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

MASTER THESIS

TITLE: Internet of things implementation with Raspberry Pi

MASTER DEGREE: Master in Science in Telecommunication Engineering & Management

AUTHOR: Nasarre Ramírez, Alex

DIRECTOR: Polo Cantero, José

DATE: November, 10th 2014

Title: Internet of things implementation with Raspberry Pi

Author: Nasarre Ramírez, Alex.

Director: Polo Cantero, José.

Date: November, 10th 2014

Overview

The aim of this master thesis is to build a low cost wireless sensor network with a Raspberry Pi.

At the day of today the presence of internet in quotidian elements has continually grown over the time. These sensing elements require an interconnection between them to transmit raw data to be processed, also requires some specific goals like low-cost, low consumption and low resources in a wireless transmission to ensure that the technology can be used in common elements like temperature, movement, humidity, speed sensors, measures, surveillance, etc.

The present document provide a guide to implement and develop a coordinator of this wireless network solving some problems that current IPV4 networks have and providing a new scenario where a sensor network can be built to in a low-cost infrastructure.

To achieve that end, several steps are realized. First, technologies are studied, compared and selected with specific criteria. Second, a solution is developed, built and tested and finally some conclusions are taken.

INDEX

INTRODUCTION 9

CHAPTER 1. STATE OF THE ART 10

1.1. WSN 10

1.2. Internet of Things 11

CHAPTER 2. STANDARDS 14

2.1. IPv6 14

2.2. IEEE 802.15.4 14
2.2.1. IEEE 802.15.4 features 15
2.2.2. Architecture 16
2.2.3. IEEE 802.15.4 frame format 16
2.2.4. IEEE 802.15.4 MAC layer operation modes 17

2.3. 6LoWPAN 18
2.3.1. 6LoWPAN protocol stack 19
2.3.2. 6LoWPAN features 20
2.3.3. 6LoWPAN architecture 22

2.4. IETF RPL 24

2.5. CoAP 25

CHAPTER 3. AVAILABLE EQUIPMENT 27

3.1. Raspberry Pi 27

3.2. Xbee module 28

3.3. Arduino Duemilanove 29

CHAPTER 4. TECHNOLOGY REVISION 31

4.1. Open-software to develop IoT 31
4.1.1. TinyOS 31
4.1.2. Contiki 31

4.2. Existing solutions 31
4.2.1. Grinch project 32
4.2.2. JenNet-IP Border-Router 32
4.2.3. NanoRouter 2.0 33
4.2.4. Cisco 500 Series WPAN Industrial Router (IR500) 35
4.2.5. Arduino IPv6 Stack 35
4.2.6. Redwire BR12 37
4.2.7. Redwire RedIO 38
4.2.8. 6LoWPAN Border Router (6LBR) 38
4.2.9. BLIP 2.0 41
4.2.10. Openlabs RPi 802.15.4 Radio 42

4.3. Comparative 6LoWPAN solutions 42

CHAPTER 5. TECHNOLOGY DEVELOPMENT 46

5.1. 6LBR Development 46

5.2. Openlabs 802.15.4 module Development 48

5.3. Testing scenarios 51
5.3.1. 6LBR scenarios 51
5.3.2. Openlabs compatibilities 55

CHAPTER 6. CONCLUSIONS AND FUTURE LINES 57

6.1. Conclusions 57

6.2. Environmental impact 57

6.3. Future lines of development 58

REFERENCES 59

GLOSSARY 64

ANNEX 67

A.I IP features 68
A.I.1 The TCP/IP Network Stack 68
A.I.2 IPv6 Header 68

A.II Hardware Support 68
A.II.1 Tiny OS hardware support 68
A.II.2 Linux kernel hardware support (Openlabs) 69
A.II.3 Contiki hardware support 69

A.III Openlabs code 70
A.III.1 IPv6 UDP Server 70
A.III.2 Openlabs IPv6 UDP Client 72

A.IV. XBee Configuration 74

A.V. RPi basic software installation 74

A.VI. Arduino pIPv6 stack compilation 75

A.VII. NAT66 implementation tutorial 76

A.VIII. RADVD Installation and configuration 77

FIGURES INDEX

Fig. 0.1. FPMT area to deploy WSN 9
Fig. 1.1. WSN Topologies 11
Fig. 1.2. Hype cycle of emerging technologies 12
Fig. 1.3. Example of IoT application 13
Fig. 2.2. 802.15.4 Topology 16
Fig. 2.3. IEEE 802.15.4 Frame format (PHY layer) 17
Fig. 2.4. IEEE 802.15.4 Frame Format (MAC layer) 17
Fig. 2.5. Beacon-enabled access mode (at 250 Kb/s) 18
Fig. 2.6 IP protocol stack 20
Fig. 2.7. IPv6 Address compression 20
Fig. 2.8. 6LoWPAN / UDP headers (compressed) 20
Fig. 2.9. LoWPAN routing "Route-Over" 21
Fig. 2.10. Data Link Layer/Mesh forwarding "mesh under" 21
Fig. 2.11. LoWPAN/Mesh forwarding "Mesh under" 22
Fig. 2.12. 6LoWPAN fragment. Initial (left) and non-initial (right) 22
Fig. 2.13. 6LoWPAN Architectures 23
Fig. 2.14. Non-storing mode (left) and storing mode (right) 24
Fig. 2.15. Abstract Layering of CoAP 25
Fig. 2.16. CoAP frame format 26
Fig. 2.17. CoAP request/response model. Confirmable (left) and non-confirmable (right) 26
Fig. 3.1. Raspberry Pi model B 27
Fig. 3.2. XBee Pro Wire antenna 29
Fig. 3.3. Xbee shield for Arduino 30
Fig. 4.1. Grinch border router project 32
Fig. 4.2. JenNET-IP example network 32
Fig. 4.3. JenNET-IP Stack 33
Fig. 4.4. Sensinode NanoRouter diagram 34
Fig. 4.5. Cisco IR500 industrial router 35
Fig. 4.6. pIPv6 stack (left) and uIPv6 stack (right) 36
Fig. 4.7. Arduino heterogeneous 6LoWPAN network 36
Fig. 4.8. Arduino shield for Raspberry Pi 37
Fig. 4.9. Redwire BR12 37
Fig. 4.10. Redwire IO 38
Fig. 4.11. Bridge mode 39
Fig. 4.12. Router Mode 39
Fig. 4.13. Smartbridge Mode 40
Fig. 4.14. Router mode 40
Fig. 4.15. Transparent Bridge 41
Fig. 4.16. BLIP 2.0 Stack 41
Fig. 4.17. RPi with Attached Openlabs module 42
Fig. 5.1. 6LBR webserver 47
Fig. 5.2. OpenLabs module interfaces 51
Fig. 5.3. OpenLabs module interfaces 52
Fig. 5.4. 6LBR WSN configuration 52
Fig. 5.4. Reachable neighbors detected by webserver 53
Fig. 5.5. 6LBR sensor detection. 53
Fig. 5.6. 6LBR Network detection 54

Fig. 5.7. 6LBR Network detection 55
Fig. 5.8. Openlabs dialog with pIPv6 stack 56
Fig. 5.9. Openlabs dialog with pIPv6 stack 56
Fig. A.1. IP protocol stack 68
Fig. A.2. IPv6 Fixed Header (320 bits). 68
Fig. A.3. USB to TTL Serial Cable 75
Fig. A.4. Shared internet connection scheme 75

TABLES INDEX

Table 2.1. 802.15.4 Main features 15
Table 2.2. RFD and FFD properties 16
Table 2.3. Upper layer protocols 19
Table 3.1. Raspberry Pi basic features 28
Table 3.2. XBee Pro Wire antenna specs 29
Table 3.3 Arduino Duemilanove main features 30
Table 4.1. JenNet-IP features 33
Table 4.2. NanoRouter 2.0 features 34
Table 4.3. Cisco IR500 features 35
Table 4.4. IPv6 Stack features 37
Table 4.5. Redwire BR12 features 37
Table 4.6. Redwire IO features 38
Table 4.7. Redwire 6LBR features 38
Table 4.8. Comparative of solutions 45
Table A.1 Contiki hardware support 69

INTRODUCTION

The aim of this master thesis is to build a low cost wireless sensor network with a Raspberry Pi (RPi) and apply it into the Mediterranean Park of Technology (PMT) (see [14]) in Castelldefels (Barcelona) where is placed the UPC-EETAC, UPC ESAB universities and research entities such as ICFO, i2Cat, CTTC, etc. Potentially, PMT has 3.800.000 m^2 (38 hectares) to cover with a WSN.

Fig. 0.1. FPMT area to deploy WSN

Probably, in early stages of the development can be just a small area with a few nodes connected in, weather sensors for example, but at the end of the development can be used, apart from weather sensing, for surveillance, some lab tests (speed, pressure, material presence, flexibility, etc.).

This document is more focused in the coordinator of the WSN that will be based on Raspberry Pi, this device has enough capabilities to realize this task and to ensure a communication with other nodes of the WSN.

This document is distributed mainly in 6 chapters. The general concepts among the WSN will be described on first chapter. In chapter 2 is discussed the most important standards that use actually deployed WSN based on Internet of Things (IoT) networks. Chapter 3 describes the available equipment on the lab that can help the establishment of the IoT network. Chapter 4 takes principal attention on the market and the available solutions to establish a WSN. In Chapter 5 is described the deployment of a solution to establish the PMT-WSN according specific criteria. Finally Chapter 6 takes some conclusions about the deployed solution and proposes some future lines.

CHAPTER 1. STATE OF THE ART

The goal of this chapter is to describe the general concepts, key points, specifications and how this technology is available and how is involved into the construction of the WSN.

1.1. WSN

WSN is a collection of autonomous nodes (with one or several sensors per node) distributed in an area with the capability to interact with other elements and share data with a coordinator (base station). The mission of the coordinator is to collect, process, treat and manipulate the information to get human-readable data.

Some properties of the WSN are described in these main points:

- **Elements**:
  - *Sensors*: elements that have the capability to transduce from non-electrical to electrical quantity (see [9]).
  - *Nodes*: elements that collect the information of its sensors and sends to the base station.
  - *Base station*: is the element that gathers all the information that becomes from the nodes and processes it.
  - *Wireless network*: protocol that specifies the rules and how-to realize the interconnection of the elements.
  - *Gateway*: elements that make possible the interconnection between TCP/IP systems and WSN.

- **Components**:
  - *Radio transceiver with antenna*: an element that transmits and receives data from coordinator or other nodes.
  - *Microcontroller*: a processing unit to compute the operations related to get the data from the sensors and send it to the transceiver or receive the data and does the related actions.
  - *Sensor interface*: port that connects and makes able the dialog between sensor and the microcontroller.
  - *Energy source*: a unit that provides a power supply to make functional the node.

- **Communication**:
  - *Unidirectional*: the communication is from sensor to coordinator just to send the data.
  - *Bi-directional*: the communication is sensor to coordinator, to collect data, and vice versa, for OAM tasks of the nodes.
- **Topologies:** bus, tree, star, ring and mesh.

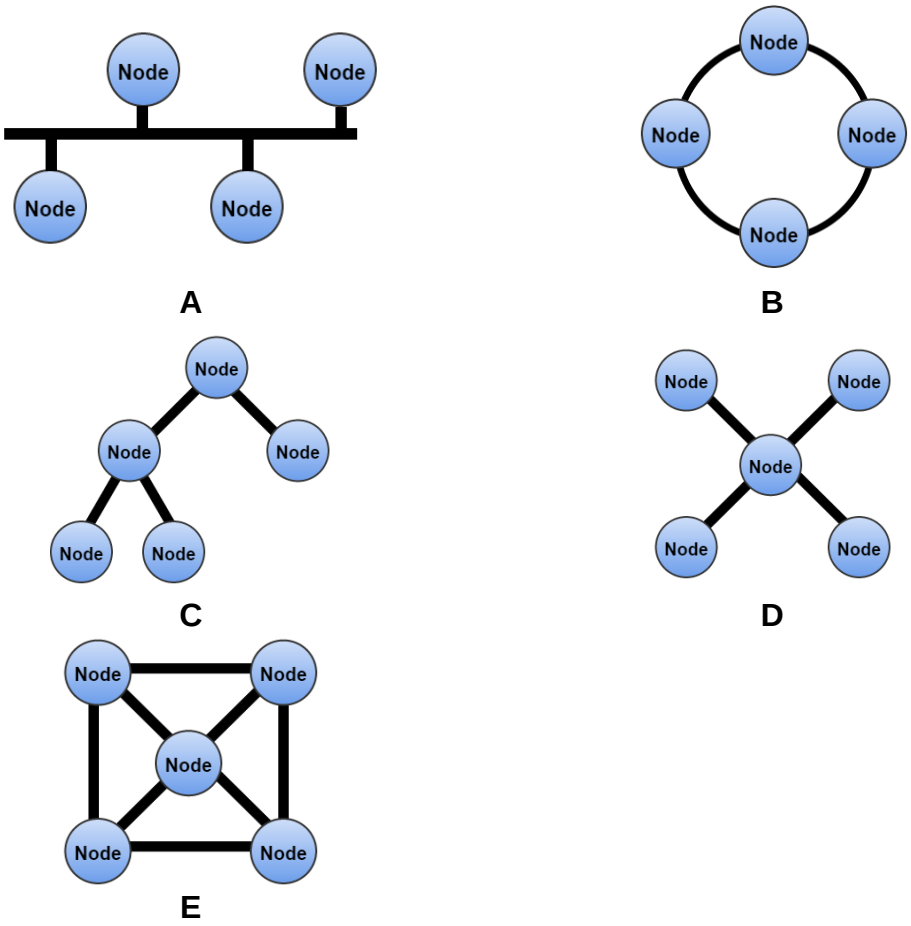

Fig. 1.1. WSN Topologies ,bus (A), ring (B), tree (C), star (D) and mesh (E)

- **Propagation**:
  - o *Routing:* establishes the best path to arrive from A to B taking into consideration some metric (minimum number of hops, maximum bit rate, etc.). The complexity due to establish the routing mechanisms is higher but reduces the overhead and the transmission of packets among the network.
  - o *Flooding:* sends the message to all possible interfaces and destinations until arrives the message to the destination. It takes less complexity compared with routing but the collisions and the overhead are much higher.

1.2. Internet of Things

IoT is a network of physical objects that contain embedded technology to communicate and sense with their internal states or the external environment and interconnect this data to the existing internet infrastructure. This concept covers multiple protocols, systems and applications that make possible this connection between these devices called "things".

Looking at Gartner (see [10]), IoT is on the top of peak of the hype cycle of emerging technologies, that means that at the day of today is a technology for start-ups that wants to be on the top of an specific market.

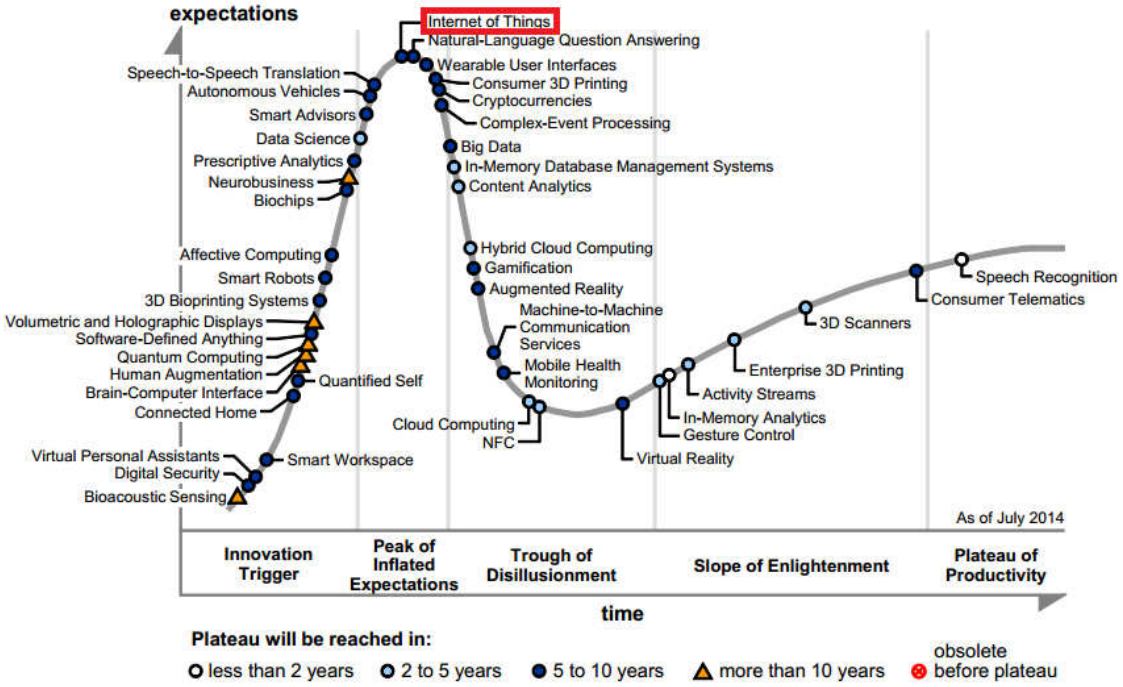

Fig. 1.2. Hype cycle of emerging technologies *(source: www.gartner.com)*

There are positive aspects that push forward IoT technology as the reduction of costs in the technology related with IoT and a diversity IoT start-ups, competing to establish their solution. By the way, IoT to be a mature tech needs to go deeply on standardization (data standards, wireless protocols, technologies) to extend the possibilities of IoT. However, the competition of entities involves a struggle to obtain the control of the IoT in the market, having less standards than expected and harming the evolution of the technology.

IoT has very broad applications but basically focused in four main areas or a combination of them:

- **Management**: monitorization and optimization of connected things to use them in a best performance instant. For example, cooling systems to use them in some specific conditions.
- **Payment**: monetization of the connected things on a pay-per-use basis. For example, car parks.
- **Operation**: remote operation to avoid on-site operations such as valves and actuators operations.
- **Extension**: things connected can have extra functionalities connected with services such as software upgrades that improve functionalities.

IoT can be introduced into a huge quantity of devices, for example monitoring activities like heart rate, water leverage, gas levels, or automotive autodiagnosis

as well. If we introduce WSN and IoT in a city the concept of smart city has born, looking to an example in the following figure:

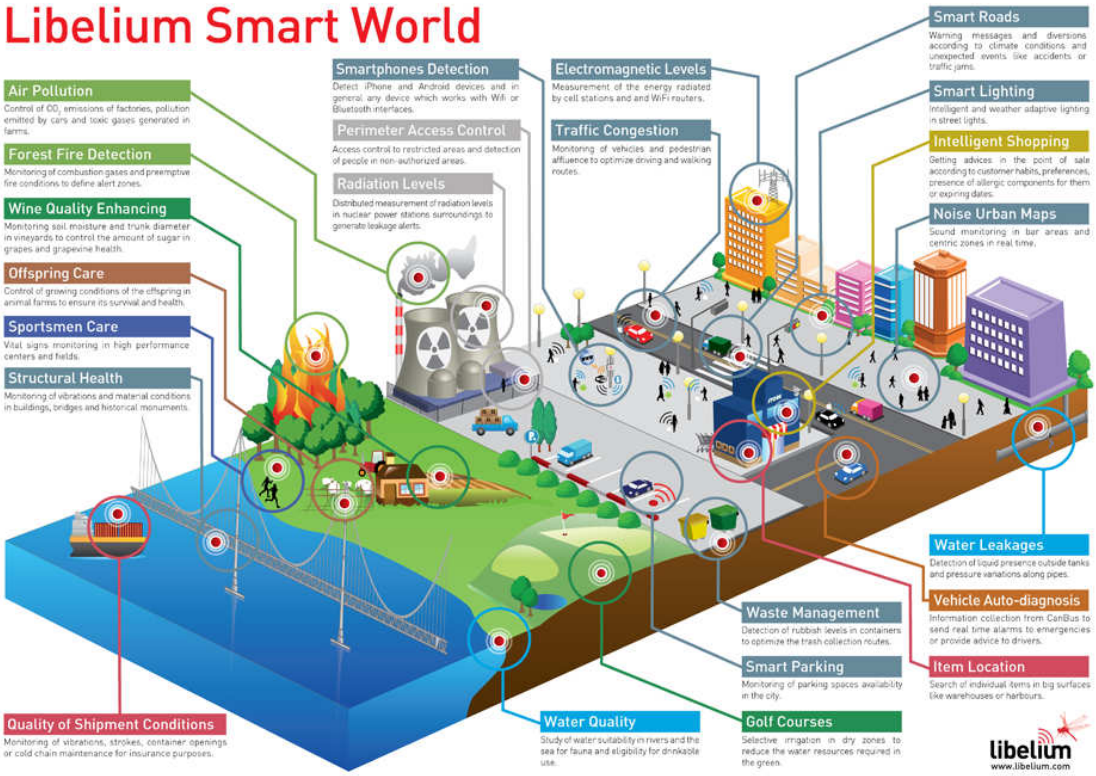

Fig. 1.3. Example of IoT application *(Source: www.libelium.com)*

As seen in the previous figure, is possible to introduce IoT in a huge quantity of fields making an evolution of traditional IP networks due to "things-things", "humans-things" communications and hybrid communications things-human (see [38]).

