Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

MASTER THESIS

TITLE: Implementation of a Personal Area Network for biomedical measurements for Internet of Things (IoT)

MASTER DEGREE: Master's degree in Applied Telecommunications and Engineering Management (MASTEAM)

AUTHOR: Miguel Angel Fornell Sánchez

ADVISOR: José Polo

DATE: July, 5th 2018

Title: Implementation of a Personal Area Network for biomedical measurements for Internet of Things (IoT)

Author: Miguel Angel Fornell Sánchez

Advisor: José Polo

Date: July, 5th 2018

Abstract

The IoT paradigm is currently seeking to penetrate more and more into society and it seems that it will inevitably do so, with ambitious projects in both Smart Cities (Sentilo, CityOS) and private industry (Zolertia, SECE). As in each new paradigm, the abundance of protocols, solutions, and implementations end up converging in standardized and robust solutions. This is the case of the Contiki operating system developed by Adan Dunkels or Riot developed by the union of several research fronts. Also, protocols adapted for embedded wireless networks such as uIP, Rime or the established standards 6LowPAN, IEEE 802.15.4. The convergences aim to satisfy problems that the Internet in its state of the art no longer contemplates, the Internet assumes characteristics that a restricted network sees as challenges to overcome. Among these challenges, we can highlight security, limited energy, fragmentation of packages, discovering neighbours efficiently and the absence of multicast that, technically possible, would be a waste of energy. For this work, a scenario has been put into practice that although at the moment the capacities of the nodes do not satisfy the demands of a commercial implementation, the technological improvements and the low cost of the elements that are used, will make it possible that absolutely everything what can be measured of the human body is recorded and analysed by algorithms within the Big Data. Using the Contiki operating system and the Zolertia motes, different configurations for the acquisition of signals have been tested, these analog signals measured directly from the human body require certain minimum bandwidth that must be met, on the other hand, the use of radio and core increases the use of the battery. An algorithm has been created for analyse the errors due to force the transmission to the maximum of the capacity, verifying the real limitations. Finally, a prototype of architecture is implemented and tested using artificially generated signals and also real signals, these signals are ECG waveforms.

I gratefully acknowledge to my Family, especially my aunt Dayse Sánchez

CONTENTS

CHAPTER 1. INTRODUCTION

1.1. Motivation

The bio-signals provide us with valuable information. If these data is inserted into the cloud, they could provide much more information for research in different fields. The availability of devices capable of autonomously monitoring biological signals has increased in recent years, but integration with new technologies has been slowly evolving.

It would be useful to measure the main functions of the body and ensure the first assistance and provide a faster response to emergencies, at least in the elderly during their daily lives. Early detection of heart problems would only bring benefits for both the patient and the medical staff, as it would help reduce waiting times and overcrowding in the hospital. There are a variety of applications in medicine, but over the years these devices must also be integrated into fashion and, as far as possible, go unnoticed. And not only in the field of health, also in athletes or recreational sports activities where there is a great demand to accurately control heart rate, step count, oxygen saturation and blood pressure.
This increases the use of integrated sensors in clothing. The consumption of energy is another problem to be overcome. The collection of energy through small solar panels to continuously support the measurement devices could be an option, other advances have been proposed in the use of body heat to generate energy, however, for this solution in the short term, it would be limited at a time of short use. For this reason, the WSN with 6LowPAN play an important role in the future next generation of wearables.

1.2. Thesis Overview

The paper is organized as follows. The first chapter presents the key concepts and breaks down the basic theory to encompass the project. Chapter 2 focuses on the solution of the problem to be solved, a general outline based on open source. The proposed design goes through different layers that try to abstract the model of an IoT scenario. Chapter 3 shows in detail the results obtained from both the tools developed and the algorithms developed. And in the last chapter the conclusions are established and what remains open to future developments

1.3. Goals

The goal of this final project is to recreate an architecture based on wireless sensor networks from the acquisition of information to its final journey to the cloud. Establishing as reference only protocols, standards, codes, free operating systems.

Verify the current capabilities of the Zolertia device to deal with analog signals that require a more significant bandwidth than was designed.

Create tools that allow the visualization of biomedical measurements to store the data and obtain access to this data from the Internet, going from the WSN to the WAN, recreating an IoT scenario.

1.4. Biomedical measurements

Measurements of the electrical signals in living organisms have had a development based on the advance of technology. Measuring everything that surrounds us has characterized man in his search for answers, with respect to the bio-signals of the body, this measurement proposes certain technical challenges to be overcome. These analog signals have been studied for years and at this time we can distinguish them by their bandwidth, their source and even how to filter them in such a way that we have a real approximation to the signal we want to measure (Figure 1.1). In this work will not delve into the measurement and instrumentation systems since that is a deeper issue, however the main biological signals from the medical point of view It is briefly discussed below.

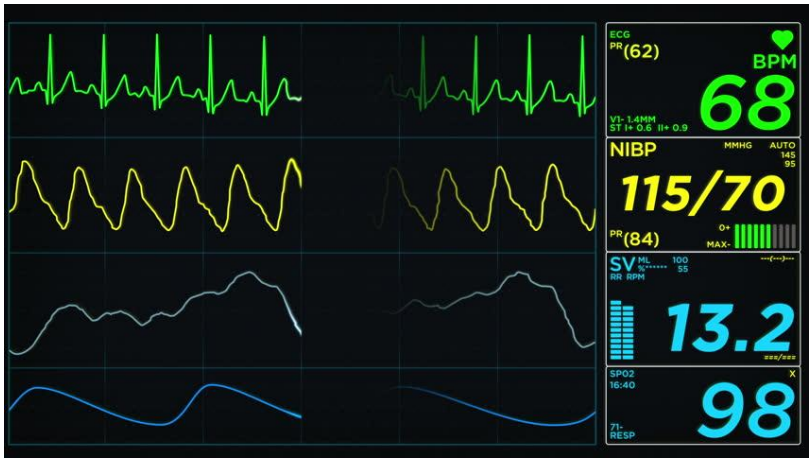

Fig. 1.1 Medical monitor Screen.

1.4.1. ECG Waveform

It is the graphic representation of the electrical activity of the heart as a function of time, which is obtained, from the body surface, usually in the chest. Analysing these signals help to diagnose some cardiovascular diseases, metabolic disorders, possible sudden cardiac death, the cardiac cycle as well. The signal is the result of a process carried out by the heart, with an electrical impulse generated in the sinus node what is transmitted to the whole heart by the conduction system, from the atrial cells to the ventricular cells. The sinus stimulus depolarizes the atria, starting from the right lateral part of the right atrium and following an anti-clockwise path (counter clockwise), first depolarizing the interatrial septum and ending in the left atrium. This results in a signal composed of the sum of several signals that for its practical study has

been standardized the placement of reference points for its measurement. In these reference points is where the electrodes are placed. Time ranges have also been established for the ECG waveform (see Figure 1.2), which can be analysed meticulously once digitized.

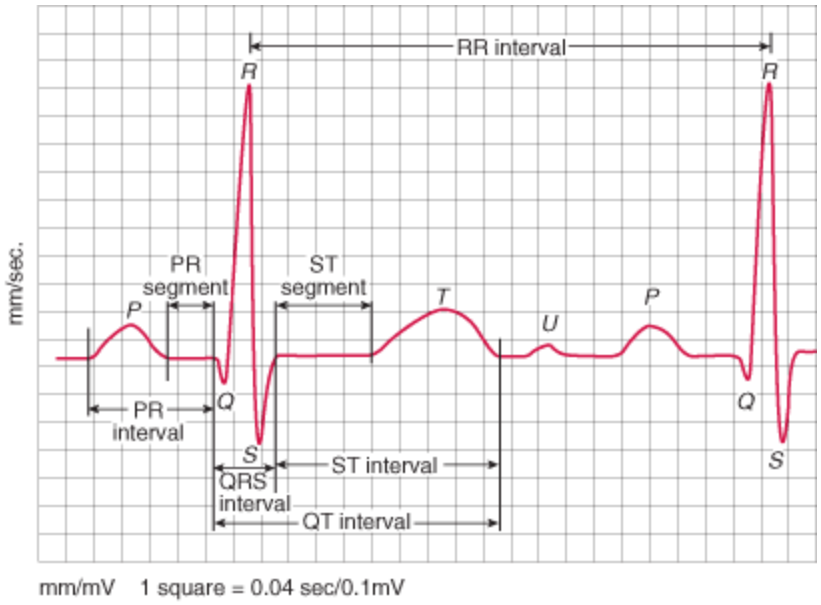

Fig. 1.2 ECG waveform intervals

1.4.2. Physiological signals

There are several signals besides ECG, all with a minimum bandwidth required. For example, the case of temperature is a physiological signal generated by the body that does not vary so fast, so with a bandwidth of 0.1 Hz is sufficient. Below, in Table 1.1 , other signals generated by the body that can be monitored with the appropriate sensors, describing the minimum bandwidth required to acquire and digitize them.

Table 1.1 Other Physiological signals

Temperature	32 – 42 °C	dc – 0.1 Hz
Phonocardiogram	80 dB above 100 µPa	5 Hz – 2 kHz
Respiratory rate	2 –50 breaths/min	0.1 – 10 Hz
Blood pH	6.8 – 7.8	dc – 2 Hz
Gastric pH	3 – 13	dc – 1 Hz
GSR	1 – 500 kΩ	0.01 – 1 Hz

1.4.3. Bio Sensors

Many portable devices and sensors are currently able to acquire the ECG to track the heart activity, apply filtering and signal conditioning . Those systems, generally, comply with a monitoring standard that sacrifices some distortion in favour of a better rejection in front of noise, interferences and artifacts.
These sensors have the challenge of guaranteeing the clinical quality of the measurements and being easy to install for the common user, before going on the market and integrating with real solutions. There are already in the market garments that do this with a good relationship between quality and price, and everything indicates that their cost will be reduced every time, opening new fields for the management of these signals. Other challenges in the sensors are given by the variation between the signals of everyone, besides that said signals vary in the same individual depending on the way in which the electrodes are placed, for example, in the case of ECG signals. It is for this reason that more and more new sensors with the ability to filter the signal for optimal integration will be necessary. In Figures 1.3 and 1.4 some corporal biosensors that have been integrated into the market are described.

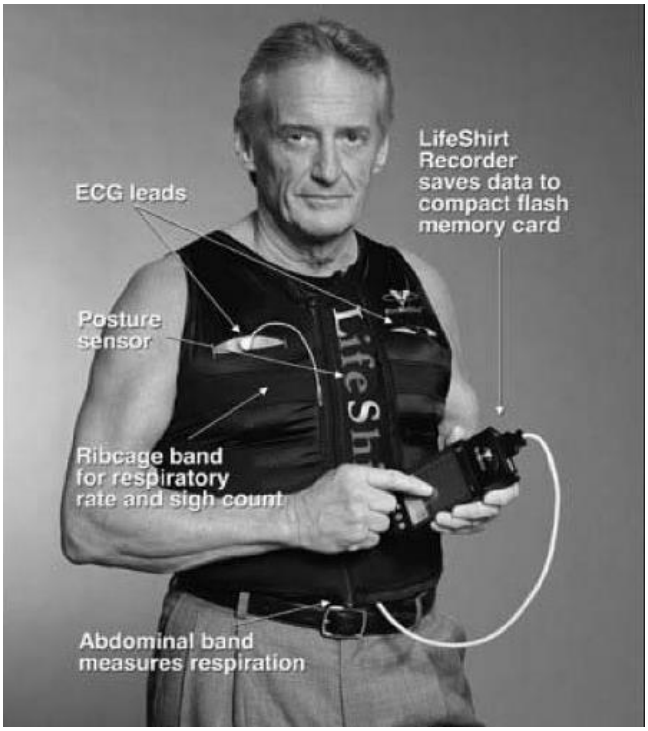

Fig. 1.3 LifeShirt ™, bio-sensors integrated in a garment

The LifeShirt is a lightweight, machine washable and comfortable shirt with built-in sensors. To measure respiratory function, sensors are woven in the shirt around the thorax and abdomen. A single-channel ECG measures heart rate and a three-axis accelerometer records posture and activity level. It is also possible to measure oxygen saturation in the blood and other physiological parameters.

Fig. 1.4 Healthcare wearable device. The Samsung Simband ™ can measure several physiological signals.

The Simband is no regular smartwatch. Instead, it's a sensor-packed wristband with a big curved screen that promises to track everything that the body does and use that data to improve health.

CHAPTER 2. IoT OVERVIEW

IoT refers to physical objects that are found in daily life that have sensors, integrated circuits and connectivity that allow them to exchange data, either between themselves or with the Internet. This current paradigm tries to digitize everything that surrounds us, with the purpose of automating processes and improving the quality of life of its users. The nodes that sense information form wireless networks are known as WSN (Wireless Sensor Network), these networks must support IP connectivity to connect to the Internet, but since their computational resources are limited, 6LoWPAN connectivity is used in the most cases.

2.1. Wireless Sensors Networks

A WSN is a network of small and inexpensive sensor nodes monitoring physical or environmental conditions and communicating among them using radio signals. The scope of WSN is the coordinated collection of data transmitted to a central location. Sensor nodes are essentially small devices with extremely basic processing power, limited memory/storage capacity, and low energy consumption. They create a WPAN (Wireless Personal Area Network).

2.1.1. IEEE Standards

The "task group 4" of the "IEEE 802.15 working group" focused on finding a standard for WPAN with low rates, efficient and support mechanisms for energy saving, among these the most prominent is the one that uses low time intervals consumption or duty cycling.

This standard has gone through several evolutions over time, its basic version was defined in 2003, adding the physical layer 3 years later. For 2009 China and Japan would create their own adaptations, but it is not until 2012 that mechanisms are added at MAC level to make it strong against interference and significantly decrease energy consumption.

The improvements in the wireless networking technologies and the integrated electronic circuits have allowed the advancement in the Wireless Body Area Network. WBAN offers many applications in remote health monitoring and medicine. IEEE 802.15.4j and IEEE 802.15.6 are standards for the medical purpose. It allows the integration of intelligent and miniaturized sensor nodes in or on a human body to monitor the human body functions. It has great potential to make a huge transformation in the future of the medical industry.

2.1.1.1. IEEE 802.15.4j - Medical body area networks (MBAN)

MBANs (Medical Body Area Networks) are made up of low-power body sensors that wirelessly transmit patient data to telemetry systems, which then send the information to the cloud or to local systems. Applications usually include wireless monitoring of blood pressure, temperature and respiration rate. FCC's decision in 2012 [8] to assign spectrum for wireless networks made with medical sensors for the body was praised by the private company, But some experts worry that the lack of rules governing interoperability in the bands can slow progress. Most of the current MBAN solutions are implemented in the industrial, scientific and medical 2.4 GHz (ISM) band, which is increasingly crowded, and uncontrolled interference can limit the large-scale deployment of MBAN applications doctor in the future. This assignment was of 40 MHz of spectrum at 2360-2400 MHz (see [9]). On security issues and band assignments. There is nothing that prevents someone from blocking all the new spectrum in a given hospital.

2.1.1.2. IEEE 802.15.6s – Wireless Body Area Network (WBAN)

Wireless Body Area Networks (WBAN) has emerged as a key technology to provide real-time health monitoring of a patient and to diagnose and treat many life-threatening diseases. WBAN operates in close vicinity to, on, or inside a human body and supports a variety of medical and non-medical applications [10].
The purpose of the IEEE 802.15.6 was to define new Physical (PHY) and Medium Access Control (MAC) layers for WBAN. The selection of the PHYs (frequency bands) were one of the most important issues. Generally, the

available frequencies for WBANs are regulated by communication authorities in different countries.

Safety is a priority in most networks; few studies have been conducted in this area for WBANs. Since WBANs are limited by resources in terms of power, memory, communication speed and computing capacity, the proposed security solutions for other networks may not be applicable to WBANs. Confidentiality, authentication, integrity and freshness of data along with availability and secure administration are the security requirements in WBAN.

2.1.2. 6LowPAN

6LoWPAN (IPv6 over low-power personal area wireless networks), is a protocol designed to allow the transmission of IP packets in networks based on the IEEE 802.15.4 standard. It is also designed for applications with a very low data flow and very limited resources.

