upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa

Vrije Universiteit Brussel

MASTER THESIS

Experimental Evaluation of Communication Protocols for Wireless Sensor Networks

Author:
Maite BEZUNARTEA

Promoters:
Prof. Dr. Ir. Kris STEENHAUT
Prof. Dr. Ir. Jacques TIBERGHIEN
Supervisor:
Ir. NGUYEN Thanh Long

Academic Year 2013-2014

A thesis submitted in partial fulfillment of the requirements for the degree of Ingeniería de Telecomunicación.

Thesis realized within the framework of the Erasmus exchange program, between Vrije Universiteit Brussel and Universidad Pública de Navarra.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Abstract

To transform the Internet of Things from a dream to a reality, a lot more research in essential issues, such as performance, is needed. Performance research is often done on simulated networks but those can seldom emulate all the hard to predict effects that appear in real-world wireless communications. This is why in the present thesis some specific performance issues were studied experimentally on real radio networks. Different techniques to measure packet delivery ratios, packet latency and power usage were developed and tested with Zolertia Z1 motes running the Contiki operating system. With these techniques more insight into the functioning of radio duty cycling and medium access protocols was obtained which allowed a better understanding of the interactions of these protocols with the higher layers required to fully integrate things into the global Internet.

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Para conseguir que el Internet de las Cosas sea una realidad, es necesario seguir investigando acerca de los problemas esenciales, como el rendimiento de dichas redes. La investigación sobre el rendimiento suele hacerse basándose en redes de sensores simuladas, pero resulta muy difícil emular los efectos que aparecen en las comunicaciones radio reales. Por este motivo se estudian en la presente tesis algunos aspectos específicos del rendimiento de redes basadas en comunicaciones radio reales. Se han desarrollado y probado diferentes tecnicas para medir el ratio de paquetes recibidos, latencia de paquetes y consumo de potencia eléctrica, con los nodos Zolertia Z1 empleando el sistema operativo Contiki. Con estas técnicas, se ha profundizado en el conocimiento acerca del funcionamiento de los protocolos empleados en la gestión de la actividad radio y de los protocolos que ceden el acceso al medio compartido. De este modo se ha podido llegar a comprender mejor cuál es la interacción de las capas más bajas con las capas superiores, requeridas para integrar completamente de las cosas en la Internet global.

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Pour faire passer l'Internet des Objets du rêve à la réalité, beaucoup de recherche dans des domaines essentiels, tels que les performances des réseaux reste nécessaire. Les questions de performance sont souvent étudiées à l'aide de simulateurs mais ceux-ci ne peuvent en général pas émuler les comportements difficiles à prédire qui caractérisent souvent les réseaux sans fils. C'est pourquoi, dans ce travail de fin d'études, certains aspects des performances de réseau ont été étudiés à l'aide de vrais réseaux radio. Différentes techniques pour mesurer le pourcentage de paquets arrivés à destination, le temps mis pour atteindre la destination ainsi que la puissance électrique requise ont été développées et testées avec des capteurs sans fil Zolertia Z1 équipés du système d'exploitation Contiki. Ces techniques ont permis d'acquérir une meilleure compréhension des protocoles qui gèrent l'activité des radios et de ceux qui donnent accès à l'espace radio. Ceci a ensuite permis de mieux comprendre l'interaction entre ces protocoles et ceux de plus haut niveau nécessaires à une pleine intégration des objets dans l'Internet global.

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Om het «Internet of Things »van droom tot realiteit om te vormen is nog veel onderzoek in essentiële domeinen zoals prestatie vereist. Prestatie aspekten worden vaak door middel van simulaties bestudeerd maar deze blijken vaak onbekwaam, moeilijk te voorspellen fenomenen eigen aan radio netwerken, na te bootsen. Daarom werden in dit afstudeerwerk bepaalde prestatieaspekten experimenteel bestudeerd door middel van echte radio netwerken. Verschillende meettechnieken om het percentage succesvolle transmissies, de vertragingen en het energieverbruik te meten, werden ontwikkeld en uitgetest door middel van Zolertia Z1 sensor knopen uitgerust met het Contiki besturingssysteem. Deze meettechnieken hebben toegelaten beter inzicht te krijgen in de werking van de protocollen die het aan- en uitzetten van radio's bepalen en van deze die de toegang tot het communicatiemedium regelen. Dit heeft dan verder toegelaten de interacties die kunnen bestaan tussen deze protocollen en deze die, op hogere lagen, moeten instaan voor de ware integratie van "dingen" in het globale Internet, beter te verstaan.

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Key Words

Wireless Sensor Network, Contiki, RPL, Radio Duty Cycle, ContikiMAC, LPP, CXMAC, NullRDC, Packet Delivery Ratio, Power Usage, Latency

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Acknowledgements

This thesis was brought to a good end with the help of some people that I would like to thank.

First of all I would like to thank all the people in the ETRO department that have always helped me with patience and kindness. I would like to thank my promoter Prof. Dr. Ir. Kris Steenhaut for the opportunity to work on a thesis in such an interesting topic and as a part of a great team. She was always ready to answer all of my questions. I would also like to thank Ir. Nguyen Thanh Long for all the time he has dedicated to me with a great deal of enthusiasm. I would also like to give very special thanks to Prof. Dr. Ir. Jacques Tiberghien for his support, encouragement, guidance, help, and teachings. I had a very valuable and enjoyable experience working with him. Without his patience and help it would be impossible to have completed this master thesis.

I would also like to thank Universidad Pública de Navarra as well as the Erasmus exchange programme.

Finally, I would like to thank my parents and sister for their love, understanding, support and help throughout my entire academic career, specially during the past year.

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Contents

0 Introduction **1**
- 0.1 Problem Statement . . . 1
- 0.2 Aims and Objectives . . . 1
- 0.3 Thesis structure . . . 2

1 Background **3**
- 1.1 WSN Standards . . . 4
  - 1.1.1 Physical and Data Link Layer IEEE 802.15.4 . . . 5
  - 1.1.2 ZigBee . . . 6
  - 1.1.3 WirelessHART . . . 7
  - 1.1.4 6LoWPAN . . . 9
- 1.2 Zolertia Z1 motes . . . 9
- 1.3 Contiki . . . 12
  - 1.3.1 Introduction . . . 12
  - 1.3.2 Overall organization of Contiki . . . 13
  - 1.3.3 Event management in Contiki . . . 16
  - 1.3.4 Communications facilities in Contiki. . . . 18

2 Preliminary study of the Zolertia Z1 Motes **27**
- 2.1 Introduction . . . 27
  - 2.1.1 The experimental set-up . . . 28
  - 2.1.2 The radio traffic . . . 29
- 2.2 Sender and Receiver Calibration measurements . . . 29
  - 2.2.1 Sender power measurements . . . 29
  - 2.2.2 Receiver sensitivity measurements . . . 30
- 2.3 Range measurements . . . 31
- 2.4 Conclusions . . . 32

3 Experimental evaluation of message latency and power usage in WSNs **33**
- 3.1 Introduction . . . 33
- 3.2 Evaluation techniques . . . 34
  - 3.2.1 Packet Delivery Ratio (PDR) . . . 34
  - 3.2.2 Latency . . . 34

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

3.2.3 Power . . . 38
3.3 Experimental Validation . . . 39
3.3.1 The experimental set-up . . . 39
3.3.2 The radio traffic . . . 40
3.4 Quantitative Observations . . . 42
3.4.1 Comparison of latency measurement techniques . . . 42
3.4.2 Comparison of power measurement techniques . . . 43
3.4.3 Comparison of RDC protocols . . . 44
3.5 Still Unexplained Observations . . . 45
3.6 Conclusions . . . 47

4 Experimental Evaluation of the performance of a RPL network 49
4.1 Introduction . . . 49
4.1.1 The experimental set-up . . . 49
4.1.2 The radio traffic . . . 50
4.2 Evaluation Techniques . . . 52
4.2.1 Packet Delivery Ratio . . . 52
4.2.2 Latency . . . 53
4.2.3 Power Usage . . . 53
4.3 Quantitative Observations . . . 53
4.3.1 Packet Delivery Ratio . . . 53
4.3.2 Latency . . . 57
4.4 Qualitative Unexpected Observations . . . 58
4.5 Conclusions . . . 59

5 Dual Network 63
5.1 Introduction . . . 63
5.2 WSN as Data Logger . . . 63
5.2.1 Packet Delivery Ratio . . . 65
5.2.2 Latency . . . 65
5.2.3 Power Usage . . . 67

6 Conclusions and Future Work 71
6.1 Conclusions . . . 71
6.2 Future Work . . . 72

References 73

A Contiki Code 77
A.1 receiver-latency.c . . . 77
A.2 sender-latency.c . . . 79
A.3 receiver-observed.c . . . 81
A.4 sender-observed.c . . . 83

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

A.5 receiver-observer.c 86
A.6 sender-observer.c 88
A.7 recv-observer.c 91

B PhP Code **95**
B.1 analyze.php 95
B.2 line_ count.php 97
B.3 latency.php 97
B.4 find_ lost.php 98

C MatLab Code **99**
C.1 script.m 99
C.2 countLines.m 101

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Todos los derechos reservados
Eskubide guztiak erresalbatu dira

List of Figures

1.1 IEEE 802.15.4 and the ZigBee protocol stack 7
1.2 WirelesHART architecture and components 8
1.3 The position of 6LoWPAN in the IPv6 protocol stack 10
1.4 The Zolertia Z1 mote 10
1.5 Ports in the Zolertia Z1 mote 11
1.6 Pins in the North Port (JP1A) 12
1.7 Process states 17
1.8 Principle of operation of contikimac. 20
1.9 The benefits of phase locking in contikimac. 21
1.10 The Low Power Probing RDC protocol. 21
1.11 The XMAC RDC protocol. 22
1.12 Simplified Control-flow chart of the Contiki implementation of the CSMA MAC protocol. 23

2.1 A mote mounted on a wooden pole 28
2.2 The receiver mote. 28
2.3 Sender power measurements 30
2.4 Receiver sensitivity measurements 31
2.5 Scattergraph presenting the relation between RSSI and LQI for some 1000 received messages 32

3.1 Message Latency 34
3.2 Circuitry schematic 37
3.3 The data acquisition circuitry 38
3.4 The experimental setup 40
3.5 Frame detected by the sniffer. 40
3.6 Format of the packet sent 41
3.7 Interval between packets 41
3.8 PDR vs RSSI for all RDC protocols 46
3.9 PDR and Power Usage with ContikiMAC 8 47

4.1 Position of the motes. 50
4.2 Serialdump when connecting to the sink 51
4.3 The last packet is served before all T1 packets. 52

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

4.4 The operation of the buffer. 52
4.5 PDR in a RPL network. 16/01/2014 54
4.6 PDR in a RPL network. 17/03/2014 56
4.7 PDR vs HopCount in three RPL networks with 9, 16 and 23 motes 17/03/2014 60
4.8 Latency measured in a RPL Network. 61
4.9 Latency measured in a RPL Network with NullRDC. 61
4.10 Latency measured in a RPL Network with NullRDC after removing the points generated by malfunctioning motes. 62

5.1 Schema of the network 64
5.2 Schematic of the GPIOs used. 66
5.3 Simplified packet latency representation. 66
5.4 Measuring packet latency. 68
5.5 Connection diagram for measuring the power usage. 69
5.6 One of the dual motes built and tested. 69
5.7 Testing the dual-mote system. 70

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

List of Tables

1.1 Frequency bands defined by IEEE 802.15.4 5
1.2 Parameters defined by IEEE 802.15.4 for the physical layer . . 6

2.1 Sender power measurements 29
2.2 Receiver sensitivity measurements 30
2.3 RSSI and PDR as a function of distance 31

3.1 Latency measured with Cooja in ms 35
3.2 Latency measured with non-synchronized real-clocks in ms . . 36
3.3 RSSI and LQI 39
3.4 Latency measured with Z1 motes in ms 42
3.5 Comparison of latency measurements 43
3.6 Power coefficients per activity 43
3.7 Power usage by mote 127 44
3.8 Latency and power comparisons (low traffic) 45
3.9 Influence of the wakeup rate on performance 45

4.1 Comparison between the two measurement techniques for latency with a one-hop RPL network 57
4.2 Comparison between the two measurement techniques for latency with a two-hop RPL network 57

5.1 Characteristics of both networks. 64

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Todos los derechos reservados
Eskubide guztiak erresalbatu dira

List of Acronyms

6LoWPAN IPv6 over Low power Wireless Personal Area Networks.

CCA Clear Channel Assessment

CSMA Carrier Sense Multiple Access

CSMA/CA Carrier Sense Multiple Access with Collision Avoidance

CSMA/CD Carrier Sense Multiple Access with Collision Detection

FIFO First In, First Out

HART Highway Addressable Remote Transducer

IEFT Internet Engineering Task Force

IoT Internet of Things

LLN Low power and Lossy Networks

LQI Link Quality Indicator

LR-WPAN Low-Rate Wireless Personal Area Network

MAC Medium Access Control

MTU Maximum Transmission Unit

OF Objective Function

PDR Packet Delivery Ratio

PHY Physical layer

RDC Radio Duty Cycle

RPL Routing Protocol for Low-power and Lossy Networks

RSSI Received Signal Strength Indication

UIP micro IP

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

WFD WirelessHART Field Device

WPAN Wireless Personal Area Network

WSAN Wireless Sensor and Actor Networks

WSN Wireless Sensor Network

WUI Wake Up Interval

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Chapter 0

Introduction

0.1 Problem Statement

The "SmartNets" research group, part of Etro-Iris, under the leadership of prof. Kris Steenhaut explores various issues related to Wireless Sensor Networks and proposes improvements to certain protocols used in these networks. For evaluating the proposed improvements, certain performance indices need to be defined and evaluated, both for the original and for the improved protocols.

Often such evaluations are done on simulated networks, which eliminates the burden of setting up real networks. In the mentioned research group the COOJA simulator, which is part of the Contiki operating system, is commonly used for testing protocols. However, due to the crude approximations used for modelling radio links, the behaviour of simulated networks differs often significantly from what can be observed on real networked radios.

Simulated radio networks are notorious for their inaccuracy [24] and some conferences, such as the realWSN (conference held in Como in September 2013), require that reported results should be obtained by experiments done on real networks rather than simulated ones.

This explains why for my Master Thesis I should set up an experimental radio network and develop measurement techniques to verify results obtained by PhD students by means of simulations.

0.2 Aims and Objectives

The purpose of this thesis is to implement different techniques for measuring the Packet Delivery Ratio (PDR), latency and power usage for real-world Wireless Sensor Networks (WSNs). Those measurements will be made with different underlying Medium Access Control (MAC) and Radio Duty Cycle (RDC) protocols. Once those techniques have been developed and validated,

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

they should be implemented in a real network, in order to study experimentally which is the performance of the IPv6 Routing Protocol for Low-power and lossy (RPL) networks using Contiki in open space.

0.3 Thesis structure

After this introduction, Chapter 1 will give a profound background study on different protocols used in wireless sensor networks, the Zolertia Z1 motes and the Contiki operating system.

After getting acquainted with the necessary basic concepts, a bottom-up approach was followed. Firstly, an initial exploration of the Z1 motes was performed (Chapter 2), by making some simple experiments.

The 3th chapter will try to evaluate some interesting parameters in a WSN, such as message latency, packet delivery ratio and power usage, by developing different measurement techniques. In the 4th chapter, all proposed measurement techniques will be applied to the RPL network under study. In Chapter 5 a new setup, based on the experience acquired with the previous measurements will be designed. Finally, the general conclusions are presented as well as the future work.

The source code of the different softwares used for the motes and for analysing the collected data is available in the appendixes.

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

Chapter 1

Background

Collecting data becomes indispensable in a large number of applications, such as environmental monitoring for the study of the conditions of a terrain, or the military applications for the control of equipments or troupes. In the past, human operators or wired equipment were used to collect the necessary data. The two options presented certain problems: the measurements made by human operators were costly in terms of time, which would restrict the accessibility to certain locations. On the other hand, the use of wired sensors forces to use large quantities of cable and would also restrict the accessibility to difficult areas.

A Wireless Sensor Network (WSN) is a set of autonomous devices randomly distributed in a certain area that allow us to monitorize any environment no matter how inaccessible or recondite it may seem. WSNs can measure certain parameters in that area, providing us with valuable information to act in consequence, activating or deactivating different types of system. Depending on the desired application, the type of sensor will be different. Normally, in each mote several kinds of sensors are included. Typically, each mote includes a low-end 16 bit microprocessor (with a few kilobytes of RAM memory and tens of kilobytes of flash memory), temperature and acceleration sensors and interfaces for more specialized analog or digital sensors, a low-power integrated radio device and a battery.

With such characteristics, the applications for those networks could be endless in any ambit and commercial sector. In the last few years, a big development in WSN systems applied in the urban environment has occurred. Some of those applications are: noise control, creation of smart streets and avenues capable of detecting the traffic through them or the parking spots available or pollution control. Having all that real-time information can help public services to canalize traffic so that no traffic-jams are formed and citizens to save time and fuel. Other feasible applications are those such as resources control, that allow to know the exact quantity of energy or water that is being consumed; security applications and emergencies; or even monitorization of the state of people by means of biometric sensors to release

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

hospital resources.

It is obvious that for each one of the mentioned applications, the choice of the sensors is fundamental. However, some other factors should be taken into account when it comes to designing the network, such as the mechanical and electrical characteristics of the motes, the data transmission towards a data center and the availability of software tools for the processing of the data obtained by the network in order to make appropriate decisions. In addition, the nodes in WSNs are restricted in their resources:

- Constrained energy resources: the nodes are usually battery powered. Power consumption is critically important in WSN because it is directly related to the lifetime of the network. The nodes can only automatically operate during a limited period of time. In other words, they will operate mostly in a sleep mode to maintain power consumption low. Energy consumption can be due to three factors: the type of sensors, the transmission and reception of data.

- Limited processing capability

- Limited communication bandwidth

- Real-time requirements: depending on the application, it may be necessary that the nodes in a WSN can send the information as soon as possible

- Failure tolerance and reliability: the sensor nodes may stop working due to a failure in the feeding or electronics. However, the failure of one or more of the motes should not lead to a general failure in the network

Generally, in this type of applications WSNs are designed in a way that they have a double layer: an inner layer, that communicates the sensors between each other, and an external layer that communicates the network with the data server. The objective of these networks is the reliable real-time transmission of the frames recorded by the network. Each one of the mentioned layers may use different protocols. However, this also depends on the application. Some real-time sensor environmental data stays within the local Low-power and Lossy Network (LLN) environment, while for example alarms and other event data will percolate to higher layers.

1.1 WSN Standards

In traditional networks, both routers and nodes are mains-powered and the links connecting them are highly stable. Wireless smart sensors, as it was

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Frequency Band	Regions	Channels	Data Rate
2.4-2.4835 GHz	All countries	16	250 Kbps
902-928 MHz	North America	10	40 Kbps
868-868.6 MHz	Europe	1	20 Kbps

Table 1.1: Frequency bands defined by IEEE 802.15.4

mentioned earlier have a very restricted power source. Hence, the communication requirements for WSNs are very different from traditional IP networks, especially since the sensors may also have to act as routers. Some adaptations are thus necessary in order to make it possible to connect sensors to the internet, and these adaptations will be discussed in the following sections.

1.1.1 Physical and Data Link Layer IEEE 802.15.4

A communication protocol must take into account the previous requirements. The majority of communications protocols for WSNs are based on the standard IEEE 802.15.4, that only defines the physical and data link layer.

The standard IEEE 802.15.4 is a subgroup inside the IEEE 802 task group, specialized in Wireless Personal Area Networks, WPAN. Specifically, the task group 805.15.4 treats the necessities of systems with low data transmission capacity but with long life time requirements but with limited energy supplies and a low complexity.

Among the most important characteristics of this protocol are the flexibility of the network, low cost and low energy consumption. IEEE 802.15.4 specifies both physical and MAC (Medium Access Control) layers. However, depending on the requirements in different scenarios, both could be altered to guarantee certain requirements.

The frequencies defined in the standard are not the same for all regions in the world. In 1.1 the frequency bands for each region of the world are shown.

The use of the frequencies responds of legal requirements of the use of the radio-electrical spectrum in each country or region. However, most of the commercial equipments use the unlicensed band of 2.4 GHz, given that in that band more channels can be used and its usage is allowed in every region in the world.

The standard also specifies some other parameters in the physical layer, shown in table 1.2.

The Media Access Control (MAC) layer is the responsible of controlling the access to the communication media, given that it is a media shared by different users and equipments. In this case, the mechanism Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) is employed. Carrier

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

	2.4 GHz	868 MHz	915 MHz
Sensitivity	-85 dBm	-92 dBm	-92 dBm
Output power (mW)	1	1	1
Number of channels	16	1	10
Data Rate (Kbps)	250	20	40
Symbol rate (Kbaud)	62.5	20	40
Modulation	O-QPSK	BPSK	BPSK
Maximum packet size (bytes)	127		

Table 1.2: Parameters defined by IEEE 802.15.4 for the physical layer

Sense refers to nodes listening to the channel for a specific amount of time to assess the activity on the wireless channel. Basically, a node first listens to the channel; if the channel is idle, the node may transmit immediately. Otherwise, the node defers transmission and continues to monitor the channel until the transmission is over.

On top of the IEEE 802.15.4 standard, several standard bodies have been formed to proliferate the development of low-power networks in various areas. The purpose of those protocols is to offer a complete solution for Low-Rate Wireless Personal Area Network (LR-WPAN), building the upper layers in the protocol stack that are not included in IEEE 802.15.4. Next, three major standardization efforts will be described in detail: namely, ZigBee, WirelessHART, and 6LoWPAN. The standard 802.15.4 was developed independently from those standards, thus it is possible to build a WPAN based only in the standard 802.15.4 without implementing layers from other protocols. However, in that case the user should develop his own network applications.

1.1.2 ZigBee

The ZigBee Alliance is an international, non-profit industrial consortium of semiconductor manufacturers and technology providers, that has developed the ZigBee standard. This alliance has defined an open, cost-effective and efficient protocol that supports low data rates, low power consumption, security and reliability through WPANs.

As shown in figure 1.1, each standard defines specific layers of the protocol stack. The Network Layer and Application framework have been defined by Zigbee specifically on top of the standard IEEE 802.15.4, which defines they PHY and MAC layers.

Five main application areas are targeted: home automation, smart energy, building automation, telecommunication services and personal health care. To accommodate this large variety of applications, three types of traffic are defined:

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

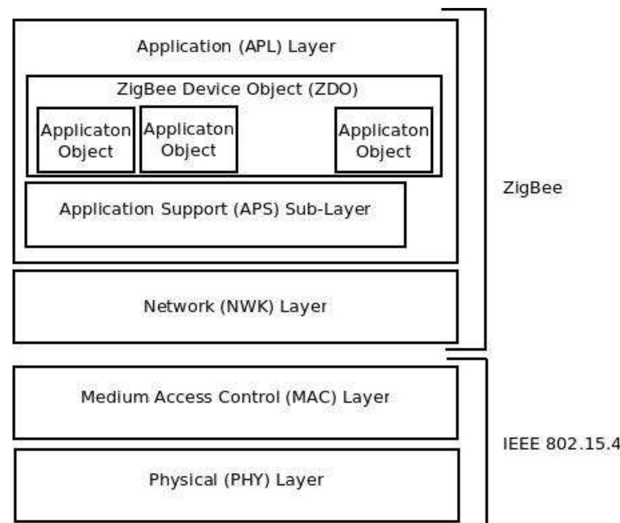

Figure 1.1: IEEE 802.15.4 and the ZigBee protocol stack

1. Periodic data traffic: for monitoring applications, where sensors provide continuous information.

2. Intermittent data traffic: for most event-based applications, that can be triggered by the application or an external factor.

3. Repetitive low-latency data traffic: for certain communications such as mouse click that needs to be completed within a certain time.

Now, however, the ZigBee alliance has also reached the conclusion that IPv6 is the way to go. This has resulted in the ZigBee IP Specification (ZIP). ZIP uses IPv6 over Low power Wireless Personal Area Networks (6LoWPAN), RPL and other protocols defined by the IETF.

1.1.3 WirelessHART

WirelessHART has been developed as a wireless extension to the industry standard Highway Addressable Remote Transducer (HART) protocol, which is the most used protocol in the automation and industrial applications that require real-time support. WirelessHART adds wireless capabilities to the HART protocol, relying on the IEEE 802.15.4 PHY layer standard for the 2.4GHz band while maintaining compatibility with existing HART devices, commands and tools.

The network architecture of the WirelessHART standard is shown in 1.2.

Each WirelessHART network defines three main elements:

upna
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

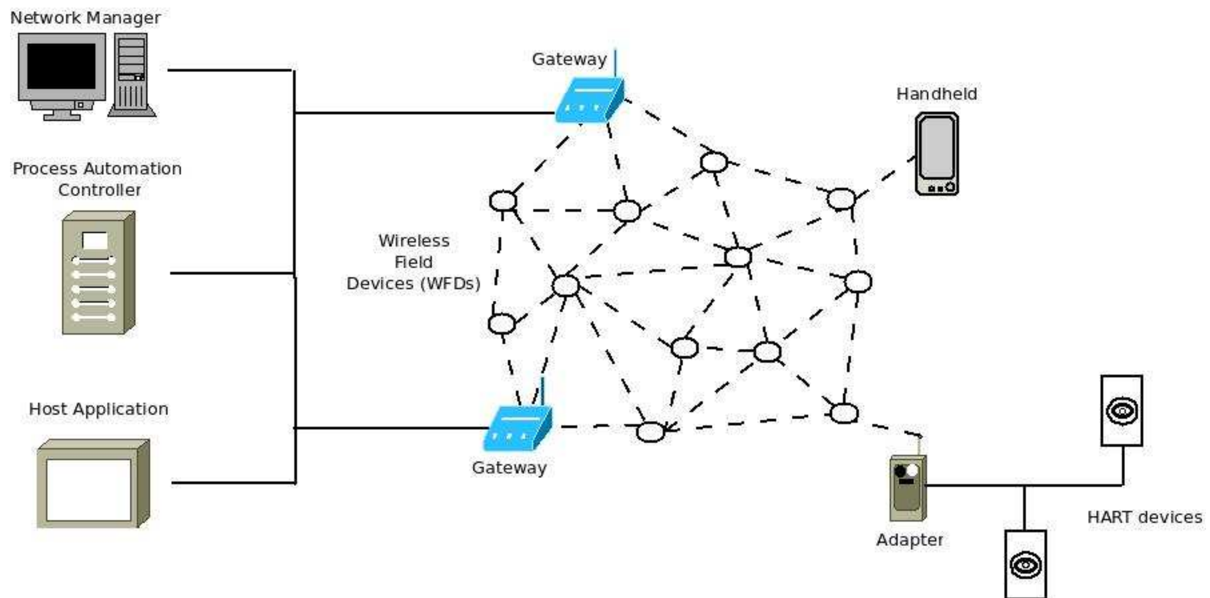

Figure 1.2: WirelesHART architecture and components

- WirelessHART Field Devices (WFDs): they are the sensor and control elements connected to process or plant equipment. These devices could be a device with WirelessHART built in or an existing HART-enabled device with a WirelessHART adapter.

- Gateways: they enable communication between these devices and host applications connected to a high-speed backbone or other existing plant communications network.

- A network Manager: responsible for configuring the network, scheduling communications between devices, managing message routes, and monitoring network health. It can be integrated into the gateway, host application or process automation controller.

In addition to the three main components, in a WirelessHART network handhelds can be used to provide on-site access to the wireless network and interface with the WFDs.

The network topology is designed as a mesh network and each device can act as a source or a router in the network. At the network layer, table-based routing is used so that multiple redundant paths are established during network formation and these paths are continuously verified. In addition, source routing techniques are used to establish ad-hoc communication paths.

The transport layer provides reliability on the end-to-end path. End-to-end monitoring and control of the network are also provided. Accordingly, WFDs continuously broadcast statistics related to their communication suc-

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

cess and neighbors, which are monitored by the network manager to establish redundant routes and improve energy efficiency.

Finally, the application layer supports the standard HART application layer, where existing solutions can be implemented.

1.1.4 6LoWPAN

Since the mentioned protocols based on IEEE 802.15.4 are not compliant with IP, those WSNs cannot be easily integrated with the Internet. Therefore, sensors cannot easily communicate with web-based devices, servers or browsers. Instead, gateways are required to collect the information from the WSN and communicate with the Internet.

