers, routers or other gateways. For this first version of the platform only a PC is considered as en external host.

Usually low-power micro-controllers use SPI,UART or I^2C ports for communication, but those are not common for platform external connexions. Fortunately, in the embedded world there are different alternatives to implement USB links. On one hand, a very extended way of connection from a micro-controller is to use a chip that bridges from UART port to USB, oftenly chips from FTDI vendor [69]. Unfortunately, the link between the micro-controller and the FTDI chip (typically an UART port) reduces drastically the velocity of the USB specification. Moreover, adding another chip to the board implies higher cost and higher power consumption.

A second possibility consists of a micro-controller with a USB module built in. This makes the bus to be much faster since no intermediate bridge between UART-USB is needed. It also reduces cost and energy consumption but it is not necessarily cheaper, a micro-controller might be more expensive.

Company	Model	Protocols	Freq. (GHz)	Peripherals	RX [a] (mA)	TX (mA)	Price ($)
Jennic	JN5148	Zigbee	2.4	2 UARTs, SPI, 2-wire, 4-wire	18	15	4.57 [100u]
Freescale	MC13213	Zigbee	2.4	2 SCI, up to 32 GPIOs	30	37	3.70 [1ku]
Ember	EM250	Zigbee	2.4	17 GPIOs, SPI, I^2C	28-30	24-34	4.57 [2ku]
Atmel	128RFA1	Zigbee	2.4	2 USARTs, SPI , 2-wire	16.6 [b]	18.6	4.78 [10ku]
Texas Ins.	CC2430	Zigbee, SimpliciTI	2.4	2 UART, 2 SPI, 21 GPIOs	27	27	3.60-6.05 [1ku]
Texas Ins.	CC251x	Zigbee, SimpliciTI	2.4	2 UART, 2 SPI, 21 GPIOs, USB	22	23	2.85-3.35 [1ku]
Texas Ins.	CC2530	Zigbee, SimpliciTI	2.4	2 UART, 2 SPI, 21 GPIOs	24	29-35.5	3.05-3.85 [1ku]
Texas Ins.	CC111x	SimpliciTi	<1	2 UART, 2 SPI, 21 GPIOs	18.9	18	3.05-3.35 [1ku]
Texas Ins.	CC430f5x	Dash7	< 1	2 SPI, UART, I^2C	16	N/A	4.15-5.00 [1ku]

[a] Power consumption in sleep mode (standby mode) is 1 μA for every chip except the JN5148 (1.25-3.25 μA)

[b] Radio transceiver + active CPU

Table 5.3: Radio modules

Company	Model	USB version	Speed	Peripherals	Architecture	Price ($)
NXP	PDIUSBD12	2.0	Basic	Parallel	Philips SIE	N/A
NXP	ISP1181A	2.0	Full	Parallel	Philips SIE	N/A
FTDI Chip	FT232RL	2.0 compatible	Limited by UART up to 3MBaud	USB-UART	FTDIChip	2.65 [2ku]
FTDI Chip	FT245RL	2.0 compatible	limited to 1Mbps	USB-parallel FIFO	FTDIChip	2.65 [2ku]
Texas Instr.	TUSB3210	2.0	Full	GPIO	8052	3.15 [1ku]
Texas Instr.	TUSB3410	2.0	Full	Enhanced UART	8052	2.25 [1ku]
Texas Instr.	MSP430F5519	2.0	Full	Enhanced UART, 2 SPI, I2C, 63 GPIO	MSP430	3.90 [1ku]
Texas Instr.	CC251x	2.0	Full	2 UART, 2 SPI, 21 GPIOs		2.85-3.35 [1ku]

Table 5.4: USB modules

Table 5.5 depicts a list of chips that can provide external USB connectivity. NXP offers interface devices that can connect to a micro-controller through the general-purpose parallel interface (GPPI), unfortunately NXP do not publish the prices. They have to be contacted and approve the request. Similarly, FTDI sells broadly known USB adapters connecting to asynchronous serial interface (UART) and parallel FIFO bidirectional interface at a reasonable price. Texas Instruments also offers chips that adapt serial UART or parallel GPIO to USB, but in addition Family 5 of MSP430 micro-controllers include a USB module integrated in the SoC providing direct external USB connectivity. The CC251x devices were already mentioned in Table 5.3 and rejected due to small memory space, but could be used as just an interface.

Three alternatives were selected from the table. **TUSB3410** is the cheapest one with the only disadvantage that the drivers can be a bit tricky to configure. Meanwhile the **FT232RL** and the **FT245RL** are slightly more costly but they are simple to use, just like a plug-n-play device. The last option is the USB module integrated in the MSP430F5519 that simplifies the hardware and avoids the slower serial links.

USB 1.0 specified data rates of 1.5 Mbps (Low Speed) and 12 Mbps (Full Speed). USB 2.0 specifies maximum rates of 480 Mbps (High Speed), but due to bus constraints the effective throughput is usually reduced [65]. We can observe that USB 1.0 at Low Speed is enough to fulfil the requirements of the network throughput. All these devices comply with USB specification Rev. 2.0 supporting either Low Speed or Full Speed that fulfilling the requirements of the platform (at least 32 Kbps throughput). They can be programmed to act as client devices in the USB communication. The computer will work as a host in that case, recognizing the attached device and configuring the environment to establish the link. The selection will be determined in Section 5.1.5 when the rest of components have been exposed in order to find the best trade-off of performance and cost.

5.1.4. RFID module

The RFID module comprises a dedicated chip, and an RF circuit adapted with an antenna at 13.56 MHz standard frequency. Reader ICs available in the market commonly work as an analog front-end towards the micro-controller. These chips usually implement some of the characteristics specified by the protocol at physical layer level, detecting signal level,gathering data frames, supporting different modulation schemes, calculating CRCs and so on.

The physical layers of different RFID protocols are typically mutually incompatible. Tags are designed to work only under certain protocol rules. When interoperability is desired it is the reader that should support and manage those protocols. The reader should also be able to modify its uplink and downlink transfer schemes and resolve collisions. Such a job is relatively easy to achieve as readers IC support computational operations.

In the market there are a wide range of products with different characteristics and supported protocols. There is a short list of some populer RFID readers in 5.5

Company	Model	Protocols	Frequency	Comm. Port	Price ($)
NXP	CLRC63201T	ISO14443A, ISO14443B, ISO15693	13.56 MHz	Parallel/SPI	10.4 [1ku]
NXP	MFRC50001T	ISO14443A	13.56 MHz	Parallel	6.35 [1ku]
NXP	MFRC53101T	ISO14443A, ISO14443B	13.56 MHz	Parallel/SPI	8.5 [1ku]
Texas Instr.	TRF7960	ISO14443A, ISO14443B, ISO15693, Tag-It ISO18000-3	13.56 MHz	Parallel/SPI	3.95-4.05 [1ku]
Texas Instr.	TRF7961	ISO15693, ISO18000-3	13.56 MHz	Parallel/SPI	3.25-3.35 [1ku]
Texas Instr.	TMS3705	ISO11785	134.2 kHz	Serial SCI	5.25
Atmel	AT88RF1354	ISO14443B	13.56 MHz	TWI/SPI	1.33 [1ku]
Atmel	EM4094	ISO14443A, ISO14443B, ISO15693	13.56 MHz	SPI	4.00 (1u)

Table 5.5: RFID modules

Not all chips in table 5.5 offer the same features and support the same protocols. They can usually be configured via register values that can be selected by the designer. However not all of them provide the same configuration versatility. For detailed information, the reader can refer to the data-sheet of each product.

From table 5.5 it is possible to select the most suitable devices for the platform. As stated in 4 the more number of protocols supported the better to extend the prototype. However, the interested is focused on the ISO15693 protocol (Tag-it is a TI proprietary protocol based on ISO15693), more suitable for low speed and low power consumption. As a second criteria, the price should also be very competitive. Considering these factors **TRF7960** from TI can be considered as the best candidate because it fulfils all the requirements. Additionally TI provided a firmware implementing all shown protocol alternatives what will ease and fasten the software implementation of the platform.

5.1.5. Platform components selection

Finally it is time to find the most optimal hardware configuration to build the gateway platform. Several different alternatives can be extracted from the analysis of the devices that are available in the market.

Possible configurations

More precisely, there are three different hardware configurations that can be selected to implement the platform. Most of the devices that can be part of these configurations have been previously selected and those devices with no price available or discarded previously are not considered here.

Config 1:								
MICRO	Price($)	RADIO	Price($)	USB	Price($)	RFID	Price($)	TOT($)
		CC2430 F128RTCR (8051)	6.03	FT232RL FT245RL	2.65	TRF7960	4	12.68
		CC2530 F128RHAR (8051)	3.5	FT232RL FT245RL	2.65	TRF7960	4	10.15
		CC2530 F128RHAR (8051)	3.5	TUSB3410	2.25	TRF7960	4	9.75
Config 2:								
MICRO	Price($)	RADIO	Price($)	USB	Price($)	RFID	Price($)	TOT($)
MSP430 F2416(92k)	5.3	CC2520 RHDT	2.1	FT232RL FT245RL	2.65	TRF7960	4	14.05
MSP430 F2416(92k)	5.3	CC2520 RHDT	2.1	TUSB3410	2.25	TRF7960	4	13.65
Config 3:								
MICRO	Price($)	RADIO	Price($)	USB	Price($)	RFID	Price($)	TOT($)
MSP430 F5519(128k)	3.9	CC2520 RHDT	2.1			TRF7960	4	10

Table 5.6: HW configurations

1. Radio module + USB adapter + RFID module
2. Radio transceiver + micro-controller + USB adapter + RFID module
3. Radio transceiver + micro-controller integrating USB + RFID module

To make more clear the comparison between the differences between the three platform alternatives, all devices are shown in table 5.6 together with its respective costs. It is important to clarify that these prices correspond to that device in the specific family that accomplish with the minimum requirements. For instance, a minimum of 90 kB of memory and a sufficient number of ports for connectivity in all micro-controllers. The prices also depends on the number of parts to be ordered, deciding in this case a minimum of 500 units. Texas Instruments provides an extensive catalogue for all MSP430 families and RF producst that can be consulted in order to get more details [48], [47].

From table 5.6, in Config 1 the CC2430 can be discarded because it is almost obsolete and not recommended. The alternative is to use a CC2530 with a controller based on the 8051 architecture and 128 kB of flash memory. The only variation is to use a USB adapter from FTDI (FT232RL and FT245RL) or the TUSB3410 from TI. The second is slightly cheaper but the development of the drivers can be more difficult, as stated by developers in the community forum.

The Config 2 uses an MSP430 belonging to Family 2 of TI micro-controllers with a minimum of 92kB that is required for Contiki and the application. The same differences are only regarding the USB converter chip. In this case, this configuration can be discarded since it is much more expensive than the alternatives in Config 1 and Config 3.

Finally Config 3 shows the most optimal selection of components for this configuration. It uses an MSP430 of TI's Family 5 that includes and internal USB module and 128kB of flash memory, with enough connectivity options. The radio is a CC2520 and the RFID chip is the versatile TRF7960.

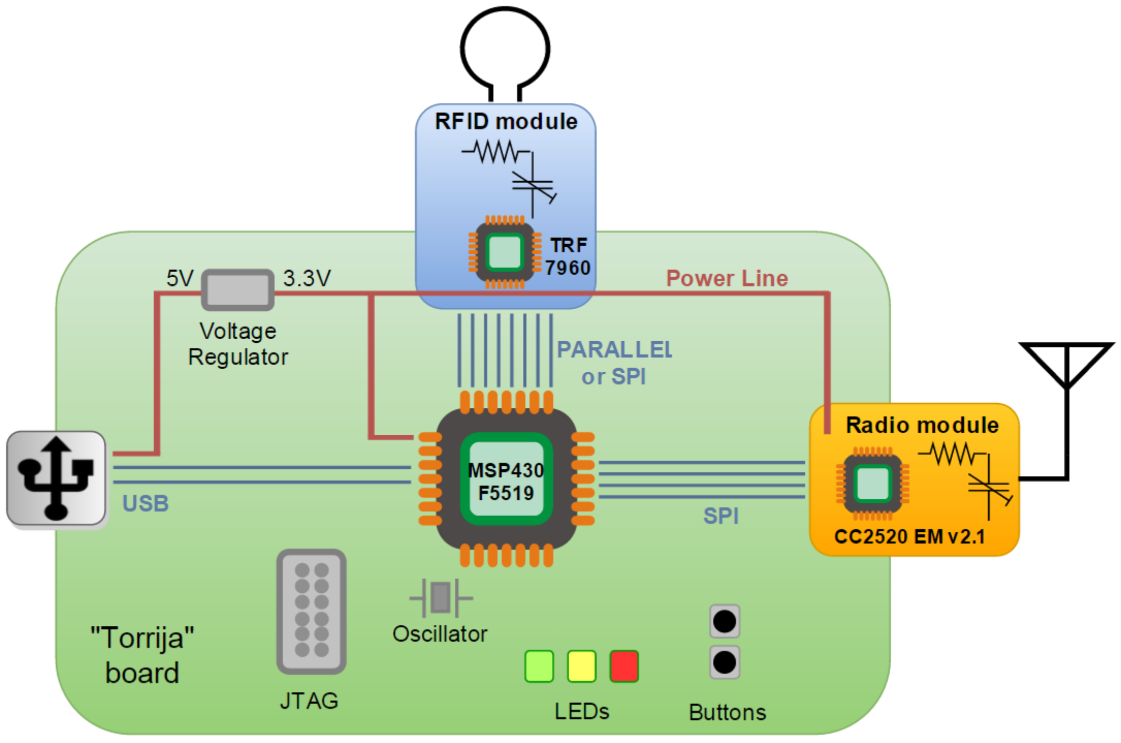


Figure 5.1: Electronic design of the gateway. It depicts the bus connections of the selected communication components (USB, RFID and Radio) towards the microprocessor (in blue), it also shows how the board is powered from the USB line (in red) and also other required elements, such as a JTAG connector used to flash the micro-controller, a external crystal, LEDs and buttons for human interaction and testing. The RFID and Radio modules include the corresponding chip model and a circuit representing the antenna adaptation that is needed.

In terms of cost, the cheapest solution is the third alternative of Config 1 (9.75 $) but in terms of performance it can be beat by the only alternative in Config 3 that is slightly more expensive (10 $) but uses an MSP430 controller which benchmarking gives much better results than that of the old 8051 controller as it is shown in TI proprietary MSP430

Competitive Benchmarking [52]. This comes to the conclusion that **Config 3** alternative is the most adequate for the platform.

5.2. Other components

Other materials are needed as part of the circuitry to adapt the signals, for testing purposes or they are just required for the chips to make then work properly. Some of these circuits are recommended in the documentation of the chips but often some calculations are required to select the adequate components. For a complete bill of materials, please look at Appendix A where the schematic is available.

Gateway Resistors, capacitors, inductors and rest of parts are standard components with small sizes and SMD packaging.

Other components such as LEDs were selected by their low power consumption (current up to 2 mA) and SMD technology.

In order to regulate and stabilize the input power signal, a signal regulator is needed (MCP1603 from Microchip), a Schottky diode and a high value inductor.

As a reference signal to synchronize the clock, the board uses two crystals. A mandatory low frequency one (32.7kHz) and an optional high frequency one (32MHz)

Several connectors are used for external connection. Specifically, a mini USB type B connector and socket connectors for the JTAG interface, the radio and RFID module and some extra to provide direct access to the unused micro-controller pins.

Radio module In order to save time, board space and likely problems with the impedance adaptation of the antena or soldering, it was decided to use a pre-built TI plug-in modules [55] based on the CC2520EM Reference Design v2.1 that provides and antenna, adaptation circuit, the CC2520 chip and the connector to attach it to another board.

RFID module The RFID module is was designed and built in the KTH Wireless lab. It requires and external antenna (made as a coil), an adaptation circuit made out of resistors and capacitors, a 13.56 MHZ crystal and connectors for the antenna and external pins connections towards the microprocessor.

Antenna (coil) The antenna is basically a coil made from a rolled up cable and a adaptation circuit compounded by a resistor ($1K\Omega$), a fixed capacitor (22 pF) and two variable capacitors with different values for gross matching (range from 10 to 120 pF) and another for accurate tunning (range from 5 to 15 pF) in order to find 50 Ω impedance to match the impedance of the TRF7960 chip.

5.3. Sensor devices

Sensor boards are designed as part of a different Thesis work [3]. In this case the requirements are changed since the constraints are focused on the power consumption and the processing of the sensors information.

Radio modules are a good alternative for sensor boards where only one radio link is needed and not much more processing other than reading data from the sensors. The required capacity and also connectivity required is less demanding than in the gateway. A USB communication interface is not necessary, but on the other hand, the micro-controller needs to have analog-to-digital converter input channels (ADC pins) to read data from analogue sensors. In addition it also needs buses such as standard I^2C that is required by some sensors. In the case of the prototype board the controller MSP430F5634 is used since it contains ADC and I^2C communication buses. The radio transceiver is the CC2520 chip from TI as in the gateway case.

5.4. Design and construction

Several circuits are needed to correct, adapt or stabilize the electrical signals between the different components. This section explains what is required and how it is implemented in the platform. The schematics and printed circuit boards are provided in Appendix A.

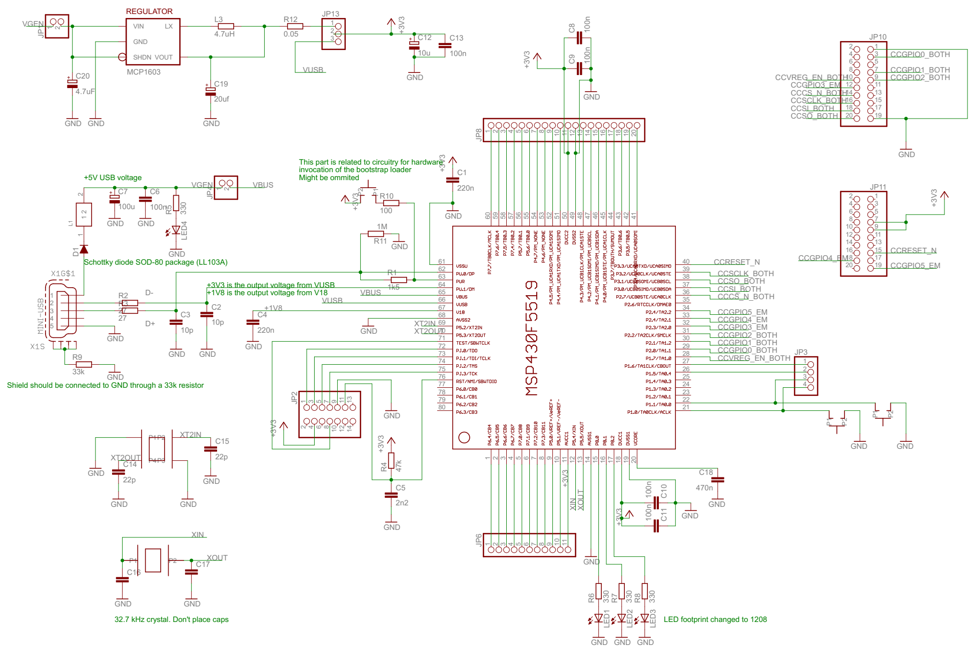
5.4.1. Development tools

For the design of the board layouts and schematics a free powerful graphic editor tool called EAGLE in version 6 was used. It is developed by CadSoft Computer (www.cadsoftusa.com). The board layouts were exported on a Gerber type file format and edited with a CAM software (CircuitCAM for Windows version 5.2). The output files from this tool were used as input for a milling machine (LPKF Laser and Electronics AG model "Protomat S42") available at the Wireless@KTH lab [1] for the construction of the boards using the machine software BoardMaster, version 5.0.1100.K02. All software related to the Protomat S42 is copyright LPKF Laser and Electronics AG. The lab was also equipped with other useful resources for the Thesis such as precise solders with micro sized tips for soldering micro components, and impedance meter for the RFID antenna design (Philips RCL meter. Model PM6303A), tags, cables and component parts and computers.

5.4.2. Gateway

As a personal choice of the designer, the prototype board under construction received the name of "Torrija" board honouring an ancient Spanish dessert made of sweet bread with a similar shape.

The board is powered from the power signal coming from a USB device or a USB charger providing 5 volts and 500 mA of current that is enough power for what the board requires. According to the specifications the CC2520 radio consumes 18.5 mA when receiving and



REGULATOR
MCP1603
MSP430F5519
This part is related to circuitry for hardware invocation of the bootstrap loader Might be ommited
+5V USB voltage
Schottky diode SOD-80 package (LL103A)
+3V3 is the output voltage from VUSB
+1V8 is the output voltage from V18
Shield should be connected to GND through a 33k resistor
32.7 kHz crystal. Don't place caps
LED footprint changed to 1208

Figure 5.2: Gateway schematics

between 33.6 mA and 25.8 mA in transmission [53]. The TRF7960 RFID reader consumes 10 mA [56] when reading actively. Finally the MSP430 varies its power consumption depending on the system clock settings (typically 8-16 MHz, up to 25 MHz) and it can be computed for both type of memory: flash or RAM. It typically varies between 5-10 mA [49]. In the worst case scenario, the amount of power required will be 33.6 mA + 10 mA + 10 mA = 53.6 mA without counting the power consumed by other passive elements in the board. In any case, it is far beyond the 500 mA provided by the USB interface.

