Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA

PROYECTO FINAL DE CARRERA

VIZWALT: A VISUALIZATION TOOL FOR WSN EXPERIMENTS ON THE WALT PLATFORM BASED ON COOJA

Studies: Ingeniería de Telecomunicaciones

Author: Jorge Luis Baranguán Castro

Supervisors at INP-Grenoble: Andrzej Duda and Franck Rousseau

Advisor at UPC: Josep Paradells Aspas

Year: March 2014

Index

Abstract

Simulators and emulators for Wireless Sensor Networks like COOJA[1] are a valuable tool for system development. However, it is necessary to make tests with real sensor nodes, and when trying to test and debug new protocols and tools with them, it becomes more difficult since nodes just have some leds blinking to know if they are performing any action and we missed any tool to make this work easier.

We propose a software tool to be able to replay the network behaviour in a visualization interface. This way, we can perform an easy analysis on what is happening in the network. Afterwards, we have implemented a prototype testbed to test this software in a real network.

1 Introduction

1.1 Motivation

Software development in wireless sensor networks is difficult and tedious due to the distributed nature of sensor networks, because the compiled code should be load to a set of sensor nodes each time before test it, and this set can be large.

The usage of COOJA, a simulator for the Contiki sensor node operating system, make the developing process easier because we can test our code in a simulator, using the number of sensor nodes that we want and simulate it with a speed faster than real time, being able to make long simulations in a short time. Anyway, after this process it is necessary to test the code in a real sensor network, either we use a testbed or a big deployed network. At this time, we find out the problem of the complexity to debug in a wireless sensor network (WSN) because of the distance between nodes, the difficulty to observe the leds and, basically, because it's not possible to see the packets flowing in the network since it uses the radio medium.

Some solutions have been explored using distributed sniffers to make a non-intrusive monitoring of the network, and complex error tracking algorithms, but our purpose is to develop a debugging tool that allow us to perform an easy debugging even whether we don't have much re-

sources to have enough sniffers to monitor all the network or if we don't believe that it is worth to develop these types of algorithms. There are also some approaches using traces generated by the sensor nodes that you can collect in a server and display them in your terminal. However, the terminal is not a debug-friendly interface, because reading and understand a big amount of raw text is not easy.

1.2 Project approach

Our approach departs from the need to obtain a method to debug simply and cheaply, and that allows us to be able to visualize what is happening in the network. To solve this need we have decided to create a tool called VizWalT, to be able to display the behaviour of a network in an easy understandable interface, making the process of testing in a WSN comfortable and cheap.

The project has consisted of the development of the debugging tool called VizWalT, and the implementation of a prototype testbed formed by 4 modules: VizWalT, an Experiment Controller, a RPi Control, and a modification to the Sensor's Radio Interface.

VizWalT is a project developed with the purpose of replaying a wireless sensor network (WSN) behaviour through traces analysis. The main concept on which this project is based is that sensors generate some traces when they perform some actions and it is possible to get them through the serial port. With this extracted information it is possible to reproduce the behaviour of the network in a computer and using VizWalT, to visualize the packet flow inside a WSN. This tool allows to debug all kinds of new protocols or tools.

VizWalT is built like a COOJA plugin, as a way to integrate this tool into the Contiki simulator. Nevertheless, it changes the paradigm of COOJA simulator because VizWalT changes its purpose from simulate sensors behaviour to show the behaviour of real sensors. VizWalT reuses the Cooja graphic interface to display all the communication main events happening in the network. Thanks to VizWalT it is possible to visualize the transmission of a packet, the reception of a packet and the state of the sensor node radio interface. This turns VizWalT into a usefull tool to evaluate the performance of any protocol in terms of energy or reliability using real sensor nodes.

The prototype testbed has been developed to permit us to test VizWalT with a real network. This testbed is composed of the Experiment Controller, which can be placed in the same computer in which VizWalT is or in other computer closer to the nodes. Each node is composed of a Raspberry Pi and one or more sensors. The Raspberry Pi controlls the configuration of the sensors and communicates the sensors with the Experiment Controller through their USB. The sensors that we use are the ST Harvesting Sensor GreenNote V.2.1.1, and we have introduced some modifications into their radio interface code to be able to extract the needed information to use VizWalT.

This document reports what has been done in the project during 6 months at Drakkar Group of Laboratoire d'Informatique de Grenoble (LIG).

2 Ensimag

Ensimag is the school belonging to the Grenoble INP group (Grenoble Institut of Technology). Ensimag is one of the top French educational institutions specialized in computer science, applied mathematics and Telecommunications. Students at Ensimag are admitted after strong competition during two years of undergraduate studies. Studies at Ensimag are of three years' duration and lead to the French degree "Diplôme National d'Ingénieur".

2.1 Laboratoire d'Informatique de Grenoble (LIG)

LIG, Grenoble Informatics Laboratory, is the largest research laboratory of Informatics in Grenoble, France. It's the union of the 24 research teams coming from Ensimag, INRIA Rhône-Alpes, the CNRS (National Scientific Research Centre), Joseph Fourier University, Pierre-Mendes France University and Stendhal University. Over 500 people, researchers, PhD students, and professor-researchers work at LIG. The scientific project of the LIG is "ambient and sustainable IT". The goal is to leverage the complementary nature and recognised quality of the 23 research teams of the LIG to contribute to fundamental aspects of the discipline (modelling, languages, methods, algorithms) and to create

a synergy between the conceptual, technological and societal challenges that surround this theme.[2]

2.2 Drakkar Group

The Drakkar group investigates various aspects of network protocols and multimedia applications with the emphasis on wireless networks and sensors networks. The Drakkar group is part of the LIG Laboratory. My final studies project has been made in the Drakkar group, under the supervision of Andrzej Duda, Franck Rousseau and Etienne Dublé. Drakkar group is developing project consisting on a wireless sensor network testbed called WalT, and VizWalT is designed to be a module of this project. The aim is to have a tool that allow the researchers to visualize what is happening inside this wireless sensor network testbed.

3 Project execution

3.1 Initial Project Objectives

The initial project objectives were to create a modular implementation of a WSN testbed prototype. This prototype should be a base to the development of a WSN testbed called WalT that is going to be deployed at Ensimag building by the Drakkar group. WalT has the objective of being a place to test all the research studies developed by some research groups, including Drakkar group. WalT should be a low-budget WSN testbed, so it is necessary to find solutions to implement all the necessary modules in the easier and cheaper way.