To mention some vendors that are involved in IoT are IBM, Honeywell, General Electric, Cisco, Schneider Electric, Siemens, Microsoft, NI, etc.

CHAPTER 2. STANDARDS

2.1. IPv6

"With IoT will include 26 billion units installed by 2020" (see [11]), that means 26 billion of things connected to internet potentially. Using actual IPv4 protocol represents that 32 bits are dedicated to provide 4.294.967.296 unique available addresses. Making a calculation with an IPv4 protocol overpasses up to 6 times the available addresses of the connected devices. Due to this problematic is required a new protocol to support this quantity of nodes. The technology must be an IP-based technology in the IoT network for some aspects:

- **Compatibility:** IP-based technology can use existing IP network infrastructure (see IP stack on A.I Annex).
- **Connectivity:** IP-based technology can be connected to other IP networks without using translation gateways or proxies.
- **Mature technology:** IP-based technologies are very well known, and have been proven to work and scale.
- **Open:** IP technology is specified in an open and free way, with standards processes and documents available..
- **OAM:** tools for managing, commissioning and diagnosing IP-based networks already exist.

Due to this aspects is reasonable to use an extension of the protocol that makes possible to connect a huge quantity of devices, this protocol is IPv6. IPv6 provides 2^{128} addresses that makes with the Gartner prediction that the load of addresses in 2020 will be approximately $7{,}64*10^{-27}$% of available addresses. To conclude, IPv6 have some interesting features to deploy WSN and IoT in general but is required to solve some drawbacks such as compression and the higher computation in IPv6, this aspects compared with IPv4 it requires more resources. (more information of IPv6 in Annex A.I).

2.2. IEEE 802.15.4

In IoT there is a set of required or recommended features that a link should provide in order to work with Internet protocols. The basic one are described below.

- **Standard**: have a protocol that establishes a common framework for all the entities that want to develop IoT related devices having a interoperability between different vendors.
- **Wireless**: that technology implies reduce costs in terms of installation avoiding cables and their maintenance. However is required to deploy low cost hardware to obtain feasible solution.

- **Very low-Power consumption**: IoT is focused in low-cost devices with limited power supply, thus require a "lightweight" protocol to establish the link in a simple way with low-power consumption.
- **Framing:** use blocks to communicate in a standardized way between the different elements of the network
- **Addressing:** to differentiate between nodes on a link with an unique address.
- **Unicast transmission:** communication one-to-one between elements.

Looking at existing protocols is possible to think about IEEE 802.11 family protocol (WLAN), but WLAN just accomplishes link requirements. Another possibility is to think about IEEE 802.15.1 or Bluetooth (WPAN) protocol, but at this moment this protocol is more focused in high data rate (not strictly required in WSN) and security in WPANs, providing high cost in terms of hardware and not as very low-power consumption as is desired. The solution is to design new protocol, and this protocol is 802.15.4.

2.2.1. IEEE 802.15.4 features

The IEEE 802.15.4 standard defines low-power wireless embedded radio communications. The first version of the standard was released in 2003 but are continuously evolving in new revisions such as IEEE 802.14.5 (CSS and UWB), IEEE 802.15.4 TG4e, TG4f (active RFID), TG4a (larger networks), TG4c (china networks) and TG4d (Japan networks). With IEEE 802.15.4 is possible to achieve a very low-cost and very low-power communications creating LR-WPANs. Basically the standard offers MAC and PHY layer protocol (OSI 1 and 2 layers) to establish a point-to-point communication between two devices. The main characteristics of IEEE 802.15.4 are described in the following table:

Table 2.1. 802.15.4 Main features

	2.4 GHz ISM band	**915 MHz (USA)**	**868 MHz (EU)**
Operating Bands	Q-QPSK at 250 kb/s	BPSK at 40 kb/s	BPSK at 20 kb/s
Output Power	20 dBm	> 10 dBm	30 dBm
Channel Numbers	11-26	1-10	0
MTU	127 Bytes per frame (including headers)		
Payload	72-116 Bytes per frame		
Channel Sharing	CSMA/CA		
Link layer security	128-AES Encryption		
Addressing	64 bit (long) or 16 bit (short) addressing with unicast and broadcast capabilities		

2.2.2. Architecture

In terms of devices the protocol 802.15.4 establishes two types: FFD and RFD, having at least one FFD that acts as PAN coordinator. The main characteristics of these devices are described in the following table:

Table 2.2. RFD and FFD properties

	FFD	RFD
Topology supported	Star, P2P	Limited to leafs
Coordinator supported	Yes	No
Protocol set	Complete	Reduced

An example of topologies a star topology (based on master-slave) and a P2P topology (point to point between RFD devices) establishing extra point to point channels is presented in the following figure:

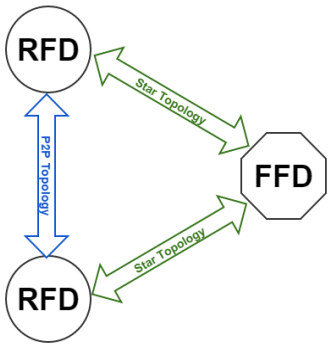

Fig. 2.1. 802.15.4 Topology

2.2.3. IEEE 802.15.4 frame format

To establish the links that are described previously, IEEE 802.15.4 standard specifies a frame format to make this communication. In IEEE 802.15.4 are defined basically three types of frames:

- **Data frames:** for the transport of actual data.
- **Acknowledgment frames:** is the response of a received data frame, this acknowledgement is well-defined in "acknowledgment request" of the frame control fields.
- **MAC layer command frames:** frames that are used to associate and disassociate from a coordinator, management of a synchronized transmission and to structure a communication between the coordinator and the associated nodes.

These types of frames are implemented on 802.15.4 MAC header that can be up to 25 Bytes and 127 Bytes in terms of frame (without security mechanisms), the format of the header and the frame are presented in the following figures:

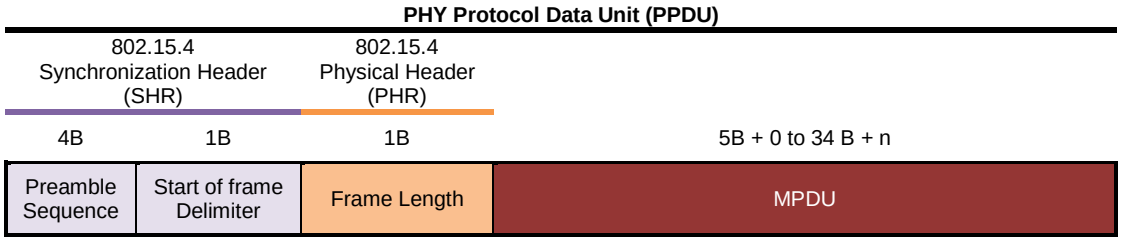

Fig. 2.2. IEEE 802.15.4 Frame format (PHY layer)

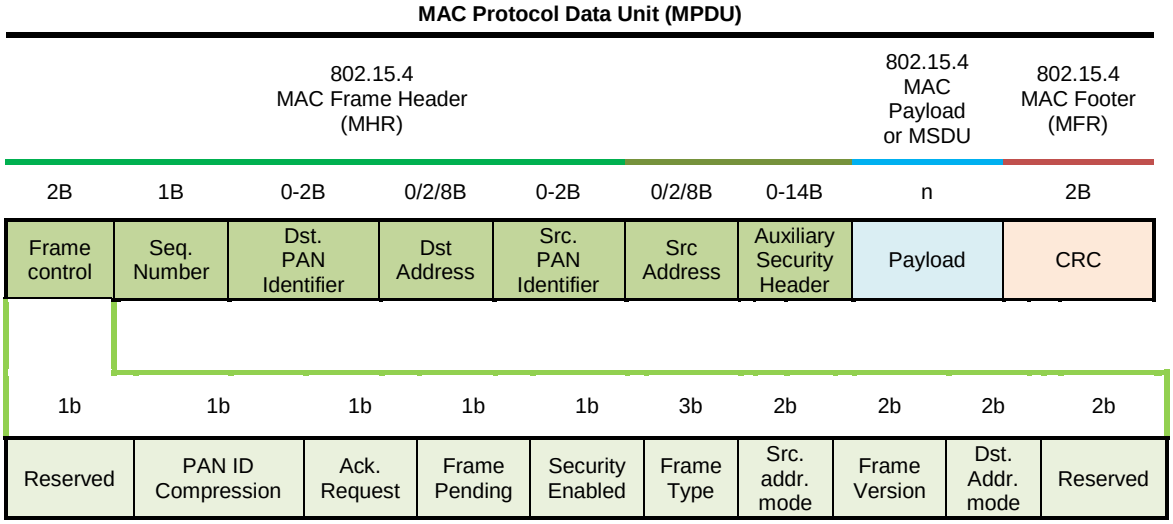

Fig. 2.3. IEEE 802.15.4 Frame Format (MAC layer)

2.2.4. IEEE 802.15.4 MAC layer operation modes

The MAC layer can be run in two modes: beaconless mode and beacon-enabled mode. Beaconless mode uses pure CSMA/CA:

- When a device wants to transmit, first waits until the channel is available.
- Once the channel is idle, the device starts sending the data frame after some random back off interval.
- Receiver acknowledges the correct reception of a data frame.
- If the sender does not receive an acknowledgement, retries the data transmission until a certain number of retransmissions.

However, beacon-enabled mode has a TDMA approach combined with a superframe structure with a maximum number of 15 slots, where on CAP the channel is available using previous commented CSMA/CA and on CFP includes reserved time-slots for priority data in GTS, assigned by the coordinator, to each node.

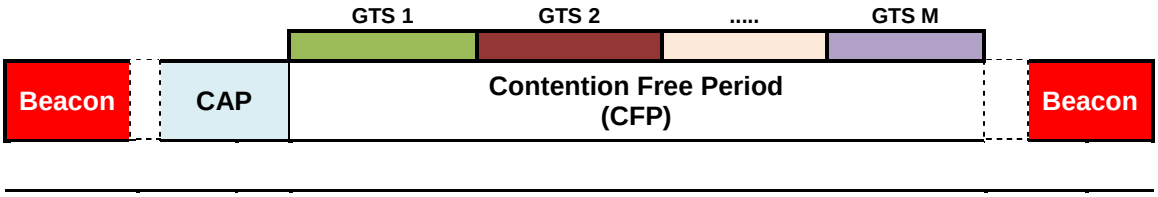

Fig. 2.4. Beacon-enabled access mode (at 250 Kb/s)

2.3. 6LoWPAN

In the previous subchapter are established OSI layers 1 and 2 of the stack for the IoT project network, also is clear that it could be good to include IPv6 technology in the stack. For that reason with IEEE 802.15.4 is not enough, because is required to know how to solve some aspects that will be described in this subchapter.

At first, standard IPv6 packet has a minimum MTU of 1.280 Bytes and IEEE 802.15.4 has a MTU of 127 Bytes, is important to find mechanisms to adapt IPv6 with IEEE 802.15.4 frames. Another important thing is how to include protocols that uses Internet services such as TCP (adapted to wireless networks), HTTP, SOAP, XML, JSON, etc. And to conclude is important to think in terms of security. to offer optional support for IPSec (more detail in [15]) authentication and encryption.

Thinking about IP technology is important to mention that IP technology is conceived to be on an "always-connected" device, that idea is totally opposite of low-consumption equipment that has some sleep periods. On the other hand, it is interesting to include multicast and flooding support that is not supported on IEEE 802.15.4 layers. Additionally, one of the most important aspects is to achieve multihop mesh topologies to take benefit in terms of wireless coverage. To conclude, since every device can have a routable IP the administration will be easier using standard tools (web-based, ssh, telnet, etc.) but the security will have an important paper to avoid unauthorized access.

However is difficult to realize these previous mentioned improvements in limited devices, in terms of performance, for a very low consumption. The duty now is how to solve these issues and see how are solved at the day of today. On the upper layers of IEEE 802.14.5, there are several quantity of protocols. In the following table are mentioned the majority of these protocols:

Table 2.3. Upper layer protocols

Name	Description
Zigbee	Proprietary mesh networking protocol
MiWi Mesh	
SimpliciTI™	
MiWi P2P	Proprietary P2P networking protocol
WirelessHART	Industrial automation
ISA100.11a	Wireless for manufacturing, control and automation
6LoWPAN	IPv6 over IEEE 802.15.4
IP500	6LoWPAN over IEEE 802.15.4 sub-GHz radio communications

Considering all previously discussed, the best solution is to develop a 6lowPAN protocol in upper layer of IEEE 802.15.4 because is open standard and makes able to use IPv6 with all the described properties.

The definition of 6LoWPAN: "6LoWPAN standards enable the efficient use of IPv6 over low-power, low-rate wireless networks on simple embedded devices through an adaptation layer and the optimization of related protocols" (see [12]). Hence, it's clear that accomplishes the requirements to be used it for IoT networks.

The IETF 6LoWPAN working group (2005) was created to enable IPv6 to be used, in combination with IEEE 802.15.4., with wireless embedded systems. Additionally, to establish minimum requirements for implementing a lightweight IPv6 stack with the most minimal devices. Finally, by designing a Neighbour Discovery (ND) version, taking into consideration low-consumption wireless mesh networks. The result of 6LoWPAN is a protocol that enables end-to-end IP networking and features for embedded applications.

In terms of improvements, there are different groups such as 6LoWPAN-hc (header compression), 6LoWPAN-nd (neighbour discovery), 6LoWPAN-uc (use cases), 6LoWPAN-rr (routing requirements) and Routing Over Low-power and Lossy Networks (ROLL).

2.3.1. 6LoWPAN protocol stack

A simple 6LoWPAN protocol stack is very similar to an standard IP stack with some differences. First, 6LoWPAN only supports IPv6, which a small adaptation layer (LoWPAN adaptation layer) has been defined to optimize IPv6 over IEEE 802.15.4 and similar link layers, allowing stateless compression of IPv6 and following headers such as UDP along with fragmentation and mesh addressing features.

In transport layer, the most common transport protocol used is UDP, which can also be compressed using the LoWPAN format. Also, Internet control message

protocol v6 (ICMPv6) is used for control messaging, for example ICMP echo, ICMP destination unreachable and Neighbour Discovery messages.

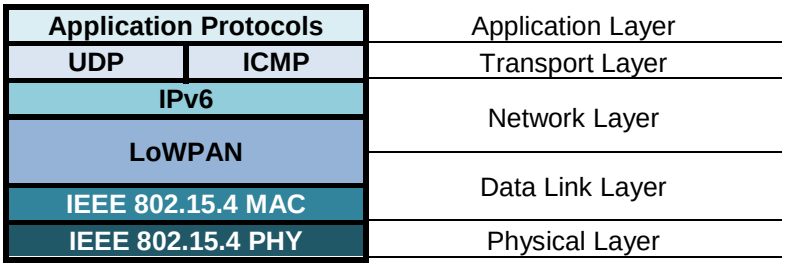

Fig. 2.5 IP protocol stack

2.3.2. 6LoWPAN features

There are some important features to mention in 6LoWPAN, one of them is the high IPv6 header compression ratio, that it's possible due to two actions on the IPv6 address. The first one is a direct relation of IID with the link layer address, that feature simplifies the task of address resolution. The second aspect is IPv6 prefix acquirement from Neighbour Discovery RA messages as IPv6. The combination of both features makes a compression of addressing in 6LoWPAN.

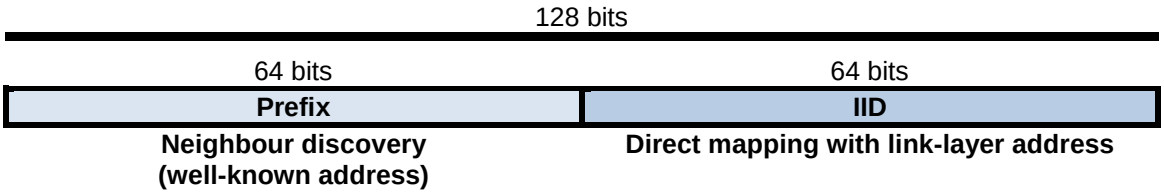

Fig. 2.6. IPv6 Address compression

By the way, LoWPAN header consists of a dispatch value identifying the type of header, followed by an IPv6 header compression byte (LOWPAN_IPHC) indicating which fields are compressed, and then any in-line IPv6 fields (UDP, IPv6 extension, etc.) in next-header compression byte (LOWPAN_NHC).

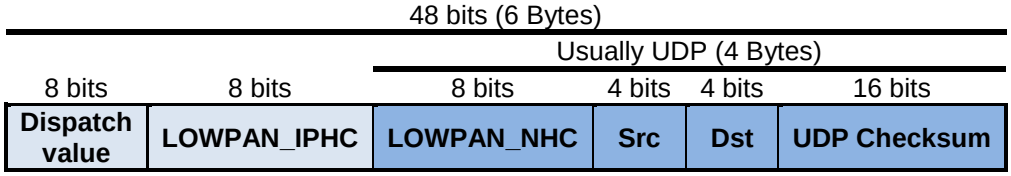

Fig. 2.7. 6LoWPAN / UDP headers (compressed)

One important feature in 6LoWPAN is 6LoWPAN-ND that becomes from IPv6 but is optimized for low-power wireless networks and specifies the operation of

the elements and the auto configuration of these elements on the network. Another interesting feature of 6LoWPAN-ND is a registry of nodes stored in the edge router becoming from all the LoWPAN domain to reduce the quantity of messages and operations inside the LoWPAN.

A key feature is reaching a mesh topology with multihop forwarding, this is one of the most important features because extends the range of the network with a reduction of the cost in infrastructure. In 6LoWPAN, this can be done in three different ways: link-layer mesh, LoWPAN mesh or IP routing. Actually, to realize multihop forwarding is used IP routing as the same as an IP stack, an algorithm updates a routing table which IP uses to make next-hop decisions and establishes an end-to-end communication. RIB usually can be simplified to a FIB, which is consulted when a packet arrives that needs to be forwarded. In the following figures will be shown forwarding in DLL, LoWPAN and IP routing.

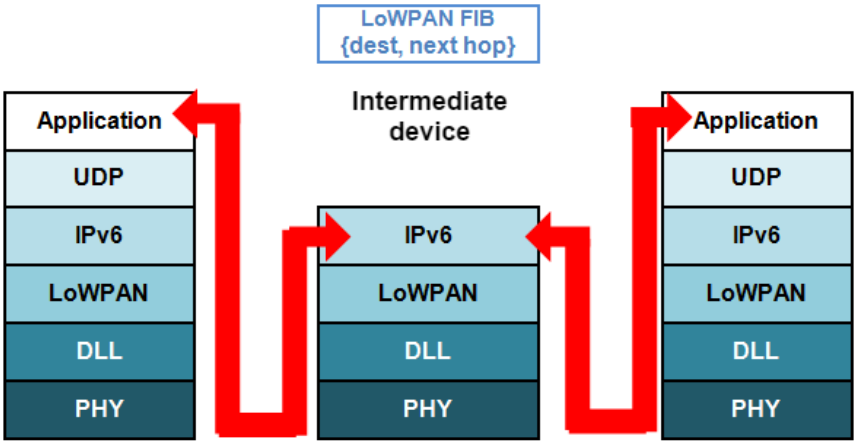

Fig. 2.8. LoWPAN routing "Route-Over"

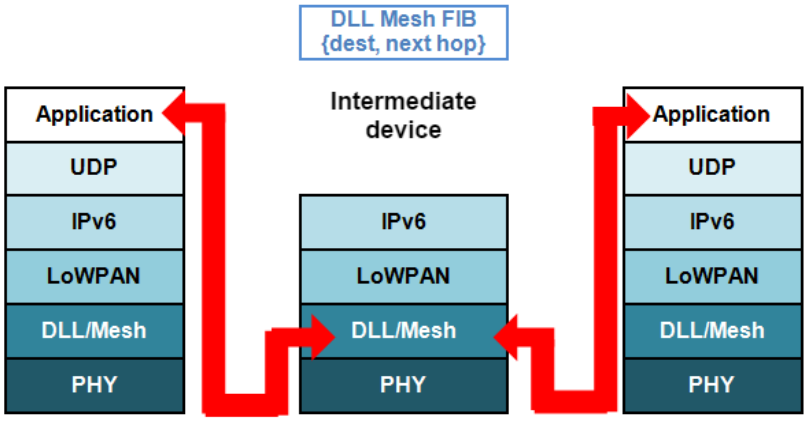

Fig. 2.9. Data Link Layer/Mesh forwarding "mesh under"

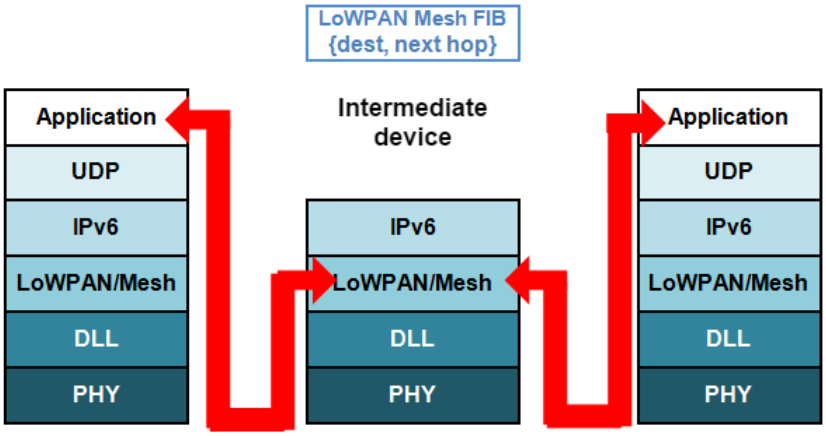

Fig. 2.10. LoWPAN/Mesh forwarding "Mesh under"

Other important functionalities of 6LoWPAN are described below:

- **Multi-homing:** that is the presence of the device in different LoWPANs at the same time.
- **Flexibility:** the device can change, due mobility or better channel conditions, to other edge router or LoWPAN even a dynamic topology change.
- **Fragmentation and reassembly:** with this technique is possible to fit up to 2.047 Bytes of frames becoming from IP-Layer to the Link-Layer. Even if the fragmentation is possible, this is not a desirable aspect due to have too much overhead. It's recommended an UDP-based application that uses less than 60 bytes of payload to avoid fragmentation.
- **Bootstrapping**: auto-setup to integrate a device into the 6LoWPAN network.

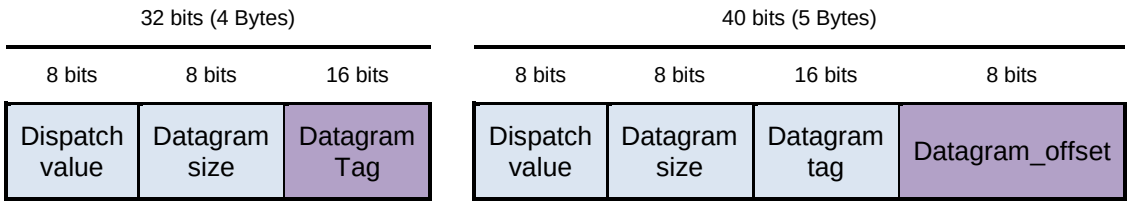

Fig. 2.11. 6LoWPAN fragment. Initial (left) and non-initial (right)

2.3.3. 6LoWPAN architecture

6LoWPAN establishes three main types of architectures based on different types of LoWPAN ,understanding the term LoWPAN as the collection of devices with the same IPv6 prefix (first 64 bits of the address). The architectures are: ad-hoc LoWPAN, Simple LoWPAN and Extended LoWPAN.

Ad-Hoc LoWPAN is an island of nodes with FFD and RFD without an internet connection but with themselves is not required any infrastructure to work with. One router must be configured to act as a simplified edge router, implementing two basic functionalities: unique ULA generation and handling 6LoWPAN Neighbour Discovery registration functionality.

The main difference between a Simple LoWPAN and Extended LoWPAN is the existence of multiple edge routers in the LoWPAN, which share the same IPv6 prefix and a common backbone link. When a node is moving from one LoWPAN to another, IPv6 address will change. In an Extended LoWPAN configuration are offloading most Neighbour Discovery messaging to the backbone link, simplifying LoWPAN Node operation with stable IPv6 addresses throughout the Extended LoWPAN. The movement between edge routers is very simple similar to a WLAN access point infrastructure (but at layer 3 instead of layer 2).

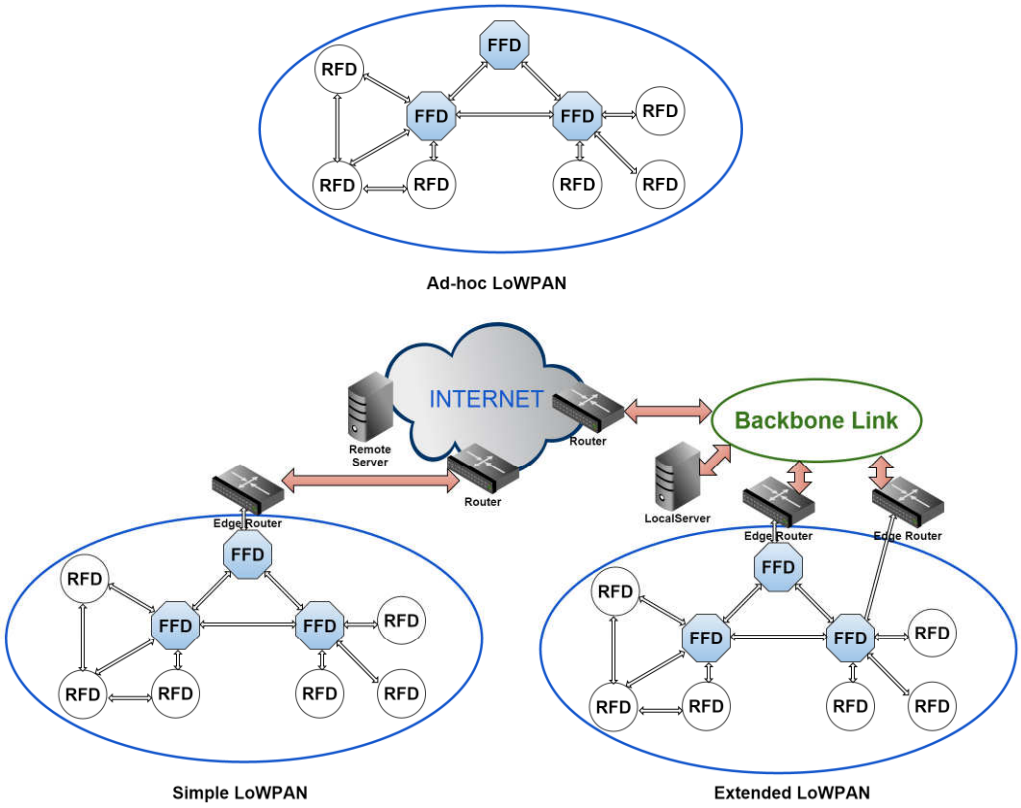

Fig. 2.12. 6LoWPAN Architectures

As seen in previous figure, in Simple and Extended LoWPAN architectures has an edge router, this router apart from, obviously, route the traffic out of the LoWPAN domain, also handles the 6LoWPAN compression, realizes the Neighbour Discovery and, to conclude, is responsible of the interconnectivity with IPv4 network (IPv6 to IPv4 conversion). The transformation of IPv6 to 6LoWPAN is transparent, efficient and stateless in both directions. Inside the LoWPAN, the devices do not actually need to work with full IPv6 or UDP header formats at any point as all compressed fields are implicitly known by each node.

2.4. IETF RPL

RPL is a routing protocol for LLN in IP layer. The protocol is especially for a networks that have a transmission media with high probability of losses, sometimes with low line of sight and high variable transfer rates.

Basically RPL specifies how to build a DODAG based on metrics and constrains to find best paths, for example: best transmissions avoiding non-encrypted links or best latency avoiding battery operated nodes. The result of the operation is a set of logical topologies that is possible set active with different approaches at the same time, involving nodes with different types of graphs and mark the traffic according the QoS and the constraints.

To interconnect with the other elements RPL specifies three ICMPv6 control messages to exchange graph related information. The messages are DIS (DODAG Information Solicitation), DIO (DODAG Information Object) and DAO (DODAG Destination Advertisement Object).

RPL distinguishes three types of elements: gateway nodes, routers and leafs. Gateways provide connectivity with other networks, on the other hand routers advertise topology information among his neighbours and finally, the leafs just have the ability to join into an existing DODAG.

Going deeply into an operation modes RPL has two, non-storing and storing mode. The first one, the non-storing mode, every node sends a DAO message to the DODAG root, extending the information of the parent set to the root. The farthest node will generate DAO messages more often that a closer one, generating a rank of logical topology and storing a routing database in the DODAG root. In the case of storing mode, DAO messages also are generated but are not longer propagated to the DODAG root, in substitution are sent unicast messages to the parent nodes which maintains routing tables of down levels.

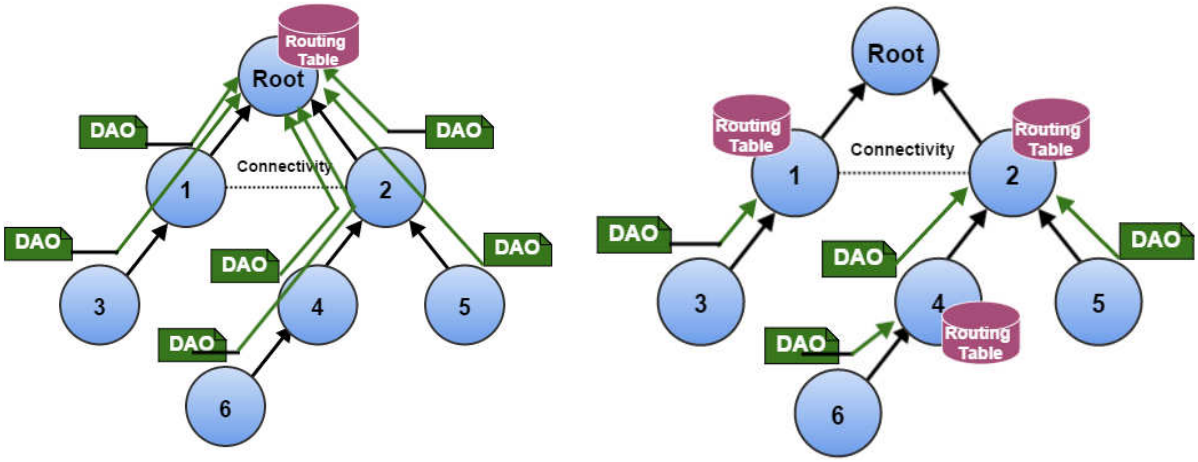

Fig. 2.13. Non-storing mode (left) and storing mode (right)

6LoWPAN networks support RPL in a "route-over" configuration. With RPL, 6LoWPAN routers operate as IPv6 routers and form routes using RPL. Border routers that connect 6LoWPAN networks to other IP networks will typically operate as RPL DODAG roots. Nodes then utilize RPL to form one or more routing topologies so that they can forward IPv6 datagrams to their destination.

Alternatively, a 6LoWPAN host may be agnostic routing-protocol by using the 6lowpan-nd protocol to discover neighbour routers, choose attachment routers, and notify one or more of those routers of the existence of new ones.

2.5. CoAP

CoAP by IETF CoRE working group is a specialized web transfer protocol for constrained nodes and LLN. The protocol works in application layer and it's designed for M2M communications with the intention to provide internet communication to a simple electronic devices.

CoAP provides request/response mechanism between application endpoints, supports built-in discovery of services and resources, includes URI's and content-type. CoAP is designed to deal with HTTP to integrate with the web easily, includes UDP support, multicast support, low overhead, less parsing complexity and statelessness. These features are important to IoT development.

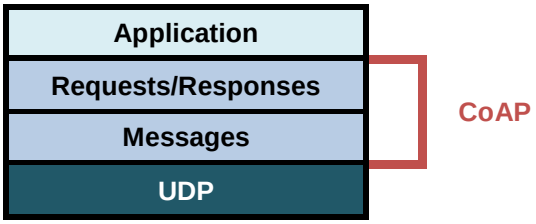

Fig. 2.14. Abstract Layering of CoAP

The interaction model of CoAP is client/server with asynchronous messages in UDP, these messages are four types: Confirmable, Non-confirmable, Acknowledgement and Reset. Also CoAP has methods that become very similar with HTTP: GET, POST, PUT and DELETE.

In terms of frame format CoAP has a 4 Byte fixed header and optional fields in Type-Length-Value (TLV) format as seen in the following figure:

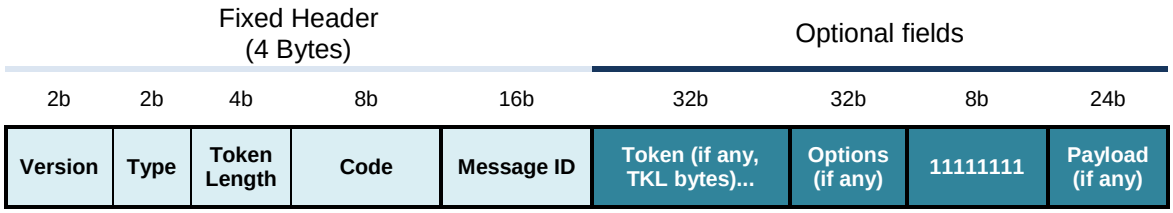

Fig. 2.15. CoAP frame format

To request an action (using a method code) a CoAP request is sent by client on a resource (identified by URI on server) and then, the Server sends the response equivalent to HTTP. There are studies that confirm a reduction of response time in 2-hop transactions over 33% and a reduction of overhead in 85% compared with HTTP (see [16]).

When the request is confirmable, is mandatory to receive an acknowledgment message. However if the message is non-confirmable the response will be a non-confirmable message, providing "connection-oriented" over an "connectionless oriented" (UDP). Is important to mention that if a message is applied into 6LoWPAN, the messages should fit into a single IEEE 802.15.4 frame to avoid fragmentation. An example of how CoAP works with request/response model and how uses the messages and methods are shown in the following figure:

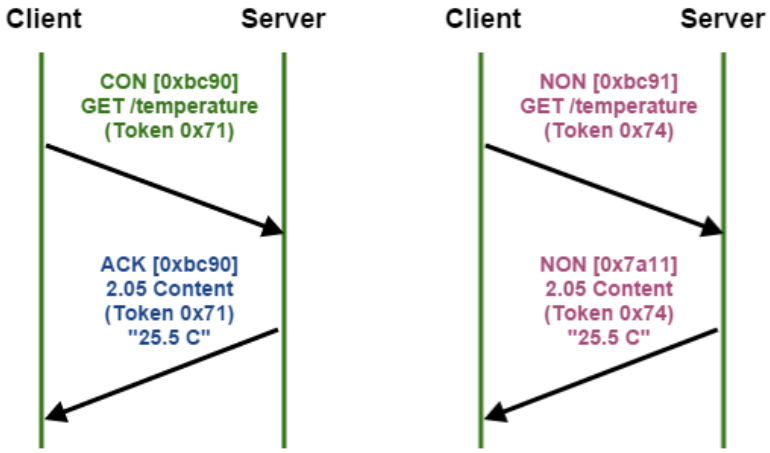

Fig. 2.16. CoAP request/response model. Confirmable (left) and non-confirmable (right)

With these capabilities CoAP can offer some advantages to deploy a IoT network. First one is in terms of discovery that makes able to recognize, add and remove new devices without human intervention. The second advantage is in terms of content negotiation, providing a data type to get a reference of data communication and to establish a REST oriented services (more compatible with existing tools). The mapping of CoAP with HTTP is also defined, allowing proxies to be built providing access to CoAP resources via HTTP in a uniform way. Finally, in terms of security is possible to bind with DTLS.

CHAPTER 3. AVAILABLE EQUIPMENT

Remembering that this document is focused in the coordinator of the WSN is required to select some equipment that supposes low-cost and low-power consumption. This chapter reflects the available equipment in the LAB to realize the project, this equipment is preferable to use and considered as "free" in the evaluation of the budget on chapter 4. The equipment to deploy PMT-WSN requires to accomplish the following directives:

- The equipment needs to establish wireless connection to create a WSN, specifically 802.14.5 with IPv6 protocol (6LoWPAN).
- The equipment requires Ethernet connection to be accessible from internet.
- The equipment must be low-cost to establish multiple coordinators that controls multiple WSN with a competitive price and must make possible the viability of the project.
- Is desirable that the equipment is as much low-power consumption as possible due similar reasons mentioned in the previous chapter. First, because is not required a very high performance equipment to collect messages from end-nodes, and second to be as more sustainable as possible providing low-consumption equipment saving costs in terms of electricity.

3.1. Raspberry Pi

Raspberry Pi (RPi) is a computer with credit-card size also called SBC that includes Ethernet, USB and HDMI interfaces that can provide high-definition video and enough capacity to run a Linux OS with GUI interface.

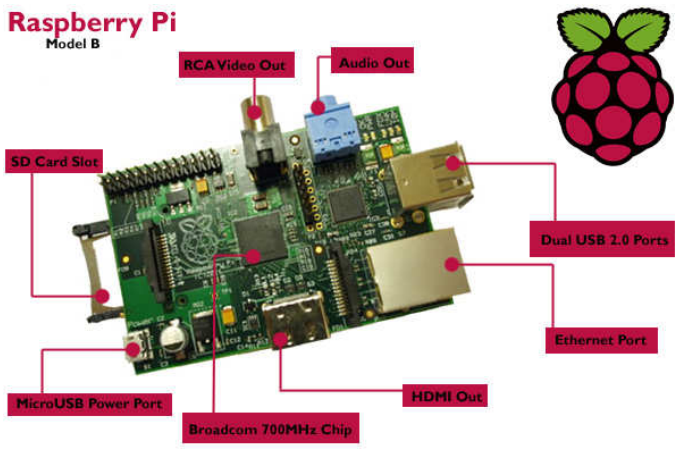

Fig. 3.1. Raspberry Pi model B (source: http://www.trustedreviews.com/)

Actually Raspberry Pi has 3 models (model A, model B and model B+) all based in Broadcom BCM2835 SoC that provides a CPU, GPU, DSP, SDRAM and a single USB port. The Main features of RPi are described on the following table:

Table 3.1. Raspberry Pi basic features

	Model A	Model B	Model B+
SoC	Broadcom BCM2835		
CPU	700 MHz ARM11 Family		
GPU	Broadcom VideoCore IV		
Memory (SDRAM)	256 MB Shared	512 MB Shared	
USB 2.0	1	2	4
Video outputs	Composite RCA and HDMI (rev. 1.3 and 1.4) PAL/NTSC		
Audio outputs	3.5mm jack and HDMI		
On board storage	SD / MMC		MicroSD
Network	None	10/100 Mbit/s Ethernet	
Peripherals	8 x GPIO		7 x GPIO
Power	300 mA (1.5W)	700mA (3.5W)	600mA (3W)

In terms of price RPi costs 39,95€ (see [17]), compared with a standard PC (cost around 300€) the reduction of the budget is 7,5 times. In terms of power consumption is possible to provide power supply up to 15 Raspberry Pi with a normal supply of standard PC[1].

Therefore RPi accomplishes some desired goals like cost, connectivity with Internet and low consumption compared with a standard PC doing the same function, the only thing that is required to be accomplished is a wireless connection.

3.2. Xbee module

XBee modules offer wireless connectivity providing point-to-point, point-multipoint and mesh architectures. XBee have two big families, XBee and XBee Pro, any family has different types of antenna with different transmission power.

[1] Standard PC is considered an AMD Athlon X2 4800+ with integrated VGA with average consumption of 55W in idle state and a Raspberry Pi supplied with 700mA (3,5W).

Fig. 3.2. XBee Pro Wire antenna

XBee modules are designed to be used with Zigbee or IEEE 802.14.5 without compatibility one to each other. For PMT-WSN case will be used 6LowPAN, for that reason is selected XBee 802.15.4, specifically XBee Pro, this XBee module has the following specs:

Table 3.2. XBee Pro Wire antenna specs

Power supply	3.3V @ 215 mA
Data rate	250kbps
Output	60mW (+18dBm)
Range	1500m
ADC pins	6x10-bit
Digital IO pins	8
Encryption	128-bit

In terms of transmission modes XBee offer basically two types of communication that are AT mode and API mode.

- **AT mode**: transparent serial transmission between nodes.
- **API mode**: this mode provides a framework to exchange normalized packets between XBee's,

XBee modules cost around 38$ (see on [25]) offering IEEE 802.15.4 communication, it will be important to find some 6LoWPAN solutions that make possible fit into an XBee module.

3.3. Arduino Duemilanove

Arduino is a microcontroller board that makes possible to build an infinity of DIY projects with a low-cost hardware. The most relevant specs are the following (see on detail on [24]):

Table 3.3 Arduino Duemilanove main features

Microcontroller	ATmega168
Operating Voltage	5V
Input Voltage	7-12V
Programming interface	USB
Number of Digital I/O pins	14 (6 of them are PWM output)
Number of Analog input pins	6
Flash memory	16 KB (2KB dedicated to bootloader)
SDRAM	1 KB
EEPROM	512 Bytes
Clock Speed	16 MHz

One of the most interesting features that Arduino has is the contribution of a huge community continuously evolving and constructing new projects and has constant evolution with new hardware or shields, one of the most interesting shields for the project developed XBee proto shield that makes possible to connect previously commented XBee antenna into an Arduino in a very easy way.

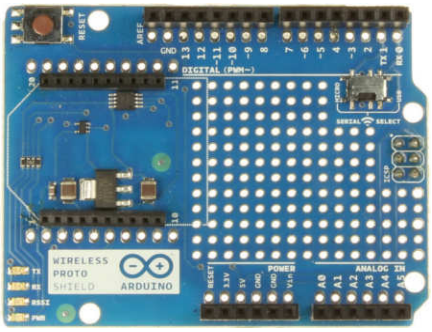

Fig. 3.3. Xbee shield for Arduino *(Source: arduino.cc)*

In terms of cost is also a very interesting option (Arduino MEGA 39 € (see [21]) and Arduino UNO 17 € (see [22] as example) but in terms of performance RPi has more capabilities and functionalities that make more fittable into the aim of the project but is a good idea to search a project to use Arduino as end node.

CHAPTER 4. TECHNOLOGY REVISION

Remembering that this document is focused in the coordinator of the IoT network this chapter details Operating Systems, built-in and open-source based solutions that are currently available on the market, describing the most relevant features of every solution and providing enough information to select one, taking into consideration some criteria according to PMT-WSN requirements.

4.1. Open-software to develop IoT

This subchapter are mentioned broadly some OS that are designed develop IoT networks. Of course, the goal of this document is not to provide a detailed information about how the different OS works, but with that information the reader will have a rough idea about different open OS that are currently available.

4.1.1. TinyOS

TinyOS was developed at the University of California in Berkeley and is maintained as an open-source project. The OS was designed for low-power wireless devices. Basically TinyOS are one of the first OS to use in embedded and limited resources systems. The hardware able to use TinyOS are listed on the annex A.I

4.1.2. Contiki

Contiki itself is an open source OS designed to work in IoT devices, that means that is focused to establish a low-power communication based on wireless, implementing IPv4 and IPv6 and low-power wireless standards such as 6LoWPAN, RPL and CoAP. There is a strong community developing features of the OS, applications and emulation software (COOJA) for deployment of a WSN. The hardware able to use Contiki are listed on the annex A.III.

4.2. Existing solutions

Looking in the market are several quantity of solutions that fits into a 6LoWPAN WSN. In the following subchapters are described existing solutions that offers a 6LoWPAN border router that usually are based on Linux, Contiki or TinyOS.

4.2.1. Grinch project

Grinch is a prototype to realize a border router based on Zigbit module. This module includes Atmega 1281 microcontroller and AT86RF230 radio transceiver. To realize this D.I.Y project are available schematics and information on the website (see [43]).

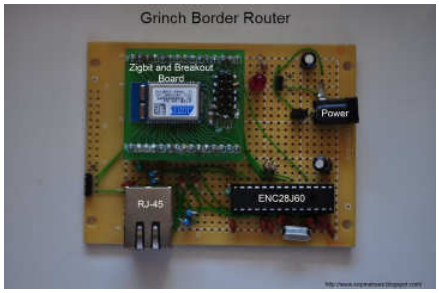

Fig. 4.1. Grinch border router project *(Source: www. http://sixpinetrees.blogspot.com.es)*

Grinch has a serious drawback that has only 8kb of RAM, limiting the number of neighbours and routes that can be maintained. Also, this limitation affects to a webserver performance and the impossibility of fragmentation of 6lowPAN packets. Grinch are one of the first test in terms of 6LoWPAN border router solution.

4.2.2. JenNet-IP Border-Router

JenNet-IP™ is an IP-based networking solution from NXP Semiconductors that uses 6LoWPAN network with a “mesh-under” approach. On the website (see [44]) the company emphasizes self-healing, highly robust and scalable network and maximum capacity up to 500 nodes.

Supported on the JN5148-J01 and JN5142-J01 wireless microcontrollers, the solution offers an ultra-low-power platform for product development. MIB API (JIP), provides OAM as a SMNP approach.

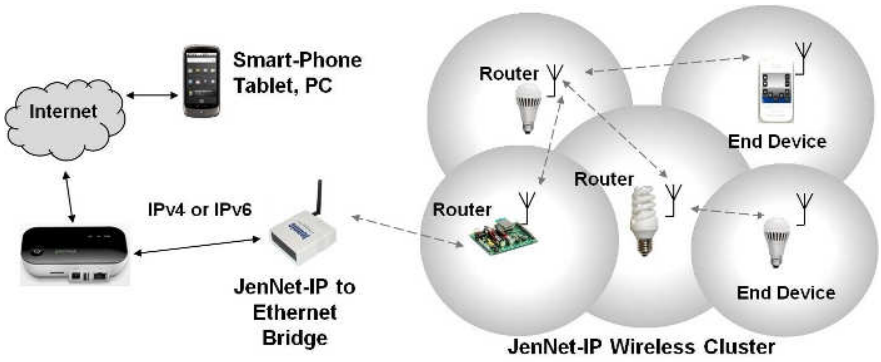

Fig. 4.2. JenNET-IP example network *(source: www.jennic.com/)*

The principal features of this built-in solution are presented in the next table:

Table 4.1. JenNet-IP features

Criteria	Description
Topologies	Tree, star, linear
Maximum network size	500 nodes
Standards	IP, UDP, 6LoWPAN and IEEE 802.15.4
3rd Party interoperability	Through public MIBs
Open source	No
Extra features	
Gateway/non-gateway option to provide translation IPv6 with 6LoWPAN.	
Security based on authentication and 128-bit AES encryption	

The architecture JenNet-IP border router includes apart from IEEE 802.15.4 Layers and intermediate layer (JenNet layer) between IEEE 802.15.4 and 6LowPAN Layer that provides "mesh-under" scheme. In upper layers from 6LowPAN includes IP,UDP and the JIP API to interact with MIB's and with the user application as seen in the following picture:

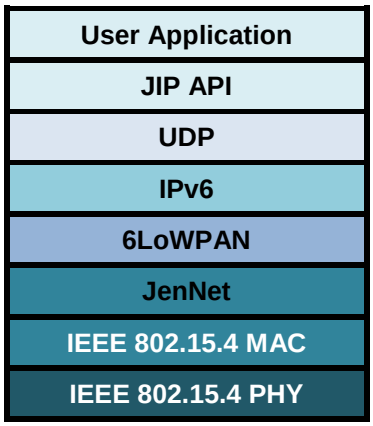

Fig. 4.3. JenNET-IP Stack

Unfortunately JenNetIP router is not "fittable" in terms of IP communication with third parties products based on typical 6LoWPAN stack, therefore is risky solution if the JenNet products discontinues and will not possible to build heterogeneous WSN in terms of devices.

4.2.3. NanoRouter 2.0

NanoRouter™ 2.0 is a border router solution from Finnish company Sensinode. This solution enables routing between IP-based 6LoWPAN low-power wireless

networks and backbone IPv4 / IPv6 networks. In addition, NanoRouter 2.0 acts as the coordinator for RPL routing within a 6LoWPAN network. The features of this built-in solution are presented in the following table:

Table 4.2. NanoRouter 2.0 features

Criteria	Description
Maximum network size	1000 nodes
Standards	IP, UDP, TCP, 6LoWPAN, RPL, ICMPv6, and IEEE 802.15.4
Open source	No
3rd Party interoperability	None
Extra features	
Web development.	
Security based on TLS and DLTS for authentification and data transport	

NanoRouter on one hand establishes a 6LoWPAN network and on the other hand are connected the services with IPv6, another key point is the existence of NanoStack™ and NanoService™.

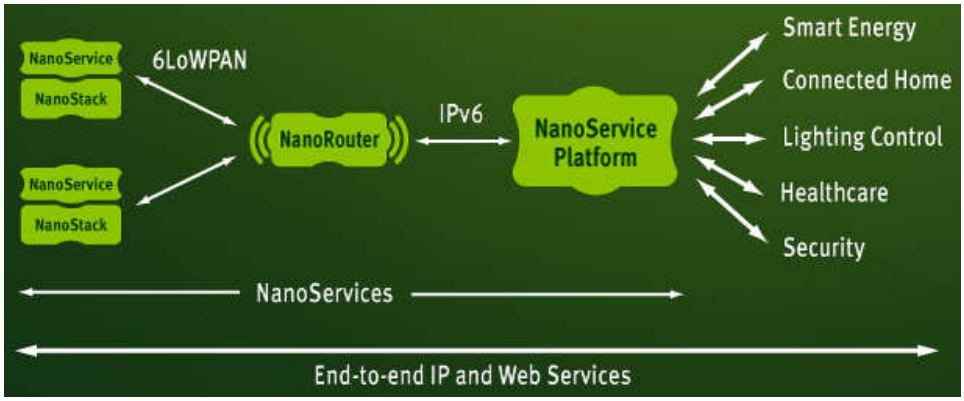

Fig. 4.4. Sensinode NanoRouter diagram *(source: mbed.org)*

NanoService Platform is a framework to develop REST applications using standards like CoAP and EXI, obtaining high performance in web applications for embedded systems. Also, NanoService has embedded device libraries to and templates for NanoService reference applications. On the other hand, NanoStack is a layer design to deploy embedded 6LoWPAN device level software.

Actually, Sensinode are acquired by ARM to include Sensinode connectivity into a embedded ARM processor into ARMbed products, that means that the continuity of this solution is not confirmable.

4.2.4. Cisco 500 Series WPAN Industrial Router (IR500)

Cisco have their industrial solution to realize the 6loWPAN border router that is IR500 Router. This equipment is a built-in solution optimum for companies that wants a ready to use equipment with the guarantee of a big network company.
.

Fig. 4.5. Cisco IR500 industrial router *(source: www.cisco.com)*

Cisco provides a very detailed product datasheet (see [49]), the principal features are reflected in the following table:

Table 4.3. Cisco IR500 features

Criteria	Description
Standards	IP, UDP, TCP, 6LoWPAN(ND,HC), RPL, ICMPv6, BGP, NAT 44, MAP-T, RAW sockets, DHCPv6, CoAP, and IEEE 802.15.4
QoS	Differentiated Services Code Point (DSCP) and priority queuing
3rd Party interoperability	Yes
Open source	No
Extra features	
IPv4 over IPv6 support	
Interference reduction with FHSS+Spatial frequency reuse	
Security based on o Access and auth based on X.509 + PKI o Data AES-128 bit encryption o Integrity based on certification-based (signatures)	

4.2.5. Arduino IPv6 Stack

There is an Arduino project that offers a solution closely related to a border router function, this project is called Arduino µIPv6 Stack (see [19]) and pIPv6 Stack (see [20]). Both projects are promoted by Telecom Bretagne university and is completely open source and compatible with Contiki OS network stack, this project is available on github repository (see [19] and [20]).

The differences between µIPv6 Stack and pIPv6 Stack are the limitations of the utilized hardware. pIPv6 Stack is the solution for Arduino's UNO or similar models (that only has 2kB of RAM) and the CoAP and RPL protocols are reduced to work with very small memory. By other hand, µIPv6 stack is the solution for Arduino MEGA model and it uses the full functionalities of Contiki OS network stack.

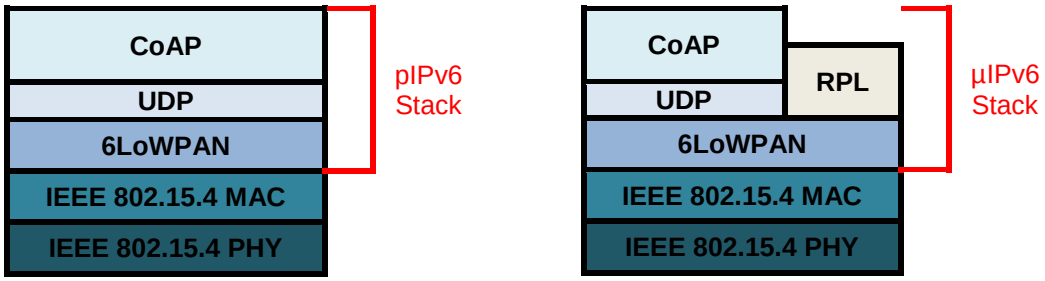

Fig. 4.6. pIPv6 stack (left) and uIPv6 stack (right)

One interesting capability is that both stacks are perfectly compatible, where is possible to establish both stacks in a IoT network. To establish a border router is preferable to be an Arduino MEGA with µIPv6 stack to have fully complete RPL and CoAP capabilities.

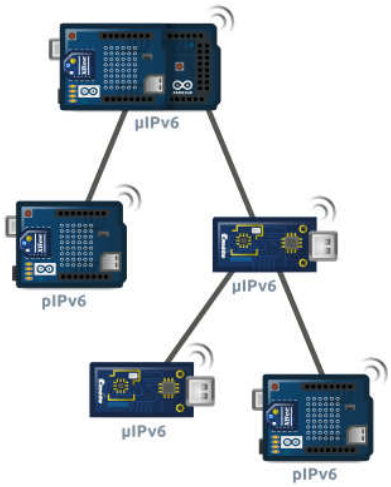

Fig. 4.7. Arduino heterogeneous 6LoWPAN network *(source: Arduino-IPv6Stack wiki)*

Another benefit using Arduino IPv6 stack are software features that makes able to modify the actuation as end node, intermediate router or a border router. In terms of cost is an interesting solution due the attractive price of Arduino hardware. Finally is good to consider a product that has very active community that are creating constant evolving solutions.

Thinking in the PMT-WSN network, is preferable to deploy in a RPi due to have higher performance and flexibility. A possibility is to implement an special shield that makes possible load Arduino UNO projects on RPi (see [23]) maintaining

the OS and RPi configuration. With this shield will be possible to implement an pIPv6 stack with a reduced (or NULL) RPL and CoAP losing full functionalities.

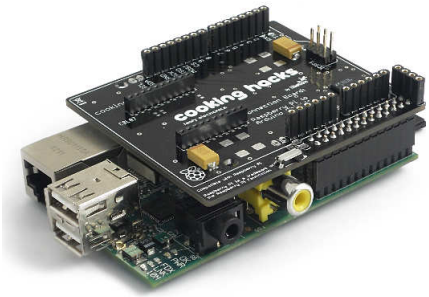

Fig. 4.8. Arduino shield for Raspberry Pi *(source: www.cooking-hacks.com)*

Table 4.4. IPv6 Stack features

Criteria	Description
Maximum network size	Unknown
Standards	UDP, 6LoWPAN, RPL, CoAP and IEEE 802.15.4
3rd Party interoperability	Yes
Open source	Yes

4.2.6. Redwire BR12

Redwire BR12 (see [27]) is a built-in solution to establish a full network based on 6LoWPAN with RPL and IPv6 connectivity including IPv4/NAT sites and the required software (BRamble) for the administration on a Linux-based router.

Fig. 4.9. Redwire BR12 *(source: www.redwirellc.com)*

Table 4.5. Redwire BR12 features

Criteria	Description
Standards	IP, UDP, 6LoWPAN, RPL, NAT and IEEE 802.15.4
3rd Party interoperability	Yes
Open source	No

4.2.7. Redwire RedIO

Redwire IO is an embedded router to establish a 6LoWPAN network based on Contiki OS router firmware (mist or thingsquare 6LBR), this feature is very interesting to interconnect devices that have Contiki OS network stack.

Fig. 4.10. Redwire IO *(source: www.redwirellc.com)*

The features that Redwire facilitates are explained in the following table:

Table 4.6. Redwire IO features

Criteria	Description
Standards	IP, UDP, 6LoWPAN, RPL, NAT64, DNS64, DHCP64 and IEEE 802.15.4
3rd Party interoperability	yes
Open source	yes

4.2.8. 6LoWPAN Border Router (6LBR)

6LBR is a project from CETIC (see [26]) and evolves the Contiki 6LoWPAN solution. 6LBR provide stand-alone router providing IPv6/NDP mechanisms and RPL. As the other options, 6LBR uses layer 2 (Ethernet and 802.15.4), layer 3 (IPv6/6LowPAN) and "layer 4" (ICMPv6/RPL) to switch and route packets. To begin to use 6LBR is possible to use a Nooliberry board (normal and T) with a raspberry or Linux equipment with USB-Mote compatible with Contiki (hardware compatibility in Annex A.III).

Table 4.7. Redwire 6LBR features

Criteria	Description
Standards	IP, UDP, 6LoWPAN, RPL and IEEE 802.15.4
3rd Party interoperability	Yes
Open source	Yes

6LBR can operate in three different modes: bridge (smartBridge), router (Router, NDP-Router, 6LR and RPL-Root) and transparent bridge (RPL-Replay and Full transparent bridge).

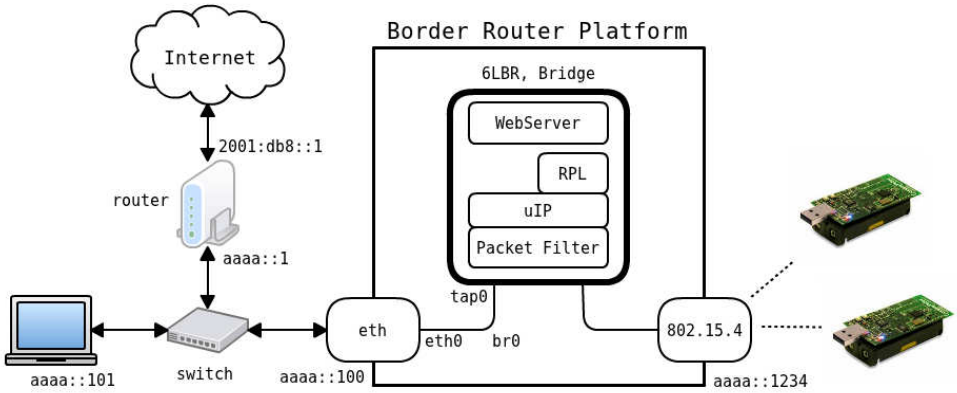

Fig. 4.11. Bridge mode *(source: 6LBR wiki)*

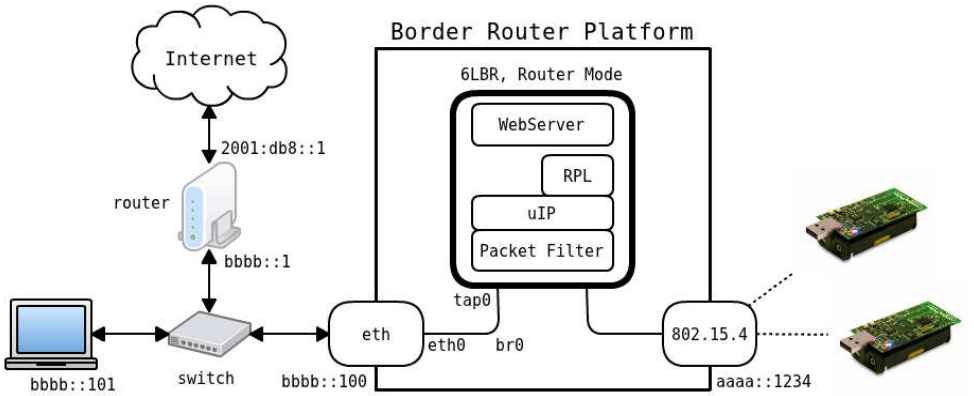

Fig. 4.12. Router Mode *(source: 6LBR wiki)*

4.2.8.1. SmartBridge Mode

In this mode, the 6LBR interconnects a standard IPv6 based network with a RPL based WSN mesh. The SmartBridge is acting as a NDP proxy on the Ethernet side and is using NDP parameters to configure the WSN mesh. Source and destination MAC addresses are translated and addresses present in ICMPv6 packets are also translated. The SmartBridge mode allows you to :

- Integrate a WSN mesh into an existing NDP based IPv6 network.
- SmartBridge will use the NDP provided configuration to set up the WSN.
- Aggregate several WSN meshes with their own DODAG into one virtual IPv6 subnet.
- Supports node mobility and handoff due to the NDP proxy and IPv6 subnet.

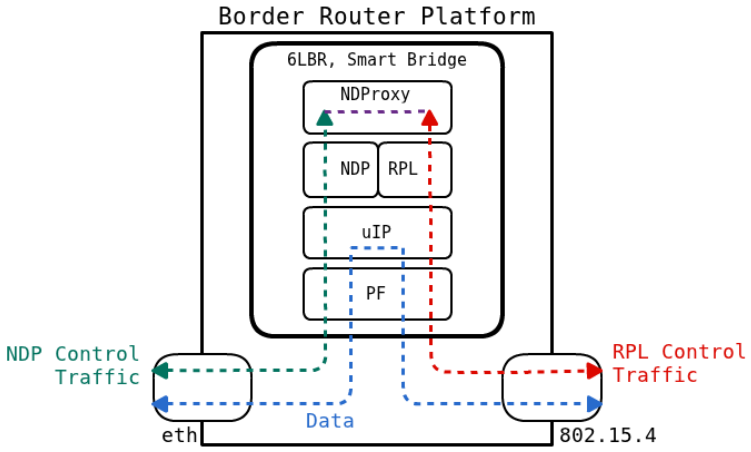

Fig. 4.13. Smartbridge Mode *(source: 6LBR wiki)*

4.2.8.2. Router Mode

In this mode, the 6LBR acts as a IPv6 Router, interconnecting two IPv6 subnets. The WSN subnet is managed by the RPL protocol and the Ethernet subnet is managed by IPv6 NDP. In this mode, the 6LBR provides a virtual second interface to Contiki thanks to the packet filter module. The Router mode allows you to isolate WSN from other networks and provide mobility with prefix switching capability.

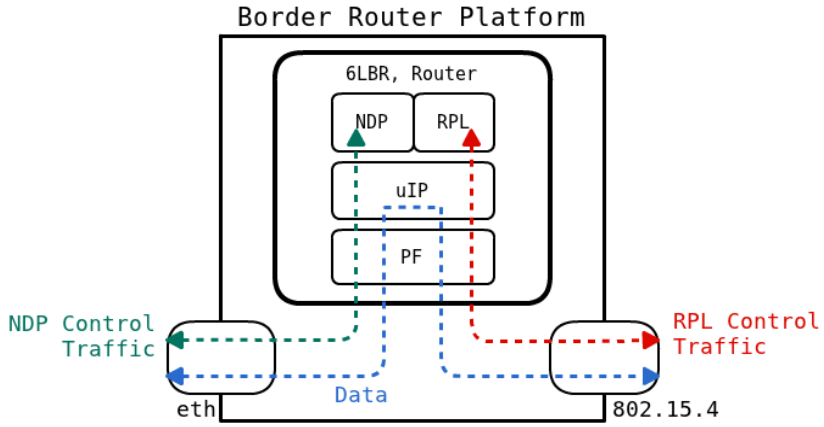

Fig. 4.14. Router mode *(source: 6LBR wiki)*

4.2.8.3. TransparentBridge Mode

In this mode, 6LBR acts as a standalone bridge, providing basic switching capabilities. All incoming packets targeting a 802.15.4 interface or incoming multicast packets on the Ethernet interface are forwarded to the WSN segment. By other hand, all incoming packets targeting an Ethernet interface or incoming multicast packets on the 802.15.4 interface are forwarded to the Ethernet segment. The 6LBR has its own address and act as a host. Source and

destination MAC addresses are translated and addresses present in ICMPv6 packets are also translated. The transparent Bridge Mode allows you to :

- Aggregate sub-WSN meshes into one global DODAG, managed by an external RPL Root node.
- Bridge a one-hop mesh with an IPv6 network using NDP (When using FullTransparentBridge).
- Bridge a statically routed mesh with an IPv6 network (When using FullTransparentBridge)

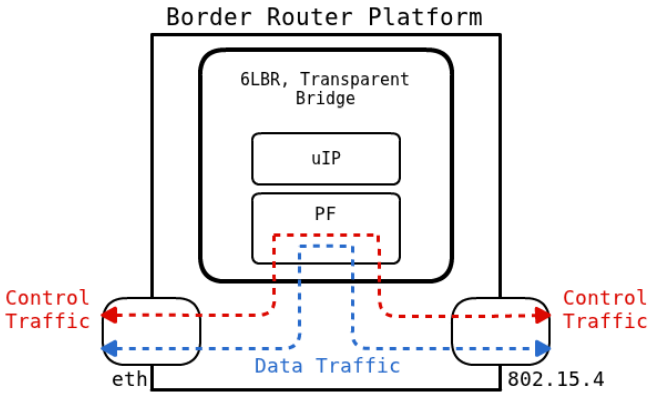

Fig. 4.15. Transparent Bridge *(source: 6LBR wiki)*

Concluding, 6LBR is a very interesting solution to develop an IoT network providing most updated technologies in an open source world, offering a considerable variety of operation modes and a powerful web administration to get a starting point to develop web services and OAM applications.

4.2.9. BLIP 2.0

BLIP stack, is an implementation in TinyOS to develop an IoT network from Berkeley University (California). Is possible to load previous mentioned tinyOS in certain motes to obtain a border router solution. The BLIP stack are:

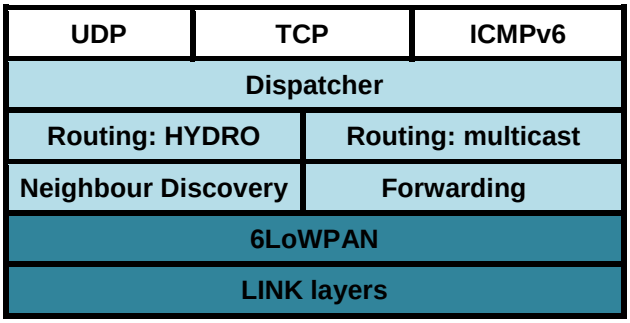

Fig. 4.16. BLIP 2.0 Stack

As seen in the previous figure BLIP 2.0 includes HYDRO protocol that seems a primitive of RPL designed by Berkley university, but is possible to use TinyRPL and CoAP to make it compatible with OpenWSN stack. In terms of community it seems that Contiki community are one step beyond becoming a reference in IoT development. By the way is a open-source TinyOS solution that is possible to install in a RPi, for that reason is a solution that will be included to decide which solution will be deployed,

4.2.10. Openlabs RPi 802.15.4 Radio

Openlabs offers an antenna add-on module to turn a RPi (or whatever that has SPI port and 4 GPIOs) into a 6LoWPAN border router solution. Unfortunately 6LoWPAN and 802.15.4 support on Linux are on the edge of developing so is recommended to implement net-net Linux kernel (see [30]) to get 6LoWPAN functionalities. Net-next kernel is a branch maintained by David Miller and after by Alexander Aring which holds a variety of network-related changesets and, of course, including 802.15.4 and 6LoWPAN (not RPL and CoAP).

The module consists in a chip antenna, crystal oscillator and Atmel AT86RF233 that includes the transceiver to get an 802.15.4 solution. The module plugs directly onto pins 15-26 of the RPi's GPIO, leaving the remaining pins free for other uses.

Fig. 4.17. RPi with Attached Openlabs module *(source: openlabs.co)*

With that solution are provided the basis to deploy a 6LoWPAN network and develop upper protocols and web applications but is important to mention that is a solution that there is not provided any performance test or application based on this hardware.

4.3. Comparative 6LoWPAN solutions

Considering all previous mentioned solutions is required to take a decision to select the most interesting using certain criteria. Therefore is defined five terms to establish a decision to develop a 6LoWPAN-based solution. First, is interesting that the solution will be open-source maintaining costs down and to

find some solution that are constantly evolving with continuous corrections and enhancements. Usually these solutions are open-source with effervescent community in the background. The second term is that the solution is compatible with the stack defined in OpenWSN (the same as defined Contiki), this stack now is the most extended into the open-source community to build WSN and to offer interconnectivity into a heterogeneous network, composed by different kinds of experimental open-source motes. With that decision is increased the possibility of communications between different hardware. Third term to mention is the compatibility with RPi, XBee or Arduino Duemilanove that are "trusted" equipment to begin to develop a 6LoWPAN solution maintaining a low-cost (provided by the department) solution and ensuring a medium term availability of the product. To conclude is important to have a cost orientation of the solution to decide which has more viability to establish.

NanoRouter, at the day of today, is not available due to the purchase of Sensinode from ARM enterprise making the product unavailable. By other hand, Grinch is a non-mature solution in a DIY product that can offer problems in a medium-size WSN, requires manufacturing the product and the result is a solution with short resources, resulting a non-recommended option for a medium-large size WSN to cover PMT-EETAC.

Once are discarded some projects according to very critical points it's time to realize a chart where all the solutions are presented, there are solutions marked in red that are previously rejected solutions and marked in green the most interesting solutions. After that will be discussed the rejected and selected options providing arguments the decision.

Is important to mention the budget limitation for the project. Due to this limitation and due to not be an open source solution (also have a different stack that makes the interconnection between elements harder) JenNet -IP becomes a non-recommendable solution to implement. Aligned with that, Cisco IR500 is a built-in solution by a company with high trustability and presence in the market offering a product with capabilities to satisfy huge quantity of nodes and of course these features will impact on the cost of the product.

Continuing with BLIP solution it has possible aspects like open-source, compatibility (at least layer 3) with openWSN stack and the possibility to install TinyOS on a RPi but it has some drawback like the supported hardware (see Annex A.I.) that makes unable to use Xbee. To propose a reference a Zolertia Z1 is selected with a cost 299 €, making a high cost compared with other solutions. Another aspect to consider is that Contiki seems that are one step forward compared with TinyOS taking a look the community, revisions and updates, that makes Contiki OS solutions more attractive.

Going into more detail on Redwire products provide attractive built-in solutions with competitive price. By the way, accomplishes some features as open-source, compatibility with openWSN stack and reduced cost but not accomplish hardware compatibility with RPi and XBee due to be a built-in solution. Between RedIO and BR12 products is clearly preferred the first one due the possibility to

load different Contiki projects (such as mentioned 6LBR), providing capabilities and enhancements with less cost compared with BR12.

Now is discussed about Arduino solutions. Both are very good solutions that includes open-source technology and low budget. Thinking in terms of deploying a border router solution will be recommended to establish Arduino µIPv6 stack to not depend for a low memory limitations, in terms of compatibility with available hardware are accomplished XBee compatibility but not RPi compatibility (is required an Arduino MEGA), avoiding that, Arduino solutions are one of the most interesting solutions to take it into account in the deployment phase. Of course, thinking in terms of nodes is a wonderful solution to use Arduino UNO (or even better, Arduino NANO (see [31])) with an Arduino pIPv6 stack to built an end-node solution.

Meanwhile 6LBR is a very powerful solution to realize a border router and is a project focused explicitly to deploy it. For that reason, is one of the most interesting solutions to realize the IoT network objective for this project, using a RPi in his basis, with a big list of features to tune the border router according to the requirements of the network and the included webserver providing a starting point of OAM. In terms of negative aspects, there is one drawback, the dependence of the company that realizes the Nooliberry T antenna (Noolitic), the cheapest one of the supported modules, and the short stock that the company has.

Other interesting solution is the Openlabs 802.15.4 module that offers a 6LoWPAN solution for RPi. There is not a full-deployed solution, requiring to implement upper layers support such as RPL, CoAP and router functionalities, but it has an opportunity to have a 6LoWPAN module with less than 10 €. Of course, it has same problem than 6LBR that is a huge dependence of the module (more dependence in this case) but the cost of each makes able to order higher quantity of modules to prevent failures or more border routers.

In case that will be possible to increase a little bit the budget and replacing RPi by Arduino MEGA it will be very interesting to deploy Arduino µIPv6 stack using a well-known antenna as XBee in an open source hardware and software projects.

To conclude, ordering the preferences to deploy the solution, from best to worst, the first are 6LBR solution, the second Openlabs module, Arduino µIPv6 stack and RedIO are the best ones according our budget.

In the following table is designed a table to get a faster comparison between the reviewed solutions, are marked in green colour the more interesting and in marked in red colour the discarded ones according to the requirements and arguments explained before according the criteria.

Table 4.8. Comparative of solutions

Name	Open-source	OpenWSN Stack compatible	RPi Compatible	Xbee Compatible	Extra cost
Grinch	Yes	Yes	No	No	Zigbit module (see [32]) = 29.00 €
JenNet-IP	No	No	No	No	Evaluation kit (see [18]) = 1,082.97 €
NanoRouter 2.0	No	No	No	No	Not available
Cisco IR500	No	Yes	No	No	Not available
Arduino µIPv6 stack	Yes	Yes	No	Yes	Arduino MEGA + Xbee shield + Xbee (provided) 39.00 + 39.00 = 78.00 €
Arduino pIPv6 stack	Yes	Yes	Yes	Yes	RPi (provided) + Arduino shield + Xbee (provided) 40.00 €
Redwire BR12	Yes	Yes	No	No	Starter kit (see [28]) = 449 $ ≈ 360.00 € Router (see[29]) 179.99 $ ≈ 141.00 €
Redwire RedIO	Yes	Yes	No	No	199.99 $ ≈ 94.00 €
6LBR	Yes	Yes	Yes	No	RPi (provided) + Nooliberry T 35.00 €
BLIP 2.0	Yes	Yes (layer 3)	Yes	No	Z1 Discovery pack (see [33]) = 299.00 €
Openlabs module	Yes	Yes (layer 3)	Yes	No	RPi (provided) + 802.15.4 module 8.00 €

CHAPTER 5. TECHNOLOGY DEVELOPMENT

This chapter is focused on explaining in detail the development process of the different chosen options and the pros and cons in terms of the developing process. This chapter describes how to install the different selected solutions. To have a basic RPi configuration, Raspbian is installed with some basic features (explained on Annex A.V).

5.1. 6LBR Development

The starting point to Develop 6LBR solution is to follow the wiki that has a lot of documentation about 6LBR project (see [36]). Previously, is recommended to install some libraries and utilities to ensure that 6LBR will work well. To install them type:

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

To communicate with Nooliberry T antenna is required to edit and remove the following parameters on */boot/cmdline.txt :*

```
console=ttyAMA0,115200 kgdboc=ttyAMA0,115200
```

Also in */etc/inittab* is required to comment the following line due the utilization of ttyAMA0 from the Nooliberry T antenna:

```
#Spawn a getty on Raspberry Pi serial line
#T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100
```

In the case of RPi, is prepared a package and apparently is easy as download, install and play. The commands to realize these actions are using previous mentioned *wget* and Debian package installer *dpkg*:

```
wget https://raw.github.com/wiki/cetic/6lbr/releases/cetic-
6lbr_1.3.2_armhf.deb
dpkg -i cetic-6lbr_1.3.2_armhf.deb
```

Now 6LBR is already installed, next step is to configure 6LBR application, to do that 6LBR uses the file *6lbr.conf* placed at */etc/6lbr.* For this project the configuration file is the following:

```
MODE=ROUTER
CREATE_BRIDGE=0
RAW_ETH=0
BRIDGE=1
DEV_BRIDGE=br0
DEV_TAP=tap0
DEV_ETH=eth0
RAW_ETH_FCS=0

DEV_RADIO=/dev/ttyAMA0
BAUDRATE=115200
LIB_6LBR=/usr/lib/6lbr

LOG_LEVEL=3 #INFO and above only
```

By other hand, the configuration of the interfaces (*/etc/network/interfaces*) is the following:

```
auto lo

iface lo inet loopback
iface eth0 inet static
address 0.0.0.0
auto br0
iface br0 inet static
    address 192.168.137.15
    netmask 255.255.255.0
    gateway 192.168.137.1

    pre-up brctl addbr br0
    pre-up brctl addif br0 eth0
    pre-up ip addr flush dev eth0

    post-down ip link set eth0 down
    post-down ip link set br0 down
    post-down brctl delif br0 eth0
    post-down brctl delbr br0

allow-hotplug wlan0
iface wlan0 inet manual
wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf
iface default inet dhcp
```

It is recommended to ensure that the OS accepts RA messages and establishes routes. To do that is required to type the following:

```
sudo route -A inet6 add aaaa::/64 gw bbbb::100
sudo sysctl -w net.ipv6.conf.all.accept_ra=1
```

Once all this steps are already done, is recommended to restart RPi and configure an external PC IPv6 Ethernet interface with an IP with bbbb prefix, in that case is configured with IP bbbb::101 to access to 6LBR web interface in bbbb::100. Typing this address in a web browser it appears the following page:

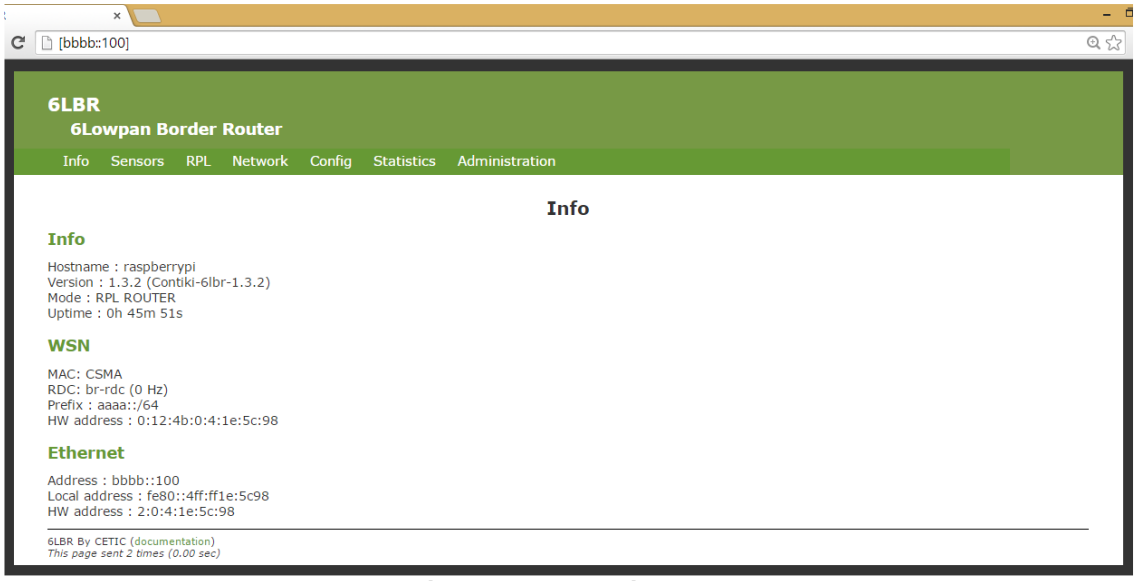

Fig. 5.1. 6LBR webserver

In the website is possible to see the internal state of 6LBR application, configure all the detailed parameters that contain the application, see the network state, networks, routes and see log and error files in the different labels of the website.

5.2. Openlabs 802.15.4 module Development

First of all is recommended to install lowpan-tools, this application will be useful once all be already done. To install these tools is required to add a new address to the repository, update and install the package, typing the following:

```
$ echo deb http://jenkins.ibr.cs.tu-bs.de/download/debian/ wheezy-unstable
main | sudo tee /etc/apt/sources.list.d/ibrdtn.list

$ sudo apt-get update

$ sudo apt-get install lowpan-tools
```

Also for future synchronizations is recommended to apply root password, in our case the same as default user in RPi ("raspberry").

```
$ sudo passwd root
```

Now is time to compile a modified kernel, the process time to compile new kernel in a RPi is around 12h (see [13]). For that reason is high recommended to have a Linux machine that speeds up compiling process with cross compiling technique. The selected machine is a PC running with Ubuntu 13.10, be careful, the following commands is to realize in a PC (not in RPi!). At first, is shown how to install cross compiler and prefix the commands to work with the explicit compiler for RPi in the Linux machine with these commands:

```
$ sudo apt-get install gcc-arm-linux-gnueabi
$ export CROSS_COMPILE=arm-linux-gnueabi- ARCH=arm INSTALL_MOD_PATH=.mods
```

Once cross compiler is installed the net-next kernel from github is downloaded and is accessed through the folder net-next with the commands:

```
$ git clone https://github.com/linux-wpan/linux-wpan-next.git

$cd linux-wpan-next
```

Now is time to configure the kernel. As starting reference the default configuration of bcm2835 is used, later the modifications will be realized.

```
$ CROSS_COMPILE=arm-linux-gnueabi- ARCH=arm make bcm2835_defconfig
```

Once is realized the default configuration, it will configure the kernel with the command:

```
$ ARCH=arm make menuconfig
```

Important to mention that will appear an X Window with all the available configurations. In this case is saved without any change, the configuration is saved on *.config* file and after it will edit the desired options searching the parameter to set, erasing the "#" prefix and the "is not set" string, replacing by

"=y" at the end. Now are described the options that will be essential, maybe some of them do not appear in *.config* file but are required to take into account in the next step:

```
CONFIG_6LOWPAN
CONFIG_IEEE802154
CONFIG_IEEE802154_6LOWPAN
CONFIG_MAC802154
CONFIG_IEEE802154_DRIVERS
CONFIG_IEEE802154_AT86RF230
CONFIG_MODULES
CONFIG_MODULE_UNLOAD
CONFIG_FUSE_FS
CONFIG_IP_NF_IPTABLES
CONFIG_IP_NF_FILTER
CONFIG_IP_NF_TARGET_REJECT
CONFIG_IP_NF_MANGLE
CONFIG_IP6_NF_IPTABLES
CONFIG_IP6_NF_FILTER
CONFIG_IP6_NF_TARGET_REJECT
CONFIG_IP6_NF_MANGLE
CONFIG_USB_DWC2_HOST
CONFIG_USB_DWC2_PLATFORM
```

Now, is time to make the modified kernel. It is important to take care about the options that are not set, they will be questioned to the user to be installed or not, it is when the other options that not appears in *.config* file will appear and need to be installed, the other options just press enter to follow "by default" option. The command to make the kernel is the following:

```
$ CROSS_COMPILE=arm-linux-gnueabi- ARCH=arm make
```

It takes several time this process, but once it's done and complete, it will make the modules and install them:

```
$ CROSS_COMPILE=arm-linux-gnueabi- ARCH=arm make modules

$ CROSS_COMPILE=arm-linux-gnueabi- ARCH=arm INSTALL_MOD_PATH=.mods make
modules_install
```

With that are obtained two files required to boot the kernel, first the kernel itself that is in *net-next/arch/arm/boot/zImage* and the device tree, that is the mechanism that ARM systems use to describe hardware at boot time, to provide something similar is like the BIOS in a PC, that device tree is placed in *net-next/arch/arm/boot/dtb/bcm2835-rpi-b.dtb* . Now is required to tell to the kernel how to access and which driver use to the 802.15.4 module. To do that is edited the file (in this project is used "nano" editor that is included on Raspbian) placed on *net-next/arch/arm/boot/dts/bcm2835-rpi-b.dts* adding the following code at the bottom of the file:

```
&spi {
	status = "okay";
	at86rf233@0 {
		compatible = "atmel,at86rf233";
		reg = <0>;
		interrupts = <23 1>;
		interrupt-parent = <&gpio>;
		reset-gpio = <&gpio 24 1>;
		sleep-tpio = <&gpio 25 1>;
		spi-max-frequency = <500000>;
	};
};
```

Now is requested to rebuild the *.dtb* files typing the command:

```
$ ARCH=arm CROSS_COMPILE=arch-arm-gnueabi- make dtbs
```

bcm2835-rpi-b.dtb contains the patched device tree and it is able to interact with 802.15.4 module. The next step is to configure U-boot application to be able to boot the modified kernel into RPi. First step is to download github project of U-boot and place into RPi branch and finally make specific configuration for RPi model B:

```
$ git clone https://github.com/swarren/u-boot.git
$ cd u-boot
$ git checkout rpi_dev
$ make rpi_b_config
```

To conclude the installation of U-boot the installation in U-boot folder will be made as follows:

```
$ CROSS_COMPILE=arm-linux-gnueabi- make
```

Once is finished the mentioned command *u-boot.bin* file is created, this file redirects the loading process to the modified kernel but needs some file to know how boot the new kernel (as */boot/cmdline.txt* in standard kernel) called *uEnv.txt* and this case needs to create in */boot/* directory and include the following lines:

```
bootargs=earlyprintk console=ttyAMA0,115200 kgdboc=ttyAMA0,115200
root=/dev/mmcblk0p2 rootwait dwc_otg.lpm_enable=0

bootcmd=load mmc 0:1 ${kernel_addr_r} zImage; load mmc 0:1
${fdt_addr_r} zImage.dtb; bootz ${kernel_addr_r} - ${fdt_addr_r}
```

At this moment all is compiled and requires to copy the kernel, device tree, modules and u-boot into a RPi in this case with the following commands:

```
$ scp linux-wpan-next/arch/arm/boot/zImage root@192.168.137.28:/boot/zImage
$ scp linux-wpan-next/arch/arm/boot/dts/bcm2835-rpi-b.dtb
root@192.168.137.28:/boot/zImage.dtb

$ rsync -rlhic linux-wpan-next/.mods/lib/modules/
root@192.168.137.28:/lib/modules/

$ scp u-boot/u-boot.bin root@alarmpi:/boot/kernel.img
```

When it's already copied is time to boot RPi. If there are some problems during boot is recommended to check that in *uEnv.txt* has the same root argument than *cmdline.txt,* if are not equal is required to modify it.

Once the boot process it's working well is good idea to verify that the 802.15.4 module is recognized by the kernel. Using the previously installed lowpan-tools is possible to realize verification with the command:

```
$ iz listphy
```

The output will show the attached module such as:

```
wpan-phy0  IEEE 802.15.4 PHY object
    page: 0  channel: n/a
    channels on page 0: 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
```

Once is detected the 802.15.4 module is recommended to write a bash script to realize it every time that is required to specify parameters of the interface. In that case the content of *cnfigIface.sh* will be:

```
#!/bin/bash

iz add wpan-phy0 wpan0 0a:0b:08:00:00:00:00:01
iz set wpan0 0xabcd 0xffff 14
ip link add link wpan0 name lowpan0 type lowpan
ip link set wpan0 up
ip link set lowpan0 up
```

Now typing ifconfig in the terminal it appears the wpan0 and lowpan0 interfaces and it is ready to be used:

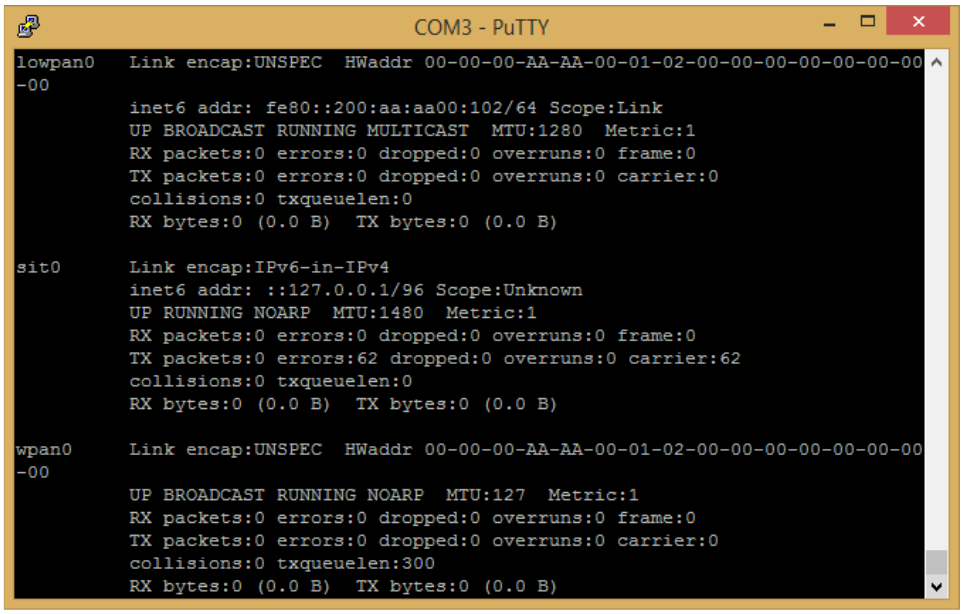

Fig. 5.2. OpenLabs module interfaces

5.3. Testing scenarios

To test 6LBR and Openlabs solutions are realized some connectivity tests to see the behaviour of the solutions with certain conditions. 6LBR is the preferred solution to establish a border router, for that reason first test is to try the connectivity of 6LBR with other solutions. After these tests it will be interesting to go forward and develop a next-step capability to establish Openlabs solution as a server for WSN network.

5.3.1. 6LBR scenarios

6LBR provides huge possibilities in terms of operation modes and protocols to use as seen in technology revision chapter. The basic 6LBR scenarios are represented in the following figure using Openlabs module and Arduino pIPv6 stack as end nodes:

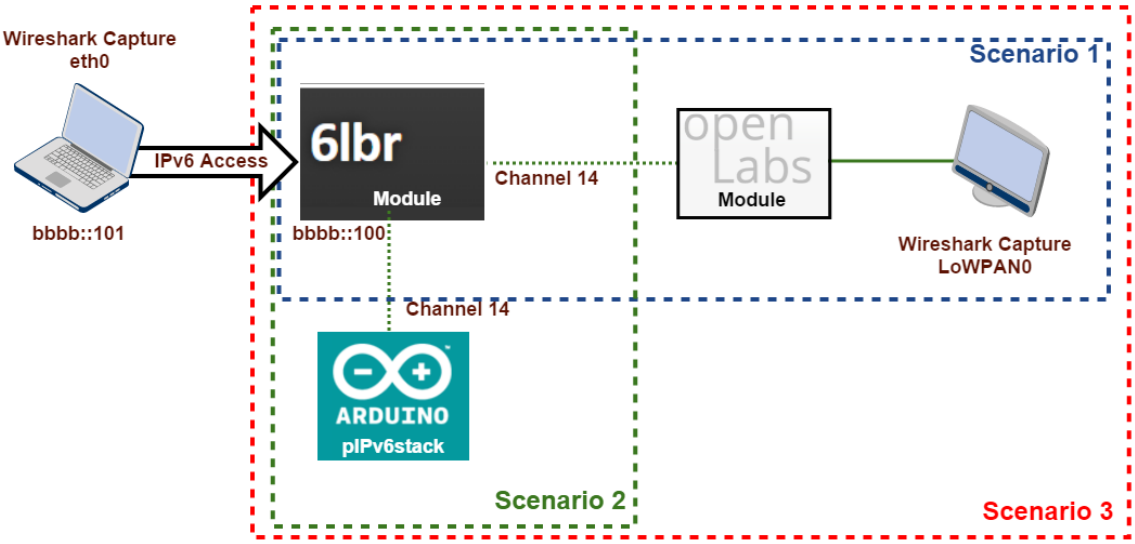

Fig. 5.3. OpenLabs module interfaces

In scenario 1, is well-known that Openlabs module is not provided RPL protocol, for that reason in */etc/6lbr/6lbr.conf* is required to substitute *MODE=ROUTER* by *MODE=NDP-ROUTER* to use 6LoWPAN-ND to get reachability between 6LoWPAN and Openlabs, also is modified in the webserver config label the following:

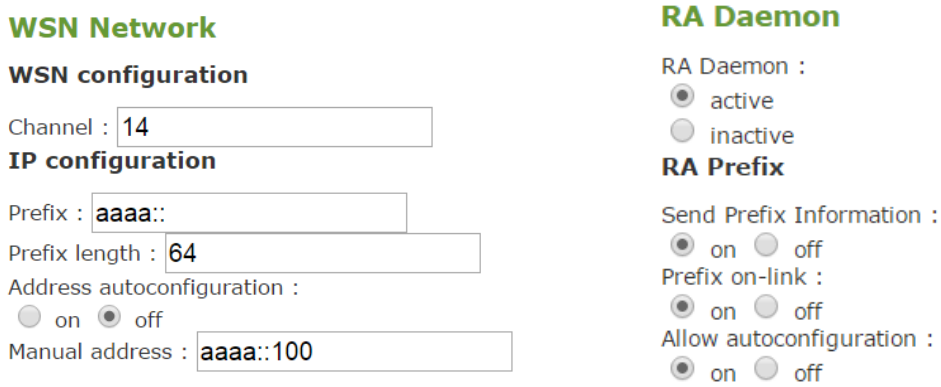

Fig. 5.4. 6LBR WSN configuration

With that configuration, there is no automatic address assignment, typing *ifconfig* just appears the link IPv6 address. Searching some solutions is possible to use RADVD, a router advertisement daemon that provides the prefix to the WSN. Applying RADVD on an external router (for installation of RADVD see Annex A.VIII), the address assignment works well; typing *ifconfig* command appears global and link address.

```
          inet6 addr: aaaa::80b:800:0:1/64 Scope:Global
          inet6 addr: fe80::80b:800:0:1/64 Scope:Link
```

One of the surprises that has been found is that, even so, 6LBR doesn't connect with Openlabs device. That situation could be because this Linux 6LoWPAN stack is not deployed Neighbour discovery or at least the

communication of them is not working properly inside the WSN. Doing "ping" connectivity tests the results are the following:

- From external host is reachable bbbb::100 and aaaa::100 addresses.
- From 6LBR is unreachable bbbb:100, aaaa::80b:800:0:1 addresses and reachable aaaa::100 address.
- From Openlabs module is unreachable aaaa::100 address.
- When Openlabs module is connected, appears as a neighbour with a STALE state but when is realized ping from external host to aaaa::80b:800:0:1 the neighbour appears as REACHABLE state, even that is an unreachable address in terms of ping.

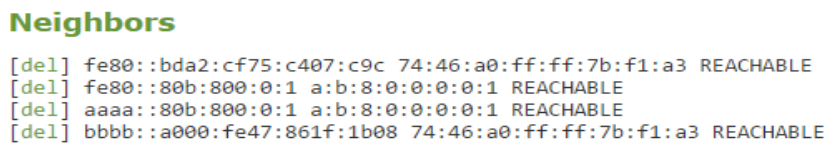

Fig. 5.5. Reachable neighbors detected by webserver

To see what it happens, Wireshark (version 1.8.2) is installed to see network traffic and tries to provide more information of this lack of connectivity, the result is that the neither lowpan0 nor wpan0 interface detects any traffic data among the network that comes from 6LBR router. Changing the configuration of 6LBR and disable all the fields of RA, the results are not satisfactory in terms of connectivity.

Once is discarded Openlabs module, is time to test second scenario of [Fig. 5.3], based on pIPv6 stack (Arduino + XBee shield + XBee). To implement the stack into Arduino Duemilanove is relatively easy; the procedure is described on Annex A.VI. With the project compiled into Arduino, the results in the webserver of 6LBR are reflected on website:

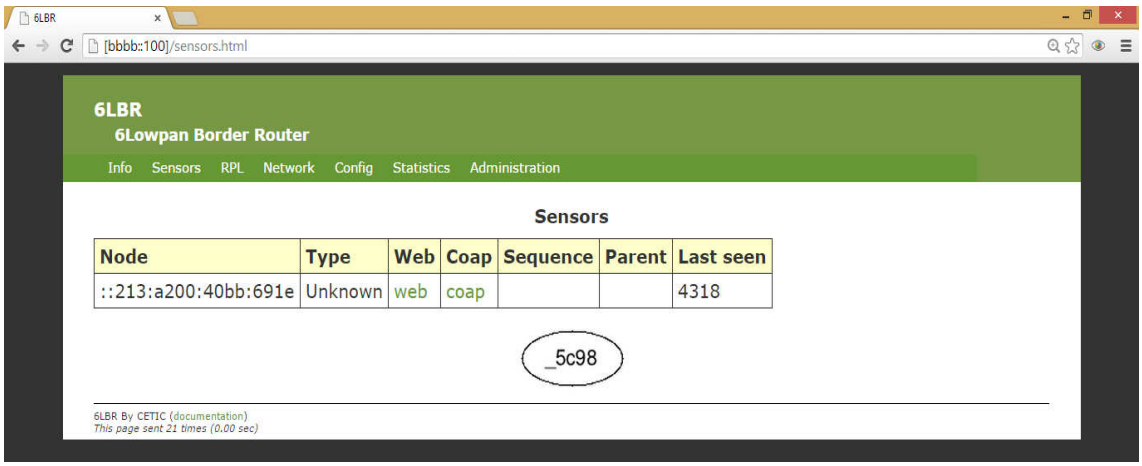

Fig. 5.6. 6LBR sensor detection.

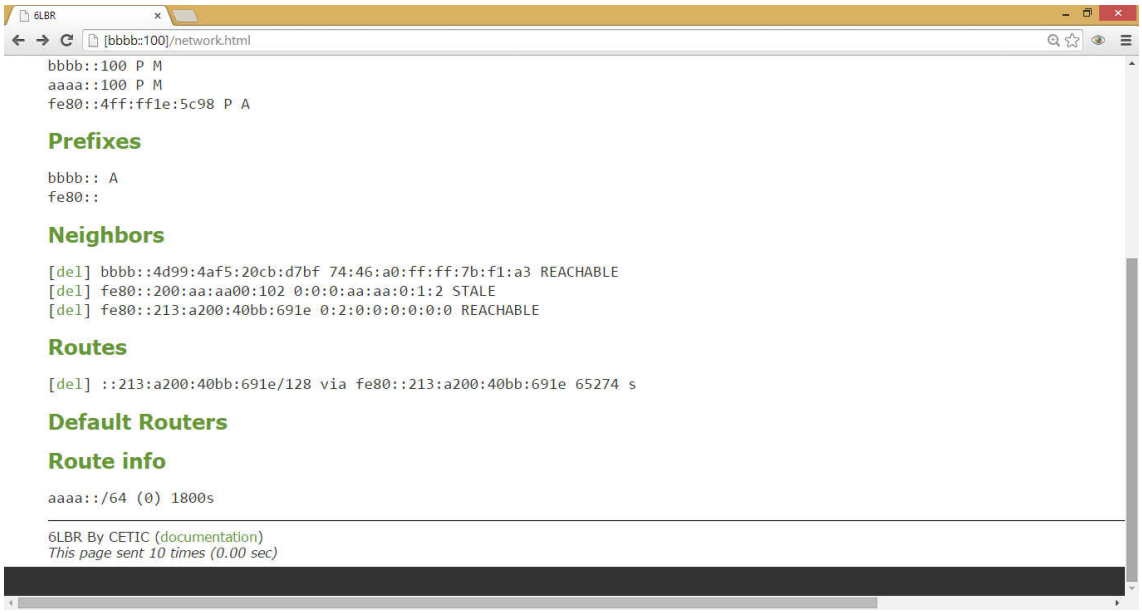

Fig. 5.7. 6LBR Network detection

The sensor is detected and REACHABLE state in the webserver (in contrast with the Openlabs module that appears STALE state), that means that the communication between 6LBR and pIPv6 stack is fine. Looking in detail the sensor appears without an automatic prefix assignment, appearing with a default prefix "::X". By the way, in pIPv6 stack (see [20]) informs that "prefix problem" and uses a linux router with NAT66 + tunnel interfaces to provide a prefix to the WSN in scenario that is not equal than the described in [Fig 5.3.].

Applying NAT66 on the scenario (installation procedure on Annex A.VII), the best option is to install it on 6LBR module because will translate the address becoming from outside the WSN and apply the correct prefix. The main problem is that the WSN interfaces of 6LBR are not visible from the OS, being impossible to apply NAT66 on 6LBR. In case to apply on 6LBR, the problem is that the module is not a Contiki module and appears an error (running tunslip6.c file).

Now, thinking in to install RADVD the problem is the same as 6LBR, it has no visible interfaces. Using third scenario of [Fig. 5.3], applying RADVD with *MODE=ROUTER* on Openlabs module the results are the same, the prefix is not assigned to the Arduino module.

Trying to realize all the possibilities with both modules, a test with *MODE=FULL-TRANSPARENT-BRIDGE* has performed, changing de external host IPv6 address by aaaa::101. With that mode 6LBR module just realizes packet forwarding between IPv6 network and WSN. The results, with or without using RADVD, are not satisfactory, because, in the case of Arduino, there is not ping connectivity from IPv6 network to WSN but is received RPL solicitation messages from Arduino pIPv6 stack. One considerable problem with bridge modes is that is not available webserver, that issue complicates a little bit the analysis to see how reacts 6LBR to the tests.

5.3.2. Openlabs compatibilities

As described in 6LBR performance subchapter the compatibility with 6LBR seems that is not possible due some incompatibility between modules. For that reason, it could be interesting a solution based on server that communicates with other Openlabs for a certain solutions with a very low cost in terms of hardware. The idea is to use equal modules, establishing a server that provides externally a webserver to see the activity in the network and the possibility to connect with the end nodes through UDP messages.

To realize this test is created a RPi with webserver (apache) and UDP server in a C code and, in the client node, with a UDP client that generates random temperatures every 8 seconds. Basically is a UDP IPv6 socket in WSN and webserver in IPv4 Ethernet interface. The data received in UDP server are saved into a logfile and the webserver shows the information. The configuration of the 6LoWPAN interface is realized in the same way just using the cnfigIface.sh described on chapter 5.2.

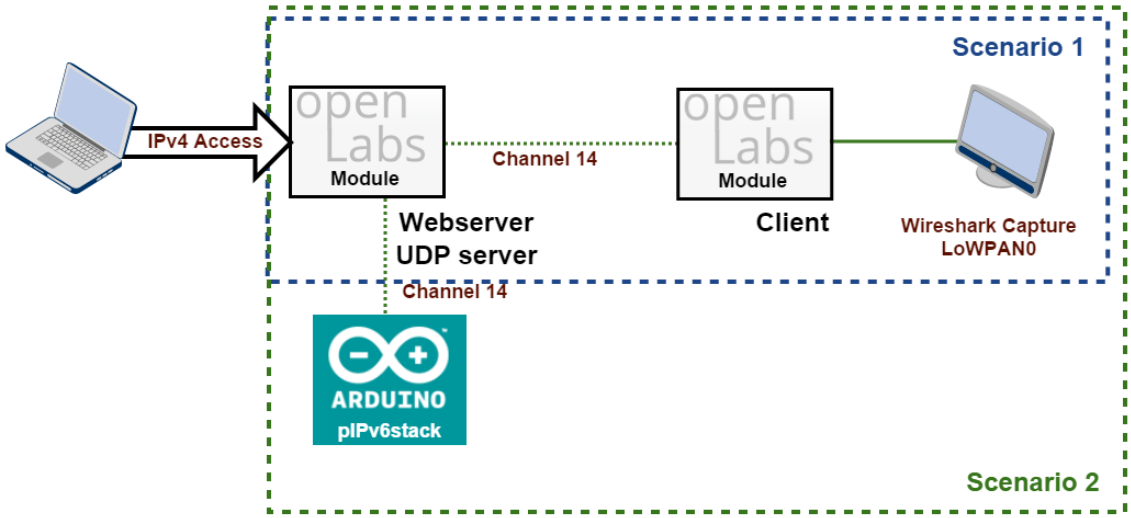

Fig. 5.8. 6LBR Network detection

First scenario of [Fig. 5.7] could be interesting to create a webserver that acts as IPv4-6LoWPAN gateway and that server collects all the information reflecting in a logfile the main activities of the network. This solution works well, the code written in C are available in annex A.III of this document. An example of how the temperatures are stored and presented in the webserver is seen in the following figure:

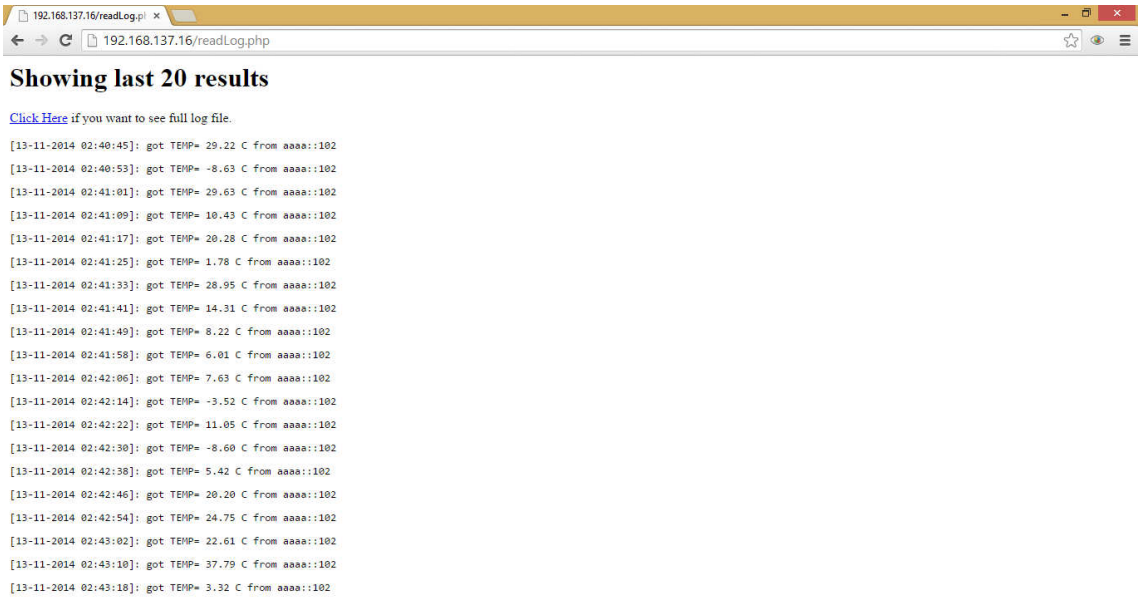

Fig. 5.9. Openlabs dialog with pIPv6 stack

Other interesting test is to build the second scenario of [Fig. 5.7], trying the communication between Openlabs module an Arduino pIPv6 stack, configuring previously described UDP server according to the characteristics of the pIPv6 stack and see if the Server receives some packet, the results are visible in the following figure.

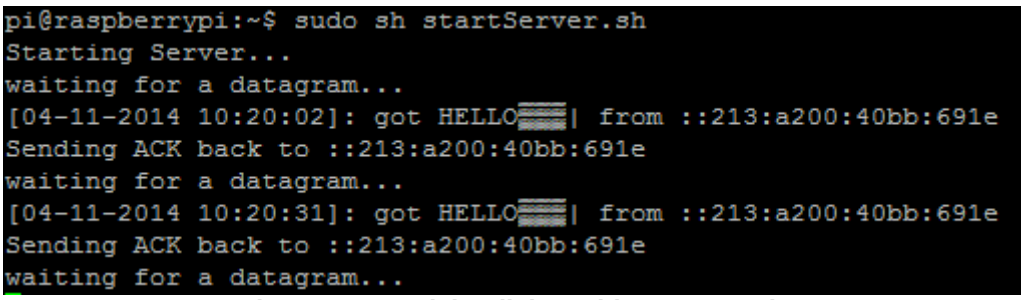

Fig. 5.10. Openlabs dialog with pIPv6 stack

As seen in the previous capture, the communication seems that at client level is success, sniffing in Openlabs module is possible to see the generation of RPL and UDP traffic becoming from the Arduino. That behaviour opens the possibility to introduce 6LoWPAN servers with external access, using this technique and have complex and heterogeneous networks, according the requirements and features that some application requires or even better, install a software router to reach the possibility to perform a border router solution.

CHAPTER 6. CONCLUSIONS AND FUTURE LINES

This chapter will try to synthesize the realization of this project providing some interesting conclusions to take into consideration. After these conclusions are described some ideas to be considerate for future enhancements and upgrades.

6.1. Conclusions

The current state of the IoT development has been checked and from the huge quantity of protocols the best one has been selected to develop an IoT network at the day of today according criteria parameters such as low-power consumption, open standard as example and the result was a combination of 802.15.4 + 6LoWPAN.

When the protocols are analysed and selected the next step was to realize a deep search to find the available solutions in the market. After that search, the best options are selected according to the closer solution with the requirements of PMT network and applying the criteria of cost, flexibility and standardization of the protocols. The result was to consider 6LBR, Arduino µIPv6 and pIPv6 stack, Openlabs 802.14.5 module and Redwire RedIO as the best ones for the project.

The main goal was to develop and test a border solution. It is noteworthy that both the 6LBR and Openlabs modules are on the edge of development products. This circumstance has made that the delivery time for these products has been very long and has affected the completion and development of this project. Even that, a development was realized with 6LBR, an open solution that provides an edge router to interconnect IPv6 network with 6LoWPAN network. 6LBR works correctly and confirmed with connectivity tests with Arduino pIPv6 stack mote.

Alternatively, a Openlabs module development was realized and the combination with 6LBR was not success in terms of connectivity but is deployed a solution that not exactly covers the requirements of the project but it can be useful to establish another kind of 6LoWPAN network based in server that collects all the data emitted for the motes of the network.

6.2. Environmental impact

In terms of environmental impact of this project, it is important to take into account the following points:

- Reduced use of materials: in WSN the connection between the different nodes is wireless, requiring less conductor material in terms of wire. Also the SoC equipment has less materials compared with an standard PC.

- Reduced Power consumption: as said on chapter 3.1 (page 28) is possible to supply of to 15 RPi with a supply of an standard PC, this reduction is considerable in terms of environmental impact. Also SLIP motes reduce the power consumption with sleep periods to get maximum power supply durability. Finally is it very important to remember that the majority of WSN protocols are specifically design to be as much efficient as possible with low-power consumption.
- Reusability: this modules are not strictly design to realize border router solution. In case that another solution will be better, the material used in this project is perfectly usable for other kind of project due his flexibility.
- Future applications deriving from IoT network: establish IoT network can increase enormously the efficiency of a broad aspects of daily life. Examples: irrigation, domotics, garbage collection, electronic noses, etc.

6.3. Future lines of development

Once the first stone of the deployment of the coordinator has been established for the IoT network there are some things that will be recommendable for the future, for that reason is provided information that can be important to get a reference for future enhancements and upgrades.

First, realize tests with 6LBR in a scenario with Arduino Mega (with µIPv6 stack) and the mentioned Arduino Duemilanove with pIPv6 and see how works in terms of prefix propagation and RPL DODAG generation. With that information will be possible to see if µIPv6 stack can "understand" prefix dialog with 6LBR and how interact the motes to generate motes tree. Also, it will be important to ensure that multihop forwarding works properly with these 6LoWPAN modules.

In Openlabs module will be interesting to install router software such as Quagga (see [50]) and try to perform a 6LoWPAN border router with CoAP and RPL capabilities, having a very low cost border-router solution based on linux 6LoWPAN stack.

Furthermore will be very convenient to dedicate some efforts doing several performance tests, since antennas are in development phase, there is not a lot of technical data a n datasheets that provide huge information about important aspects to establish this IoT networks. Tests as maximum number of supported nodes, error rates, transmission distance, etc. will be some parameters that will be interesting to appreciate for the establishment of WSN. Also, another aspect to contemplate is to use a well-known sniffer (based also on Contiki) as RZRAVEN USB Stick (Jackdaw) (see [51]), to see exactly what it happens with an interface that can ensure the traffic that flows in the WSN.

To conclude will be useful to contemplate the visibility of 6LBR server, public or private, because in the first case it requires tunnelling access with the IPv6 interface (see [52]).

REFERENCES

[1] Kushalnagar N., Montenegro G., Schumacher C., "*IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals"*, IETF "RFC 4919". Homesite: https://tools.ietf.org/html/rfc4919

[2] *IEEE Std 802.15.4™-2011 (Revision of IEEE Std 802.15.4-2006)* http://standards.ieee.org/

[3] Montenegro G., Kushalnagar N., "*Transmission of IPv6 Packets over IEEE 802.15.4 Networks"*, IETF "RFC 4944". Homesite: https://tools.ietf.org/html/rfc4944

[4] Hui J., Thubert P., "*Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks"*, IETF "RFC 6282". Homesite: https://tools.ietf.org/html/rfc6282

[5] Shelby Z., Chakrabati S., Nordmark E., Bormann C., "*Neighbour Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)"*, IETF "RFC 6775". http://tools.ietf.org/html/rfc6775

[6] Kim E., Kaspar D., Gomez C., Bormann C., "*Problem Statement and Requirements for IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Routing"*, IETF "RFC 6606". Homesite: http://tools.ietf.org/html/rfc6606

[7] Winter T., Thubert P., Brandt A., Hui J., Kelsey R., Levis P., Pister K., Struik R., Vasseur JP., Alexander R., "*RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks"*, IETF "RFC 6550". Homesite: https://tools.ietf.org/html/rfc6550

[8] Tsenkov T., "*RPL: IPv6 Routing Protocol for Low Power and Lossy Networks"* http://www.net.in.tum.de/fileadmin/TUM/NET/NET-2011-07-1/NET-2011-07-1_09.pdf

[9] Palàs-Areny R., *"Section 2 Sensing methods and sensors"*, Electronic Instrumentation course, (2013).

[10] LeHong H., Fenn J., Leeb-du Toit R., "*Hype Cycle for Emerging Technologies*", Gartner, Worldwide, July 2014.

[11] Middleton P., Kjeldsen P., Tully J., *"Forecast: The Internet of Things"*, Worldwide, November 2013.

[12] Shelby Z., Bormann C., *"6LoWPAN the wireless embedded internet"*, Wiley (2009), United Kingdom, ISBN: 9780470747995.

[13] Kernel compilation tutorial, homesite:
http://elinux.org/Raspberry_Pi_Kernel_Compilation

[14] *Parc Mediterrani de la Tecnologia*, Homesite:
http://www.pmt.es/

[15] Kent S., Seo K., "*Security Architecture for the Internet Protocol*", IETF "RFC 4301". Homesite: https://tools.ietf.org/html/rfc4301

[16] Villaverde B. C., Pesch, D, "*Constrained Application Protocol for Low Power Embedded Networks*": A Survey. In Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference, Palermo, Italy, ISBN: 978-1-4673-1328-5.

[17] *Raspberry Pi product*, sparkfun shop. Home site:
https://www.sparkfun.com/products/11546

[18] *Jennet-IP evaluation kit*. NXP semiconductors Homesite:
http://www.jennic.com/products/development_kits/
. Price, Digikey shop product site:
http://www.digikey.es/product-search/en?x=27&y=8&lang=en&site=es&KeyWords=JenNet-IP+EK040

[19] *Arduino µIPv6 Stack*. Telecom Bretagne. Github project wiki:
https://github.com/telecombretagne/Arduino-IPv6Stack

[20] *Arduino pIPv6 Stack*. Telecom Bretagne. Github project wiki:
https://github.com/telecombretagne/Arduino-pIPv6Stack

[21] *Arduino MEGA*, Diotronic Shop. Product site:
http://www.diotronic.com/raspberry-pi-arduino/arduino/placas/arduino-mega-atmel-atmega2560_r_1080_26634.aspx

[22] *Arduino UNO*, Diotronic Shop. Product site:
http://www.diotronic.com/raspberry-pi-arduino/arduino/placas/avr-arduino-uno-rev-3_r_1080_26633.aspx

[23] *Raspberry Xbee Shield*. Cooking-hacks Shop. Product site:
http://www.cooking-hacks.com/raspberry-pi-to-arduino-shield-connection-bridge

[24] *Arduino Duemilanove specifications*. Product wiki:
http://arduino.cc/en/pmwiki.php?n=Main/arduinoBoardDuemilanove

[25] *XBee Pro 802.15.4 Antenna*. Sparkfun shop website:
https://www.sparkfun.com/products/8742

[26] *Centre of Excellence in Information and Communication Technologies*. Homesite: https://www.cetic.be/

[27] *Redwire BR12*. Redwire description and price of the product website:

http://www.redwirellc.com/collections/frontpage/products/br12

[28] *Redwire BR12, Price of 6LoWPAN starter-kit*. Redwire Shop product site: http://www.redwirellc.com/collections/frontpage/products/6lowpan-starter-kit

[29] *Redwire IO*, Redwire description and price of the product website: http://www.redwirellc.com/collections/frontpage/products/io-embedded-router-contiki-based

[30] *Net-next Kernel*, Github project homesite: https://github.com/linux-wpan/linux-wpan-next.git

[31] *Arduino NANO*. Diotronic shop product price. Homesite: http://www.diotronic.com/raspberry-pi-arduino/arduino/placas/arduino-nano_r_1080_26760.aspx

[32] *Zigbit module*. Atmel shop product price homesite: http://store.atmel.com/PartDetail.aspx?q=p:10500243#tc:description

[33] *Zolertia Z1 discovery pack*. Zolertia shop product homesite: http://webshop.zolertia.com/product_info.php/products_id/30

[34] *Raspberry Pi*, Raspberry Pi downloads homesite: http://www.raspberrypi.org/downloads/

[35] *Raspberry Pi*. Raspberry Pi tutorial to install images homesite: http://www.raspberrypi.org/documentation/installation/installing-images/

[36] *CETIC 6LBR*. CETIC 6LBR wiki homesite: https://github.com/cetic/6lbr/wiki

[37] Sharma D., Verma S., Sharma K., "*Network Topologies in Wireless Sensor Networks: A Review*" IJECT Vol. 4, April - June 2013, India, ISSN : 2230-9543. Homesite: http://www.iject.org/vol4/spl3/c0116.pdf

[38] Sallent S., " *Chapter 1 Introduction*", Dimensionat de Xarxes lecture, UPC-EETAC slides, 2014.

[39] Arduino IDE, Arduino downloads homesite: http://arduino.cc/en/Main/Software

[40] Arduino pIPv6 stack. Github project homesite: https://github.com/telecombretagne/Arduino-pIPv6Stack

[41] Faludi R.,"Building Wireless Sensor Networks", O'Reilly, 2011, United States, ISBN: 978-0-596-80773-3

[42] Shelby Z., Hartke K., Bormann C., "*Constrained Application Protocol (CoAP) draft-ietf-core-coap-18*". Homesite: https://tools.ietf.org/html/draft-ietf-core-coap-18

[43] Grinch Border Router Blog. Homesite: http://sixpinetrees.blogspot.com.es/

[44] NXP semiconductors. Homesite: http://www.jennic.com/

[45] Zheng J., Lee MJ., "*A Comprehensive Performance Study of IEEE 802.15.4*". Homesite: http://hackipedia.org/Hardware/Zigbee/A%20Comprehensive%20Study%20Of%20Ieee%20802.15.4%20And%20Zigbee%20506.pdf

[46] Ott A., "*Wireless Networking with IEEE 802.15.4 and 6LoWPAN*" slides, Embedded Linux Conference, Europe, November 5, 2012. Homesite: http://elinux.org/images/7/71/Wireless_Networking_with_IEEE_802.15.4_and_6LoWPAN.pdf

[47] García Arano C., "*Impacto de la seguridad en redes inalámbricas de sensores IEEE 802.15.4*." UCM Master thesis, 2010, Madrid.

[48] SchönwälderInter J., "*Internet of Things: 802.15.4, 6LoWPAN, RPL, COAP*" Jacobs University slides. Homesite: http://www.utwente.nl/ewi/dacs/Colloquium/archive/2010/slides/2010-utwente-6lowpan-rpl-coap.pdf

[49] Cisco IR500. Product Datasheet. Homesite: http://www.cisco.com/c/en/us/products/collateral/routers/500-series-wpan-industrial-routers/datasheet-c78-730550.pdf

[50] Quagga Routing suite (Linux). Homesite: http://www.nongnu.org/quagga/

[51] RZRAVEN USB Stick (Jackdaw). Contiki wiki Homesite: http://contiki.sourceforge.net/docs/2.6/a01799.html

[52] Dickinson B., "*Setting up a Raspberry Pi as an IPv6 gateway using Hurricane Electric*". March, 2013. Homesite: http://www.dickson.me.uk/2013/03/15/setting-up-a-raspberry-pi-as-an-ipv6-gateway-using-hurricane-electric/

[53] Prentice S., LeHong H., "*Uncover Value From the Internet of Things With the Four Fundamental Usage Scenarios*". Gartner, Worldwide, May 2013.

[54] LeHong H., Cerley DW, Steenstrup K., Prentice S., "*The Internet of Things Is Moving to the Mainstream*". Gartner, Worldwide, January 2013.