The signal goes first then into a Schottky diode for protection against reversing power supply. Then, since the signal from a computer can be very noise, a small circuit is needed to filter it that includes a ferrite bead to suppress high frequencies in parallel with two capacitors and a LED that informs that the power is on. Once the signal is clean, it needs to be converted from 5 V to 3.3 V that is the typical voltage that is required by all the chips in the platform. There are two options to do that, it is possible to use the MSP430 internal voltage regulator (LDO) in the USB module or the external regulator. In this this circuit both options are available (See schematics in Appendix A).

The disadvantage of the internal LDO voltage regulator is that it can only provide up to 12 mA [66] [67] so it would be more adequate in simpler designs with fewer requirements,

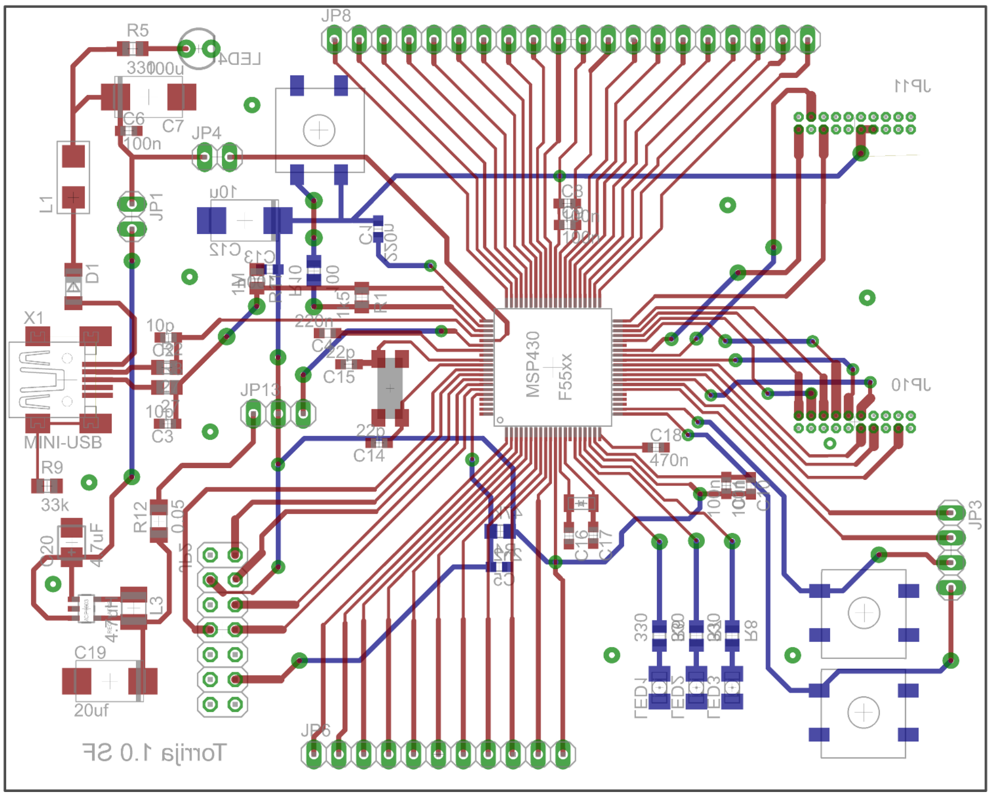


Figure 5.3: MSP430 circuit board

but it might be not enough for the RFID and 2,4GHz radio modules that are also fed from the same power line. It was decided then to use a external voltage regulator (MCP1603T from Microchip) that can convert a variable input signal (2.7 V - 5.5 V) into a fixed voltage signal of 3.3 V offering a maximum of 500 mA current. A jumper included on the board let you choose the input source between both regulators. There are additionally two capacitors of different value set close to every power supply pin of the micro-controller to eliminate possible noise and provide a stable wideband power supply to the pins. These capacitors give also stability to those pins that need to be permanently in a high state and are fed by the power signal.

The design of the USB communication circuit is based on TI's guidelines [66] and it includes some capacitors for filtering (decoupling capacitors) and resistors to set a desired value on certain pins (high or low state). Moreover, there is an additional circuit to invoke the bootstrap loader (BSL) if firmware updates are desired through the USB connection.

In order to provide a clock source, the micro-controller can use an oscillator from external crystals an a resonance circuit. Two different sources are commonly needed, a mandatory low frequency one that oscillates at 32.768 kHz in this case (it will be used to set Timers in Contiki. See Chapter 6), and an optional one for high frequency that can also vary but in this case it oscillates at 32 MHz. The maximum speed of the MSP430F5519 is 25 MHz but it can run at other speeds with the use of the internal micro-controller divisors (normally it will be set to run at 16 MHz, but it can be configured for 8 MHz and 24 MHz for performance testing). There is also the possibility to use the internal VCO that can provided a clock source as well but the external high speed oscillator is required by the USB module to work [67], in addition, it normally generates a more stable clock and clean signal that the internal VCO improving the design.

In order to achieve the oscillation the crystals need a external load capacitance that adds the right phase to the loop. This load capacitance must be specified in the device's datasheet. The frequency of the oscillation can deviate depending on the load capacitance and the parasite capacitances appearing internally or at the soldered junctions to the board. These extra capacitances must be compensated with additional capacitors mounted in parallel at both pins to adapt to the right load capacitance. The value of these additional capacitors can be obtained from the formula:

$$C_{Load} = \left(\frac{C_1^{'} \cdot C_2^{'}}{C_1^{'} + C_2^{'}} \right)$$

with

$$C_1^{'} = C_1 + C_{Parasitic}$$

$$C_2^{'} = C_2 + C_{Parasitic}$$

Considering that both capacitors can have the same value. The equation can be simplified into:

$$C_{Load} = \frac{C_1}{2} + C_{Parasitic}$$

or

$$C_1 = 2 \cdot C_{Load} - C_{Parasitic}$$

In the case of the low frequency crystal (32.768 kHz), the MSP430 provides and internal selectable capacitance from 2 to 12 pF to match the load capacitance required by the crystal including parasitic bond and package capacitance(approximately 2 pF per pin) [49]. For the specific crystal model used in the board, the datasheet specifies a load capacitance of 12 pF and that's the value selected via software in the MSP430.

For the high frequency oscillator, the load capacitance need to be added with external capacitors. For the specific 32 MHz crystal used in the board, the datasheet specifies a required load capacitance of 12 pF for the particular model we selected (ref. not available). Considering 2 pF of parasitic capacitance in each pin, according to the formula above, the additional capacitors should provide 22 pF of capacitance.

$$C_1 = C_2 = 2 \cdot 12pF - 2pF = 22pF$$

The board is also provided with a circuit and connector to support the common JTAG standard to program the micro-controller and flash its memory.

Additionally, some low power consumption LEDs and buttons are connected to different pins for testing purposes. The rest of the pins are connected to an external connector in order to be able to access the pins form outside.

Radio

The radio module is a pre-built evaluation board provided by TI and bought by the company Sense.se. The reason behind was to save time in the construction and possible troubleshooting due to impedance adaptation and antenna design.

5.4.3. RFID

The RFID module is not part of the main board and it was design and constructed separately. But the external connexion was carefully designed to make the module to match properly with the micro-controller connector.

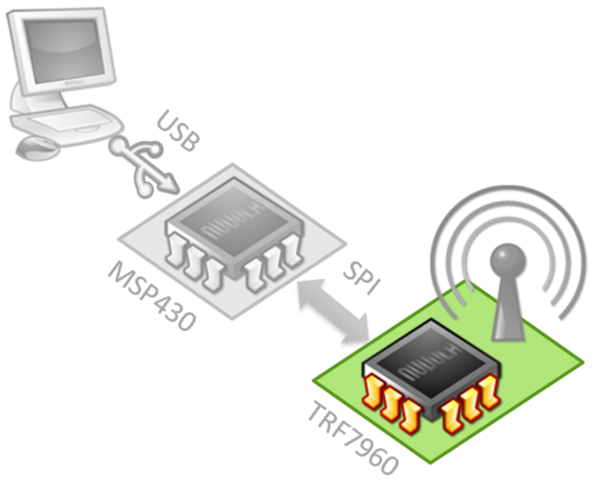


Figure 5.4: TRF7960 module placed in the RFID system

There are some pins from the micro-controller routed to output pinhead connectors for easy access to it. In the connector towards the RFID module, there are available a 4-wire *SPI* port, general *I/O* pins in case of parallel communication is desired and a pin that triggers interrupts.

TRF7960 reader

The TRF7960 chip[57] works as a framing filter and an analog front end. It receives data modulated on a 13.56MHz signal and across a matching circuit in the input/output antenna stage. This matching circuit is fixed in order to get a (real) impedance of 50 Ω towards the reader.

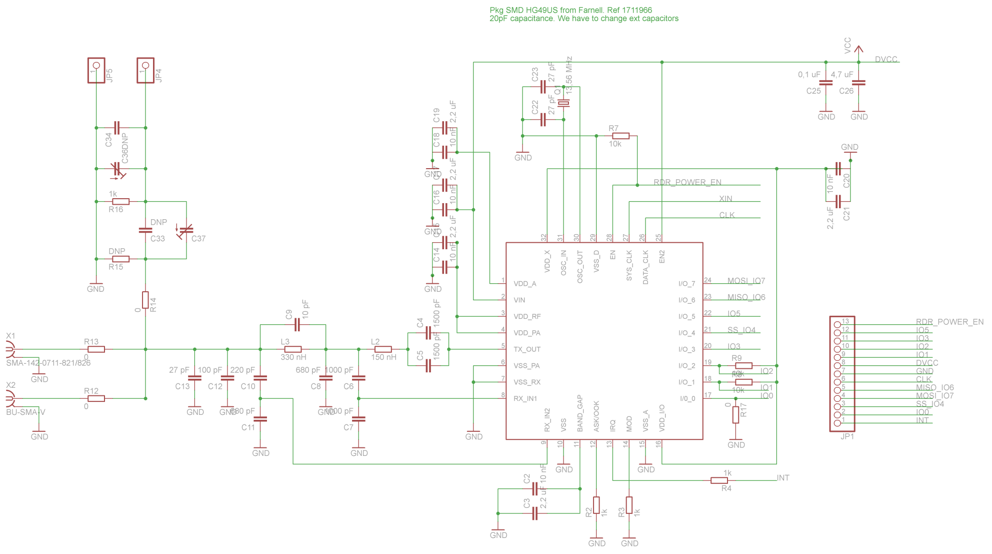


Figure 5.5: TRF7960 schematic implementing parallel and *SPI* modes

The board offers the possibility of having three different antennas attached [64]. Two of them connected to the fixed matching radio-frequency module and the other with an extra matching circuit to provide an additional degree of freedom. This simple matching circuit consists of a fixed value capacitor in parallel with a variable capacitor from 6.5 pF to 30 pF both connected to the antenna SMA connector. The scheme was followed by a parallel resistor and a group of a fixed capacitor and a parallel variable capacitor connected in series.

There is also a filter section in the circuit that split the signal into two signals with the phase shifted 45°, letting the TRF7960 reader to have two different inputs. That allows the detection of AM modulation signals on one pin and FM modulation at the other input, depending on what is received from the tag. These two signal are multiplexed in the reader under the control of registers.

One of the inputs is considered as the main receiver signal (RX_IN1 pin) meanwhile the secondary one (RX_IN2 pin) is mainly used as a signal power detector, measuring the received signal strength indicator (RSSI) of the modulation signal. The signal goes then through different stages: RF detection, gain, filtering with automatic gain control (AGC) and digitalizing in order to send it to the digital processing unit afterwards.

The TRF7960 gives the user the option of choosing different configuration (filter, gain, AGC...) by modifying register bits.

Power configuration

The TRF7960 can be fed with an operating voltage varying from 2.7 V to 5 V in the VIN pin. There are three integrated voltage regulators that can also provide an output signal from 2.7 V to 3.4 V to the different sections of the RFID board (RF output stage, analog circuits, digital *I/O* pins). The micro-controller could also be fed from this power source.

The selection of the different voltages can be configured automatically or manually in 100 mV steps. In our design we use an external voltage of 3.3 V to supply the reader and no output to the micro-controller because it is supplied independently from a regulator using the USB cable as the external source.

Pin out settings

Some pin-out set up configurations were considered in the board design. There are seven *I/O* ports that can be used in a parallel communication with the micro-controller and some of them can be also used for a serial connection via 3 pin *SPI*. This can be done by attaching V_{CC} to I/O pin 1 and pin 2 with IO pin 0 wired to GND. Data clock, enable pin, interrupt pin, V_{CC} and V_{SS} are used sharing the same pin-out. This is an advantage since it gives the possibility of choosing two different types of communication for prototyping.

There is also the option of taking the RFID clock signal from the chip to the MSP430 but this is not needed or used in this design.

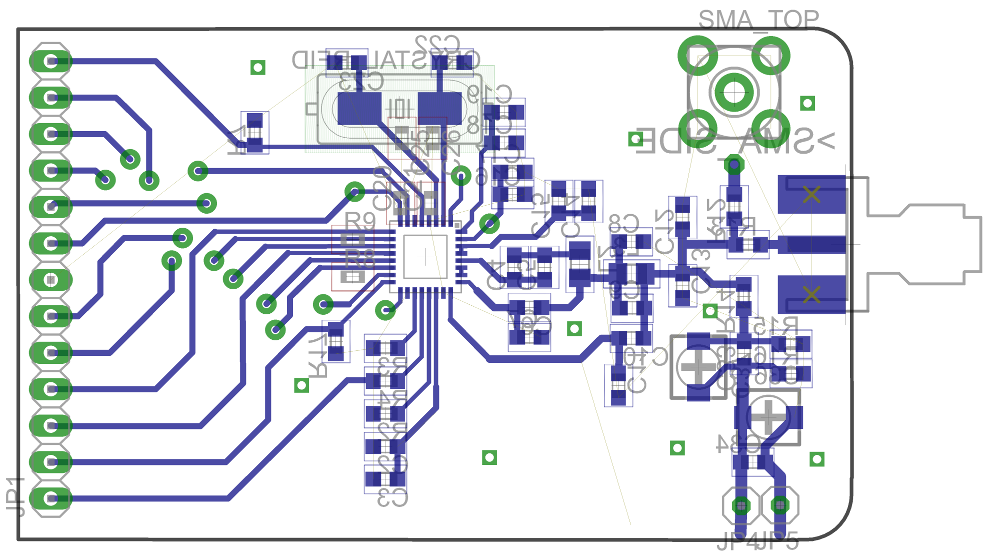


Figure 5.6: TRF7960 circuit board

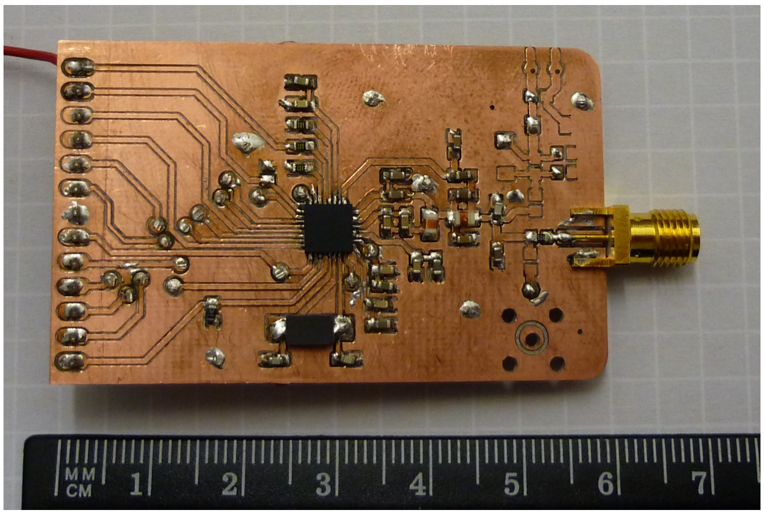


Figure 5.7: First prototype of the TRF7960 board

RFID Antenna

The antenna consists of a cable coil and a matching circuit. The problem is to find the inductance (by varying the number of turns or section of the cable) and capacity required to achieve resonance at 13.56 MHz frequency, usually using an external capacitor. Additionally, concepts such as Q-factor, bandwidth or impedance must be considered.

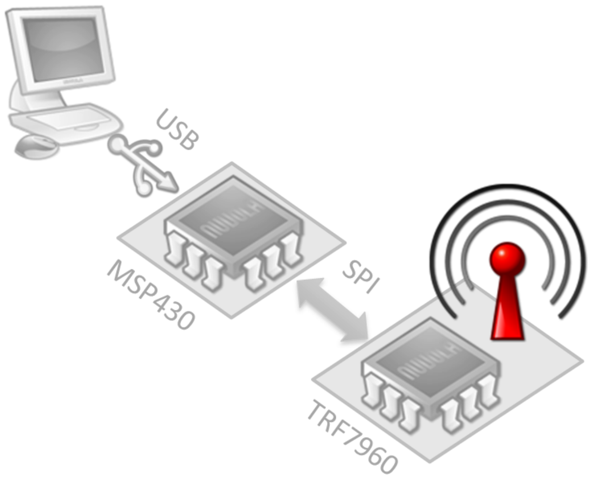


Figure 5.8: Antenna placed in the RFID system

Once the resonance frequency is achieved. The impedance of the antenna must be real (to avoid reflections and interference) and the same of the input device. That can be done from antenna theory[63] and using a Smith Chart.

Matching circuit

The impedance was measured using the impedance analyzer (RCL meter from Philips. Model PM6303A) available in the lab. The matching circuit was designed using variable capacitors. Varying the capacitance of the capacitors the impedance can be matched and the antenna can be brought to the right resonance.

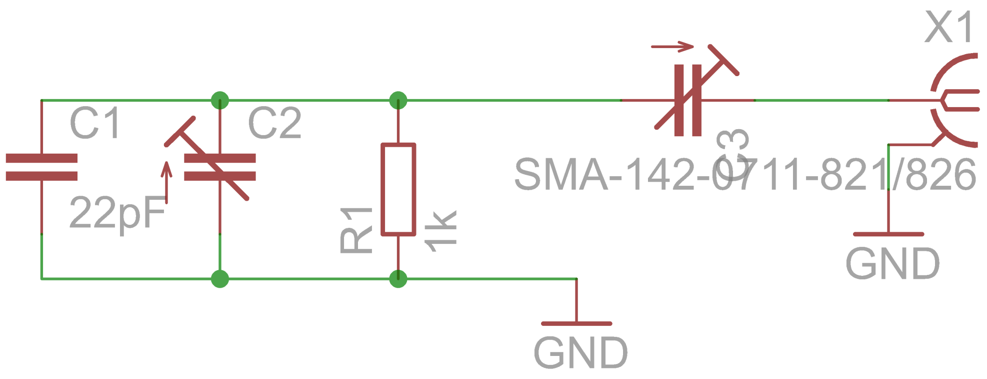


Figure 5.9: Matching circuit schematic

Three matching elements were used, and quality factor calculated for an operating bandwidth of 2 MHz considered sufficient for the prototype.

$$Q = \frac{f_0}{BW} = \frac{13,56MHz}{2MHz} = 6,78$$

$$Q = \frac{R_p}{X_L} = \frac{R}{2 \times \pi \times 13,56MHz \times 1,721\mu H} = \frac{R}{146,63}$$

Where L = 1,721 μH as we measured using the automatic RCL meter from the laboratory.

$$R_p = Q \times X_L = 6,78 \times 146,63 = 994,2\Omega \approx 1k\Omega$$

Using the evaluation Smith Chart software [83] to find values for the capacitors a fixed capacitor of 22 pF needs to be placed in parallel with a variable capacitor (range from 5 to 15 pF). The resistor has a value of 1kΩ and the variable capacitor connected in series is of a range from 10 to 120 pF.

An extra resistor of 10 Ω in parallel with the bigger-ranged capacitor was added in order to get more resistance and less reactance. Eventually, the best antenna impedance achieved was $61 + 10j$.