Due to the partnership between the Drakkar group and France Telecom, WalT should be as compatible as possible with the France Telecom analysis tools, that use tools so extended like WireShark, which uses pcap format. That is one of the main reasons to make a modular implementation, that allows us to make our software tool really compatible with any software we want to use, by making few changes. That is thanks to the little dependency between nodes, that allows us to replace one of the modules without having to change the code of the other modules.

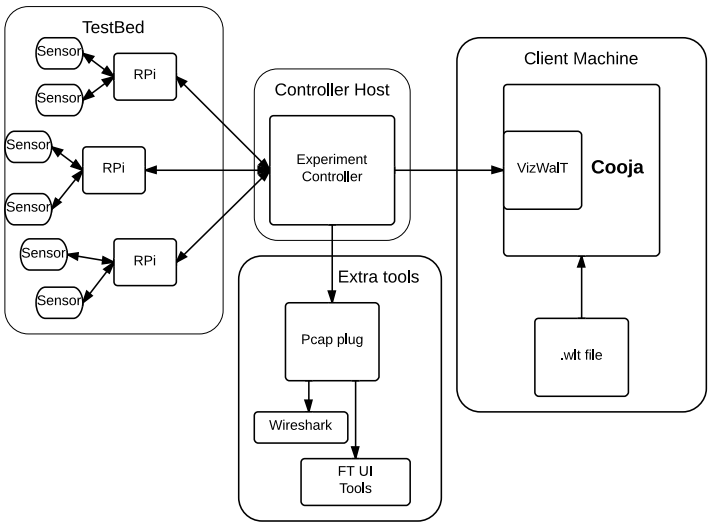

Figure 1: Project overview

3.2 Implementation process

This project was programmed to be realized in 5 phases. Each phase is based on the development of an autonomous software module that can be joined to the other modules to create the WSN testbed. These 5 phases are structured in the development of four modules and the test of the testbed. The modules that we have developed are VizWalT, the Experiment Controller, the RPi Control and the Sensors modification.

VizWalT is a visualization tool for replaying the network in a graphic interface. VizWalT is the first developed moduled, because it can work with the rest of the modules of the testbed or act as a standalone module, replaying an input file properly formatted.

The Experiment Controller is the module that manages the experiment of the wireless sensor network, and makes the communication between the network and VizWalT.

The Raspberry Pi Controller is a part of the network, and manages the performance of the sensor, the flashing of the sensors, to upload the code, and to transmit the traces extracted from the sensors to the Experiment Controller.

Figure 2: GreenNet sensor

Sensor code modifications is the module that introduces some modifications and tools into the sensors code to extract some traces that allow us to replay what happens in the network in the graphic interface of VizWalT.

The last phase is the real test in a small testbed with the GreenNet harvesting sensors from ST Microelectronics (See Figure **??** and **??**). To that purpose we have implemented a testbed consisting of 3 sensors, two Raspberry Pis, and a laptop.

4 Project Modules

4.1 VizWalT

We first present a high-level overview of VizWalT before describing how VizWalT achieves to display the network simulation.

VizWalT, as a Cooja plugin, is a Java-based plugin that introduces a new window to the basic COOJA interface where it is possible to see some properties, instructions and to choose some options. As COOJA is a Contiki OS simulatior, and VizWalT is a COOJA plugin it has an event driven approach.

VizWalT is designed to work in two possible scenarios: **post-analysis**

Figure 3: GreenNet sensor

and **real-time** simulation. In the second one, VizWalT reads in real time the information provided by the traces extracted from the sensor nodes.

4.1.1 Working modes

Real-Time simulation In the Real-Time scenario, VizWalT works on "pseudo real time". This means that VizWalT will replay the behaviour of the WSN with a delay of 1 second from the event happening. This delay is introduced to permit VizWalT to make some computations and determine some properties as which sensors are receiving each transmitted packet. This time has been set to 1 second because, despite the delay between the sending and the reception of a packet through the radio medium, 1 second is larger than any expectant delay in a "normal dimensioned" WSN, so it is expected to be a big enough "guard interval".

This working mode has been designed to integrate VizWalT like a module of a more complex system. It has been developed to work with a WSN and distributed data testbed called WalT, created by UJF/INP (Grenoble - FR). The communication between VizWalT and the module who creates this input information is through the standard input. See Figure 5.

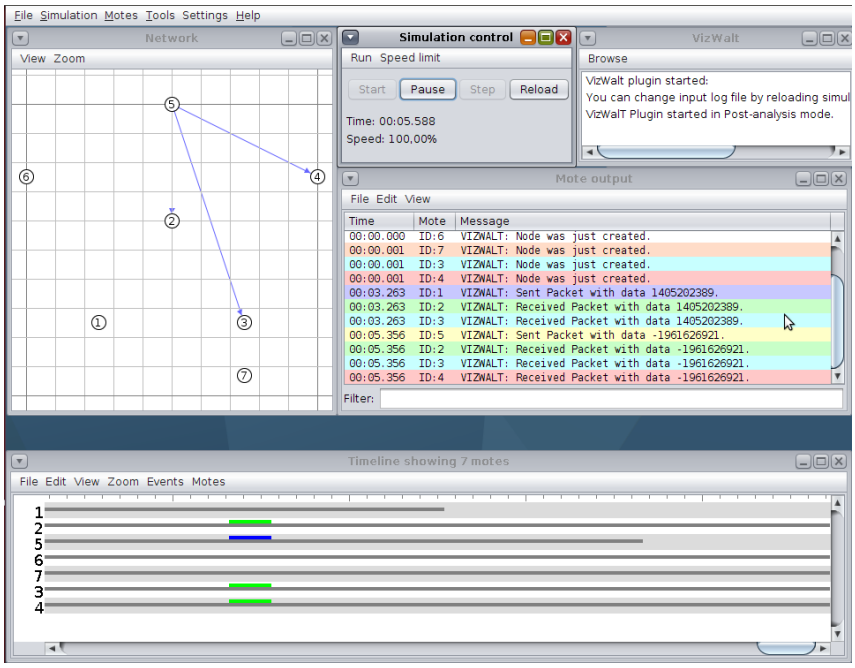

Figure 4: VizWalT interface

Post-Analysis Simulation In the Post-Analysis scenario, VizWalT reads a whole log file properly formatted, and afterwards it replays all the events supplied by this file. This log file is a text file extracted from a simulation of the nodes, and properly formatted to be able to be read by VizWalT. This working mode has been implemented to allow any person to use VizWalT without having the testbed that we have developed to work with WalT, if they already send the necessary information to use VizWalt, by formatting this log file a posteriori.