2.1.3. Physical Layer

The IEEE 802.15.4 standard defines the physical layer (PHY) for low-speed wireless data connectivity with moving devices, with very limited battery consumption. These requirements, in addition to 6LoWPAN are also applicable by other protocols such as ZigBee ™, Wireless HART, SmartLink and others.

Figure 2.1 shows the process of mapping bits of the physical layer from higher layers. These bits corresponding to the PPDU (Physical Protocol Data Unit) are sent to the medium once mapped and properly modulated.

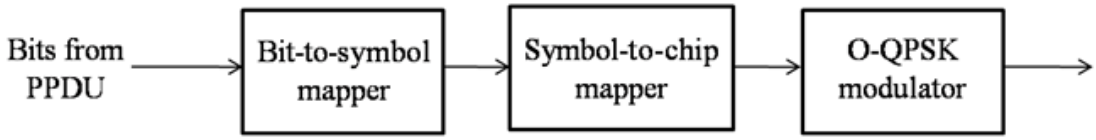

Fig. 2.1 Physical Layer of 6LowPAN

The payload is defined as PSDU (Physical Service Data Unit) that has a maximum size of 128 bits and are established by the 7 bits of the frame length section in the header. The "Preamble" is also defined in the header. In each frame the information of the RSSI (Radio Signal Strength Indication) and the LQI (Link Quality Indication) are also provided, which serve to establish the best routes of links or to control the channel's metrics. The following figure 2.2 shows a schematic and general view of the PHY PPDU format and the included fields.

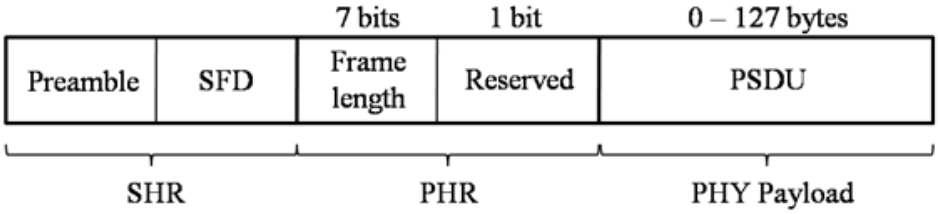

Fig. 2.2 Frame distribution

Synchronization Header (SHR): It consists of the preamble sequence followed by the start of frame delimiter (SFD). The preamble sequence is defined to be 4 bytes (i.e. 8 symbols) of 0x00. The SFD is one byte set to 0xA7.

PHY Header (PHR): The frame length field defines the number of bytes in the PSDU.

PHY Service Data Unit (PSDU): The MAC frames (MPDU) are passed to the PHY as the PHY service data unit (PSDU), which becomes the PHY payload.

The 2.4 GHz band has channels established from 11 to 26 with spacings of 5 MHz, usually with O-QPSK modulation and taking advantage of the spread spectrum by direct sequence or DSSS, unlike the bands 868/915 Mhz using BPSK modulation. The 2.4 GHz band has a sensitivity of -84dBm, while the 868/915 MHz bands are less sensitive at reception, above -90 dBm.

To determine the maximum distance we can transmit, we rely on the relationship between powers, equation 2,1. The power transmitted less the power lost in the medium must be equal to the power of reception.

$$P_{Tx} - P_{L(d)} = P_{Rx\,min}\ (Receiver\ Sensitivity) \quad \textbf{(2,1)}$$

P_{Tx}: $Power\ Transmitted$
$P_{L(d)}$: $Power\ Lost$

For instance, with a transmission power of 0dBm, a theoretical distance with line of sight of 50.8 m can be achieved, in 2.4 GHz band. Which is reduced in case of having obstructions or interferences by multipath or by channel saturation

2.1.4. MAC Layer

Each successive protocol layer is added to this structure with headers and footers specific to the layer. MAC frames, for example, are passed to the PHY as the service data unit PHY (PSDU), which becomes the payload PHY.
The different fields of the PPDU appear in a fixed order and must be part of each package. The SHR and PHR format depends on the type of PHY used (see Table 1).

With several possible configurations, this layer represents the biggest challenge in wireless protocols, access to the medium can be defined by beacons or without them, using beacons it's possible to configure it with time slots, very large frames or synchronized. But in most cases the nodes will be communicated in an asynchronous manner so that for this first the medium is sensed, then the packets are sent to avoid collisions, this way to access the medium is inherited from the CSMA / CA protocol, which together with Acknowledgment mechanisms gives reliability to the transmission. This layer also integrates security mechanisms.

2.2. Open Source Operating Systems

Operating systems are useful in low-end devices to facilitate the development and portability of IoT applications. Some operating systems are tied to a hardware vendor and only target at a single hardware architecture. Contiki is an outstanding free and open source operating system, although not the only one. Next, the most used options are described in terms of OS for IoT.

2.2.1. TinyOS

It is an open source operating system based on components for wireless sensor networks. It is written in the programming language nesC as a set of tasks and processes that collaborate with each other under the BSD License. It is designed to work under the strong memory restrictions that exist in sensor networks. Actually, It is still developed by a consortium led by the University of California at Berkeley in cooperation with Intel Research.

2.2.2. RIOT

It is an operating system designed for a low memory footprint, high energy efficiency, real-time capabilities, a modular and configurable communication stack, and support for a wide range of low power devices. RIOT provides a microkernel, utilities such as cryptographic libraries, data structures (bloom filters, hash tables, priority queues) or a shell, different network stacks. It’s possible to adapts to a wider range of architectures (8 to 32 bits) compared to most other OSes. Also support cross-platform hardware according to [11].

2.2.3. OpenThread

It is an open source implementation of the Thread network protocol. Nest launched OpenThread to make the technology used in Nest products more widely available to developers to accelerate the development of products for the connected home.

2.3. Contiki

It is an open source operating system for WSN developed by a worldwide team of developers, led by Adam Dunkels, who developed the basic core and most of the main functions. Contiki is developed for use in a number of small systems, ranging from 8-bit computers to integrated systems on microcontrollers, including sensor network nodes. The Contiki operating system supports different architectures and hardware platforms. Exist many examples for Internet of Things (IoT) implementation with Zolertia RE-Mote nodes to be able to start using the platform easily and conveniently. Account with a simulation mode called Cooja where It's possible to run simulations without the need of hardware. It includes the TCP / IP stack and only requires a few kilobytes of code and a few hundred bytes of RAM.

Contiki processes use proto-wires, an abstraction mechanism designed to provide a sequential programming style on the event-oriented kernel. The communication between processes is done through the message passing technique, which is implemented through the kernel's event system.

The network protocol stack of Contiki, known as (NETSTACK), organizes the network modules into a protocol stack that covers all the traditional OSI layers. Each of these layers has different configurations in which we will not enter into detail. These layers are defined by their respective primitives. The following Table 2.1, relates each layer with its primitive in C, starting with some applications, like COAP or HTTP until the lower layer of radio that abstracts the internal registers of the radio module CC2538.

Table 2.1. Contiki Network Stack

LAYERS MODEL	*PRIMITIVES*
APPLICATION	*coap.c websocket.c*
TRANSPORT	*udp-socket.c tcp-socket.c*
NETWORK	*uip6.c, rpl.c*
ADAPTATION	*sicslowpan.c*
MAC	*csma.c*
RADIO DUTY CYCLING	*nullrdc.c , contikimac.c*
FRAMER	*framer-802154.c*
RADIO	*cc2538.c*

2.3.1. Network environment

The communications stack can be adjusted to work in IP mode, being able to be IPv6 or its shortened version of 16 bits, as well as non-IP protocol called RIME. These communication paradigms are briefly discussed below.

2.3.1.1. RIME

Rime tries to abstract the particularities that a WSN poses, such as the automatic discovery of neighbours, fast re-adaptation of the routes, among others. Defined as a Lightweight Layered Communication Stack for Sensor Networks is organized in layers as shown in Figure 2.3, with headers very small in all the layers that help to outgoing messages, a few bytes each. The thin layers in Rime enable code reuse within the stack. For example, reliable communication is not implemented in a single layer but divided into two layers, one that implements acknowledgments and sequencing, and one that resends messages until the upper layer tells it to stop. The latter layer “stubborn” is not only used by reliable protocols but also by protocols that send periodic messages such as neighbor maintenance for routing protocols and repeated transmission of messages in Rime’s network.

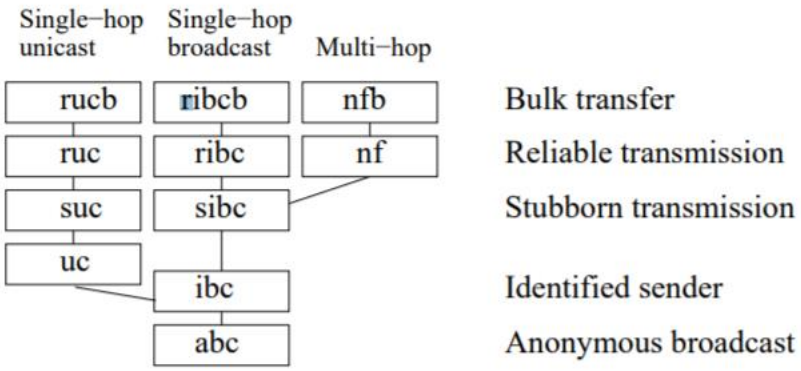

Fig. 2.3 Rime Stack

The lowest level primitive in Rime is anonymous best effort broadcast, abc. The abc layer provides a 16-bit channel abstraction but no node addressing; it is added by upper layers. The identified sender best-effort broadcast, ibc, adds a sender identity header field and the unicast abstraction, uc, adds a receiver header field. An underlying MAC or link layer may choose to implement parts of the Rime stack, such as the abc, ibc, or uc layers, in hardware or firmware.

2.3.1.2. uIP

The uIP is an open source TCP / IP stack designed for use with 8 and 16 bit microcontrollers. It holds a BSD (Berkeley Software Distribution) license and was subsequently developed by a large group of workers. Despite being small and simple, uIP do not require their peers to have complex, full-size stacks, but can communicate with peers running a similarly light-weight stack. The code size is on the order of a few kilobytes and RAM usage can be configured to be as low as a few hundred bytes.

The uIP make use of Rime and provides a socket-like API for use by applications called proto-sockets. Both built-in and user applications are run over Contiki using a lightweight thread model called protothreads [5].

2.3.2. Radio Duty Cycling

Low-power wireless devices usually keep their radio transceivers turned off as much as possible to achieve low power consumption. If they do not, a sensor node runs out of batteries in a matter of days. But when the radio is off, the node can't send or receive messages. For this purpose, different configurations have been created for the radio service cycles. The following are the most important:

ContikiMAC
It is a protocol based on the principles of low power listening in RDC layer, but with better energy efficiency. It uses a simple time scheme to allow its activation mechanism to be highly efficient, phase locking mechanism to make transmissions efficient, and fast sleep optimization to allow receivers to sleep quickly when faced with false radio interference. It is a default radio service cycle mechanism in Contiki

X-MAC is another protocol that could be used in the RDC layer, defined as a short-preamble protocol from 2006 that was ported to ContikiOS. It reduces energy consumption and maintains good network conditions.

LPP is another option for RDC layer. It is based on the low-power test protocol (LPP) but with improvements that improve power and provide mechanisms to send transmission data.

Nullrdc is the last configuration of RDC layer. It works as a pass layer that only transmits a packet and returns the results of that transmission (success or collision) and does not save energy.

2.4. Hardware Used

2.4.1. Zolertia RE-MOTE

The RE-Mote is a hardware development platform that allows the development of IoT applications. It is characterized by offering ultra-low energy consumption thanks to its core Zoul (see Figure 2.4). Zoul It is based on a SoC of Texas Instruments, an CC2538 ARM Cortex-M3 that has an integrated radio based on IEEE 802.15.4 at 2.4 GHz and that works at up to 32 MHz with 512 KB of programmable flash and 32 KB of RAM.
It also includes a Texas Instruments CC1200 868/915 MHz RF transceiver, so transmitting by double band is possible.

Fig 2.4 Zolertia ™ Zoul Remote B.

2.4.2. CC2538

CC2538 belongs to the CC253X family and provides solutions for a wide range of applications. It has the ability to connect to networks in the 2.4 GHz band, works very well in the upper layers of the IEE802.15.4 protocol and solutions such as Z -Stack of Zigbee. It is suitable for wireless sensor networks, 6LoWPAN and WirelessHart. Its voltage regulator allows working at 1.8 V, it has three Low-Power modes for low power applications. Use a specific timer for the IEE802.15.4 protocol or another time-slotted protocol created by software. A timer for the Sleep mode that uses the internal clock of 32 KHz, this RC oscillator works for two of the three working modes and turns off completely in the PM3.

On the most outstanding peripherals we can say that it has a random number generator, an internal module to encrypt and decrypt based on AES-128 that are suitable for security purpose, also two interfaces USART and I2C.

The ADC of this SoC supports 7 bits which gives it a bandwidth of up to 30 KHz, but with its maximum resolution of 12 bits the bandwidth is only 4 kHz due to the conversion times (see table 2.1). The inputs can be selected as single-ended or differential. The reference voltage can be internal, AVDD or an external signal of single or differential termination. The ADC also has a temperature sensor input channel. The ADC can automate the process of periodic sampling or conversion through a sequence of channels.

Table 2.1. Conversion times

Conversion time	7-bit setting	20	µs
	9-bit setting	36	
	10-bit setting	68	
	12-bit setting	132	

The CC253x family of devices provides a radio transceiver compatible with IEEE 802.15.4. The RF stage is controlled by the analog radio modules. The interface between the MCU and the radio allows to issue commands, read the status and automate and sequence radio events. The radio is also capable of filtering packets and recognizing addresses independently of the MCU.

CHAPTER 3. DESIGN OF THE SOLUTION

3.1. Design of the solution

The Internet of Things can potentially have applications in almost any area. These systems are responsible for collecting information in different environments, from natural ecosystems to buildings and factories. An IoT network has the ability to connect embedded devices with limited CPU, memory and power capabilities. However, many manufacturers are still waiting to see what to do and when to start. This is an advantage for small businesses since they can get ahead and create new designs adapted to the Internet of things.

The open source systems allow to re-configure and customize all the layers of design and also allows the integration of new developers. For the design of an IoT system, it is important to take into account that the architecture is designed from the bottom up, starting with the sensors that are usually devices that sense events and in real time.

For our design, we consider a WSN network, a border router, a server, a database and finally a Front End for the presentation of information. The WSN network is raised by Zolertia sensors.
The analog signals are acquired by the remote Zolertia using ADC acquisition embedded in the SoC, then, they are sent to the border router via uIP using UDP as the transport protocol. The Zolertia node sees the border router as a server to which it is sending data using an IP compressed for Wireless Sensor Networks, but first, the udp6 socket must be settled up. An udp6 socket is an UDP service over IPv6. This means that in the border router, a service is listening to the selected port. After this process the remote transmitter and the UDP session is lifted.

While there are other operating systems for WSN such as RIOT and OpenWSN, the manufacturer of the Zolertia nodes has a GitHub page well documented, in which it has developed projects with examples in Contiki. and given the extensive documentation and its stability makes it our option to choose over the other operating systems.

3.1.1 Architecture

Figure 3.1, shows the different stages of the proposed scenario. Devices that create the WSN network connected to bio-sensors, a border router, various services that allow the storage and processing of other applications to a higher programming environment. In this way the bio-signals can be stored and processed by programs in operating systems with more resources than those offered by restricted resource devices.