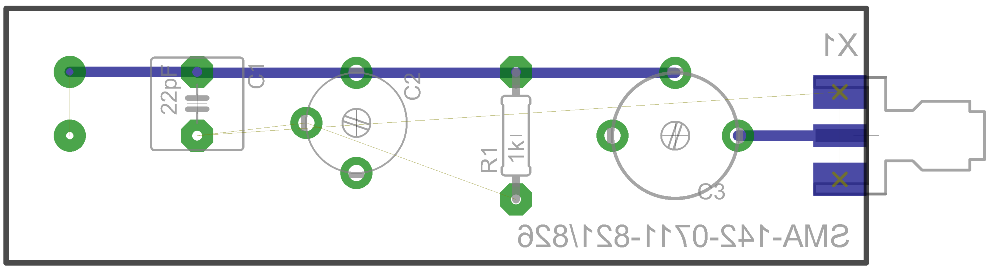


Figure 5.10: Matching circuit board

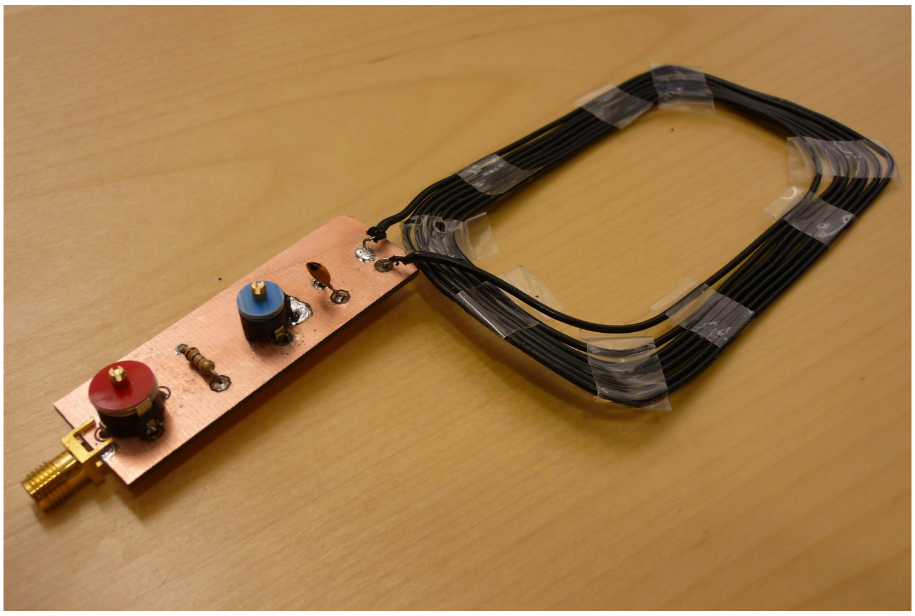


Figure 5.11: Final antenna with matching circuit attached

5.4.4. Printed Circuit Board

General considerations

There are different undesired side effects that appear in all electronic circuits between lines and components that can cause malfunctioning or unexpected behaviours in the systems. Most commonly we can find EMC, ESD, parasitic capacitance or cross talk interference.

In the boards under design in this Thesis there are RF signals of different high frequencies, and digital and clock lines carrying low currents. That makes these lines very sensitive to the effects mentioned before. In order to minimize the consequences of those disturbances, there are several design rules and good practices that should be followed. For the boards in this projects these are some of the rules followed, based mainly on recommendations from vendors [61], [66], [49].

1. Two decoupling capacitors were mounted close to the chips in every power feeding tracks, with the high frequency decoupling cap (10 nF) closer than the low frequency decoupling capacitor (2.2 μF). Keep all decoupling caps as close to the IC as possible

2. Added some ground vias between gorund planes to reduce possible ground loops or avoid isolation from ground of the components.

3. Multiple ground sections forming islands should be avoided. There should be a ground plane, possibly separating the digital and analog sections.

4. Try to place inductors affecting RF signals oriented at 90° to each other. To minimize coupling between them.

5. Crossing digital data or control lines from both sides of the board might be also a problematic. Try to avoid that.

6. Power track can be thicker than RF and digital lines. Avoid loops and 90^o corners.

7. Using SMD type components to improve resistance to interference since no holes are needed and much tinier sizes and integration can be achieved.

8. Track lengths should be as short as possible. Especially regarding RF lines, crystal connections and digital lines from the reader to the microprocessor. Proper placement of the reader, microprocessor, crystal and RF connection/connector will help facilitate this.

9. Avoid the following:

   a) Crossing of digital lines under analog and RF signal lines.

   b) Crossing of digital lines with other digital lines whenever possible.

   c) If the crossings are unavoidable, 90° crossings should be used to minimize coupling of the lines.

10. Make distance between crystal, load capacitors and micro-controller as short as possible to minimize the loop area around the combination of the components. This will decrease the effect of the resonant current. Moreover, the current between the oscillator and the microcontroler is very small (less than 1 μ A according to datasheets [49]), if signal lines are long it makes the oscillator very sensitive to EMC, ESD, parasitic capacitance and cross talk interference.

11. Keep digital signal lines, clock lines or switching signals far from the crystal

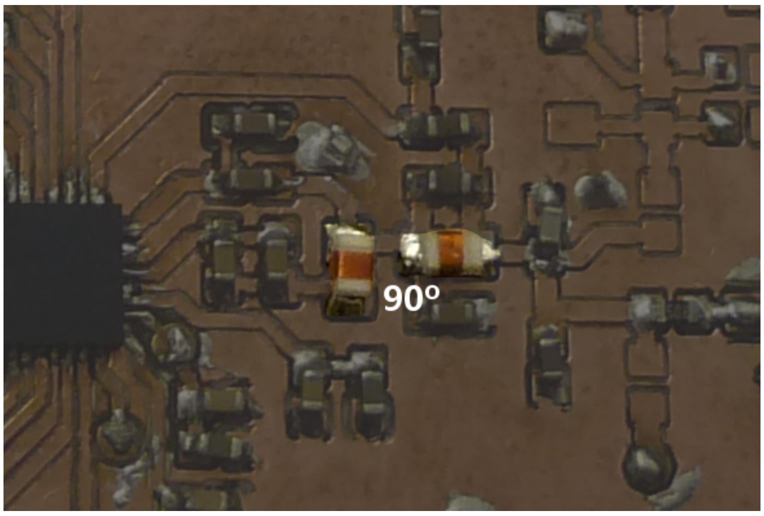


Figure 5.12: Pair of inductors at 90° orientations with respect to each other

5.4.5. Printing the board

After following the layout recommendations and adding optimizations, the gateway and the RFID boards they were finally printed using the milling machine available in the laboratory. The electrical components were soldered afterwards using a microscope and special soldering micro probes due to the small SMD package of most of the components.

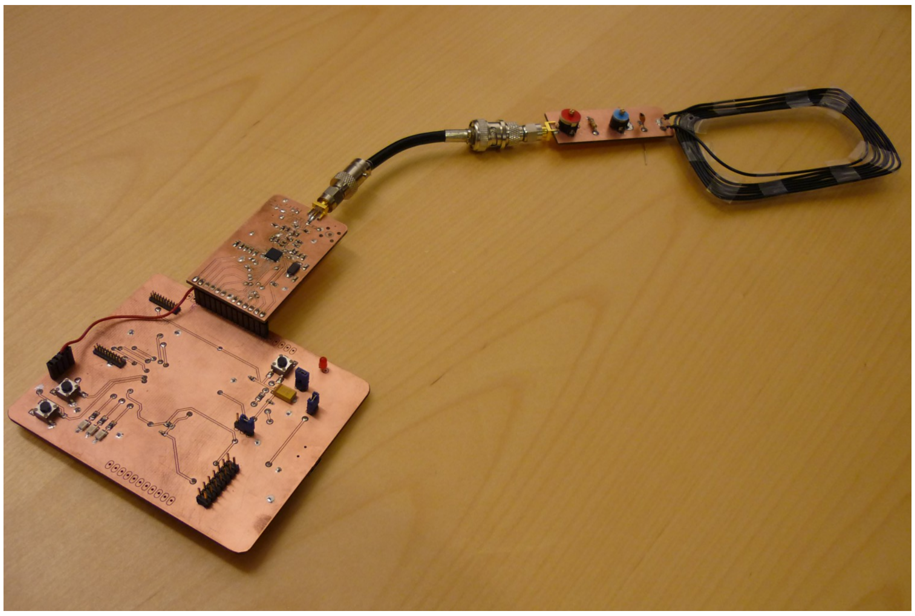


Figure 5.13: MSP430 board attached to the TRF7960 board and the RFID antenna

Chapter 6

Implementation of the platform

This chapter describes the platform from a software architecture and implementation perspective. In section 4.2.5 Contiki SO is briefly explained together with the API. Section 6.5 depicts the driver implementation of the different modules that are part of the platform. Finally, the application, the bootstrap process and the security issues are documented in section 6.7.

6.1. Contiki

As mentioned in previous chapters, Contiki is an open source operating system specially oriented for constrained devices and low power wireless networks. It was created by Adam Dunkenls at SICS, Sweden in 2002 and it is continuously developed for different hardware platforms by companies world-wide.

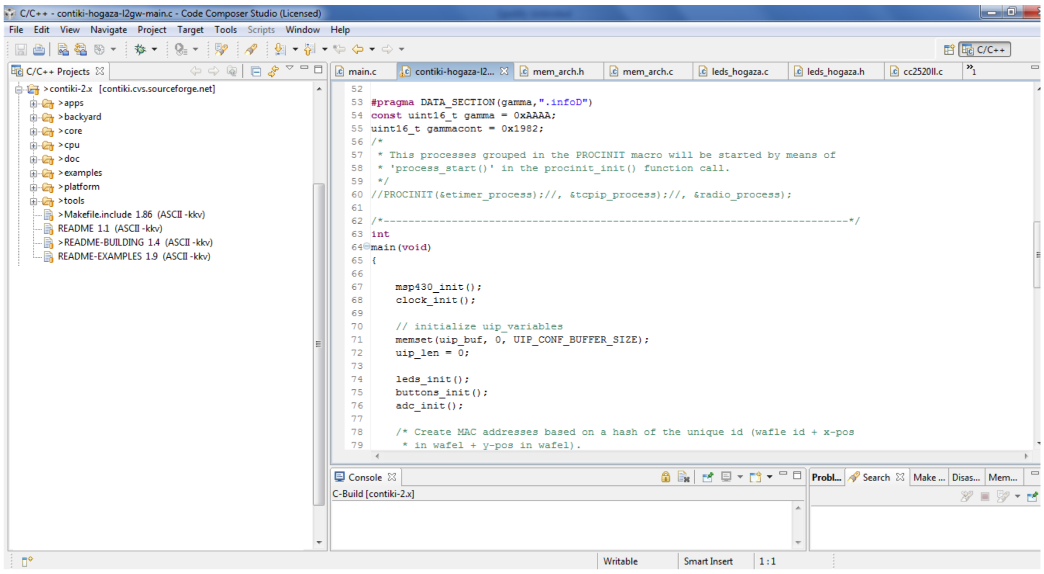
Contiki can be freely downloaded from the website [40] under a BSD-type license. The code in this Thesis is based on Contiki 2.4 version (16 February 2010). The development IDE tool was Code Composer Studio Core Edition Version: 4.1.3.00034 from Texas Instruments, similar to the popular Eclipse IDE.

6.1.1. Directory tree

Contiki is delivered in a compressed file containing several directories with C code and header files, build files and different documentation. The folders in the root directory are classified as follows:

- **apps/** Several applications that can be used in this version of Contiki. It includes the implementation files of a dhcp client and a server, a web browser, a web server, the implementation of ping6 command, a telnet client, a shell and a small implementation of client and server of the CoAP protocol for constrained devices.

- **core/** Here is where the system core files of Contiki are stored. It contains subdirectories with drivers for different radio modules, serial communication ports, memory


C/C++ - contiki-hogaza-l2gw-main.c - Code Composer Studio (Licensed)
File Edit View Navigate Project Target Tools Scripts Window Help
C/C++ Projects
>contiki-2.x [contiki.cvs.sourceforge.net]
>apps
>backyard
>core
>cpu
>doc
>examples
>platform
>tools
>Makefile.include 1.86 (ASCII -kkv)
README 1.1 (ASCII -kkv)
>README-BUILDING 1.4 (ASCII -kkv)
README-EXAMPLES 1.9 (ASCII -kkv)
main.c
contiki-hogaza-l2...
mem_arch.h
mem_arch.c
leds_hogaza.c
leds_hogaza.h
cc2520ll.c
Console
C-Build [contiki-2.x]
Probl...
Search
Make ...
Disas...
Mem...
Writable
Smart Insert
1 : 1

```

#pragma DATA_SECTION(gamma,".infoD")
const uint16_t gamma = 0xAAAA;
uint16_t gammacont = 0x1982;
/*
 * This processes grouped in the PROCINIT macro will be started by means of
 * 'process_start()' in the procinit_init() function call.
 */
//PROCINIT(&etimer_process);//, &tcpip_process);//, &radio_process);

/*---------------------------------------------------------------------------*/
int
main(void)
{

    msp430_init();
    clock_init();

    // initialize uip_variables
    memset(uip_buf, 0, UIP_CONF_BUFFER_SIZE);
    uip_len = 0;

    leds_init();
    buttons_init();
    adc_init();

    /* Create MAC addresses based on a hash of the unique id (wafle id + x-pos
     * in wafel + y-pos in wafel).
```

Figure 6.1: Code Composer IDE.

and power management, watchdog control, binary loader, etc. Under the **net** subdirectory there are code files for the uIP TCP/IP stack, the uIPv6 stack, 6LoWPan and the Rime stack for network communication. Also the implementation of different MAC protocols(mac, nullmac, sicslowmac, xmac, contikimac) and 802.15.4 framers. All Contiki system files implementing the protothreads, process init, timers and so on are stored under the **sys** folder.

- **cpu/** Includes low level implementation for porting to different microprocessor architectures. For instance, arm, avr, cc2430, msp430, native or x86.
- **doc/** Contains Contiki's Doxygen documentation files.
- **examples/** With several subdirectories that include examples of the use of the applications in different platforms.
- **platform/** It contains files in sub-folders for porting to many different platforms and computers such as Atari, Avr-Raven, Cooja, Esb, Micaz, Netsim, Redbee, Sesinode, Sky...
- **tools/** Under this directory there are a few specific tools for use with the platforms above. There are compilers, configuration files, libraries, scripts, etc.

Not all the files available in the package are needed from the original Contiki project. For the purposes of this Thesis the most useful files are the Contiki system core files and some platform and hardware specific files to facilitate help on the porting to the *Torrija* board. The examples and application files are interesting as a base for the rest of the development and learning of the different parts of Contiki.

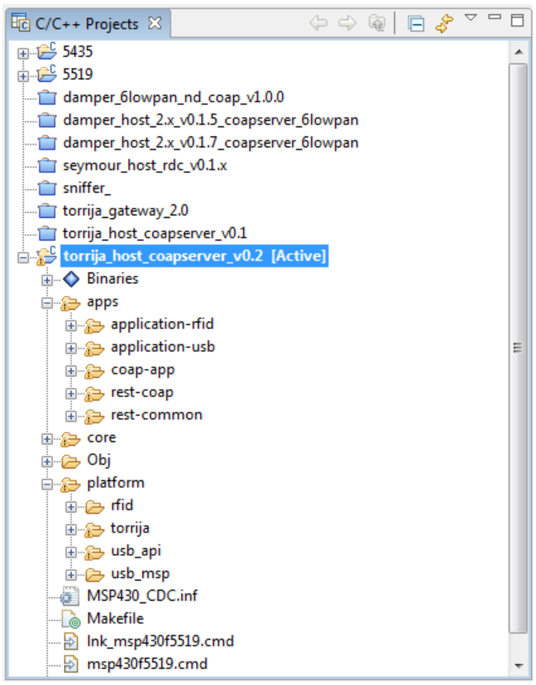

```
C/C++ Projects
+ 5435
+ 5519
  damper_6lowpan_nd_coap_v1.0.0
  damper_host_2.x_v0.1.5_coapserver_6lowpan
  damper_host_2.x_v0.1.7_coapserver_6lowpan
  seymour_host_rdc_v0.1.x
  sniffer_
  torrija_gateway_2.0
  torrija_host_coapserver_v0.1
- torrija_host_coapserver_v0.2 [Active]
    + Binaries
    - apps
        + application-rfid
        + application-usb
        + coap-app
        + rest-coap
        + rest-common
    + core
    + Obj
    - platform
        + rfid
        + torrija
        + usb_api
        + usb_msp
      MSP430_CDC.inf
      Makefile
      lnk_msp430f5519.cmd
      msp430f5519.cmd
```

Figure 6.2: Code Composer Torrija project view.

The list of project directories of the complete platform is shown below.

- **apps/** (containing the RFID, USB and CoAP applications)
  - application-rfid/
  - application-usb/
  - application-coap/
  - rest-coap/
  - rest-common/
- **core/** (containing Contiki's system and network files)
  - cfs/
  - ctk/
  - dev/
  - lib/

- loader/
- net/
- sys/

- **platform/** (drivers for the GW board and the RFID and USB modules)
  - rfid/
  - torrija/
  - usb/

6.1.2. Protothreads

Protothreads introduce a new programming abstraction that helps to implement event-driven systems in a thread-like environment. It combines the benefits of threads in terms of flow control and code structure with the low memory overhead of event-driven programming implemented with state machines [46]. They were developed for Contiki by Adam Dunkels and Oliver Schmidt [42] at SICS.

Protothreads are specially designed for memory-constrained devices providing a very lightweight mechanism for thread handling that requires minimum memory resources (two bytes on the MSP430 [46]). In order to achieve good memory efficiency protothreads do not use individual stacks among threads, but only one shared stack. Context switching is possible but it can only occur after controlled blocking operations. Global and static variables are used to preserve information during the context switching since automatic variables loose their content after returning from the blocking protothread so the design must be done with care with re-entrant function.

Commonly event-driven programming style is based on state machines to build applications what could make programs difficult to implement, debug or understand. Protothreads simplify and replace the state machine model with a sequential model based on a conditional blocking call: `PT_WAIT_UNTIL (condition)`. That makes the program more intuitive and easy to interpret since it follows the natural flow of the process. It also reduces drastically the number of lines of code needed for the implementation [46].

Protothreads are implemented through a few simple statements. They must start with `PT_BEGING()` and end with `PT_END()` statements, but can exist prematurely by means of `PT_EXIT()`. `PT_WAIT_UNTIL()` can block so the programmer is always aware of the potentially blocking calls. Unconditional blocking wait is provided with the statement `PT_YIELD()` that blocks until the next invocation of the process. In addition, `PT_SPAWN()` starts a child of the process and blocks the parent until the child has ended. There are a few more statements that are documented in Contiki's API [41]. In Figure 6.3 there is an implementation of a basic example using protothreads.

There are two important limitations when using protothreads. One has already being mentioned referring to the fact that local variables loose their content across the blocking wait. This issue can be solved by using static variables. It is also inconvenient the

```
PT_THREAD(struct pt *pt) {
  PT_BEGIN(pt);
  PT_WAIT_UNTIL(pt, condition1);
  if(something) {
    PT_WAIT_UNTIL(pt, condition2);
  }
  PT_END(pt);
}
```

Figure 6.3: Example of protothreads

constraints on the usage of switch statements within protothreads. Since the protothreads implementation hides a C switch statement behind their macros, the introduction of a switch statement inside may result in an undefined behaviour.

6.1.3. Processes

Any program running in Contiki is also a process. Processes are usually started at boot up time or when a module is loaded in the system. Processes in Contiki are implemented and scheduled by means of protothreads. Each process can only have one protothread associated when executing and it should contain the code of the process. Contiki also provides a simple API in C for creating and managing processes [41]. This API consists of a set of macros and functions that are mentioned below in this section. Their implementation is located at *core/sys/process.c* file.

A Contiki process is defined by two parts: a **process control block** and a **process thread**. In figure 6.4 there is an example of a basic program implemented with Contiki's processes. The first line corresponds to the process control block. This macro, `PROCESS (name, str_name)`, with two arguments declares and internal structure that contains information about the process such as the state, a pointer to a thread, a pointer to the next process in the list of processes, etc. The first argument gives name the process structure variable. The second argument is the name of the process in string format.

The protothread is declared in the next line using the macro `PROCESS_THREAD (name, ev, data)` with three arguments where `name` is the name of the process structure variable, `ev` is the identifier event that has caused the invocation of the thread and `data` is a pointer to additional date passed to the thread during the invocation.

There are still two more macros that define when the process starts and when it exits as it can be seen in 6.4. Both macros are mandatory in every process, `PROCESS_BEGIN()` should go right after the declaration of the thread (any code placed in between will always be executed when re-executing the process) and `PROCESS_END()` at the exit point of the process. The application code is implemented in between them.

In this example, the process prints out a message and after that it enters into an infinite loop. Inside the loop it calls the macro `PROCESS_WAIT_EVENT()` that will block the process until an event is received. The kernel gets back the control and gives service to other processes. If an event occurs, the kernel delivers it to the process (with and event id and possibly some data) so it can continue the execution from the point where it stopped the last time.

```
PROCESS(hello_process, "Hello␣process");

PROCESS_THREAD(hello_process, ev, data) {
  PROCESS_BEGIN();
  printf("Hello␣Gateway!\n");
  while(1) {
    PROCESS_WAIT_EVENT();
  }
  PROCESS_END();
}
```

Figure 6.4: Example of processes

There are several more macros that are available in Contiki's API that helps managing processes and events. A full list can be obtained from the official Contiki documentation of the API [41]:

`PROCESS_WAIT_EVENT()`:
Wait until an event is received.