The information we need from the network to be able to replay are basically the following events:

Radio ON

Radio OFF

Transmission started + unique packet info

Transmission ended

Reception started

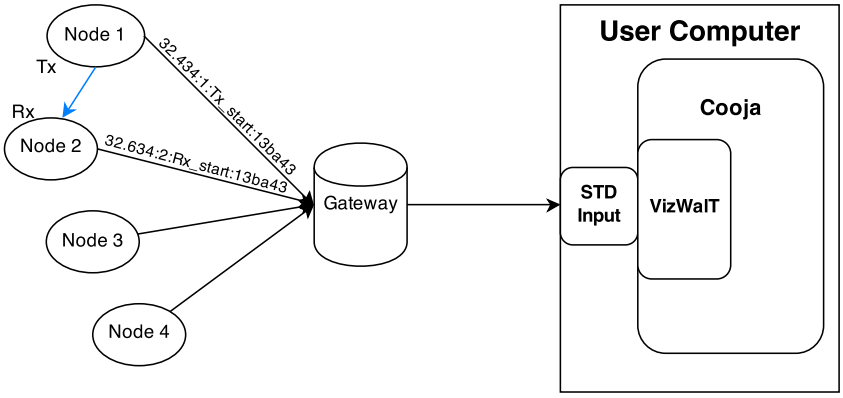

Figure 5: VizWalT real-time scheme

Reception ended + unique packet info

Position (Optional)

The unique packet info should be any data that identifies unequivocally a packet. It can be defined, for instance, as a hash of the payload of the packet, or a combination of the sender id, the source address and the sequence number. If all the information is already available from the network it is not necessary to make any modification to the sensor code to use VizWalT.

4.1.2 VizWalT Implementation

This section discusses in detail the implementation of VizWalT. The focus will be on how the VizWalT motes used in COOJA have been defined and the main modules we have implemented to add to the COOJA code, in order to be able to change how it works as well as on how VizWalT receives the information needed to replay the network.

COOJA plugin VizWalT, as we have already said, is defined as a COOJA plugin and it permits us to add some modules to COOJA and, without modifying any part of the COOJA original code, add that new functionality to this simulator. The main modules of VizWalT are the VizWalT plugin, the VizWalT RadioMedium, the VizWalT Controller, the VizWalT Mote and the Log line Manager.

VizWalT Plugin is the class that permit us to integrate VizWalT in COOJA. It is the user interface to VizWalT and it displays some messages, instructions, and when working in the Post-Analysis mode, it permits the user to select different log files to load and replay in VizWalT.

VizWalT RadioMedium is an extension of the basic COOJA AbstractRadioMedium class. VizWalT RadioMedium implements the function that creates the radio connections between the different motes and allows us to specify which should be the receivers of any sent packet.

VizWalT Controller is the main class of VizWalT. It controlls the VizWalT is simulation and perform the send of the packets between the different motes.

VizWalT Mote is a Java Mote that extends the AbstractApplicationMote COOJA's class. This customized mote type allows us to specify what to do when a packet is received or sent, and how and when VizWalT creates its motes.

Log Line Manager is the class that performs the reception of the packets from the standard input and schedule all the events that VizWalT Controller should manage.

Communication with VizWalT In the Post-Analysis working mode VizWalT reads directly from the input file, but in the Real-Time working mode is necessary to stablish a communication channel between the network and VizwalT. The communication with VizWalT is through its standard input. We have decided to use this method because it allows us to link processes just with a shell pipe (i.e. '|'), and thanks to that we obtain a really modular project. It also permits us to combine our modules with the standard tools like ssh. Particularly, this way we can communicate a process that is running in another machine, as a Raspberry Pi, with VizWalT by using ssh and we can visualize in VizWalT a remote experiment that is happening in another place.

```
i.e. ssh rpi() RPiControl.py | vizwalt.sh
```

Due to the fact that Cooja is launch with the ant tool, and ant cannot read directly from its standard input, we had a problem on how to send the logs from the Experiment Controller to VizWalT. To solve it, we have created the vizwalt.sh script, that creates a fifo file in which it sends

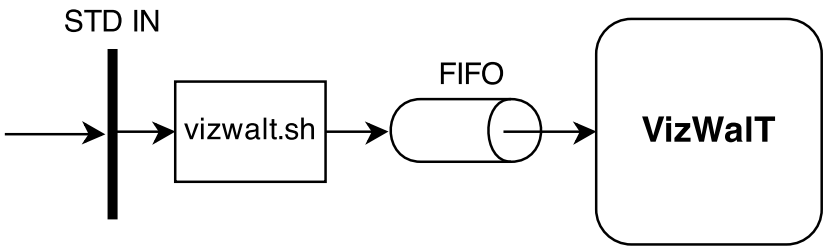

Figure 6: VizWalT input

the logs, and it launch VizWalT. Afterwards, VizWalT reads these logs directly in the fifo, solving this problem. See Figure 6.

4.1.3 Usage

Input definition The input of VizWalT should be always a log line properly formatted. This line can be obtained from the input log file in the Post-Analysis mode, or from the fifo file in the Real-Time mode. The format of the log line should be as follows: <time(s)>:<moteID>:<packet_type>(:<data>)(:<position>)
Where each field corresponds to:

Time: The event time in seconds.

SensorId: Number Id.

PacketType: It's the packet type which defines the event. There is 7 possibilities:

- RadioON
- RadioOFF
- Tx_start (+ sent packet data)
- Tx_end
- Rx_start
- Rx_end (+ received packet data)
- MoveTo (+ position)

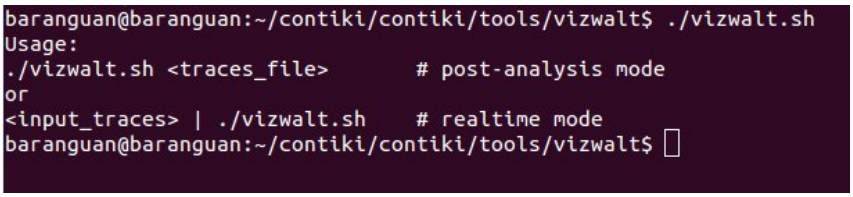

Figure 7: VizWalT script

Data: (If necessary) It is some data extracted from the packet and it is what determines who are the sender and the receiver/s of the packet. It is in hexadecimal format.

Position: (If necessary) It is defined as <x>,<y>,<z>, and it represents the position in the visualizer of cooja. Its usefull to represent a determined sensor distribution.

E.g.: 5.3335:5:Tx_start:33ef33babacd

In case of using the Post-Analysis mode the log file should have the same format explained above in all of its lines separated with a line break.

E.g.:
1.000:1:Tx_start:abac1384ba32
1.000:2:Rx_start
1.000:1:Tx_end
1.000:2:Rx_end:abac1384ba32

VizWalT script Our main purpose is to make VizWalT usage really simple, and to achieve this goal we have created a script, vizwalt.sh (See the figure 7). It defines two possibilities to launch the program depending on the working mode. To work in the Real-Time scenario this script should be launched as follows:

```
<input-program> | ./vizwalt.sh
```

The input program can be a script that gets the logs from the gateway or a server and prints them in the standard output. The vizwalt.sh script gets the lines from the pipe and retransmits the lines from its standard input to the fifo file, to have the lines available in the fifo, in order that

VizWalT could read them. In this working mode VizWalT schedules the events respecting the given timestamp, but there is the possibility to send to VizWalT a "ignore_timestamp" message, and after this message VizWalT ignores the timestamp and schedules the events 1 second after it reads.

To work in the Post-Analysis scenario the script should be launched as:

```
./vizwalt.sh <input-file.txt>
```

The input file should be formatted as it is explained in 4.1.3.

4.2 Experiment Controller

The experiment Controller module is responsible for controlling the simulation and it communicates VizWalT with the Raspberry Pi's. This module has been designed with the purpose of having a testbed composed by many Raspberry Pi's and some sensors attached to each of them. It is also in charge of sending the instructions to begin a simulation correctly. Its function is to ensure that each node is flashed and rebooted with the code we are using in the simulation. It sends the instructions to the RPis and waits until the sensor starts sending packets to VizWalT.

The Experiment Controller is also in charge of sending packets to VizWalT through the standard output. In the first design, these packets could have two formats a log string or a Google Protocol Buffer message, that was also defined in VizWalT. This second option made possible to send more complex packets with some extra information that can be added to the information extracted from sensor in the RPis or in the Experiment Controller, as the sender node ID, or mac addresses, and we thought that it was interesting tool in WalT.

However, once we saw the usefulness of the VizWalT plugin, we thought about proposing to the Contiki Community the integration of VizWalT in their project. But the possibility of using Google Protocol Buffer in the messaging format was implying a too much heavy dependance in order to apply to Contiki for the VizWalT integration, so we decided to remove this message formatting possibility between the Experiment Controller and VizWalT and to keep it between the other parts of WalT

platform.

4.2.1 Experiment protocol used

The protocol used by the experiment to start the simulation is the following:

1. Send a "Flash Sensor" message to RPis.
2. Receive the "Sensor Rebooted" message from each sensor.
3. Wait until all sensors have been rebooted correctly.
4. Send a "Platform Ready" message to VizWalT to notify that it is going to receive messages from sensors.
5. Send a "Start Sensor" message to RPis.
6. Start receiving packets from the sensors via RPis.
7. Format this packets properly before forwarding them if necessary (Google Protocol Buffer).
8. Send packets to VizWalT.

It communicates with the RPis through a zeromq socket, with a Publisher-Subscriber pattern to send the instructions to all RPis connected to Experiment Controller, and a Push-Pull pattern to receive the messages from RPis to the Experiment Controller. See figures 9 and 8 .

4.2.2 Module development

This module has been developed in Python and implements some functionalities to improve the performance of the simulation. A Mac Address Manager has been developed to be able to treat the information related to mac addresses. Each sensor sends a packet indicating which is its mac address. In the Mac Address Manager a mote ID is assigned to each sensor, and there is a table that matchs the sensor id while providing the mac address. In this way, we can receive the packets with the mac address, MacID packets, and translate it into mote ID to display it in VizWalT. Due to the possibility to use short mac addresses (2 Bytes) or long mac addresses (8 Bytes) in the procotol IEEE 802.15.4[3], sensors can send both of them to the Experiment Controller, and they will be

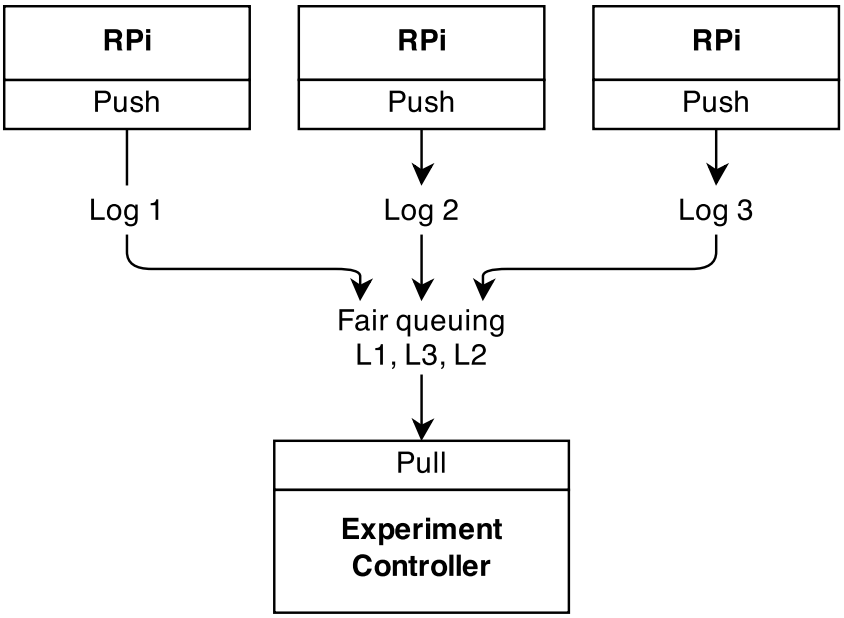

Figure 8: Scheme of pull-push pattern of zmq

stored, so a packet received with the sender mac address long or short will allow the Experiment Controller to know which is the sender mote ID, and to specify it in the message sent to VizWalT.

4.3 RPi Controller

The RPi Controller module is the module in charge of managing the sensors and of communicating the sensors with the Experiment Controler.

This module has been developed in Python and it will be compiled and charged on each RPi composing the testbed. The function of this module is basically to flash the sensors, loading the corresponding code to this sensors and to manage the messages extracted from the sensors to allow VizWalT to display the real performance of the network.

The main trouble we have found in the communication between the sensors and VizWalT is that the sensors are connected to RPis through a serial link, which is not really fast, so it is not possible to send large amounts of data through it without disturbing the normal behaviour of the sensors.

The solution applied to solve this problem is to send as little information as possible from the sensors to the RPis. To do that we have made a study of which is the essential information in each type of message and we have developed a Firmware Translator.

4.3.1 Firmware Translator

The Firmware Translator parses the packets providing this essential information in a packet with all the necessary information. The essential information we have fixed is the following:

- SensorRebooted: 'B:<mac>'
- MacId: 'I:<mac>'
- RadioOn: 'N'
- RadioOff: 'F'
- Tx_start: 'T:<packet_info>'
- Tx_end: 'E'
- Rx_start: 'R'
- Rx_end: 'X:packet_info'
- MoveTo: 'M:x:y'

We have considered this is the essential information to VizWalT because we consider that with this information we can know what is happening in the network as we can get the information of the duty cycle, when is the radio interface listening, sending or receiving a packet and, by looking for which sensor has received a packet with the same packet info as the transmitted packet, we can know which is the sender, and which the receiver or receivers of a packet.

4.3.2 Message format

To ensure the reliability of the source and destination of each packet we have defined this packet information as a combination of 3 mac fields: the sequence number, the source address and the frame control type. With this 3 mac fields, we have an unequivocal identifier for each packet

transmission. So we will add to the Tx_start and the Rx_end messages this packet information. The packet information format is:

$$4\ \mathrm{Bytes} = @\mathrm{shortmac}(2\mathrm{B}) + \mathrm{seq\_num}(1\mathrm{B}) + \mathrm{FC\_type}(1\mathrm{B})$$

As the RPis are connected to the sensors through a serial link, this link is constant, so once the sensor has sent its mac address, using the MacId message, the RPis can add the information related to the mac address of the sensor that has transmitted a message through the serial link, and as we mentioned above, the Experiment Controller will assign the mote ID to the packet.

Finally, the RPi controller is the responsible for adding a timestamp to the message in order to be able to represent the packets in the correct order. To get a good accuracy in the events time, the synchronization of all RPis is required. This synchronization is not trivial, so we let this improvement for the people following with this project. For the moment, we have two possibilities, to add a timestamp relative to the system time, or to add a 1.000 second time and activate the ignore_timestamp flag by sending a message requiring this functionality. If VizWalt works with this flag on, VizWalT replays the events when it receives the logs, without taking into account the timestamp of them.

Summing up, we will receive a little packet (from 1 Byte to 6 Bytes) and we will obtain a log with the essential information to replay the network properly.

For instance, we will receive a sequence and will translate it as follows (sender short @mac:0241, sequence number: 7a, FC type beacon: 00):

- 'N' $\longrightarrow$ '1.424:2:RadioOn'
- 'N' $\longrightarrow$ '1.425:1:RadioOn'
- 'T:02417a00' $\longrightarrow$ 1.427:1:Tx_start:02417a00'
- 'R' $\longrightarrow$ '1.428:2:Rx_start'
- 'E' $\longrightarrow$ '1.432:1:Tx_end'
- 'F' $\longrightarrow$ '1.433:1:RadioOff'
- 'X:02417a00' $\longrightarrow$ '1.434:2:Rx_end:02417a00'

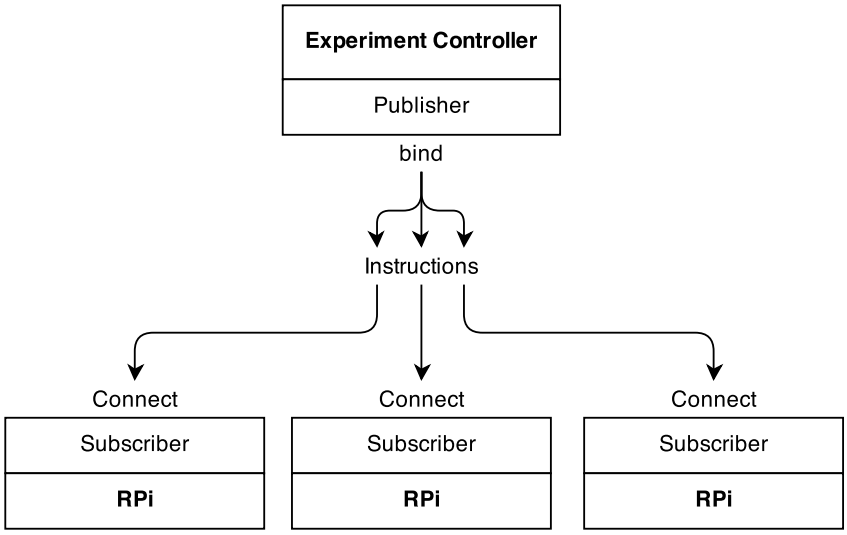

Figure 9: Scheme of publisher-subscriber pattern of zmq

- 'F' ⟶ '1.435:2:RadioOff'

The reception of this packet is through the serial link, and in order to send this formatted logs to the Experiment Controller we have used two zeromq sockets. One with the publisher-subscriber pattern to read from the experiment controller, and other with push-pull pattern to send the logs from the RPi Controller to the Experiment Controller. See figure 9 and figure 8 .

4.4 Sensor Modification

This module is the last module developed in this project and its goal is to introduce a modification to the sensor code to obtain the traces needed to generate the logs that allows VizWalT to replay what is happening in the network. This module is implemented in C, as the sensor we have used are developed using Contiki OS, whose programming language is C.

The objective is to make these modifications as least intrusive as possible, without modifying the behaviour of the sensors while they are performing any application. To make this possible we have decided to advantage of the **Energest** Contiki application. In order to know how

to obtain all the event information we need, we have studied the ContikiMAC protocol[4], and we will use the Energest tool to obtain them.

4.4.1 Using Energest

This Contiki application measures the time that the sensor is in the different states, and it has been developed to calculate the amount of energy spent by the sensors. We found that Energest fills almost all of our requirements because it calculates the time that the radio interface is listening to the radio medium and the amount of time that it is transmitting. The only information we miss is the time that the sensor is receiving a packet.In order to solve this lack, we have added one reception state to Energest.