However, IPv6 possesses some key features, such as universality, extensibility and stability, that may become the de facto solution for future Internet technology. To integrate WSNs with the Internet, the Internet Engineering Task Force (IEFT) has developed the IPv6 over Low-power Wireless Personal Area Network (6LoWPAN) standard. Specifically, the 6LoWPAN protocol discusses how to apply the IPv6 stack on top of the MAC and PHY layers of IEEE 802.5.4 to let any device be accessible from and with the Internet.

There are two basic challenges integrating IPv6 and WSNs. The first one is the addressing structure of IPv6, which defines a header and address information field of 40 bytes. However, IEEE 802.15.4 allows up to 127 bytes for the whole packet including header and payload information, which makes straightforward integration not efficient. On the other hand, since the minimum value of Maximum Transmission Unit (MTU) specified by IPv6 is 1280 bytes, the data link layer must fragment and reassemble data packets.

To address this issues, 6LoWPAN serves as an adaptation layer that lets the radio stack and IPv6 communications operate together. This adaptation layer is situated right above the Data Link Layer to segment the IPv6 packet into the small pieces required by the lower layer. Moreover, 6LoWPAN specifies header compression for the IP header in order to reduce the overhead of IPv6. The position of 6LoWPAN in the IPv6 protocol stack is shown in figure 1.3.

1.2 Zolertia Z1 motes

A wide variety of platforms have been developed in recent years, such as Mica, Tmote, Sentilla, Jennic, Iris and Zolertia.

The Z1 module is a general purpose development platform for WSNs. It is equipped with two on board digital sensors: a programmable accelerometer (ADXL345) and a temperature sensor (TMP102). Many other sensors can be connected. It also supports some of the most employed open source operatin systems for WSN, such as TinyOS or Contiki.

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

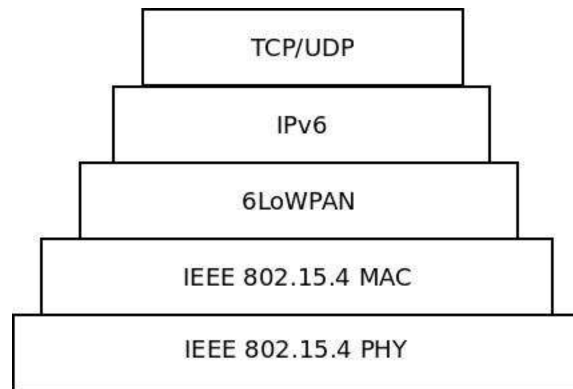

Figure 1.3: The position of 6LoWPAN in the IPv6 protocol stack

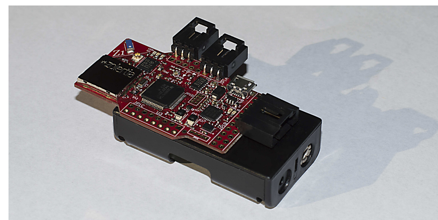

Figure 1.4: The Zolertia Z1 mote

The Z1 platform is equipped with a second-generation MSP430F2617 low-power micro-controller, which features a powerful 16-bit RISC CPU 16MHz clock speed, built-in clock factory calibration, 8KB Ram and a 92KB flash memory. At radio level, it includes the well-known CC2420 transceiver, IEEE 802.15.4 compliant, which operates at 2.4GHz with an effective data rate of 250Kbps.

The MSP430 micro-controller uses a limited number of GPIO (General Purpose Input-Output) hardware pins that are assignable to several functions depending on the user's needs , that can be independently programmable. The setup and operation of the digital I/O is configured with user software. The pins are organized into ports, in the case of the MSP430F2617 four ports are encountered, each one containing pins that can be used as GPIOs (see Figure 1.5). Two of the ports (P1 and P2) have also interrupt capability. More information can be found in the Zolertia Z1 Datasheet ([1]).

An example on how to set-up and work with this GPIOs pins can be found in Appendix A.

Another of the interesting features of the MSP430 micro-controller is the ADC. Such peripheral device allows us to make analog measurements with the MSP430 integrated in the motes, such as voltage measurements. In the particular case of the MSP430F2617 controller, it can perform A/D

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

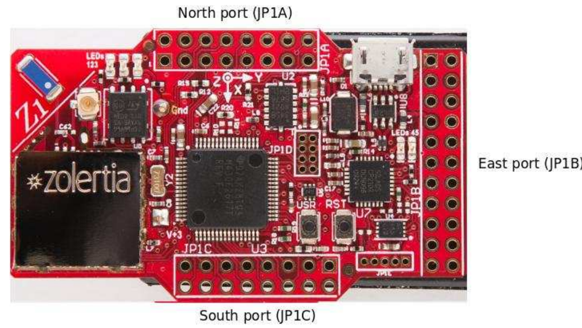

Figure 1.5: Ports in the Zolertia Z1 mote

conversion using 12 bits. Hence the binary number representing the result of a conversion can vary over the range 0 to $2^{12} - 1$, i.e., $4096 - 1 = 4095$.

In general,

$$n = \frac{V_{IN} - V_{REF_-}}{V_{REF_+} - V_{REF_-}} \times 4095$$

being n the decimal number obtained after sampling the analog signal.

Once the value n has been obtained, the conversion can be undone by following the formula:

$$V_{IN} = n \times \frac{V_{REF_+} - V_{REF_-}}{4095}$$

Supposing all voltages are referenced to $V_{SS} = 0V$ (ground), if a voltage being sampled has value V_{IN} and the reference voltage for conversion is V_{REF}, the decimal number representing that input will then become:

$$n = \frac{V_{IN}}{V_{REF}} \times 4095$$

and the conversion can be undone by the expression:

$$V_{IN} = n \times \frac{V_{REF}}{4095}$$

In order to access to the results of the conversion, there are 16 ADC12MEMx conversion memory registers to store conversion results. Furthermore, the ADC12 can operate under four different modes:

- Single channel single-conversion: A single channel is sampled and converted once. The ADC result is written to the corresponding ADC12MEMx.
- Sequence-of-channels: A sequence of channels is sampled and converted once. The ADC results are written to the conversion memories starting with the defined ADCMEMx.

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

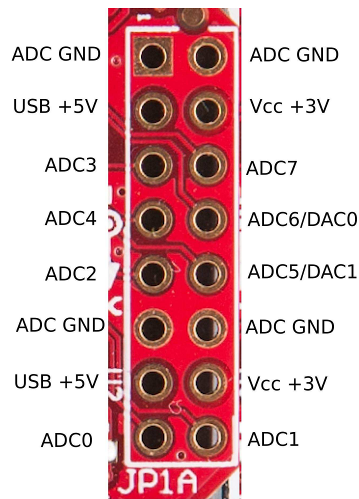

Figure 1.6: Pins in the North Port (JP1A)

- Repeat-single-channel: A single channel is sampled and converted continuously. The ADC results are written to the ADC12MEMx defined by the user.

- Repeat-sequence-of-channels: A sequence of channels is sampled and converted repeatedly. The ADC results are written to the conversion memories starting with the defined ADC12MEMx.

Most of the ADC12 pins from the MSP430 can be found in the North Port (JP1A) of the Z1 mote (see Figure 1.6).

Further information about the ADC and other features of the MSP430 can be found in [2].

1.3 Contiki

1.3.1 Introduction

When considering a wireless sensor network, the application should be the main goal and most of the resources for development should be dedicated to it. However the implementation of almost any application requires a lot of software that is not specific to any particular application and that requires a lot of programming and debugging efforts. For this reason, most wireless sensor network developers prefer to find means to share these efforts among many applications. A first possible approach consists in buying such software, typically from companies that sell so called "real time operating systems". These systems are however targeted at hardware with much more resources

upna
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

than the typical WSN motes, and the proprietary character of the software makes it difficult to discard the resource hungry but not really needed parts.

Fortunately, an alternative exists: some pioneers in the fields of WSNs have understood the need for an easily adaptable operating system for severely resource constrained systems. They also perceived that only the "open source" approach could provide the critical mass of (occasional) developers to maintain such an operating system and to adapt it to the continuously evolving hardware and applications. Two such operating systems gained considerable visibility and are widely used today. Chronologically the first one is TinyOS [3], initially developed at the University of California at Berkeley by David Culler et al. as early as 1999. Contiki [4], developed at the Swedish Institute for Computer Science by Adam Dunkels et al. became widely available in 2002. These operating systems pursue similar goals, but differ in their underlying paradigms (non-pre-emptible, but interruptible tasks in TinyOS, protothreads in Contiki) and in their programming language (nesc for TinyOS, plain C in Contiki).

The SmartNets group at the VUB which is exploring wireless sensor networks uses almost exclusively Contiki. Therefore the remainder of this chapter will entirely be devoted to Contiki. The reader should however understand that our preference for Contiki is purely circumstantial and that we did not compare the performance of these two very popular systems.

1.3.2 Overall organization of Contiki

Basically, Contiki is a huge collection of macros written in plain C and a set of makefiles allowing to assemble precisely these parts of Contiki required for a specific application and to compile and link them into an object file that can be loaded in motes. Contiki comes also with a collection of software tools, such as the COOJA simulator, written in Java. The successive versions of the entire source code are available through Github.

We observed that, sometimes, minor code changes do not result in new version numbers, with, as a consequence, that differences might exist between successive downloads of the same version of Contiki, which, obviously, complicates debugging!

The different parts of Contiki are grouped in separate directories, so that it is relatively easy to find the pieces of code that need to be tailored to a specific application. This grouping has been significantly changed between versions 2.6 and 2.7, in order to streamline the structure which had become quite intricate due to more than 10 years of additions of new functionalities. However, as most of the work reported in this thesis has been done with version 2.6, the older structure will be described here.

At the top level of the contiki directory one finds following directories:

platform : this directory contains mainly specific sub-directories associated with

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

the different motes and small computers for which contiki has been configured. Each sub-directory contains a specific main program (such as contiki-z1-main.c) for running contiki on the corresponding device. In these main programs hardware addresses are specified, subsystems such as serial I/O and timers are configured and initialized and the process scheduler is started. It contains also a device specific configuration file (named *contiki-conf.h*) that can be edited to select the software to be included in contiki. It allows, for instance, to choose among different MAC protocols or between IPv4 and IPv6. Finally, it can also contain directories with specific applications developed for the device. It is noteworthy that the COOJA simulator is included among the available platforms, so that simulating systems simply consist in running the unchanged software on the COOJA platform rather than on a specific device. Of course, if some settings are different on the COOJA platform definition, the impact of these settings on the behavior of the simulated system cannot be explored by simulation.

cpu : The sub-directories of this directory contain, for the different cpu's, software that is specific for that cpu, regardless of the platform. Examples are functions to access flash memory, to put the cpu in low power (sleeping) mode or to communicate, at the bit level, with popular radio chips.

core : this directory contains the most essential parts of contiki in separate sub-directories:

- sys contains all functions responsible for the management of processes, interrupts and timers.
- dev is a set of header files and drivers for input/output devices commonly used in motes and small embedded systems.
- net groups most of the software related to communications between motes. It contains three subdirectories and a large number of header and program files. The three subdirectories are
- mac which groups all programs that belong to the Medium Access Control and Radio Duty Cycle layers.
- rime which contains a simple set of application programs for data unicast and broadcast communications as well as data collection and distribution in a multihop network.
- rpl which provides an implementation of the rpl routing protocol for the Internet of Things. The other header and program files mainly implement, on one hand, inter-layer data structures and buffer management functions and, on the other, a TCP/IP stack.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

- cfs contains the contiki file system, largely inspired by the Linux file system.
- ctk provides a graphical users interface for contiki.
- lib is a collection of library functions for diverse applications, it contains integer Fast Fourier functions, various data encoding functions, checksum and cyclic redundancy check calculations, functions to build data structures such as linear lists and rings, random number generators, trickle timers, etc.
- loader provides a relocating linker-loader for object files in the Executable Linkable Format (ELF) used in Linux. This loader is built in two parts, one independent from the CPU and one tailored for the specific CPUs. In contrast with other cpu specific pieces of software, the different versions of the loader are part of the loader subdirectory rather than of the cpu top-level directory. The loader subdirectory contains also building blocks for a contiki dynamic linker-loader.

In addition to the contents of the listed subdirectories, the core directory contains some default configuration files with comments explaining how to modify them.

- apps : this directory groups three different kinds of application software. The first and most numerous are programs developed when contiki was being used as operating system for very small (often 8 bit) networked personal computers and their servers. Among them one finds typical desktop components such as programs to display process lists or directories and even a calculator. A quite complete set of internet applications such as telnet, ftp, dhcp, webbrowsers and webservers can also be considered part of this first category. In the much more recent second and third categories, one finds implementations of current developments in wireless sensor networks such as antelope [5] (a database) and erbium [6] (an alternative to the http web protocol for resource restricted systems) and also tools for debugging and optimizing applications. The programs ping6, powertrace and unit-test belong to this last category. Even if the original goals leading to the first category of applications is now quite outdated, some of these programs can be quite useful building blocks when developing new wireless sensor network applications.
- tools : this directory contains software tools written in Java and in Perl to simulate Wireless Sensor Networks (COOJA), display the behavior of a RPL network (Collect-View), insert and extract data in a contiki database (Coffee) and load operating systems in older personal computers.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

examples : The programs included in this directory, ranging from very simple (*"hello world"*) to quite complex (an IPv6 webserver for instance) show how to start with actual real or simulated devices and how to use the programs contained in the apps directory.

projects : This is the directory intended to contain the projects developed by users. Some project examples are already included in the distributed version of contiki.

1.3.3 Event management in Contiki

Most software on WSNs can be considered as event driven embedded systems. Events are changes of the state of I/O devices or timers. While it is possible to detect such state changes by polling periodically the concerned devices, the vast majority of systems (those that are not safety critical) rely upon interrupts for that purpose. This implies that the software is no longer purely sequential, but has become non-deterministic, as each interrupt causes the current program to be suspended while the specific interrupt handler is being executed. The primary task of an operating system consist in managing the interrupts and their associated handlers in such a way that the application programmer, instead of being overwhelmed by the non-deterministic behaviour of the software, can rely upon clearly defined and preferably easy to understand abstractions of the event driven reality.

Concurrent processes

The most common of these abstractions is the sequential process [7] a system that controls or observes several different concurrent phenomena is decomposed in purely sequential processes that are executed concurrently under the supervision of the process scheduler, one of the important components of the operating system. Typically each process can have three different states: ready, active and waiting (fig 1.7).

In the active state, the process is executing its sequential code, which can eventually contain an explicit wait statement, requiring that the process waits till a certain event x, such as the reception of a data packet, has occurred. When such a statement is executed, the state of the process becomes Waiting for x. While a process is waiting, other processes can become active. When event x occurs, the process scheduler changes the state of the waiting process into Ready and prepares to activate the process when processing resources become available. In some cases the process scheduler would stop ("preempt") an active process in order to allow another, more urgent Ready process to become active.

The main disadvantage of this mechanism is the necessity to save the entire state (all its variables and the stack) of a process when it stops to

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

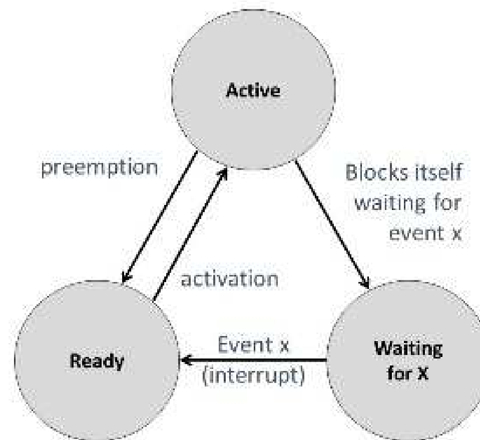

Figure 1.7: Process states

be active and to restore that state when it becomes active again. This can jeopardize system performance even on powerful computers when the number of processes is large and sophisticated memory management and protection techniques are being used.

Threads

To avoid the overhead of saving and restoring the state of processes threads have been introduced. Instead of being strictly sequential, a process contains several concurrent "mini-processes" called threads that share the memory space of the global process. This avoids saving the entire state of a process whenever an event occurs. As long as activity switches between threads of a same process, the size of the state to be saved remains reasonable as it can be reduced to the private stack each thread has to maintain to manage function calls and their parameter passing. Threads have become a standard feature of most operating systems and are part of the most popular programming languages.

Finite state machines

When resources are very restricted as in typical WSNs applications the memory requirements of threads are still excessive when the complexity of the application requires many of them. This is often solved by structuring the system as a single process that handles a set of finite state machines, each finite state machine playing the role that should normally be held by a separate thread. When an event occurs, the event handler selects the finite state machine for which this event is significant and updates accordingly its state. This approach requires very little memory and is fast as there is only one

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

process and one stack, which does not need to be saved when an event occurs and as the state of a finite state machine is just one byte if the finite state machine does not have more than 255 states. However, from the applications programmer point of view, the finite state machine approach has some severe drawbacks: the different logical threads of the application share the same code space and no syntactical rules prevent mixing them up and, as no explicit blocking wait statement exists, it has to be implemented by stepping through the successive states of a finite state machine which appears to be an error prone piece of programming.

Protothreads

Contiki offers a new programming abstraction, called protothreads [8] that can be used to replace finite state machines in event driven programs. Protothreads do not require significantly more memory than finite state machines while offering programming facilities almost similar to those of threads. Protothreads are constructs similar to threads as they provide a conditional blocking wait abstraction but they require much less memory than true threads as they do not provide local dynamic variables and therefore do not require saving a stack when the protothread is blocked. In fact, only the continuation address is saved when a protothread executes a blocking wait statement. Programming with protothreads appears to be much simpler and more compact than using finite state machines while not requiring significantly more RAM. The lack of dynamic local variables does not appear to be an important practical shortcoming. Protothreads can be added to the C language by means of ordinary macros when using the gcc compiler, or with any ANSI C compiler only when no switch statements are used in the same context as the protothread [9].

1.3.4 Communications facilities in Contiki.

The communications dedicated software is doubtless the most abundant and varied asset of Contiki as it is the research topic of a large part of the community that supports this open source project.

The parts of it that have been specifically involved in this Master Thesis will be shortly described in this section.

The physical (PHY) layer

Specific drivers for the various radios used in motes are included in Contiki. They can be found in the *core/dev* directory, but some parts of the drivers are specific for a given processor and are to be found in the appropriate subdirectory of the *core/cpu* directory. Finally some platforms use a radio

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

that is used nowhere else and the corresponding software can be found via the appropriate platform directory.

The Radio Duty Cycling (RDC) layer

To save energy in WSNs it is necessary to limit the time the radio is on as this is the most energy-consuming part of a mote. The problem resides mostly with the receiver as the transmitter can be switched on whenever something needs to be transmitted while the receiver, not knowing when a message is arriving, should continuously be kept on [10]. Instead, the receiver is kept in a low power sleeping mode most of the time but awakes periodically. RDC protocols try to control the timing of the transmissions so that they take place when the receiver is awake. Two approaches are commonly used: the synchronous and the asynchronous approach. With synchronous protocols sender and receiver keep synchronized real time clocks and time slots for communication are predefined. This requires, however to keep clocks synchronized, which is a complex task that can cost a non-negligible amount of energy [11],[12]. With asynchronous protocols a sender initiates a simple synchronization protocol for each message to transmit. Some protocols are hybrids as they start in an asynchronous way but keep timing information about successful transmissions so that subsequent transmissions will require less synchronization overhead.

Contiki has implementations of the most common asynchronous RDC protocols and they will be briefly described here.

Contikimac (CM) Contikimac is the RDC protocol invented by the main authors of Contiki [13]. Its functioning is summarized in fig. 1.8.

Most of the time the receiver sleeps, but a few times per second the receiver wakes up to perform two consecutive Clear Channel Assesments (CCA). The time interval between two consecutive CCA pairs is called the "Wake Up Interval" (WUI) and is set to 125 ms by default. If a pair of CCAs observes no radio-activity above a predetermined threshold the receiver returns to sleep. The sender, when it has a frame to transmit, does it repeatedly, for a duration longer than the WUI.

When a CCA detects radio-activity, the receiver stays awake and waits till it has received an entire and correct frame. When that has happened, the receiver causes the transmission of an Ack, and returns to sleep. Before returning to sleep, the receiver waits some more time to eventually receive and acknowledge subsequent frames transmitted in the same time-slot.

To optimize the power at the sender side, it is desirable to minimize the number of retransmissions.

When a sender receives an Ack, it knows that the receiver was awake just before the transmission of the acknowledged frame. As the wake-ups occur strictly periodically with a period of WUI, the sender can set up a table

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

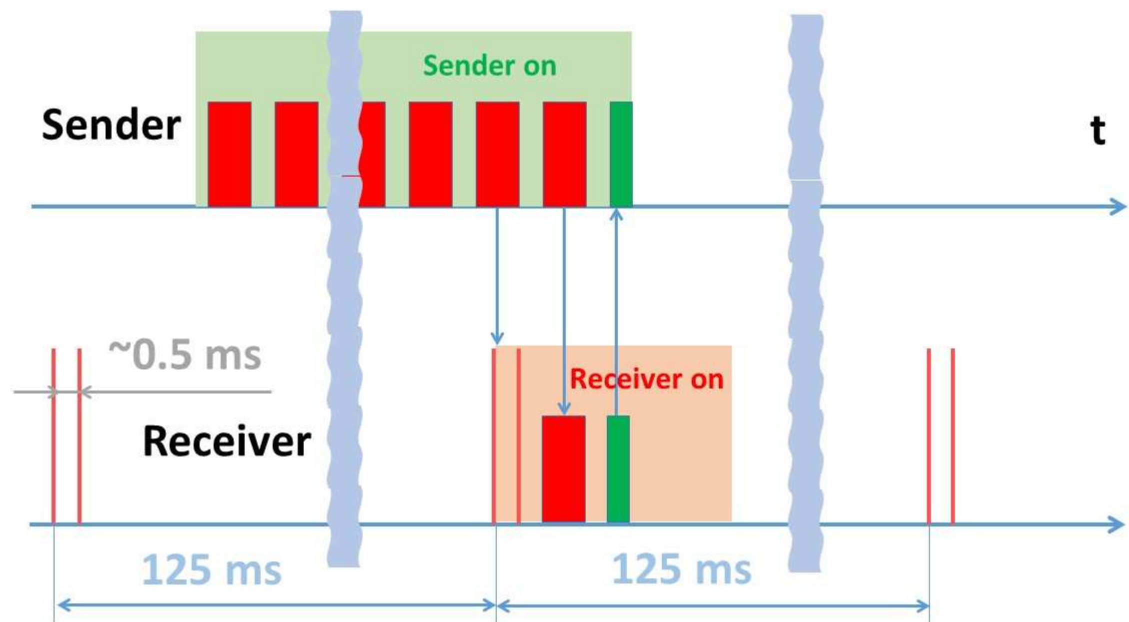

Figure 1.8: Principle of operation of contikimac.

giving for each destination the optimal time to start transmitting a frame (phase locking the sender and receiver). This is represented schematically in fig. 1.9. One can observe that, once the phase lock has been established, instead of retransmitting a large number of times each frame, the sender has to retransmit them only a few times as retransmission starts only when the receiver is going to wake up. By updating the timing when receiving each Ack, the phase lock can continue to operate over a long period, even if the sender and receiver clocks have slightly different frequencies.

When a sender repeatedly doesn't receive an Ack, it resumes sending frames for the whole duration of the WUI in order to reestablish a phase lock (this is not shown in fig. 1.9).

Low Power Probing (LPP) Low Power probing is a RDC protocol where the receiver announces by a broadcast when it is awake. A sender that has a frame to transmit switches on its receiver until it hears that the receiver is awake [14]. Operation of LPP is summarized in fig. 1.10.

Neither the original description of LPP, nor the implementation in Contiki use phase locking, despite the fact that this could reduce the power required by the sender as it would shorten the time the sender's receiver must be kept on.

upna
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

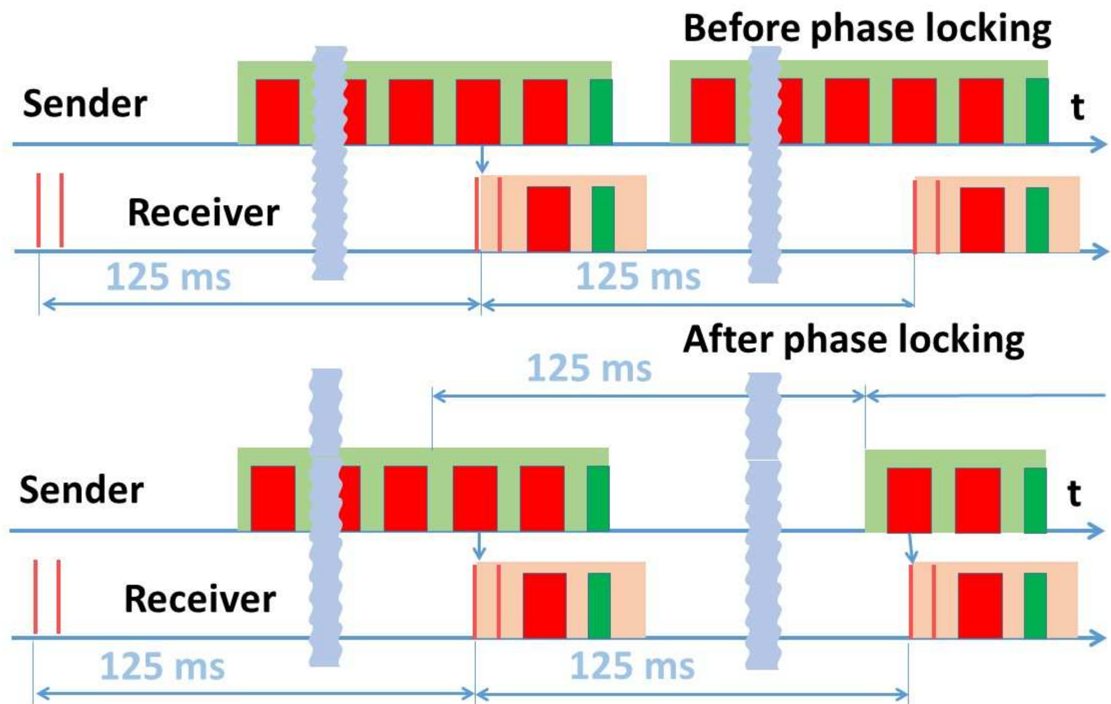

Figure 1.9: The benefits of phase locking in contikimac.