`PROCESS_WAIT_EVENT_UNTIL(c)`:
Wait until an event is received and a condition.

`PROCESS_YIELD()`:
Yields the current process.

`PROCESS_YIELD_UNTIL(c)`:
Yield the current process until a condition occurs.

`PROCESS_WAIT_UNTIL(c)`:
Wait for a condition to occur.

`PROCESS_PAUSE()`:
Pauses (yields) the process for a short time.

`PROCESS_POLLHANDLER(handler)`:
Specify an action when a process is polled.

`PROCESS_EXITHANDLER(handler)`:
: Specify an action when a process exits.

Aditionally, a set of functions are provided to help with the development of the application [41]:

`void process_start(struct process p, const char arg)`
: Start a process.

`void process_exit(struct process p)`
: Make a process to exit.

`int process_post(struct process p, process_event_t ev, void data)`
: Post an asynchronous event to a process.

`void process_post_synch(struct process p, process_event_t ev, void data)` Post
: a synchronous event to a process.

`void process_poll(struct process p)`
: Request a process to be polled. A `PROCESS_EVENT_POLL` event will be sent to the process.

Events

Events in Contiki determine the interactions and behaviour of the applications and the kernel. They give life to the system. Processes are started, paused, restarted or stopped by means of events. They can either be sent by another process, the kernel or the special case of drivers that use a polling method.

Events can be classified as asynchronous or synchronous depending on how the event is delivered in time. Asynchronous events are kept in the system's event queue and delivered whenever the scheduler finds the right time slot for it. Events are dispatched in the same order as in an FIFO queue. Asynchronous events can be posted to single processes or broadcasted using the function `int process_post(struct process p, process_event_t ev, void data)`. Since they are scheduled for a later delivery, this type of events require specific parameters to identify the type of event and the receiving process. It is also possible to pass data between processes by means of a data pointer in the arguments, since processes share the same memory space.

Contrarily, synchronous event are delivered immediately. The receiving process is then directly scheduled and the sending process pauses until the receiving process has finished. Synchronous event can only be sent to specific processes with the function `void process_post_synch(struct process p, process_event_t ev, void data)`. It must not be called from an interrupt context.

Polling

It is a special case of communication with events associated principally with interrupts handlers. In order to avoid interrupt routines locking and race conditions Contiki does not send events form the interrupt space. Instead, processes must be polled from the interrupt context by means of `void process_poll(struct process p)`. This function sets a polling flag in the process structure when called within the interrupt handler when a hardware event occurs. Polled processes have the highest priority. When the scheduler is executing, it first goes through the list of polled processes. The kernel then sends them a special `PROCESS_EVENT_POLL` event type, so the processes know that they can operate on received data. When there are no more polled processes to be processes, the scheduler runs normally processing the events in the FIFO event queue.

6.1.4. Timers

Another essential feature in Contiki is the implementation of timers. They are be used by applications and by the system. Timers help to control tasks periodically as, for instance, collecting data from sensor, reading I/O buffers, waking up the CPU from sleeping mode, etc. Their implementation is located under *core/sys/* directory.

Contiki provides a lightweight timer library as a base on which timers are built. There is a range of functions that help to set, reset and restart a timer or to evaluate if the timer is expired for instance. There is a set of timers available in Contiki for different purposes provided by the clock module:

- *timer*: Simple timer that only keeps track of its expiration time. It is mostly used for short periods, as it counts clock ticks.
- *stimer*: This timer only keeps track of its expiration time but it is used for long periods since it counts seconds.
- *etimer*: The event timer sends an event to a process when it expires. Used to schedule events in Contiki.
- *ctimer*: The callback timer schedules calls to functions when it expires. Typically used by protocols such as Rime.
- *rtimer*: Real-time timer, calls a function at an exact time pre-empting processes if necessary in order to execute the task on time. It is used when time is critical.

Applications commonly use tick-based timers, second-based timers and event timers for scheduling processes. Basic timers and etimers can be safely used from interrupt contexts. Event timers are useful to let the CPU sleep for a certain time or by a process to yield the execution time to other processes and tasks. A system event timer process (etimer process), based on the event timer library, periodically evaluates every timer that has been set and posts an event (`PROCESS_EVENT_TIMER`) to the process that set the timer if it has expired.

As an example, below is a small part of typical code using events. The event structure must be declared first, before the execution block in this case. After the `PROCESS_BEGIN()` macro the process enters into an infinite loop, in each iteration it sets the duration of the event timer using the library function `&etimer_set(et, CLOCK_SECOND)` where `CLOCK_SECOND` variable is the time in seconds and it is defined somewhere else. After the timer is set the macro `PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et))` is called. This macro will tell the process to pause until an event is received, so the control is given to the system and other processes. When the event timer expires it sends an event (`PROCESS_EVENT_TIMER`) to the process so it can continue the execution the next time it is scheduled.

```
PROCESS_THREAD(hello_world_process, ev, data) {
  static struct etimer et;
  PROCESS_BEGIN();
  while(1) {
    etimer_set(&et, CLOCK_SECOND);
    PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
  }
  PROCESS_END();
}
```

6.1.5. Networking

Contiki provides two network stacks implementations: the uIP TCP/IP stack providing IPv4 and IPv6 certified connectivity and the more lightweight Rime stack, specifically designed for constrained low-power networks [40].

Rime stack

Rime implementation contains lightweight protocols suitable for low-power sensor networks where the uIP stack becomes too heavy. Rime provides a set of basic communication primitives useful for the design of low-power networks. These default primitives provide different features varying from best-effort local broadcast to reliable network flooding and address-free data collection [44].

Rime stack is arranged in a way that communication primitives can be combined to create more complex protocols. Rime stack is distributed in layers thinner than traditional architectures such as the TCP/IP stack [41] in order to simplify the implementation.

The code size of Rime is around two kilobytes and RAM requirements are on the order of tens of bytes [44].

uIP stack

The uIP stack provided by Contiki is a small TCP/IP compliant stack intended for embedded systems using 8-bit or 16-bit micro-controllers. It does not include the full set of protocols and features of a fully compliant TCP/IP stack, but it includes a set of basic communication features such as RFC compliant versions of the TCP, UDP and IP protocols, as well as the mandatory maintenance protocol ICMP, the RPL routing protocol for low-power IPv6 networks, the 6LoWPAN header compression mechanism and adaptation layers for the standard IEEE 802.15.4 compliant radios [43].

Th uIP stack optimizes code size by eliminating some uncommon mechanisms. For instance, mechanisms in the interface between the application and the stack that very few applications use. However, it is important to mention that all host-to-host communication mechanisms are indeed implemented [41].

Among other optimizations, uIP also economizes memory by not using explicit dynamic memory allocation, using only one packet buffer and having a fixed array working as a table where connections are held [41] [43].

All the communication capabilites provided by Contiki can be accessed by means of specific processes or as library functions part of the different modules of the architecture.

- **MAC:** Contiki includes implementations of different MAC layers for low level communication such as *nullmac*, *xmac*, *lpp* or *sicslowmac* to sent and receive packets over IEEE 802.15.4 radio based networks.

- **6LoWPAN:** Contiki's 6LoWPAN implementation used in this Thesis supports different encapsulation and header compression mechanisms to carry IPv6 datagrams over the constrained radio links. It supports basically stateless header compression LOWPAN HC1 and LOWPAN HC2 defined in RFC 4944 [37] and later by RFC 6282 [38] but also a more advanced LOWPAN IPHC based on sharing state within contexts [34]

- **IPv4:** Contiki provides a reduced but standard-compliant IPv4 stack with UDP, TCP and ICMP support running on a protothread. Additionally also DHCP and ARP protocols are implemented on separated modules with DHCP running on a dedicated protothread and ARP defined as a set of functions, both easily accessible.

- **IPv6:** The IPv6 stack is also running on a protothread and supports UDP, TCP and ICMPv6 protocols. It provides Neighbor Discovery for IPv6 [36] that will be used to discover nodes in the proximity of the network (with the same purpose as ARP does for IPv4).

- **Socket interface:** A light weight socket API is provided in order to manage UDP and TCP connections, requiring specific information such as source address and local port in case of UDP or destination address and remote port in case of TCP. We are basically only using UDP sockets in our application.

- **CoAP:** Event though CoAP does not really belong to the network modules since it is part of the application layer, but it is important to mention that the communication towards lower layers is based on UDP sockets. CoAP uses a client-server model for the remote communication. Contiki 2.5 implements a basic version of the CoAP protocol (draft-ietf-core-coap-04) including the client-server mechanisms and a basic application to use as a base to our implementation.

6.2. Overview of the platform

Contiki is a hardware independent software platform that can provide functionality to different platform but it does not intrinsically support specific hardware. In the current distribution of Contiki there are some *ports* available from different projects. In this sense, *porting* code to a hardware platform means that a set of basic functions have been provided to Contiki so it can make use of the specific hardware functionality. Those are the drivers that are developed in this Thesis for the specific platform requirements.

The development of the WSN application consists of a CoAP client that needs to be installed in the gateway and send requests to the sensor boards. Reciprocally these nodes nodes will act as CoAP servers sending responses upon those requests. Even though the sensor boards are specifically designed in close collaboration for another Thesis [3] work, it will be useful to also describe the application requirements for the nodes, since their implementation will affect to the overall design.

Gateway

Contiki is the OS managing the hardware resources of the board and holding the applications that make use of them. In order to do that the designer needs to provide the functionally for the control of that specific hardware. In the case of the gateway some elements can be easily identified:

- One of the key components when porting a new hardware to Contiki is the clock module that timers are based on. This module requires the selection of a clock source and a defined constant (`CLOCK_CONF_SECOND`) specifying the duration of a second in system ticks. This value can be inferred from the value of the selected clock signal and the CPU system clock, both depending on the value of external oscillators and the settings of the microprocessor. Additionally, a set of functions has to be provided to initialize the module and handle the counter of system ticks.
- To make use of the network stack and establish communication between nodes some functions need to be provided to Contiki on different network layers. These set of functions are grouped by layers into C structs (one struct of functions per layer) that get the name of *drivers*. The "NETSTACK" module (Contiki's network stack) is informed of which driver is associated to each layer by means of a set of C macros. This is how the stack is configured. Each layer can be configured with different interchangeable drivers, depending on the desired selection. For instance, the MAC layer can be configured with the csma or nullmac drivers, the RDC layer with the xmac, lpp or sicslowmac drivers or the FRAMER with the framer_802154 or framer_nullmac,

all of them implemented and documented in Contiki [40]. Below is an real example of how the different layer of the network stack are assigned their respective drivers (defined somewhere else).

```
#define NETSTACK_CONF_NETWORK   sicslowpan_driver
#define NETSTACK_CONF_MAC       nullmac_driver
#define NETSTACK_CONF_RDC       sicslowmac_driver
#define NETSTACK_CONF_FRAMER    framer_802154
#define NETSTACK_CONF_RADIO     cc2520_driver
```

- All the drivers for the netstack are implemented in Contiki except the radio driver since it is dependant of the hardware. We provide a set to functions for the `radio_driver structure` defined in Contiki (called `cc2520_driver` for the TI CC2520 radio). A Contiki process is then polled periodically by the interrupt routine when an incoming packet has been received and need to be read, it will then be delivered to the stack for processing. On the other hand, in case a frame needs to be sent out, the specific IP stack process will handle it by directly calling the sending function from the library, without the need of waking up the radio process.

  Other hardware-specific concerns such as low-power modes, duty cycles, transmission power, system clock or communication interface towards the microprocessor (e.g parallel or serial) for the CC2520 chip are described in section 6.5 or the documentation [53] [54].

- The different modules (clock, network stack, applications...) in Contiki OS are configured through a set of macros and headers in the file `contiki-conf.h` under `/platform` directory. Here the designer should include those compiler and hardware specific header files, select the CPU frequency, define data types, select the applications, drivers, and many other options for the configuration of the network, optimizations, etc.

- Specific REST CoAP messages and commands can be sent to the gateway via the USB port to communicate with nodes and configure the gateway. This is not a Contiki requirement but an application specific issue. The user must be able to interact with the network in order to authorise new nodes, send requests to the nodes via the CoAP client, or get the sensor data in a specific format.

  The microprocessor selected for the gateway is an MSP430F5519 from Texas Instruments that integrates a USB module so it makes somehow easier and more efficient the development of the driver since no extra hardware is required. The USB stack is developed and freely provided by Texas Instruments, but it needs to be ported to the specific microprocessor model and integrated into the Contiki system. The data received by the USB interface should be processed by a Contiki protothread whenever the buffer is filled and available. The data must be parsed as command-line arguments (see section 6.7), generating error messages for incorrect inputs and showing the available options. When sensor information needs to be sent out, the application should make use of the USB available functions to deliver the data externally. All this functionality needs to be developed.

- The TRF7960 from TI is an RFID data-framer reader that will read tags from the nodes in order to authorise and include them in the network as part of the security application. This is not a requirement from the Contiki SO perspective but again it is part of the overall gateway functionality. It supports different protocols as mentioned in Section 5.1.4 but only **Tag-it** protocol by Texas Instruments is enabled and implemented since it is the only tag available in the Wireless@KTH lab [1] for testing purposes.

- The data received by the RFID can be handled in different ways. The preliminary proposal for the security application is that the recived data should coincide with the MAC address of the sensor node device. The gateway should then store all the MAC addresses in a internal secure table. Neighbor Discovery protocol should only accept those learned addresses that have an entry in that table. It can be stored in a dedicated sector of the flash memory of the MSP430 so it can be permanently accessible even after restart of the gateway device.

  Another use of the RFID data can be the use of it as a secret used for the data encryption. This can be easily performed in a link layer level using the AES 128bit encryption for confidentiality option provided by the CC2520 radio hardware.

- Other useful drivers that need some implementation comprise a set of LED for testing, debugging and information purposes, two buttons also for debugging and testing purposes that are not used in the current version of the implementation but can be useful for future more advanced purposes.

Sensors

Regarding the development of the nodes, since it belongs to a different work [3] it is only interesting to mention the parts that are needed for the CoAP server application that interacts with the client installed in the gateway or the security implementation. We have no concern of unrelated issues such as system specific settings, radio modes or similar assuming that the radios and the stacks agree on the same protocols and the communication can be established.

- First of all a number of sensors should be available on each board. It is not part of this Thesis to describe how the drivers are implemented but only to inform that they are available for the CoAP application. On the first version of the boards developed by [3] they are equipped with a temperature sensor, a light detector, a humidity sensor, a battery load meter, an accelerometer and a LED.

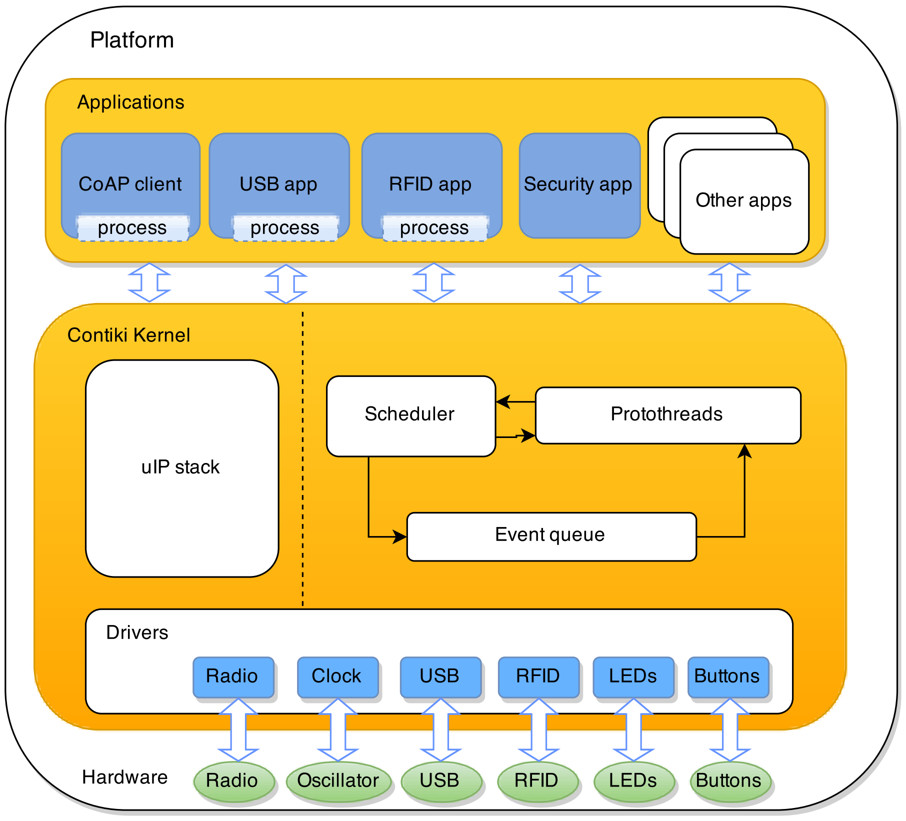


Figure 6.5: Platform architecture. The parts marked in blue have to be developed. At the bottom, the drivers needed for Contiki that communicates with the hardware modules at the low level (in green). On top, also in blue, the main applications, some of them requiring a process to poll for events and some just handling data to be stored or to be sent out to the user for instance. Some interactions such as direct communication from the applications towards driver output ports are not shown here.

- The CoAP server defines *resources* for each sensor or actuator. These resources can be simple resources accessed by request or periodic resources that continuously deliver data for a requested period. Resources are defined by a macro with three parameters: resource name, the http methods it handles and the url where it will be "located" within the node (`RESOURCE(hello, METHOD_GET, "hello")`)

  Each resource that has been defined, needs a handler method that should be defined as `[resource name]_handler`. This function must handle the requests and read data from sensors or perform an specific action. `PUT` or `POST` methods are normally used in create a resource or request an action while `GET` requests usually read data from the sensors. The handler function should finally build-up the CoAP response packet that will be sent out.

In case of periodic resources the `PERIODIC_RESOURCE` macro should be used. This time with four parameters: resource name, the http methods, the url and the period in seconds (e.g. `PERIODIC_RESOURCE(temperature, METHOD_GET, "temperature",1)`). Two functions should be provided:

- A periodic handler declared as `<resource name>_periodic_handler`.
- A request generator function that generates the payload every time the timer expires defined as `<resource name>_periodic_request_generator`.

Observable resources can also be defined using the `OBSERVABLE_RESOURCE` macro. Requests to this resource allows a number of client *observers* to "subscribe" to it. Every time a change occurs (switch toggled, sensor value goes beyond a limit...) in the resource a notification is sent to the registered observers. However this type of resource is not used in the current version of the nodes.

There is additionally a default resource known as the `".well-known/core"` resource that is defined by the CoAP protocol and is used to discover the available resources of the node. It should send the reply in a specific text format defined by the protocol [71] [73].

6.3. Development tools

Code Composer Studio v4 (based on Eclipse GUI) from Texas Instruments available at www.ti.com/msp430 was used to develop the code. There is a code limited version that was used for writing the code and a full version provided with a time limited licensed that we could extend during the debugging phase with the boards.

Regarding the third-party code in this prototype we are using SO Contiki v2.5 and the firmware provided by TI for the different modules available at www.ti.com. For the radio it is cc2520 example code (swrc090a), the RFID module uses a TRF7960 Firmware Source Code for CCS (sloc203) and the USB CDCv1.19 stack for Code Composer v4.

For debugging the application on the MSP430 we used a JTAG debug interface from Texas Instruments (MSP430-FET430UIF) and Code Composer Studio v4. As mentioned in Chapter 5 EAGLE in version 6 was used for the design of the board layouts and schematics developed by CadSoft Computer (www.cadsoftusa.com).

Windows XP SP3 was used as the OS even though Linux was preferred. However, at the time of the design there was no free open source available toolchain for the MSP430F5519 microprocessor. The open *msp-gcc* compiler and debugger tool for MSP430 micro-controllers did not support family 5 at that time. For other purposes, e.g. testing the USB interface, we used the Ubuntu 12.04 Linux distribution.

6.4. Contiki configuration

Contiki can be configured via a set of directives selecting the hardware, enabling and disabling different modules and different features related to the network or system specific. This set is grouped inside `contiki-conf.h` that is located under `/platform` directory and it is needed by the compiler to generate the application. The basic functionality is configured in Contiki as shown below. For a complete version of `contiki-conf.h` see Apendix B.