The advantage of using Energest is that this application is already implemented in lots of sensor drivers, and it specifies when the sensor starts transmitting or when the sensor switchs on its radio interface to listen to the radio medium to know if a packet is being transmitted to itself. So we just had to add the energest calls in the radio driver to set when a packet is being received.

The final modification to the sensor code is to extract the traces from energest. To do that we added a printf() function to print the traces that needs to be formatted to include variables and send the packet information, and putchar() function to print the other traces, since putchar() function is much lighter than printf().

4.4.2 Traces information

To achieve a good debugging we have considered necessary not only to see that a packet has been sent or received, but also which type of packet it is. To obtain this information we have formatted the packet information we are providing to VizWalT as four bytes, one byte for the sequence number, 2 bytes to the short mac address and one byte for the packet type. We print this values as hexadecimal values, and we can get back the information in VizWalT to know the sequence number, the source address and the packet type.

To know the packet type we need to use the Frame Type field defined in the IEEE 802.15.4 MAC protocol [3], that defines 3 bits to distinguish

the 4 types of messages.

Frame type value	Description
000	Beacon
001	Data
010	Acknowledgment
011	MAC command

But to make easier the debugging process, we thought that if the frame type is a MAC command, it is interesting to know which type of MAC command frame the packet is. To make this we have added to this 4 packet types the information needed to know the 9 types of MAC command frames. This information is provided by the Command frame identifier field [3] in the MAC command frame.

Command frame identifier	Command name
0x01	Association request
0x02	Association response
0x03	Disassociation notification
0x04	Data request
0x05	PAN ID conflict notification
0x06	Orphan notification
0x07	Beacon request
0x08	Coordinator realignment
0x09	GTS request
0x13	DSME Association request
0x14	DSME Association response
0x15	DSME GTS request
0x16	DSME GTS reply
0x17	DSME GTS notify
0x18	DSME Information request
0x19	DSME Information reply
0x1a	DSME Beacon allocation notification
0x1b	DSME Beacon collision notification
0x1c	DSME link status report

In order to send all the information without having to send 2 Bytes, we have defined an only scale that let us know which is the Frame Type and, in case that it is a MAC command, which is the command name. We've shifted up the MAC command type 3 positions to join all in a single table, so the packet information format that we are sending VizWalT is the following:

Packet information field	**Type of packet**
0x00	Beacon
0x01	Data
0x02	Acknowledgment
0x03	MAC command
0x04	Association request
0x05	Association response
0x06	Disassociation notification
0x07	Data request
0x08	PAN ID conflict notification
0x09	Orphan notification
0x0a	Beacon request
0x0b	Coordinator realignment
0x0c	GTS request
0x16	DSME Association request
0x17	DSME Association response
0x18	DSME GTS request
0x19	DSME GTS reply
0x1a	DSME GTS notify
0x1b	DSME Information request
0x1c	DSME Information reply
0x1d	DSME Beacon allocation notification
0x1e	DSME Beacon collision notification
0x20	DSME link status report

For example, if a sensor is sending an association request, and this packet is the first packet, the traces that the sensor will send to the RPi will be:

T:01024c04 → 1.434:2:Tx_start:01024c04

E → 1.436:2:Tx_end

And VizWalT will display in Cooja:
00:01.434 ID:2 Mac command: Association request sent with packet id: 1.

5 Test implementation

5.1 First VizWalT test

To check that the different modules work properly, we have made some tests during the development of VizWalT. After finishing each module we have created a simple test to check if it works properly. Once we have had the VizWalT module, the Experiment Controller and the RPi Controller developed, we have created a Python script that sends the packet with the same format as the sensor does. And we have been able to visualize in Cooja all the events that we've introduced in the Python script, so we can say that we have checked that it works properly.

5.2 Final VizWalT test

To test the success of the VizWalT project, we have implemented a small testbed composed by 3 sensors ST harvesting nodes GreenNet V.2.1.1, and two Raspberry Pi's. In order to be able to develop this test we've implemented it in two steps. Firstly we have made some simple test with 2 sensors connected directly to a laptop and we tried to visualize the discussion between the 2 sensors. Secondly we've implemented the final testbed where we've introduced another sensor and the Raspberry Pis between the sensors and VizWalT.

5.2.1 Simple test

In this first simple test, we have introduced the sensor modification module in a simple example, and we have loaded it into the sensors. We have used the contiki example of one border-router and one leaf.

Figure 10: 2 Sensors in a discussion

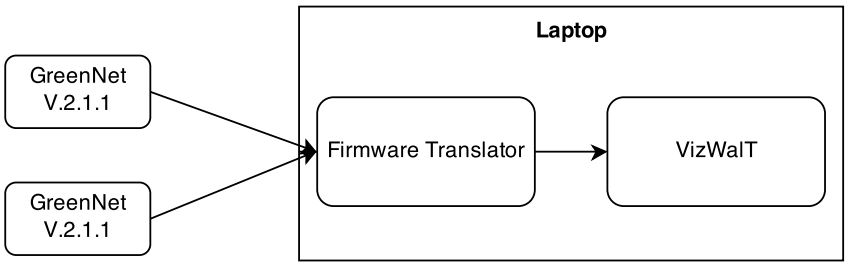

Figure 11: First test scheme

We have run the python code of the Firmware Translator in the laptop, and we read from the serial link the sensor traces, and we send the information to VizWalT through a pipe.

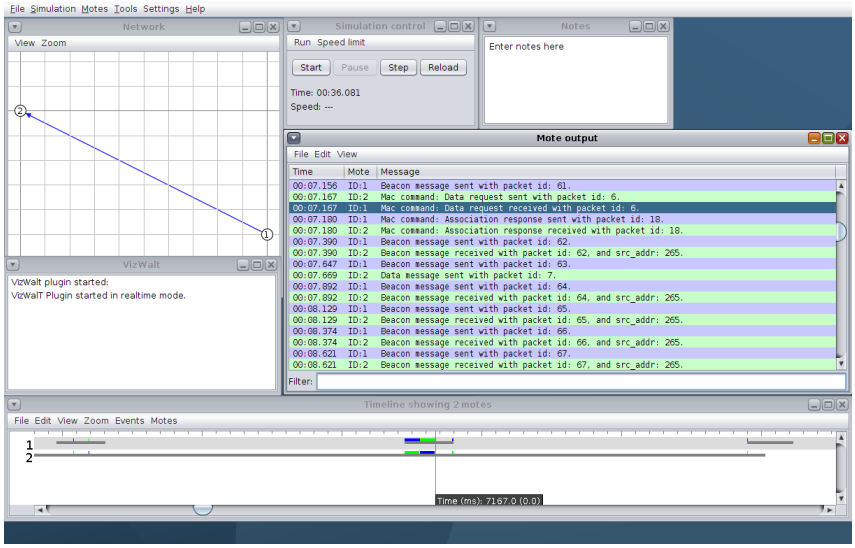

Figure 12: Screenshot of VizWalT

As we can see in figure 12, we have been able to visualize the discussion between the 2 sensors successfully.