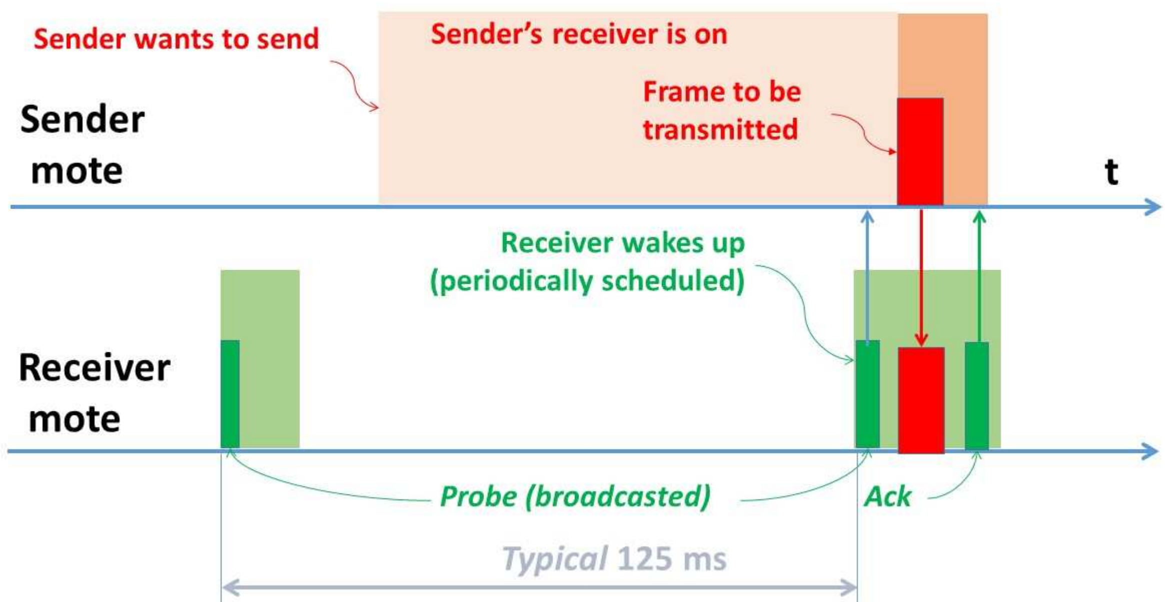

Figure 1.10: The Low Power Probing RDC protocol.

upna
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

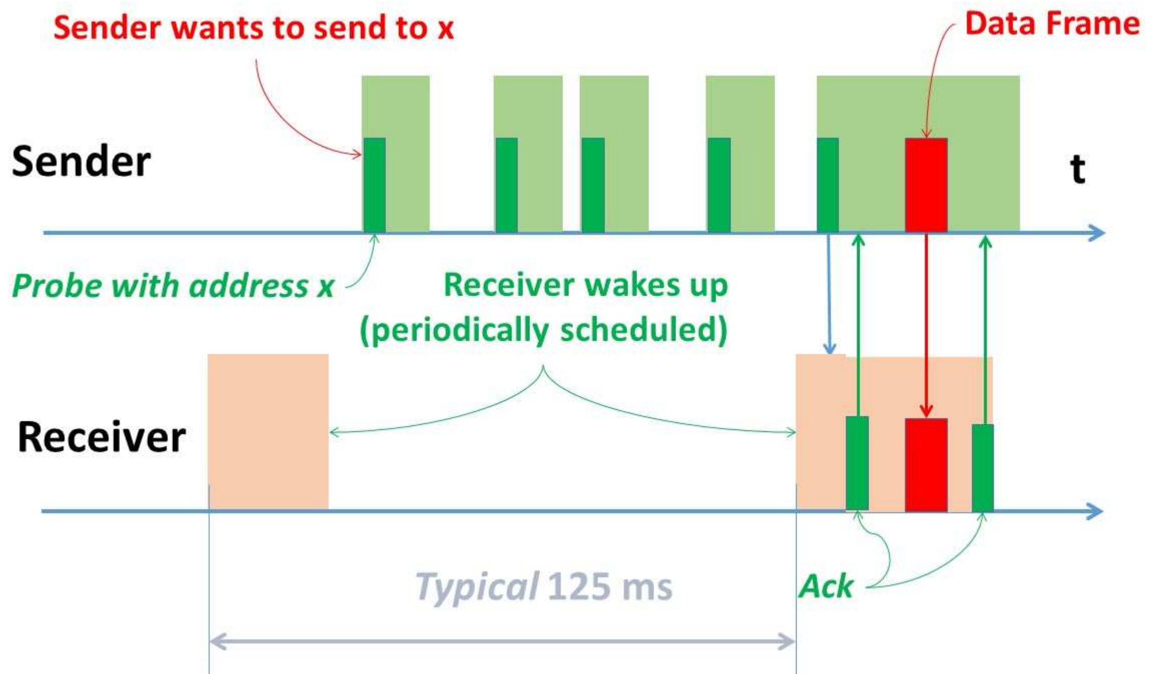

Figure 1.11: The XMAC RDC protocol.

The XMAC protocol This protocol was originaly described in [15] and is the most commonly used RDC protocol in TinyOS.

When a sender wants to transmit a frame, it sends at random moments a short request frame to the destination. When the receiver wakes-up and notices such a request, it answers with an Ack inviting the sender to send its data-frame. This operation is summarized in fig. 1.11.

The Contiki implementation of XMAC has a phase locking option similar to the one developed for contikimac allowing the sender to start sending probes just before the receiver should start listening.

NullRDC In order to allow switching off radio duty cycling, contiki provides nullrdc that can be configured just as the other radio duty cycling protocols, but that leaves the radio always on.

The Medium Access Control (MAC) layer.

Two MAC protocols are implemented in Contiki, CSMA and NullMAC.

CSMA As the two more efficient versions (CSMA/CD and CSMA/CA) of the Aloha derived medium access control protocols do not fulfil the memory and topological requirements specific for WSNs, Contiki has implemented a

upna
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

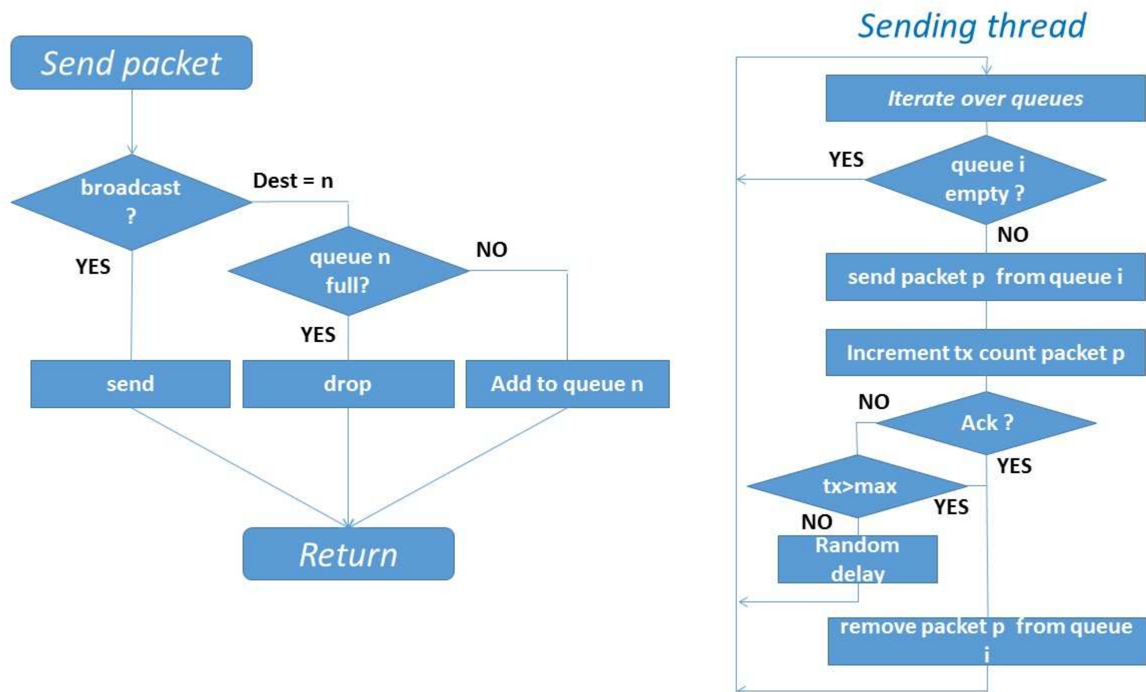

Figure 1.12: Simplified Control-flow chart of the Contiki implementation of the CSMA MAC protocol.

simplified CSMA protocol that manages a separate FIFO (First In, First Out) queue for the different possible destinations an tries repeatedly to transmit each frame and eventually drops the frame after three unsuccessful attempts. In CSMA the delay between successive attempts to transmit should be random, with a mean that grows exponentially. As the maximum number of attempts is limited to three, the exponential is just approximated by a linear function. Fig. 1.12 shows a simplified control-flow chart of CSMA as implemented in Contiki.

NullMAC The NullMAC protocol just sends once every frame, without checking if its gets acknowledged. This piece of software is mainly used for testing and debugging purposes.

The routing and application layers.

Contiki provides two different groups of protocols that can run above the MAC and RDC layers. The first one, called Rime is a fairly simple set of communication protocols specific to Contiki, while the second one, called uIP, is an implementation of the internet protocol stack adapted for devices with restricted resources (the u seems to stand for the μ in microsystem!).

upna
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Rime A comprehensive description of the objectives and the underlying software architecture of Rime is given in [16]. The authors describe the functionality of Rime as follows: "The Rime protocol stack provides a set of communication primitives, ranging from best-effort local neighbour broadcast and reliable local neighbour unicast, to best-effort network flooding and hop-by-hop reliable multi-hop unicast. Applications or protocols running on top of the Rime stack may use one or more of the communication primitives provided by the Rime stack".

uIP As the Internet has become the de facto ubiquitous standard telecommunications infrastructure, WSNs or the Internet of Things (IoT) need to be integrated into the Internet and using directly internet protocols in the WSNs or IoT simplifies tremendously such integration, but several constraints need to be considered:

- First, the addressing space: as WSNs and the IoT are conceived for large numbers of devices, the almost exhausted IPv4 address space is obviously inadequate. uIP uses the IPv6 addressing schemes.
- Next, the internet protocols optimize throughput and responsiveness, at the cost of memory and energy usage, while WSNs and the IoT seldom have significant throughput but have often severe hardware restrictions. Even the smallest IPv6 packets are larger than the maximum size IEEE 802.15.4 frames. Header compression and packet fragmentation have to be added to the traditional TCP/IP stack. 6LowPan is a sub- layer designed to be inserted between the MAC layer and the IP layer to satisfy these needs. It is implemented in Contiki under the name *sicslowpan*.
- Finally, it seems logical to consider each WSN and each physically integrated part of the IoT as a subnet in the Internet addressing scheme, but most internet protocols suppose that every subnet is a single broadcast domain, while communications between motes is generally based upon multi-hop links. A routing protocol with suitable algorithms applicable to a single multi-hop subnet is needed. RPL has been designed for that purpose by the Internet Engineering Task Force (IEFT) [17] and is implemented in Contiki. RPL is a fairly sophisticated variety of the Distance Vector routing algorithm. It leaves to the user (the application implementer) the choice of the function that defines the cost of each link and each node. This is the so called "Objective Function" (OF). In the Contiki implementation of RPL, an OF based upon ETX [18] parameter of the different links is available. The ETX value of a link is obtained from the CSMA MAC layer which keeps track of the average number of attempts that are needed to transfer successfully a

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

packet over each link. It is calculated in *neighbor-info.c*, included in the directory *core/net*.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Chapter 2

Preliminary study of the Zolertia Z1 Motes

As mentioned in Chapter 0, the goal was to develop various measurement techniques in order to evaluate certain performance parameters in any real world WSN deployment.

It was therefore necessary to get familiar with the use and management of the real motes. Hence, some preliminary measurements on the Z1 motes were made.

In this chapter the results obtained from such preliminary measurements are presented. Part of the data presented was recorded by the PhD student M.P. Uwase.

2.1 Introduction

The quality of a radio link can be characterized by three indicators. Two of them are evaluated by the radio receiver itself:

RSSI : "Received Signal Strength Indication". It is the measured power of a received radio signal[19]. If the RSSI value is high, the signal is likely to be less affected by noise and thus will be seen as "cleaner" or more "correct" by the receiver. However, reflection, scattering and other physical properties have an extreme impact on RSSI[19].

LQI : "Link Quality Indicator". Gives an estimate of how easily a received signal can be demodulated by accumulating the magnitude of the error between ideal constellations and the received signal [21] and considering noise in the channel [19]. The value is estimated by the demodulator. For the cc2420 radio, it is a dimentionless 7-bit integer, for which a value of 110 corresponds to a perfect correlation while a value of 50 is the lowest quality frames detectable by the cc2420 [20].

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Figure 2.1: A mote mounted on a wooden pole

Figure 2.2: The receiver mote.

For measuring such parameters, Contiki provides with two variables in the cc2420 driver (*dev/cc2420.h*), called *cc2420_last_rssi* and *cc2420_last_correlation*.

The third is directly relevant to the applications: the Ratio of Packets correctly Delivered (PDR). Studies showing that these three indicators do not correlate very well can be found in [22] and [23]. Therefore, whenever relevant, the values of the three will be given. To evaluate these indicators, a point to point unicast link was set up, using the communication functions provided by Contiki Rime.

2.1.1 The experimental set-up

The motes were mounted on 1m wooden poles by means of nylon straps. The battery powered transmitter could freely be positioned anywhere in the garden and given any orientation (see Figure 2.1).

The receiver was permanently positioned at approximately 1m from the veranda and was connected by a USB cable to a PC recording the received messages (see Figure).

All range measurements were done along one straight line, with no obstacles between sender and receiver, but some additional measurements were done along two other lines for checking influences of unsuspected causes. For

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

avoiding interference with possible WiFi networks all tests were performed in channel 26, the channel which is least susceptible to WiFi interference.

2.1.2 The radio traffic

To test a link, the sender transmitted every second a 45 bytes packet. For 50 consecutively received packets the values of RSSI, LQI and the sequence number of the packet were recorded by the PC connected to the receiver. The global values of RSSI and LQI were obtained by averaging the recorded values, while the PDR value was computed by dividing the number of received packets by the number of sent packets. The latter was obtained from the sequence number included by the sender in each packet. For these tests, the radio was continuously left on.

2.2 Sender and Receiver Calibration measurements

2.2.1 Sender power measurements

The antenna connector of the sender was connected to the input of a spectrum analyser. The spectrum of the signal, averaged over several seconds was displayed. The amplitude of the displayed spectrum was estimated in 4 points within the transmission band and averaged. The obtained results are shown in Table 2.1 and in Figure 2.3

Node-id	Peak1 (dBm)	Peak2 (dBm)	Peak3 (dBm)	Peak4 (dBm)	Average (dBm)
100	-21.0	-23.0	-23.0	-23.0	-22.4
109	-21.0	-22.5	-23.5	-24.0	-22.6
112	-20.0	-22.6	-20.5	-25.5	-21.7
113	-21.5	-24.0	-27.0	-27.0	-24.2
114	-20.0	-22.0	-23.0	-22.0	-21.6
116	-22.6	-23.0	-23.5	-24.0	-23.2
117	-23.0	-24.5	-25.0	-27.0	-24.6
118	-20.5	-21.5	-23.5	-23.0	-22.0
119	-28.0	-32.0	-31.5	-32.0	-30.5
121	-20.0	-22.5	-21.0	-23.0	-21.5

Table 2.1: Sender power measurements

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

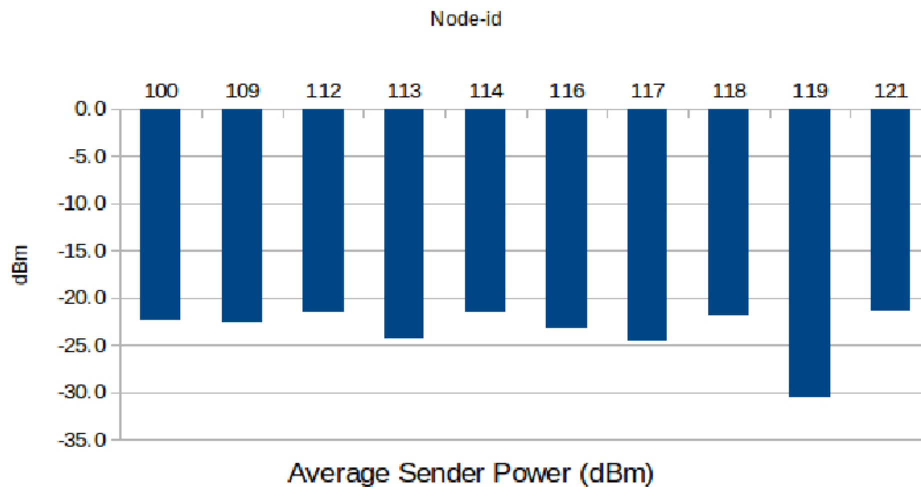

Figure 2.3: Sender power measurements

2.2.2 Receiver sensitivity measurements

The 10 evaluated receivers are successively placed in a fixed location, at some 6m from the sender. For each receiver 100 successive frames are received and the RSSI values estimated for each frame by the receiver are averaged (see Table 2.2 and Figure 2.4).

Node-id	RSSI (dBm)
100	-62.0
109	-83.2
112	-62.8
113	-62.9
114	-63.3
116	-61.9
117	-64.1
118	-62.6
119	-63.8
121	-62.2

Table 2.2: Receiver sensitivity measurements

upna
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

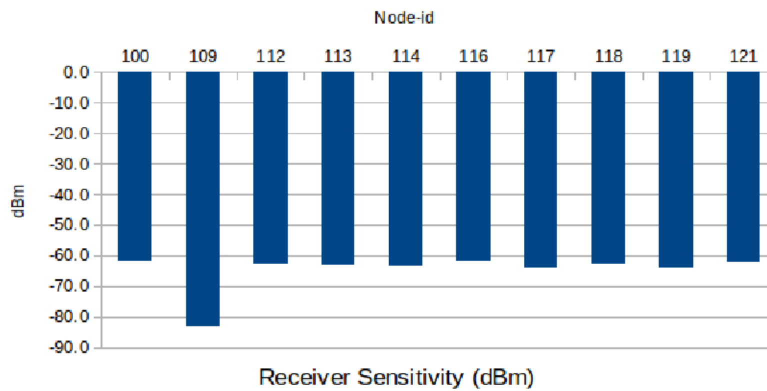

Figure 2.4: Receiver sensitivity measurements

2.3 Range measurements

First, the maximal range was evaluated at a nominal transmission power of 0 dBm in an open field (table 2.3).

Distance (m)	Min.RSSI (dBm)	Average RSSI (dBm)	Max.RSSI (dBm)	PDR (%)
20	-64	-62.5	-62	100
40	-66	-64.4	-64	100
60	-72	-70.0	-69	100
80	-74	-63.9	-73	100
100	-77	-76.0	-76	100
120	-80	-78.8	-78	100
140	-82	-82.0	-82	100
160	-92	-90.1	-87	100
180	-92	-91.2	-91	98
200	-95	-93.2	-92	16

Table 2.3: RSSI and PDR as a function of distance

Thereafter the quality of radio links was measured several times in three different directions in the garden, over distances ranging between 3 and 21m with steps of 3 m and with four different levels for the transmitted power (0 dBm, -10 dBm, -15 dBm and -25 dBm), to investigate how LQI evolves in function of the RSSI.

The LQI values did not show a predictable monotonic pattern: often they were better at medium distances (12m) than nearby (3m). However, most observed values were above 90 which corresponds to a good digital link. The

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

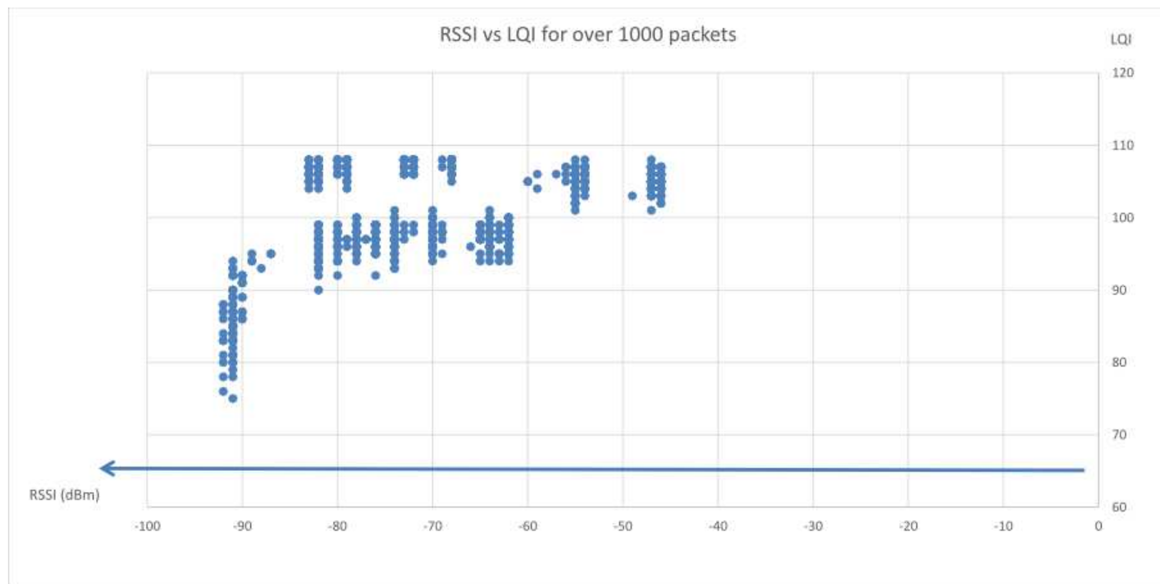

Figure 2.5: Scattergraph presenting the relation between RSSI and LQI for some 1000 received messages

quality of the digital link was confirmed by the PDR values, which were 100% for all links, except for one 21 m link with -25 dBm sending power were RSSI (-93 dBm), LQI (60) and PDR (< 10%) clearly showed that this was out of range. After such measurements, it was clear that the quality of the digital link degrades quickly below a RSSI value of -90dBm, but that at "good" RSSI levels almost no correlation between RSSI and LQI values exists (see Figure 2.5).

2.4 Conclusions

After this chapter we conclude that the behaviour of each mote when transmitting or receiving a packet depends on the mote itself. It can be useful for future users of these or new motes to make a series of measurements like those we presented to better predict causes of malfunctioning of the motes, before setting-up a complete sensor network. For example, in our case, after these measurements mote 109 was no longer used, since its receiver did not behave as expected, while the sender worked fine.

In an open field, at full power, Z1 motes with an external antenna can cover in a reliable way distances up to 160 m. However, the attenuation of the radio signal can vary considerably (up to 10 dB) from one day to another. Furthermore, moderate rain (2 mm/h) increases the attenuation by another 3 dB.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Chapter 3

Experimental evaluation of message latency and power usage in WSNs

3.1 Introduction

To study the interactions of Radio Duty Cycling (RDC), Medium Access Control (MAC) and Routing over Low power and Lossy Networks (RPL) protocols for wireless sensor and actuator networks (WSAN), an experimental study of the different RDC protocols readily available under Contiki for the Zolertia Z1 motes is being made. To draw credible conclusions from such a study, it is important to minimize the perturbations induced by the observation process in the studied protocols. Often simulators are being used for studying the behaviour of WSANs, and more specifically in a Contiki environment, Cooja which is part of the software development system, is quite popular. However, some simplifications inherent to simulations limit the accuracy of the results when studying low level protocols. For instance, a common clock for all simulated motes hides possible consequences of clock drifts between motes, and the rather simple definitions of simulated radio ranges in Cooja hide the complexity of actual radio propagation in the field. As we have the opportunity to do measurements with real motes in a large garden located far away from any disturbing radio transmitters, we developed tools for measuring packet loss, packet latency and power usage that minimally disturb the observed systems.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

3.2 Evaluation techniques

3.2.1 Packet Delivery Ratio (PDR)

This is the percentage of the packets that has been correctly received. To measure this ratio, the sender includes in the application level payload of each packet a sequence number. The receiver counts the number of received packets. The PDR is obtained by dividing the number of received packets by the number of packets sent, as derived from the sequence numbers.

3.2.2 Latency

We consider latency at the application layer. A message is sent when the unicast_send function is called and is received when the recv_uc function is activated after an interrupt from the radio receiver.

Latency is composed of three parts as shown in fig. 3.1.

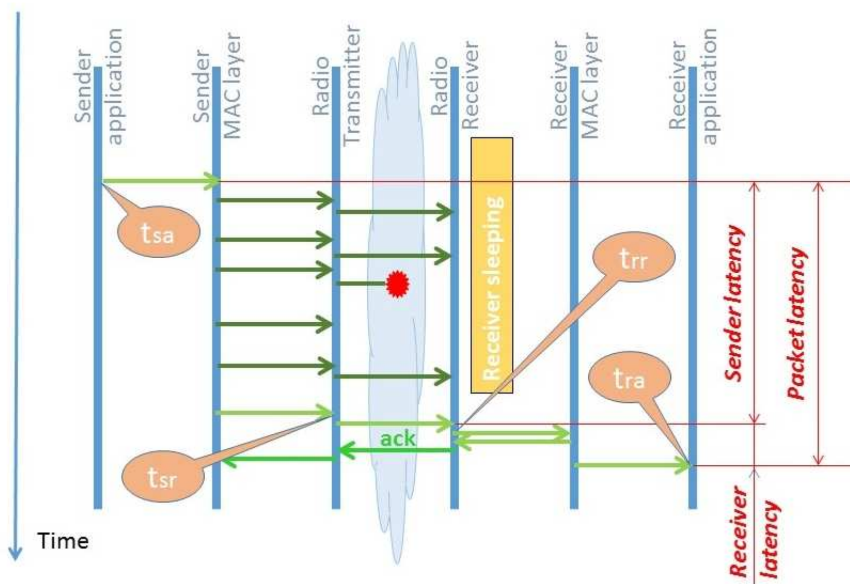

Figure 3.1: Message Latency

- First the message is passed from the sender application to the MAC/RDC layer. This layer tries to send the message until it is acknowledged by the receiver. This constitutes the largest part of the latency.
- The message is carried by radio from the transmitter to the receiver. Considering typical distances in WSANs, the time this takes can be neglected in the evaluation of latency.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

- When a message is received by the radio, it is decoded, its correctness and its destination are checked and, if these checks are positive, it is acknowledged and passed to the application layer.

The simplest technique for measuring latency would consist in including in each message the time it was sent and subtracting it from the time it was received. This would include three communication delays:

- Access delay: necessary time to grant a message access to the medium according to rules of the corresponding MAC protocol
- Propagation delay: necessary time to physically propagate the signal from the sender to receiver radios
- Reception delay: time between the reception of the start of frame by the receiver radio and the moment the entire message is passed to the application layer

This however would require synchronized real-time clocks in sender and receiver, given that each node has its own notion of time according to a clock located on the node's hardware platform. As the Cooja simulator included in the Contiki operating system uses a common clock for all simulated motes, this simple estimation method can be used in simulations, provided that there is room in the transmitted packets for one time stamp. Contiki provides with a set of different modules of timers. In this case, the function *clock_time()* which returns the current system time in clock ticks was used. This function can be found in the module *clock.c*.

In table 3.1 the latencies recorded with the presented method under Cooja for different MAC protocols and different wakeup rates are shown. The scenario for all simulations was two motes (a unicast sender and a unicast receiver) sending packets at the maximum transmission power level, one within the range of each other.

If the system under study does not include such synchronization (for example, in the case of a WSN with real motes), adding it would create a communications overhead that could jeopardize the accuracy of the observations[25] [26] [27]. With the function *clock_times()*, the system time starts from zero when the Contiki system starts. Those times are recorded

protocol wakeups/s	4	8	16
ContikiMAC	143.8	76.3	41.8
LPP	143.6	63.4	29.2
XMAC	150.0	72.6	40.1

Table 3.1: Latency measured with Cooja in ms

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

protocol wakeups/s	4	8	16
ContikiMAC	252.7	57.5	109.1
LPP	62.3	105.8	50.5
XMAC	107.5	83.2	60.9
NullRDC		82.2	

Table 3.2: Latency measured with non-synchronized real-clocks in ms

when the application layer sends a packet in the sender, and when the application layer receives that packet in the receiver node. The results are shown in table 3.2. However, as a consequence of the lack of synchronization between the two motes, the results for the real testbed with this method are obviously pointless.

A second technique is based upon two macros provided by radio drivers in Contiki. The first, to be used in the sender, appends to a packet the reading T_{sr} of the sender clock at the moment the radio starts sending. The second, for the receiver, copies in a variable the reading T_{rr} of the receiver clock at the moment the radio starts receiving a packet. In order to measure packet latency by means of these macros, the sender application inserts in the payload the reading T_{sa} of the sender clock at the moment the application passes the packet to the MAC layer. When a packet is correctly received, the difference of the T_{sr} and T_{sa} fields gives accurately the sender latency as both T_{sr} and T_{sa} are derived from the sender clock. Similarly the receiver latency can be expressed as a difference of two timestamps derived from the receiver clock (the receiver latency is mainly caused by the time needed to receive the entire packet). As a consequence, packet latency can be measured without synchronizing sender and receiver clocks, at the expense of two timestamps in the payload (four in case of multihop) and one additional radio interrupt in the sender and in the receiver.

As the additional timestamps and interrupts could disturb the normal operation of a protocol under study, a third way of measuring packet latency has also been implemented: both the sender and the receiver application briefly set and reset one of the I/O pins of the mote when a message is sent and received and an external data logger (also used for power measurements, see section) records the state of both I/O pins. Due to the low sampling frequency of the data logger (100Hz) these pulses need to be enlarged to 12ms by a monostable pulse shaper. As a result, a packet arriving less than 12 ms after another will not be recorded. This is a limitation of our latency measurements, but as in our experiments the average interval between packets was chosen to be 1s, the resulting error could be neglected.The schematic of the circuitry employed in this technique is shown in figure 3.2.