- Specific compiler data types and hardware specific header files must be specified

```
#include <msp430f5519.h>
#include <inttypes.h>
```

- Other CPU related and compiler options are selected by means of macros.
- Specify Contiki specific options and data types.
- Select the application

```
#define WITH_COAP       1
```

- Select IP version 6

```
#define WITH_UIP6       1
```

- Use rime addresses to take advantage of the address handling functions provided by Rime.

```
#define RIMEADDR_CONF_SIZE      8
```

- Select link layer as IEEE 802.15.4

```
#define UIP_CONF_LL_802154      1
```

- NetStack definitions as it was specified in the overview of the platform
- IPv6 options are set by default.
- Transport layer enabling only UDP support (for CoAP)

```
#define UIP_CONF_UDP    1
```

- Select context-aware type of 6LoWPAN compression (IPHC):
- No fragmentation
- Select buffer size: 110 longest IEEE802154 payload + 40 IPv6 header + 8 UDP header

```
#define UIP_CONF_BUFFER_SIZE    110 + 40 + 8
```

- Node's MAC address

```
#define NODE_BASE_ADDR0 0x00
#define NODE_BASE_ADDR1 0x07
#define NODE_BASE_ADDR2 0x62
#define NODE_BASE_ADDR3 0xff
#define NODE_BASE_ADDR4 0xfe
```

There might be other configuration options residing in other configuration files such as `contiki-net.h` or `coap-conf.h` that could be interesting for the developer. However the default configurations are good enough for this application so we do not see the need for modifications.

6.5. Drivers

It is not the the purpose of this section to explain in detail the low level firmware of the different modules in the platform. It is aimed to be used a guide for the interested reader to start exploring the code.

6.5.1. Micro-controller driver

The MSP430F5519 micro-controller provides a large amount of interesting features. Some of them must be understood because they are needed in the platform and some code firmware must be implemented. Normally the modules firmware provides sufficient libraries of functions for the control of the devices. However normally the code need to be ported to a particular platform, adapted to the hardware requirements or the desired configuration in terms of clock frequency or power mode for example. Or one could just use some extra features of the micro-controller such as writing to flash memory, selecting different clock sources or enabling an interrupt on a specific port. Here below is a list of some of the basic features:

- System reset and initialization.
- Low power operation modes (LPM0 to LPM4).
- Interrupt handling.
- Configuration of I/O pins and internal registers.
- Unified Clock System (UCS) that provide various clocks for the micro-controller.
- Operation of the flash memory controller to be able to write on permanent memory.
- Configure timers with multiple capture/compare for interval timing and with interrupt capabilities.

- Control of Serial Communication Interfaces: UART, SPI or I2C.
- Configure ports for parallel communication (towards RFID module).
- Control operation and transfers for the USB module.

If the reader wants to get deeper information the MPS430 datasheet [49] and User's Guide [50] are available at TI website (www.ti.com/msp430). He can also refer to the code implementation for examples (partly in Appendix B).

6.5.2. Radio driver

In this Thesis we are using a CC2520 radio from Texas Instruments and a set of functions specified by the structure of a generic radio driver have to be provided. The specific driver (C struct) for the specific radio chip has been named `cc2520_driver` and it provides the functions required by the `radio_driver` struct prototype defined in Contiki.

```
  struct radio_driver {
  int (* init)(void);
  /** Prepare the radio with a packet to be sent. */
  int (* prepare)(const void *payload, unsigned short payload_len);
  /** Send the packet that has previously been prepared. */
  int (* transmit)(unsigned short transmit_len);
  /** Prepare & transmit a packet. */
  int (* send)(const void *payload, unsigned short payload_len);
  /** Read a received packet into a buffer. */
  int (* read)(void *buf, unsigned short buf_len);
  /** Perform a Clear-Channel Assessment (CCA) to find out if there is
  a packet in the air or not. */
  int (* channel_clear)(void);
  /** Check if the radio driver is currently receiving a packet */
  int (* receiving_packet)(void);
  /** Check if the radio driver has just received a packet */
  int (* pending_packet)(void);
  /** Turn the radio on. */
  int (* on)(void);
  /** Turn the radio off. */
  int (* off)(void);
};
```

Communication towards the CC2520 radio is done via an SPI interface. Either initialization, configuration of the radio or data transfers are done through the SPI port. Some modifications are needed to the vendor's firmware (cc2520 Example Code - swrc090a) since the pin configuration is variable among different MSP430 models and normally various SPI buses are available. Additionally, interrupt handlers have to be enabled on the specific ports of the microntroller. Moreover, the size of the firmware is big for the 128 Kb flash memory and many of the library functions are not needed in our implementation, as a consecuence the vendor's implementation had to be reduced.

The hardware abstraction interface is reduced to the file `hal_cc2520.c`. On the upper layer, the functions library for initialization of the radio, handling packets and other link-local related functions can be found at `cc2520ll.c`. Both under `/platform/torrija/dev` directory.

More information about basic CC2520 radio features such as low-power modes, serial communication, the instruction set or the transmit and receive modes are available in the data sheet [53]. Some extra features have been also used in the implementation of the driver. For debugging purposes it is possible to configure the radio on promiscuous mode so we can inspect the packets travelling in the air even if they are malformed. The random number generation engine from environmental noise is used as a seed for generation of random numbers in Contiki. Finally, the CC2520 has extensive support for security operations that are used for encryption of messages as defined in IEEE 802.15.4 [32]. More information about the software implementation can be found at the vendor's firmware example code [54] available at `www.ti.com/product/CC2520/toolssoftware` or in the platform implementation (Appendix B).

6.5.3. USB driver

The USB protocol is quite complex so there is no point to describe its features in this section, but the user can refer to the USB specification, Texas Instruments documentation [65] [67] [50] and firmware API description [68] or many sources available in the Internet for simpler description (http://www.beyondlogic.org/usbnutshell/usb1.shtml). Texas Instruments provides a firmware for the integrated USB module in an also complex architecture that should not be described here. Instead we will give a basic description of what is provided and what we need from the driver implementation.

The USB protocol is not trivial to implement and it requires strict compliance even in the simplest applications. TI provides an intuitive API stack (USB CDCv1.19 for Code Composer v4) that eliminates most of the underlying complexity of implementing USB. For documentation TI also provided a very useful Programmer's as a reference for the API. It also clearly describes the basic concepts.

Descriptors: A USB Descriptor Tool for the MSP430 USB [68] is provided for quick configuration. It automatically creates descriptors that must be reported to the host saving great amount of time to the developer. Descriptors inform the USB host about several details of the device of different kind. There are a device descriptor, configuration descriptors, interface descriptors and endpoints descriptors. These descriptors are sent to the USB host upon its request when a device is connected.

Any USB device contains a certain number of what are called endpoints. Support for multiple endpoints allows for composite USB devices which can have more flexible communication with hosts. An USB Composite Device is a peripheral device that supports more than one device class.

Class stacks: The MSP430 USB API provides the most common device classes [67]:

- Communications Device Class (CDC): It is used mainly for telecommunication devices (primarily for modems, fax machines, telephony applications, digital phones, or in computer networking devices as an interface for transmitting Ethernet or ATM frames, as well as COM-port devices). When connected to a SO like Windows XP it appears on the host as a virtual COM port (considered a flexible, fast and simple interface]. The CDC provides high bandwidth by using bulk transfers. The main disadvantage is that a simple info file (.inf) should be available in Windows so it can associate driver built-in with the CDC.
- Human Interface Device (HID): It is mostly targeting peripheral device (mice, keyboards or game controllers) but it is flexible and suitable for a wide variety of applications. As a disadvantage, it has limited bandwidth (64 KBps) [68]. As a benefit it does not require any info file to be available at Windows. The installation is silently done in the OS if the driver is available. Another benefit is the great amount of device drivers available in most operating systems nowadays. Generic drivers allows for faster deployment and easier installation. The basic HID functions are defined in USB-IF documentation (www.usb.org) and [65].
- Mass Storage Class (MSC): It enables high data transfers between host and device (primarly used for digital cameras, flash card readers, etc.). It provides similar bandwidth to that of CDC and it also loads silently in Windows like the HID. However, it it is a bit more complex (e.g. developers need to implement a file system to make it work)

Host connectivity: In Windows when a new USB device is connected to a host machine, the operating system then queries the bus driver for the hardware IDs associated with the device. The device USB bus driver sends the VID (vendor) and PID (product) ID numbers. They are taken directly from the device descriptor that we should be installed in the device. Other fields in the device descriptor are the device class, subclass, or protocol. After retrieving the corresponding ids, the operating system searches for `.inf` files in the system (assuming Windows). If one of these files contains a match for the device ID, Windows loads the driver that is indicated by that `.inf` file. In Linux (Ubuntu 12.04) the connection does not need any specific information file and it is done automatically. The device appears as a usb0 directory that can be read and written.

API: The TI USB API [68] support three data transfer types, but the developer does not need to worry about the details associated with each type since their usage is determined by the device class:

- Control for status data.
- Interrupt for low bandwidth.
- Bulk for high bandwidth.

By default, the API provides a single CDC interface resulting in a single COM port on the host (Windows). The provided implementation requires one interrupt IN endpoint for notifications to the USB Host, one bulk IN and one bulk OUT endpoint for data transfers plus the default IN/OUT for status control.

The provided code is supporting HID and CDC devices, a separate project is provided for MSC devices implementation. In our application we decided to only implement CDC support for communication devices for being considered the most versatile and suitable for our purposes. In order to save memory, the code had to be reduced to only support CDC.

The full USB provided implementation allows the developer to focus only on the data handling. There are two differentiated layers, the API space (`usbcdc.c`, `usb.c` and `usbisr.c`) and the application space that the developer can focus on (`usbcdc_constructs.c`, `usb_eventHandling.c` and `descriptors.c/h`)

For handling data in and out the interface we are interested in two recommended application level functions [68]:

- `sendData_inBackground(BYTE* dataBuf, WORD size, BYTE intfNum, ULONG ulTimeout)`

- `receiveDataInBuffer(BYTE* dataBuf, WORD size, BYTE intfNum)`

Interrupt handling resides within the API layer but interrupts can be handled at the application layer through *events* that can be thought of as the same as an interrupt. Handler functions for each event are defined by the API in application space. When some new data is available the corresponding event handler calls texttprocess_sync(..) to inform the USB process about it.

Regarding the clock System MSP430 devices supporting USB include a PLL that sources from an external high-frequency crystal. The API function call needs to be informed of the frequency the crystal oscillator is operating in order to configure the internal PLL.

Among other changes the pins distribution in the hardware layer had to be associated to that of the MSP430F5519 pin map.

6.5.4. RFID driver

The TRF7960 is a flexible and highly configurable RFID front-end chip and data framer at 13.56 MHz. It provides selectable protocols, auto-configuration modes, user programmable registers, selectable output power, adjustable band-pass filter, variable input voltage from 2.7 V to 5.5 V, parallel (8-bit) and serial (4-bit SPI) communication interfaces, etc. A set of built-in programming options make this chip very versatile.

Many features can be selectable by the developer (clock frequency, power modes, voltage of operation range, communication interface, modulation or protocol) by setting the control register available in the chip. The TRF7960 needs to be operated through the MSP430 micro-controller via an SPI serial bus or in parallel, the platform hardware has been designed to support both options but on the current version of the software only parallel support is implemented.

A driver handling the data for the different supported protocols is freely provided by Texas Instruments (TRF7960 Firmware Source Code for CCS - sloc203), but it has to be adapted and optimized to the specific desired options and integrated into Contiki. The firmware provides a hardware abstraction library (HAL), implementations of the supported protocols (see Chapter 5) and an loop application.

The hardware functions located at `hardware.c` are adapted for the platform hardware by configuring the parallel ports, enabling the interrupt port and associating the interrupt handler, selecting the clock source and configuring a timer. It has been previously decided in Section 4.6 that Tag-It protocol will be the only one implemented on the current prototype. It is implemented at `tiris.c` by a function that read on the 16 possible slots on the protocols. There is a 20 ms waiting time on each slot with and additional 20 ms time for initializations. More about the implementation can be read at the firmware description of the TI TRF796x [58], the implementation description of Tag-It protocol [59] or looking at the actual code implementation.

When data is available it should be handled by a protothread, delivered to the security application and sent out via USB interface so that the user is aware of it and can be authorised. Therefore, the main function of the provided Tag-It implementation need to be adapted to the processes framework in Contiki.

6.5.5. Other drivers

- As it was mentioned in the overview of the platform the clock module is a basic part in Contiki that timers are based on. We should provide the value for a defined constant (`CLOCK_CONF_SECOND`) with the duration of a second in system ticks.

  The MSP430F5519 has three clock signal (MCLK, SMCLK and ACLK) that can be sourced from external crystals or the internal micro-controller's oscillator VCO, but external crystals are considered more stable [51]. In the platform design MCLK and SMCLK are sourced from a external 32 MHz high-frequency crystal and ACLK from a 32,768 Hz external crystal. The MCU is running referenced to the MCLK external source and the Timer modules are using the ACLK as source. ACLK frequency is divided by 8 for Timer A (32768/8 = 4096 HZ) Hzand it is configured to execute the interrupt routine every 256 cycles (not too high, not too low) so the timer expires once every (1/4096) * 256 = 62.5 ms (1/16) that is the time of a tick. Therefore `CLOCK_CONF_SECOND` should be 16 since that is the number of times that the timer should execute the routine to fulfil one second.

  Additionally, a variable holding the number of ticks is also needed (`clock_time_t`), a initializing function for the clock module (`clock_init()`) that initializes the ACLK, set the divisor and the timer interrupt. Two more functions returns the number of system ticks and the number of seconds since the boot up of the system (`clock_time()` and `clock_seconds()` respectively). See `msp430_arch.c`

- There are two buttons in the platform that are useful for testing or for future designs. Each button has two functions returning zero when the button is not pressed and other than zero otherwise. A function permits Contiki processes to be attached and listen on the buttons status. When a button is pressed, the registered processes gets notified. This allows for external interaction on running processes. See `buttons.c`.
- Several function are provided to turn on/off or toggle thres LEDs (red, yellow and green) at `leds_torrija.c`
- Some other functions for writing or erasing directly on specific flash memory addresses is also provided under `mem_arch.c`.

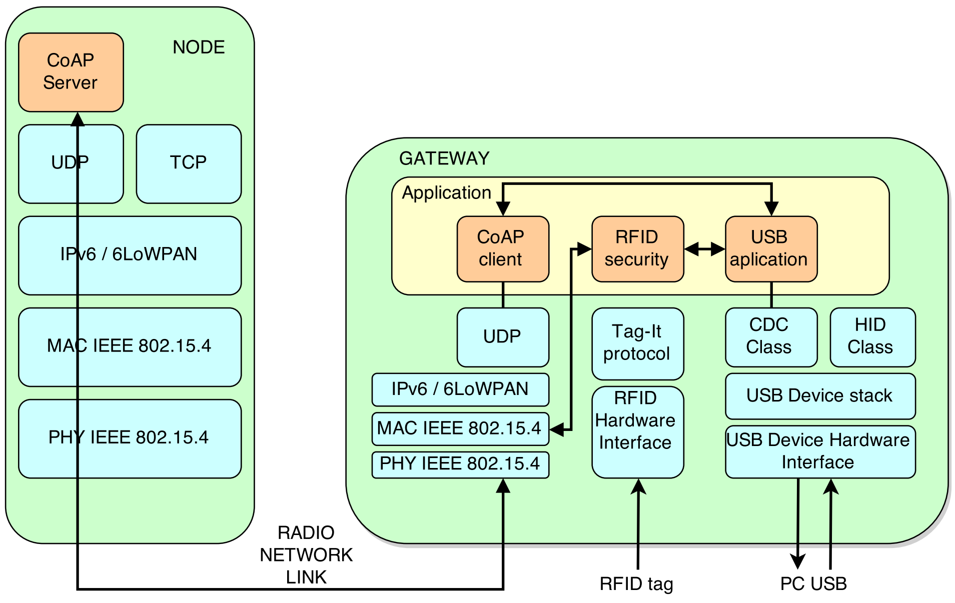


Figure 6.6: Protocol stacks. This figure shows the abstraction layers that allow the communication between the different modules.

6.6. Network

The implementation of the platform uses the IPv6 Contiki stack to establish IP communication between nodes. On a lower level the IEEE 802.15.4 MAC layer is implemented by Contiki's "sicslowmac" and partly TI radio firmware. It sets link-layer addresses on outgoing packets and parses headers from incoming packets to send them to upper 6LoWPAN layer.

Regarding the 6LoWPAN implementation, Contiki handles traffic in this way:

- For incoming traffic, packets are decompressed by the 6LoWPAN layer and handled in to the TCP/IP process.
- For outgoing traffic, the 6LoWPAN layer compresses the IPv6 packet header and it is forwarded to the MAC layer.

6LoWPAN: is an adaptation layer that allows IPv6 communication over IEEE 802.15.4 links in an efficient way. It is meanly performing header compression in our implementation, fragmentation for example is not enabled. The IPv6 header fields are compressed from 40Bytes of into 2 Bytes of 6LowPAN header. These fields can then be inferred from a packet when the adaptation layer obtains them from link-level information carried in the 802.15.4 frame. But it can also be elided based on shared context assumptions.

Regarding the 6LoWPAN implementation, Contiki handles traffic in this way:

- For incoming traffic, packets are decompressed by the 6LoWPAN layer and handled in to the TCP/IP process.
- For outgoing traffic, the 6LoWPAN layer compresses the IPv6 packet header and it is forwarded to the MAC layer.

Addresses configuration: In order to establish network links between nodes, the first problem is to resolve network addresses. Typically in the case of IPv4 traffic, ARP protocol is used to perform link-layer address resolution. It determines link-layer addresses based on the destination IPv4 address of packets. Nodes build an ARP cache then with the learned address for next received messages. IPv6 on the contrary does not need address resolution on the link-layer, a Neighbor Discovery protocol [36] is defined and implemented in Contiki. It performs auto-configuration of addresses, duplicate address detection or determine neighbors link-layer addresses.

When a node wish communicate within a network it first sends a multicast neighbor solicitation message with a tentative link-layer address (pre-defined on each node and on the RFID corresponding tag). If no advertisement message is received then it considers the link-layer address as unique and builds the IPv6 address from it. Then other nodes can infer link-layer addresses from IPv6 addresses.

It is important to mention that we are assuming that all nodes will have a unique link-layer address (also readable from the RFID tag) so there will be no conflict of addresses.

Star topology: For the first version of the WSN it was decided to configure a star topology that could be scalable to more complex topologies in the future (see Chapter 4). Nodes are therefore only implementing basic IPv6-6LoWPAN links controlled by the gateway. The network could easily be extended to a mesh network by enabling some of the nodes to act as IPv6 routers, this could be done by enabling the RPL routing protocol implemented in the Contiki stack.

UDP: CoAP requires UDP protocol to work and both implementations are also available in Contiki. The reason for using UDP instead of TCP protocol is that TCP is not suitable for short-lived request/response models. The first handshake to establish communication can take some time on low duty cycle radios. It introduces a long overhead compared to UDP header, and that is important in such short IEEE 802.15.4 radio frames (128 bytes).

Coap: The CoAP protocol is based on a REST architectural style similar to HTTP but lightweight and adequate to constraints networks. The CoAP application layer is implemented by means of a Contiki processes (either in the client and the servers) that handles the incomming IP packets and the UDP sockets connections. It is explained in detail in section 6.7

6.7. Application

The application layer conform the logic that handles all the incoming and outgoing data within the gateway. It makes possible to accomplish the goals that have been proposed for this Thesis.

The application layer can seen as a group of applications with their own purpose, acting in different phases and interacting independently with different modules in the architecture. In this sense we can have an application in charge of booting up the system and setting all the modules up, providing the necessary configuration for them. Another application in charge of the security, encryption and recognition of new nodes via the RFID reader. A CoAP client application receiving external orders from the USB in order to perform requests to a CoAP server and a similar application that sends responses outside the gateway through the USB interface.

6.7.1. Bootstrapping

At bootstrap time all the modules composing the gateway need to be initialized in a specific order, starting with the microprocessor internal elements, peripherals, communication ports and radio modules. Once the hardware is ready the Operating System can initialize the processes module, the netstack and finally the application processes reading from the USB port, reading RFID tags and the CoAP client. The complete sequence is implemented in the main function that can be found in the `contiki-torrija-host-main.c` file. It has additionally been attached in Apendix B at the end of this document.

1. Initialize micro-controller: stop the watchdog, select clock sources, wait until clocks are stable, reset pins...

2. Initializes the clock library and the ticks counter (needed for timers)

3. Clear buffers, initialize uIP variables, create a MAC address based on a unique id (it is obtained from the manufactured wafle writen in the chip). The sicslowmac

layer and the radio require this mac address that will be placed in the global variable `rimeaddr_node_addr` (declared in `rimeaddr.h`).

4. Sets the link layer address (Should be done before the `tcpip_process` is started since the default IPv6 address will be set by combining the link local prefix (fe80::/64)and the link layer address inside `uip_init()` function.
5. Initialize other peripherals: buttons, leds (they inform about the current state. Only green will be on if the initialization squence is successful)
6. Initialize USB and check state (the user can get useful information about the state of the gateway in real time)
7. Initialize the process module
8. Start radio and radio input process
9. Initialize stack protocols
10. Initialize tcpip process
11. Initialize applications: rfid to find tags, usb input process and coap client.
12. Enter main loop: a while loop where the function `process_run()` continuously poll every process that requested to be polled.