5.2.2 Final Testbed

As we have said, in this final Testbed we use 3 Sensors, 2 Raspberry Pis and one Laptop where we run Cooja. See figure 13.

These 3 sensors are connected through the serial link to the different ports of the Raspberry Pis. And the Raspberry Pis are connected to a computer where VizWalT is running. The sensors will be flashed with the code from the Raspberry Pi. After being flashed, sensors start communicating between them, and they start to print the traces through their serial link to the Raspberry Pi. We will run the python RPiController program from the Experiment Controller using ssh. The Raspberry Pis will parse this traces, and they will create the properly formatted logs. This logs will be sent to the Experiment Controller where a python select module will manage the packet arrivals from the 2 Raspberry Pis. VizWalT will receive the messages by piping the standard output of the Experiment Controller program, to the standard input of the VizWalT running script.

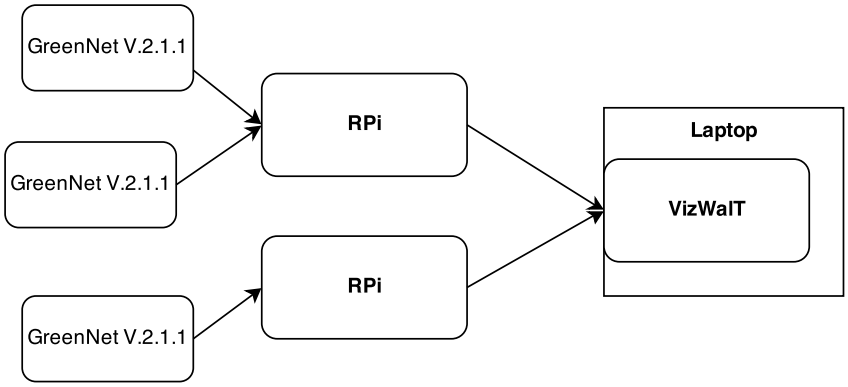

Figure 13: Final testbed scheme

Figure 14: Final testbed

Finally we can observe in the Cooja graphic interface what is happening in the network. We are focusing in two events, the association procedure (see fig 15), and the sensor association in the beaconed DSME Mac protocol. We can see the first one in the figure 16.

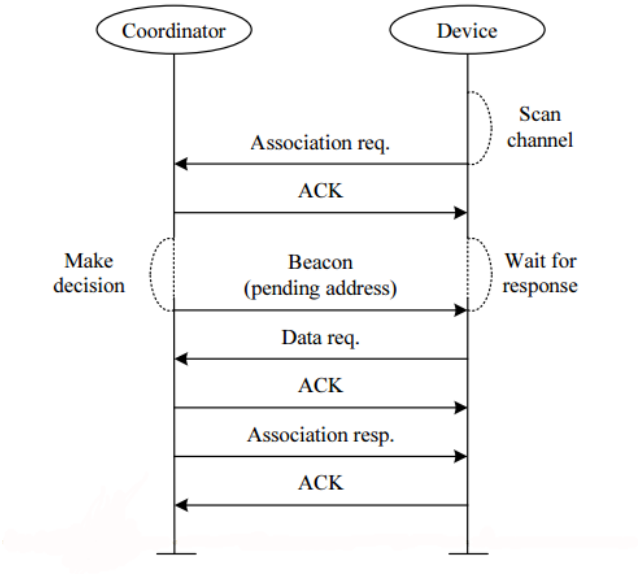

Figure 15: Association Request procedure

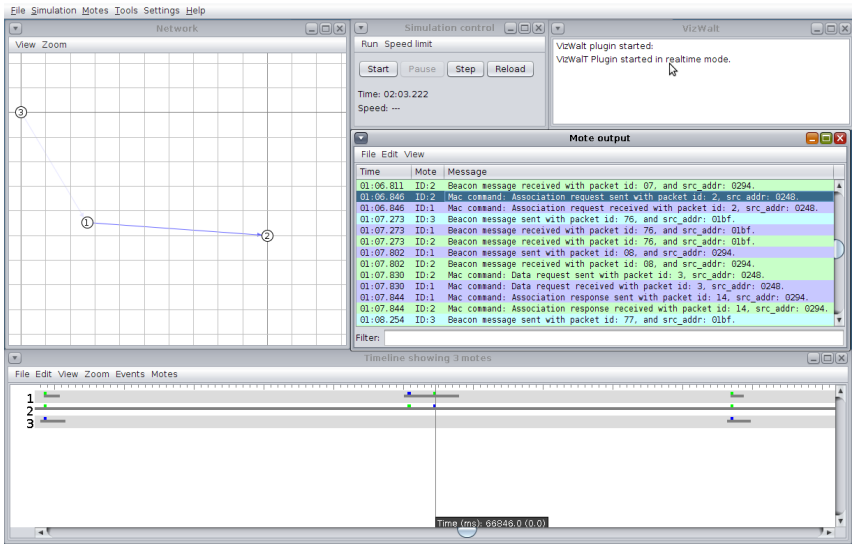

Figure 16: VizWalT screenshot of Association Request procedure

And we're able to see the some other things like how is the performance of the beacon enabled IEEE 802.15.4, for example we visualize the GTS allocation [5] in this protocol (see figure 17), and the data message sending between sensors(see figure 18).