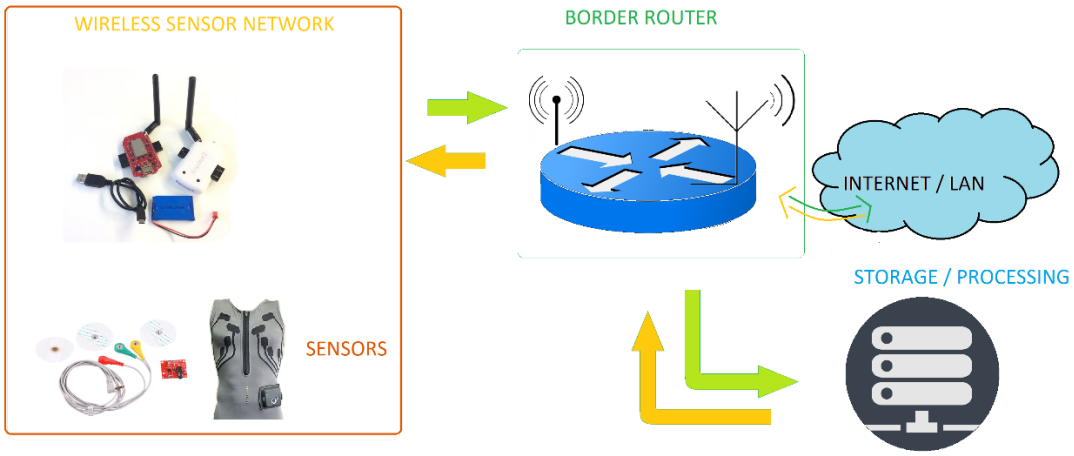

Fig. 3.1 Design of the solution

3.1.1.1 WSN

It is the network formed by the Zolertia teams, a client-server link is established between two Re-Mote equipment. Both the client and server code are developed in C, using the Contiki operating system. At the IPv6 network level, hosts can talk different services. A service that comes preconfigured is the echo request or better known as ping. From the border router, it is possible to ping any node in the WSN.

3.1.1.2 Border Router

It is the combination of one Zolertia Re-Motes and a virtual network interface that provides us the Raspberry Pi. This virtual interface has all the TCP / UDP sockets available and is directly connected to the serial USB interface of the Raspberry Pi.

3.1.1.3 Storage

The information of the bio-signals is stored in a database, which is managed by a non-relational database service. There are several options for non-rational database managers, or also called NoSQL, such as MongoDB, Cassandra and Dynamo, to name the most used. Due to MongoDB is open source and has support in many languages of programming as C, python, and PHP, it is chosen for the design.

3.1.1.4 Web Service

The information needs to be presented in a web page, which can be hosted on any server. The Raspberry hosts the server that can be reached by local

networks or with the correct architecture, by external networks such as the Internet.

3.1.2 Sensors

There are several sensor types in the market, but the scope of this work does not lie in the sensors or analyse their performance in detail. Two types of sensors are implemented, real and virtual. For to test the design with a real sensor. A module designed by DuinoPeack was chosen and consisting of an ADB8232 IC with its corresponding electronic configuration, It is used to sense the ECG signal. It has three inputs that correspond to signals RL (Right Leg), LA (Left Arm) and RA (Right Arm). An audio jack as input, 3 cables for electrodes and operates at 3.3 V, so its output operation range goes from 0 to 3.3 V. Analog signal is sampled and then it has added an offset to keep it within these operating values. ECGs can be extremely noisy, the AD8232 single-wire heart rate monitor acts as an operational amplifier to help get a clear signal of the PR and QT intervals with ease.

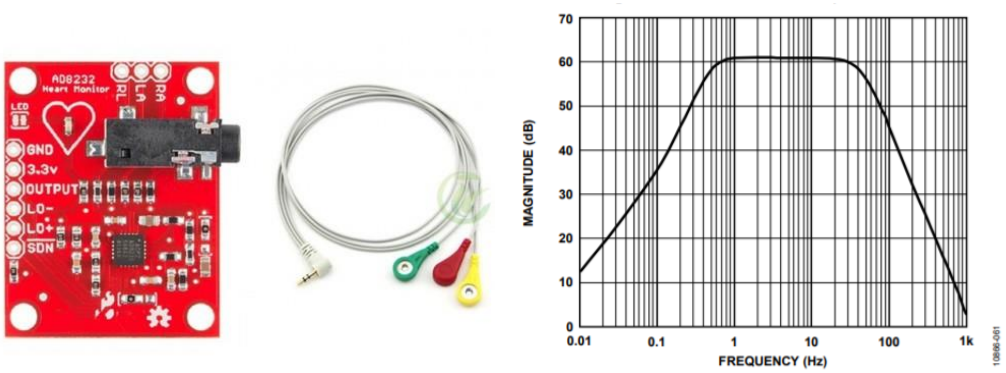

Fig. 3.2 a) ECG sensor module
b) Frequency Response of ECG sensor Module.

The AD8232 extract, Figure 3.2 a, amplifies and filter small biopotentials in the presence of noisy conditions. To obtain an ECG waveform with minimal distortion, the AD8232 is configured with a two-pole high-pass filter of 0.5 Hz followed by a two-pole low-pass filter at 40 Hz (Figure 3.2 b). A third electrode is driven for optimal rejection in common mode. It's important to clarify that this module is not a medical device and is not intended to be used as such.

3.1.3 Bio-Signal Generator

To establish the virtual scenario analog signals were generated by an Arduino module, making use of its PWM modules that allow a fast variation of the Duty Cycle. This signal generated with PWM passes through low pass filters and as a result the analog curves that try to approach an ECG signal already properly treated, filtered and amplified, are achieved. The signals generated did not allow the rapid variation expected and its limited processor was not optimal to generate different types of analog signals. For this reason it was decided to use the Raspberry pi at the end to generate these signals, for which a previous algorithm in C was adapted but in Python now. Using the GPIO library for

python, the integration was simple and the performance was considerably higher.

To perform tests, it is convenient to have isolated reliable signals as input. Although it is possible send simple sinusoidal signals and then vary the frequency to analyse see their response, It's difficult to infer information of interest about them. In the case of ECG signals, the curve is divided into several segments which have their pre-established times. By studying the above, a developer could analyse the information storage in the database with algorithms that allow the identification of the times of the segments, which in turn will allow synthesizing information of interest by medical professionals or for the end user. Eventually, the flow of information will create patterns in the data, as well as establishing normality thresholds that allow the identification of anomalous behaviours or performance indicators.

One hundred points that make up the pre-set curve are read, which are defined in an array of numbers in floating point format. The GPIO pin to be used is defined, also the PWM frequency. Then an infinite loop is established where each point of the previously defined array has to be read consecutively. It is convenient to place a small delay of time between the reading of each data to give the respective frequency to the periodic signal, for the case of 1 Hz a delay of 0.01 seconds has to be placed between each reading.

To model an analog curve is easy to perform a plot of points of the voltage values. Any table processor such as Excel can be used, once the points are added, this must be transformed to the respective duty cycle values multiplying by a factor When the analog signal is under designing stage, It is important take into a count that this must have an operating range that ranges from 0 V to 3.3 V for our case, so the points to be plotted should not be outside this range, see Figure 3.3. This range is extrapolated to a range that goes from 0% to 100% where 0.33 V represents 10% of the duty cycle, for instance. It should be noted that the library in Python allows to establish duty cycle values quite accurately, but it loses accuracy and stability as the PWM frequency increases.

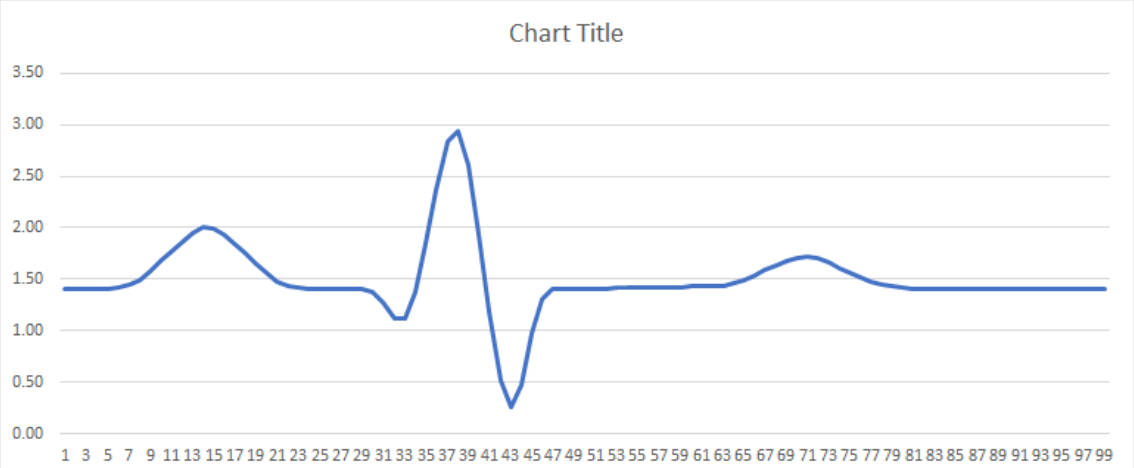

Fig. 3.3 ECG waveform generated

Using an appropriate filter, it is possible to visualize this signal with any device that has an ADC sampler, given that already exist familiarity with the Arduino modules and considering that the platform integrates a plotter by serial printing, it was possible to recover that analog signal established in the different GPIO pins of the RaspberryPi without the need for an Oscilloscope, this greatly facilitates the development. The recovered signal after the respective filtering in the following images sequences.

For this signal with an LPF filter with a cut off frequency lower than 10 Hz, see Figure 3.4, it would be sufficient to obtain the curve established in the generator algorithm.

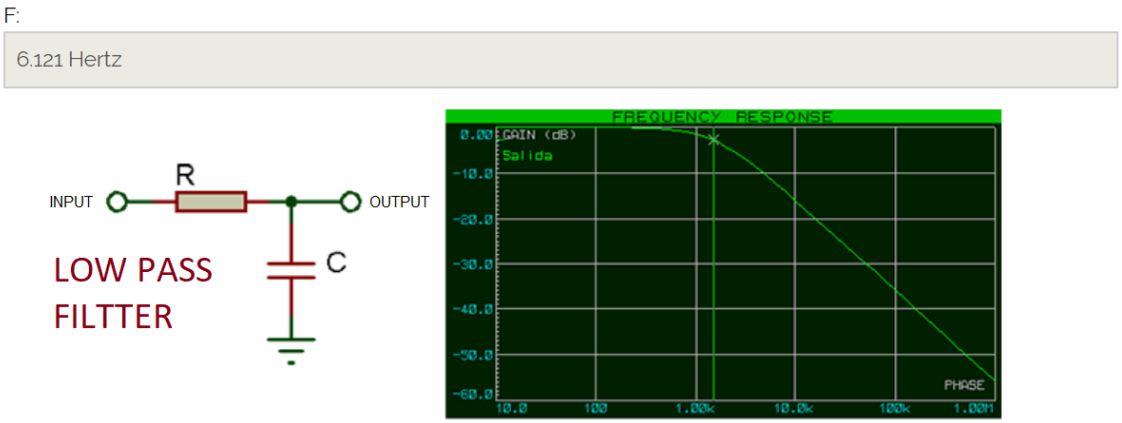

Fig. 3.4 First order passive LPF RC

The cut off frequency is set at 6 Hz with values of R = 2.6 K and C = 10 uF. The slope is 20 decibels per decade. At the cut off frequency, the output voltage is 3 dB (0.707 times) below the input.

The capacitor should be polarized so that the filter works with an alternating input signal. In addition, if an electrolytic capacitor is used, there are problems that are usually very high tolerance percentages, which makes the cut-off frequency different from the desired one.

3.1.4 ADC acquisition in the WSN

The Zolertia module from the hand of its manufacturers has already developed functions to simplify the work of developers, these libraries are already hosted in the Contiki repository. However, these code libraries are designed to detect analog values between long periods of time, normally exceeding 1 second, and not continuously. They add unnecessary validations, they do not allow a detailed configuration. For this reason, the ADC libraries were analysed directly in the SoC and, in this sense, functions were structured that directly writes the internal registers of the hardware at the lowest level, see Figure 3.5.

```
while(1){
    REG(0x400D7008)=0xB5;
    while(!(REG(SOC_ADC_ADCCON1) & SOC_ADC_ADCCON1_EOC));
    printf("%c%c\n",REG(0x400D7010),REG(0x400D700C));
}
```

Fig. 3.5 Algorithm for the acquisition of analog signals in Contiki

The objective is to take full advantage of the sampling rate that the CC2538 allows us to execute. The algorithm proposed to sample in test mode all the channels can be seen in Annex VI

3.2 Raspberry PI as Border Router

Border routers have the function of connecting the WSN to larger networks such as a private LAN or the Internet, they are connected to a constant power source, and it is assumed that it is always connected to the network of interest. Normally, there are products on the market that are responsible for this task, which usually have more features and resources than wireless sensors. An example of a commercial Border Router is the Zolertia Orion Ethernet Router (see Figure. 3.6), which has more features at the processing and memory level than the Re-Motes. Regarding the radio environment, the WSN have a world apart from the classic wireless networks because in some cases they work on another frequency, with other mechanisms, another modulation to the classic local networks such as Wifi or LAN wired environments. The Border Router is a gateway to these two different worlds.

Fig. 3.6 Zolerta Orion Router, Border Router

Our BR is composed of another module Zolertia Re-Mote plus a Raspberry pi, which will be responsible for adding network layers and information processing to be able to have this information it in other networks.
The connection between Raspberry Pi and the Zolertia radio device is through a serial cable that emulates network layers through simple commands, this serial protocol is known as "slip radio", Which is specified later

3.2.1 Raspberry PI

Raspberry Pi (RPI) is a low cost single board computer (SBC) developed in the United Kingdom by the Raspberry Pi Foundation. As well as a computer, it has input and output peripherals and its official operating system is an adapted version of Debian, called Raspbian. Although it allows to use other operating systems, including a version of Windows 10. Among its most outstanding features can be highlighted the following features:

- 1.2 GHz 64-bit Broadcom processor
- 1GB of RAM
- GPU
- USB ports
- 1 HDMI port
- 1 Ethernet port
- 40 GPIO pins

In the proposed design, the raspberry Pi is used to deal with the following network services and embedded applications:

- Border Router
- UDP6 server
- Database Server
- Web server

It can also be used as a bio-signal generator, but for this feature it is recommended to use another additional RPI.

3.2.2 Commissioning Process of RPI

With the help of the ETCHER program we load the JESSI Raspbian distribution operating system based on Debian. It is recommended to use a memory greater than 16 GB of class 10. Once the OS is loaded, it must be updated, so as not to have problems with the installations of the programs

```
$sudo apt-get update
$sudo apt-get upgrade
```

Setup basic configuration as Country Zone, Keyboard etc.

```
$sudo raspi-config
```

To facilitate development, it is important to install services such as FTP, RDP, and SSH. With this, we ensure a total integration with our host in which we have worked. Next are the simple commands to enable these services on the RPI

FTP

```
$sudo apt-get install vsftpd
$sudo nano /etc/vsftpd.conf
```

Change #write_enable=YES

```
$sudo service vsftpd restart
```

RDP

```
$sudo apt-get install xrdp tighvncserver
```

3.2.3 GPIO

In order for the bio-signals to be generated, GPIO of RPI will be used. A GPIO is a general-purpose input/output system, that is, a series of connections that can be used as inputs or outputs for multiple uses. GPIOs represent the interface between Raspberry Pi and the outside world, but, As shown in the Figure 3.7, not all pins have the same function, so we have:

Power pins: It has pins of 5v, 3v3 (limited to 50mA) and ground (GND or ground), which provide power to these voltages for external circuits.

Normal GPIO: configurable connections that can be programmed.

Special GPIOs: within these are some pins for a UART interface, with TXD and RXD connections that are used for serial communications.