[55] Raspberry Pi, Homesite: http://www.raspberrypi.org/

[56] TinyOS. Homesite: http://www.tinyos.net/

[57] Contiki. Homesite: http://www.contiki-os.org/

[58] ARM Mbed. Homesite: http://mbed.org/

[59] Arduino. Homesite: http://arduino.cc/

[60] Redwire. Homesite: http://www.redwirellc.com/

[61] Openlabs. Homesite: http://openlabs.co/

[62] Malmsteen P., "Python-xbee Documentation" Release 2.1.0, April, 2013.

[63] Hauweele D., "*Linux-based 6LoWPAN border router*" slides, University of Mons, August, 2013.

[64] "*XBee/XBee-PRO 802.15.4 Professional Kit, Getting started guide*". Digi International, June, 2012.

[65] "*XBee/XBee-PRO RF Modules, product manual v1.xEx - 802.15.4 Protocol*". Digi International. June, 2012

GLOSSARY

BLIP	Berkeley Low-power IP
CAP	Contention-Access-Period
CFP	Contention Free-Period
CoAP	Constrained Application Protocol
CoRE	Constrained RESTful Environment
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CSS	Chirp Spread Spectrum
DLL	Data Link Layer
DODAG	Destination Oriented Directed Acyclic Graph
DTLS	Datagram Transport Layer Security
EXI	Efficient XML Exchange
FFD	Full Function Device
FIB	Forwarding Information Database
GPIO	Generic Port Input/output
GTS	Guaranteed Time Slots
GUI	Graphical User Interface
IID	Interface IDentifier
IoT	Internet of Things
IPSec	IP security
LLN	Low power and Lossy Networks
LR-WPAN	Low Rate Wireless Personal Area Network
M2M	Machine-to-Machine
MAC	Media Access Control

MAP-T	Mapping of Address and Port Translation
MIB	Management Information Base
MSDU	MAC Service Data Unit
MTU	Maximum Transmission Unit
NAT66	IPv6-to-IPv6 Network Address Translation
OS	Operating System
P2P	Peer to Peer
PHY	PHYsical
QoS	Quality of Service
RA	Router Advertisement
RADVD	Router ADVertisment Daemon
RFD	Reduced Function Device
RFID	Radio Frequency IDentification
RIB	Routing Information Database
SBC	Single Board Computer
SoC	System on Chip
SPI	Serial Peripheral Interface
TDMA	Time Division Multiple Access
ULA	Local Unicast address
UWB	Ultra Wide Band
WLAN	Wireless Local Area Network
WPAN	Wireless Personal Area Network
WSN	Wireless Sensor Network

ANNEX

TITLE: Internet of things implementation with Raspberry Pi

MASTER DEGREE: Master in Science in Telecommunication Engineering & Management

AUTHOR: Nasarre Ramírez, Alex

DIRECTOR: Polo Cantero, José

DATE: November, 10 th 2014

A.I IP features

This subchapter is focused in general features of IP protocol that complements the information provided in the document.

A.I.1 The TCP/IP Network Stack

The basic TCP/IP stack is defined in the following figure. In case of IPv6 the protocols need to be prepared to work with IPv6 addresses.

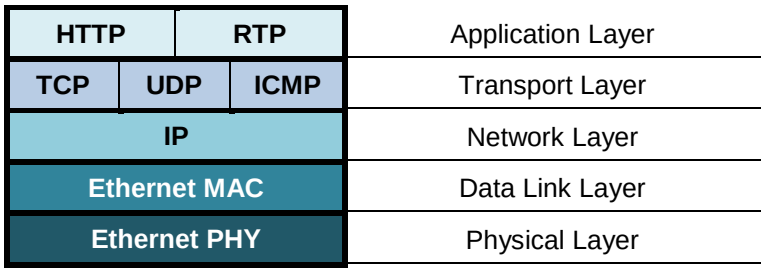

Fig. A.1. IP protocol stack

A.I.2 IPv6 Header

In the following figure is represented the fixed header in IPv6:

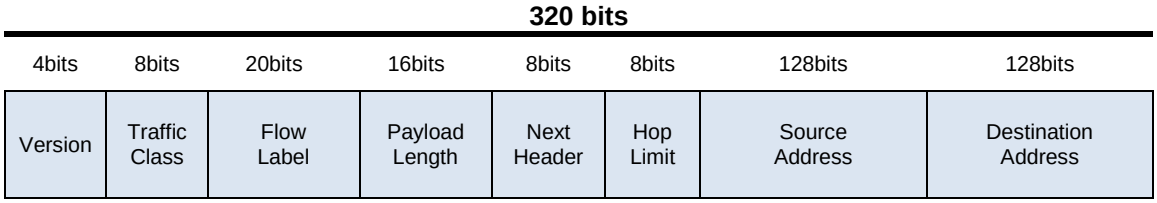

Fig. A.2. IPv6 Fixed Header (320 bits).

A.II Hardware Support

This subchapter describes the hardware support for the different solutions described on Chapter 4.

A.II.1 Tiny OS hardware support

Platforms:

- EPIC
- Lmote
- Shimmer
- IRIS
- Telos Rev. B
- MicaZ
- NXTMOTE
- Mulle
- TinyNode
- Zolertia Z1
- UCMote Mini
- Mica2

- Mica2dot

Microcontrollers:

- Atmel ATmega128
- Texas Instruments MSP430
- Intel XScale PXA271

Radios:

- CC1000
- CC2420
- CC1100/CC2500
- AT86RF212

A.II.2 Linux kernel hardware support (Openlabs)

Radios:

- AT86RF230
- MRF24J40
- CC2520

A.II.3 Contiki hardware support

Table A.1 Contiki hardware support

MCU/SoC	Radio	Platforms	COOJA Support
RL78	ADF7023	EVAL-ADF7023DB1	
TI CC2538	Integrated	Cc2538dk	
TI MSP430x	TI CC2420	Exp5438, z1	Yes
	TI CC2520	Wismote	Yes
Atmel AVR	Atmel RF230	Avr-raven, avr-rcb, avr-zigbit, iris	
	TICC2420	Micaz	Yes
Freescale MC1322x	Integrated	Redbee-dev, redbee-econotag	
ST STM32w	Intengrated	Mb851, mbxxx	
TI MSP430	TI CC2420	Sky, jcreate, sentilla-usb	Yes
	TI CC1020	Msb430	
	RFM TR1001	Esb	Yes
Atmel Atmega128 RFA1	Integrated	Avr-atmega128rfa	
Microchip pic32mx795f512l	Microchip mrf24j40	Seed-eye	
TI CC2530	Integrated	Cc2530dk	
RC2300/RC2301	Integrated	Sensinode	
6502		Apple2enh, atari, c128, c64	
Native		Native, minimal-net, cooj	Yes

A.III Openlabs code

This subchapter describes how to build a centralized server that collects the information of clients and represents into a PHP file in a webserver. Also is proposed a code to establish an UDP client that sends random temperature every 8 seconds.

A.III.1 IPv6 UDP Server

First step is to install possible application that makes possible the creation of a webserver and applications for future development:

```
$ apt-get install apache2 php5 mysql-client mysql-server autoconf vsftpd
wireshark tcpdump
```

serverUDP.c

```
/*
  Alex Nasarre IPv6 UDP Server for Openlabs Module. See the
  GNU General Public License (http://www.gnu.org/copyleft/gpl.html)
  for more details.
*/

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
#include <time.h>

#define PORT 8765
#define MESSAGE "ACK"
char ts[50];

/* Getter of TimeStamp */
char * getTS(void){
  time_t t = time(NULL);
  struct tm * p = localtime (&t);
  strftime(ts, sizeof(ts),"[%d-%m-%Y %H:%M:%S]:", p);
  return (char *)ts;
}

int main(void)
{
  int sock;
  socklen_t clilen;
  struct sockaddr_in6 server_addr, client_addr;
  char buffer[1024];
  char addrbuf[INET6_ADDRSTRLEN];

  /* create a DGRAM (UDP) socket in the INET6 (IPv6) protocol */
  sock = socket(PF_INET6, SOCK_DGRAM, 0);
  if (sock < 0) {
    perror("creating socket");
    exit(1);
  }

#ifdef V6ONLY
  // setting this means the socket only accepts connections from v6;
  // unset, it accepts v6 and v4 (mapped address) connections
  { int opt = 1;
    if (setsockopt(sock, IPPROTO_IPV6, IPV6_V6ONLY, &opt, sizeof(opt)) < 0) {
      perror("setting option IPV6_V6ONLY");
      exit(1);
    }
  }
#endif
```

```
  /* create server address: this will say where we will be willing to
     accept datagrams from */

  /* clear it out */
  memset(&server_addr, 0, sizeof(server_addr));

  /* it is an INET6 address */
  server_addr.sin6_family = AF_INET6;

  /* the client IP address, in network byte order */
  /* in this example we accept datagrams from ANYwhere */
  server_addr.sin6_addr = in6addr_any;

  /* the port we are going to listen on, in network byte order */
  server_addr.sin6_port = htons(PORT);

  /* associate the socket with the address and port */
  if (bind(sock, (struct sockaddr *)&server_addr,
           sizeof(server_addr)) < 0) {
    perror("bind failed");
    exit(2);
  }

  char strLog[500];

  while (1) {
    /* now wait until we get a datagram */
    printf("waiting for a datagram...\n");
    clilen = sizeof(client_addr);
    if (recvfrom(sock, buffer, 1024, 0,
                 (struct sockaddr *)&client_addr,
                 &clilen) < 0) {
      perror("recvfrom failed");
      exit(4);
    }

    /* now client_addr contains the address of the client */
    sprintf(strLog, "%s got %s from %s\n", getTS(), buffer,
           inet_ntop(AF_INET6, &client_addr.sin6_addr, addrbuf,
                     INET6_ADDRSTRLEN));
    printf("%s",strLog);

    /* Open a logfile */
    FILE *log = fopen("/var/www/server.log", "a+");
    if (log == NULL)
    {
           printf("Error opening file!\n");
           exit(1);
    }

    /*Print into logfile*/
    fprintf(log, "%s",strLog);
    /*close logfile*/
    fclose(log);

    printf("Sending     ACK     back     to     %s\n",     inet_ntop(AF_INET6,
&client_addr.sin6_ad$
                     INET6_ADDRSTRLEN));

    if (sendto(sock, MESSAGE, sizeof(MESSAGE), 0,
               (struct sockaddr *)&client_addr,
               sizeof(client_addr)) < 0) {
      perror("sendto failed");
      exit(5);
    }

  }

  return 0;
}
```

Initialization script

startServer.sh
```
#!/bin/bash
chmod -R 777 /var/www
gcc -o servC serverUDP.c
echo "Starting Server..."
./servC
```

PHP files in /var/www/ directory

readLog.php
```
<html><body><h1>Showing last 20 results</h1>
<?php
header("refresh: 57;");
?>

<p>
<a href="fullLog.php">Click Here</a> if you want to see full log file.
</p>
<?php
$file = "server.log";
    $data = file($file);
    $end = count($data);
    $first = $end-20;

    $number = range($first,$end);

    foreach($number as $n) {
        echo '<pre>';
        echo $data[$n];
        echo '</pre>';
    }
?>
</body>
</html>
```

fullLog.php
```
<html><body><h1>Showing server.log</h1>
<?php
header("refresh: 30;");
?>

<?php
    $file = "server.log";
    $data = file($file);
    $end = count($data);
    $first = 0;

    $number = range($first,$end);

    foreach($number as $n) {
        echo '<pre>';
        echo $data[$n];
        echo '</pre>';
    }
?>
</html>
```

A.III.2 Openlabs IPv6 UDP Client

clientUDP.c
```
/* Alex Nasarre IPv6 UDP Client for Openlabs Module. See the
  GNU General Public License (http://www.gnu.org/copyleft/gpl.html)
  for more details.
*/

#include <stdio.h>
```

```
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
#include <time.h>

#define PORT 8765
#define MESSAGE "HELLO!"
#define SERVADDR "0::ffff" /*Broadcast Address*/

int main(void)
{
  /* Generator of random temp*/
  srand(time(NULL));
  float r = -10 + rand() % 50 + (rand() % 100)/100.00;
  char msg[200];
  sprintf(msg, "TEMP= %.2f C", r);

  int sock;
  socklen_t clilen;
  struct sockaddr_in6 server_addr, client_addr;
  char buffer[1024];
  char addrbuf[INET6_ADDRSTRLEN];

  /* create a DGRAM (UDP) socket in the INET6 (IPv6) protocol */
  sock = socket(PF_INET6, SOCK_DGRAM, 0);

  if (sock < 0) {
    perror("creating socket");
    exit(1);
  }

  /* create server address: where we want to send to */

  /* clear it out */
  memset(&server_addr, 0, sizeof(server_addr));

  /* it is an INET address */
  server_addr.sin6_family = AF_INET6;

  /* the server IP address, in network byte order */
  inet_pton(AF_INET6, SERVADDR, &server_addr.sin6_addr);

  /* the port we are going to send to, in network byte order */
  server_addr.sin6_port = htons(PORT);

  /* now send a datagram */
  if (sendto(sock, msg, sizeof(msg), 0,
             (struct sockaddr *)&server_addr,
             sizeof(server_addr)) < 0) {
      perror("sendto failed");
      exit(4);
  }

  printf("waiting for a reply...\n");
  clilen = sizeof(client_addr);
  if (recvfrom(sock, buffer, 1024, 0,
               (struct sockaddr *)&client_addr,
               &clilen) < 0) {
      perror("recvfrom failed");
      exit(4);
  }

  printf("got '%s' from %s\n", buffer,
         inet_ntop(AF_INET6, &client_addr.sin6_addr, addrbuf,
                   INET6_ADDRSTRLEN));

  /* close socket */
  close(sock);

  return 0;
}
```

Initialization script

startCli.sh
```
#!/bin/bash

gcc -o cliUDP clientUDP.c

while :
do
        ./cliUDP
        sleep 8
done
```

A.IV. XBee Configuration

This subchapter relates the configuration of XBee module to be part of Arduino pIPv6 stack solution, in this project has done with XCTU application starting with a XBee factory default configuration:

- Channel (CH) = E
- PAN ID (ID) = ABCD
- Destination Address High (DH) = 0
- 16-bit Source Address (MY) = 2 (can be higher than 0)
- MAC Mode (MM) = 802.15.4 with ACKs [2]
- API Enable (AP) = API enabled w/PPP [2]

A.V. RPi basic software installation

To establish a basic OS for the project Raspbian OS is selected, based on Linux Debian, the selection it's because is the most used OS on development projects due to his well-known robustness and to has high compatibility with common software and obtaining good performance.

In this project is used Raspbian OS "Wheezy" (Release 2014-01-07) with 3.10 kernel. This image is available in RPi on downloads section (see [34]). Once downloaded the distribution, is possible to load the image in a SD in Windows (using SD Formatter + Win32DiskImager) or Linux (dd tool) just following the steps of the documentation (see [35]).

To develop with RPi is recommended to buy a USB to TTL Serial cable to have direct communication with the RPi ignoring IP configuration in the ssh channel on the Ethernet port, this cable is available on eBay with a cost less than 3€ (shipping included).

Fig. A.3. USB to TTL Serial Cable *(source: www.adafruit.com)*

Once Raspbian is installed and serial cable plugged correctly on the RPi and on the USB a console (minicom, putty, cygwin...), is required to configure connection parameters (*COM port, speed 115200 bauds, 8 data bits, 1 stop bit, without parity and flow control XON/XOFF*) and after connect the power cable is obtained the boot of the RPi on the terminal. Another recommended aspect is provide Internet connection on RPi to install and update applications. For this project is configured in a shared connection on WLAN interface of the Internet connected PC as seen in the following picture:

Fig. A.4. Shared internet connection scheme

All the mentioned open-source projects works with github, is a good idea to install *git* application to work with these kind of projects easily. Other useful application is *wget* that makes possible, in a very easy way, to download files in Linux terminals. To install both applications is required to type the following command:

```
$ sudo apt-get install git wget
```

With these steps are established the minimum to develop on 6LoWPAN on a RPi, after that, are realized the specific steps of the selected options.

A.VI. Arduino pIPv6 stack compilation

First is download Arduino IDE and pIPv6 stack (see [39] and [40]). Second step is install Arduino IDE and create a folder called *picoIPv6MacV1* and paste inside all the **.h, *.cpp* and *picoIPv6MacV1.ino* into this created folder. After that, is required to plug Arduino on a USB port and open *picoIPv6MacV1.ino*

with the Arduino IDE and finally verify and load the project into the Arduino Duemilanove

A.VII. NAT66 implementation tutorial

The tutorial provided by pIPv6 stack to install NAT66 on the computer specifies the following steps:

- Copy the nat66_linux_deployment.tar.gz code to the linux machine. It is located in the picoIPv6/tutorial folder. Decompress it in that machine.
- In the nat66.c file, change the value of the delegated prefix that is going to be used (this case aaaa::) on the sensor/INTERNAL/6LoWPAN network:

```
/* Computation of the src addr - ::/64 prefix must be replaced by
cccc::/64 prefix*/
    ip6h->ip6_src.s6_addr[0]=0x20;
    ip6h->ip6_src.s6_addr[1]=0x01;
    ip6h->ip6_src.s6_addr[2]=0x06;
    ip6h->ip6_src.s6_addr[3]=0x60;
    ip6h->ip6_src.s6_addr[4]=0x73;
    ip6h->ip6_src.s6_addr[5]=0x01;
    ip6h->ip6_src.s6_addr[6]=0x00;
    ip6h->ip6_src.s6_addr[7]=0x57;
```

to

```
/* Computation of the src addr - ::/64 prefix must be replaced by
cccc::/64 prefix*/
    ip6h->ip6_src.s6_addr[0]=0xaa;
    ip6h->ip6_src.s6_addr[1]=0xaa;
    ip6h->ip6_src.s6_addr[2]=0x00;
    ip6h->ip6_src.s6_addr[3]=0x00;
    ip6h->ip6_src.s6_addr[4]=0x00;
    ip6h->ip6_src.s6_addr[5]=0x00;
    ip6h->ip6_src.s6_addr[6]=0x00;
    ip6h->ip6_src.s6_addr[7]=0x00;
```

- In the launch.sh file, change the following line by specifying the delegated prefix again:

```
# ROUTE CONFIGURATION FOR NATING FROM OUTSIDE TO SENSOR NETWORK
# Add a route for packets that goes to sensor network in order to nat
them thanks to tunnel tun_otos - Only cccc::/64 packets must be natted
    ip -6 route add 2001:660:7301:57::/64 dev tun_otos
```

to

```
# ROUTE CONFIGURATION FOR NATING FROM OUTSIDE TO SENSOR NETWORK
# Add a route for packets that goes to sensor network in order to nat
them thanks to tunnel tun_otos - Only cccc::/64 packets must be natted
    ip -6 route add aaaa::/64 dev tun_otos    # generic case: ip -6
route add ****:****:****:****::/64 dev tun_otos
```

- Go to ~/contiki/tools and compile the tunslip6 tool:

```
      $ cc -o tunslip6 tunslip6.c
```

- Copy the tunslip6 executable to the linux machine which will run the NAT66.
- Connect the linux machine (NAT66 router) to the ipv6 network where the delegated prefix is configured (RSM-B25 in this case).
- Connect the Tmote Sky with the uploaded rpl-border-router code to the linux machine with the NAT66 and run the tunslip6 application:

```
$ sudo ./tunslip6 -v -t tap0 aaaa::212:7400:115e:c49b
($ sudo ./tunslip6 -v -t tap0 2001:660:7301:57:0012:7400:10cf:c311 )
```

- Run the launch.sh script in order to configure the NAT66 router in the linux machine:

```
$ sudo ./launch.sh
```

A.VIII. RADVD Installation and configuration

To install RADVD type the following command in terminal:

```
$ apt-get install radvd
```

Once is installed is required to create /etc/radvd.conf and for the project is edited with the following:

```
#The interface can be tap0, eth0 or lowpan0 depending the device.
interface tap0
{
   AdvSendAdvert on;
   AdvManagedFlag off;     #stateless autoconfiguration
   MaxRtrAdvInterval 10;   #resend RA @ random times, max 10sec delay
   prefix aaaa::/64  #announce prefix to clients
   {
       AdvOnLink on;
       AdvAutonomous on;
       AdvRouterAddr on;
   };
};
```