A more serious drawback of this latency measurement technique, results

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

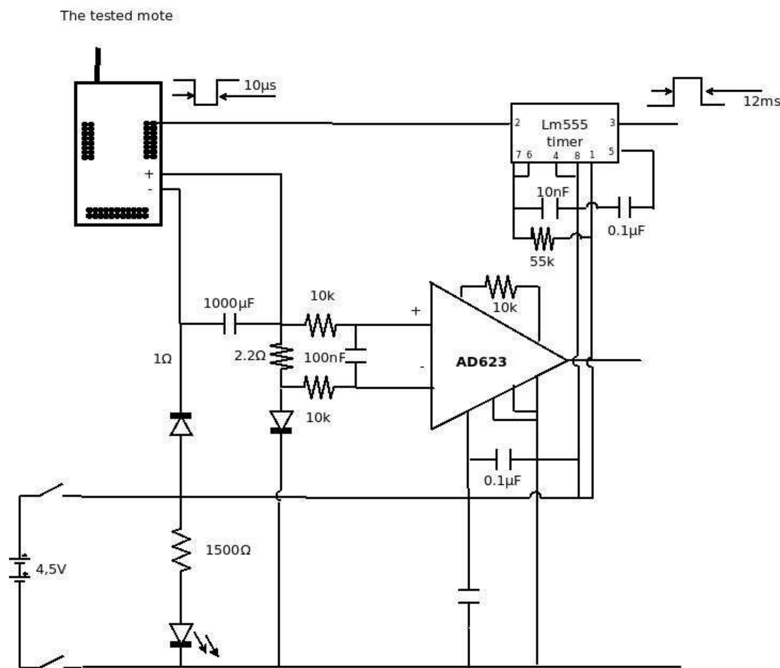

Figure 3.2: Circuitry schematic

from the difficulty to associate a sender pulse with the corresponding receiver pulse when latencies exceed the interval between packets. Various heuristics have been imagined and tested for associating sender and receiver pulses. The best one till now builds a table with two columns, the first containing the times packets have been passed by the sender application to the MAC layer (t_s) and the second the times these same packets have been received by the receiver application (t_s). It works as follows:

1. All recorded t_s are written down in the first column in the order the corresponding packets were sent.

2. The t_r are put consecutively in the second column, in the latest possible position allowed by causality (a packet cannot be received before it has been sent). This can result in some t_s with no corresponding t_r, implying that packets have been lost.

3. If a t_r cannot be added due to causality,

   (a) If one of the five previous sent packets was considered lost, all t_r after the lost packet are shifted back in time, creating space for the t_r that could not be added.

   (b) If none of the five previous packets was considered as lost, the received packet is considered as a duplicate and is ignored.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

3.2.3 Power

In 2007, Dunkels et al. [28] proposed to measure the power used by the motes of a WSAN by adding to their operating system four variables that would totalize the times the sender, the receiver and the CPU were active, as well as the time the mode was in sleeping mode. By multiplying these times by the power required by the different components of a mote, one could compute the energy required for a given task. In 2011, Hurni et al. [29] showed that this technique could result in high precision (1%) power measurements, provided that the power requirements of the components of each mote should be calibrated individually by means of accurate current measurements with enough temporal resolution to distinguish the different activities. They used for that purpose a sensor node management device [30], sampling the mote current at 1000 Hz. Recently another device, with even better resolution has been presented at the RealWSN workshop in Como [31] for similar purposes. However, with modern radios such as the CC2420 [32] these high frequency data loggers are quite useless, as the radio draws its energy from a large capacitor that is replenished from the external batteries whenever needed by a built-in DC to DC converter. Lengthy observations by means of an oscilloscope showed that power drawn by the mote was a poor indicator of the instantaneous activity of the radio and we concluded therefor that we should only rely upon current measurements averaged over a period exceeding the time constants built in the internal power supplies. We decided to rely for our performance studies exclusively on power figures obtained by measuring the average current absorbed by our motes and to collect enough experimental data to verify if these measurements would also allow calibration of individual motes as done by Hurni et al. to reduce the need of costly current measurements in the field.

Fig. 3.3 shows the block diagram of the data acquisition system and how it is connected to unmodified Z1 motes. To measure the current drawn by the

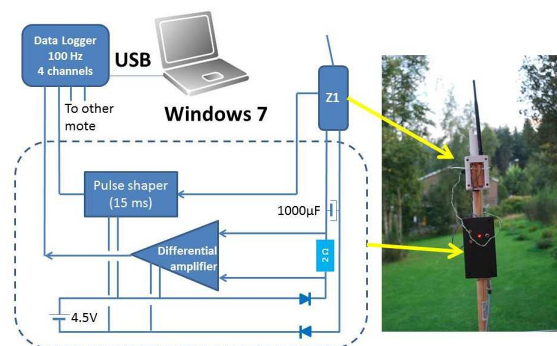

Figure 3.3: The data acquisition circuitry

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

mote, its batteries are replaced by a local power supply equipped with a 2Ω series resistor. The voltage across this resistor is amplified by a differential amplifier and sampled 100 times per second by a low cost USB data logger. A 1000μF capacitor ensures that no significant aliasing error will result from the 100Hz sampling. Another channel of the data logger is used to record the pulses generated by the mote on one of its digital output pins whenever a packet is sent or received.

3.3 Experimental Validation

In order to validate the different measurement techniques, and, by the same effort, gain better insight in the behavior of the different RDC protocols, a series of PDR, latency and power measurements were made with Zolertia z1 motes, using four different RDC implementations available under Contiki , with, when applicable, three different wake-up rates. The results obtained by different measurement techniques were compared.

3.3.1 The experimental set-up

The Z1 can be powered either by batteries or via the USB bus. For the PDR calculations, power estimation by the Contiki timings and latency measurements by means of timestamps, the received messages need to be recorded via a USB connection between the receiver and a PC. For the power measurements, the sender and receiver USB ports need to be disconnected, because whenever USB communications take place, the Z1 mote is powered via the USB port (given that the 5V input and the USB bus are directly connected to the Z1 printed circuit board). This forced us to make systematically two sets of consecutive measurements, one with USB connection and one without. The stability in time of the measurements allows us to use the results obtained with the two sets for comparison purposes.

	RSSI (dBm)		LQI	
	Average	Variance	Average	Variance
30 first packets	-57.9	0.3	107.1	1.1
30 last packets	-57.9	0.7	106.8	2.0

Table 3.3: RSSI and LQI

Table 3.3 shows the average and variance of RSSI and LQI of one of the experiments carried out. The first row shows the average and variance in the RSSI and LQI of the signal in the first 30 received packets. The same parameters are computed for the last 30 packets received during the

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

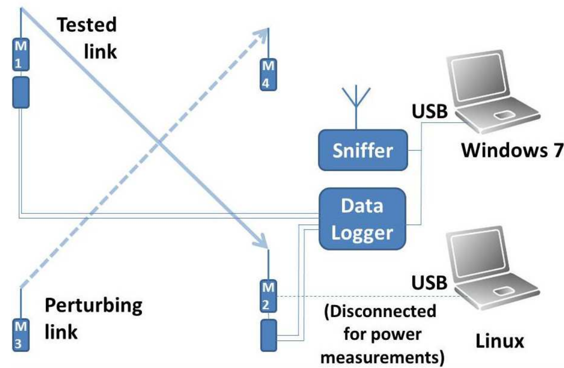

Figure 3.4: The experimental setup

P.nbr. RX	Time (us) +3069	Length	Frame control field					Sequence number	Dest. PAN	Dest. Address	Source Address
			Type	Sec	Pnd	Ack.req	PAN_compr				
2	=3069	61	DATA	0	0	1	1	0x3A	0xABCD	0x6400	0x7F00

MAC payload	LQI	FCS
00 30 85 00 64 00 7F 00 95 00 07 79 E6 4A 00 00 61 03 00 00 9E 7C 00 00 01 01 00 00 FC 01 00 00 00 00 00 00 00 00 00 00 00 00 00 05 07 00 06 E6 24 E9	91	OK

Figure 3.5: Frame detected by the sniffer.

experiment. As we observe, those figures remain stable during the whole experiment. Fig. 3.4 shows the experimental set-up.

Two motes (M1 and M2), located at some 10m from each other, are equipped with the external hardware described in paragraph IIc. For power measurements, motes M1 and M2 are only connected to the data logger because other connections would impair power measurements. The behaviour of a radio link is evidently influenced by the activity of other radio transmissions in the neighbourhood. Such unrelated radio-traffic is created by two additional communicating motes M3 and M4. The global radio traffic is monitored by a dongle (TI, CC2531DK) and a free sniffer software (Texas Instrument, "Instrument SmartRF Packet Sniffer").

3.3.2 The radio traffic

The traffic generated by the sending mote M1 consists in 59 bytes unicast messages transmitted at an average rate of 1 message per second. Figure 3.5 shows the format of each one of the packets sent by the sender, recorded by means of the TI software SmartRF and the sniffer device.

When connecting to the receiving mote, we obtain an output similar as the one shown in Figure 3.6. From the file generated by means of the command "make serialdump login", the data for our analysis can be extracted and easily analysed.

The inter-message interval is randomly distributed between 20 and 180

upna
Todos los derechos reservados
Eskubide guztiak erreslabatu dira

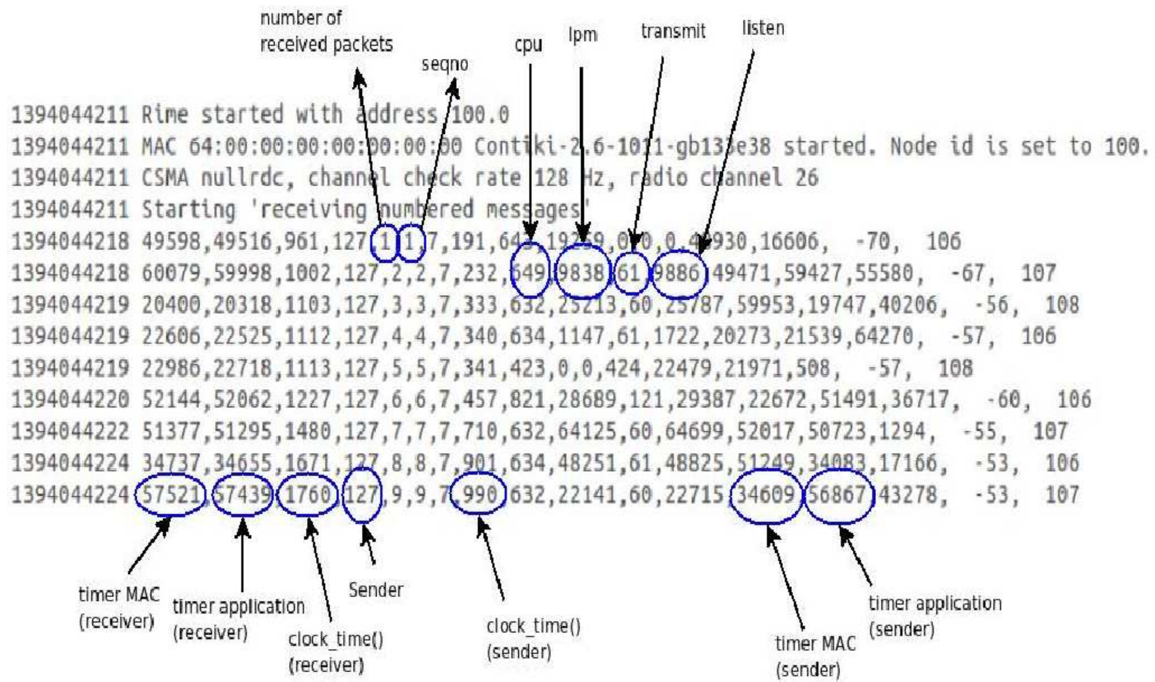

Figure 3.6: Format of the packet sent

ms in those experiments. Random inter-message delay has been chosen for the experiments providing that with a fixed delay every packet would be sent always at the same time with respect with the duty cycle. The experiments could have also been performed using a fixed delay between packets, but thirty independent measurements would have been required to compute latency without the influence of the phase with respect of the duty cycle. With a random delay we only need one experiment in order to obtain the mean value of latency. Fig 3.7 illustrates this.

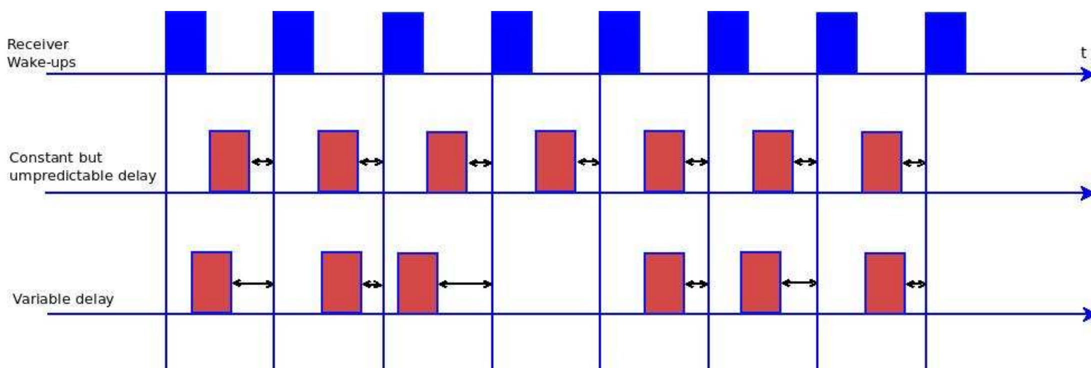

Figure 3.7: Interval between packets

upna
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

In order to obtain statistically valid measurements, at least 100 messages are observed per experiment. Additional traffic can be created by the second sending mote M3 unicast link from M3 to M4 to study the effect of global radio traffic. Lightly interfering traffic is generated when both M3 and M4 are active. Heavier traffic results from switching off M4 as then, M3 will repeatedly try to transmit its packets to the non-responding M4. The Contiki Rime software is used for managing the unicast transmissions and CSMA is used as MAC protocol. Four different RDC protocols, readily available on Contiki, have been used for the tests in order to obtain clearly defined, but different traffic and power usage conditions. They are ContikiMAC [33], XMAC, [34], LPP [35] and, as a reference, NullRDC (this protocol leaves the radio always on). The previously mentioned protocols are used in both the tested link and in the perturbing link, as in a realistic WSN scenario all the motes will use the same MAC protocol.

3.4 Quantitative Observations

3.4.1 Comparison of latency measurement techniques

Firstly, the latencies measured with the simulation in Cooja and with the real motes were compared. In order to do that, the same set of 9 experiments were carried out (ContikiMAC, LPP and XMAC with three different wake-up rates). Table 3.4 summarizes the total latencies measured for the real motes with no additional radio traffic. Comparing these results with the results shown previously in table 3.1, differences up to a 33% are observed.

protocol wakeups/s	4	8	16
ContikiMAC	176.8	81.6	55.6
LPP	160.4	63.5	33.8
XMAC	159.6	77.0	42.3

Table 3.4: Latency measured with Z1 motes in ms

In the second place, latencies for ContikiMAC and NullRDC were recorded according to the two different described techniques. Tab. 3.5 gives the results of latency measurements with no additional radio traffic.

The latency measurements by timestamps for NullRDC are clearly erroneous. The transmitter latency appears to be the time between packets rather than the time the packet is upheld in the transmitter. As this time is very short, it appears that the request to store the time the transmission starts arrives after the start, so that the recorded time is the start time of the next packet. The average latencies measured for ContikiMAC by the two methods dont́ differ by more than 2%, which is good considering the

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

RDC	by timestamps			pulses
	Tx lat.	Rx lat.	Total lat.	Latency
	ms	ms	ms	ms
NullRDC	993.7	2.6	996.3	6.2
ContikiMAC	82.7	3.5	86.2	87.8

Table 3.5: Comparison of latency measurements

Activity	units	CPU	LPM	Tx (-15 dBm)	Rx
Calibration	mW	15.4	3.9	543.3	40.3
DataSheets	mW	33	0.0017	32.7	62

Table 3.6: Power coefficients per activity

considerable variations in the latencies of individual packets due to random positioning of the send requests with respect to the RDC cycles.

As a conclusion for this comparison, one can state that the timestamp method is an accurate and low cost alternative for the external data logger when the sender latencies exceed 5 ms and there is room in the payload of the packets for two 16 bit time stamps (four time stamps would be required for multi-hop communications) whenever latencies measurements are required for a WSN implemented with real motes.

3.4.2 Comparison of power measurement techniques

As power requirements of electronic components can vary due to tolerances in the manufacturing process, we first compared the energy used by 10 randomly selected Z1 motes for sending 100 packets. The power level for transmissions was set to -15 dBm in all experiments described in this section. The measured power requirements for transmitting 100 packets were quite uniformly spread between 298 and 342 mJ. This justifies why Hurni et al. [29] recommend to calibrate individually all motes before using the four activity timers provided by Contiki to estimate energy usage.

Such a calibration of one mote was attempted by performing 12 different experiments, where energy usage and activity timers were recorded. Table 2 shows in the Calib. row the power corresponding to each of the four activities as computed by means of a least squares regression. It also gives the corresponding values derived from the data sheets of the components of the motes.

One can observe huge differences. The values corresponding to the low power mode and to the transmitter mode are unrealistically high when compared with the data sheets. The high value for low power might be caused

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Protocol	units	Datasheet	Calibration	Measured
NullRDC	mJ/pkt	62.1	45.2	42.4
ContikiMAC8	mJ/pkt	2.1	9.7	3.2
ContikiMAC16	mJ/pkt	9.8	996.3	3.5
LPP8	mJ/pkt	8.4	12.8	16.9

Table 3.7: Power usage by mote 127

by Contiki not accounting the devices that are permanently on such as the 32768 Hz real time clock. The small fraction of the time the transmitter was active during all the calibration experiments explains why the error on the corresponding value can be large. To evaluate the effect of the low accuracy of the power coefficients, four additional experiments were made, consisting in sending 100 packets, using four different MAC protocols. The required energy per packet was measured and computed with the coefficients obtained by the least squares regression and from the data sheets. Table 3.7 gives the results of these experiments.

Globally, the results obtained from the software activity timers are poor, both when using data sheets and when using coefficients determined by a mote calibration process. Clearly calibrations based on measurements made with standard MAC protocols do not give accurate enough results when applied to motes that do not allow current measurements with a time resolution fine enough to distinguish the different states of the mote. The design of special purpose calibration protocols that would emphasize each of the four mote activities separately could improve the accuracy of the calibration but would require a considerable programming effort. In conclusion, we will restrict the usage of the activity timings to comparisons of different protocols on real or simulated motes and use real motes and hardware measurement techniques when actual power values need to be evaluated.

3.4.3 Comparison of RDC protocols

As examples of investigations made possible by the described measurement techniques, we present some performance comparisons between the different RDC protocols available under Contiki. To the best of our knowledge, very few comprehensive comparisons of the performance of RDC protocols have been published and those available are very specific or mainly based upon simulations [36]. Table 4 shows a comparison of the performance of the different available RDC protocols measured with a low global traffic level (M3 and M4 active) and one wake-up every 125 ms for all motes. The transmission power was set at 0dBm for these measurements. Table 5 shows, for the same global traffic level, the influence of the wake-up rate on the performance of

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

Protocol	PDR	Latency	Tx Energy	Rx Energy
	%	ms	mJ/packet	
NullRDC	100	5	38.73	38.43
ContikiMAC	90	103	3.55	2.47
XMAC	98	543	12.57	5.04
LPP	96	122	10.48	6.2

Table 3.8: Latency and power comparisons (low traffic)

ContikiMAC				
rate	PDR	Latency	Tx Energy	Rx Energy
wakeups/s	%	ms	mJ/packet	
4	87	170	3.32	2.7
8	90	103	3.55	2.47
16	100	50	3.94	2.69

Table 3.9: Influence of the wakeup rate on performance

ContikiMAC. From table 4 one can clearly deduce that power is reduced significantly at the expense of message latency when the radio is periodically switched off. In this example, the use of the ContikiMAC RDC protocol rather than NullRDC reduces power by a factor 13 but increases latency by a factor 20. Big differences between the performance of the evaluated RDC protocols are visible: with a similar workload, XMAC, despite requiring 5 times more power than ContikiMAC causes a 3 times longer latency.

Table 3.8 shows also that RDC protocols cause the loss of some messages. This loss reaches 13% for ContikiMAC, which is abnormally high. This will further be discussed in section 3.5.

From table 3.9, it appears that for ContikiMAC, the wakeup rate has a limited effect on the required power, while it has a very important effect on the latency. The limited influence on power is probably a result of the "fast sleep" [33] algorithms used in ContikiMAC to keep the power cost of a wake-up very low. Moreover, increasing the wake-up rate seems to reduce the loss of messages. Therefore we are considering adopting 16 rather than 8 as the default wake-up rate for our future experiments.

3.5 Still Unexplained Observations

The abundant data about transmitted packets collected through our experimental set-up gave us much deeper insight in the fate of these packets than whatever we had been able to collect previously by means of application

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

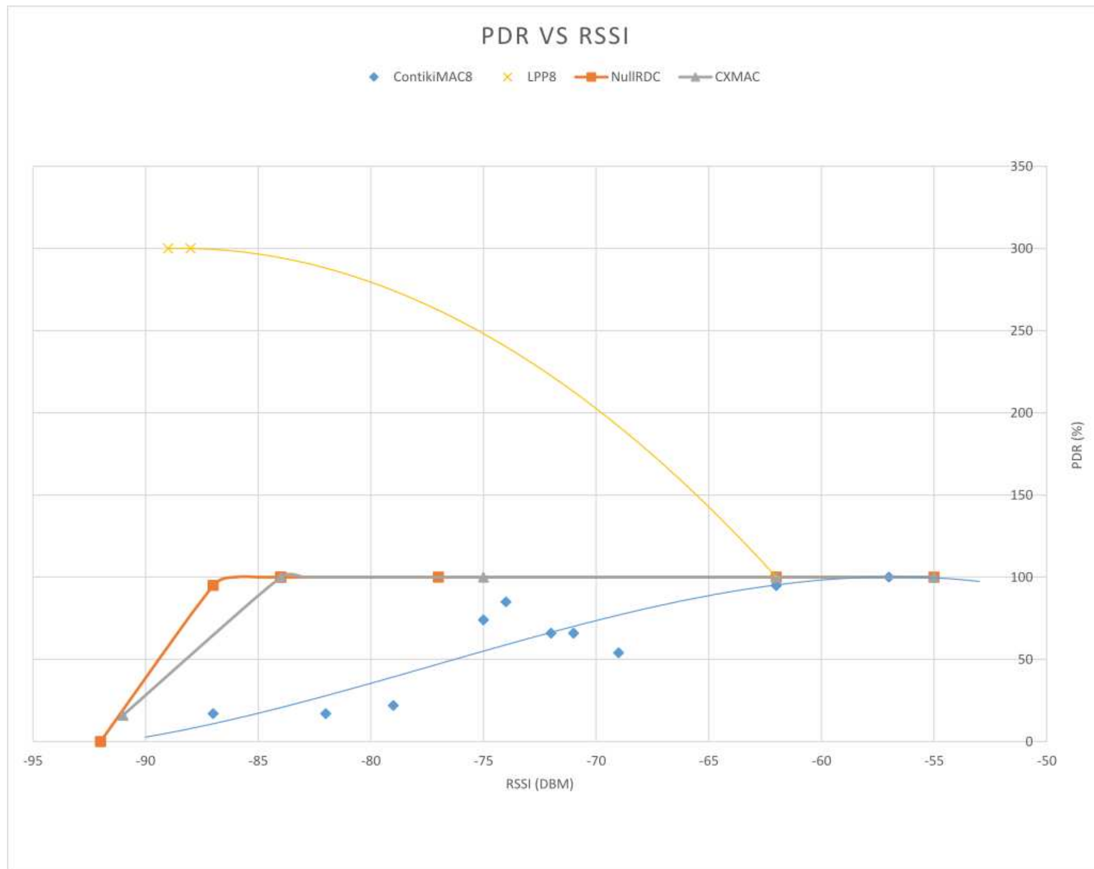

Figure 3.8: PDR vs RSSI for all RDC protocols

layer software. By combining the sent and received times from our data logger with the contents of the transmitted frames collected by the sniffer, we could elaborate some possible, still to be verified, explanations for unexpected observations. First of all, we tried to understand why sometimes RDC protocols caused serious reductions in the PDR figures, while the constant 100% figures obtained with NullRDC point to the RDC driver as the culprit. Losses of 1 or 2% could be attributed to random accidents such as a collision involving an acknowledgement. However, some experiments resulted in PDR figures as low as 60%. Figure shows a graph that summarizes all those results, representing for all four RDC protocols the PDR in function of the RSSI. Detailed analysis of the recorded data shows that there are two type of unexpected message losses. The first occurs when, for unknown reasons, an Ack is ignored by the receiver, and therefore duplicated packets are received (for example, this occurs with LPP). The second happens because the delay between two consecutive packets in the sender is much higher than what the ContikiMAC specifications required.

Exploring more in depth the behaviour of PDR with ContikiMAC8, an

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

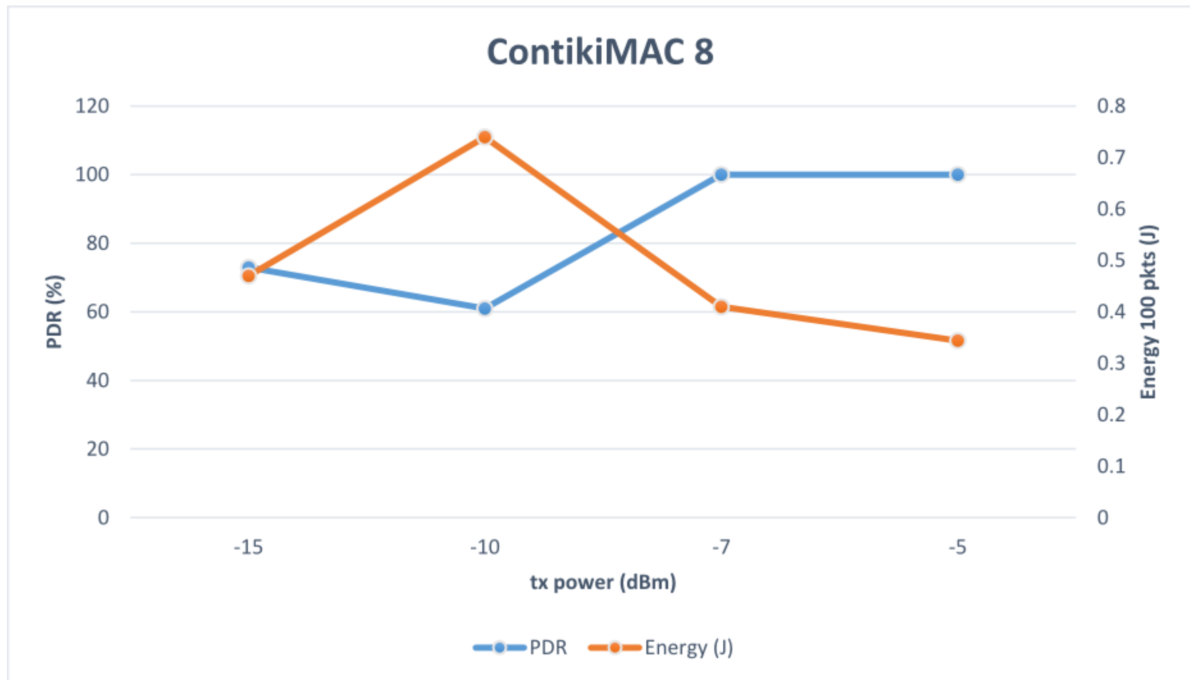

Figure 3.9: PDR and Power Usage with ContikiMAC 8

unexpected relationship between PDR, transmission power and power usage was found (see Figure 3.9).

As expected, when increasing the transmission power, the PDR is increased. Surprisingly, when the transmission power is higher, the power usage becomes lower, which indicates that the power used by the sender is almost not influenced by the selected output power. Furthermore, when a packet is not acknowledged by the receiver (as it happens with low transmission power levels) the sender tries to retransmit the same packet until it receives the Ack, which makes the power usage more efficient when we increase the transmission power.