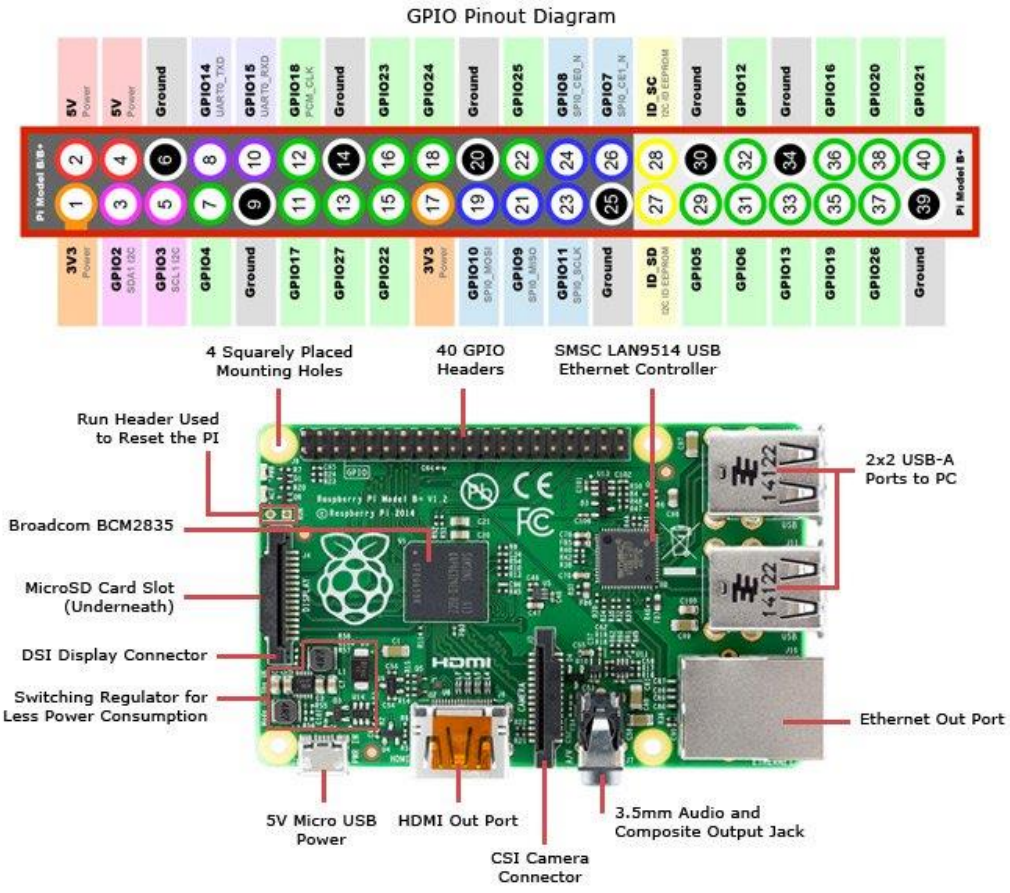

Fig. 3.7 Raspberry PI overview and its GPIO headers.

3.2.4 6LBR

A 6LoWPAN border router is a border router that connects 6LoWPAN devices to a local network or to the Internet directly. Handles traffic to and from the IP and 802.15.4 interfaces (see Figure. 3.8). The 6LBR is a solution developed by CETIC, open source and can play a role as a router or bridge. When it comes to a router, it separates IPv6 and 6LoWPAN networks into different subnets, behaving like a normal router. Join two different networks.

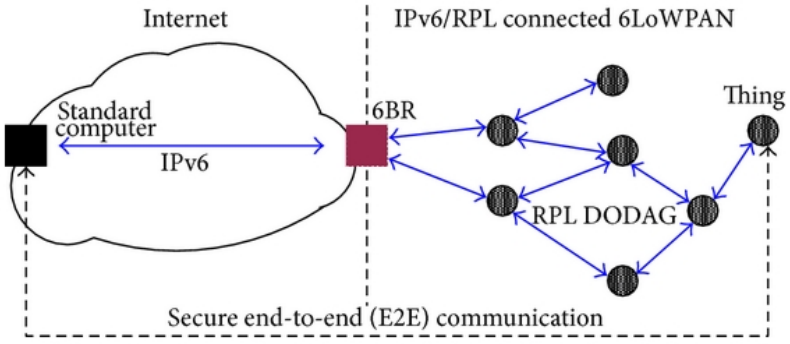

Fig. 3.8 6LBR in the IoT network architecture

The 802.15.4 network is managed by the RPL protocol and the Ethernet side is managed by NDP. These two networks are logically separated. Each network has a different prefix, for example [aaaa ::] for ethernet and [bbbb ::] for WPAN. The nodes are configured to route the traffic destined to WPAN through 6LBR and vice versa.
This virtual router that can be raised in different operating systems, offers several modes of operation in addition to operating as a router, example we have:

- Smart Bridge
- Transparent bridge
- Router
- NDP-Router
- 6LR
- RPL-Root
- RPL-Replay
- Full Transparent Bridge

In this design, it is configured the router mode, for its ease of use and its network-level features. For the installation details, see the Annex III.

3.3 SERVICES

The network services created in the RPI, respond to the need to distribute work. This division of services can be abstracted in the following: The service to receive information from the WSN, service to store the information and then the service to present the information to the end user.

3.3.1 Storage Service

Within the Raspberry pi, a data base service (MongoDB) is running, which allows it to store everything the sensors can send in a non-relational database. This is to guarantee a record of the information acquired in our WSN and in order for any higher-level application to process this information. It could be a front-end application or a desktop application.

3.3.1.1 MongoDB

MongoDB (from the English word "humongous" which means enormous) is a document-oriented NoSQL database system, developed under the concept of open source. MongoDB is part of the new family of NoSQL database systems. Instead of saving data in tables as it is done in relational databases, MongoDB saves data structures in documents similar to JSON with a dynamic schema (MongoDB uses a specification called BSON), making the data integration in certain applications more easy and faster. What makes it suitable for our design, however, there is no official installation for Raspbian, but thanks to its open source spirit, it has been possible to install all the configuration files, as well as to use the executable binaries created by different developers

```
$wget https://andyfelong.com/downloads/core_mongodb.tar.gz
$tar -zxvf core_mongodb.tar.gz
$grep mongodb /etc/passwd
$sudo adduser --ingroup nogroup --shell /etc/false --disabled-password --gecos "" \--no-create-home mongodb
$sudo chown root:root mongo*
$sudo chmod 755 mongo*
$sudo strip mongo*
$sudo cp -p mongo* /usr/bin
$sudo mkdir /var/log/mongodb
$sudo chown mongodb:nogroup /var/log/mongodb
$sudo mkdir /var/lib/mongodb
$sudo chown mongodb:root /var/lib/mongodb
$sudo chmod 775 /var/lib/mongodb
cd /etc
$sudo nano mongodb.conf
```

The service is set up with the next command:

```
$sudo service mongodb start
```

It is possible to verify that everything is working correctly with

```
$sudo service mongodb status
```

After this a message appears in green "Active". Additionally, it is possible to verify it at the TCP service level. In the following image, with the help of "netstat" we can distinguish that the system is already listening on port 27017 (see Fig 3.9). For this port, we will raise the session at the client level with a GUI application called Robo 3T to verify the database (see Fig 3.10).

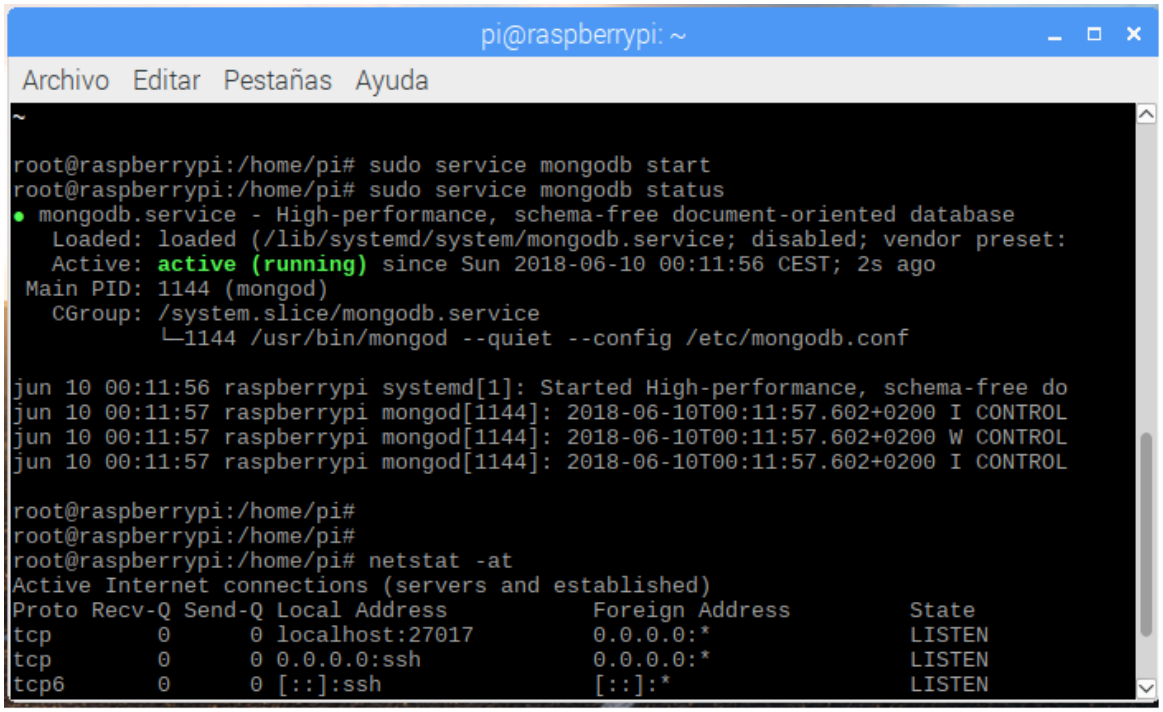

Fig. 3.9 MongoDB service status

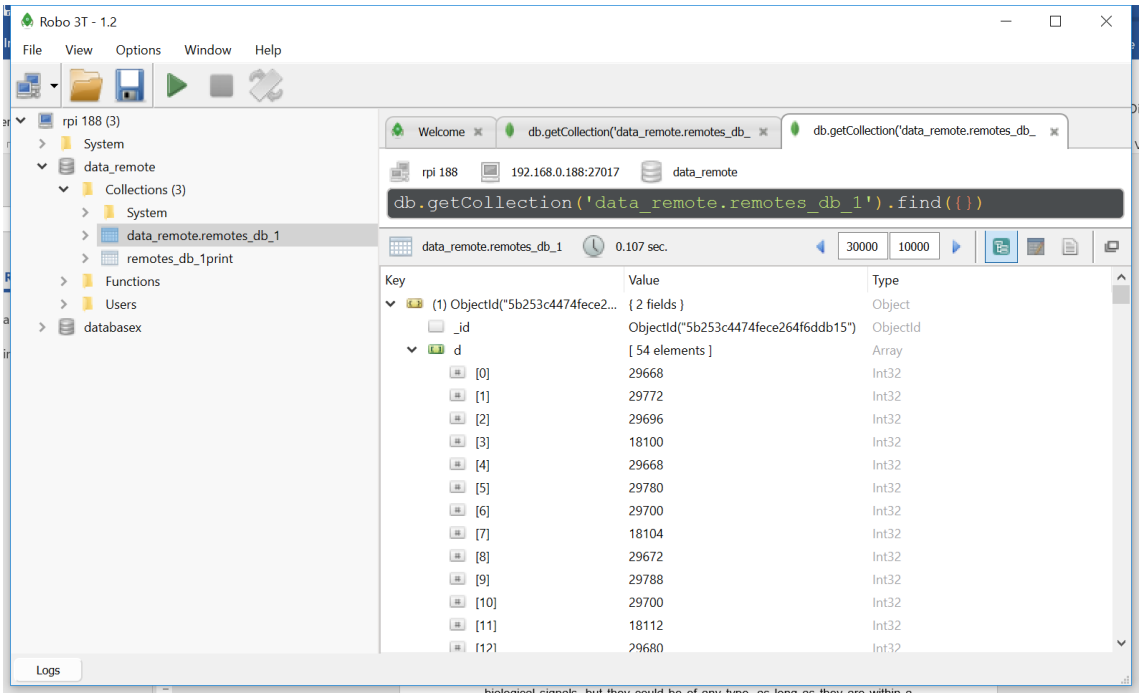

Fig. 3.10 Robo 3t GUI Data base

3.3.2 UDP6 Server (Receiving Service)

In the communication of data as transport protocol it is possible to use UDP or TCP when we talk about classic IPv4 networks or for IPv6 networks. That is, it adds support for IPv6 networks. This communication uses an abstract representation for the local endpoint of a network communication path as a file descriptor in the Unix philosophy: socket. It provides a common interface for the input and output of data flows.

This service stores the information coming from 6LowPAN in the MongoDB database. To perform the whole process, it is first necessary to listen to the information that arrives at the udp6 socket, this socket is established from the client's node. To make sure that we are really listening in the correct UDP port, we can use the following command:

```
$ netstat -au
```

Once the socket service has been lifted, a permanent listening must be established. Then, we must also start a permanent session with the database service in MongoDB. When a data buffer arrives at the port, it is separated and placed in a temporary list, the format is changed to decimals and inserted into the corresponding collection of MongoDB. In the end, the created temporal list must be released so that the same information is not reinserted. This process can be visualized in the following flow chart, Fig 3.11

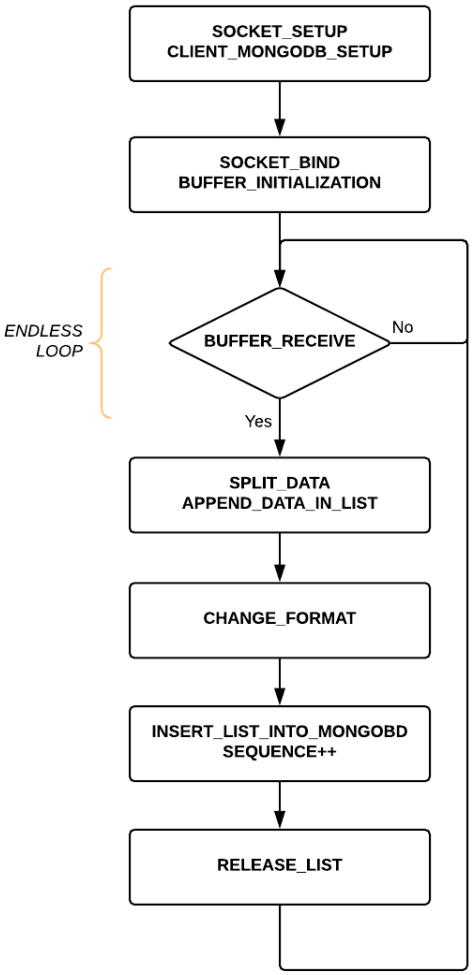

Fig. 3.11 Flowchart of the Reception service in pseudocode

Once installed MongoDB, it is possible to run scripts that could insert data in the database, the detail of the code in python are in the Annex. The algorithm described in Figure 3.11 makes use of the socket and PyMongo libraries for python. The first one is already pre-installed with python, while the second must be installed. Below, it is described briefly.

3.3.2.1 MongoDB with Python

PyMongo is a Python distribution that contains tools to work with MongoDB and is the recommended way to work with MongoDB from Python. The library abstracts the connectivity to the server in a simple way, It has implemented all the functionalities with respect to the manipulation of the data in the collections.

In each test scenario, it has been used to insert into the database all the values that arrive mapped to the udp6 socket. To install this library is only necessary to have python installed and execute the following command lines:

```
$ sudo python -m pip install pymongo
```

```
$ sudo apt-get install build-essential python-dev
```

3.3.3 Front End for IoT

There are several platforms for IoT, with some free features, designed to simplify the life of the solution integrators with respect to the programming of applications for the user, which integrate pre-designed dashboards to show the incoming data, store them, etc. However, these platforms are designed for discrete times and signals where there are no continuous signals with a high bandwidth. In the market there are several, but we can highlight two that are Ubidots and Thethings.io.

There are also more robust platforms, that were designed for the integration of more features and functionalities in the cloud and can easily integrate the Front-End for IoT signals with higher bandwidth. Among them we have AWS Kinesis from Amazon or Azure IoT Hub from Microsoft.

3.3.3.1 Ubidots

It tries to solve the problem generated by the creation of a platform that easily collects the data captured by sensors and converts them into useful information. It facilitates the design for the storage and visualization of the data. For this, we must be connected to the server located on the Internet using access codes that are generated on the platform.

3.3.3.2 TheThings.IO

Like the previous one, it allows creating Dashboards and storage information but with more features, its free version is limited but has more features, ideal for industrial environments.

3.3.3.3 *AWS Kinesis*

It facilitates the collection, processing and analysis of transmission data in real time to obtain data and react quickly to new information. It has capabilities to process transmission data at any scale, as well as the flexibility to choose different tools. Real-time data can be incorporated, such as videos, audios, application logs and IoT telemetry data. It also allows to process and analyse data as it is received and respond instantly.

3.3.3.4 Azure IoT HUB

Quite similar to the previous one, it allows the supervision and administration of billions of devices and the development of Internet of Things (IoT) applications. IoT Hub is a cloud platform as an open and flexible service that supports open source SDK and multiple protocols.