Within the general boot-up sequence there are specific and important boot-up operations that are performed inside some modules. For the radio, the RFID chip and partly the USB interface the sequence is very similar:

1. Initialize interface pins
2. Initialize communication port (SPI, parallel)
3. Check hardware: voltage regulators, oscillator is stable...
4. Write non-default register values
5. Register interrupt handler
6. Enable reception

Considering the radio, before the gateway starts building up the network, the radio must take care of some task at the physical level: scan the radio environment to find occupied channels and other networks, choose appropriate channel, select a network identifier (PAN id) and select a node address. The last two are mandatory and they are selected previously by the developer: the network identifier is a 2 bytes number that must be common for all nodes and the node address is obtained from the previously set `rimeaddr_node_addr`. The functionality regarding the channel selection is optional and the implementation is available in our code.

6.7.2. Input data

Data is going in and out the gateway in three different ways, through a radio, through an RFID reader and through a USB interface. Each of them need a process to handle the data that is being received. In case some data needs to be sent out, the corresponding application directly sends the data using the driver library functions. The TCP/IP process is in charge of sends network packets to the radio driver. The CoAP client process sends out messages to the USB interface in case some requested sensor data has been received or if the RFID a tag has been found by the RFID process.

- The radio process (`cc2520_process`) handles the input packet received on the radio interface by using a polling mechanism. It expects to be polled when packets are received to perform an action. All processes that have been requested to be polled are scheduled by the Contiki kernel. In case a packet is received on the radio driver the interrupt handler causes the radio `cc2520_process` to be polled by means of the `process _poll(&cc2520_process)` function.

  The process declares a poll handler function `radio_pollhandler()` (by means of the macro `PROCESS_POLLHANDLER()`) that is called when the process is polled by the scheduler. There is an option to declare a function as an exit handler not used in this case. When the process begins it requests to be polled to make sure it happens at least once, then it waits until it receives an event to exit.

  The function `radio_pollhandler()` check if there is a some data pending to be read on the input buffer , it reads and stores the data in a packet buffer `cc2520_read(packetbuf_dataptr(), PACKETBUF_SIZE)` and passes the packet to the upper level in the Contiki stack `NETSTACK_RDC.input()` where it will be handled by the TCP/IP process.

```
PROCESS(cc2520_process, "cc2520␣process");
PROCESS_THREAD(cc2520_process, ev, data)
{
 PROCESS_POLLHANDLER(radio_pollhandler());
  PROCESS_EXITHANDLER(_nop());

  PROCESS_BEGIN();

  process_poll(&cc2520_process);
  PROCESS_WAIT_EVENT_UNTIL(ev == PROCESS_EVENT_EXIT);

  PROCESS_END();
}
```

Figure 6.7: Radio process to handle input packets

- The USB process works in a similar way than the radio process. It declares a process handler (`usb_pollhandler()`) but there is no need for an exit handler. Once the process starts it sends a poll request for itself to make sure it is executed at least once and then it just waits until it receives a termination request. Every time a character is received by the USB driver the routine software sends a request for the USB process to be polled.

When the process gets polled the function `usb_pollhandler()` checks that some data is in the usb buffer by checking the flag `bDataReceived_event`. The data is read `receiveDataInBuffer((pieceOfString,MAX_STR_LENGTH,1)` and stored (concatenated) into a global string buffer and echoed back to the interface (it will be shown to the user on the screen). This is continuously done until the Return character is pressed (0x0D), if a New line is found (0x0A) or until the buffer has reached the size limit `MAX_STR_LENGTH`. If this happens, an even is sent to the CoAP client process where the input message will be processed as a CoAP packet by calling `process_sync(..)`. Finally the flag is set to false, the buffers are cleared and the process is polled again in case some new data is available during the previous execution with `process_poll(..)`.

```
PROCESS_THREAD(usb_input_process , ev, data)
{
 PROCESS_POLLHANDLER(usb_pollhandler());
 PROCESS_EXITHANDLER(_nop());

 PROCESS_BEGIN();

 process_poll(&usb_input_process);
 PROCESS_WAIT_EVENT_UNTIL(ev == PROCESS_EVENT_EXIT);

 PROCESS_END();
}
```

Figure 6.8: USB process that receives input data

- For the RFID reader, the procedure to detect tags in the near field it is quite complex especially if the anti-collision algorithm is required [58] [59]. It also depends on the external conditions. Reading a tag implies that the user get the tag close enough to the reader and during a certain time so the reader can read all the data from it. Moreover, if other tags are present at the same time in the proximities an anti-collision control needs to filter the data from each tag. To sum up, reading a tag needs a big amount of time compared to the other functions of the gateway. In order to avoid the RFID to consume most of the CPU time and constantly block other processes, the RFID process is scheduled once every 500 ms. They way to do that is by means of two processes yielding each other. One process being in charge of establishing the timer `rfid_find_tags` and another executing the actual sequence and algorithms to read the tags.

```
PROCESS_THREAD(rfid_find_tags, ev, data)
{
  static struct etimer find;
  PROCESS_BEGIN();
  proc_register(&rfid_find_tags);
  initial_settings();   //Set Port Functions for Parallel Mode
  process_start(&tagit_inventory_request, NULL);

  while(1){
    etimer_set(&find, CLOCK_SECOND/4);
    PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&find));
    red_led_off(LED_RED);
    etimer_set(&find, CLOCK_SECOND/4);
    PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&find));

    start_find_tagit();
    process_post(&tagit_inventory_request, PROCESS_EVENT_CONTINUE, data);
    PROCESS_WAIT_EVENT_UNTIL(ev == PROCESS_EVENT_CONTINUE);
    end_find_tagit();
  }
  PROCESS_END();
}
```

Figure 6.9: RFID process that schedules the tag reading

Since the code of both processes is quite large, we are shortening them just showing the most important parts of the functionality and commenting other parts. In brief `rfid_find_tags` process initialises the communication between the reader and the micro-controller, starts the `tagit_request` process that implements the protocol to read the tags and enters an infinite loop. Inside the loop the process waits half a second before starting the reading, then it yields control to the `tagit_request` process and waits until it returns control. Eventually `rfid_find_tags` closes disables the reading on the RFID chip and restarts the loop again.

The reading sequence implemented by the `tagit_request` process is a bit more complex. Basically it starts by waiting for the `rfid_find_tags` process to yield control. Once it gets scheduled the initial functions prepare the buffer, set reader options for the Tag-it protocol and once it is ready it check the different slots defined by the protocol where the tag might be transmitting the data. If data is received it is processed, the led is turned on, a message is sent to the USB interface and the security application stores the data as a node MAC address to be filtered in the network communication. If a collision occurred the antin-collision algorithm tries to recover the data or there is a timeout on the slot, it jumps to the next slot. At the end the control is yielded to the `rfid_find_tags` process.

At the beginning of both processes the function `proc_register(..)` registers each process each time they are scheduled. This is useful for the interrupt routine, whenever

```
PROCESS_THREAD(tagit_request, ev, data)
{
/* Declaration of static variables */
PROCESS_BEGIN();
proc_register(&tagit_request);

while(1){
        // Wait until rfid_find_tags process let it continue:
        PROCESS_WAIT_EVENT_UNTIL(ev == PROCESS_EVENT_CONTINUE);
        // Get the reader prepared: Prepare buffer, set options,
        // send command "RESET", Continous Mode...
        // Wait for end of TX interrupt
        // The Tag-it protocol defines 16 slots to be read
        for(i = 1; i < 17; i++){
                // Wait for RX complete:
                while((i_reg == 0x01)){
                        etimer_set(&i_r, CLOCK_SECOND/16);
                        PROCESS_WAIT_EVENT_UNTIL((ev == PROCESS_EVENT_TIMER)||
                                (ev == PROCESS_EVENT_CONTINUE));
                }
                if(i_reg == 0xFF){...}  /* received SID in buffer */
                else if(i_reg == 0x02){...} /* collision occurred */
                else if(i_reg == 0x00){...} /* slot timeout */
        }
        // If a tag is found turn red LED on, send message to USB interface
        // and perform other actions.
        // Post continue event to the rfid_find_tags process when finishing
}
PROCESS_END();
}
```

Figure 6.10: RFID process that detects and read data from the tag.

is detects a tag it will post a continue event and the IRQ status to the process that is registered at that moment so the data can be processed:

`process_post(registered_proc, PROCESS_EVENT_CONTINUE, &Register[0])`

6.7.3. CoAP

The application handling the sensor data is based on client-server model provided by CoAP that is at the same time based on a REST model where servers define resources under a URL that are publicly available. These resources are accessible by clients using well-known methods such as GET, PUT, POST or DELETE.

A preliminary version of CoAP based on the Internet draft `core-coap-04`[71] by the IETF is available in Contiki 2.5. It implements a client and a server each of them running on a Contiki process.

The gateway is hosting a client process that requires four functions:

- `void handle_incoming_ip_data()` checks that there is new data waiting to be read (`uip_newdata()`), initializes and allocates a buffer of a defined CoAP packet size and stores it. Then it parses the message and fills it into a CoAP packet structure (`coap_packet_t*`). If the packet is a valid CoAP packet it is passed to the `response_handler`. The buffer is eventually deleted.

- `void response_handler(coap_packet_t* response)` gets the payload from the response message, stores it and sends useful information to the USB interface.

- `void handle_incoming_usb_data()` checks that new data is available at the USB buffer and allocates a buffer to store the string data of `MAX_STR_LENGTH` size. Then it allocates memory for the CoAP packet the message and parses the message. The CoAP packet structure is filled with the parsed data obtained from the string message and it is passed to the `send_data(coap_packet_t* )` function. The buffers are eventually deleted.

- `void send_data(coap_packet_t* request)` check that the CoAP packet is valid and fills missing fields such as the etag option, sequence number or other options for internal use if possible. A new UDP connection is established with `udp_new(&server_ipaddr, UIP_HTONS(REMOTE_PORT), NULL)` and `udp_bind(client_conn, UIP_HTONS(LOCAL_PORT))` with the server with the parsed address. The packet is moved into a `MAX_PAYLOAD_LEN` buffer and it is sent to the remote server through the client UDP socket connection using `uip_udp_packet_send(client_conn, buf, data_size)`.

```
PROCESS_THREAD(coap_client_app, ev, data)
{
  PROCESS_BEGIN();
  while(1) {
    PROCESS_YIELD();
    if (ev == usb_event) {
      handle_incomming_usb_data();
    } else if (ev == tcpip_event) {
      handle_incoming_ip_data();
    }
  }
  PROCESS_END();
}
```

Figure 6.11: CoAP client process waiting and handling incoming messages from the USB interface and the TCP/IP stack.

The client process shown above is simply an infinite loop that waits until two kind of events are received. In case some data is received from the USB interface a `usb_event` is received by the `coap_client` process. Then the function `handle_incomming_usb_data()`

process the data and send a message to a CoAP server if necessary. If a packet has been received from the network then the `coap_client` process should received a `tcpip_event` and call `handle_incoming_ip_data()` to handle the packet.

Typical CoAP REST messages are described in section 4.7. Below is the prototype of a typical message that could be receive by the USB interface. It is similar to the scheme followed by web browser:

```
coap://<server_address>:<port>/<resource_name>?<action>
```

The sensor nodes act as servers for the application. As it has been mentioned in previous sections, the design and development of the sensor boards is not part of this Thesis, but the design of the application is. In that sense this Thesis has to provide the framework and mechanisms over which the application is built. In the case of the sensor boards that is to install a CoAP server on each node providing a set of resources that should be accessible within the network.

The server application is also built on a Contiki process. This process requires *resources* that are be activated within the process. These resources are representations of the data obtained by the sensors or actuators that performs an action in the system (e.g. a switch turning on/off a LED).

Resources can be of three different types:

- Basic resources that return a value or perform a task at a certain moment.
- Periodic resources that periodically return data within a defined interval.
- Observable resources that the clients can subscribe to so they can receive an status every time there has been a change in the resource.

At least for this prototype and for easier testability and design we are only defining basic resources and periodic resources. The sensor boards integrate different sensors in it: light, temperature, humidity, a battery meter and an accelerometer, all of them represented as resource on the board that can be accessed via a URL. As additional resources there is a LED that can be toggled, a *discover* resource that is the default resource located at generic /.well-know/core URL providing a list of available resources.

Each basic resource requires a `[resource name]_handler` function to read the sensor data or perform an action, build the payload and send out the CoAP message. There are different formats that can be used to represent the data. In this Thesis we are using `plain text` format for the sensor data representation and `application link format` for the discovery resource whose main function is to provide a list of the available resources URI:s hosted by the server.

Periodic resources are declared by means of the macro:

```
PERIODIC_RESOURCE(<resource>, <REST_METHOD>, "<resource>", seconds)
```

It requires a `<resource name>_periodic_handler` function to perform an action and also a `<resource name>_periodic_request` function that generates the response CoAP

```
RESOURCE(discover, METHOD_GET, ".well-known/core");
void discover_handler(REQUEST* request, RESPONSE* response)
{
  char payload[MAX_PAYLOAD];
  uint8_t index = 0;

  index += sprintf(payload + index, "%s,", "</lit>;n=\"L\"");
  index += sprintf(payload + index, "%s,", "</tmp>;n=\"T\"");
  index += sprintf(payload + index, "%s,", "</hum>;n=\"H\"");
  index += sprintf(payload + index, "%s", "</bat>;n=\"B\"");

  rest_set_response_payload(response, (uint8_t*)payload, strlen(payload));
  rest_set_header_content_type(response, APPLICATION_LINK_FORMAT);
}
```

Figure 6.12: Default discovery resource.

```
RESOURCE(toggle, METHOD_PUT | METHOD_POST, "toggle");
void toggle_handler(REQUEST* request, RESPONSE* response)
{
  led_toggle(LED_RED);
}
```

Figure 6.13: Example of a simple resource accepting PUT and POST methods and toggling a LED upon request.

packet and sends it to the client that requested the periodic resource. This is done every time the timer declared internally by the period resource macro expires.

Periodic resources are not recommended for future development, only used for prototyping reasons. They are replaced by the `observable resources` that are currently under discussions for standardization by the IETF and not fully implemented and tested by the current version of Contiki. The CoAP server process initializes the REST framework and starts the COAP process. Those basic or periodic resources wanted to be accessible should be activated with the `rest_activate_resource` functions as shown in 6.14

When building response packets it is important to consider the available payload size in CoaP. The size of a single IEEE 802.15.4 frame must be 127 octets. The link layer header size depends on several variable factors such as compressed/uncompressed PAN id, short or long of addresses are used, addressing modes, or if security is enabled. Typically 802.15.4 MAC header can be up to 25 octets (no security) or 25+21=46 octets (with AES-128 encryption). Additionally 40 octets are consumed by the IPv6 header, but with fully HC2 6LoWAN compression it can be reduced up to 2 octets. Another 8 bytes correspond to the UDP header. Finally the CoAP header consumes a variable length depending on the requested options with a mininum of 6 bytes. We can estimate a maximum payload size of 127-46-2-8-6 = 65 bytes very likely to be inferior.

```
PROCESS(coap_server_app, "CoAP␣Server␣Application");
AUTOSTART_PROCESSES(&coap_server_app);
PROCESS_THREAD(coap_server_app, ev, data)
{
  PROCESS_BEGIN();
  rest_init();
  rest_activate_resource(&resource_toggle);
  rest_activate_resource(&resource_discover);
  rest_activate_resource(&resource_light);
  rest_activate_resource(&resource_temperature);
  rest_activate_resource(&resource_humidity);
  rest_activate_resource(&resource_accelerometer);
  rest_activate_periodic_resource(&periodic_resource_light);
  rest_activate_periodic_resource(&periodic_resource_temperature);
  rest_activate_periodic_resource(&periodic_resource_battery);
  PROCESS_END();
}
```

Figure 6.14: CoAP server application activating a group of simple resources and periodic resources.

6.7.4. Security

The security application is providing a very basic functionality in the first version of the prototype. Every time a reader detects a tag the read data is identified as a node link address and it is stored in a specific area of the permanent (flash) memory using the MSP430 functionality. Every time a packet is received, it is filtered according if the link address is not present in the table.

A second version associates this table to the table of addresses generated from the IPv6 neighbour discovery mechanism, allowing only the addresses in the permanent table.

Another proposal is to associate the information in the tag with a security key that can be exchanged for encryption purposed but it is not implemented in this prototype.

Finally, also optional but available in the code, every time a packet is sent/received it can be encrypted/decrypted by the radio module built-in encryption mechanism with a default key that is the same in all the devices. It increases the header size of the link layer frame but we secure the data exchanged by the nodes. The fact that the key is common for all manufactured devices generates a security risk in case the key is discovered. A solution to this would be to store individual keys on the RFID tag of the nodes so the gateway can read them and perform individual encryption to the corresponding nodes.

Chapter 7

Results and Conclusions

7.1. Results

This section briefly describes and give some comments about the results obtained after the design and implementation of the platform and the deployment of the Wireless Sensor Network. It analyses what have been achieved and if the objectives and requirements proposed in Chapter 2 have been fulfilled.

7.1.1. Hardware

Once the hardware boards were finished all parts and modules and were tested prior to the integration of Contiki into the micro-controller. Starting with small programs to test simple parts such as clock frequencies, LEDs, buttons, voltages or power consumption. The next step was developing the firmware of the radio, USB and RFID modules individually based on the vendor firmware and test them separately.

As a result of all tests performed on the hardware we concluded that it is stable and suitable for the platform development. Moreover, it is also very versatile since it is provided with extra elements for future use such as three LEDs (red, yellow, green), two buttons connected to interrupt capable ports, switchable connectivity port serial(SPI)-parallel between the microprocessor and the RFID module, a BSL circuit for programming the device directly from the USB, different types of power resets (full or only modules) provided by pin connectors, selectable power source for the micro-controller and connectivity for selecting different RFID and 2,4GHz antennas.

Low cost has been achieved concerning the gateway with a price that is approximately 10 USD for the prototype considering only the main chips. It can increases up to 15 USD adding the prices of passive components. A price that can be easily reduced if manufacturing on a large scale. The price of sensor boards is even cheaper since the features of the microprocessor can be simplified and they do not integrate an RFID reader or USB module, however they are not part of our study [3].

Slow processing capability: was one of the constraints to overcome but it has been showed that a 8-bit micro-processors running at 16 MHz is capable of handling the network and the client application and the external connectivity in the gateway. We detected wrong behaviours and bad performance running at 8 MHz.

Small data storage capacity: the chosen MSP430F5519 micro-controller has 128 Kb of flash and 16 Kb of RAM. The code footprint with basic compiler (-O3) optimizations consumes approximately 62 Kb but it can be improved as well as the memory consumed at execution time.

Efficient energy management: the radio transceiver and microprocessor may be put on sleep mode at some intervals of time depending on the application needs either in the gateway or in nodes.

7.1.2. Network communication

The network connectivity is mainly provided by Contiki. Only IPv6 is used with 6LoWPAN compressed headers. Neighbor Discovery implementation is used to detect other nodes and the link layer is partly provided by the MAC implementation in Contiki, the firmware provided by Texas Instruments for the CC2520 radio and our adaptation.

The functionality within a local network scope has been tested: the filtering of radio channels, selection of a network id, startup to IPv6 stack, discovery of nodes, set of network addresses and direct communication between the sensor boards and the gateway has been successfully achieved (ICMPv6 protocol is also enabled).

It is convenient to remind that the current implementation only considers a star network topology. A hybrid and mesh topology is a future target and can be achieved by enabling routing protocols (like RPL) on the nodes. The star topology is in place and nodes can communicate directly with the gateway. The Neighbor Discovery protocol implemented in Contiki was tested on wireshark using one of the boards as a network sniffer and it works well.

Address autoconfiguration (stateless configuration of network interfaces addresses) and address resolution (to map link-layer addresses and local IPv6 addresses) are successfully performed [36]. There was also a test performed using IPv6 global public addresses but not in this platform that was also successful.

It has not been possible to test maximum data rate and throughput due to dependencies with the sensor nodes. Typically WSNs data rate values are a few tens of kilobits per second and the behaviour is usually irregular depending on the environment interaction with frequent periods of bursty traffic. The IEEE 802.15.4 radio link supports a maximum theoretical rates of 250 Kbps of raw data. It has not been possible to determine data rates or throughputs due to dependencies with the sensor nodes but we can refer to other platforms also using Contiki, the same protocol stacks and CoAP based application. If eliminating protocol over-heads, and considering the constraint of the hardware, limited processing capabilities, etc. the typical throughputs achieved in networks with a similar configuration varies from 20 kbps to 100 kbps depending on packet size and number of nodes [26] [25]. Contiki based networks obtains similar results than typical WSN values from 10 Kbps - 60 Kbps [42]. The requirement for the WSN is to support a maximum of 32 kbps throughput.

The network can be considered self-organized since it uses standard TCP/IP stack and supports IPv6 mechanisms for neighbour discovery and auto-configuration of addresses. It can also support routing. Only the action of the user is required to physically authenticate sensor boards and request the desired information to node.