3.6 Conclusions

The implemented hardware measurement technique allows a precise evaluation of the performance of a properly working single hop link and constitutes a precious tool for gaining insight in the (correct or incorrect) behaviour of the communication protocols, without disturbing significantly the timing of these protocols. The software techniques are less costly but could interfere with the normal operation of the network as they require insertion of measurement data in the transferred packets. Their accuracy is excellent for latency but is problematic for power if an accurate calibration of nodes

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

cannot be done.

The work presented in the following chapters consists in extending the data acquisition system in order to allow multi-hop measurements in RPL networks and to further explore unexpected protocol behaviours. Considering the volume of data such systems will provide, better data analysis software will also be required.

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Chapter 4

Experimental Evaluation of the performance of a RPL network

4.1 Introduction

Once the different techniques presented in the previous chapter have been developed and validated, they can be extended and applied to a RPL multi-hop network, in order to analyse the performance of such a network in terms of PDR, packet latency and power usage.

In this chapter, a real RPL network was set-up. The purpose is to evaluate the performance of such a network with two different types of traffic defined.

4.1.1 The experimental set-up

In order to study experimentally the performance of a RPL network, a test facility was built in the garden of a countryside house. As it was explained in Chapter 3, this way it is easier to interpret far from potentially disturbing radio transmitters and from reflecting structures causing multi-path interferences. The facility consists in a WSN with between 2 and 21 motes, depending on the experiment. Those motes were positioned all over a 1000 m^2 lawn, and one gateway mote, located near a veranda, where computers and test equipment were housed.

As described in Chapter 2, some RSSI and LQI preliminary measurements were carried out in the testbed in order to build a multihop RPL network. Several values for the transmission power were tested, as well as different positions for the motes. We were interested in having at least two motes connected to the sink via a multi-hop link (motes 105 and 106). As a result of such measurements, the transmission power of each node was set as level 7 (-15 dBm) and the position of the motes was settled as shown in Figure 4.1. Motes 105 and 106 were situated further away from the sink to force them to stablish communication via other motes (predictably via 124 and 125).

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

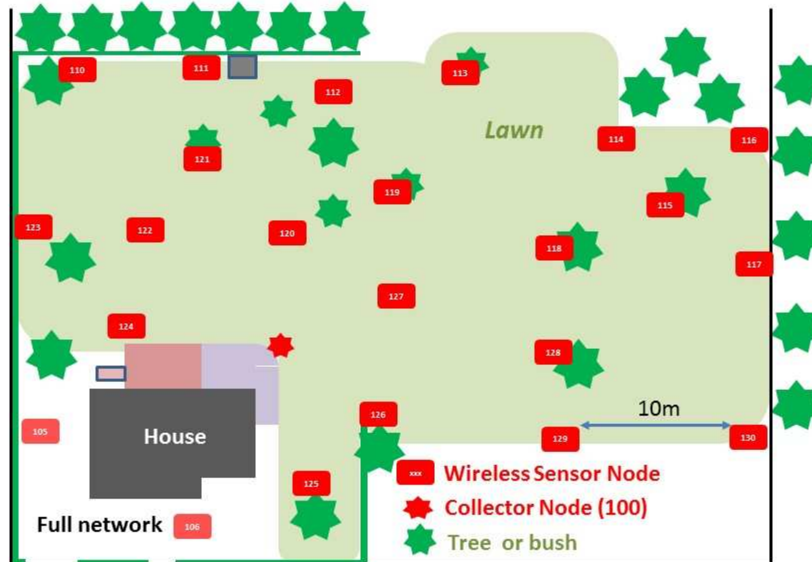

Figure 4.1: Position of the motes.

Since those experiments result in massive amounts of data, several software tools were developed, in order to evaluate systematically the parameters we are interested in. The details and code are tabulated in Appendix A.

In order to see the impact of increasing the traffic on the indices under examination, the number of nodes in the network was progressively increased in steps of seven motes, starting with a seven-node network.

Each one of the Zolertia Z1 motes is running Contiki. The software *udp-sender.c* (contained in the directory */examples/ipv6/rpl-collect*) was uploaded in the senders. The RDC protocol chosen for this set of experiments was the one by default in Contiki, ContikiMAC with a wake-up rate of 8Hz.

4.1.2 The radio traffic

Each one of the senders in the RPL network generates a packet every second. Those packets are received by the sink, connected to a pc via a USB cable. In figure 4.2 an example is shown, with the most important data highlighted.

The design proposed by N.T Long et al. in [37] was implemented in a RPL network. The proposed model intends to support priority traffic in a LLN, assuming that there are two types of nodes in the network (except for the sink):

1. Type 1 (T1) is a set of regular nodes, which are responsible for physical

upna
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

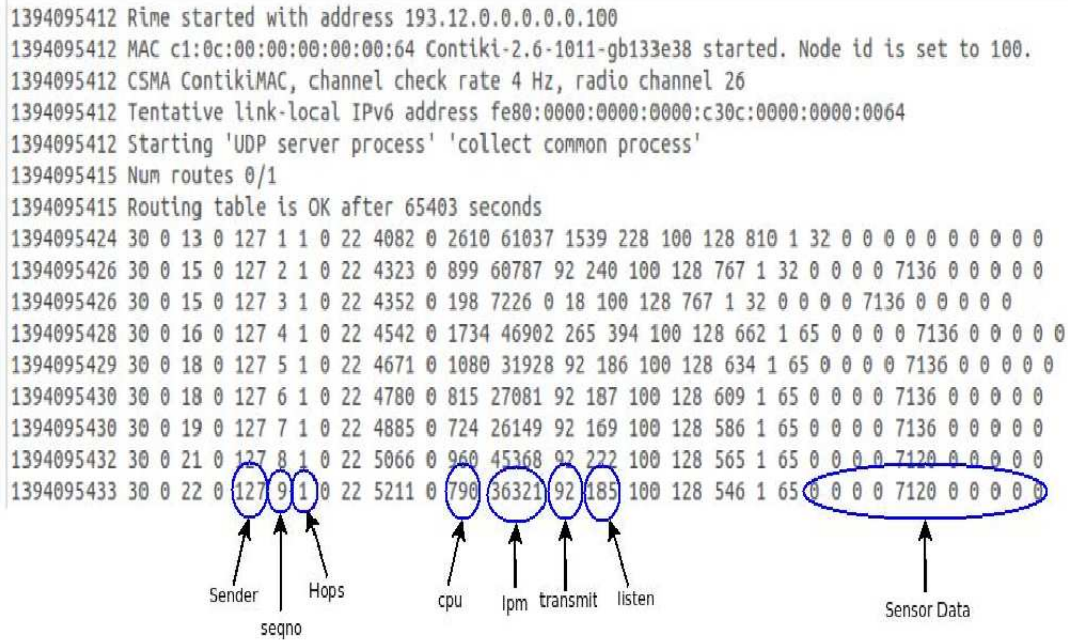

Figure 4.2: Serialdump when connecting to the sink

environment monitoring and generates data packets at 1 packet per second towards the sink.

2. Type 2 (T2) are alarm nodes. These nodes do not send packets periodically. Instead they randomly generate small-size alert packets in high-rate towards the sink. To increase the odds of packet reception, every T2 packet is sent several times (i.e 3) consecutively.

In the set-up network, the number of T1 nodes is much higher that T2s. We start with seven T1 motes, and we increase them in steps of 7 motes. The network in the presented experiments always had two T2 motes.

Even though N.T Long et al. propose a model with multiple RPL instances, in our implementation we employ only a single RPL instance.

In order to distinguish the two types of traffic in the network layer, alert packets are defined as Class 0, while the regular packets have been clasified as Class 5. These values are set in an 8-bit field, Traffic Class, in the IPv6 header. In our case, 6LoWPAN does not remove all the control fields in the IPv6 header, it only compresses the source and destination fields in the header. Since a node can access this identification, it sets an internal flag to tell lower layers how to treat the incoming packets from the Network layer.

To improve communication performance of priority traffic in the MAC layer, a new feature was added: queue scheduling. This helps to decrease end-to-end packet delivery latency.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

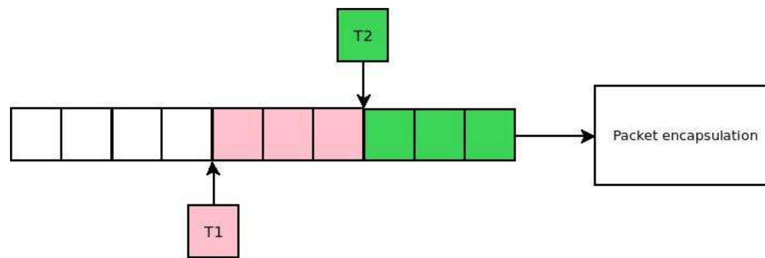

Figure 4.3: The last packet is served before all T1 packets.

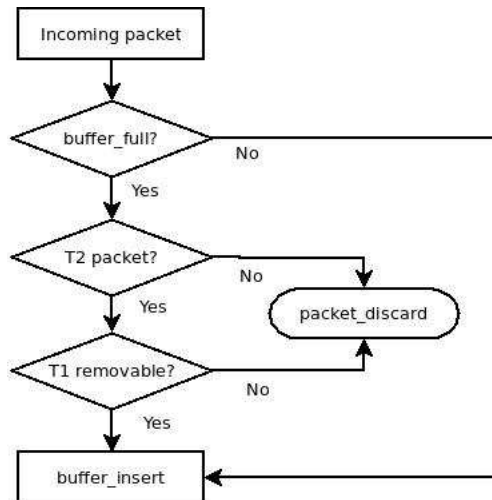

Figure 4.4: The operation of the buffer.

In WSNs, FIFO queueing is widely implemented for buffering packets. However, when two types of packets with different priorities have to be handled, FIFO is quite inadequate. N.T Long et al. propose a queueing model to support both types of traffic in the network, by adding a physical buffer between 6LoWPAN and MAC layer. Whenever a T2 packet is received, it will be inserted in the queue before T1 packets. When the queue is full, a T1 frame is simply dropped, while a T2 frame is only dropped if the buffer is full and contains only T2 packets. The opperation of the buffer is summarized in Figures 4.3 and 4.4.

4.2 Evaluation Techniques

4.2.1 Packet Delivery Ratio

As it was defined in the previous chapter, the PDR is the percentage of correctly received packets. To measure this, the application in the sender

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

(*udp-sender.c*) includes a sequence number in every packet (see Fig. 4.2).

In our first attempt to compute PDR in the RPL network, Collect View was used to gather data from our network. This tool provides with some useful data, such as number of packets received and number of packets lost.

4.2.2 Latency

Unlike for the PDR, Collect View does not provide with means to compute latency directly. So, in order to compute latency in a RPL network, the same techniques that were presented in Chapter 3 were implemented, meaning software techniques and hardware measurements. As a first approach, a simple single-hop RPL network was set-up in the laboratories of the ETRO department. The sink was connected with the DataLogger as it was explained in the previous chapter, and so was the RPL sender.

In such a simple network, both methods can be applied, but when the network was expanded, the hardware method could not be used. This is due to the fact that with the DataLogger used in Chapter 3, we can observe when a mote is active, but we cannot identify the mote itself. Therefore, technique is based on the macros provided by radio drivers in Contiki was used here.

4.2.3 Power Usage

As it was concluded in Chapter 3, our attempts to find the coefficients for calculating the total power consumption in each mote from the cpu, lpm, listen and transmit values were not conclusive. Even though Collect View provides us with such information, it cannot be used for calculating the power usage. The alternative to the sofware method could be the data logger. However, the packets received cannot be unambiguously identified when we have more than one sender in the network. In conclusion, with the available measurement techniques at that moment, the power usage in a RPL network could not be computed.

4.3 Quantitative Observations

4.3.1 Packet Delivery Ratio

For evaluating PDR in a RPL network, at the beginning we set-up a network with 4 motes: two of them running the firmware *udp-sender*, and the other two *udp-sender-urgent*. In the next experiments we increased in steps of 3 the number of nodes running the software *udp-sender*. The results are shown in Figure 4.5. Each bar represents the total Packet Delivery Ratio for each type of mote: orange for the normal motes and blue for the priority motes.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

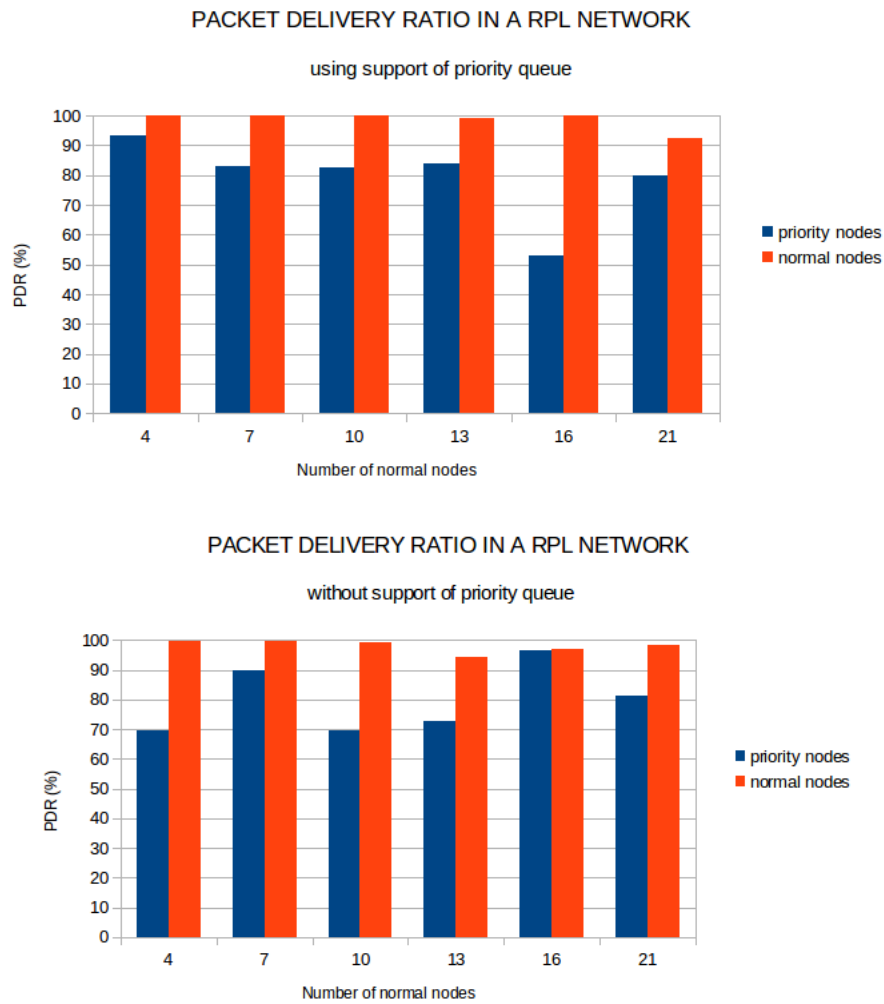

Figure 4.5: PDR in a RPL network. 16/01/2014

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

As it is observed in the graphs, the results are not consistent. The moments when the RPL network seemed to behave poorly, for example, while we run the network with 7, 16 and 21 motes, coincided with a heavy pouring rain. This forced us to wait for better atmospheric conditions in order to have consistent data over time to be able to study the performance of the optimization proposed by N.T Long et al.

In the following set of measurements, we considered that for obtaining more regular results over time, we had to optimize our time. For obtaining statistically significant results, we waited until we received some 1000 packets from the priority motes. That means that each one of the measurements is very time-consuming, that is the reason why we decided that only three networks would be under study: with 9, 16 and 21 motes. As in the previous case, each one of the networks contained two priority motes. The results are shown in Figure 4.6.

When analysing those results, it became more clear that the performance of each mote differed. In figure 4.7 we represent each one of three experiments in terms of PDR versus hop count (with 9, 16 and 23 motes), when using and not using the priority queue under study.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

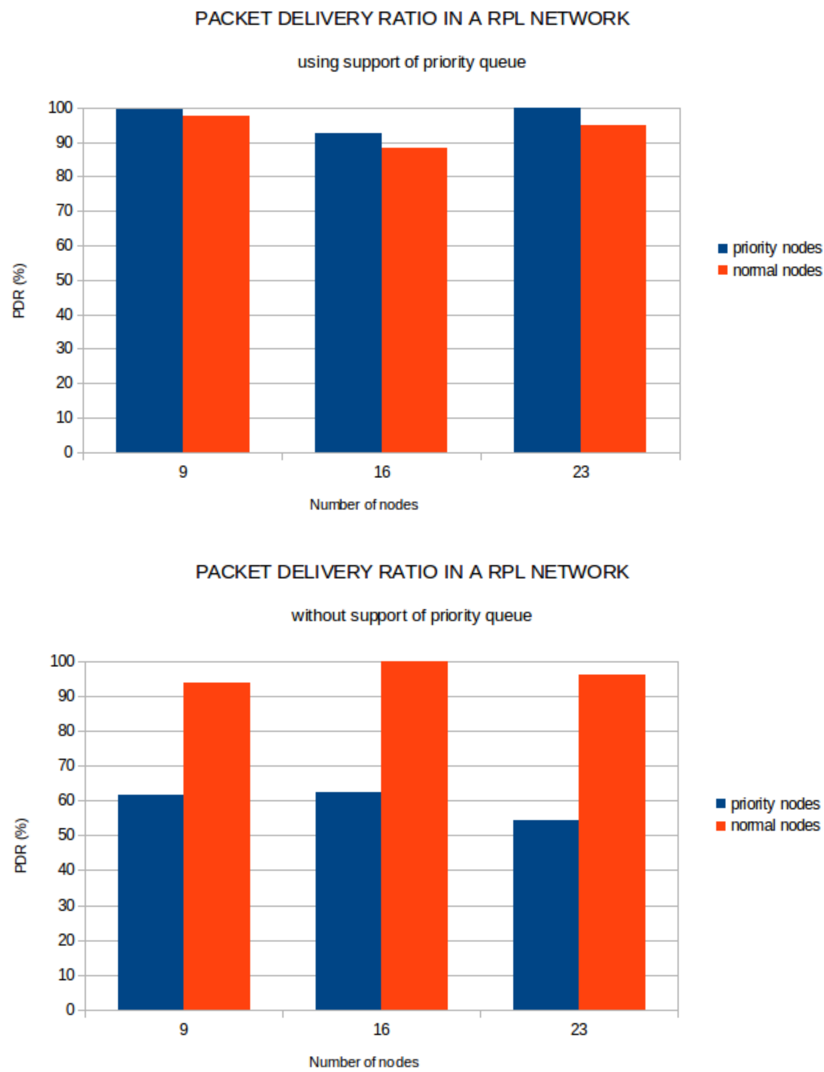

Figure 4.6: PDR in a RPL network. 17/03/2014

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

4.3.2 Latency

In order to validate the results obtained with the SFD MACRO's provided by Contiki, a few preliminary tests were carried out. The first set of measurements consisted in setting up a single-hop RPL network, with only one mote. This mote was connected to the DataLogger in the same fashion as it was explained in Chapter 3. The RPL sink was also connected to the DataLogger, so that the pulses generated when sending and receiving packets were recorded in a *.txt* file. At the same time, the code for adding the timestamps in the application and in the radio level was added in the motes' firmware, following the same reasoning as in Chapter 4.

Table 4.1 shows the results obtained in such experiment.

	Software	DataLogger
Average Latency (ms)	14.5	18.6
Standard Deviation (ms)	3	6

Table 4.1: Comparison between the two measurement techniques for latency with a one-hop RPL network

As we observe, the average difference between the two methods is of 30%, meaning an average difference of 6 ms. In this case all the latencies are in the order of a few tens of milliseconds, due to the fact that this is a small one-hop RPL network. Taking into account that, as mentioned in Chapter 3, the software provides with a resolution of ms while the resolution given by the data logger is 10ms, an difference of 6 ms is still acceptable. However, at this stage we determined that it would be convenient to find a way to have higher latency values.

On a second step, a two-hop-network was settled. This time, the mote on the second hop was connected to the DataLogger. The mote on the first hop was configured in a way that it would only send packets coming from other motes, but not its own. This allows us to measure higher latencies for the packets coming from the mote situated on the second hop (see Table 4.2).

	Software	DataLogger
Average Latency (ms)	85	90
Standard Deviation (ms)	40	37

Table 4.2: Comparison between the two measurement techniques for latency with a two-hop RPL network

This time, the average difference is of 6 ms (10%). As we expected, the error decreased because of the increase of the latency. Moreover, the average

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

difference is still within the limits of the resolution of the datalogger, so we can conclude that it is due to the lack of resolution in one of the two methods, but the other method is valid and can be implemented in a larger RPL network.

The results of applying such method in a RPL network are shown in Figure 4.8.

In the Figure we represent how latency varies with the hop-count. Each point represents a mote, that sent at least 30 packets with a packet rate of 0.5 packets per second. Blue points correspond to normal nodes while orange points are priority nodes. The presented RPL network was using ContikiMAC with a wake-up rate of 8Hz as Radio Duty Cycle Protocol.

The same experience was repeated but with a different RDC protocol: NullRDC. The results are shown in Figure 4.9.

Results obtained in 4.9 show that six motes behave abnormally: four normal motes and two priority motes. After further analysing, some malfunctioning of such motes was detected: the two priority motes only sent between 5 and 6 packets, and the normal motes present in some cases abnormally high latency values. For this reasons, those results were ignored, obtaining Figure 4.10.

4.4 Qualitative Unexpected Observations

After making the RSSI and LQI measurements, we expected to be able to set-up a stable multi-hop network. However, this did not occur as expected: several problems were faced during the time we performed the measurements. First of all, as we can conclude from 4.3.1, the weather conditions play a fundamental role in the good performance of the RPL network. As our goal was to study the performance of a certain protocol optimization, it was necessary to wait for good weather conditions to obtain coherent results.

Secondly, it was observed that when using ContikiMAC 8, even if the motes are on the same hop, the variance in the latency is very high. We suspected it was due to some malfunctioning in ContikiMAC, and it was confirmed by the results obtained with NullRDC. When the RDC protocol is NullRDC, the network becomes much more stable (see for example the low standard deviation for the hopcount in each mote), and the latency results are much more coherent with our expectations.

Furthermore, it was observed that the actual implementation of ETX in Contiki seems to favour one-hop communication. That phenomena contributes to the instability of the network, since some of the motes were located far from the sink, but they try to establish communication to the sink directly, and not via other motes that could offer better link quality. That leads us to suspect that the combination between ContikiMAC and ETX contributes to the lack of stability of our network.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

4.5 Conclusions

In terms of PDR, the algorithm prosposed by N.T Long et al. presents an improvement for the priority packets. However, in terms of latency the results are not yet very clear: both normal and priority packets present a very highly variable latency. This may be due to instabilities in the network and/or instabilities in the underlying RDC protocol.

Regarding the measurement techniques, we have proved that at this point we have enough means to measure PDR and latency correctly by means of software, with a very good accuracy. However at this point, both parameters cannot be evaluated at the same time: Collect-View does not provide us with latency measurements and when we add the MACRO's provided by Contiki we it cannot be run at the same time. The reason for that is that when the timestamps are added, the packet format defined by Collect-View (see *collect_view_data_msg* in *contiki/apps/collect-view/collect-view.h*). But besides, we intended to measure also power usage in the motes of a RPL network which was not possible so far.

All this motivated us to try to gather all the measuring techniques in one unique device.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

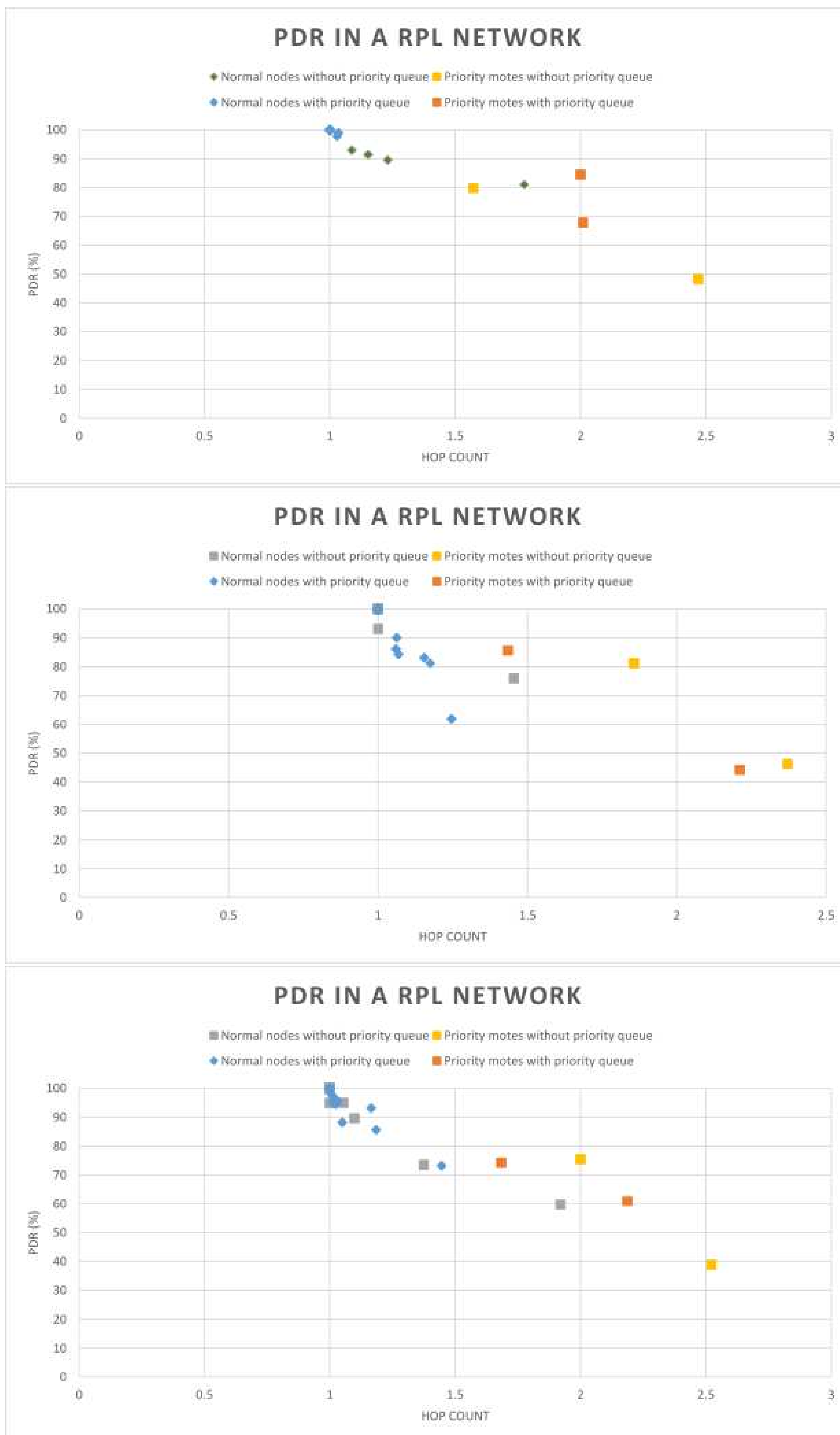

Figure 4.7: PDR vs HopCount in three RPL networks with 9, 16 and 23 motes 17/03/2014

upna
Todos los derechos reservados
Eskubide guztiak erreserbatuak dira

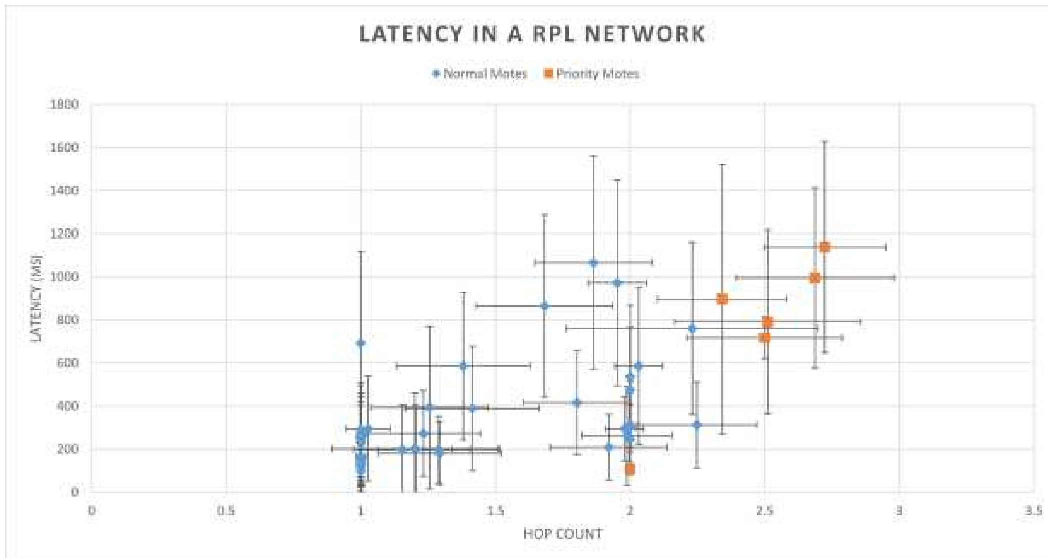

Figure 4.8: Latency measured in a RPL Network.