3.3.4 Design of a Front End

A front-end application was developed to link the entire stack of a possible scenario of WSN networks oriented to analog values. Our analog variables are biological signals, but they could be of any type, as long as they are within a limit bandwidth. In the Raspberry pi, an Apache server was added to which JavaScript libraries were located for dynamic graphics environments, PHP functionality was also installed for enable the controller that acquires the data from the database, see Figure 3.12. All of this was possible due to the computational power of the RaspberryPi.

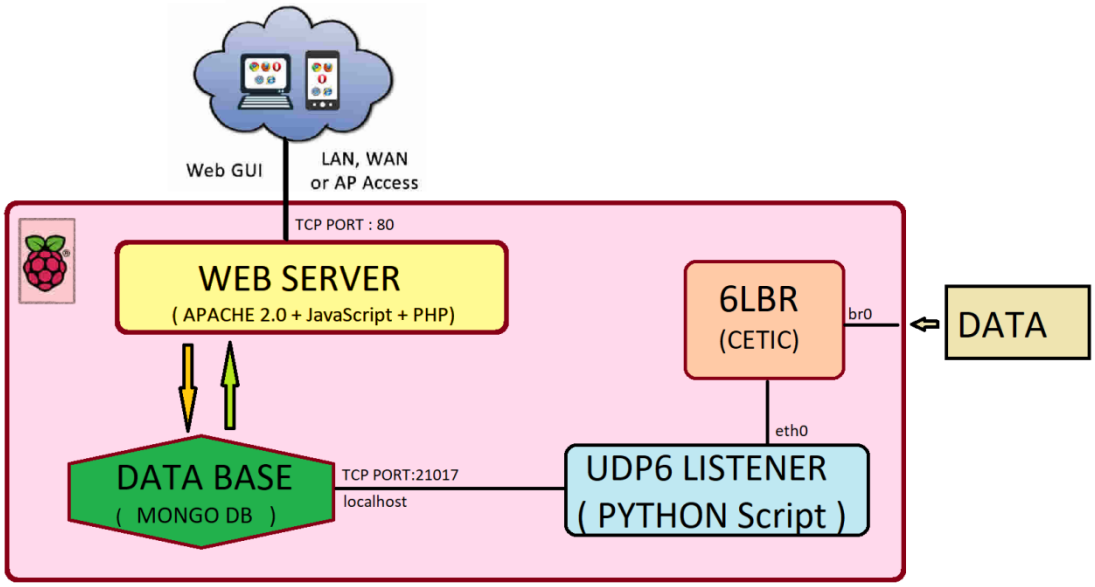

Fig. 3.12 Front End Architecture

3.3.4.1 PHP Controller

The algorithm in PHP establishes a connection with the database server, which in this case is MongoDB. This server could be in another host, but for our solution, it was resolved to use the same one in which the Border Router is installed. When the connection to the database has run smoothly, it points to the last N values of a collection, these values are read as an array format and then passed to a JSON format to be printed as an output of the PHP script.

JSON (JavaScript Object Notation) is a syntax for storing and exchanging data. When exchanging data between a browser and a server, the data can only be text. It is the most used data exchange structure at present. A few years ago, XML was used, but it seems that this trend has changed completely betting most of the developers by JSON, a less restrictive alternative, lighter, more dynamic and probably easier structure of data to read.

.

```php
<?php
    // connect to mongodb
    /*  READ DATA FROM DATABASE (SKIP, AND LIMIT) CONTROLS THE AMOUNT OF DATA */
    $m = new MongoClient();  //same host as server
    $K = 100;
    $NumberOFframes = 1;
    $skips = 0; //start of database
    // select a database
    $db = $m->data_remote;
    //echo "Database mydb selected<br/>";
    $collection = $db->createCollection("data_remote.remotes_db_1");//if not exist, create..
    $cursor = $collection->find()->skip($skips)->limit($NumberOFframes);
    foreach ($cursor as $value) {
        echo json_encode($value['d']);
    }

?>
```

Fig. 3.13 PHP controller for Web server

As shown the Figure 3.13, "$m" is the container for the session that points to the MongoDB service, from here, the variable will contain everything concerning the collections and database hosted there. Next, the database is specified in a new variable "$db" and its respective collection in the variable "$collection". The values returned with the function "find" are printed as output, having been previously encoded with the JSON format, for easy manipulation.

CHAPTER 4. TEST SCENARIOS AND RESULTS

To test the developed system, two scenarios are used. In such a way that the capabilities of implementing this type of IoT architectures are validated. The broad objective of the first scenario is to establish the maximum capabilities that could be achieved in an environment that demands all the features of the devices. The objective of the second scenario is to leave the path traced to sense a biopotential using a specific sampling frequency, as well as being able to be widely configurable as needed.

4.1. Scenario 1

In the first scenario, a configuration was set to test the limits of the device capabilities for this type of solution. That is, to sample at the maximum rate and send the data buffer immediately, without separators or labels that distinguish the channel, in consecutive byte format, without additional information related to the acquired signal. All this to maximize the throughput and see the thresholds of work, also analysed different configurations of the Radio Duty Cycle to see to what extent energy can be saved sampling at the maximum capacity. Also, analyse the response of the designed architecture and how the Front End interacts with the sensed bio-signals in the WSN.

4.1.1. Transmission of data

The data is sent at the byte level consecutively, that is, two bytes for each acquired sample and then two bytes without separators for the next sampled channel. This can be done because the packet loss with the set configuration is practically null (see Figure 4.1). It is expected that on the side of the border router this information will be interpreted in the same way.

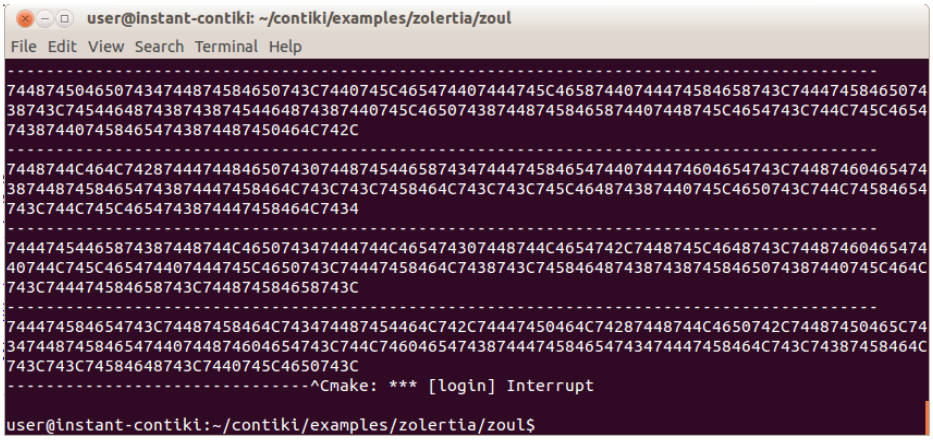

Fig. 4.1. Transmission data in Hexadecimal format

The network connection is established, the sensor is initialized as a client and sees the frontier router as a server. Unlike TCP, UDP does not require three-way link authentication, although the communication process is lighter. Then, the thread of the process is waiting for events, which could be of a different nature, such as, for example, the reception of data. For this scenario, the expected event is a timer controlled by a parallel instance called "etimer", which controls the speed at which it requests to sample all four channels. Each time the timer of the “etimer” overflows, it starts reading channel by channel and saves the information each time it gets the information. After filling the established buffer, the data is sent. In Figure 4.2 shows the flowchart of the whole scenario.

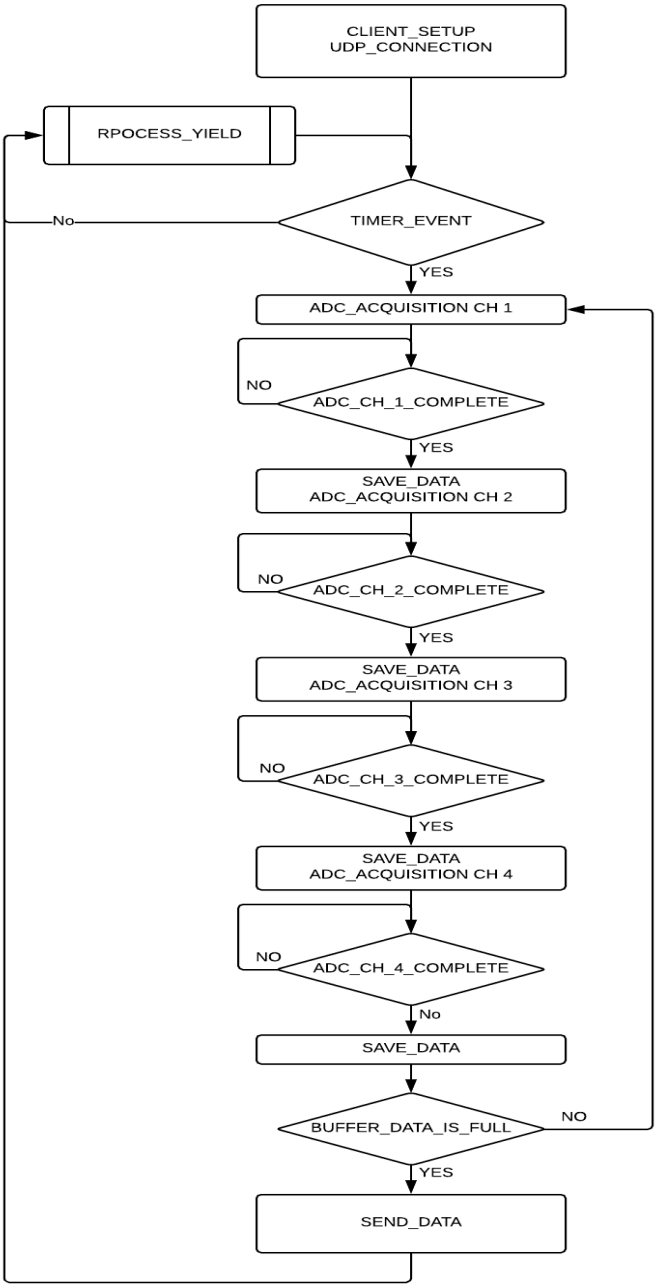

Fig. 4.2 Flowchart of Scenario 1

4.2. Scenario 2

In the second scenario, a JSON structure was proposed for the data buffer, Figure 4.4, also delays are inserted to reduce the sampling time, data sequence, referential time of data acquisition.

As we mention in the previous chapter in the literal 3.3.4.1, JSON is a very extended format for data exchange. In this scenario also permit the easy integration of new values or arrays in the data buffer. An easy decoding of the receiving server side is possible since libraries exist in all languages.

```
{
    "tag": [295, 295, 295, 295, 295, 295, 295, 295, 295, 295, 295, 296],
    "$": [54, 130]
}
```

Fig. 4.3 JSON structure of data

4.2.1. Transmission of data

The data is sent packaged in arrays, the largest array in the image represents the sampled values of the acquired voltage multiplied by 100. On the receiving side, it should be divided for 100, this is done to take full advantage of the buffer size. It's possible to add decimal values, for this case there are only two. The following arrangement represents data relevant to the acquired sample, the first is the sequence that goes from 0 to 999 in a cyclical way, the second value of the array is the time that the sampling in ticks of the 32 KHz oscillator clock (rtimer).

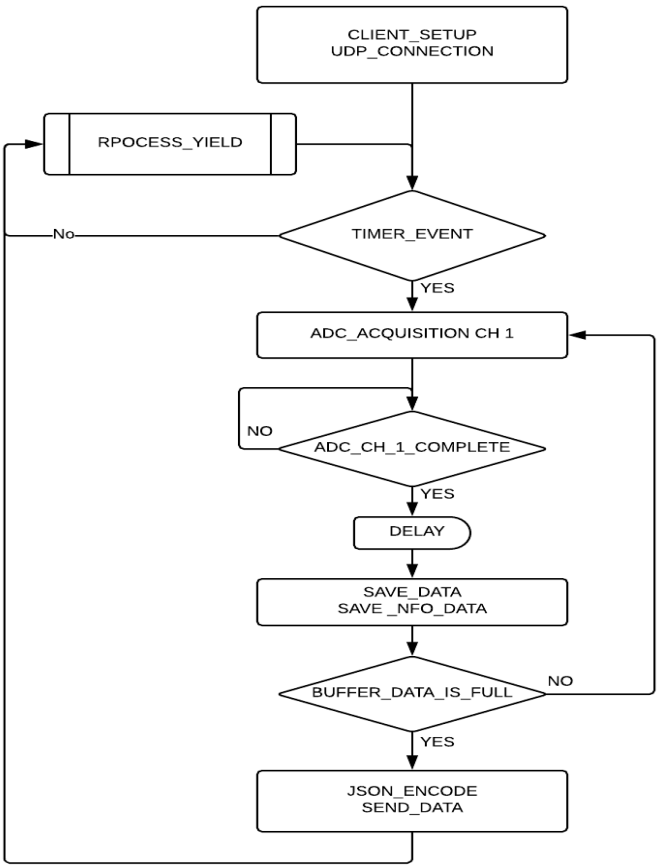

Fig. 4.4 Flowchart of scenario 2

4.3. ADC acquisition

The first results are based on the sampling capacity of the analog signals, then the equipment was put under stress test and took it to the limits of the capabilities. Next, what was achieved with the analog sampling is discussed.

4.3.1. Limitations of Sampling

The maximum sampling speed indicated by the manufacturer of the CC2538 is 20 KHz but with 7 bits of resolution, it also indicates that it has 8 channels, but for Zolertia's Zoul module there are only 4 available, plus one configurable that gives a total of 5 channels, all assigned to the P-AN port. In this case to use the other 3, a small modification should be made to the hardware, which could affect the integrity of the equipment if we do not have the appropriate tools.

The best resolution is 12 bits, which is wider in terms of time of acquisition, at least 5 times more than with 7 bits. So, there is a trade-off between resolution and bandwidth. The conversion time is defined in the base for the following form, where for 12 bits the "decimation rate" is 512. For 10 bits it is 256.

Using the primitive Contiki libraries based on the CC2538 it is possible to sample 2.4 KSPS. But, directly using the registers and putting them in byte format at the time of being transmitted, that is, in hexadecimal format, a bandwidth is obtained for sampling higher than 3.8 KSPS.

4.3.2. Testing the real Bandwidth

A signal generator in the lab was used as input and a graphing tool in Linux print the signal, this plotter is a very useful tool called Gnuplot. The algorithm used is described in the Annex IV. For signals of frequencies lower than 10 Hz, the sampling of a channel is quite acceptable. See Figure 4.5.

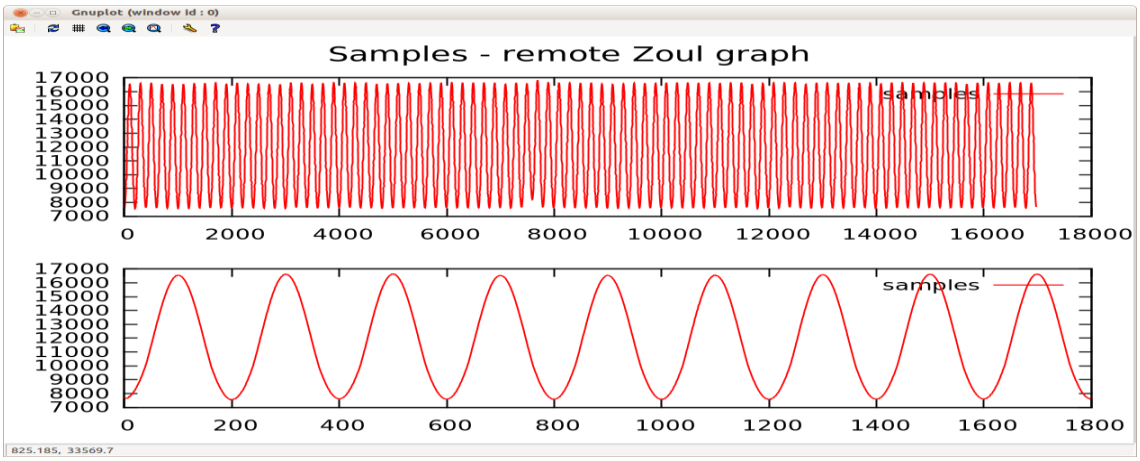

Fig. 4.5 ADC acquisition Zoul Re-Mote B

The problem starts from the 400 Hz where the sinusoidal input signal begins to degrade, see Figure 4.6, although a reconstruction would be possible, this limit of bandwidth has been established for the acquisition of a single channel. In practice, bio-signals do not have very high frequencies, but medical equipment standards in the market require a minimum bandwidth of 40 Hz or higher for ECG as minimal design requirements, specified by IEC 60601-2-25, IEC60601-2-27 and IEC60601-2-47 specifications.