Since it uses IPv6 stack, the network can be scaled to a worldwide network if it can be accessed externally. This feature is not implemented but it is possible using Ethernet connectivity [6].

The network has been designed primarily with a star topology useful in indoor environment where links do not require long distances. The protocols does not limit the number of nodes but it has a physical limit due to the memory restrictions since nodes keep a cache table with network addresses and a context.

7.1.3. USB interface

The USB module is integrated in the MSP430F5519 micro-controller and therefore no extra hardware is required except of course the one concerning physical connection. The firmware is provided by Texas instruments with support for different USB Device Classes [65] [68]. In order to reduce the code footprint it was modified to only give support to the Communication Device Class (CDC) and later integrated into the Contiki process framework to handle input messages.

The interface has been tested isolated from the complete system using either HyperTerminal Version 5.1 on Windows XP SP3 or directly reading and writing to the `/proc/bus/usb` interface in Linux Ubuntu 12.04 LTS (Linux kernel 2.3.15) with successful results.

The integration with Contiki and the rest of the applications is more complex. The USB input process is polled every time there is data to be read, then it has to compose the full message and send it to the application process for parsing and delivering to the network. There is also a waiting time since the interface needs to be ready when sending messages outside.

The gateway USB interface can act as a client for the bus. The USB connection can deal with high data rates (over 64 kbps at least) and it is convenient in the case that burst of information are delivered by the nodes at a certain moment due to some events. The requirement for the USB is the same as for the WSN to support a maximum of 32 kbps throughput.

7.1.4. RFID and security

The RFID module is intended to be used as part of the security application, but its functionality has not been fully exploited in the current version of the gateway.

Security for the platform can be provided by means of RFID node identification and optional packet encryption on the radio MAC layer.

It is implemented by means of an external RFID reader TRF7960 by Texas Instrument who also provides a firmware. Similarly to the USB firmware, it has been reduced to provided just the required functionality. In this case, the firmware provides an implementation for different protocols [58] but only Tag-It protocol is supported. It works successfully independently from the system but the default reading functions consume a long time (in the order of 100 milliseconds) compared to rest of the applications.

The RFID firmware library was integrated into the Contiki framework to make it available through processes. A tag-reader process is scheduled by another process (RFID scheduler) after a certain period (500 ms was considered enough time for the user interaction), avoiding the reader-process to run consecutively consuming large amounts of time. The RFID reader gets active only once every 500 ms so the problem of starvation of other processes is fixed by this mechanism of two processes.

Tag-it TI tags that can store up to 256 bits of configurable user data and a UID Number of 64 bits. That is considered enough for the security application.

7.1.5. Sensor application

The main application consisting on the REST client-server CoAP model was tested locally using different type of resources (sensors, LEDs and discover) on the server node and a gateway client installed on the gateway. Application packets that needs to be sent out to the nodes also are handled by the CoAP client process and directly delivered. Application packets received by the TCP/IP process are delivered to the CoAP application and eventually sent out through the USB interface. These two paths of communication are successfully working.

The gateway sends CoAP requests as an action from previously parsed messages received on the USB interface. Direct CoAP requests sent from the gateway to the node receive proper CoAP responses with the data requested (discover resource, get sensors data, periodic requests) or confirmation of an action performed (toggle a LED) on the node, so it is proven successfully.

The application works correctly on a small scale but it have not been tested with a big number of sensors. In order to extend the test scope of the application and verify the integration of the local network into the real Internet-of-the-Things world we used the 6LoWAN proxy platform developed in [6]. It is a gateway using Ethernet connectivity and IPV6 on one side and 6LoWPAN based wireless that allows the nodes to get public global IPv6 addresses (IP addresses are constructed from an link-local address of the boards and a prefix given by the IPv6-compliant router at Wireless@KTH lab [1]) and be accessed remotely from outside the network. This opened the possibility of testing the server nodes from a remote client application. This was done with the help of a add-on for Mozilla Firefox version 4.0 called Copper (version 0.8.0) [79] [80] that implements a CoAP client [71]. Copper supports URI handling for the CoAP scheme, implementation of RESTful methods (GET, POST, PUT, and DELETE), resource discovery, blockwise transfers and observing resources. As a result we were able to run CoAP servers on sensor devices with public IPv6 addresses communicating with the remote Copper CoAP client, proving that the application can be scaled to massive networks.

7.2. Conclusion

It has been a long journey since the proposal of this Thesis until its finalization and many obstacles had to be overcome along the way. The huge amount of information, hardware devices, technologies, protocols and tools available in the market made the research of the optimal platform a tough problem. Moreover, the design of both the hardware and the software planes in parallel increases dramatically the complexity of the design and it becomes a source for instability, uncertain failures and multiple types of errors. However, it provides a great degree of freedom in the design allowing the most optimal combinations.

On the other hand the advantages gained from such complexity compensates for the hard work. On one side the rich knowledge and experience achieved in different fields of the embedded world (micro-controllers, low-power radios, RFID readers, PCB design, software architecture, constrained programming or debugging), also the possibility of participating in the complete cycle of construction of an embedded device and deployment of a network and of course the satisfaction of seeing it alive. We have gone through a conceptual design that gave us insights of other platforms and technologies currently available in the market, costs, trends. So we could look for opportunities for optimizations, improvements, minimization of cost with maximum versatility.

A cost-efficient wireless sensor network with star topology using a simple client-server model application and basic authentication of nodes has been achieved. That concludes that the main requirement of this report has been basically fulfilled. Of course there is still room for improvements, additional functionality and performance optimizations but it has been proven the great versatility and cost-efficiency of the designed gateway platform (and indirectly the nodes), the simplicity and scalability of the network using the standard IPv6 stack, the flexibility of the CoAP application protocol and the potential of the RFID module.

The search for a trade-off between costs, functionality and flexibility for the platform consumed a vast amount of time. Much more efforts could have been put on the implementation of the software functionality, testing and debugging. Even though the results are still acceptable these are tasks that need to be performed but that are considered at this point as a future work.

Chapter 8

Future work

Even though the basic functionality and the primary challenges have been fulfilled, there are still many improvements and updates that can be considered as a continuous work. Moreover, other applications

- General code improvements, testing and debugging of the different modules and the applications can be done more exhaustively. Several risks have been observed that can be mitigated.

  There is a risk that the USB implementation blocks other processes and it is a potential source of segmentation faults and memory leaks since it shares memory with other processes and it has to handle external string inputs and different buffers.

  The problem of the process starvation occurring in the RFID module is solved with two processes scheduling each other after a certain time. However this module is still considered a potential source of reading errors (collisions), segmentation faults, and memory leaks for the same reasons as the USB module.

  Due to time constraints at the time of design there was no opportunity for performing exhaustive testing. The whole system running with all its modules enabled is not stable and need some more debugging. A crash in the system was observed when exhausting it with many packets, reading tags and using USB at the same time, as well as a progressive decay in the performance along time in this situation.

- In this version of the gateway a USB port is used for external communication towards the network due to former requirements. A new version with an Ethernet interface converting it into a compliant TCP/IP gateway would suppose a great advantage for scalability and integration of the devices in the Internet with no need of computer or adhoc messages through the USB interface. A working solution has already been tested on a different but very similar platform developed by another Master Thesis work [6]. This platform is a 6LoWPAN gateway that communicates externally through an Ethernet interface and a with the local sensor network by means of a IEEE 802.15.4 compliant radio.

- Several aspects can be considered to improve the security. Some actions need to be engineered to be performed by nodes in order to keep them locked to only one network, avoiding other networks created by other 6LoWPAN gateways in the vicinity. The current implementation only read RFID tags to get MAC addresses from new nodes and authorise them in the network, but there is a potential of security keys being also shared this way to implement encryption and key exchange mechanisms such as Diffie-Hellman. Some features of the CC2520 radio can be considered such as the built-in encryption algorithms at link layer or the MAC filter that unfortunately is limited to 12 long IEEE 802.15.4 link addresses [53]. To conclude, a implementation of the Datagram Transport Layer Security (DTLS) [78] protocol can be also considered for providing security on UDP level and hence CoAP.

- Due to time constraints at the time of design there was no big opportunity for performing exhaustive testing, so a proper and more formal testing is needed for each component and from a black box perspective at a higher abstraction where the system is seen as a whole. Different suites of test cases can be written for this purposes in different languages. It would be also beneficial to perform some static and dynamic analysis on the code and check the code coverage. The whole system running with all its modules enabled is not stable and need also some debugging. Some system crashes were observed in different situations as well as a progressive decay in the performance along the running time.

- Another big change that is proposed is a code update and re-factoring to the newer Contiki 2.7 version (15 November 2013), including among other things severe bug-fixes in the 802.15.4 framer, the ContikiMAC, the 6lowpan fragmentation handling, the IPv6 stack and it supports the finalized standard proposal for CoAP [75](RFC7252 June 2014) called Erbium for Contiki[81]. It also provides additional functionality on the IPv6 stack, it includes a new testing framework and support for new platforms.

  Today nodes operate as simple hosts deployed in a star topology network. Installing routing capabilities on some nodes would be needed to create more scalable mesh networks. In order to do that, routing protocols such as RPL are available in Contiki's IPv6 stack.

- Since the platform is just accepting basic CoAP messages as an input and receiving raw data from sensor. In order to provide more functionality for interoperability, it is a good idea to design and implement a set of commands to be available for the user on the USB interface (in a similar way to the popular AT commands or the linux prompt and command line). This would also allow the design of more sophisticated test tools and graphical user interfaces (GUI) to present the sensor data, show the network configuration, activate/deactivate nodes and more in a much nicer way. That can be developed for any host SO where the board is connected. An additional improvement for the USB firmware could be to implement HID instead of CDC since HID does not require any kind of installation and the data rates event though lower, it is still acceptable for the application (max 64kB/s).

- With little modifications on the board, by removing the USB interface, adding a battery as a power source and optimizing the software for low-power. The gateway can be easily converted into a wireless 6LoWPAN-based RFID reader. It would act as a server node in this case so the CoAP and the security applications should be modified to convert it into a CoAP server with an RFID reader as an observable resource. The construction of a portable multi protocol RFID reader based on 6LoWPAN opens a wide range of possible future applications in the Internet of the Things world.

Bibliography

[1] Wireless@KTH Lab. KTH-Royal Institute of Technology. Stockholm, Sweden. 2015. [Online]. Available: wireless.kth.se. [Accessed: 03- Jun- 2015].

[2] M. Smith. "Wasa Board Project", KTH-Royal Institute of Technology. Stockholm, Sweden. 2015. [Online]. Available: http://people.kth.se/~msmith/wasa/wasa_board_project.html. [Accessed: 03- Jun- 2015].

[3] Joaquin Juan Toledo. "Wireless Sensor Architecture for a Home Event Managment System". KTH Masters Thesis. Stockholm, Sweden. Jun 2015.

[4] Sotiris Falieris. "Sensor and actuator data fusion architecture for a home-event management system". KTH Masters Thesis TRITA number TRITA-ICT-EX-2010:64. KTH ICT Dept., Stockholm, Sweden. Jun 2010.

[5] Lagnajita De. *Application Framework for a Home Event Management System.* KTH Masters Thesis TRITA number TRITA-ICT-EX-2010:63. KTH ICT Dept., Stockholm, Sweden. Jun 2010.

[6] Luis Maqueda Ara. "Neighbor Discovery Proxy-Gateway for 6LoWPAN-based Wireless Sensor Network". KTH Masters Thesis TRITA number TRITA-ICT-EX-2011:221. KTH ICT Dept., Stockholm, Sweden. Oct 2011.

[7] ZigBee Alliance. *ZigBee Specification.* ZigBee Document: 053474r06, Version 1.0. Dec, 2004. [Online]. Available: https://docs.zigbee.org/. [Accessed: Jun- 2012].

[8] One-Net community. *One-Net overview.* Technical report, March 2008. [Online]. Available: http://www.one-net.info/. [Accessed: Mar- 2011].

[9] One-Net community. "One-Net: wireless control for everyone". [Online]. Available: http://www.one-net.info/. [Accessed: Mar- 2011].

[10] Texas Instruments. "SimpliciTI Compliant Protocol Stack - SIMPLICITI - TI Software Folder", 2011. [Online]. Available: http://www.ti.com/tool/SimpliciTI. [Accessed: Jun- 2011].

[11] Y. Yang , D. Flowers. *MiWi Wireless Networking Protocol Stack. Technical report: AN1066. Microchip Technology Inc., 2006. [Online]. Available: http://www.microchip.com. [Accessed: Jun- 2011].*

[12] "TinyOS Official Site", 2013. [Online]. Available: http://www.tinyos.net/. [Accessed: Jun- 2011].

[13] JP. Norair. Introduction to DASH7 Technologies. Technical report 1st Edition. Dash7 Alliance. March 2009. [Online]. Available: https://dash7.memberclicks.net/assets/PDF/dash7%20wp%20ed1.pdf. [Accessed: Jun- 2011]

[14] "Radio frequency identification for item management - Part 7: Parameters for active air interface communications at 433 MHz". International Standard ISO/IEC 18000-7:2009. Published 2009.

[15] Sigma Online, "Z-Wave : Home control", Z-wave.com. [Online]. Available: http://www.z-wave.com/. [Accessed: Jun-2011].

[16] *SynkroRF Network Reference Manual* Document Number: SYNKRORM Rev. 1.4, [Online]. Available: http://www.freescale.com/. [Accessed: Jun-2011]. Freescale Semiconductor. Denver, Colorado. 2011.

[17] *Insteon: The details* Whitepaper Version 2.0. http://www.cache.insteon.com/documentation/insteon_details.pdf. Insteon. 2008-2013.

[18] H. Foundation, "WirelessHART Overview", En.hartcomm.org, 2015. [Online]. Available: http://en.hartcomm.org/hcp/tech/wihart/wireless_overview.html. [Accessed: Mar- 2011].

[19] Sanjuansw.com, "San Juan Software - PopNet: The Easy, Economical Wireless Sensor and Control Network", 2009. [Online]. Available: http://www.sanjuansw.com/?p=15. [Accessed: Mar- 2011].

[20] Bluetooth.com, "Bluetooth Low Energy", 2015. [Online]. Available: http://www.bluetooth.com/Pages/Bluetooth-Smart.aspx. [Accessed: Mar- 2011].

[21] K. Holger, W. Andreas. *Protocols and Architectures for Wireless Sensor Networks.* Hoboken, NJ: Wiley, 2005.

[22] Y. Zhang, L. Yang and J. Chen. *RFID and Sensor Networks. Architectures, Protocols, Security and Integrations.* Boca Raton: CRC Press, 2010.

[23] John Wilson. *Sensor Technology Handbook.* Amsterdam: Ed. Newnes, Elsevier. 2005.

[24] M. Obaidat and S. Misra. *Principles of Wireless Sensor Networks.* Cambridge, England: Cambridge University Press. 2004.

[25] M. O. Farooq and T. Kunz, "Contiki-Based IEEE 802.15.4 Channel Capacity Estimation and Suitability of Its CSMA-CA MAC Layer Protocol for Real-Time Multimedia Applications," *Mobile Information Systems*, vol. 2015, Article ID 398637, 9 pages, 2015.

[26] Buratti, A. Conti, D. Dardari and R. Verdone, "An Overview on Wireless Sensor Networks Technology and Evolution", *Sensors*, vol. 9, no. 9, pp. 6869-6896, 2009. [Online]. Available: www.mdpi.com/journal/sensors. [Accessed: 03- Jun- 2015].

[27] D. Gislason. *Zigbee Wireless Networking.* Oxford: Newnes, 2008.

[28] D. M. Dobkin. *The RF in RFID, Passive UHF RFID in Practice.* Amsterdam: Elsevier / Newnes, 2008.

[29] K. Finkenzeller. *RFID Handbook.* Chichester, England: John Wiley and Sons, 2003.

[30] NFC Forum, "Specs for Wi-Fi Pairing with NFC Get Boost; Wi-Fi Alliance Sees Advantages with NFC - NFC Forum", 2014. [Online]. Available: http://www.nfc-forum.org/specs. [Accessed: Jun- 2012].

[31] Z. Shelby and C. Bormann. *6LoWPAN The Wireless Embedded Internet.* Chichester, England: John Wiley and Sons, 2009.

[32] "IEEE 802.15.4 standard Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)". IEEE Standards Association. IEEE Std 802.15.4-2006. Sep 2006.

[33] IETF 6LoWPAN Working Group. "IPv6 over Low power WPAN (6lowpan)", 2011. [Online]. Available: https://datatracker.ietf.org/wg/6lowpan/charter/. [Accessed: 04- Jun- 2015].

[34] Thubert P. Hui J. "Compression Format for IPv6 Datagrams in Low Power and Lossy Networks (6LoWPAN)". IETF Internet Draft. draft-ietf-6lowpan-hc-15 (work in progress). Feb 2011.

[35] S. Chakrabarti, J. Laganier, K. Kim, W. Haddad. "IPv6 over Low Power WPAN Security Analysis". IETF Internet Draft. draft-daniel-6lowpan-security-analysis-05 (work in progress). Jun 2011.

[36] S. Kent. "Neighbor Discovery for IP version 6 (IPv6)" IETF Network Working Group. Request for Comments Ref. 4861. Sep 2007

[37] G. Montenegro, N. Kushalnagar, J. Hui, D. Culler "Transmission of IPv6 Packets over IEEE 802.15.4 Networks" IETF Network Working Group. Request for Comments Ref. 4944. Sep 2007

[38] J. Hui, P. Thubert "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks" IETF Network Working Group. Request for Comments: 6282. Sep 2011

[39] Swedish Institute of Computer Science, 2013. [Online]. Available: http://www.sics.se/contiki/. [Accessed: 04- Jun- 2015].

[40] Contiki-os.org, "Contiki: The Open Source Operating System for the Internet of Things". [Online]. Available: http://www.contiki-os.org/. [Accessed: 04- Jun- 2015].

[41] Contiki.sourceforge.net, "Contiki 2.6: The Contiki Operating System", 2015. [Online]. Available: http://contiki.sourceforge.net/docs/2.6/index.html. [Accessed: 09- Jun- 2015].

[42] A. Dunkels, B. Grönvall, T. Voigt. "Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensor". Swedish Institute of Computer Science, May 2004.

[43] A. Dunkels. *uIP - A Free Small TCP/IP Stack.* Tech. Description. Stockholm, Sweden. November 2001. [Online]. Available: http://www.dunkels.com/adam/download/uip-doc-0.5.pdf. [Accessed: 04- Jun- 2013].

[44] A. Dunkels. "Rime - A Lightweight Layered Communication Stack for Sensor Networks". Poster Abstract. *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*, Poster/Demo session Delft, The Netherlands. Jan 2007 [Online]. Available: http://dunkels.com/adam/dunkels07rime.pdf. [Accessed: 04- Jun- 2013].

[45] "Cisco, Atmel and the Swedish Institute of Computer Science (SICS) Collaborate to Support a Future Where Any Device Can Be Connected to the Internet" . Cisco. October 14, 2008. Retrieved February 2, 2015. [Online]. Available: http://newsroom.cisco.com/dlls/2008/prod_101408e.html. [Accessed: 04- Jun- 2013].

[46] Dunkels A., Schmidt O., Voigt T., Ali M. "Protothreads: Simplifying Event-Driven Programming of Memory-Constrained Embedded Systems". Swedish Institute of Computer Science, Stockholm. May 2004.

[47] Texas Instruments. *RF Connectivity with TI Microcontrollers.* Technical report ref. slab034w, Texas Instruments, 2013. [Online]. Available: http://www.ti.com/lit/sg/spab089a/spab089a.pdf. [Accessed: 04- Jun- 2013].

[48] Texas Instruments. *MSP430 Ultra-Low-Power Microcontrollers.* Technical Report ref. slab034w. Texas Instruments, 2013. [Online]. Available: http://www.ti.com/lit/sg/slab034w/slab034w.pdf. [Accessed: 04- Jun- 2013].

[49] *MSP430F5519 datasheet.* Technical Report ref. slas590l. Texas Instruments, 2009.

[50] *MSP430x5xx/MSP430x6xx Family user's guide.* Technical Report ref. slauu208g. Texas Instruments. June 2008 (revised July 2010).

[51] Peter Spevakand, Peter Forstner. *MSP430 32-kHz Crystal Oscillators.* Texas Instruments. Application Report ref. slaa322b. August 2006 (revised April 2009)

[52] K. Venkat, G. Morton. "MSP430 Competitive Benchmarking (ref. slaa205b)". Technical report, Texas Instruments, June 2005. [Online]. Available: https://www.eecs.berkeley.edu/~boser/courses/40/labs/docs/microcontroller%20benchmarks.pdf. [Accessed: 04- Jun- 2013].

[53] *CC2520 Radio 2.4 GHZ IEEE 802.15.4/ZIGBEE® RF TRANSCEIVER* Datasheet ref. swrs068. Texas Instruments. December 2009.