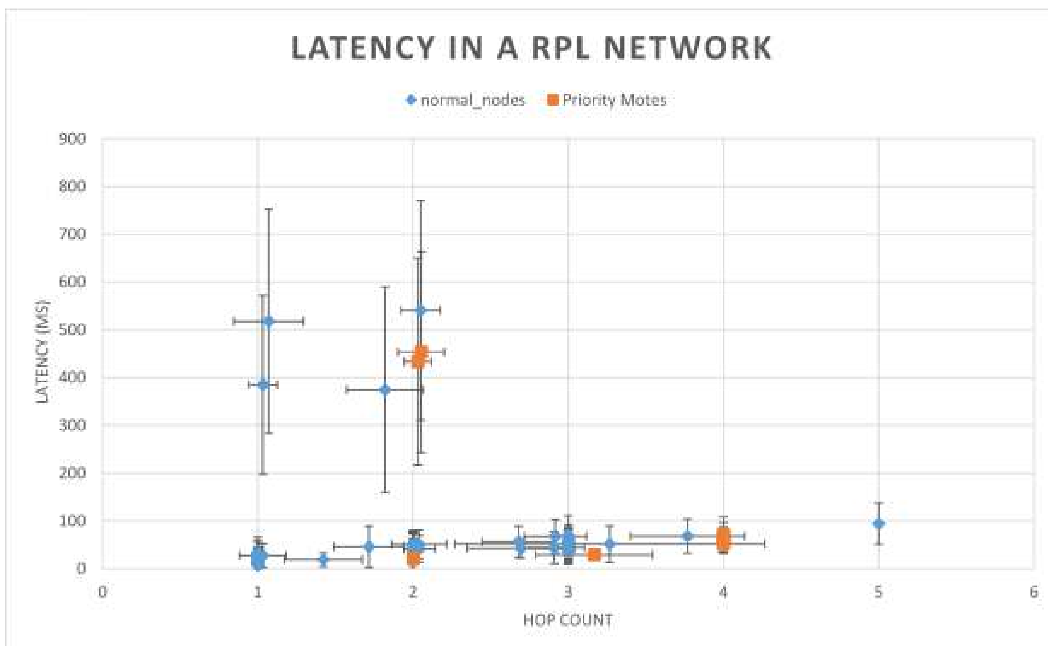

Figure 4.9: Latency measured in a RPL Network with NullRDC.

upna
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

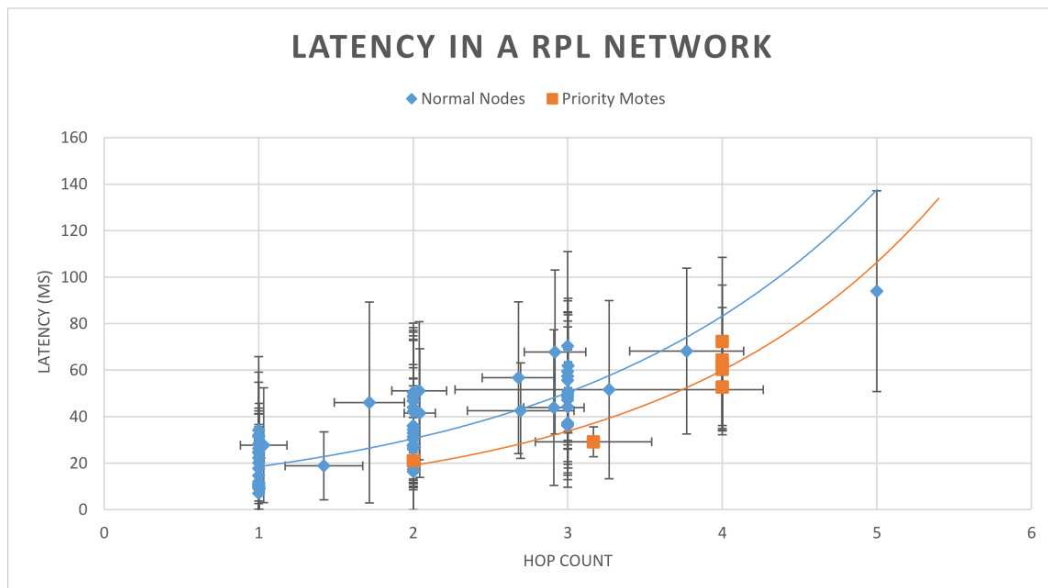

Figure 4.10: Latency measured in a RPL Network with NullRDC after removing the points generated by malfunctioning motes.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Chapter 5

Dual Network

5.1 Introduction

After all the measurements that were made without any conclusive result, we were forced to find a new way to be able to evaluate the parameters we are interested in. This new method must meet some minimum requirements:

- It should not interfere with the normal functioning of the RPL network
- It should provide means for measuring:
  - Packet Delivery Ratio
  - Latency
  - Power Usage

After studying all the features available in the Zolertia Z1 motes (see chapter 1), it was clear that they provided us with the necessary tool for such measurements: the ADC12 allows us to measure voltage, while we could connect via the GPIOs, allowing almost instantaneous communication between the two motes without requiring significant processing in the observed mote.

5.2 WSN as Data Logger

The idea is pretty simple: we set-up two networks working in parallel in different radio-channels. One of them will be the RPL network under study, while the other will record the collected data. The schema of the two parallel networks is given in Figure 5.1:

The characteritics of each one of the represented networks are shown in Table 5.1.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

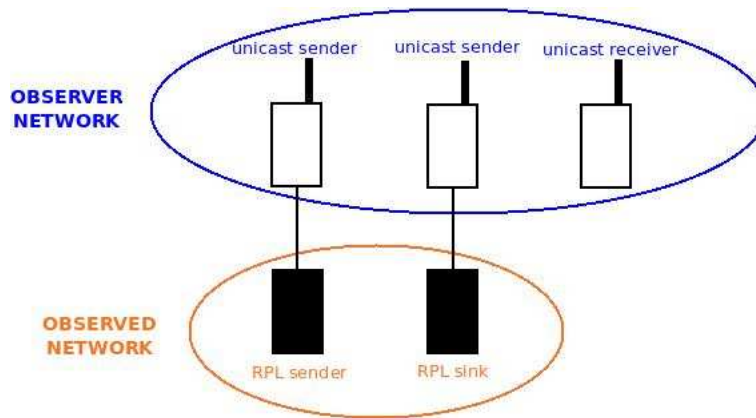

Figure 5.1: Schema of the network

	Observed Network	Observer Network
Type	RPL	Unicast
Channel	26	16
RDC Protocol	Chosen by user	NullRDC
Tx_power	Chosen by user	31
Type of Antenna	Built-in antenna	External dipole antenna

Table 5.1: Characteristics of both networks.

upna
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

5.2.1 Packet Delivery Ratio

As it was presented in Chapter 1, the msp430 has some pins that can be used as General Purpose I/O pins. In our dual-mote system, seven pins will be used to record the sequence number of the current packet. The pins are highlighted in Figure 5.2

The PDR can be calculated as follows:

1. Every time the RPL sender sends a packet, it changes the state of its output pin P1.0 and at the same time, copies the 6 least significant bits of the sequence number of the packet in the remaining 6 pins configured as GPIOs. The sequence number of the packet is also included in the payload.

2. P1.0 is read 200 times per second in the observer mote. Whenever it changes its state, the observer understands that a packet has been sent in the application layer, and reads the state of the pins that have been set as input. The bits are converted to a number, corresponding to the sequence number of the packet sent by the RPL mote.

3. The observer mote inserts that sequence number in the payload of a packet that is sent by unicast to the receiver of the observer network, which is connected to a PC via USB.

4. When the RPL sink receives a packet, the state of its P1.0 is changed, and the 6 least significant bits of the sequence number of that packet are copied to the appropriate GPIO pins.

5. When a change in P1.0 is detected, the observer of the RPL sink sends a new unicast packet containing the sequence number to the receiver of the observer network.

6. All packets arriving in the receiver of the observer network are recorded by means of a serialdump in the connected PC. The PDR can be easily calculated, by comparing the sequence numbers of the packets coming from the RPL sender and the RPL sink.

In Figure 5.2 the pins that are used for writing or reading the seqno are shown.

5.2.2 Latency

Based on the system proposed in Chapter 3, the SFD MACRO's provided by Contiki were used in the observer motes. With them, we can measure latency, considering latency as the difference of times between receiving the

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

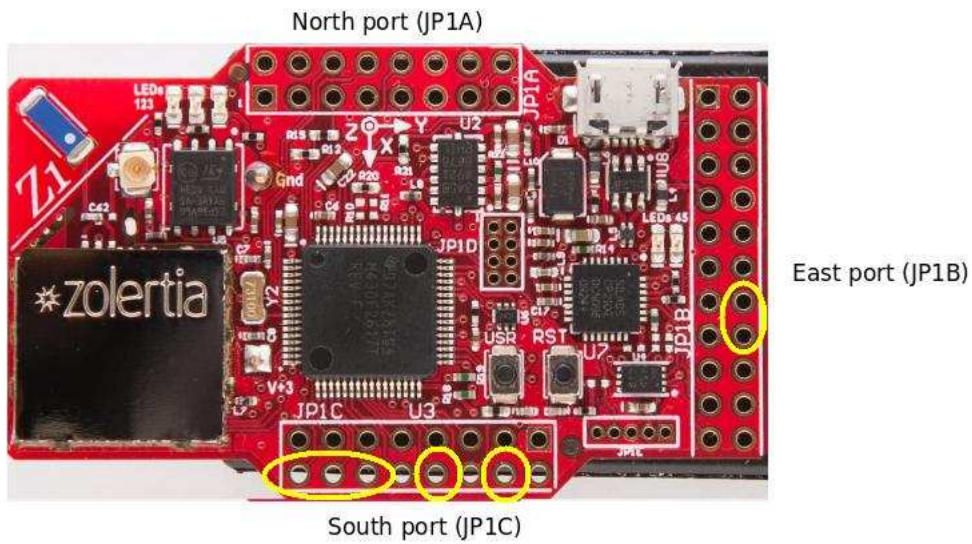

Figure 5.2: Schematic of the GPIOs used.

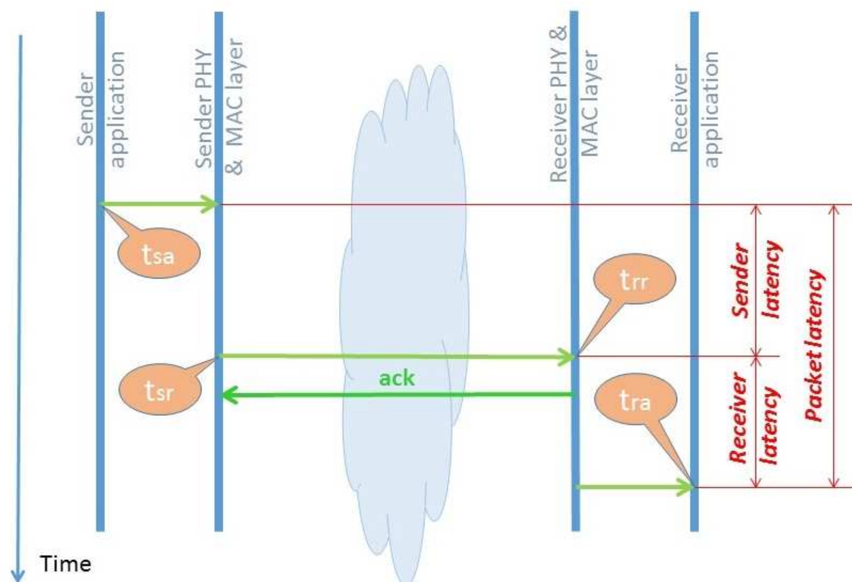

Figure 5.3: Simplified packet latency representation.

upna
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

packet in the application layer in the receiver and sending the packet in the application layer in the sender, as it is shown in Figure 5.3:

However, in the system we present, measuring latency is not so simple (see Figure 5.4):

1. The RPL sender sends a packet and changes the state of its output pin P1.0.

2. P1.0 is read 200 times per second. Whenever it changes its state, the observer understands that a packet has been sent in the application layer and that timestamp (t_{sa}) is immediately recorded.

3. The observer mote sends a new packet, including the timestamp t_{sa} in its payload. Such packet is sent through the observer MAC layer.

4. The packet is received in the observer receiver at t_{sor} in the MAC layer, and arrives to the application layer at t_{soa}. Both timestamps are recorded and printed out in the receiver mote.

5. In the RPL sink, whenever a packet is received, P1.0 changes its state. The observer understands that a packet has been received, so that its timestamp is also recorded (t_{ra}).

6. The observer of the RPL sink sends a new packet with the timestamp recorded at the moment the packet was received, t_{ra}.

7. The packet is received in the observer receiver at t_{ror} in the MAC layer, and arrives to the application layer at t_{roa}. Both timestamps are recorded and printed out in the receiver mote.

In our system, all the recorded timestamps are received in the receiver mote, that is connected to a PC via USB. Once we have all the timestamps, the latency can be calculated as:

$$Latency = t_{ra} - t_{sa} = [(t_{roa} - t_{ror}) + (t_{rr} - t_{ra})] - [(t_{soa} - t_{sor}) + (t_{sr} - t_{sa})]$$

5.2.3 Power Usage

The msp430 allows us to measure analog signals such as voltage by means of its ADC12. It can be easily configured following the example given in Contiki, in the directory */platform/z1/dev/battery-sensor.c*. In our case, it was necessary to change a few lines in the example code, since our analog signal would be an input in port P6.7:

```
#define INPUT_CHANNEL     (1 << INCH_7)
#define INPUT_REFERENCE SREF_1
#define BATTERY_MEM        ADC12MEM7
```


upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

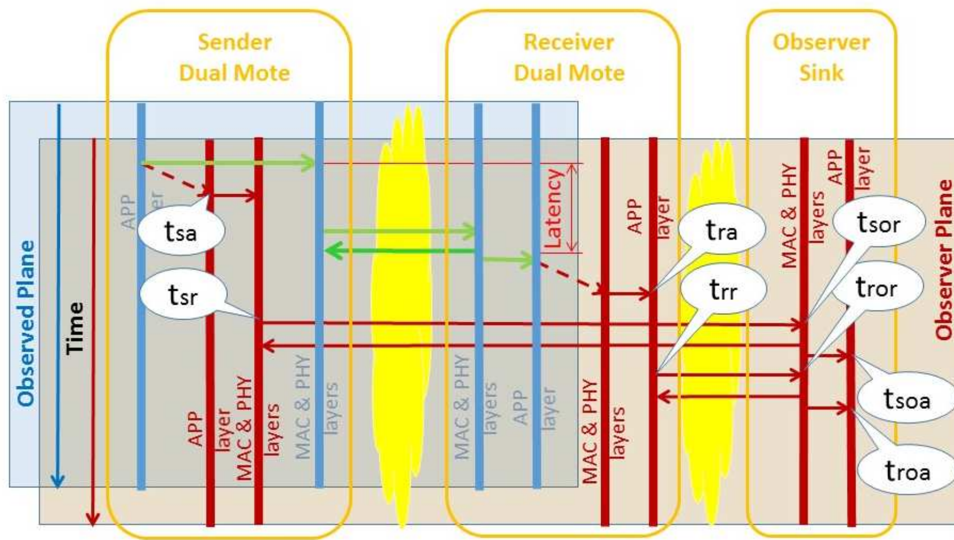

Figure 5.4: Measuring packet latency.

In order to measure the power usage in each mote, the voltage is sampled 100 times per second (by reading the register ADC12MEM7 100 times per second). The cumulative value of the samples recorded between two consecutive packets is sent in a reserved field of the packet sent by the observer mote (both when the observer is observing a RPL sender or the RPL sink). The conversion made by the ADC12 can be undone by the expression given in Chapter 1.

In Figure 5.5 the schematic of the circuit designed for measuring the power usage is shown.

At the moment of submitting the present Master's Thesis, the dual-mote system proposed in this Chapter has been tested. Several motes have been built and tested, proving that our system works (see Figure 5.6 and 5.7).

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

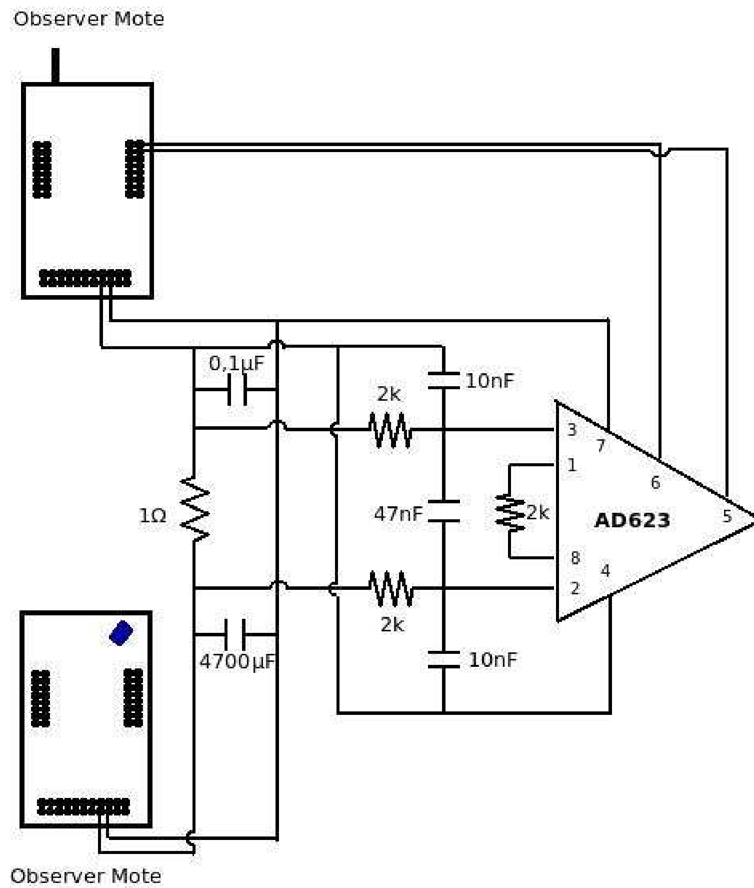

Figure 5.5: Connection diagram for measuring the power usage.

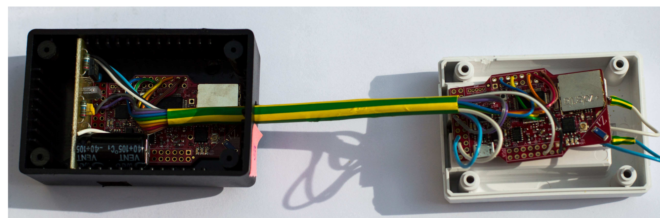

Figure 5.6: One of the dual motes built and tested.

upna
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Figure 5.7: Testing the dual-mote system.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

Chapter 6

Conclusions and Future Work

6.1 Conclusions

All the measurements techniques developed in the present Master thesis represent a worthy tool for studying the behaviour of any real WSN deployment. Such measurements are necessary for better understanding the different communication protocols for WSNs and their interactions in order to optimize such protocols, as well as for analysing in depth the performance of any proposed optimization. Such measurements could also be made by means of a simulator. However, we learned that radio communications and the interaction between processes (timing) can be much more complex than the models used in our simulator, which leads to very different results when simulating a network or when testing it in a real testbed.

While making all the measurements presented in this thesis, we realised that the behaviour of each mote differs significantly, which can interfere with the good functioning of the tested network. For avoiding unexpected behaviours due to some hardware defects, it is advisable to make measurements like those presented in Chapter 2 before setting-up any real network or building a dual-mote.

When analysing all the recorded data, we noticed that ContikiMAC and LPP seem to present some issues in their implementation in Contiki. As it was explained, such problems were only observed when the RDC protocols were tested with real motes. This may present a problem when setting-up a RPL network (see Chapter 4) since it contributes to the instability of the network. Certainly, this should be further explored. After all, ContikiMAC is the default RDC protocol for a RPL network in Contiki.

Obviously, this also affected directly to the results obtained when the optimization suggested by N.T Long et al. was studied: if ContikiMAC was used as RDC protocol, the results obtained were not conclusive. However, when we switched to NullRDC, where the radio in the mote is always on, the results confirmed our expectations: the algorithm presented an improvement

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

in the performance of the network.

6.2 Future Work

As it was mentioned in the Conclusions, ContikiMAC presents some issues that should be further explored.

As explained, the dual-mote system was tested. However, data for the RPL network has not yet been recorded due to lack of time. In the coming weeks we intend to make the same experiments as in Chapter 4, with different underlying RDC protocols, to study more in depth the interaction between the RDC protocols and RPL.

Recently, through some industrial contacts we got access to a totally different variety of radios than those we studied. These so called LOng RAnge (LORA) allow for trading dynamically range for throughput between limits we could only dream of with classical IEEE 802.15.4 radios. We intend to repeat, if possible, some of the tests described in chapter 3 with these radios in order to understand better their properties and to examine if they could be used in traditional Wireless Sensor Networks for those links that have excessive path losses or that easily get congested.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

References

[1] *Zolertia Z1 datasheet*. Obtained from: `http://zolertia.sourceforge.net/wiki/images/e/e8/Z1_RevC_Datasheet.pdf`

[2] Texas Instruments. *MSP430x2xx Family User's Guide*. December 2004-Revised July 2013. Obtained from: `http://www.ti.com/lit/ug/slau144j/slau144j.pdf`

[3] J. Hill, P. Levis, S. Madden, A. Woo, J. Polastre, C. Whitehouse, R. Szewczyk, C. Sharp, D. Gay, M. Welsh, D. Culler and E. Brewer: *TinyOS: An Operating System for Sensor Networks*, obtained from: `http://www.cs.berkeley.edu/~culler/cs294-f03/papers/tinyos.pdf`

[4] Adam Dunkels, Björn Grönvall, and Thiemo Voigt: *Contiki - a lightweight and flexible operating system for tiny networked sensors.* In Proceedings of the First IEEE Workshop on Embedded Networked Sensors (Emnets-I), Tampa, Florida, USA, November 2004.

[5] N. Tsiftes and A. Dunkels: *A Database in Every Sensor*, Proceedings of ACM SenSys 2011.

[6] Matthias Kovatsch, Simon Duquennoy and Adam Dunkels: *A Low-Power CoAP for Contiki*. Proceedings of the 8th IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS 2011). Valencia, Spain, October 2011.

[7] Silberschatz, Abraham; Cagne, Greg, Galvin, Peter Baer : *"Chapter 4 - Processes". Operating system concepts with Java (Sixth Edition ed.)*.2004. John Wiley & Sons. ISBN 0-471-48905-0.

[8] Adam Dunkels, Oliver Schmidt, Thiemo Voigt, Muneeb Ali: *Protothreads: Simplifying Event-Driven Programming of Memory-Constrained Embedded Systems*. Proceedings of SenSys'06, November 1-3,2006, Boulder, Colorado, USA.

[9] A.Dunkels, O.Schmidt, and T.Voigt: *Using protothreads for sensor node programming*. In Proceedings of the Workshop on Real-World

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

Wireless Sensor Networks (REAL WSN'05), Stockholm, Sweden, June 2005.

[10] G.U. Gamm, M.Sippel, M. Kostic, L.M. Reindl: *Low power wake-up receiver for wireless sensor nodes*. Proceedings of Sixth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2010.

[11] L. Lamport: *Time, Clocks, and the Ordering of Events in a Distributed System*, Communications of the ACM, Vol.21.7, July 1978.

[12] C. G. Cocan: *Analysis and design of synchronization in single- and multi-channel Medium Access Control (MAC) protocols for Wireless Sensor Networks: a study based on simulation and lab tests*, Master Thesis, Vrije Universiteit Brussel, ETRO, 2011.

[13] Dunkels A: *The ContikiMAC radio duty cycling protocol*, Technical Report T2011:13, Swedish Institute of Computer Science, December 2011.

[14] R. Musaloiu-Elefteri, C. Liang, A. Terzis. Koala: *Low power probing - Ultra-Low Power Data Retrieval in Wireless Sensor Networks*, Proceedings of the 7th international conference on Information processing in sensor networks, (IPSN 2008).

[15] M. Buettner, G. V. Yee, E. Anderson, R. Han: *X-MAC: A Short Preamble MAC Protocol for Duty-Cycled Wireless Sensor Networks*, Department of Computer Science University of Colorado, USA, 2006.

[16] A. Dunkels, F.Österlind and Zhitao He: *An Adaptive Communication Architecture for Wireless Sensor Networks*, SenSysâĂŹ07, Sydney, Australia, 2007.

[17] Internet Engineering Task Force (IETF) Request for Comments: 6550: *RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks*. ISSN: 2070-1721, 2012.

[18] D. S. J. De Couto, D. Aguayo, J. Bicket and R. Morris: *A High-Throughput Path Metric for Multi-Hop Wireless Routing*, MobiCom'03, San Diego, 2003.

[19] K. Benkič, M. Malajner, P. Planinšič, Ž. Čučej: *Using RSSI value for distance estimation in Wireless sensor networks based on ZigBee*. In Proceedings of 15th International Conference Systems, Signals and Image Processing, IWSSIP. Bratislava, Slovakia, 2008.

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

[20] Texas Instruments 2.4 GHz IEEE 802.15.4 / ZigBee-Ready RF Transceiver Obtained from: http://www.ti.com/lit/ds/symlink/cc2420.pdf

[21] Ana Bildea, Olivier Alphand, Andrzej Duda: *Link Quality Metrics in Large Scale Indoor Wireless Sensor Networks*. Published in "15èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications (AlgoTel). Pornic, France, 2013.

[22] M. M. Holland, R. G. Aures and W. B. Heinzelman: *Experimental Investigation of Radio Performance in Wireless Sensor Networks*. Published by the Department of Electrical and Computer Engineering, University of Rochester, Rochester NY. Obtained from: www.ece.rochester.edu/projects/wcng/papers/conference/holland_secon06.pdf.

[23] T.Liu and A-E. Cerpa: *Foresee (4C): Wireless Link Prediction using Link Features*, published by Electrical Engineering and Computer Science, University of California - Merced. Obtained from: http://andes.ucmerced.edu/papers/Liu11a.pdf

[24] Robert Lübke, Peter Büschel, Daniel Schuster, Alexander Schill: *Measuring Accuracy and Performance of Network Emulators*. Proceedings of the IEEE BlackSeaCom, Chisinau, Moldavia, May 2014.

[25] Kieu-Ha Phung, Huy Tran, Jelmer Tiete, Lan Tran, Kris Steenhaut: *Low-Overhead Time Synchronization for Schedule-Based Multi-Channel Wireless Sensor Networks*. Proceedings LANMAN Brussels, Belgium, 2013.

[26] Bart Lemmens, Kris Steenhaut, Peter Ruckebusch, Ingrid Moerman, Ann Nowé: *Network-Wide Synchronization in Wireless Sensor Networks*. IEEE 19th Symposium on Communications and Vehicular Technology in the Benelux (SCVT), 2012.

[27] Cristina Gabriela Cocan: *Analysis and design of synchronization in single and multi-channel medium access control protocols for wireless sensor networks: a study based on simulation and lab tests*. Master thesis, Faculty of Engineering, Vrije Universiteit Brussel, Belgium, 2012.

[28] Dunkels, A., Osterlind, F., Tsiftes, N., He, Z.: *Software-based On-line Energy Estimation for Sensor Nodes*. In: IEEE EmNets, 2007.

[29] Philipp Hurni, Benjamin Nyffenegger, Torsten Braun, Anton Hergenroeder: *On the accuracy of software-based energy estimation techniques*. Proceedings EWSN 2011, Bonn, Germany.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

[30] Hergenröder, A., Wilke, J., Meier, D.: *Distributed Energy Measurements in WSN Testbeds with a Sensor Node Management Device (SNMD)*. Proceedings ARCS, Hannover, Germany, 2010.