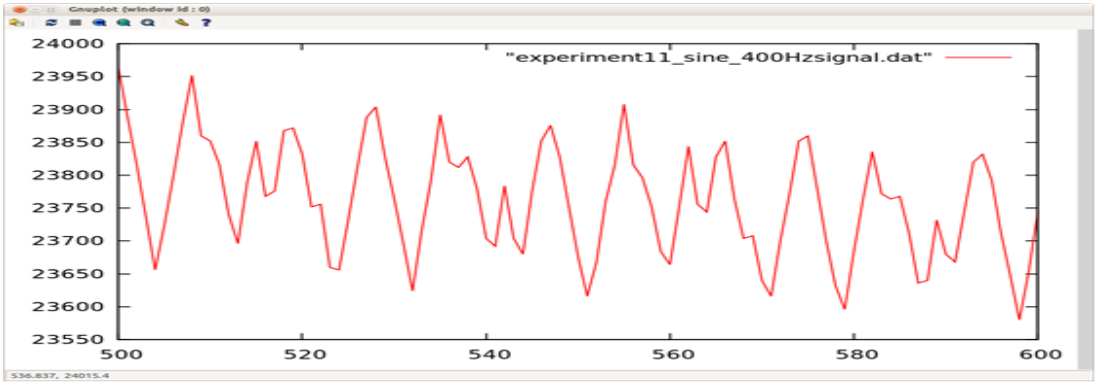

Fig. 4.6 ADC acquisition and transmission of 400 Hz.

4.4. Bio-Signal Generation Waveforms

The following Figure 4.7 shows the artificial curves generated by the bio-signal generator. It is possible to distinguish an ECG waveform. These same signals were used as input in the Zolertia modules. Then, it is possible to vary the frequency of the signal to produce artificial scenarios where the cases of heart malfunction are recreated, to develop intelligent algorithms that distinguish or filter the signals digitally.

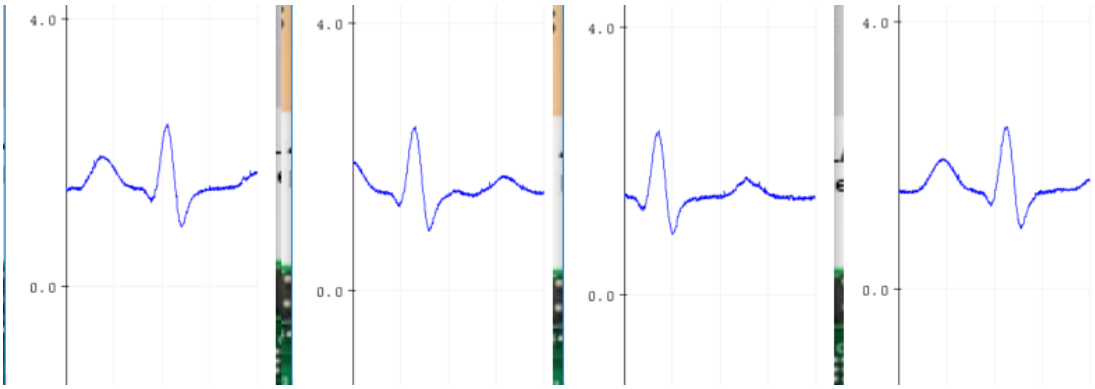

Fig. 4.7 ECG generated with Bio-Signal Generation Waveforms

4.5. Real Acquisition using Sensors

The real sensor, module DuinoPeak, gave signals similar to those simulated , see Figure 4.8, these real ECG signal shown below clearly have a noise added by the 50 Hz frequency, since the CMRR of the device is not very high and the measurement environment is subject to electromagnetic interference.

Fig. 4.8 ECG real acquire with sensor module DuinoPeak

4.6. Verification of the medium

To verify the wireless transmission and validate how reliable could be in different configurations a sequential mechanism of bytes of different size was established, it was verified that the maximum length of the payload is 108 Bytes. The transmitting device sends the receiver a consecutive sequence without stopping in order to detect possible errors in the transmission. The receiver fills a variable Data Buffer that is dimensioned itself given the first frame. Knowing the number of bytes per frame and considering that the bytes will always be consecutive to each other, it is possible to establish an error verification algorithm. The algorithm analyses byte by byte and tries to find two types of errors:

4.6.1. The error of byte

This error could occur if a byte of the frame sent has been modified, we will be able to know it since the sequence of bytes must always be maintained.

4.6.2. The frame error

Complete frames may be lost and regardless of the amount, the algorithm quantifies the number of frames lost.

The validation of the data allows us to know how the transmission responded according to the configuration of the work cycle layer. Figure 4.9 shows the results. By testing different configurations, it was determined that “nullrdc” is the best option to work with this type of signals, , in order to get the best performance.

Fig. 4.9 Results of the validation algorithm in the sequential transmission tests

4.7. Services Listening

Each service had its functionality separately, the integration of all of them made the system work as a full stack architecture

4.7.1. UDP6 socket service

The following Figure 4.10, shows the data obtained with the python algorithm that opens the udp6 socket and fills the data buffer. This buffer is mapped with hexadecimal format. First, it is separated by interest samples and then each value is internally transformed to an integer, 32 bit integer format. In this part a data verification algorithm can be implemented for error filtering, however it was not done since all data arrive in a synchronized manner.

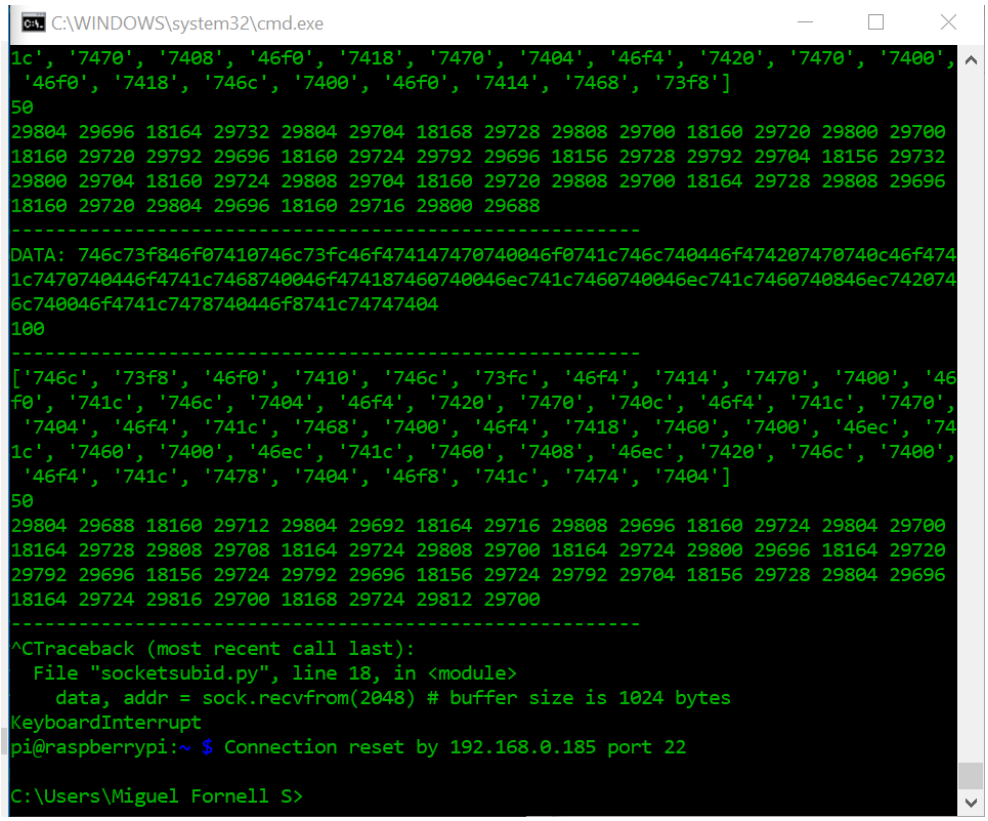

Fig. 4.10 Data buffer received at UDP port

4.8. Final Front End

The user can see the data sent by the nodes on the page. The server in the Raspberry Pi, is running some open source libraries with JavaScript functions, which receive the information of the PHP driver. The more information received from the controller, the more resources of the web client machine will be needed. For the tests it was determined that a matrix of approximately 50 * 53 was transferred, that is, a little more than 2500 values that are divided between the 4 bio-signals. Figure 4.11 shows the acquisition of a noisy signal; in Figure 4.12, the acquired ECG signals are shown.

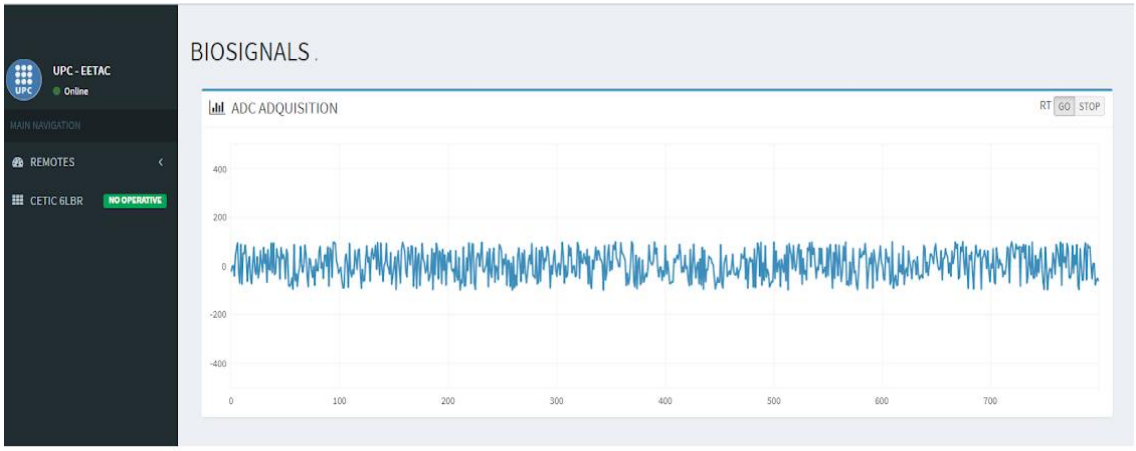

Fig. 4.11 Basic front end of a single analog signal.

4.9. Costs overview

The design costs are shown in Table 4.1, 2 RPIs were used, since initially each service was installed separately, leaving the 6LBR in a single RPI that only works for the moment with Jessie distribution of Raspbian. In the end, all the services could be implemented in the same RPI.

Table 4.1. Cost of developing

Item	Model / Description	Unit Price	Qty	Total
RasperryPi	3 B	40	2	80.0
SD card	Sandisk 16 GB	8.5	1	8.5
SD card	Sandisk 64 GB	20	1	20.0
Power Supply	Generic	5	2	10.0
Wires	Arduino wires kit	5.2	1	5.2
Arduino module	Uno Wifi	30	1	30.0
Electrodes	100 u / Clinical	5.9	2	11.8
Protoboard	Small	4.5	1	4.5
Remote	Zolertia rev B	91.9	3	275.7
Electronic components	Varius Capacitances and Resistances	10	1	10.0
ECG sensor	ADB232 DuinoPeak	18.8	1	18.8
Temperature sensor	LM 35	1	1	0.5

TOTAL	**€ 475.0**

Also in table 4.2, a summary of what would be needed to implement a solution of this type is shown, costs as is evident, are reduced. A commercial border

router is around 200 euros at the moment and does not have all the services and scalability that an RPI offers.

Table 4.2 Tentative Cost for Hardware for implementation

Item	Model / Description	Unit Price	Qty	Total
RasperryPi	3 B	40	1	40.0
SD card	Sandisk 64 GB	20	1	20.0
Power Supply	Generic	5	1	5.0
Remote	Zolertia rev B	91.9	2	183.8

TOTAL	**€ 248.8**

CHAPTER 6. CONCLUSIONS

A WBAN has several challenges to overcome, the biggest challenge now is the energy issue. However, great advances have been made in data compression and efficient data management. We have seen that the key is to have control of all processes, for this reason it is concluded that through the use of architectures and open codes, it is possible to implement IoT architectures of different types. But in the case of bio-signals, Developers must consider the most efficient radio service cycle mechanisms, their maximum working frequency for analog sampling. And as far as possible, the powers consumption in active mode and sleep mode, more optimized.

WSNs oriented to continuous sensing, such as those that are bio-signals, must go through more optimized mechanisms in order to get the most out of energy resources. To improve the performance in this aspect, the bi-directionality of the communication can be applied, the scenarios are designed so that this integration is as simple as possible since it handles interruption events

The transmission of bio signals in the ISM spectrum, need to have defined standardized security mechanisms, the protocols 802.15.4j and 802.6s are options for the communication layer, however, more work should be done on interoperability and security.

The use of a low cost SBC such as the RPI in this type of solutions, allows a greater integration of services and ease of deployment compared to using commercial devices intended to be Borders Routers. It takes advantage of the power that computer resources offer and saves money. More tests would be required to establish the extent to which a solution can be relied on given that it does not have an industrial focus. The use of a non-relational database such as MongoDB, greatly helps the rapid response of the service in general, even in the most demanding scenario. What makes them ideal for abstracting the storage service in an IoT architecture, where the volume of data is growing exponentially in a vertical direction.

An IoT architecture must take into account, first and foremost, security in its border routers, having vulnerabilities of open ports on the Internet makes them easy targets for hackers. The nodes of the WSN are also potentially attackable by having an IP and active ports. Each network service that is created, must have implemented at least one level of security in the case of implementing a real scenario. For instance, In the case of MongoDB, authentication is possible, and in the case of the web server, installing SSL certificates would be an option.

Given the limited capabilities of the equipment, an application that includes an exhaustive measurement of all body signals is not feasible, more than anything is limited by the transmission rate. Although, since this equipment was not designed for this purpose, it responds quite well and can be used for specific situations where the bio-signals are collected during a certain time and then, another process, is responsible for sending them to the cloud or to local processing services.

ACRONYMS

6LoWPAN	IPV6 Over Low Power Wireless Personal Area Networks
6LBR	Six LoWPAN Border Router
ADC	Analog to Digital Converter
API	Application Programming Interface
BPSK	Binary Phase Shift Keying
CPU	Central Processing Unit
CSMA	Carrier Sense Multiple Access
DAG	Directed Acyclic Graphs
DAO	Destination Advertisement Object
DHCP	Dynamic Host Configuration Protocol
ECG	Electrocardiogram
EU	European Union
GPIO	General Purpose Input/Output
HTTP	Hypertext Transfer Protocol
ICMP	Internet Message Control Protocol
KSPS	Kilo Sample Per Second
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
LAN	Local Area Network
LPM	Low Power Mode
MAC	Media Access Control
MBAN	Medical Body Area Networks
MCU	Microcontroller Unit
MTU	Maximum Transmission Unit
MSPS	Mega Sample per Second
NAT	Network Address Translation
NDP	Neighbor Discovery Protocol
QPSK	Quadrature Phase-Shift Keying
P2P	Point-to-Point
PAN	Personal Area Network
QoS	Quality of Service
RA	Router Advertisement
RAM	Random Access Memory
RDC	Radio Duty Cycle
RF	Radio Frequency
RPL	Routing Protocol for Low-Power and Lossy Networks
RPI	Raspberry Pi
SLIP	Serial IP
SoC	System on Chip
SBC	Single Board Computer
SPI	Serial Peripheral Interface
TCP	Transmission Control Protocol
UART	Universal Asynchronous Receiver/Transmitter
UDP	User Datagram Protocol
USB	Universal Serial Bus

WAN	Wide Area Network
WBAN	Wireless Boody Area Network
WPAN	Wireless Personal Area Network
WSN	Wireless Sensor Network

REFERENCES

[1] Instrumentation and Measurement in Medical, Biomedical, and Healthcare Systems, Shervin Shirmohammadi, Kurt Barbé, Domenico Grimaldi, Sergio Rapuano, and Sabrina Grassini, Oct 2016

[2] A. Dionisi, D. Marioli, E. Sardini, and M. Serpelloni, “Autonomous wearable system for vital signs measurement with energy-harvesting module,” IEEE Trans. Instrum. Meas. vol. 65, no. 6, pp. 1423–1434, June 2016.

[3] Goodwin MS1, Velicer WF, Intille SS. “Telemetric monitoring in the behavior sciences” Behav Res Methods. 40(1):328-41, Feb 2008.

[4] Busch C1, Baumbach C, Willemsen D, Nee O, Gorath T, Hein A, Scheffold T. “Supervised training with wireless monitoring of ECG, blood pressure and oxygen-saturation in cardiac patients” , J Telemed Telecare. 15(3):112, 2009

[2] Zach Shelby, and C. Bormann, “6LoWPAN The Wireless Embedded Internet" (6), 153 (2009).

[7] D. Evans, “TG 4j Amendment Draft,” IEEE P802.15-11-0836-06-0004j, Mar. 2012.

[8] FCC, “Medical Body Area Networks First Report and Order,” May 24, 2012.

[9] Steven Jones, "Medical Body Area Networks - TCB Workshop," Oct 12, 2014.

[10] Kyung Sup Kwak, Sana Ullah, and Niamat Ullah, "An Overview of IEEE 802.15.6 Standard"

[11] E. Baccelli, C. Gündoğan, O. Hahm, P. Kietzmann, Martine S. Lenders, H. Petersen, K. Schleiser, T. Schmidt, and M. Wahlisch, "RIOT: An Open Source Operating System for Low-end Embedded Devices in the IoT", March 2018

[12] Adam Dunkels, Swedish Institute of Computer Science for Sensor Networks, Poster Abstract: Rime — A Lightweight Layered Communication Stack , http://dunkels.com/adam/dunkels07rime.pdf

[13] Isam Ishaq , David Carels, Girum K. Teklemariam, Jeroen Hoebeke, Floris Van den Abeele, Eli De Poorter, Ingrid Moerman and Piet Demeester,“IETF Standardization in the Field of the Internet of Things (IoT): A Survey”, Journal of Sensor and Actuator Networks, April 2013 .

ANNEX

Annex I: Sensors Programs

```
// Scenario 1

#include "contiki.h"
#include "contiki-lib.h"
#include "contiki-net.h"
#include "net/ip/resolv.h"

#include <string.h>
#include <stdbool.h>
//....
#include "cpu.h"
#include "sys/etimer.h"
#include "sys/rtimer.h"
#include "dev/leds.h"
#include "dev/uart.h"
#include "dev/button-sensor.h"
#include "cpu/cc2538/dev/adc.h"
#include "dev/adc-zoul.h"
#include "cpu/cc2538/dev/soc-adc.h"
#include "dev/zoul-sensors.h"
#include "dev/watchdog.h"
#include "dev/serial-line.h"
#include "dev/sys-ctrl.h"
#include "net/netstack.h"
#include "net/rime/broadcast.h"
#include "reg.h"

#define DEBUG DEBUG_PRINT
#include "net/ip/uip-debug.h"
#define SERVER_PORT         3000
#define SEND_INTERVAL       CLOCK_SECOND/40
#define MAX_PAYLOAD_LEN     108
char msg_payload[MAX_PAYLOAD_LEN];
int count=0;
unsigned short seq=0,seq2=0;
static struct uip_udp_conn *client_conn;
/*---------------------------------------------------------------------------*/
PROCESS(udp_client_process, "UDP client process");
AUTOSTART_PROCESSES(&resolv_process,&udp_client_process);
/*---------------------------------------------------------------------------*/
static void
tcpip_handler(void)
{
  char *str;

  if(uip_newdata()) {
    str = uip_appdata;
    str[uip_datalen()] = '\0';
    printf("Response from the server: '%s'\n", str);
  }
}
/*---------------------------------------------------------------------------*/
static char buf[MAX_PAYLOAD_LEN];
static void
timeout_handler(void)
{   int i=0;
  //static int seq_id;


  //PRINT6ADDR(&client_conn->ripaddr);
    while(i<106){
    REG(0x400D7008)=0xB5; //0x85 7bit, 0xA5 10 bit  //ADC1 -AN5
    while(!(REG(SOC_ADC_ADCCON1) & SOC_ADC_ADCCON1_EOC));
            msg_payload[i++] = REG(0x400D7010);
            msg_payload[i++] = REG(0x400D700C);

    REG(0x400D7008)=0xB4; //            //ADC2 - AN4
    while(!(REG(SOC_ADC_ADCCON1) & SOC_ADC_ADCCON1_EOC));
            msg_payload[i++] = REG(0x400D7010);
```

```
            msg_payload[i++] = REG(0x400D700C);

    REG(0x400D7008)=0xB2; //             //ADC3 - AN2
    while(!(REG(SOC ADC ADCCON1) & SOC ADC ADCCON1 EOC));
            msg_payload[i++] = REG(0x400D7010);
            msg_payload[i++] = REG(0x400D700C);

    REG(0x400D7008)=0xB2; //             //ADC3 - AN2
    while(!(REG(SOC ADC ADCCON1) & SOC ADC ADCCON1 EOC));
            msg_payload[i++] = REG(0x400D7010);
            msg_payload[i++] = REG(0x400D700C);

    }
//sequence validation
    seq++;
        if(seq==256)
        {seq2++;
         seq=0;
        }
    msg_payload[106] = seq2;
    msg_payload[107] = seq;

#if SEND_TOO_LARGE_PACKET_TO_TEST_FRAGMENTATION
  uip_udp_packet_send(client_conn, "1234567890abc", UIP_APPDATA_SIZE);
#else /* SEND_TOO_LARGE_PACKET_TO_TEST_FRAGMENTATION */
  uip_udp_packet_send(client_conn, msg_payload,MAX_PAYLOAD_LEN);
#endif /* SEND_TOO_LARGE_PACKET_TO_TEST_FRAGMENTATION */
}
/*---------------------------------------------------------------------------*/
static void
print_local_addresses(void)
{
  int i;
  uint8_t state;

  PRINTF("Client IPv6 addresses: ");
  for(i = 0; i < UIP_DS6_ADDR_NB; i++) {
    state = uip_ds6_if.addr_list[i].state;
    if(uip_ds6_if.addr_list[i].isused &&
       (state == ADDR_TENTATIVE || state == ADDR_PREFERRED)) {
      PRINT6ADDR(&uip_ds6_if.addr_list[i].ipaddr);
      PRINTF("\n");
    }
  }
}
/*---------------------------------------------------------------------------*/
#if UIP_CONF_ROUTER
static void
set_global_address(void)
{
  uip_ipaddr_t ipaddr;

  uip_ip6addr(&ipaddr, UIP_DS6_DEFAULT_PREFIX, 0, 0, 0, 0, 0, 0, 0);
  uip_ds6_set_addr_iid(&ipaddr, &uip_lladdr);
  uip_ds6_addr_add(&ipaddr, 0, ADDR_AUTOCONF);
}
#endif /* UIP_CONF_ROUTER */
/*---------------------------------------------------------------------------*/
static resolv_status_t
set_connection_address(uip_ipaddr_t *ipaddr)
{
#ifndef UDP_CONNECTION_ADDR
/*#if RESOLV_CONF_SUPPORTS_MDNS
#define UDP_CONNECTION_ADDR       bbbb::c0d2:2396:5312:e8f4//contiki-udp-server.local*/
#if UIP CONF ROUTER //elif
#define UDP_CONNECTION_ADDR
bbbb::c0d2:2396:5312:e8fd//fd00:0:0:0:0212:7404:0004:0404    fe80::212:4b00:60d:b2da
(zoul)
#else
#define UDP_CONNECTION_ADDR
fe80:0:0:0:8fdb:446d:7f32:5ea5//fe80:0:0:0:6466:6666:6666:6666
#endif
#endif /* !UDP_CONNECTION_ADDR */

#define _QUOTEME(x) #x
#define QUOTEME(x) _QUOTEME(x)
```

```
  resolv_status_t status = RESOLV_STATUS_ERROR;

  if(uiplib_ipaddrconv(QUOTEME(UDP_CONNECTION_ADDR), ipaddr) == 0) {
    uip_ipaddr_t *resolved_addr = NULL;
    status = resolv_lookup(QUOTEME(UDP_CONNECTION_ADDR),&resolved_addr);
    if(status == RESOLV_STATUS_UNCACHED || status == RESOLV_STATUS_EXPIRED) {
      PRINTF("Attempting to look up %s\n",QUOTEME(UDP_CONNECTION_ADDR));
      resolv_query(QUOTEME(UDP_CONNECTION_ADDR));
      status = RESOLV_STATUS_RESOLVING;
    } else if(status == RESOLV_STATUS_CACHED && resolved_addr != NULL) {
      PRINTF("Lookup of \"%s\" succeded!\n",QUOTEME(UDP_CONNECTION_ADDR));
    } else if(status == RESOLV_STATUS_RESOLVING) {
      PRINTF("Still looking up \"%s\"...\n",QUOTEME(UDP_CONNECTION_ADDR));
    } else {
      PRINTF("Lookup of \"%s\" failed. status =
%d\n",QUOTEME(UDP_CONNECTION_ADDR),status);
    }
    if(resolved_addr)
      uip_ipaddr_copy(ipaddr, resolved_addr);
  } else {
    status = RESOLV_STATUS_CACHED;
  }

  return status;
}
/*---------------------------------------------------------------------------*/
PROCESS_THREAD(udp_client_process, ev, data)
{
  static struct etimer et;
  uip_ipaddr_t ipaddr;

  PROCESS_BEGIN();
  PRINTF("UDP client process started\n");

#if UIP_CONF_ROUTER
  set_global_address();
#endif

  print_local_addresses();

  static resolv_status_t status = RESOLV_STATUS_UNCACHED;
  while(status != RESOLV_STATUS_CACHED) {
    status = set_connection_address(&ipaddr);

    if(status == RESOLV_STATUS_RESOLVING) {
      PROCESS_WAIT_EVENT_UNTIL(ev == resolv_event_found);
    } else if(status != RESOLV_STATUS_CACHED) {
      PRINTF("Can't get connection address.\n");
      PROCESS_YIELD();
    }
  }

  /* new connection with remote host */
  client_conn = udp_new(&ipaddr, UIP_HTONS(SERVER_PORT), NULL);
  udp_bind(client_conn, UIP_HTONS(3001));

  PRINTF("Created a connection with the server ");
  PRINT6ADDR(&client_conn->ripaddr);
  PRINTF(" local/remote port %u/%u\n",
    UIP_HTONS(client_conn->lport), UIP_HTONS(client_conn->rport));

  etimer_set(&et,2 ); //SEND_INTERVAL
  while(1) {
    PROCESS_YIELD();
    if(etimer_expired(&et)) {
      timeout_handler();
      etimer_restart(&et);
    } else if(ev == tcpip_event) {
      tcpip_handler();
    }
  }

  PROCESS_END();
}
/*---------------------------------------------------------------------------*/
```

```
// Scenario 2
#include "contiki.h"
#include "contiki-lib.h"
#include "contiki-net.h"
#include "net/ip/resolv.h"
#include "dev/i2c.h"
#include <stdio.h>
#include <stdlib.h>
#include "dev/gpio.h"
#include "dev/leds.h"
#include <string.h>
#include <stdbool.h>
#include "cpu.h"
#include "sys/etimer.h"
#include "sys/rtimer.h"
#include "sys/timer.h"
#include "dev/leds.h"
#include "dev/uart.h"
#include "dev/button-sensor.h"
#include "cpu/cc2538/dev/adc.h"
#include "dev/adc-zoul.h"
#include "cpu/cc2538/dev/soc-adc.h"
#include "dev/zoul-sensors.h"
#include "dev/watchdog.h"
#include "dev/serial-line.h"
#include "dev/sys-ctrl.h"
#include "net/netstack.h"
#include "net/rime/broadcast.h"
#include "reg.h"

#define SERVER_PORT         3000
#define MAX_PAYLOAD_LEN     108
#define NUMBER_OF_SAMPLES   15          // 17 max, 1 min
#define N_DECIMALS      2
#define ENABLE_SAMPLE_INFO  1
char msg_payload[MAX_PAYLOAD_LEN];
int count=0;
static struct uip_udp_conn *client_conn;



char tag_1[] = "{\42tag\42\72";
char adc_array[100]; // constant
    int  len_tag = 0;
    int sequence=0;
    short *p;
    short i,j;
unsigned short seq=0,seq2=0;
unsigned long old_time,old_time_buffer;
int delay=5;


/*---------------------------------------------------------------------------*/
PROCESS(udp_client_process, "UDP client process");
AUTOSTART_PROCESSES(&resolv_process,&udp_client_process);
/*---------------------------------------------------------------------------*/
static void
tcpip_handler(void)
{
  char *str;

  if(uip_newdata()) {
    str = uip_appdata;
    str[uip_datalen()] = '\0';
    printf("Response from the server: '%s'\n", str);
  }
}
/*---------------------------------------------------------------------------*/
int * get_adc_( ) {

   static short  r[(N DECIMALS + 2)*NUMBER OF SAMPLES+2];
   unsigned short i;
   int longi,dummy;
   char buffer_[7];

    //old_time = RTIMER_NOW();
   for ( i = 0; i < ((N_DECIMALS + 2)*NUMBER_OF_SAMPLES+1); ++i) {
```

```
    if(i==0)
    {   r[i++]  = '[';
    }

    longi = RTIMER_NOW();while((RTIMER_NOW()-longi)<delay);
    REG(0x400D7008)=0xB5; //0x85 7bit, 0xA5 10 bit  //ADC1 -AN5 //start adc
    while(!(REG(SOC_ADC_ADCCON1) & SOC_ADC_ADCCON1_EOC)); //end adc 138 u secs

    dummy = REG(0x400D7010)*256 + REG(0x400D700C);

    sprintf(buffer_, "%d", dummy);
    r[i++]  = buffer_[0];
    r[i++]  = buffer_[1];
    r[i++]  = buffer_[2];
    //r[i++]  = buffer [3];
    //r[i++]  = buffer_[4];


    r[i]    = ',';
   }
     r[i-1]=']';
     r[i]='\0';  //end of array

   return r;
}


static void
timeout_handler(void)
{   char buffer[12]; //12?
    len_tag = strlen(tag_1);
    old_time_buffer=RTIMER_NOW();
    p = get_adc_();

    //sequence validation
    seq++;
    if(seq==999)
    seq=0;
    //end sequence validation

    i=0;
    do{
        adc_array[i] = *(p+i);
        i++;
    }while(*(p+i)!='\0');


    strcpy(msg_payload,tag_1);
    strcat(msg_payload,adc_array);

#if ENABLE_SAMPLE_INFO
    strcat(msg_payload,",\42$\42\72[");  //,"$":[
    sprintf(buffer, "%d", seq);
    strcat(msg_payload,buffer);
    strcat(msg_payload,",");
    sprintf(buffer, "%ld", RTIMER_NOW()-old_time_buffer);
    strcat(msg_payload,buffer);
    strcat(msg_payload,"]");
    strcat(msg_payload,"}");
#else
    strcat(msg_payload,"}");
#endif


    printf("%s %d \n",msg payload,strlen(msg payload)); //debug
    uip_udp_packet_send(client_conn, msg_payload,strlen(msg_payload));
// send payload


}
/*---------------------------------------------------------------------------*/
static void
print_local_addresses(void)
{
  int i;
  uint8_t state;
```

```
  PRINTF("Client IPv6 addresses: ");
  for(i = 0; i < UIP_DS6_ADDR_NB; i++) {
    state = uip_ds6_if.addr_list[i].state;
    if(uip ds6 if.addr list[i].isused &&
       (state == ADDR_TENTATIVE || state == ADDR_PREFERRED)) {
      PRINT6ADDR(&uip_ds6_if.addr_list[i].ipaddr);
      PRINTF("\n");
    }
  }
}
/*---------------------------------------------------------------------------*/
#if UIP_CONF_ROUTER
static void
set_global_address(void)
{
  uip_ipaddr_t ipaddr;

  uip_ip6addr(&ipaddr, UIP_DS6_DEFAULT_PREFIX, 0, 0, 0, 0, 0, 0, 0);
  uip_ds6_set_addr_iid(&ipaddr, &uip_lladdr);
  uip_ds6_addr_add(&ipaddr, 0, ADDR_AUTOCONF);
}
#endif /* UIP_CONF_ROUTER */
/*---------------------------------------------------------------------------*/
static resolv_status_t
set_connection_address(uip_ipaddr_t *ipaddr)
{
#ifndef UDP_CONNECTION_ADDR
/*#if RESOLV_CONF_SUPPORTS_MDNS
#define UDP_CONNECTION_ADDR       bbbb::c0d2:2396:5312:e8f4//contiki-udp-server.local*/
#if UIP_CONF_ROUTER //elif
#define UDP_CONNECTION_ADDR
bbbb::c0d2:2396:5312:e8fd//fd00:0:0:0:0212:7404:0004:0404    fe80::212:4b00:60d:b2da
(zoul)
#else
#define UDP_CONNECTION_ADDR
fe80:0:0:0:8fdb:446d:7f32:5ea5//fe80:0:0:0:6466:6666:6666:6666
#endif
#endif /* !UDP_CONNECTION_ADDR */

#define _QUOTEME(x) #x
#define QUOTEME(x) _QUOTEME(x)

  resolv_status_t status = RESOLV_STATUS_ERROR;

  if(uiplib_ipaddrconv(QUOTEME(UDP_CONNECTION_ADDR), ipaddr) == 0) {
    uip_ipaddr_t *resolved_addr = NULL;
    status = resolv_lookup(QUOTEME(UDP_CONNECTION_ADDR),&resolved_addr);
    if(status == RESOLV_STATUS_UNCACHED || status == RESOLV_STATUS_EXPIRED) {
      PRINTF("Attempting to look up %s\n",QUOTEME(UDP_CONNECTION_ADDR));
      resolv_query(QUOTEME(UDP_CONNECTION_ADDR));
      status = RESOLV_STATUS_RESOLVING;
    } else if(status == RESOLV_STATUS_CACHED && resolved_addr != NULL) {
      PRINTF("Lookup of \"%s\" succeded!\n",QUOTEME(UDP_CONNECTION_ADDR));
    } else if(status == RESOLV_STATUS_RESOLVING) {
      PRINTF("Still looking up \"%s\"...\n",QUOTEME(UDP_CONNECTION_ADDR));
    } else {
      PRINTF("Lookup of \"%s\" failed. status =
%d\n",QUOTEME(UDP_CONNECTION_ADDR),status);
    }
    if(resolved_addr)
      uip_ipaddr_copy(ipaddr, resolved_addr);
  } else {
    status = RESOLV_STATUS_CACHED;
  }

  return status;
}
/*---------------------------------------------------------------------------*/
PROCESS_THREAD(udp_client_process, ev, data)
{
  static struct etimer et;
  uip_ipaddr_t ipaddr;


  PROCESS_BEGIN();
  PRINTF("UDP client process started\n");
```

```
#if UIP_CONF_ROUTER
  set_global_address();
#endif

  print_local_addresses();


  static resolv_status_t status = RESOLV_STATUS_UNCACHED;
  while(status != RESOLV_STATUS_CACHED) {
    status = set_connection_address(&ipaddr);

    if(status == RESOLV_STATUS_RESOLVING) {
      PROCESS_WAIT_EVENT_UNTIL(ev == resolv_event_found);
    } else if(status != RESOLV_STATUS_CACHED) {
      PRINTF("Can't get connection address.\n");
      PROCESS_YIELD();
    }
  }

  /* new connection with remote host */
  client_conn = udp_new(&ipaddr, UIP_HTONS(SERVER_PORT), NULL);
  udp_bind(client_conn, UIP_HTONS(3001));

  PRINTF("Created a connection with the server ");
  PRINT6ADDR(&client_conn->ripaddr);
  PRINTF(" local/remote port %u/%u\n",
    UIP_HTONS(client_conn->lport), UIP_HTONS(client_conn->rport));

  etimer_set(&et,2);
  while(1) {
    PROCESS_YIELD();
    if(etimer_expired(&et)) {
    etimer_restart(&et);
        old_time = RTIMER_NOW();

      timeout_handler();

    printf("Time is %ld mili seconds\n", (RTIMER_NOW()-old_time)*1000/32768);


    } else if(ev == tcpip_event) {
      tcpip_handler();
    }
  }

  PROCESS_END();
}
```