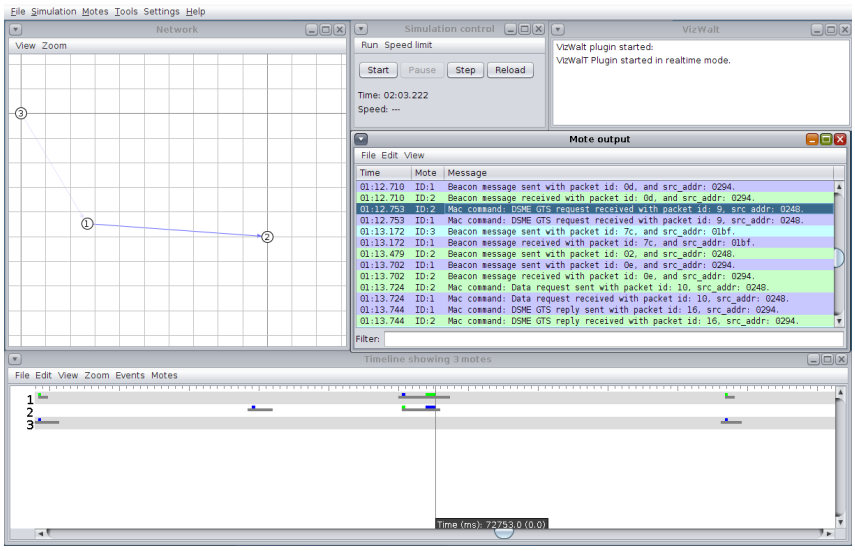

Figure 17: GTS allocation in beacon enabled IEEE 802.15.4

To implement this test, we have created a team with Joao Guilherme Zeni and Iacob Juc, in order to mix some parts of our work made under the WalT project, to create a most interesting test. Joao Guilherme Zeni is the person entrusted to continue this project. In the following step, he is in charge of implementing a Direct Memory Access method to be able to print the traces without interfering in the sensor CPU performance and manage the communication between the sensors and the RPi in a faster and more efficient way. Iacob Juc has worked in a protocol modification, that has been loaded in the sensors, and we have been able to observe this modification in an understandable way thanks to VizWalT.

A Workshop has been scheduled at ST Microelectronics in Crolles (France) on 7th April, and we have decided to make a demonstration

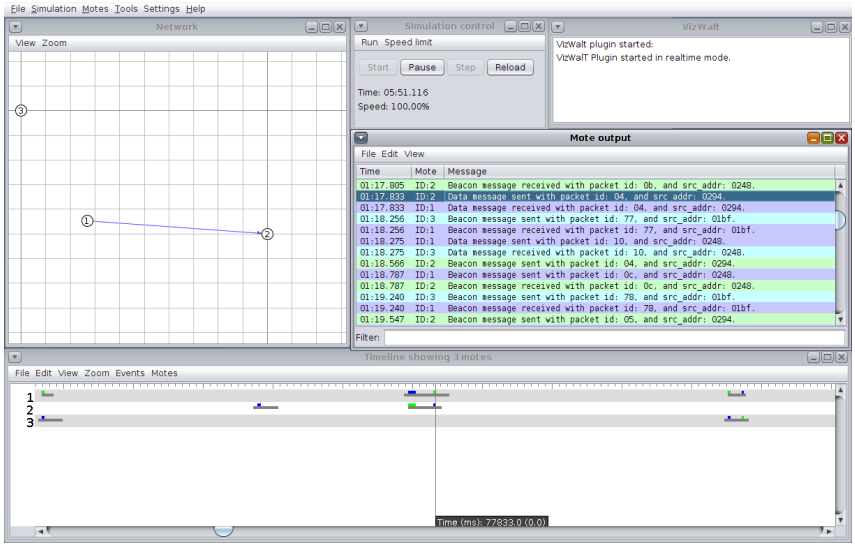

Figure 18: Data message exchange in beacon enabled IEEE 802.15.4

there. As we didn't have much time to incorporate all the parts developed by Joao, Iacob and me to this test, we have fit the different modules to be able to join them without a great complexity, but keeping the main structure. We have decided to implement the RPi Controller and the Experiment Controller in a lighter version in the Raspberry Pis and the computer on which VizWalT will run. Consequently, we don't have to use zeromq sockets to communicate between RPis and the Experiment Controller, and we are not going to use Google Protocol Buffer messages, because we read directly from the serial link, we parse the messages and we print it to the standard output where VizWalT will read them.

From my point of view, this demonstration is a good way to close a master thesis because we will have the possibility to show that this project is really useful.

6 Conclusion

The project subject on making easier the debug of Wireless Sensor Networks (WSN) is an interesting improvement in the development of WSN protocols and services.

The studying part of this subject has been very instructive because it has allowed me to deepen my understanding of the behaviour of this networks, and also to understand the difficulties when debugging WSN, due to the wireless nature of this technology and its still short life, resulting in the lack of developing tools that makes easier and more comfortable the development of WSN.

Since the main project, WalT, in which VizWalT was integrated, was in an initial phase, the project goal has been continuously modified, making that we have been adapting VizWalT to each moment requirements. This has caused that we have centered more in some objectives, and we have left apart some of the other initial objectives, as the compatibility with France Telecom analysis tools. Anyway, thanks to the modular structure of the developed project, this is not a closed door, because it is not very difficult to add a translator module from the VizWalT format to pcap.

The modular programming, using different languages like Python, Java or C, for the different modules, and the completely different purpose of each module has brought to me a wide range of issues from wich I learnt a lot while trying to solve them.

Working with Etienne Dublé and Franck Rousseau has taught me how to focus a problem, and try to find a solution, showing that a simple and smart solution is often much better than a complex solution with a lot of new and difficult technologies, and I think this will help me in the future.

7 Acknowledgments

First of all, I would like to acknowledge the ETSETB, for forming me during those hard five years, and the INP-Grenoble, and particularly the Laboratoire d'Informatique de Grenoble for hosting and mentoring me during those last 7 months.

I want to express my very great appreciation to my advisor Josep Paradells for agreeing to be my advisor and for solving all my doubts and guide me. Also to Professor Andrzej Duda, for offering me this great opportunity and hosting me in his research group; to Franck Rousseau and Etienne Dublé for showing me all their work, teaching me so many things on how to research and for being always so much willing to solve my doubts and help me with all the issues I found; and to all the group members for making me feel so comfortable working with them. Thank you all for your help.

References

[1] Fredrik Österlind, Adam Dunkels, Joakim Eriksson, Niclas Finne, Thiemo Voigt, Swedish Institute of Computer science, *Cross-Level Sensor Network Simulation with COOJA*. 2006.

[2] Hervé Martin, LIG Director, *http://www.liglab.fr/spip.php?article107*.

[3] IEEE Std 802.15.4-2011, *Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)*. 2011.

[4] Adam Dunkels, SICS, *The ContikiMAC Radio Duty Cycling Protocol*. December 2011.

[5] Pangun Park, Carlo Fischione, Karl Henrik Johansson, *Performance analysis of GTS allocation in Beacon Enabled IEEE 802.15.4*. June 2009.