[31] Markus Buschho, Christian Ghunter, and Olaf Spinczyk: *MIMOSA, a Highly Sensitive and Accurate Power Measurement Technique for Low-Power Systems*. Real-WSN, Como, 2013.

[32] Datasheet: SmartRF CC2420 2.4 GHz IEEE 802.15.4 ZigBee-ready RF Transceiver. Obtained from `www.ti.com/product/cc2420` on February 12, 2014.

[33] Dunkels A: *The ContikiMAC radio duty cycling protocol*. Technical Report T2011:13, Swedish Institute of Computer Science, December 2011.

[34] Michael Buettner, Gary V. Yee, Eric Anderson, Richard Han: *X-MAC: A Short Preamble MAC Protocol for Duty-Cycled Wireless Sensor Networks*. Department of Computer Science University of Colorado, USA, 2006.

[35] R. Musaloiu-Elefteri, C. Liang, A. Terzis. Koala: *Low power probing - Ultra-Low Power Data Retrieval in Wireless Sensor Networks*. IPSN 2008.

[36] Christopher Pinola: *Evaluating the Performance of Synchronous and Asynchronous Media Access Control Protocols in the Contiki Operating System*. Bachelor thesis, Worcester Polytechnic Institute, UK, 2012.

[37] Nguyen Thanh Long, Marie-Paule Uwase, Jacques Tiberghien, Kris Steenhaut: *QoS-aware Cross-layer Mechanism for Multiple Instances RPL*. Proceedings International Conference on Advanced Technologies for Communications (ATC), Ho Chi Minh City, Vietnam, 2013.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Appendix A

Contiki Code

A.1 receiver-latency.c

```
/**
 * \file
 *            Unicast receiving test program
 *            Accepts messages from everybody
 * \author
 *          Marie-Paule Uwase
 *          August 13, 2012
 *          Tested and modified on September 8, 2012. Includes a useless but necessary timer!
 *          Writes with a fixed format in the serialdump to facilitate transfer of data to excel.
 *          Ack power is 0dBm, regardless of the sender settings.
 *
 *          Modified by Maite Bezunartea for allowing latency measurements
 *             with datalogger and with SFD MACRO'S
 *          May, 2014
 *
 */

// imported libraries.

#include <stdio.h>
#include "contiki.h"
#include "dev/i2cmaster.h"
#include "dev/tmp102.h"
#include "net/rime.h"
#include "dev/button-sensor.h"
#include "dev/cc2420.h"          // To use RSSI and LQI variables
#include "sys/energest.h"        // Module to measure ON time of hardware component
#include "dev/leds.h"
#include "dev/relay-phidget.h" // to access ADC7 pin on JP1A connector.


/* In Rime communicating nodes must agree on a 16 bit virtual
 * channel number. For each virtual channel one can define

 * the Rime modules for communicating over that channel.
 * Channel numbers < 128 are reserved by the system.
 */
uint16_t channel =        133;

// sender power
// possible values =  0dBm = 31;  -1dBm = 27;  -3dBm = 23;  -5dBm = 19;
//                   -7dBm = 15; -10dBm = 11; -15dBm =  7; -25dBM =  3;
uint8_t power = 7;

// message counters
static uint16_t received = 0 ;

//variables for channel quality
int8_t lqi_val ;
int16_t rssi_val ;

/* Altough it doesn't have anything to do every 10 seconds this timer is needed in order
 * for the receiver to work ???
 */
#define TMP102_READ_INTERVAL (CLOCK_SECOND*10)
static struct etimer et;
```


upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

```
// Writes a title on the console

PROCESS(temp_process, "receiving numbered messages");
AUTOSTART_PROCESSES(&temp_process);

struct testmsg {
        uint16_t seqno;
        uint8_t power;
        uint32_t clock;
        uint32_t cpu;
        uint32_t lpm;
        uint32_t transmit;
        uint32_t listen;
        uint8_t padding[13];
        uint16_t rtimer_ticks;
        //uint8_t rtimer_ticks2;
        uint16_t timestamp;
        //uint8_t timestamp2;

};


// This is the receiver function

recv_uc(struct unicast_conn *c, const rimeaddr_t *from)
{
        unsigned long rtime; // received time (long)
          rtimer_clock_t now = RTIMER_NOW();            //received time (for the latency)
          struct testmsg msg;
          memcpy(&msg, packetbuf_dataptr(), sizeof(msg));


        //     switch 0ff P6.7 to mark the initial send time in external power recording
        relay_off();
        /*leds_on(LEDS_BLUE);*/

        //  get the rssi and lqi values for the received packet
        rssi_val = ( cc2420_last_rssi ) - 45;
        lqi_val = cc2420_last_correlation;

        received ++;
        rtime=clock_time();
        uint16_t timestamp = packetbuf_attr(PACKETBUF_ATTR_TIMESTAMP);
        /*output format:rtimer-receiver|timestamp-receiver|received_time|sender|
        received|seqno|tx_power|sent_time|cpu|lpm|transmit|listen|timestamp-sender
        |latency-sender|RSSI|LQI*/

        printf("%u,%u,%lu,%u,%u,%u,%u,%lu,%lu,%lu,%lu,%lu,%u,%u,%u,%5d,%5d\n",
                                now, timestamp,rtime,from->u8[0], received,
                                msg.seqno,msg.power,msg.clock,msg.cpu,msg.lpm,
                                msg.transmit,msg.listen,msg.timestamp,
                                msg.rtimer_ticks,(msg.timestamp-msg.rtimer_ticks),
                                rssi_val,lqi_val);

/*      printf("%u,%lu,%u,%u,%u,%u,%lu,%lu,%lu,%lu,%lu,%u,%5d,%5d\n",
                        (now- timestamp),rtime,from->u8[0], received,
                        msg.seqno,msg.power,msg.clock,msg.cpu,msg.lpm,msg.transmit,
                        msg.listen,(msg.timestamp-msg.rtimer_ticks),
                        rssi_val,lqi_val);*/


        //     switch on P6.7 to mark the end of receiving time in external power recording
        relay_on();
        /*leds_off(LEDS_BLUE);*/
}

static const struct unicast_callbacks unicast_callbacks = {recv_uc};

static struct unicast_conn uc;

PROCESS_THREAD(temp_process, ev, data)
{
        PROCESS_BEGIN();

        // Initialisation of the useless but needed timer ???

        tmp102_init();

        unicast_open(&uc, channel, &unicast_callbacks);

        // adjust power
        cc2420_set_txpower(power);

        // Enables output pin 7

        relay_enable(7);
        relay_on();          // Active low detector!
```


upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

```
        PROCESS_END();
}
```

A.2 sender-latency.c

```
/**
 * \file
 *         Unicast sending test program
 * \author
 *         Marie-Paule Uwase
 *         August 7, 2012
 *         Tested and modified on September 8, 2012.
 *
 *         Send address is directly read from rime.
 *
 *         Modified by Maite Bezunartea for allowing latency measurements
 *            with datalogger and with SFD MACRO'S
 *         May, 2014
 */

// imported libraries.

#include <stdio.h>
#include <string.h>
#include <ctype.h>

#include "contiki.h"
#include "dev/i2cmaster.h"
#include "dev/tmp102.h"
#include "net/rime.h"
#include "dev/button-sensor.h"
#include "dev/cc2420.h"   //To use RSSI and LQI variables
#include "sys/energest.h" // Module to measure ON time of hardware component
#include "dev/leds.h"
#include "dev/relay-phidget.h" // to access ADC7 pin on JP1A connector.

/* In Rime communicating nodes must agree on a 16 bit virtual
 * channel number. For each virtual channel one can define
 * the Rime modules for communicating over that channel.
 * Channel numbers < 128 are reserved by the system.
 */
uint16_t channel =        133;   //133

// sender mote id
uint8_t sender;           // set automatically

// sender power
// possible values =  0dBm = 31;  -1dBm = 27;  -3dBm = 23;  -5dBm = 19;
//                   -7dBm = 15; -10dBm = 11; -15dBm =  7; -25dBM =  3;
uint8_t power = 7;

// receiver mote id
uint8_t receiver = 109; //let's put ID for the actual node in cooja

// message counters
uint16_t send = 0 ;

/* The sender will send a message every second
 * the timer of the temperature sensor is misused for this purpose
 * because it exists and does what is needed
 */
#define MAX_SEND_INTERVAL CLOCK_SECOND  * 2
#define SEND_INTERVAL ((MAX_SEND_INTERVAL +random_rand()) % MAX_SEND_INTERVAL)
//#define SEND_INTERVAL (CLOCK_SECOND)

/* Data structure of messages sent from sender
 *
 */
struct testmsg {                  //FORMAT OF THE PACKET SEND
uint16_t seqno;
uint8_t power;
uint32_t clock;
uint32_t cpu;
uint32_t lpm;
uint32_t transmit;
uint32_t listen;
uint8_t padding[13];
uint16_t rtimer_ticks;
uint16_t timestamp;
};

uint16_t seqno=0;


PROCESS(temp_process, "Sending numbered messages");
```


upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

```
AUTOSTART_PROCESSES(&temp_process);

/*---------------------------------------------------------------------------
 * Receiver function which seems necessary to operate the sender with a normal MAC
 *---------------------------------------------------------------------------*/

static void
recv_uc(struct unicast_conn *c, const rimeaddr_t *from)
{
        printf("unicast message received from %d.%d\n",
                        from->u8[0], from->u8[1]);
}

static const struct unicast_callbacks unicast_callbacks = {recv_uc};
static struct unicast_conn uc;

/*---------------------------------------------------------------------------*/
void
packet_send(void)
{

        static unsigned long last_cpu, last_lpm, last_transmit, last_listen;

        rimeaddr_t destination;
        seqno++;
        // define the destination address
        destination.u8[0] = receiver;
        destination.u8[1] = 0;


        /*Set sender's info*/
        energest_flush();

        /*input format:seqno/power/clock/cpu/lpm/transmit/listen/padding/rtimer_ticks/timestamp*/
    struct testmsg msg;
    msg.seqno=seqno;

        /*Set general info*/

        msg.seqno;
        msg.power=power;
        msg.clock= clock_time();
        msg.cpu=energest_type_time(ENERGEST_TYPE_CPU) - last_cpu;
        msg.lpm = energest_type_time(ENERGEST_TYPE_LPM) - last_lpm;
        msg.transmit = energest_type_time(ENERGEST_TYPE_TRANSMIT) - last_transmit;
        msg.listen = energest_type_time(ENERGEST_TYPE_LISTEN) - last_listen;
        msg.padding[0]=0;

        msg.padding[1]=0;
        msg.padding[2]=0;
        msg.padding[3]=0;
        msg.padding[4]=0;
        msg.padding[5]=0;
        msg.padding[6]=0;
        msg.padding[7]=0;
        msg.padding[8]=0;
        msg.padding[9]=0;
        msg.padding[10]=0;

        msg.padding[11]=5;
        msg.padding[12]=7;
        msg.rtimer_ticks = RTIMER_NOW();
        msg.timestamp = 0;


        /* Make sure that the values are within 16 bits. If they are larger,
             we scale them down to fit into 16 bits. */
                while(msg.cpu >= 65536ul || msg.lpm >= 65536ul ||
                                msg.transmit >= 65536ul || msg.listen >= 65536ul) {
                        msg.cpu /= 2;
                        msg.lpm /= 2;
                        msg.transmit /= 2;
                        msg.listen /= 2;
                }
                last_cpu = energest_type_time(ENERGEST_TYPE_CPU);
                last_lpm = energest_type_time(ENERGEST_TYPE_LPM);
                last_transmit = energest_type_time(ENERGEST_TYPE_TRANSMIT);
                last_listen = energest_type_time(ENERGEST_TYPE_LISTEN);


        /*Avoid sending to itself*/
        if(!rimeaddr_cmp(&destination, &rimeaddr_node_addr))
        {
                packetbuf_copyfrom(&msg, sizeof(msg));
            packetbuf_set_attr(PACKETBUF_ATTR_PACKET_TYPE,
                                PACKETBUF_ATTR_PACKET_TYPE_TIMESTAMP);
        //printf(" Packet sent is %s\n",payload);
        printf(" size of payload %u\n",sizeof(msg));
```


upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

```
                unicast_send(&uc, &destination);
        }

}
/*---------------------------------------------------------------------------
 * Send packets with all needed statistic information
 *---------------------------------------------------------------------------*/

/*---------------------------------------------------------------------------*/
static struct etimer et;

// Sending thread

PROCESS_THREAD(temp_process, ev, data)
{
        PROCESS_BEGIN();


        // adjust power
        cc2420_set_txpower(power);

        tmp102_init();
        unicast_open(&uc, channel, &unicast_callbacks);
        relay_enable(7);
        relay_on();        // The pulse shaper circuit is active low.


        // infinite loop
        while(1)
        {
                //    wait for the SEND_INTERVAL time
                etimer_set(&et, SEND_INTERVAL);
                PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));

                //    switch off P6.7 to mark the initial send time in external power recording
                relay_off();
                /*leds_on(LEDS_BLUE);*/

                packet_send();

                //    switch on P6.7 to end the pulse corresponding to sending in external power recording
                relay_on();
                /*leds_off(LEDS_BLUE);*/

        }
        PROCESS_END();
}
```

A.3 receiver-observed.c

```
/**
 * \file
 *
 *      Test Software used for the Dual-Mote system
 *      Application for the receiver of the observed network.
 *      It sets the GPIOs as outputs
 *
 * \author
 *          Maite Bezunartea
 *          June, 2014
 *
 */

// imported libraries.

#include <stdio.h>
#include "contiki.h"
#include "dev/i2cmaster.h"
#include "dev/tmp102.h"
#include "net/rime.h"
#include "dev/leds.h"


/* In Rime communicating nodes must agree on a 16 bit virtual
 * channel number. For each virtual channel one can define

 * the Rime modules for communicating over that channel.
 * Channel numbers < 128 are reserved by the system.
 */
uint16_t channel =       133;

// sender power
// possible values =  0dBm = 31;  -1dBm = 27;  -3dBm = 23;  -5dBm = 19;
//                   -7dBm = 15; -10dBm = 11; -15dBm =  7; -25dBM =  3;
```


upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

```
uint8_t power = 31;              //This value can be changed by the user

// message counters
static uint16_t received = 0 ;

#define TMP102_READ_INTERVAL (CLOCK_SECOND*10)
static struct etimer et;


// Writes a title on the console

PROCESS(temp_process, "receiving numbered messages, observed network");
AUTOSTART_PROCESSES(&temp_process);

struct testmsg {
        uint16_t seqno_rpl;
        uint8_t seqno_unicast;
        uint32_t energy;
        uint32_t timestamp_app;
        uint16_t counter_ADC;
        uint16_t timestamp_mac;

};

/*---------------------------------------------------------------------------
 * SETTING THE GPIOS
 *---------------------------------------------------------------------------*/

void
GPIOS_init(void)
{
        //configuring pin as GPIO

        P1SEL &= ~0xC1;                 //GPIO P1.0 P1.6 P1.7
        P2SEL &= ~0x18;                 //GPIO P2.3 P2.4
        P4SEL &= ~0x0D;                 //GPIO P4.0 P4.2 P4.3

        //configuring pin as OUTPUT

        P1DIR |= 0xC1;
        P2DIR |= 0x18;
        P4DIR |= 0x0D;


}

void
clear_GPIOS(void)
{
//function for clearing all output pins
        P1OUT &= ~BIT6;         //P1.6 AS 0
        P1OUT &= ~BIT7;         //P1.7 AS 0

        P2OUT &= ~BIT3;         //P2.3 AS 0
        P2OUT &= ~BIT4;         //P2.4 AS 0

        P4OUT &= ~BIT0;         //P4.0 AS 0
        P4OUT &= ~BIT2;         //P4.2 AS 0
        P4OUT &= ~BIT3;         //P4.3 AS 0

}


int
read_GPIOS(void)
{
        //reading the value in each pin (unconvert from bits to integer)
        int number=0;

        int flag=(P1IN & BIT0);
    printf(" flag %u \n",flag);

        if (P1IN & BIT6)
                number=number+1;

        if (P1IN & BIT7)
                number=number+2;

        if (P2IN & BIT3)
                number=number+4;

        if (P4IN & BIT0)
                number=number+8;

        if (P4IN & BIT2)
                number=number+16;

        if (P4IN & BIT3)
                number=number+32;
```


upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

```
        return number;
}
}

// This is the receiver function

recv_uc(struct unicast_conn *c, const rimeaddr_t *from)
{
        static int seqno_bits[6];                       //here we will save the value of the seqno in bits

        unsigned long rtime; // received time (long)
          struct testmsg msg;
          memcpy(&msg, packetbuf_dataptr(), sizeof(msg));


        received ++;
        rtime=RTIMER_NOW();              //received time (for the latency)
        /*output format:rtimer-receiver|sender|received|seqno|ENERGY|msg timestamp|cpu|lpm|
         transmit|listen|timestamp-sender|rtimer-sender|RSSI|LQI*/

        printf("%lu,%u,%u,%u,%u,%lu,%lu,%lu\n",
                        rtime,from->u8[0], received,
                        msg.seqno_rpl,msg.seqno_unicast,msg.energy,msg.timestamp_app,msg.timestamp_mac);
        P1OUT ^= BIT0;          // change state of P1.0 to indicate a packet was sent

        //convert seqno into bits
        int i;
        for (i = 0; i < 6; i++) {
                seqno_bits[i] = msg.seqno_rpl & (1 << i) ? 1 : 0;
        }               //least significant bit in seqno_bits[0]


        if ( seqno_bits[0]==1 )
                P1OUT |= BIT6;          //  write a 1 in P1.6
        if ( seqno_bits[1]==1 )
                P1OUT |= BIT7;          //  write a 1 in P1.7
        if ( seqno_bits[2]==1 )
                P2OUT |= BIT3;          //  write a 1 in P2.3
        if ( seqno_bits[3]==1 )
                //P2OUT |= BIT4;        //  write a 1 in P2.4
                P4OUT |= BIT0;          //  write a 1 in P4.0

        if ( seqno_bits[4]==1 )
                //P4OUT |= BIT0;        //  write a 1 in P4.0
                P4OUT |= BIT2;          //  write a 1 in P4.2

        if ( seqno_bits[5]==1 )
                //P4OUT |= BIT2;        //  write a 1 in P4.2
        //if ( seqno_bits[6]==1 )
                P4OUT |= BIT3;          //  write a 1 in P4.3

    printf(" READ GPIOS %u \n",read_GPIOS());

        clear_GPIOS();          //after a packet was sent, we clear the state of the pins

}

static const struct unicast_callbacks unicast_callbacks = {recv_uc};

static struct unicast_conn uc;

PROCESS_THREAD(temp_process, ev, data)
{
        PROCESS_BEGIN();

        // Initialisation of the useless but needed timer ???

        tmp102_init();
        GPIOS_init();

        unicast_open(&uc, channel, &unicast_callbacks);

        // adjust power
        cc2420_set_txpower(power);

        PROCESS_END();
}
```

A.4 sender-observed.c

```
/**
 * \file
 *
```


upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

```
 *       Test Software used for the Dual-Mote system
 *       Application for the receiver of the observed network.
 *       It sets the GPIOs as outputs
 *
 * \author
 *           Maite Bezunartea
 *           June, 2014
 *
 */

// imported libraries.

#include <stdio.h>
#include <string.h>
#include <ctype.h>

#include "contiki.h"
#include "dev/i2cmaster.h"
#include "dev/tmp102.h"
#include "net/rime.h"
#include "dev/leds.h"
#include <msp430.h>
#include <legacymsp430.h>


/* In Rime communicating nodes must agree on a 16 bit virtual
 * channel number. For each virtual channel one can define
 * the Rime modules for communicating over that channel.
 * Channel numbers < 128 are reserved by the system.
 */
uint16_t channel =        133;
// sender mote id
uint8_t sender;           // set automatically

// sender power
// possible values =  0dBm = 31;  -1dBm = 27;  -3dBm = 23;  -5dBm = 19;
//                   -7dBm = 15; -10dBm = 11; -15dBm =  7; -25dBM =  3;
uint8_t power = 31;                //can be set by the user

// receiver mote id
uint8_t receiver = 109; //ID for the receiver mote of the observed network

// message counters
uint16_t send = 0 ;

/* The sender will send a message every second
 * the timer of the temperature sensor is misused for this purpose
 * because it exists and does what is needed
 */
#define MAX_SEND_INTERVAL CLOCK_SECOND  * 2
#define SEND_INTERVAL (MAX_SEND_INTERVAL +random_rand()) % MAX_SEND_INTERVAL)

/* Data structure of messages sent from sender
 *
 */
struct testmsg {
        uint16_t seqno_rpl;
        uint8_t seqno_unicast;
        uint32_t energy;
        uint32_t timestamp_app;
        uint16_t counter_ADC;
        uint16_t timestamp_mac;
};


uint16_t seqno=0;
uint16_t ADCResult=0;


PROCESS(temp_process, "Sending numbered messages");
AUTOSTART_PROCESSES(&temp_process);

/*---------------------------------------------------------------------------
 * SETTING THE GPIOS
 *---------------------------------------------------------------------------*/

void
GPIOS_init(void)
{
        //configuring pin as GPIO

        P1SEL &= ~0xC1;                 //GPIO P1.0  P1.6  P1.7
        P2SEL &= ~0x18;                 //GPIO P2.3  P2.4
        P4SEL &= ~0x0D;                 //GPIO P4.0  P4.2  P4.3

        //configuring pin as OUTPUT

        P1DIR |= 0xC1;
        P2DIR |= 0x18;
```


upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

```
        P4DIR |= 0x0D;

}

void
clear_GPIOS(void)
{
//clear all output pins
        P1OUT &= ~BIT6;         //P1.6 AS 0
        P1OUT &= ~BIT7;         //P1.7 AS 0

        P2OUT &= ~BIT3;         //P2.3 AS 0
        P2OUT &= ~BIT4;         //P2.4 AS 0

        P4OUT &= ~BIT0;         //P4.0 AS 0
        P4OUT &= ~BIT2;         //P4.2 AS 0
        P4OUT &= ~BIT3;         //P4.3 AS 0

}

int
read_GPIOS(void)
{
        //reading the value in each pin (unconvert from bits to integer)
        int number=0;

        int flag=(P1IN & BIT0);
        printf(" flag %u \n",flag);

        if (P1IN & BIT6)
                number=number+1;

        if (P1IN & BIT7)
                number=number+2;

        if (P2IN & BIT3)
                number=number+4;

        if (P4IN & BIT0)
                number=number+8;

        if (P4IN & BIT2)
                number=number+16;

        if (P4IN & BIT3)
                number=number+32;


        return number;
}


/*---------------------------------------------------------------------------
 * Receiver function which seems necessary to operate the sender with a normal MAC
 *---------------------------------------------------------------------------*/

static void
recv_uc(struct unicast_conn *c, const rimeaddr_t *from)
{
        printf("unicast message received from %d.%d\n",
                        from->u8[0], from->u8[1]);
}

static const struct unicast_callbacks unicast_callbacks = {recv_uc};
static struct unicast_conn uc;

/*---------------------------------------------------------------------------*/
void
packet_send(void)
{

        rimeaddr_t destination;
        seqno++;
        // define the destination address
        destination.u8[0] = receiver;
        destination.u8[1] = 0;

        static int seqno_bits[6];       //here we will save the value of the seqno in bits


        /*Set sender's info*/
        energest_flush();
    struct testmsg msg;
        /*Set general info*/
        msg.seqno_unicast=0;
        msg.seqno_rpl=seqno;            //this value has to be read from the gpios
        msg.energy=0;
```


upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

```
msg.timestamp_app= clock_time();
msg.counter_ADC=0;


//convert seqno into bits
int i;
for (i = 0; i < 6; i++) {
        seqno_bits[i] = seqno & (1 << i) ? 1 : 0;

}               //least significant bit in seqno_bits[0]

/*Avoid sending to itself*/
if(!rimeaddr_cmp(&destination, &rimeaddr_node_addr))
{
        packetbuf_copyfrom(&msg, sizeof(msg));
    packetbuf_set_attr(PACKETBUF_ATTR_PACKET_TYPE,
                    PACKETBUF_ATTR_PACKET_TYPE_TIMESTAMP);
//printf(" Packet sent is %s\n",payload);
printf(" size of payload %u\n",sizeof(msg));

        unicast_send(&uc, &destination);
        /*SET FLAG P1.0*/
        P1OUT ^= BIT0;          // change state of P1.0

        if ( seqno_bits[0]==1 )
                        P1OUT |= BIT6;          //  write a 1 in P1.6
                if ( seqno_bits[1]==1 )
                        P1OUT |= BIT7;          //  write a 1 in P1.7
                if ( seqno_bits[2]==1 )
                        P2OUT |= BIT3;          //  write a 1 in P2.3
                if ( seqno_bits[3]==1 )
                        //P2OUT |= BIT4;        //  write a 1 in P2.4
                        P4OUT |= BIT0;          //  write a 1 in P4.0

                if ( seqno_bits[4]==1 )
                        //P4OUT |= BIT0;        //  write a 1 in P4.0
                        P4OUT |= BIT2;          //  write a 1 in P4.2

                if ( seqno_bits[5]==1 )
                        //P4OUT |= BIT2;          //  write a 1 in P4.2
                //if ( seqno_bits[6]==1 )
                        P4OUT |= BIT3;          //  write a 1 in P4.3

printf(" READ_GPIOS %u \n",read_GPIOS());


}


}
/*---------------------------------------------------------------------------
 * Send packets with all needed statistic information
 *---------------------------------------------------------------------------*/

static struct etimer et;

// Sending thread

PROCESS_THREAD(temp_process, ev, data)
{
        PROCESS_BEGIN();

        // adjust power
        cc2420_set_txpower(power);

        tmp102_init();
        GPIOS_init();
        unicast_open(&uc, channel, &unicast_callbacks);
        // infinite loop
        while(1)
        {
                //      wait for the SEND_INTERVAL time
                etimer_set(&et, MAX_SEND_INTERVAL);
                PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
                clear_GPIOS();
                packet_send();

        }

        PROCESS_END();
}
```

A.5 receiver-observer.c

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

```
/**
 * \file
 *
 *       Test Software used for the Dual-Mote system
 *       Application for the receiver of the observer network.
 *       This mote will be connected to a PC via USB
 *       All the information for computing PDR, latency and power usage is
 *       printed out
 *
 * \author
 *           Maite Bezunartea
 *           June, 2014
 *
 */

// imported libraries.

#include <stdio.h>
#include "contiki.h"
#include "dev/i2cmaster.h"
#include "dev/tmp102.h"
#include "net/rime.h"
#include "dev/leds.h"


/* In Rime communicating nodes must agree on a 16 bit virtual
 * channel number. For each virtual channel one can define

 * the Rime modules for communicating over that channel.
 * Channel numbers < 128 are reserved by the system.
 */
uint16_t channel =        133;

// sender power
// possible values =  0dBm = 31;  -1dBm = 27;  -3dBm = 23;  -5dBm = 19;
//                   -7dBm = 15; -10dBm = 11; -15dBm =  7; -25dBM =  3;
uint8_t power = 7;                //This value can be changed by the user

// message counters
static uint16_t received = 0 ;

/* Altough it doesn't have anything to do every 10 seconds this timer is needed in order
 * for the receiver to work ???
 */
#define TMP102_READ_INTERVAL (CLOCK_SECOND*10)
static struct etimer et;


// Writes a title on the console

PROCESS(temp_process, "receiving numbered messages");
AUTOSTART_PROCESSES(&temp_process);

struct testmsg {
        uint16_t seqno_rpl;
        uint8_t seqno_unicast;
        uint32_t energy;
        uint32_t timestamp_app;
        uint16_t counter_ADC;
        uint16_t timestamp_mac;

};

// This is the receiver function

recv_uc(struct unicast_conn *c, const rimeaddr_t *from)
{
        unsigned long rtime; // received time (long)
          struct testmsg msg;
          memcpy(&msg, packetbuf_dataptr(), sizeof(msg));

          received ++;
        rtime=RTIMER_NOW();
        uint16_t timestamp = packetbuf_attr(PACKETBUF_ATTR_TIMESTAMP);

        /*output format:rtimer-receiver|sender|received|seqno|ENERGY|msg timestamp|cpu|lpm|
         transmit|listen|timestamp-sender|rtimer-sender|RSSI|LQI*/
        printf("%lu,%u,%u,%u,%u,%u,%lu,%lu,%lu, %lu, %lu\n",
                rtime,from->u8[0], received,
                msg.seqno_rpl,msg.seqno_unicast,msg.energy,msg.counter_ADC,
                msg.timestamp_app,msg.timestamp_mac,timestamp);

}

static const struct unicast_callbacks unicast_callbacks = {recv_uc};

static struct unicast_conn uc;

PROCESS_THREAD(temp_process, ev, data)
```


upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

```
{
        PROCESS_BEGIN();

        // Initialisation of the useless but needed timer ???

        tmp102_init();

        unicast_open(&uc, channel, &unicast_callbacks);

        // adjust power
        cc2420_set_txpower(power);
        PROCESS_END();
}
```