Annex II. Contiki Installation

The following steps are required to get Instant Contiki.

Download Instant Contiki. When downloaded, unzip the file, place the unzipped directory on the desktop.

Download and install VMWare Player. It is free to download, but requires a registration. It might require a reboot of your computer.

Start Instant Contiki by running InstantContiki3.0.vmx. Wait for the virtualUbuntu Linux boot up.
Log into Instant Contiki. The password is user.
Compiling an application on Contiki
To compile an application in Contiki there are 3 files: the Contiki application file, explained in the previous section, which contains the application to be used, the optional configuration file of our application and the Makefile. The Makefile structure used by Contiki is quite simple:

```
CONTIKI = ../../../..
all: app-name
include $(CONTIKI)/Makefile.include
```

The first line indicates the location of the Contiki source root, the second linespecifies the applications to be compiled, and the third line includes the entireContiki system, which contains the Contiki core definitions and points to the
specific Makefile to our target platform.
A project in Contiki may have an optional configuration file called projectconf.h. This file is not active by default, to activate it, the following line must be added to the Makefile file of the project:

```
CFLAGS += -DPROJECT_CONF_H=\"../project-conf.h\"
```

The project-conf.h file can activate a number of Contiki configuration options or modify default Contiki parameters.

To compile an application, target hardware platform must be specified using the TARGET = syntax platform, and if it is not specified by default the native platform will be compiled.

```
make TARGET= zoul
```

To compile and load the application on Zoul node:

```
make TARGET=zoul BOARD=remote-revb  foo.upload
```

Annex III. 6LBR Instalation

6LBR requires the installation of libncurses for building Contiki and bridge-utils for the tap-bridging mode.
To install them the following command was executed.

```
apt-get install libncurses5-dev bridge-utils
```

To download source code of the latest version, following commands are needed:

```
git clone https://github.com/cetic/6lbr
cd 6lbr
git submodule update --init --recursive
cd examples/6lbr
```

The following commands compile and install 6LBR from source code:

```
make all
make plugins
make tools
make install
make plugins-install
update-rc.d 6lbr defaults
```

Once 6LBR is installed, next step is 6LBR configuration. The file /etc/6lbr/6lbr.conf contains all configuration parameters. As previously mentioned, 6LBR will run in router mode with bridge mode as interface configuration. The 6LBR configuration is the following:

```
MODE=ROUTER
RAW_ETH=0
BRIDGE=1
CREATE_BRIDGE=0

DEV_BRIDGE=br0
DEV_TAP=tap0
DEV_ETH=eth0
RAW_ETH_FCS=0
DEV_RADIO=/dev/ttyUSB0
BAUDRATE=115200
```

NODE_CONFIG=/etc/6lbr/node_config.conf
It is also necessary to create the Bridge interface. As DHCP is enabled, the content of the interface configuration file (/etc/network/interfaces) is the following:
iface eth0 inet static
address 0.0.0.0
auto br0
iface br0 inet dhcp
bridge_ports eth0
bridge_stp off
up echo 0 > /sys/devices/virtual/net/br0/bridge/multicast_snooping
post-up ip link set br0 address `ip link show eth0 | grep ether | awk '{print $2}'`
Whenever the configuration is changed it is necessary to restart the service.

```
service 6lbr restart
```

Annex IV. GNUPLOT algorithm

```
GNUPLOT CODE

#plot.p

unset label
set multiplot layout 2, 1 title "Samples - remote Zoul graph" font ",14"
plot samples with lines
print "Limit in plot ",GPVAL_X_MAX
set xrange [GPVAL_X_MIN:GPVAL_X_MAX/10]
plot samples with lines
unset multiplot
pause -1


BASH SCRIPT FOR CALL GNUPLOT SCRIPT

#!/bin/bash
#shell script for plot ADC values using gnuplot, "count_data.dat" must be
created previously
sed -i '1d' count_data.dat
#line deleted!
VAR=$(wc -l count_data.dat)
VAR=${VAR/count_data.dat/}
echo -e $VAR>plot.SAMPLES
echo "Number of samples $VAR "  #print number of samples in ADC
gnuplot -e "samples='count_data.dat'"  plot.p  #Call an external plot.p script
#end
```

Annex V. Final PHP Controller

```
<?php
  /*  READ DATA FROM DATABASE (SKIP, AND LIMIT) CONTROLS THE AMOUNT OF DATA */
   $m = new MongoClient();     // connect to mongodb
   $NumberOFframes = 50;
   $skips = 10;
   //echo "Connection to database successfully<br/>";
   // select a database
   $db = $m->data_remote;
   $collection = $db->createCollection("data_remote.remotes_db_1");//if not
exist, create..

        $valueN = $collection->count();
        $cursor = $collection->find()->skip($valueN - $NumberOFframes);
        $valuex = [];
        foreach ($cursor as $value) {
            array_push($valuex,$value['d']);

                }
        echo json_encode($valuex);

        //      echo json_encode($value);

    //echo count($cursor['d']);
   //echo "<br/>";
?>
```

Annex VI. ADC acquisition Test

```
// FULL ADC  acquisiton
#include "contiki.h"
#include "cpu.h"
#include "sys/etimer.h"
#include "sys/rtimer.h"
#include "dev/leds.h"
#include "dev/uart.h"
#include "dev/button-sensor.h"
#include "cpu/cc2538/dev/adc.h"
#include "dev/adc-zoul.h"
#include "cpu/cc2538/dev/soc-adc.h"
#include "dev/zoul-sensors.h"
#include "dev/watchdog.h"
#include "dev/serial-line.h"
#include "dev/sys-ctrl.h"
#include "net/netstack.h"
#include "net/rime/broadcast.h"
#include "reg.h"

#include <stdio.h>
#include <stdint.h>
static struct etimer et;
int16_t res;
/*---------------------------------------------------------------------------
*/
PROCESS(bw_test, "Bandwith_test");
AUTOSTART_PROCESSES(&bw_test);
/*---------------------------------------------------------------------------
*/

/*---------------------------------------------------------------------------
*/
PROCESS_THREAD(bw_test, ev, data)
{
  PROCESS_BEGIN();

  adc_zoul.configure(SENSORS_HW_INIT, ZOUL_SENSORS_ADC_ALL);

REG(0x400D7000)=0x95;  //ADCON1 ECO, ST,STSEL, - , - : 0X95 _FULL SPEED, 0XA5
_ TIMER1 EVENT (default)
  while(1)
  {
       REG(0x400D7008)=0xB5; //ADCCON3 EREF[2],EDIV[2],ECH[4], 0x85 7bit, 0xA5
10 bit  //ADC1 -AN5
       while(!(REG(SOC_ADC_ADCCON1) & SOC_ADC_ADCCON1_EOC)); //TRIGGERED BY
ADCCON3 WRITTING
       printf("%04X\n",(REG(0x400D7010)<<8)+REG(0x400D700C));
       REG(0x400D7008)=0xB4; //      //ADC2 - AN4
       while(!(REG(SOC_ADC_ADCCON1) & SOC_ADC_ADCCON1_EOC));
       printf("%d ",(REG(0x400D7010)<<8)+REG(0x400D700C));
      REG(0x400D7008)=0xB2; //       //ADC3 - AN2
       while(!(REG(SOC_ADC_ADCCON1) & SOC_ADC_ADCCON1_EOC));
       printf("%d ",(REG(0x400D7010)<<8)+REG(0x400D700C));
      REG(0x400D7008)=0xB7;//       //ADC5 - AN7
       while(!(REG(SOC_ADC_ADCCON1) & SOC_ADC_ADCCON1_EOC));
       printf("%d;",(REG(0x400D7010)<<8)+REG(0x400D700C));
/*----------------------delay ---------------------------------//
  etimer_set(&et, CLOCK_SECOND * 0.5);
  PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
//----------------------delay!!--------------------------------*/
}
  PROCESS_END();
}
```

Annex VII. Validation Algorithm

```
// VALIDATION
// We have to define a file caled "datos.txt" ("Consecutive" bytes in hex
form)
//gcc -o validation validationOfSequence.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define SIZE_DUMMY 11
#define PACKETS 100
int
htoi (char *p) {
    /*Look for 'x' as second character as in '0x' format*/
    if ((p[1] == 'x') || (p[1] == 'X'))
                return (strtol (&p[2], (char **) 0, 16)); else
                return (strtol (p, (char **) 0, 16));
}
int n_times_(int byte1,int byte2)
{ while((0x0FF&(byte1+100))!=byte2)
    {   return (n_times_(0x0FF&(byte1+100),byte2))+1;
    }
    return 1;
}
void read_data_segments_(char caracteres[], int *buffer__)
{   int byte1,byte2,byteresult;
    char dummy;
    int i,j;
            for (i = 0; i < sizeof (caracteres) - 1; i=i+2) {
                dummy = caracteres[i];
                byte1 = htoi (&dummy);
                dummy = caracteres[i+1];
                byte2 = htoi (&dummy);
                byteresult=(byte1<<4)+byte2;
                buffer__[j]=byteresult;
                j++;
            }
}

void findErros_(long Size, int *buffer_)
{ int i,noks=0,noseqs=0,bytes_er=0,bytestotal_err=0;
        for (i=0;i<Size/2;i++) {
            printf("%02X",buffer_[i]);
            if(  ((0x0FF)&(buffer_[i]+1))!=((0x0FF)&(buffer_[i+1])) &&
i!=((Size/2)-1))
            {   if(((0x0FF)&(buffer_[i]+2))!=((0x0FF)&(buffer_[i+1])))
                {
                          bytes_er=n_times_(buffer_[i],buffer_[i+1]-
1)*PACKETS;
                     bytestotal_err=bytes_er+bytestotal_err;
                       //

                noseqs++;
                printf("<< ERROR %d BYTES >>",bytes_er);
                }
                else{
                noks++;
                printf("| error |");
                }
            }
        }
        printf("\n\nByte ERRORS     : %d\n",noks);
        printf("FRAME ERRORS    : %d\n",noseqs);
        printf("RECEIVE         : %ld BYTES\n", Size/2);
        printf("LOST        : %d BYTES -->  %ld %% de TOTAL \n",
bytestotal_err,200*(bytestotal_err)/(Size+bytestotal_err));
        printf("TOTAL           : %ld BYTES \n", bytestotal_err+Size/2);
```

```
}

Int main () {
    FILE * archivo;
    char caracteres[SIZE_DUMMY];
    char dummy;
    long lSize;
    int i=0,j=0;
    int noks=0,byte1,byte2,byteresult;
    int *buffer;
//---------------------------------------------------------------------
    archivo = fopen ("datos.txt", "r");
    dummy=getc(archivo);
        if(dummy==0x2e)//0x2e= .  (point)
        {system("sed -i 1d datos.txt"); // for a line
        }
    fclose(archivo);
    system("sleep 1");
///---------------------------------------------------------------------
    archivo = fopen ("datos.txt", "r");
    if(archivo == NULL){
        exit (1);
    }
    else {
        fseek (archivo , 0 , SEEK_END);
        lSize = ftell (archivo);
        buffer = (int*) malloc (sizeof(int)*lSize);
        rewind(archivo);//printf("%ld\n",lSize);
        printf ("\n\n DATOS:\n\n");

        //fsetpos (archivo, &position);
        //printf("%d",fseek (archivo,10,SEEK_SET));
        while (!feof (archivo)) {
            fgets(caracteres, sizeof (caracteres), archivo);
            //read_data_segments_(caracteres,buffer);

            for (i = 0; i < sizeof (caracteres) - 1; i=i+2) {
                dummy = caracteres[i];
                byte1 = htoi (&dummy);
                dummy = caracteres[i+1];
                byte2 = htoi (&dummy);
                byteresult=(byte1<<4)+byte2;
                buffer[j]=byteresult;
                j++;
            }
        }
    }
    findErros_(lSize,buffer);
    fclose (archivo);
    free(buffer);
    return 0;
}
```