[54] *CC2520 Software Examples (Rev. B)* Compressed file ref. swrc090. Texas Instruments. Nov 2009

[55] Ti.com, "CC2520 Evaluation Module Kit - CC2520EMK", 2011. [Online]. Available: http://www.ti.com/tool/cc2520emk. [Accessed: 03- Jun- 2012].

[56] *Multi-standard fully integrated 13,56 MHz RFID analog front end and data-framing reader system.* Texas Instruments. Application report, SLOU186F, August 2006 (revised August 2010).

[57] *TRF7960EVM user"s guide.* Texas Instruments. SLOU192C, November 2006 (revised December 2008).

[58] ShreHarsha Rao, *Firmware Description of the TI TRF796x Evaluation Module (EVM).* Texas Instruments. Application report, SLOA134. March 2009.

[59] *Tag-it Transponder Protocol.* TIRIS Tech. by Texas Instruments. Reference Manual. March 2000.

[60] *Tag-it HF-I Pro Transponder Chip/Inlays.* Texas Instruments. Reference Guide, Ref. Number: SCBU004B April 2010.

[61] J. Varghese, *TF796x HF-RFID Reader Layout Design Guide.* Texas Instruments. Application report, SLOA13, April 2009.

[62] H. Liu, M. Bolic, A. Nayak, and I. Stojmenovie, "Integration of RFID and wireless sensor networks, in Proceedings of Sense IP 2007 Workshop at SenSy". University of Birmingham, United Kingdom. August 14, 2008

[63] Youbok Lee, *Antenna circuit design for RFID applications.* Microchip Technology Inc. DS00710C, 2003.

[64] John Schillinger, *Antenna Matching for the TRF7960 RFID Reader.* Texas Instruments. Application report, SLOA135, May 2009.

[65] "Universal Serial Bus Specification" Revision 2.0. [Online]. Available: http://www.usb.org/developers/docs/. [Accessed: 03- Jun- 2013].

[66] "Starting a USB Design Using MSP430 MCUs". Technical report, Ref. slaa457. Texas Instruments, 2013.

[67] Bhargavi Nisarga, Keith Quiring. "The ultra-low-power USB revolution". White paper. Ref. slay014. Texas Instruments. 2011.

[68] "MSP430 USB Communications Device Class (CDC) API Programmer"s guide". Tech. Report. Texas Instruments. 2013.

[69] Future Technology Devices International Ltd., 2013. [Online]. Available: `http://www.ftdichip.com/`. [Accessed: 03- Jun- 2013].

[70] IETF CoRE Working Group., "Constrained RESTful Environments (core)", 2011. [Online] Available: `https://datatracker.ietf.org/wg/core/charter/`. [Accessed: Jun-2011].

[71] Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)". IETF Internet Draft. draft-ietf-core-coap-04 (work in progress) [Online]. Available: `https://tools.ietf.org/html/draft-ietf-core-coap-05`. [Accessed: Mar-2011]. March 2011.

[72] Z. Shelby, C. Bormann "Blockwise transfers in CoAP", IETF. Internet Draft, draft-ietf-core-block-02 (work in progress), March 2011.

[73] Z. Shelby, K. Hartke "Observing Resources in CoAP", IETF. draft-ietf-core-observe-02 (work in progress), March 2011.

[74] Z. Shelby "CoRE Link Format", IETF. draft-ietf-core-link-format-03 (work in progress), March 2011.

[75] Z. Shelby, K. Hartke, C. Bormann "The Constrained Application Protocol (CoAP)" Internet Engineering Task Force (IETF) Request for Comments: 7252. ISSN: 2070-1721. June 2014

[76] E. Rescorla, N. Modadugu, RTFM, Inc. "Datagram Transport Layer Security (DTLS)" Request for Comments: 4347, Network Working Group Stanford University, April 2006

[77] S. Kent. "IP Encapsulating Security Payload (ESP)" Request for Comments: 4303, Network Working Group BBN Technologies, December 2005

[78] E. Rescorla, N. Modadugu "Datagram Transport Layer Security Version 1.2" Request for Comments: 6347, Internet Engineering Task Force (IETF) Inc. RTFM, Inc. Google. January 2012

[79] Matthias Kovatsch. "CoAP for the Web of Things: From Tiny Resource-constrained Devices to the Web Browser" *4th International Workshop on the Web of Things*. Institute for Pervasive Computing, ETH Zurich, Zurich, Switzerland. 2013 [Online]. Available: urlhttp://www.vs.inf.ethz.ch/publ/papers/mkovatsc-2013-wot-copper.pdf. [Accessed: Jun- 2013].

[80] Matthias Kovatsch. "Copper (Cu) CoAP user-agent - JavaScript CoAP Implementation", newblock Version 0.8.0. ETH Zurich, Switzerland. Oct 2011. [Online]. Available: http://people.inf.ethz.ch/mkovatsc/copper.php. [Accessed: 09- Jun- 2015].

[81] Matthias Kovatsch."Erbium (Er) REST Engine and CoAP Implementation for Contiki", ETH Zurich, Switzerland, 2015. [Online]. Available: url-http://people.inf.ethz.ch/mkovatsc/erbium.php. [Accessed: 03- Jun- 2015].

[82] Federal Communications Commission. USA. [Online]. `https://www.fcc.gov/`. [Accessed: 09- Jun- 2015].

[83] Antennadesignassociates.com, "Smith Chart 3.0", 2015. [Online]. Available: `http://www.antennadesignassociates.com/smith.htm`. [Accessed: 03- Jun- 2015].

Appendix A: Schematics and printed circuit boards

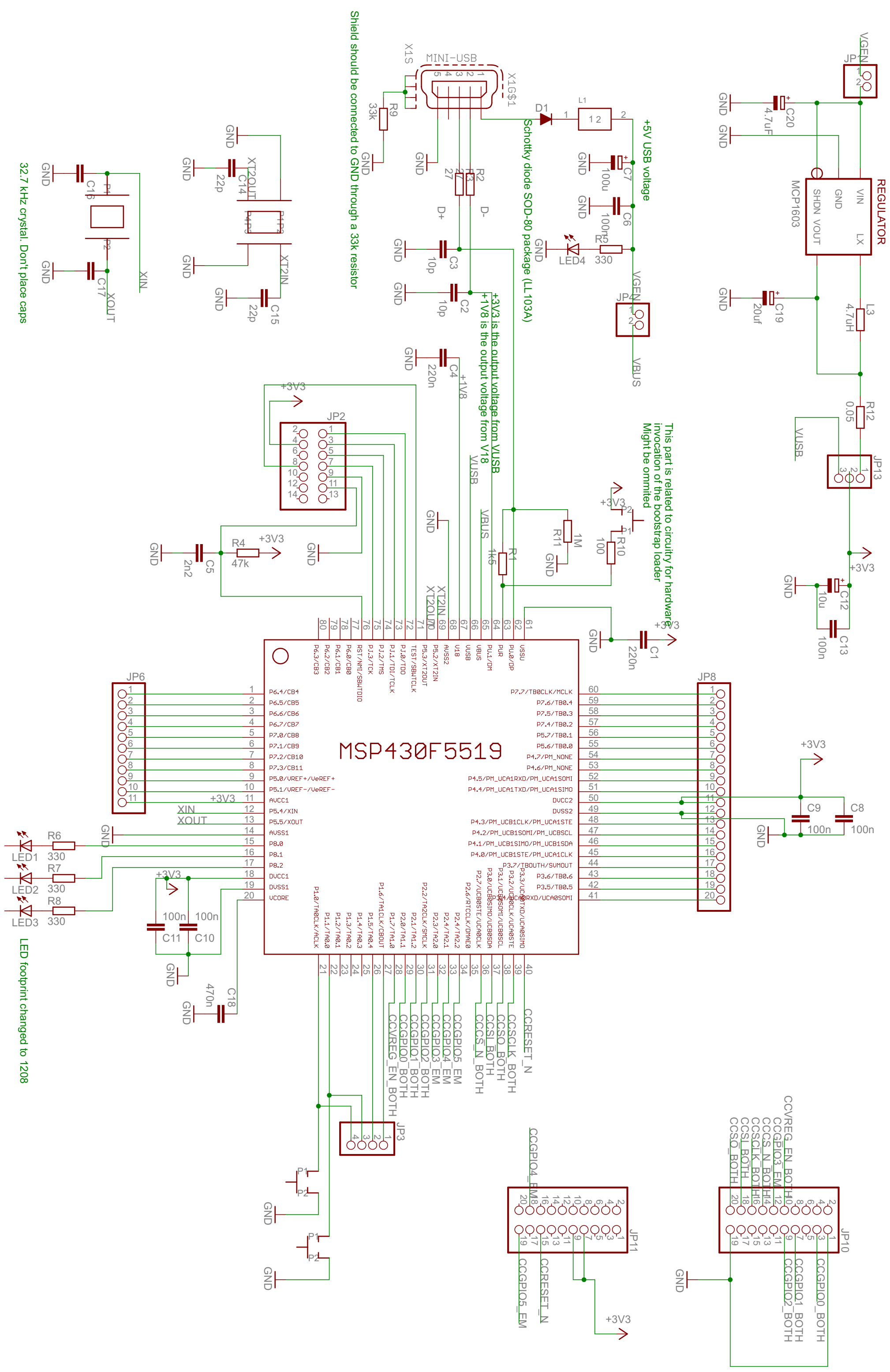

MSP430F5519
MINI-USB
REGULATOR
MCP1603
Shield should be connected to GND through a 33k resistor
Schottky diode SOD-80 package (LL103A)
+5V USB voltage
+3V3 is the output voltage from VUSB
+1V8 is the output voltage from V18
This part is related to circuitry for hardware invocation of the bootstrap loader
Might be ommited
32.7 kHz crystal. Don't place caps
LED footprint changed to 1208
JP1
JP2
JP3
JP4
JP6
JP8
JP10
JP11
JP13
GND
+3V3
VUSB
VBUS
XIN
XOUT
XT2IN
XT2OUT
LED1
LED2
LED3
LED4
CCRESET_N
CCSCLK_BOTH
CCSO_BOTH
CCSI_BOTH
CCCS_N_BOTH
CCGPIO5_EM
CCGPIO4_EM
CCGPIO3_EM
CCGPIO2_BOTH
CCGPIO1_BOTH
CCGPIO0_BOTH
CCVREG_EN_BOTH

Figure 1: Gateway schematics

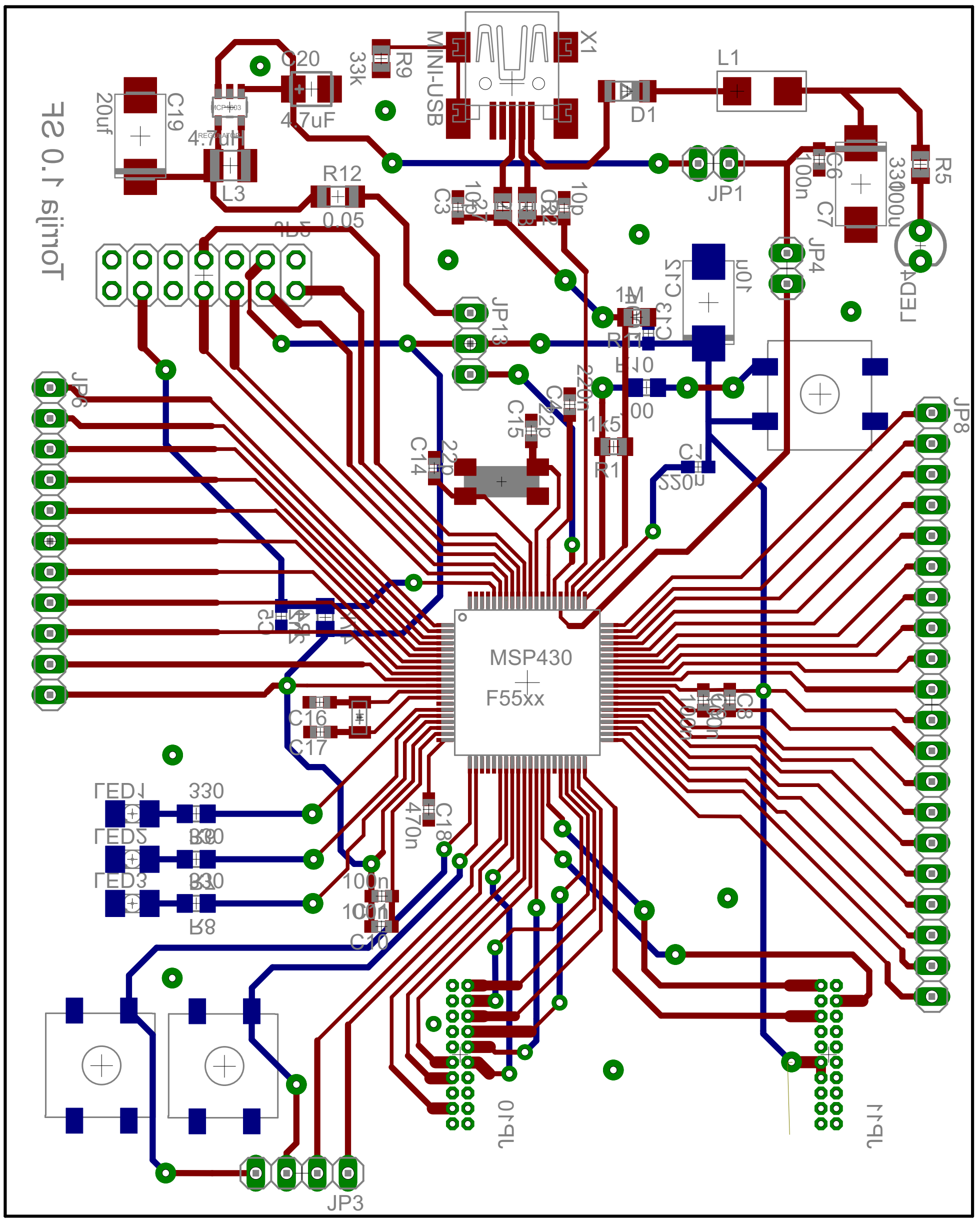


Figure 2: Gateway PCB

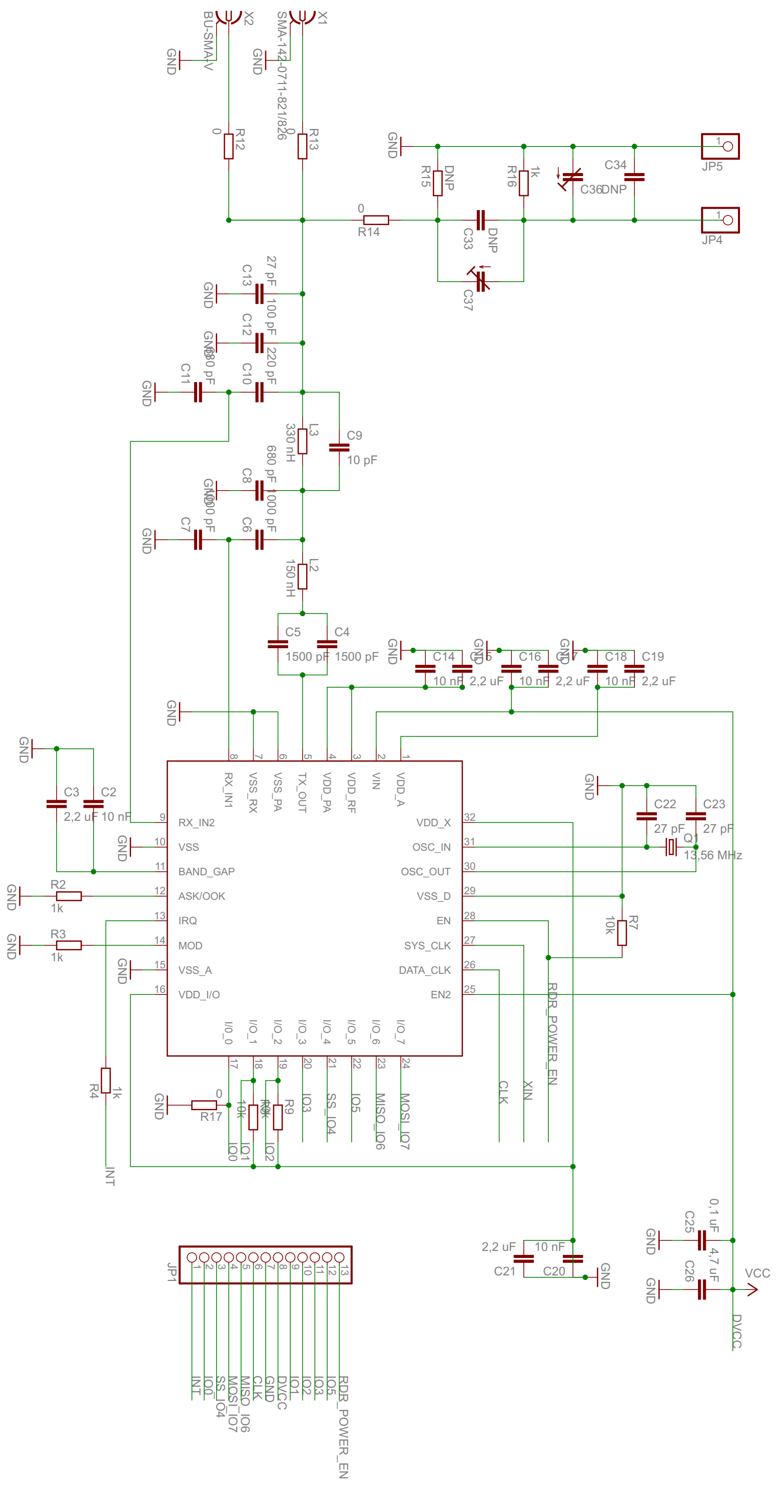


Figure 3: RFID schematic

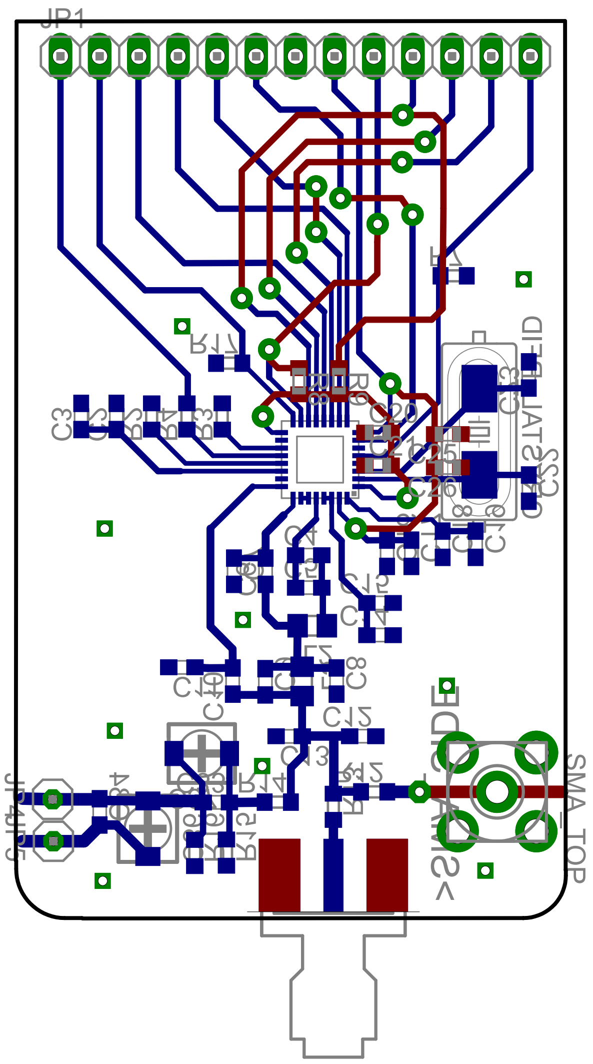


Figure 4: RFID board

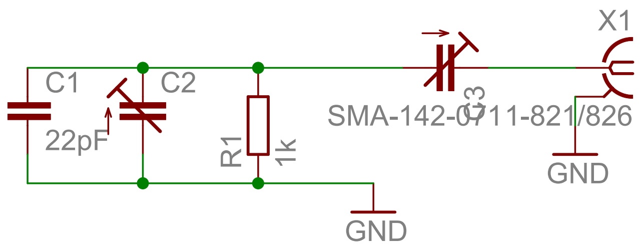


Figure 5: Antenna impedance adaptation circuit

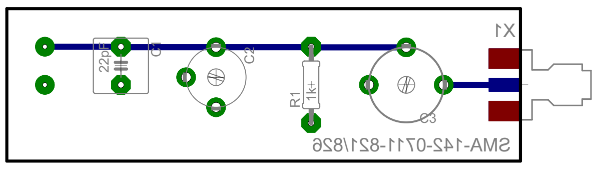


Figure 6: Antenna impedance adaptation board

Appendix B: Source code

The source code can be accessed from this remote repository:
https://github.com/serflosa/torrija_gateway