A.6 sender-observer.c

```
/**
 * \file
 *
 *       Test Software used for the Dual-Mote system
 *       Application for the receiver of the observed network.
 *       It sets the GPIOs as outputs
 *       It collects information and sends it to the receiver of the
 *               observer network
 *
 * \author
 *          Maite Bezunartea
 *          June, 2014
 *
 */

// imported libraries.

#include <stdio.h>
#include <string.h>
#include <ctype.h>

#include "contiki.h"
#include "dev/i2cmaster.h"
#include "dev/tmp102.h"
#include "net/rime.h"
#include "dev/leds.h"
#include <msp430.h>
#include <legacymsp430.h>

#include "dev/battery-sensor.h"



/* In Rime communicating nodes must agree on a 16 bit virtual
 * channel number. For each virtual channel one can define
 * the Rime modules for communicating over that channel.
 * Channel numbers < 128 are reserved by the system.
 */
uint16_t channel =        133;
// sender mote id
uint8_t sender;           // set automatically

// sender power
// possible values =  0dBm = 31;  -1dBm = 27;  -3dBm = 23;  -5dBm = 19;
//                   -7dBm = 15; -10dBm = 11; -15dBm =  7; -25dBM =  3;
uint8_t power = 31;

// receiver mote id
uint8_t receiver = 100; //ID for the receiver mote of the observer network

// message counters
uint16_t send = 0 ;

/* We read the ADC 100 times per second
 * the timer READ_INTERVAL is defined for
 * such purpose
 */

#define READ_INTERVAL (CLOCK_SECOND/100)

/* Data structure of messages sent from sender
 *
 */
struct testmsg {
        uint16_t seqno_rpl;
        uint8_t seqno_unicast;
        uint32_t energy;
        uint32_t timestamp_app;
        uint16_t counter_ADC;
```


upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

```
        uint16_t timestamp_mac;

};


uint16_t seqno=0;
int32_t ADCResult=0;
int      flag;
int32_t ADCcounter=0;

float
floor(float x)
{
  if(x >= 0.0f) {
    return (float) ((int) x);
  } else {
    return (float) ((int) x - 1);
  }
}

PROCESS(temp_process, "Sending numbered messages");
AUTOSTART_PROCESSES(&temp_process);

/*---------------------------------------------------------------------------
 * SETTING THE GPIOS
 *---------------------------------------------------------------------------*/

void
GPIOS_init(void)
{
        //configuring pin as GPIO

        P1SEL &= ~0xC1;                    //GPIO P1.0 P1.6 P1.7
        P2SEL &= ~0x18;                    //GPIO P2.3 P2.4
        P4SEL &= ~0x0D;                    //GPIO P4.0 P4.2 P4.3

        //configuring pin as INPUT

        P1DIR &= ~BIT0;         //P1.0 AS INPUT
        P1DIR &= ~BIT6;         //P1.6 AS INPUT
        P1DIR &= ~BIT7;         //P1.7 AS INPUT

        P2DIR &= ~BIT3;         //P2.3 AS INPUT
        P2DIR &= ~BIT4;         //P2.4 AS INPUT

        P4DIR &= ~BIT0;         //P4.0 AS INPUT
        P4DIR &= ~BIT2;         //P4.2 AS INPUT
        P4DIR &= ~BIT3;         //P4.3 AS INPUT

}

int
read_GPIOS(void)
{
        //reading the value in each pin
        int number=0;
        int bit0=(P1IN & BIT6);                 //read value of P1.6
        int bit1=(P1IN & BIT7);                 //read value of P1.7
        int bit2=(P2IN & BIT3);                 //read value of P2.3
        //int bit3=(P2IN & BIT4);               //read value of P2.4
        int bit4=(P4IN & BIT0);                 //read value of P4.0
        int bit5=(P4IN & BIT2);                 //read value of P4.2
        int bit6=(P4IN & BIT3);                 //read value of P4.3*/

        if (P1IN & BIT6)

                number=number+1;        //  write a 1 in P1.6
        if (P1IN & BIT7)
                number=number+2;        //  write a 1 in P1.7

        if (P2IN & BIT3)
                number=number+4;        //  write a 1 in P2.3

        if (P4IN & BIT0)
                number=number+8;        //  write a 1 in 4.0

        if (P4IN & BIT2)
                number=number+16;        //  write a 1 in P4.2

        if (P4IN & BIT3)
                number=number+32;        //  write a 1 in P4.3

        return number;
}

/*---------------------------------------------------------------------------
 * Receiver function which seems necessary to operate the sender with a normal MAC
 *---------------------------------------------------------------------------*/
```


upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

```c
static void
recv_uc(struct unicast_conn *c, const rimeaddr_t *from)
{
        printf("unicast message received from %d.%d\n",
                        from->u8[0], from->u8[1]);
}

static const struct unicast_callbacks unicast_callbacks = {recv_uc};
static struct unicast_conn uc;

/*---------------------------------------------------------------------------*/
void
packet_send(void)
{

        rimeaddr_t destination;
        seqno++;
        // define the destination address
        destination.u8[0] = receiver;
        destination.u8[1] = 0;


        /*Set sender's info*/
        energest_flush();

    struct testmsg msg;
        /*Set general info*/

        msg.seqno_unicast=seqno;
        msg.seqno_rpl=read_GPIOS();       //this value has to be read from the gpios
        msg.energy=ADCResult;             //this value has to be read from the ADC12
        msg.timestamp_app= RTIMER_NOW();
        msg.counter_ADC=ADCcounter;

        /*Avoid sending to itself*/
        if(!rimeaddr_cmp(&destination, &rimeaddr_node_addr))
        {
                packetbuf_copyfrom(&msg, sizeof(msg));

            packetbuf_set_attr(PACKETBUF_ATTR_PACKET_TYPE,
                                        PACKETBUF_ATTR_PACKET_TYPE_TIMESTAMP);

                unicast_send(&uc, &destination);
        printf(" Seqno %u\n",msg.seqno_rpl);

        }


}
/*---------------------------------------------------------------------------
 * Send packets with all needed statistic information
 *---------------------------------------------------------------------------*/

/*---------------------------------------------------------------------------*/

static struct etimer et;

// Sending thread

PROCESS_THREAD(temp_process, ev, data)
{
        PROCESS_BEGIN();
        SENSORS_ACTIVATE(battery_sensor);

        flag=(P1IN & BIT0);

        // adjust power
        cc2420_set_txpower(power);

        //tmp102_init();
        GPIOS_init();
        //ADC_init();
        unicast_open(&uc, channel, &unicast_callbacks);
        // infinite loop
        while(1)
        {
                //    wait for the READ_INTERVAL time
                etimer_set(&et, READ_INTERVAL);
                PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
                int32_t bateria = battery_sensor.value(0);

                ADCResult = ADCResult +bateria;
                ADCcounter++;

                if (flag!=(P1IN & BIT0))        //read state of P1.0IN. if it has changed, we should send a packet
                {

                flag=(P1IN & BIT0);                     //save the new state of P1.0
                packet_send();                          //packet sent to the receiver of the observer network
```


upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

```
                printf("Battery: %li \n", ADCResult);
                ADCResult=0;                                    //reset for the next packet
                ADCcounter=0;

                }

        }

        SENSORS_DEACTIVATE(battery_sensor);

        PROCESS_END();
}
```

A.7 recv-observer.c

```
/**
 * \file
 *
 *       Test Software used for the Dual-Mote system
 *       Application for the receiver of the observed network.
 *       It sets the GPIOs as outputs
 *       It collects information and sends it to the receiver of the
 *               observer network
 *
 * \author
 *          Maite Bezunartea
 *          June, 2014
 *
 */

// imported libraries.

#include <stdio.h>
#include <string.h>
#include <ctype.h>

#include "contiki.h"
#include "dev/i2cmaster.h"
#include "dev/tmp102.h"
#include "net/rime.h"
#include "dev/leds.h"
#include <msp430.h>
#include <legacymsp430.h>

#include "dev/battery-sensor.h"



/* In Rime communicating nodes must agree on a 16 bit virtual
 * channel number. For each virtual channel one can define
 * the Rime modules for communicating over that channel.
 * Channel numbers < 128 are reserved by the system.
 */
uint16_t channel =        133;
// sender mote id
uint8_t sender;           // set automatically

// sender power
// possible values =  0dBm = 31;  -1dBm = 27;  -3dBm = 23;  -5dBm = 19;
//                   -7dBm = 15; -10dBm = 11; -15dBm =  7; -25dBM =  3;
uint8_t power = 31;

// receiver mote id
uint8_t receiver = 100; //ID for the receiver mote of the observer network

// message counters
uint16_t send = 0 ;

/* We read P1.0 1000 times per second
 * the timer READ_INTERVAL is defined for
 * such purpose
 */

#define READ_INTERVAL (CLOCK_SECOND/1000)

/* Data structure of messages sent from sender
 *
 */
struct testmsg {
        uint16_t seqno_rpl;
        uint8_t seqno_unicast;
        uint32_t energy;
        uint32_t timestamp_app;
        uint16_t counter_ADC;
    uint16_t timestamp_mac;
```


upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

```
};

uint16_t seqno=0;
int32_t ADCResult=0;
int      flag;
int  counter=0;

PROCESS(temp_process, "Sending numbered messages");
AUTOSTART_PROCESSES(&temp_process);

/*---------------------------------------------------------------------------
 * SETTING THE GPIOS
 *---------------------------------------------------------------------------*/

void
GPIOS_init(void)
{
        //configuring pin as GPIO

        P1SEL &= ~0xC1;                   //GPIO P1.0 P1.6 P1.7
        P2SEL &= ~0x18;                   //GPIO P2.3 P2.4
        P4SEL &= ~0x0D;                   //GPIO P4.0 P4.2 P4.3

        //configuring pin as INPUT

        P1DIR &= ~BIT0;           //P1.0 AS INPUT
        P1DIR &= ~BIT6;           //P1.6 AS INPUT
        P1DIR &= ~BIT7;           //P1.7 AS INPUT

        P2DIR &= ~BIT3;           //P2.3 AS INPUT
        P2DIR &= ~BIT4;           //P2.4 AS INPUT

        P4DIR &= ~BIT0;           //P4.0 AS INPUT
        P4DIR &= ~BIT2;           //P4.2 AS INPUT
        P4DIR &= ~BIT3;           //P4.3 AS INPUT

}

int
read_GPIOS(void)
{
        //reading the value in each pin
        int number=0;

        int bit0=(P1IN & BIT6);                 //read value of P1.6
        int bit1=(P1IN & BIT7);                 //read value of P1.7
        int bit2=(P2IN & BIT3);                 //read value of P2.3
        //int bit3=(P2IN & BIT4);                       //read value of P2.4
        int bit4=(P4IN & BIT0);                 //read value of P4.0
        int bit5=(P4IN & BIT2);                 //read value of P4.2
        int bit6=(P4IN & BIT3);                 //read value of P4.3*/

        if (P1IN & BIT6)
                number=number+1;        //  write a 1 in P1.6

        if (P1IN & BIT7)
                number=number+2;        //  write a 1 in P1.7

        if (P2IN & BIT3)
                number=number+4;        //  write a 1 in P2.3

        if (P4IN & BIT0)
                number=number+8;        //  write a 1 in 4.0

        if (P4IN & BIT2)
                number=number+16;        //  write a 1 in P4.2

        if (P4IN & BIT3)
                number=number+32;        //  write a 1 in P4.3

    printf(" seqno %u %u %u %u %u %u \n",bit0,bit1,bit2,bit4,bit5,bit6);
    //printf(" number %u \n",number);

        return number;
}

/*---------------------------------------------------------------------------
 * Receiver function which seems necessary to operate the sender with a normal MAC
 *---------------------------------------------------------------------------*/

static void
recv_uc(struct unicast_conn *c, const rimeaddr_t *from)
{
        printf("unicast message received from %d.%d\n",
                        from->u8[0], from->u8[1]);
```


upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

```
}

static const struct unicast_callbacks unicast_callbacks = {recv_uc};
static struct unicast_conn uc;

/*---------------------------------------------------------------------------*/
void
packet_send(void)
{

        rimeaddr_t destination;
        seqno++;
        // define the destination address
        destination.u8[0] = receiver;
        destination.u8[1] = 0;


        /*Set sender's info*/
        energest_flush();

    struct testmsg msg;
        /*Set general info*/

        msg.seqno_unicast=seqno;
        msg.seqno_rpl=read_GPIOS();                //this value has to be read from the gpios
        msg.energy=ADCResult;                      //this value has to be read from the ADC12
        msg.timestamp_app= RTIMER_NOW();
        msg.counter_ADC=counter;


        /*Avoid sending to itself*/
        if(!rimeaddr_cmp(&destination, &rimeaddr_node_addr))
        {
                packetbuf_copyfrom(&msg, sizeof(msg));
        //printf(" Packet sent is %s\n",payload);
        //printf(" size of payload %u\n",sizeof(msg));
            packetbuf_set_attr(PACKETBUF_ATTR_PACKET_TYPE,
                               PACKETBUF_ATTR_PACKET_TYPE_TIMESTAMP);

                unicast_send(&uc, &destination);
        printf(" Seqno %u\n",msg.seqno_rpl);

        }


}
/*---------------------------------------------------------------------------
 * Send packets with all needed statistic information
 *---------------------------------------------------------------------------*/

/*---------------------------------------------------------------------------*/

static struct etimer et;

// Sending thread

PROCESS_THREAD(temp_process, ev, data)
{
        PROCESS_BEGIN();
        SENSORS_ACTIVATE(battery_sensor);

        flag=(P1IN & BIT0);      //we check the state of p1.0

        // adjust power
        cc2420_set_txpower(power);
        //tmp102_init();
        GPIOS_init();
        //ADC_init();
        unicast_open(&uc, channel, &unicast_callbacks);
        // infinite loop
        while(1)
        {
                //      wait for the READ_INTERVAL time
                etimer_set(&et, READ_INTERVAL);
                PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));

                counter++;
                //every ten readings of P1.0 we record ADC12 (100 times per second)
                if (counter%10==0){
                        int32_t bateria = battery_sensor.value(0);
                        ADCResult = ADCResult +bateria;
                        counter=0;
                }

                if (flag!=(P1IN & BIT0))        //read state of P1IN. if it has changed, we should send a packet
                {

                        flag=(P1IN & BIT0);                     //save the new state of P1.0
                        packet_send();
```


upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

```
                    printf("Battery: %li \n", ADCResult);
                    ADCResult=0;                        //reset for the next packet

            }

        }

        SENSORS_DEACTIVATE(battery_sensor);

        PROCESS_END();
}
```


upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Appendix B

PhP Code

B.1 analyze.php

```
<?php

/* Script for analysing the data provided by the Datalogger.

1. All recorded sent moments are written down in the first column.
2. The received moments are added in the second column, in the latest possible
position allowed by causality. This implies that some sent messages are considered
as lost.
3. If a received message can not be added in the second column due to causality,
3.1 If one of the five previous sent messages was considered lost, this is undone:
all messages following the lost one are shifted one position back in time, creating
place for the message that could not be added (the figure 5 results from manual
simulations of the algorithm, it might be improved)
3.2 If none of the five previous messages was considerd as lost, the newly received
message is considered as a duplicate and is ignored.

The output file will be recorder in the /tmp/ directory
ods.php is used. Obtained from http://sourceforge.net/projects/ods-php/

Author: Maite Bezunartea
 */

include("ods.php"); //include the class and wrappers
include("line_count.php");
include("latency.php");
include("find_lost.php");


$file='cm8pr15.txt';            //write here the name of the .txt file to read
$number_lines= line_count($file)-14;
echo $number_lines;


if (($handle = fopen($file, "r")) !== FALSE) {
    $lines = file($file);
    $object = newOds(); //create a new ods file
    $row = 0;       //counter for the line in .txt
    $line_sent=0;       //counter for the line in .ods
    $line_rcv=0;        //counter for the line in .ods
    $prev_ch1=0;
    $prev_ch2=array(0, 0);


    echo "<pre>LEYENDO_RCV</pre>";

    while ((($data = fgetcsv($handle, NULL, " ")) !== FALSE) && ($row <= $number_lines)) {
        $data=implode(" ", $data); //convert array data into string
        $array = preg_split("/[[:space:]]+/",$data);

        if ($row >= 12){
            if ((($array[3]>=20) && ($prev_ch2[1]<20)) || (($array[3]>=20) &&
                ($prev_ch2[0]>=20) && ($prev_ch2[1]>=20))){
                $object->addCell(0,$line_rcv,1,$array[0],'string');
                //add a cell to sheet 0, row line_Sent, cell 0, with value 1 and type string
                    $line_rcv++;
            }
        array_shift($prev_ch2);
        array_push($prev_ch2,$array[3]);
```


upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

```
        }
        $row++;

    }

}

    fclose($handle);

$n_lines=$line_rcv;


if (($handle = fopen($file, "r")) !== FALSE) {


    echo "<pre>LEYENDO_SENT</pre>";
    $row=0;


    while ((($data = fgetcsv($handle, NULL, " ")) !== FALSE) && ($row <= $number_lines)) {
        $data=implode(" ", $data); //convert array data into string
        $array = preg_split("/[[:space:]]+/",$data);

        if ($row >= 12){
            if (($array[1]>=20) && ($prev_ch1<20)){     // only save in case we have tx
                $object->addCell(0,$line_sent,0,$array[0],'string');
                //add a cell to sheet 0, row line_Sent, cell 0, with value 1 and type string
                $line_sent++;
            }
        }
        $row++;
        $prev_ch1=$array[1];                            //save latest ch1 read

    }

 }


fclose($handle);

$lost=array_fill(0,5,0);
$row=0;
$rcv=$object->readCell(0,0,1)[value];
$next_rcv=$object->readCell(0,1,1)[value];
$sent=$object->readCell(0,$row,0)[value];
$next_sent=$object->readCell(0,$row+1,0)[value];


//echo $rcv;
//echo $next_rcv;
//echo $sent;
$object->addCell(0,$row,3,$rcv,'string');

$row++;

$rcv=$object->readCell(0,1,1)[value];
$next_rcv=$object->readCell(0,2,1)[value];
$sent=$object->readCell(0,$row,0)[value];
$next_sent=$object->readCell(0,$row+1,0)[value];



for ($i = 1; ($i < $n_lines); $i++) {
     while (latency($next_sent, $rcv)>=0) {               //consider packet as lost
                $object->addCell(0,$row,3,'0','string');

                array_unshift($lost, 1);
                array_pop($lost);

                $rcv=$object->readCell(0,$i,1)[value];
                $next_rcv=$object->readCell(0,($i+1),1)[value];

                //$row++;
                $sent=$object->readCell(0,$row,0)[value];
                $next_sent=$object->readCell(0,$row+1,0)[value];

                $i++;
     }


    if (latency($sent, $rcv)>=0) {
                $object->addCell(0,$row,3,$rcv,'string');

                array_unshift($lost, 0);
                array_pop($lost);

                $rcv=$object->readCell(0,$i+1,1)[value];
                $next_rcv=$object->readCell(0,($i+2),1)[value];
```


upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

```
//                    $row++;
//                    $sent=$object->readCell(0,$row,0)[value];
//                    $next_sent=$object->readCell(0,$row+1,0)[value];

         }

         if (latency($sent, $rcv)<0) {                    //duplicated packet-> ignore
                 //printf(latency($sent, $rcv));
             if(in_array(1,$lost)==TRUE){

                 $position_lost=find_lost($lost);
                 $pos=$row-($position_lost+1);

                 for ($c = 1; ($c < $position_lost+1); $c++) {
                     $next=$object->readCell(0,$pos+1,3)[value];
                     $object->addCell(0,$pos,3,$next,'string');
                     $pos++;
                 }

                 $lost[$position_lost]=0;
                 $object->addCell(0,$row,3,$rcv,'string');

                 array_unshift($lost, 0);
                 array_pop($lost);

                 $rcv=$object->readCell(0,$i+1,1)[value];
                 $next_rcv=$object->readCell(0,($i+2),1)[value];

                 //$row++;
                 $sent=$object->readCell(0,$row,0)[value];
                 $next_sent=$object->readCell(0,$row+1,0)[value];

             }
//              if(in_array(1,$lost)==FALSE){
//                  $rcv=$object->readCell(0,$i+1,1)[value];
//                  $next_rcv=$object->readCell(0,($i+2),1)[value];
//                  //printf('ignored',$sent);
//              }
         }
                     $row++;

//echo $i;
}

saveOds($object,'/tmp/cm8pr15.ods'); //save the object to a ods file
echo "FIN";
?>
```

B.2 line_ count.php

```
<?php
function line_count($file)
{

//function for counting the lines in a file

    $linecount = 0;
    $handle = fopen($file, "r");
    while(!feof($handle)){
      $line = fgets($handle);
      $linecount++;
    }

    fclose($handle);

    return $linecount;
}
?>
```

B.3 latency.php

```
<?php
function latency($time_sent, $time_rcv)
//function for calculating the difference between
//the time a packet was received and the time where it was
//received

{
```


upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

```
    $latency=$time_rcv - $time_sent;
    return $latency;
}
?>
```

B.4 find_ lost.php

```
        <?php

/*
function for finding the position of one lost packet between the last five packets received
 */
        function find_lost($array)
{
        $indice=array_search(1,$array);
        echo $indice;
        return $indice;
}


            ?>
```


upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Appendix C

MatLab Code

C.1 script.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Script for comsputing average and standard deviation for
% latency, hopcount for a given .csv file, obtained from the serialdumps
%
% Author: Maite Bezunartea
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fname='rplnull4.csv';                                  %name of the .csv file we want to analyze
lines=countLines(fname);

limitsender=strcat('G1..G',num2str(lines));   %until where we are going to read for senders
senders=csvread(fname,0,6,limitsender);          %array only with senders
array_senders=unique(senders);
array=array_senders;
array1=[0;1];

 [M,lines]=CreateCSV(fname);              %create file with the required data
 fname='mrplnull4.csv';
 seqno(fname);

limitread=strcat('C1..D',num2str(lines));   %until where we are going to read for senders
data=csvread(fname,0,2,limitread);          %array only with senders


for i=1:length(array_senders)
    count=find(senders==array_senders(i));

    for j=1:length(count)
    hops(j)=data(count(j),1);
    latency(j)=data(count(j),2);
    end

    hops_mean(i)=mean(hops);
    hops_std(i)=std(hops);

    latency_mean(i)=mean(latency);
    latency_std(i)=std(latency);

    clearvars hops latency;

end

%in the array called results we have all the desired data for each mote of the network
% format of array: sender id | mean hopcount | std hopcount | mean latency | std latency
results=horzcat(array_senders, hops_mean', hops_std',latency_mean', latency_std');

figure
errorbar(results(:,2),results(:,3)./2,'r')
title('Number of hops');
xlabel('Node ID');
ylabel('Hop');

figure
errorbar(results(:,4),results(:,5)./2,'g')
title('Latency');
xlabel('Node ID');
ylabel('Latency (ms)');
```


upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

```
set(gca,'XtickLabel',results(:,1));


%from here we calculate global statistics (not for each node)

count_110=find(senders==array_senders(1));
count_111=find(senders==array_senders(2));
count_112=find(senders==array_senders(3));
count_113=find(senders==array_senders(4));
count_114=find(senders==array_senders(5));
count_115=find(senders==array_senders(6));
count_116=find(senders==array_senders(7));
count_117=find(senders==array_senders(8));
count_119=find(senders==array_senders(9));
count_121=find(senders==array_senders(10));
count_122=find(senders==array_senders(11));
count_123=find(senders==array_senders(12));
count_124=find(senders==array_senders(13));
count_126=find(senders==array_senders(14));
count_127=find(senders==array_senders(15));
count_128=find(senders==array_senders(16));
count_129=find(senders==array_senders(17));
count_130=find(senders==array_senders(18));
count_135=find(senders==array_senders(19));
count_136=find(senders==array_senders(20));

count_1_hop=vertcat(count_110,count_111,count_117,count_119,count_122,count_126,count_127,count_130);
    for j=1:length(count_1_hop)
    hops(j)=data(count_1_hop(j),1);
    latency(j)=data(count_1_hop(j),2);
    end

    final_hops_mean(1)=mean(hops);
    final_hops_std(1)=std(hops);

    final_latency_mean(1)=mean(latency);
    final_latency_std(1)=std(latency);
    clearvars hops latency;


count_12_hop=count_128;
    for j=1:length(count_12_hop)
    hops(j)=data(count_12_hop(j),1);
    latency(j)=data(count_12_hop(j),2);
    end

    final_hops_mean(2)=mean(hops);
    final_hops_std(2)=std(hops);

    final_latency_mean(2)=mean(latency);
    final_latency_std(2)=std(latency);
    clearvars hops latency;

    count_13_hop=count_114;
    for j=1:length(count_13_hop)
    hops(j)=data(count_13_hop(j),1);
    latency(j)=data(count_13_hop(j),2);
    end

    final_hops_mean(3)=mean(hops);
    final_hops_std(3)=std(hops);

    final_latency_mean(3)=mean(latency);
    final_latency_std(3)=std(latency);
    clearvars hops latency;

    count_14_hop=vertcat(count_124,count_129);
    for j=1:length(count_14_hop)
    hops(j)=data(count_14_hop(j),1);
    latency(j)=data(count_14_hop(j),2);
    end

    final_hops_mean(4)=mean(hops);
    final_hops_std(4)=std(hops);

    final_latency_mean(4)=mean(latency);
    final_latency_std(4)=std(latency);
    clearvars hops latency;


    count_18_hop=count_116;
    for j=1:length(count_18_hop)
    hops(j)=data(count_18_hop(j),1);
    latency(j)=data(count_18_hop(j),2);
    end

    final_hops_mean(5)=mean(hops);
    final_hops_std(5)=std(hops);
```


upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

```
    final_latency_mean(5)=mean(latency);
    final_latency_std(5)=std(latency);
    clearvars hops latency;

    count_19_hop=count_123;
    for j=1:length(count_19_hop)
    hops(j)=data(count_19_hop(j),1);
    latency(j)=data(count_19_hop(j),2);
    end

    final_hops_mean(6)=mean(hops);
    final_hops_std(6)=std(hops);

    final_latency_mean(6)=mean(latency);
    final_latency_std(6)=std(latency);
    clearvars hops latency;

    count_2_hop=vertcat(count_112,count_113,count_115,count_121);
    for j=1:length(count_2_hop)
    hops(j)=data(count_2_hop(j),1);
    latency(j)=data(count_2_hop(j),2);
    end

    final_hops_mean(7)=mean(hops);
    final_hops_std(7)=std(hops);

    final_latency_mean(7)=mean(latency);
    final_latency_std(7)=std(latency);
    clearvars hops latency;

    count_27_hop=vertcat(count_135,count_136);
    for j=1:length(count_27_hop)
    hops(j)=data(count_27_hop(j),1);
    latency(j)=data(count_27_hop(j),2);
    end

    final_hops_mean(8)=mean(hops);
    final_hops_std(8)=std(hops);

    final_latency_mean(8)=mean(latency);
    final_latency_std(8)=std(latency);
    clearvars hops latency;

    results_final=horzcat(final_hops_mean', final_hops_std',final_latency_mean', final_latency_std');
```

C.2 countLines.m

```
function count = countLines(fname)              %function used for counting lines in a file
fh = fopen(fname, 'rt');
assert(fh ~= -1, 'Could not read: %s', fname);
x = onCleanup(@() fclose(fh));
count = 0;
while ~feof(fh)
    count = count + sum( fread( fh, 16384, 'char' ) == char(10) );
end
end
```


upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira