Experimental Comparison of Radio Duty Cycling Protocols for Wireless Sensor Networks

Dissertation submitted in fulfillment of the requirements for the award of the degree of Doctor in Engineering Sciences and Technology by

Marie-Paule Uwase

Promotors : Prof. dr. ir. Jean-Michel Dricot (ULB)
Prof. dr. ir. Kris Steenhaut (VUB)
Prof. dr. ir. Jacques Tiberghien (VUB)

October 2018

Jury Members

President	Prof. dr. ir. Van Biesen Leo ELEC department , Vrije Universiteit Brussel lvbiesen@vub.be
Vice president	Prof. dr. ir. Pintelon, Rik ELEC department, Vrije Universiteit Brussel Rik.Pintelon@vub.be
Secretary	Prof. dr. ir. Goossens Marnix ETRO department, Vrije Universiteit Brussel mgoossens@etrovub.be
External Members	Prof. dr. ir Reali Gianluca Networks and Services lab, University of Perugia gianluca.reali@unipg.it Prof. Aimé Lay-Ekuakille Department of Innovation Engineering, University of Salento aime.lay.ekuakille@unisalento.it
Internal members	Prof. Antoine Nonclercq Beams, Université Libre de Bruxelles anoncler@ulb.ac.be
Promoters	Prof. dr. ir. Steenhaut, Kris ETRO department, Vrije Universiteit Brussel ksteenha@etrovub.be Prof. dr. ir. Dricot Jean-Michel Opera Department, Université Libre de Bruxelles jdricot@ulb.ac.be Prof. dr. ir. Tiberghien Jacques ETRO department, Vrije Universiteit Brussel jgtiberg@etrovub.be

To Dana and Benoît

To my late mother

Abstract

Wireless sensor networks are often battery powered and therefore their power consumption is of critical importance. Power requirements can be reduced by switching off radios when they are not needed and by using multi-hop communications to reduce the length of the radio links. Multi-hop communications however require message routing through the network. The Routing Protocol for lossy networks (RPL) has been designed by the Internet Engineering Task Force (IETF) for seamless integration of wireless sensor networks in the Internet. For switching on and off radios, radio duty cycling (RDC) protocols have been added to the traditional medium access control (MAC) protocols. Despite the fact they belong to different layers in the communications stack, it is intuitively clear that the choice of a specific RDC protocol for saving energy can influence the performances of RPL.

Exploring experimentally this influence was the initial goal of this research. A 25 nodes wireless sensor network using Zolertia Z1 motes and the Contiki software was used for this investigation. Performance measurements without RDC protocol and with the three different RDC protocols readily available in Contiki were organized and the results of the experiments were compared. Unfortunately, with all three RDC protocols, serious malfunctions obscured the experimental results. Those malfunctions did not show up in absence of an RDC protocol and they could not be reproduced by our simulation studies. To tackle this issue, the behavior of the RDC protocols was scrutinized by means of experimental set-ups that eliminated as much as possible all non RDC related issues.

Many, quite varied, malfunctions were discovered which all could have caused the observed RPL issues. Further research and better experimental set-ups made clear that all the discovered RDC malfunctions could be attributed to two real-world facts that were not considered by the implementers of the Contiki RDC protocols. The first cause is the small frequency difference between hardware real time clocks in stand-alone motes. The second is that the threshold built in the receiver to detect radio activity is much higher than the minimum level of signal that the same receiver can decode. Work-arounds have been designed for the observed malfunctions and they have been tested by means of a systematic comparison of the performance of the three modified RDC protocols.

Acknowledgments

Foremost, I would like to express my sincere gratitude to my promoters Prof. Kris Steenhaut, Prof. Jean-Michel Dricot and Prof. Jacques Tiberghien for the continuous support of my Ph.D study and research, for their patience, motivation, enthusiasm, and immense knowledge. Your guidance helped me in all the time of research and writing of this thesis. I could not have imagined having better promoters and mentors for my Ph.D study.

Besides my promoters, I would like to thank the members of the jury for their valuable time, effort, patience and collaboration.

I thank my fellow PhD candidates in ETRO and OPERA departments for the stimulating discussions. In particular, I am grateful to Maite Bazunartea for her help on the experimental work of my research.

I am immensely grateful to the "Fonds Jacques Lewin – Inès Henriques de Castro" from ULB for their financial support during the first 5 years of my PhD research.

Some special words of gratitude go to my friends who have always been a major source of support when things would get a bit discouraging: Benita, Chantal, Jeannette, Aimée, Clara and Assumpta. Thank you, ladies, for always being there for me.

A very special word of thanks goes for my mother, for giving birth to me at the first place, for her moral and emotional support and her love throughout my life.

Finally, I want to thank my husband and love of my life, Benoît, for keeping things going at home, for his love, encouragement, and sacrifices. I could not have gone through this journey alone without your help and support.

The last word goes for Dana, my baby girl, who has been the light of my life for the last 18 months and who has given me the extra strength and motivation to get things done. This thesis is dedicated to her.

Table of Contents

General Introduction

Wireless Sensor networks have drawn the attention of researchers, educators as well as industry and have become a hot topic under the name Internet of Things (IoT).
The work presented in this thesis originated from the desire to investigate, on one hand, which role wireless sensors networks could play as a low-cost support for teaching computing and telecommunications engineering, and, on the other hand, which courses and lab sessions would be required by traditional engineering students to demystify the functioning of WSNs, their power saving issues, their communications stack and its cross-layer issues.

A preliminary study allowed me to get familiar with the WSN's typical protocol stack and to pinpoint design problems and implementation issues with currently available open source solutions. While doing so, an interesting research opportunity presented itself: wireless sensor networks observed serious performance degradations when using Radio Duty Cycling (RDC) protocols for extending the lifetime of the batteries, together with the "Routing Protocol for Lossy networks" (RPL) [1], whereas, according to simulations, this combination should have worked fine. This issue, also hampering the energy-efficient operation of sensor nodes commercially available from one of our industrial contacts triggered our work on the performance study of asynchronous RDC protocols.

As simulated radio networks are notorious for their inaccurate reflection of reality[2], it was decided to set-up up a real RPL network with some 25 motes in an area in which uncontrolled radio interference was not going to jeopardize accurate measurements. Important performance indices were chosen: Packet Delivery Rate (PDR), Packet end to end latency and used power. The goal was to measure them for characterizing the behavior of RPL with different RDC protocols implemented for the widely used operating system Contiki and to deduce from these measurements which RDC protocol should be used underneath the RPL protocol. Unfortunately, it appeared that the planned measurements did not give consistent nor reproducible results.
The routing tables (called "DODAG") computed by the RPL algorithms were unstable and a fraction of the motes failed to join the network. The ad-hoc testbed appeared to be inadequate to explore in detail the causes of these problems but malfunctions of the different available

RDC protocols were suspected. The research goals were then reoriented towards the identification and correction of these RDC malfunctions.

A first experimental set-up, specifically oriented towards testing of RDC protocols, was designed and used intensively to try to diagnose RDC malfunctions while comparing performance indices of the different RDC protocols in a well-defined test environment. The large amount of data collected allowed to compare different measurement techniques, to establish guidelines for better testbeds and last but not least, to uncover several unidentified malfunctions of the tested RDC protocols. Unfortunately, the collected data were not accurate and detailed enough to diagnose accurately the malfunctions. The findings made during this part of the research were presented at the IEEE International Black Sea Conference on Communications and Networking,(BlackSeaCom 2014)[3].

Based upon the conclusions of the previous phase of the research, an entirely new testbed was designed and built with the help of Maite Bezunartea, a master's student in electrical engineering and later a PhD candidate in our research group, who contributed to the performance studies, which she is now extending towards newer motes and protocols.

The new testbed permitted quasi automated analysis of the huge volumes of data collected from RDC performance tests and also allowed performing similar tests on an entire RPL network with up to 23 nodes. The previously suspected malfunctions occurring occasionally could be confirmed and analyzed accurately. A few were just small software bugs that were easily corrected, but most resulted from an over-simplified view of the physical radio layer. The addition to that view of the clock drift between motes and the distinction between sensitivity of a receiver and the threshold for detecting radio activity were two significant contributions to the design rules for RDC protocols. Corrections and work-arounds were added to existing protocols and tested. These were presented for the first time at the Embedded Wireless Systems and Networks conference (EWSN) in 2016 in Graz[4] and later at a workshop at the Sensys'17 conference in Delft [5]. At both places these results helped many participants to understand difficulties they had encountered themselves but never fully diagnosed nor corrected.

Finally, the availability of corrected RDC protocols led to another important contribution to the field of WSN design as, with the testbed designed for diagnosing the malfunctions, the performances of the different available protocols could be determined experimentally in a set-

up that eliminated as much as possible all external factors jeopardizing objective comparisons. The most important results of these comparisons were published in the IEEE Sensors journal[6].

The research goal of identifying which asynchronous RDC protocol is best associated with RPL was not pursued any further because new, more promising, approaches to power savings in RPL networks were being proposed in the literature and studied by other team members.

The Communication stack in WSNs

To situate our research in the vast domain of WSNs and of the IoT, Figure 1 shows the communications stack mainly used for our research. The parts of the stack on which the research was focused is surrounded in red.

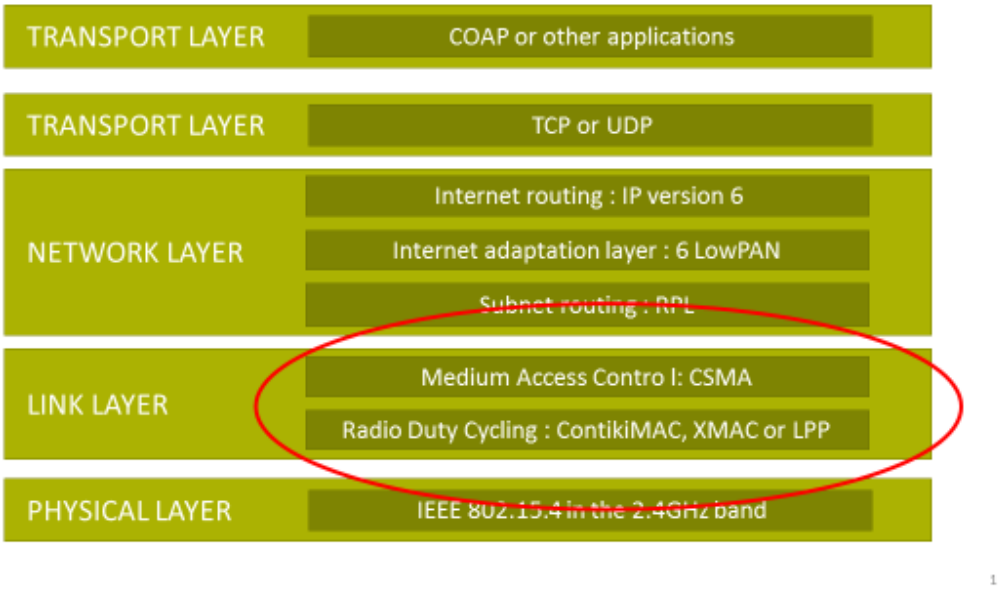

Figure 1: The Communication stack in WSNs

Publications

2017

Uwase, M-P, Bezunartea Pascual, M, Tiberghien, J, Dricot, J-M & Steenhaut, K 2017, 'Experimental Comparison of Radio Duty Cycling Protocols for Wireless Sensor Networks' *IEEE Sensors Journal*, vol 17, no. 19, pp. 6474-6482. DOI: 10.1109/JSEN.2017.2738700

2016

Uwase, M-P, Bezunartea Pascual, M, Tiberghien, J, Dricot, J-M & Steenhaut, K 2016, Poster: ContikiMAC, some critical issues with the CC2420 Radio. in K Römer , K Langendoen & T Voigt (eds), *EWSN '16 Proceedings of the 2016 International Conference on Embedded Wireless Systems and Networks.* ACM, pp. 257-258 , EWSN 2016, Graz, Austria, 15/02/16.

2015

Bezunartea Pascual, M, Uwase, M-P, Tiberghien, J, Dricot, JM & Steenhaut, K 2015, Demonstrating the versatility of a low-cost measurement testbed for Wireless Sensor Networks with a case study on Radio Duty Cycling protocols. in *Lecture Notes of ICST (LNICST).*, 144318119371487, Springer, EAI International Conference on CYber physiCaL systems, IoT and sensors Networks, Rome, Italy, 26/10/15.

2014

Uwase, M-P, Bezunartea Pascual, M, Nguyen Thanh, L, Tiberghien, J, Steenhaut, K & Dricot, J-M 2014, Experimental evaluation of message latency and power usage in WSNs. in *2014 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom).* EDAS Conference Services, 313 Westview Ave Leonia, NJ 07605 USA, pp. 69-72, 2014 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom), Chisinau, Moldova, Republic of, 27/05/14.

2013

Nguyen Thanh, L, Uwase, M-P, Tiberghien, J & Steenhaut, K 2013, QoS-aware Cross-layer Mechanism for Multiple Instances RPL. in *Proceedings of the 2013 International Conference on Advanced Technologies for Communications (ATC'13).* International Conference on Advanced Technologies for Communications (ATC), pp. 61-66, 2013 International Conference on Advanced Technologies for Communications (ATC 2013), Ho Chi Minh City, Viet Nam, 16/10/13.

Uwase, M-P, Nguyen Thanh, L, Tiberghien, J, Steenhaut, K & Dricot, J-M 2013, Poster abstract: Outdoors Range Measurements with Zolertia Z1 Motes and Contiki. in W Hu, K Langendoen, F Ferrari, M Zimmerling & L Mottola (eds), *Proceedings of the 5th International Workshop, REALWSN 2013.* vol. 281, Springer.

2012

Uwase, M-P, Nguyen Thanh, L, Tiberghien, J, Steenhaut, K & Dricot, J-M 2012, Design Hints for didactic Simulators. in *Proceedings of the 15th International Conference on Interactive Collaborative Learning.* IEEE, 15th International Conference on Interactive Collaborative Learning (ICL), Villach, Austria, 26/09/12.

M. Paule Uwase, L. Nguyen, J. Tiberghien, K. Steenhaut and J. Dricot, "Reversing Time in a Didactic Simulator for Wireless Sensor Networks", 17 Feb 2012 *Institute for Systems Engineering and Automation.* Grosspietsch, E. & Klöckner, K. (eds.). (Proceedings of the Work in Progress Session of the 20th Euromicro International Conference on parallel, Distributed and Network-based Processing PDP 2012)

2011

Paradisi, A., Uwase, M-P., Tiberghien, J., Steenhaut, K. & Dricot, J-M, Didactic simulators for understanding routing protocols in wireless sensor networks, International Association of Technology, Education and Development, p. 5807-5814 8 p. (Proceedings of the 4th International Conference of Education, Research and Innovation, iCeRi, Madrid , 14th-16th of October)

Structure of the thesis

The thesis text is structured as follows:
The first chapter summarizes briefly a number of examples of applications of wireless sensors and actuators so that the reader unfamiliar with these devices can understand their broad application field and perceive the issues that make their deployments challenging.

The second chapter introduces the Contiki operating system for WSNs. It explains the overall organization of the system and, in more detail, the parts for which the performance and correctness was investigated, as presented in the following chapters.

The third chapter describes the first steps taken to experimentally explore the performance of multi-hop wireless sensor networks. Prior to using them in full-fledged networks, the characteristics of the used motes were compared to estimate the variability between supposedly identical devices. Their radio ranges with built-in and external whip antennas were measured under different topological and meteorological conditions. Power consumption, Packet Delivery Rate and packet latency for unicast links using different RDC protocols were compared.

As the experiments described in chapter 3 had brought to light several RDC protocol malfunctions which could not be fully diagnosed it was decided that a better testbed was needed. After studying the existing testbeds accessible via the Internet, it was finally concluded that none could address the malfunctions that had to be diagnosed. Therefore, a new testbed was designed and built.

Chapter 4 discusses the existing testbeds, compares different techniques to measure packet latency and describes the dual network used for this research.

Chapter 5 contains an extensive list of RDC design issues and malfunctions revealed by the multiple experiments that were done, their diagnosis and possible corrections or work-arounds. As mentioned earlier in this introduction most of them were due to only two different causes but manifested themselves in very different ways in the different RDC protocols.

Chapter 6 describes the measurement and reporting techniques used in a concluding measurement campaign using the improved RDC protocols, which all worked properly, to compare their performances.

Chapter 7 presents the general conclusions of this research and gives some suggestions about the future work. It summarizes what can be learned from this long journey through Contiki, a widely used open source software designed to be eminently portable. These findings have often been discussed with the people supporting professionally Contiki and their recent launching of Contiki NG makes us believe that they share many of our points of view.

Chapter 1: Wireless Sensor and Actuator Networks

1.1. What is a wireless sensor network

Wireless Sensor Networks (WSNs) are made with autonomous intelligent sensors, usually powered by battery, which can measure certain characteristics of their environment, such as temperature, pressure, noise level, acceleration, etc. [7]. These sensors (often called "motes") communicate by radio with their neighbors to forward their measurements to a central data collection point from where all the data is sent to some processing and analysis center via any classical network, such as the Internet (Figure 2). Some of these networks not only observe their environment, but act upon it. They are called Wireless Sensor and Actuator Networks (WSANs). A trivial example being a network of movement and light sensors and smart light bulbs, that bring light when and where needed.

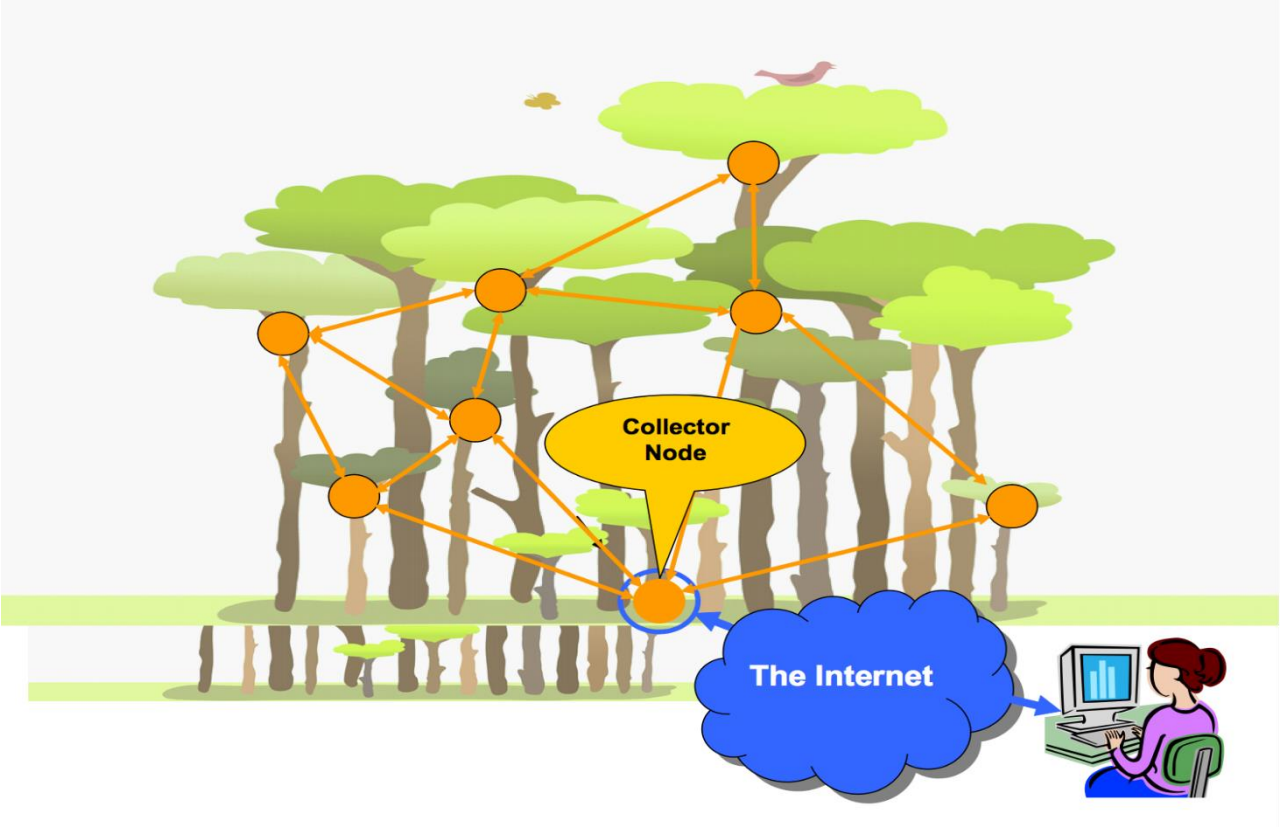

Figure 2: A typical wireless sensor network

Many sensors monitor some physical phenomena, such as temperature, humidity or bird's chirps in nature. These data are transmitted by radio to a collector node, which is itself connected to the Internet, making these data accessible worldwide.

1.2. Initial Wireless Sensor Network applications

Sensor Networks provide potentially endless opportunities in a number of different applications, a few of which will shortly be described here to illustrate how broad this application field is. The selection of applications is inspired by the conclusions of a workshop organized in 2004 to evaluate the possible applications of this, then new, research and development field[7]. For some applications, the description given by the authors will be quoted literally rather than rephrasing them.

1.2.1. Wireless sensor networks and the environment

- **ZebraNet:**

  The first non-military large scale wireless sensor network that was widely described and had great influence for the later development of the field was designed at Princeton University by Philo Juang et al.[8]. It allowed to study the movements of Zebras across a large wild area in Kenya. The experience gained during deployment of the sensors in the wild was described by Pei Zhang et al. in 2004 at the second SENSYS conference[9]. This paper was given in 2017, at the 18th SENSYS conference, the award for the most significant paper presented at the 8 first SENSYS conferences.

- **Water catchments and eco-system monitoring:** According to K. Branko et al.[10], "A network of sensors can be utilized to monitor water flows into catchment areas and areas where access is difficult or expensive. This information can be combined with other sensor networks providing information on water quality and soil condition, together with long term weather forecasting to assist with the equitable and efficient distribution of water for irrigation and environmental purposes. Similar technology can be utilized to provide an early warning system for flood prone regions, particularly flash flooding."

- **Glacsweb monitoring system:** According to K. Martinez et al. [11], [12], "Monitoring ice caps and glaciers provides valuable information about global warming and climate change. The GlacsWeb system consists of probes inserted in the glacier, a base station on the glacier, and a reference station that relays data to the SNS in Southhampton, England. Most of the probes are at the ice-sediment boundary from 50 to 80 m deep. Each probe is equipped with pressure, temperature, and orientation (tilt in three dimensions) sensors. The probes are not recoverable. The base station serves as a communications relay between the probes and the reference station and as the controller for autonomous operation. It includes temperature

and tilt sensors, a snow meter, a webcam, and a differential GPS to follow ice movements. The reference station is a mains powered, Linux based gateway for transferring data. It acts as the position reference point for the differential GPS."

- **Elephant detection:** to prevent dangerous interactions between elephants and humans it is important to detect elephants wandering at night close to villages. Low cost infrasound microphones detecting the very low frequency sounds generated by elephants and connected via radio with a central warning station is being tested successfully in Sri Lanka[13].

1.2.2. Wireless sensor networks and health

- **Real-time health monitoring**: In their paper, H. Alemdar and C. Ersoy [14] wrote: "A network of advanced bio-sensors can be developed using nanotechnology to conduct point-of-care testing and diagnosis for a broad variety of conditions. This technology will reduce delays in obtaining test results, thus having a direct bearing on patient recovery rates or even survival rates. On the basis of the sensed data, physicians can make a more rapid and accurate diagnosis and recommend the appropriate treatment. WSNs may also enable testing and early treatment in remote locations, as well as assist triage on location at accident or disaster sites".
- **Nike+ shoes**: Quoted from the Nike web site[15]**:** "Nike+ is designed for athletes who like to run with music and who want to measure and monitor their progress vs. their goals. Nike+ is an innovative product and information system that enhances any running experience. A sensor placed under the sock liner of the left Nike+ ready shoe measures the pace, distance, time elapsed and calories burned. This information is transmitted wirelessly to a receiver on an Apple iPod nano® or an I-phone for real-time audio feedback while one listens to his/her favorite workout music".

1.2.3. Wireless senor networks and safety

- **Bush fire response:** In their paper, A. Bayo et al. [16] asserted that "A low-cost distributed sensor network for environmental monitoring and disaster response could assist in responding to bushfires by using an integrated network of sensors combining on-the-ground sensors – monitoring local moisture levels, humidity, wind speed and direction – with

satellite imagery to determine fire-risk levels in targeted regions and offering valuable information on the probable direction in which fires may spread. This type of WSN can provide valuable understanding of bushfire development and assist authorities in organizing a coordinated disaster response by providing early warning for high risk areas".

- **Remote Sensing in Disaster Management:** A. Mangla et al. stated in their paper [17] that "Remote sensing systems have proven to be invaluable sources of information that enable the disaster management community to make critical decisions based on information obtained from study of satellite imagery for better preparedness and initial assessments of the nature and magnitude of damage and destruction. Information derived from satellites can be combined with on-the-ground data from a WSN. High resolution remote sensing data is especially useful for documenting certain hazards, for determining where to locate response facilities and supplies, and for planning related facilities for reconstruction and relocation activities. Data availability and its timely delivery are crucial to saving lives and property during disasters, and technological developments are making positive contributions in this area. Some of the most significant progress in disaster reduction is being made in mitigation, using historical and contemporary remote sensing data in combination with other geospatial data sets as input to compute predictive models and early warning systems".

1.2.4. Wireless sensor networks and security

- **Radiation detection with distributed sensor networks:** In any assessment of potential terrorist attacks, the nuclear threat takes center stage. A plausible scenario involves terrorists detonating a simple radiological dispersion device broadcasting highly radioactive particles over a densely populated area. Prevention requires detection of vehicles carrying unauthorized radio-active material. Around major cities in the US, radiation detection portals have been installed on some highways, but these portals slow traffic, are highly visible and easily circumvented. The Distributed Sensor Network project at Los Alamos National Laboratory, in cooperation with the University of New Mexico has developed a wireless sensor network-based system that can easily and discretely be deployed in streets. To compensate for the lack of sensitivity of small sized radiation detectors, a large number of detectors cumulatively measures the radiation emitted by a specific vehicle, whose position is monitored over a certain distance by means of vibration and/or magnetic field sensors [18].

- **Shooter localization in urban terrain:** In their paper, M. Maroti et al. [19] wrote: **"**Detecting and accurately locating snipers has been an elusive goal of the armed forces and law enforcement agencies for a long time. Most existing counter sniper systems use acoustic signals related to gunfire to determine a shooter's location. The performance of such systems significantly degrades when used in urban environment due to multipath effects. The PinPtr system uses a wireless sensor network of many low-cost sensors to determine both a shooter's location and the bullet's trajectory by measuring both the muzzle blast and the shock wave." Similar to traditional systems, PinPtr estimates the source location based on the measured time of arrival of acoustic events, known sensor locations and the speed of sound. To eliminate multipath effects, the PinPtr sensor-fusion algorithm, running on a base station, computes a function of time and space giving the number of sensor measurements that are consistent with hypothetical shooter positions and shot times. Finding the global maximum of the function yields the sniper position".

1.2.5. Wireless sensor networks and transportation

- **Intelligent Transportation Systems (ITS):** According to K. Selvarajah et al. [20], "A network of sensors set up throughout a vehicle can interact with its surroundings to provide valuable feedback on local roads, weather and traffic conditions to the car driver, enabling adaptive drive systems to respond accordingly. For instance, this may involve automatic activation of braking systems or speed control via fuel management systems. Condition and event detection sensors can activate systems to maintain driver and passenger comfort and safety through the use of airbags and seatbelt pre-tensioning. Sensors for fatigue and mood monitoring based on driving conditions, driver behavior and facial indicators can interact to ensure safe driving by activating warning systems or directly controlling the vehicle. A broad city-wide distributed sensor network could be accessed to indicate traffic flows, available parking places, administer tolls or provide continually updated destination routing feedback to individual vehicles. The feedback may be based on global and local information, combining GPS information with cellular networks".

1.2.6. Wireless sensor networks for monitoring the integrity of structures

- The Aquila Tower, that contains beautiful historical frescos, is a 31 m tall medieval tower located in the city of Trento in Italy. It has been severely damaged by earthquakes and its preservation requires permanent monitoring of the structure of the building.

Some thirty, battery powered wireless sensors, equipped with accelerometers, strain gauges and a thermometer have been sealed in the walls of the tower [21] for early detection of any displacements in the structure

1.2.7. Wireless sensor networks and education

- **Wireless sensor networks to introduce distributed and embedded processing:** In their paper, B. Hamingway et al. [22] stated that "The University of Washington's Computer Science & Engineering "Flock of Birds" project integrates the theory and practice of wireless sensor networks into the mainstream undergraduate curriculum early enough to form a basis for all student's understanding of embedded computing. Using motes equipped with TinyOS and acoustic IO devices, the project has sound generation as a major focus. Students can understand and implement it in a month, yet it taxes a system's cycle and memory capacity enough to make efficiency an important design constraint. Each student programs a mote to act as a bird that has several songs stored in its local memory. The programs execute a common rule base, but each bird acts independently – deciding which song to sing based on what the other birds within radio range are singing. In combination, the songs create the sound of a flock of birds".

- **Sensor motes to teach digital signal processing:** According to H. Kwon et al. [23], **"**The introduction of hardware wireless sensors in a signal processing education setting can serve as a paradigm for data acquisition, collaborative signal processing, or simply as a platform for obtaining, processing and analyzing real-life real-time data. A two-way interface between the java-digital signal processing (J-DSP) visual programming environment and a wireless sensor network equipped with TinyOS allows J-DSP to issue commands to multiple motes, activate specific transducers, and analyze data using any of the existing J-DSP signal processing functions in real-time**"**. The combination of a user-friendly software environment for digital signal processing and numerous easily deployable intelligent data acquisition and processing motes has been favorably experienced both by graduate and undergraduate students and has motivated many of the latter to further explore digital data processing.

1.3. Power issues

The motes used in most of the applications given as examples in the previous paragraphs cannot be powered from the mains, either because it is not available in the vicinity of the motes or because wires would be prohibitively inconvenient or costly. Even if in some specialized applications power can be harvested from the environment [24], a majority of wireless sensor networks rely upon batteries for their power. Due to the practical difficulties associated with battery replacement, one often considers the exhaustion of batteries as the practical end-of-life of the application. Therefore, a lot of research is targeted at reducing the power consumption in wireless networks [25].

1.4. The Internet of Things

The first WSNs and WSANs were mostly standalone networks designed and build for a specific purpose that used ad-hoc protocols optimized for specific motes and environments. The evolution of the Internet towards THE universal communications medium made integration of WS(A)Ns in the Internet a hot issue. For relatively powerful devices such as webcams and Arduino or Raspberry-pi card computers this is not a problem because they have enough processing power, memory and I/O capacity to support a full-fledged Ethernet or Wi-Fi interface, but for smaller, battery powered devices without external sources to recharge batteries, this is almost impossible. This problem has been tackled by the Internet Engineering Task Force working group "IPv6 Low power wireless Personal Area Networks", known by the acronym "6LowPAN" [26], [27].

The future oriented choice of IPv6 was logical, because only the 128 bits addresses of IPv6 could handle the billions of devices one considers as likely parts of the so called "Internet of things". The simplifications of the IP layer were obtained by considering each WSN as a subnet of the Internet. This way, the WSN is identified by a 64-bit network identifier and the addresses inside the WSN become 64-bit host numbers.

As inside the WSN many of the features of IPv6, such as flow identifiers, are useless, the packet header can also be considerably simplified, but two features have to be added to the protocol: routing inside the subnet (traditionally an IP subnet is a single broadcast domain, which is not the case with a multi-hop WSN)) and packet fragmentation because the size of IPv6 packets exceeds the maximum size of WSN packets.

A routing protocol to be used inside a 6LowPAN network has also been defined by the internet engineering task force (IETF) and is called “Routing Protocol for low Power and lossy networks” (RPL) [28].

The possibility of including almost seamlessly wireless sensor networks in the Internet was the motivation for the development of a new generation of application software: Instead of creating for each application a specific user-interface, simplified web interfaces, such as CoAP [29] were developed to provide a request/response interaction model between application endpoints that supports built-in discovery of services and resources, and includes key concepts of the Web such as URIs and Internet media types. CoAP is designed to easily interface with HTTP for integration with the Web while meeting specialized machine to machine communication requirements [30].

Chapter 2: Operating Systems for Wireless Sensor Networks

2.1. 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 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 [31], initially developed at the University of California at Berkeley by David Culler et al. as early as 1999. Contiki [32], 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, ANSI C in Contiki).

Some review papers [33] provide interesting comparisons between TinyOS and Contiki.

More recently, a third player has gained some popularity. It is the RIOT operating system developed jointly by the French INRIA and the Freie Universität Berlin [34], [35]. RIOT is a multi-threading operating system, written in C and C++ and made as similar as possible to Linux while emphasizing energy savings and minimal resource requirements.

As the research group in which this research was done uses almost exclusively Contiki, the remainder of this chapter will essentially be devoted to that operating system. The reader should however understand that our preference for Contiki is purely circumstantial and that we did not compare the performance of the three popular systems mentioned here.

A totally different approach to support advanced programming on motes consists in implementing a Java Virtual Machine and in using the resources of the Java language to create distributed applications for WSNs. This approach would be particularly attractive when wireless sensor motes were to be used for educational purposes because it would minimize the threshold students have to cross before being able to build meaningful applications running on motes. A Java environment for wireless motes was available as a commercial product from a Californian company called Sentilla [36] and was used successfully by our team. Unfortunately, Sentilla discontinued that product, as well as the support for its users. A similar approach to mote programming in Java became available as open source code under the name Takatuka [37]. It was developed in the University of Freiburg in Germany by a PhD student, Faisal Aslam. After a visit to the University of Freiburg, we had to conclude that using Takatuka for new projects would require too much additional work to be done.

In this chapter, first the central issue of all operating systems, event handling, will be briefly discussed. Thereafter, an extensive overview of Contiki will be made and finally the power saving features of Contiki will be explained in more detail, as they have played a central role in this research.

2.2. Event management

Most software on WSNs can be considered as event driven. Events are created when changes of the state of I/O devices or timers take place. Whereas 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 at random locations while the specific interrupt handler is being executed.

The primary task of an operating system consists in managing the interrupts and their associated handlers in such a way that the application programmer, instead of being overwhelmed by the non-deterministic behavior of the software, can rely upon clearly defined and preferably easy to understand abstractions of the event driven reality.

2.2.1. Concurrent processes

The most common of these abstractions is the sequential process [38]: 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 (Figure 3)

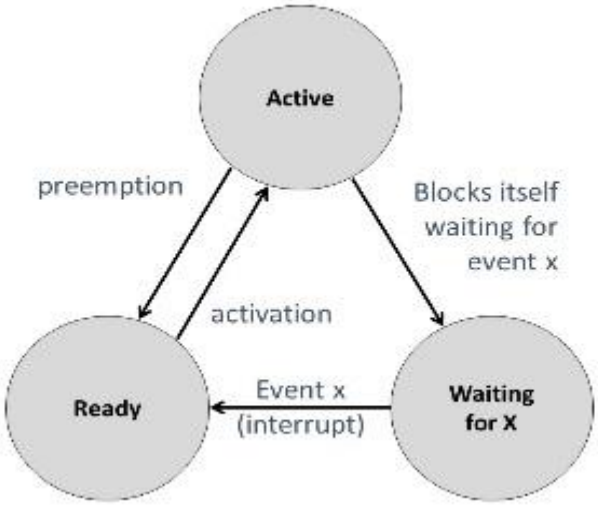

Figure 3: Process states

In the Active state, the process is executing its sequential code, which can possibly 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 it does not use CPU resources and other processes can become active and use them or the processor can enter a low power state often called "sleeping". 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 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.

2.2.2. 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 is switched 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 local variables. Threads have become a standard feature of most operating systems and are part of the most popular programming languages.

2.2.3. Finite state machines

When resources are very restricted as in typical WSN 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 its state accordingly. This approach requires very little memory and is fast, as there is only one process and one stack, which does not need to be saved when an event occurs and as the state of a finite state machine holds in just one byte if the number of states does not exceed 255. However, from the application programmer's 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, waiting has to be implemented by stepping through the successive states of a finite state machine which appears to be an error prone piece of program (or style of programming).

Protothreads

Contiki offers a new programming abstraction, called protothread [39] 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 quite similar to those of threads. Protothreads provide a conditional blocking wait abstraction but they require much less memory and have less overhead than true threads as they do not provide local dynamic variables and therefore do not require saving a stack when being switched between states.

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. They can also be implemented for any ANSI C compiler but only when no switch statements are used in the same context as the protothread

2.3. Contiki

2.3.1. Overall organization

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 updated version numbers in the Contiki source code, with, as a consequence, that differences might exist between versions carrying the same version number, 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 the 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 subdirectories associated with 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. Some of the platform subdirectories have a **dev** subdirectory containing specific software for devices that belong to that platform and that require other drivers than the generic ones contained in the **core/dev** directory.
Finally, it can also contain an **apps** subdirectory 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 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.
    - **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 early internet applications such as telnet, ftp, dhcp, web browsers 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 [40] (a database) and erbium (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 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 an RPL network (collect-view), insert and extract data in a Contiki database (coffee) and load operating systems in older personal computers.

- 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.

2.3.2. Communications facilities

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. Figure 4 shows a global layered model of the Contiki communications stack.

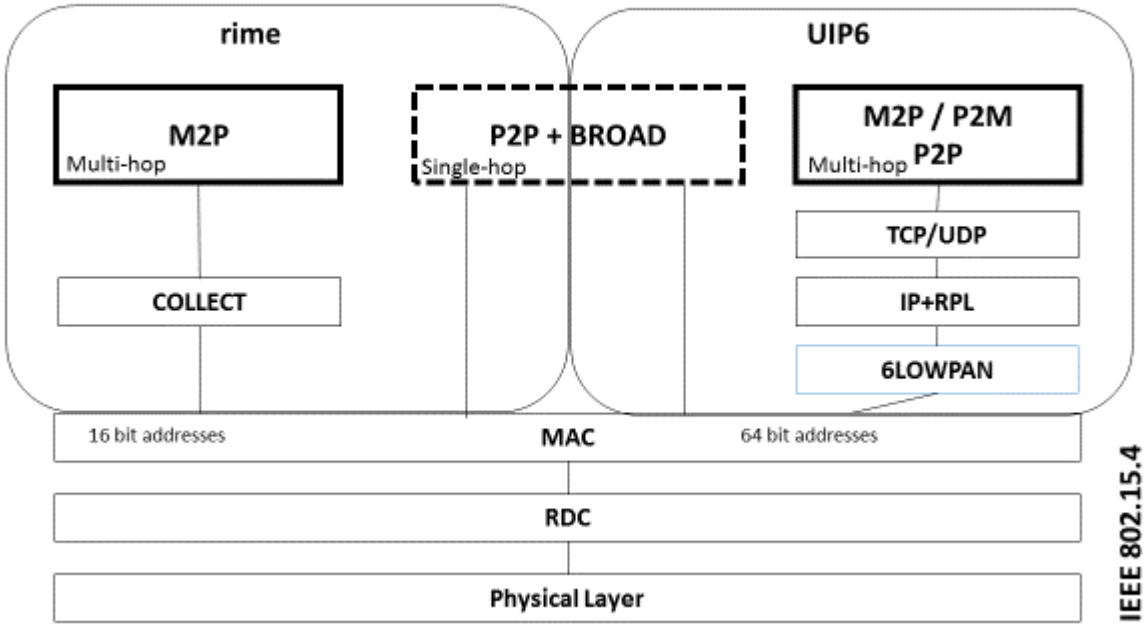

Figure 4: Global view of the Contiki communication stacks:

The three lower layers are common to both upper stacks. The older rime stack offers multi-point to point (M2P) multihop services, whereas the modern UIP6 stack offers not only multihop M2P services but also point to multipoint (P2M) and point to point (P2P) services. Both stacks offer single hop P2P and broadcast services.

Two full-fledged stacks can be configured by means of make-files. Both offer single-hop point to point and broadcast communications. The "rime" stack offers in addition multi-hop routing from any point to a central sink. The UIP6 stack, which can be connected to the IPv6 internet, offers full routing facilities between all nodes. The Medium Access Control (MAC), Radio

Duty Cycling (RDC) and Physical layers are common to both stacks and obey the IEEE 802.15.4 specifications. Rime and UIP6 are mostly outside of the scope of this work as only the single-hop unicast, and broadcast communications have been studied in detail. The MAC and RDC layer will be described in detail further in this chapter. At the physical 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 that is used nowhere else and the corresponding software can be found via the appropriate **platform** directory.

2.4. Energy saving features

Energy efficiency is a fundamental theme pervading the design of communication protocols developed for wireless sensor networks, including routing and MAC layer protocols. 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. One of the primary mechanisms for achieving low energy consumption in energy-constrained WSNs is duty cycling. New MAC/RDC protocols and surveys of the different existing protocols are a significant part of the scientific literature about wireless sensor networks. The very large number of proposed protocols is a clear indication that no protocol obviously better than the others exists, different application requirements favor different MAC/RDC protocols. One can distinguish two main classes of protocols:

- The deterministic protocols are based upon the Time Domain Multiple Access (TDMA) algorithm for the MAC layer. They offer a reliable, service with moderate and predictable latency [41]. Assigning some time slots for communication and others for keeping the radio asleep allows to reduce the power consumption to the strict minimum, but synchronizing clocks in all motes can cost a non-negligible amount of energy [42]–[44]. One such synchronous MAC protocol, combining TDMA and frequency hopping has been recently (2016) added to the IEEE 801.15.4e standard. It is called “Time Slotted Channel Hopping” (TSCH). It has been implemented on Contiki [45]. The performance study of this multi-channel MAC protocols is out of the scope of this study.

- The more common asynchronous protocols clearly distinguish the MAC layer and the RDC layer. The MAC layer essentially implements a Carrier Sense Multiple Access (CSMA) protocol. In the RDC layer, the 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.

Many RDC protocols have been proposed and some of them are widely available in the operating systems for wireless sensor networks. One can distinguish two different approaches: in the first, it is the sender that announces repeatedly its desire to transmit. When the addressed receiver wakes up, it invites the sender to transmit its message. In the second approach, it is the receiver who announces that it is awake. Upon reception of such an announcement, the sender who has some message to transmit can proceed. Among the many RDC protocols with probing senders XMac [46] and ContikiMac [47] are quite popular in the literature. The best known representative of the other category is LPP [48].

A quite extensive overview of the many other RDC protocols is given in [49]. Both approaches can be optimized by keeping, in the sender, a table that gives for all neighbors the exact time when they were last noticed as awake. This table allows then a sender to stay off until the receiver is likely to wake up, This mechanism, originally proposed by A. El-Hoiydi and J.-D.Decotignie [50] is called “encounter optimization” or “phase locking”. Unfortunately, as analyzed in chapter 5, inaccuracies in the determination of the wake-up moment and drift of the clocks of different motes can jeopardize the correct operation of this mechanism [51].

In the following sections, the essentials of the MAC layer and of the different RDC layers available in Contiki will be described.

2.4.1. The MAC layer in Contiki

The MAC layer receives incoming packets from the RDC layer and uses that layer to transmit packets. When configuring the communications stack of Contiki, one has the choice between two MAC protocols: NullMAC and CSMA.

NullMAC is just an empty interface between the upper layers and the RDC layer. It provides the same set of functions as CSMA, but the functions do not perform any task except the blind transmission of packets.

CSMA is similar to the MAC protocols commonly used in wired and wireless local area networks. The Collision Detection (/CD) specified in IEEE 802.3 cannot be used due to the incomplete connectivity (hidden station) that characterizes most radio networks and the Collision Avoidance (/CA) mechanism introduced in IEEE 802.11 for wireless networks, such as WiFi, is not used because of its overhead and implementation complexity. A partial flowchart of CSMA, as specified in IEEE 802.15.4, is shown in Figure 5 to explain how packets are being sent. It first distinguishes between packets to be broadcasted and packets with a specified destination. When the packet needs to be broadcasted, it is directly passed to the RDC layer, which is in charge of sending. When it has a specified destination, the packet is put in a queue for that destination, unless that queue is full. In that case, the packet is just dropped. The destination specific queues are handled by a protothread that passes, one by one, the packets to the RDC layer for sending. If an acknowledgment is received in due time afterwards, the packet is removed from the queue. Otherwise the count of failed attempts associated with the packets is increased and, if that count does not exceed a predefined maximum a new attempt to send is made after a random delay. The default value for the number of attempts in Contiki is 3.

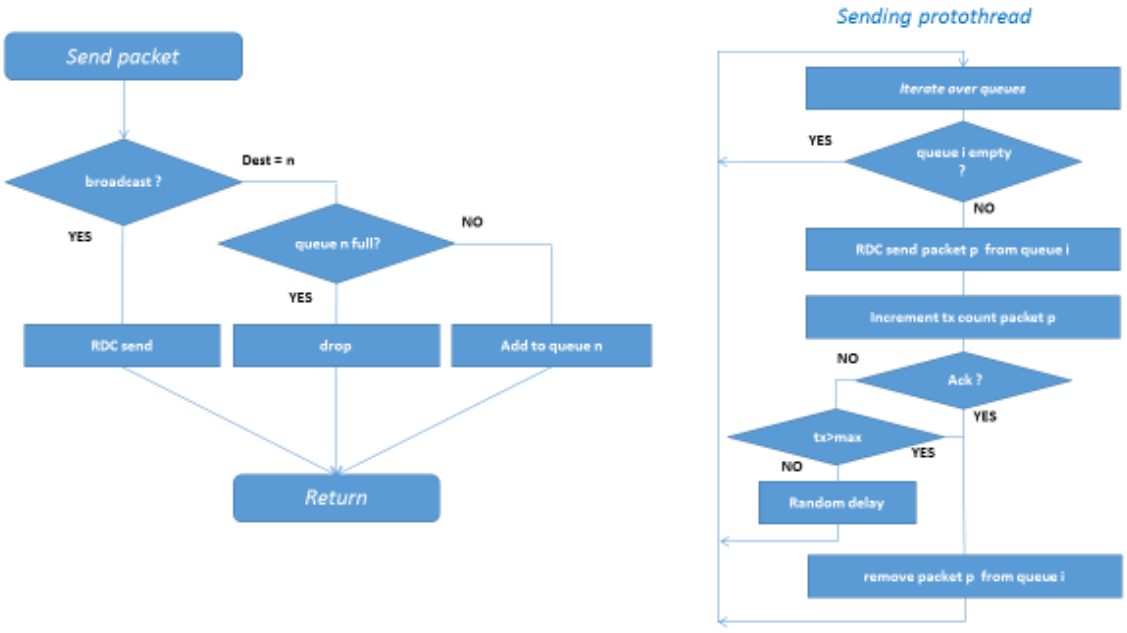

Figure 5: Simplified flow chart of the part of CSMA that sends packets

2.4.2. ContikiMac

In his paper introducing ContikiMac [47], Adam Dunkels describes the ContikiMac protocol as follows:

"*ContikiMac is a radio duty cycling protocol that uses periodical wake-ups to listen for packet transmissions from neighbors. If a packet transmission is detected during a wake-up, the receiver is kept on to be able to receive the packet. When the packet is successfully received, the receiver sends a link layer acknowledgment. To transmit a packet, a sender repeatedly sends its packet until it receives a link layer acknowledgment from the receiver. Packets that are sent as broadcasts do not result in link layer acknowledgments. Instead, the sender repeatedly sends the packet during the full wake-up interval to ensure that all neighbors have received it*".

The next section gives details on the ContikiMac functionalities.

2.4.2.1 Sending a frame with a specified destination address (Figure 6)

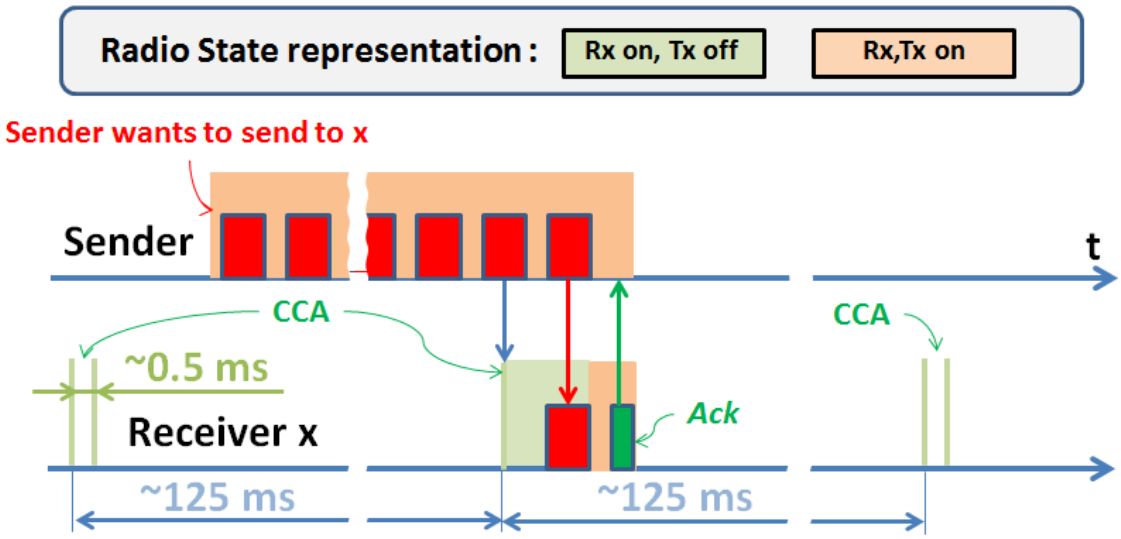

Figure 6: ContikiMac: unicast transmission

The red blocks represent the data packet to be transmitted.

Most of the time the receiver sleeps, but a few times per second (default value = 8) the receiver wakes up to perform two consecutive Clear Channel Assessments (CCAs). The CCA is a mechanism that uses the Received Signal Strength Indicator (RSSI) of the radio transceiver to give an indication of radio activity on the channel. If the RSSI is below a predefined threshold,

the CCA returns positive, indicating that the channel is clear. If the RSSI is above the threshold, the CCA returns negative, indicating that the channel is in use. If, at wake-up, the CCAs observe no radio-activity the receiver returns to sleep.

The transmitter, when it has to transmit a frame, transmits that frame repeatedly, for a duration longer than the interval between two pairs of CCAs. When, in a woken-up receiver, 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, waits for possible follow-up packets and/or returns to sleep.

2.4.2.2. Broadcasting a frame (Figure 7)

As broadcast frames cannot be acknowledged (the ACKs would collide with each other). The transmitter retransmits the frame for the whole duration of a receiver wake-up cycle to ensure that all receivers have had the opportunity of receiving the packet.

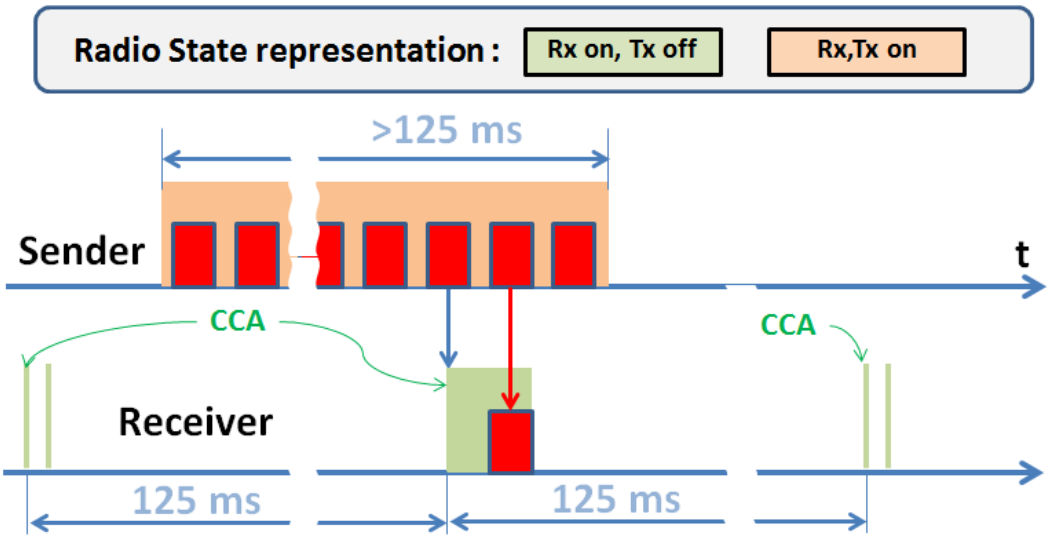

Figure 7: ContikiMac: Broadcast transmission

The red blocks represent the data packet to be broadcasted

2.4.2.3. Timing requirements (Figure 8)

ContikiMac requires precise timings.

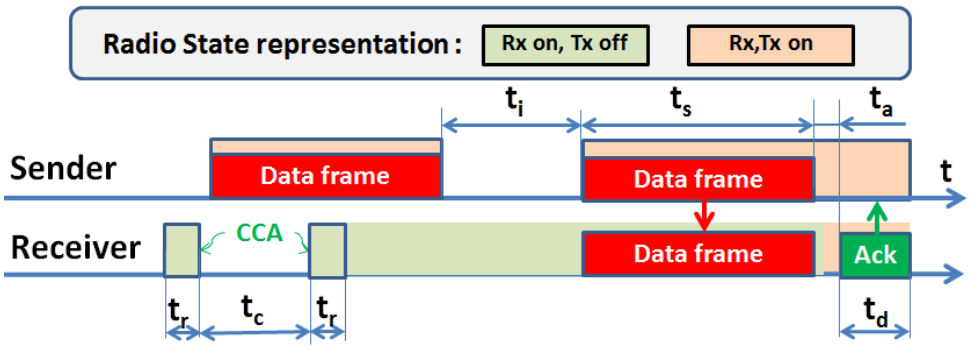

Figure 8: ContikiMac: timing requirements

t_i = interval between data frame retransmissions.
t_r = time required for a stable Clear Channel Assessment.
t_c = time interval between the 2 CCAs performed at a wake-up.
t_c = time between receiving a frame and sending the Ack.
t_d = time required for recognizing an Ack.
t_s = transmission time of shortest frame

The timing parameters shown in Figure 8 must satisfy a certain number of constraints:

- Between retransmissions, the sender waits a predefined time t_i. If a single CCA would be performed during that interval, no radio-activity would be detected, while the transmitter is attempting to get a frame through. Therefore, ContikiMac uses two consecutive CCAs, separated by a time interval t_c larger than t_i.
- The time t_c between the two consecutive CCAs should be shorter than the duration of the shortest possible frame, otherwise such a frame could go undetected.
- To avoid collisions between a retransmitted frame and an Ack for the just transmitted frame, the time necessary to generate the Ack on the receiver side and to recognize it at the sender side should be smaller than the time tj between successive retransmissions.

These constraints are summarized as follows:

ti < tc (no frame should go undetected by 2 successive CCAs).

ts > tc + 2tr (shortest frame should be detectable).

ta + td < ti (Ack should preempt retransmission of frame).

Combining all the above equations give the full constraint equation:

$$t_a + t_d < t_i < t_c < t_c + 2t_r < t_s$$

Some of these variables are defined according to the IEEE 802.15.4 and CC2420 radio specifications and some are Contiki implementation choices (in these specifications, a symbol is the time it takes to transmit half a byte).

t_a = 12 symbols = 192 µS (IEEE 802.15.4)

t_d = 10 symbols = 160 µS (IEEE 802.15.4)

t_r = 12 symbols = 192 µS (CC2420).

t_s > 736 µS or frame length > 23 bytes, all included.

Contiki choices:

Receiver wake-up period = 125 ms.

t_i = 400 µs.

t_c = 500 µs.

t_s = 884 µs (padding may be required in case of short addresses)

2.4.2.4. Fast sleep optimization

To maximally reduce useless energy consumption, ContikiMac switches off the receiver's radio as quickly as possible when it detects a non-valid frame or noise. This mechanism is referred to as "fast sleep optimization". Three situations cause a woken-up receiver to go immediately back to sleep:

a. ***Too long frame (*Figure 9*)***

   When a CCA detects radio-activity, this does not imply that an IEEE 802.15.4 frame is being received. It could result from any variety of radio transmission or noise. As the duration of an IEEE 802.15.4 frame cannot exceed 4.5 ms, the receiver returns to sleep when it did not yet observe any period of radio-silence 4.5 ms after waking up.

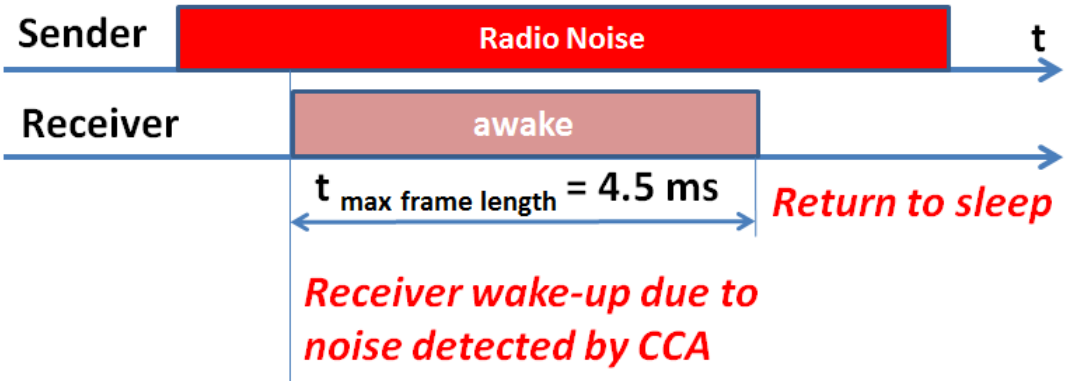

Figure 9: ContikiMac fast sleep optimizations: too long frame

b. Too long silence (*Figure 10*)

When data frames are being transmitted repeatedly, the time interval between successive transmissions is specified for ContikiMac: when the receiver, after being woken-up by a CCA observes a radio-silence longer than the allowed interval between successive frames, it returns to sleep.

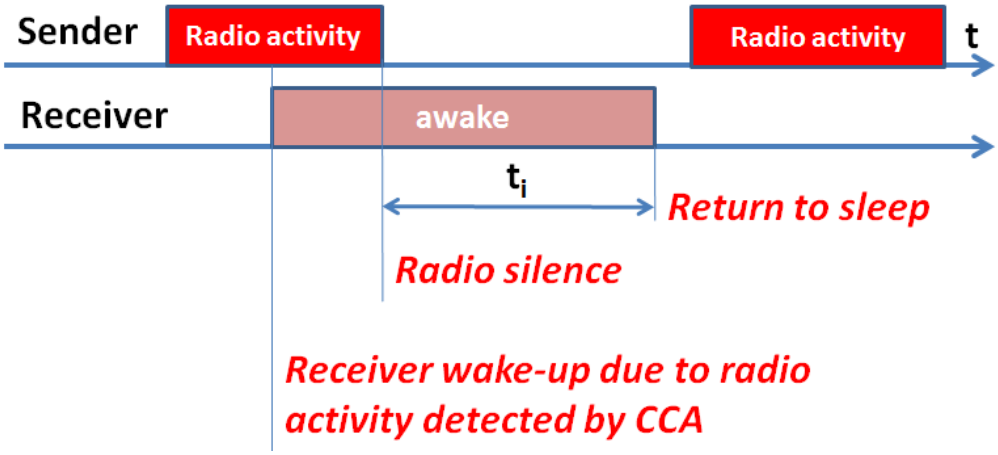

Figure 10: ContikiMac fast sleep optimizations: too long silence

c. No frame header (*Figure 11*)

After a silence of the correct duration, an IEEE 801.15.4 frame should be received. If the header of the frame being received doesn't obey the standards rules, the receiver returns to sleep.

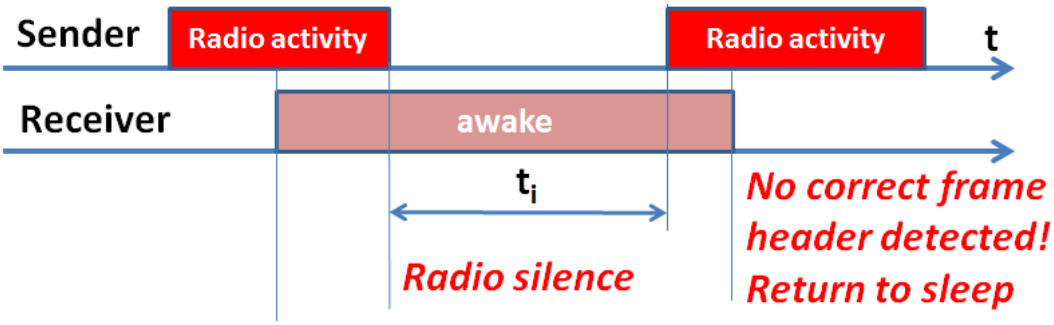

Figure 11: ContikiMac fast sleep optimizations: no frame header

ContikiMac uses two mechanisms to optimize the energy usage. One is by switching off the receiver's radio as quickly as possible Another one is by making the sender recording the wake-up times of receivers in its neighborhood so as to schedule the frame transmissions accordingly. This mechanism is referred to as "Phase Lock" or "Encounter Optimization". These two mechanisms are depicted on the Figure 9 up to Figure 12.

2.4.2.5 Encounter optimization (Figure 12)

To optimize the power at the sender side, it is desirable to minimize the number of retransmissions. From a received Ack, the sender can deduce the phase of the wake-up cycle of a specific receiver and memorize it. When a transmitter receives an Ack, it knows that the receiver was awake just before the transmission of the acknowledged frame. As the wake-ups occur strictly at a fixed rhythm, the transmitter can set up a table giving for each destination the optimal time to start transmitting a frame (phase locking the transmitter and receiver). As shown, the phase lock reduces the number of the transmissions hence, the energy efficiency.

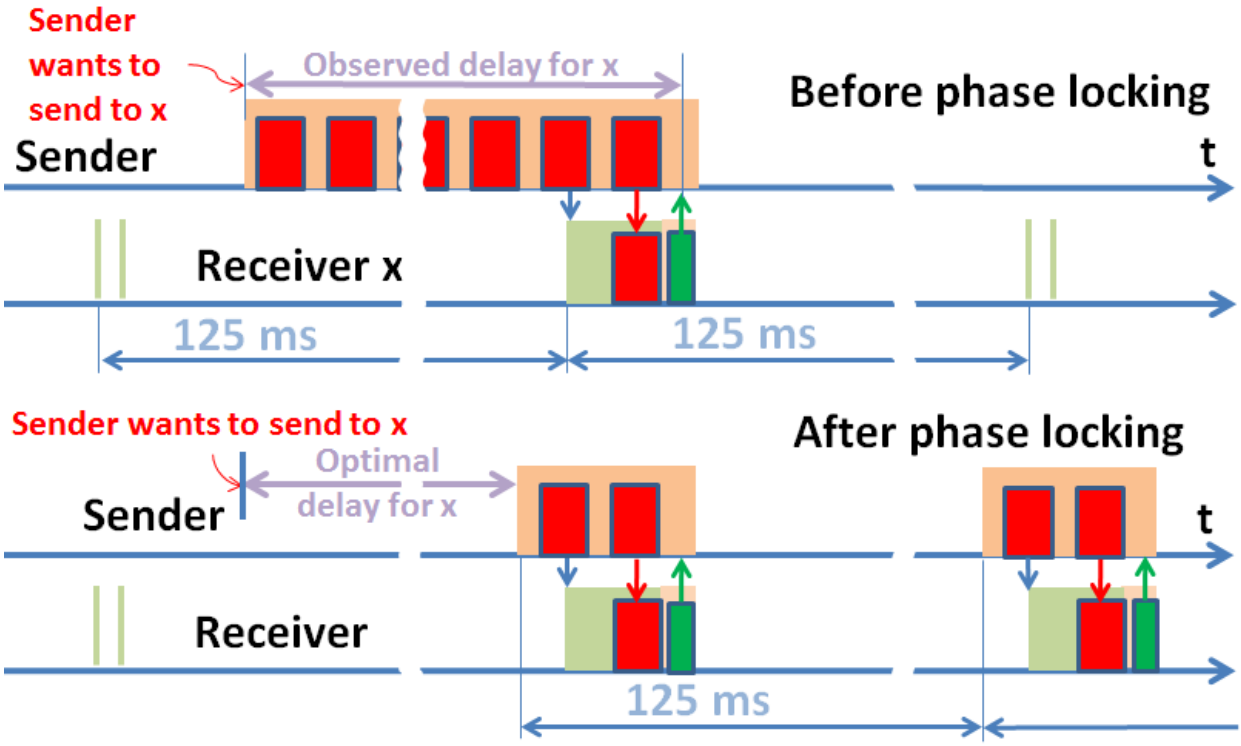

Figure 12: ContikiMac encounter optimization or phase locking

Ideally, with a good working phase lock, each packet should only be transmitted twice. Unfortunately, the time between the receiver wake-up and the sending of an ACK after successful reception of a packet, can vary widely[52]. When the receiver wakes up while a packet is being transmitted, that packet must first have been entirely transmitted and the next copy of the packet entirely received before an ACK is generated and transmitted. The delay between wake-up and ACK, when everything works properly, has a random component equal to the duration of an entire packet. Due to this imprecision, the repeated transmission has to start earlier and, most implementations use safety margin that is equivalent to the duration of 4 maximum length packets. In [52], Michel et al. propose to replace hardware acknowledgments by software ones containing the delay between wake-up and acknowledgment to allow a better optimization of the encounters.

2.4.3. X-Mac

X-Mac [46] is an older mechanism that does not provide the same power-efficiency as ContikiMac but has less stringent timing requirements. Figure 13 shows how it works.

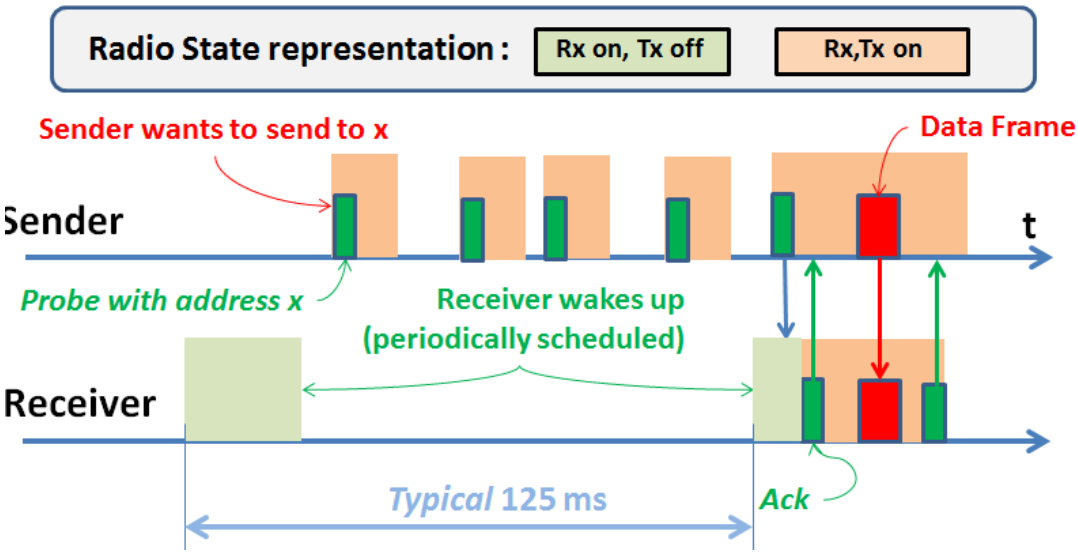

Figure 13: X-Mac unicast transmission

The green blocks transmitted by the sender are probes whereas the green blocks transmitted by the receiver are acknowledgments.

The sender, when it needs to send a packet, sends repeatedly a short probe that contains the destination address, leaving a random time interval between probes to allow other candidate senders to try their chance. When the intended receiver wakes up, it answers with an ACK and stays awake. When the sender receives that Ack, it sends its frame. When the receiver has stored the frame, it sends an ACK and returns to sleep.

The handling of broadcasts is identical to what is done in ContikiMac: the packet to be broadcasted is repeatedly sent during a full wake-up period. The only significant difference between ContikiMac and XMac for broadcasts is the interval between retransmissions: due to the random delays between probes, a XMac receiver cannot return to sleep as quickly as what the "fast sleep" rules of ContikiMac allow and therefore the interval between repeated broadcast packets can be longer, which saves energy when many broadcasts take place.

Two versions of X-Mac are available in Contiki: a full implementation and a simplified implementation called CXMac. The latter, for instance, sends the probes after a fixed time interval rather than after randomly chosen intervals. Both versions can optimize their encounters.

2.4.4. Low Power Probing (LPP)

LPP is a simple, receiver initiated, RDC protocol that can be useful to understand the principles of radio-duty-cycling.

2.4.4.1 Transmitting a packet to a specific destination (Figure 14).

All the receivers broadcast a probe packet each time they wake-up. If a node wants to transmit a packet, it can do so after hearing the probe from the destination. To send a packet, the sending node has to turn on its radio to wait for probe packets. The sender spends energy while listening, possibly a full sleep period, before it is allowed to send. The receiver has to broadcast its probe as it does not know who is waiting to transmit a packet. LPP can be very useful in shared frequency bands where each radio is restricted to send only a very small fraction of time to avoid congestion. None of the protocols with senders who have to repeatedly send a request to send can comply with the restrictions imposed on these frequency bands. Even in frequency bands with no such restrictions, LPP should allow a higher data throughput because it causes less radio traffic than other protocols. It has however also some serious weaknesses: when several motes have to send packets to a common destination, as is often the case near the root of a data acquisition tree, their packets will systematically collide when the common destination announces that it is awake. Only the random delays introduced by CSMA when retransmitting unacknowledged packets can alleviate that issue.

Encounter optimization is possible with LPP: it just consists in the sender delaying the switching on of its receiver to just before the moment the wake-up probe of the destination can be expected. The safety margin required for this optimization is much smaller than the one required for ContikiMac and XMac, because the receivers announce their wake-up as soon as they are woken-up.

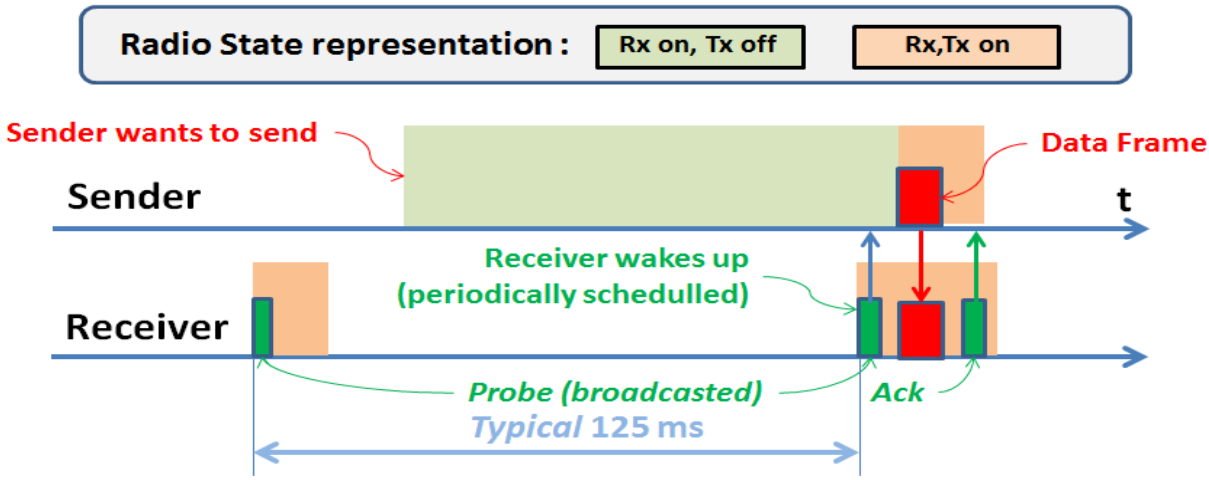

Figure 14: LPP unicast transmission

The receiver announces when it wakes up by broadcasting short probes, represented here by green blocks. The sender has to keep its receiver switched on to listen for the wake-up probes.

2.4.4.2. Broadcasting a packet (Figure 15).

Broadcasting in LPP is done by keeping the receiver of the sender on for a full wake-up period and by responding individually to all waked-up probes.

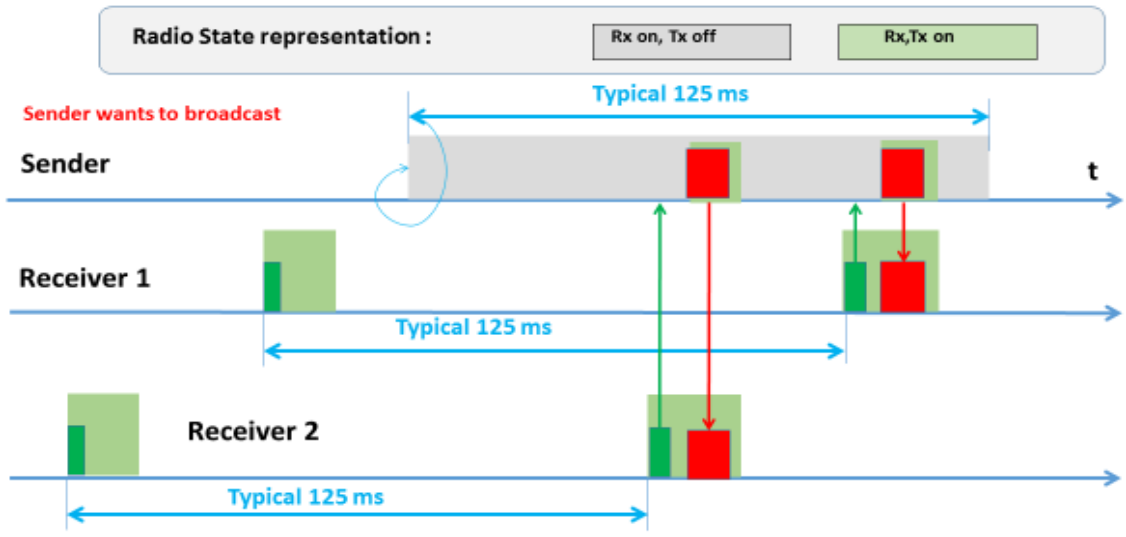

Figure 15: LPP broadcasts

The sender sends copies of the packet that needs to be broadcasted each time it notices the waking-up of a receiver.

Here again, multiple senders can be a problem, which cannot even be softened by CSMA, as broadcasted packets are not acknowledged. Figure 16 shows why all packets sent when two or more senders are broadcasting simultaneously are lost.

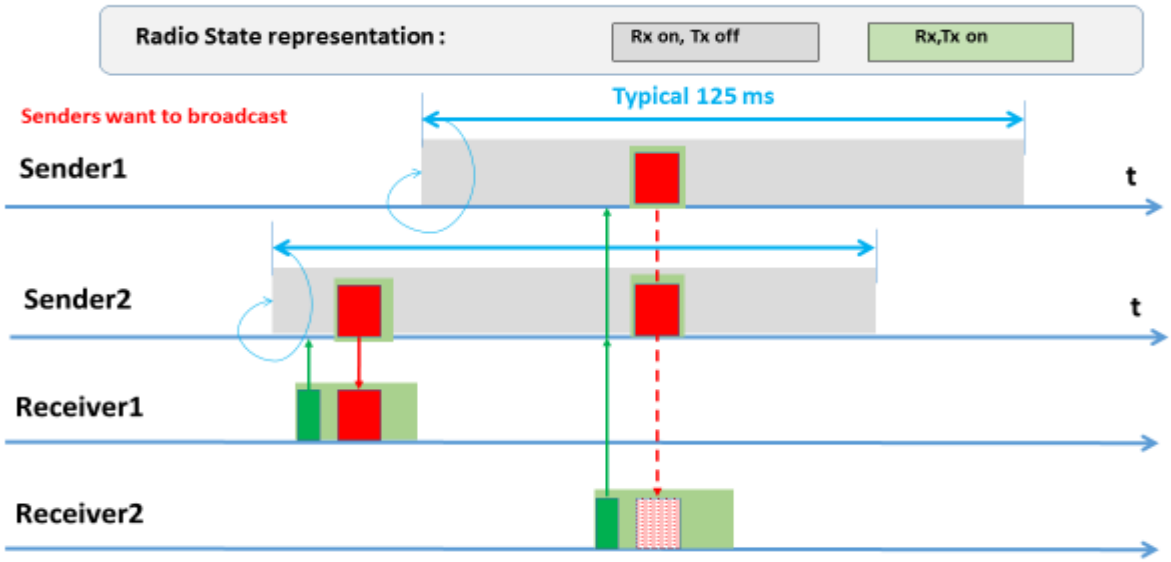

Figure 16: With LPP, senders cannot broadcast simultaneously
When receiver 2 wakes up, sender 1 and 2 will simultaneously send their respective broadcast packets.

2.4.4.3. Low Power Probing with pending broadcasts (Figure 17)

The inability of several senders to broadcast simultaneously has been circumvented by a trick called “pending broadcasts”. This trick will be explained here by means of an example shown in Figure 17, where sender 1 is already broadcasting and sender 2 wants to start broadcasting.

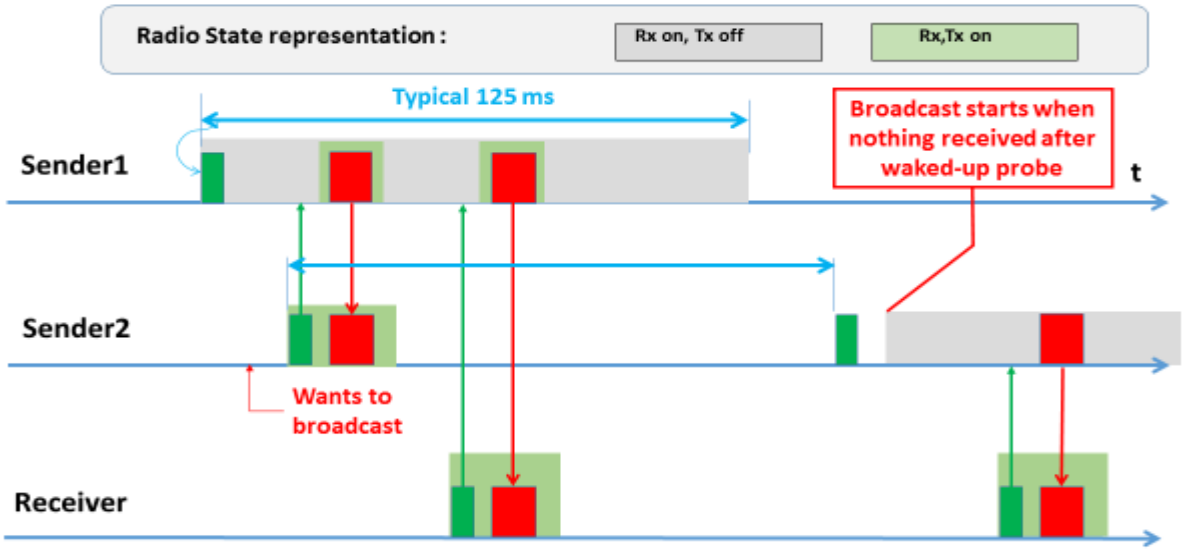

Figure 17: Low Power Probing with pending broadcast

Before entering the broadcast state, the sender waits till its own receiver does not get any broadcast when it wakes up.

Before starting a broadcast, a sender (sender 2 in the example) waits till its own receiver has broadcasted its waked-up probe. If another sender (sender 1 here) is broadcasting at that moment, the wake-up probe will be answered by a packet from that other sender, informing the candidate broadcaster that another broadcaster is already active and that it is of no use to start broadcasting. The candidate broadcaster has to wait till the waked-up probe of its own receiver is not answered before starting its broadcast. This trick increases the latency of broadcasted packets, but improves considerably their chances to reach their destinations, as will be shown experimentally in chapter 6. In Contiki, pending broadcasts handling is an option that can be activated in LPP.

2.4.5. NullRDC

The NullRDC protocol is an alternative RDC protocol that leaves the radio always on and that therefore can be used for testing or for comparison with the other RDC drivers.

Chapter 3: The Initial Research Project[1]

3.1. Introduction

Little or no attention is paid in the different documents introducing and defining the RPL layer 3 protocol to its coexistence with RDC protocols commonly used in layer 2 of WSANs for reducing power usage. This is not abnormal if one considers the underlying principles of a properly layered communications stack, as stated in the ITU-T standard X200 [53]. However, it is quite obvious that both the routing algorithm and the actual transmission of data packets can be negatively impacted when the probability is high that the destination mote of a transmitted packet is sleeping.

Quite a lot of literature has already been generated about performance of WSNs and its evaluation, recently an overview paper has made an inventory of the papers presented at the main conferences that have covered the state of the art during the last ten years [54]: according to that inventory a large number of studies are based upon simulations, but their validity was often challenged due to excessive simplifications in the simulations and the authors of the paper concluded that simulation studies should always be validated by means of experiments with real motes, validating a posteriori, the approach taken in this work. Similar concerns even motivated researchers to start a new series of conferences called REALWSN accepting only papers based upon experiments with real instead of simulated motes [55].

Exploring experimentally, in a real network, the impact of the choice of RDC protocols in an RPL network was initially the central topic of this thesis. For the real network, Zolertia Z1 [56] motes running the Contiki [57] operating system would be used, as some know-how about these devices had already been accumulated in the context of other projects. Three performance indices were considered: The Packet Delivery Ratio (PDR), the packet latency and the power consumption in each mote. To start, only "multipoint to point" (M2P) traffic from individual motes to the sink would be considered, as such traffic is the most common in WSNs and could easily be generated and monitored by means of the collect-view software included in the

[1] Part of this chapter comes from the publication "Experimental evaluation of message latency and power usage in WSNs" by Uwase, M.-Paule et al. presented at the 2014 IEEE International Black Sea Conference on Communications and Networking, BlackSeaCom 2014

Contiki distribution. By modifying collect-view, it would be, later, possible to extend the study to “point to multipoint” (P2M) and “point to point” (P2P) traffic.

The experimental study would have consisted in operating a 20 motes RPL network for several hours, in a countryside environment where no significant radio activity except that caused by the experiment could be observed in channel 26 of the 2.4 GHz band used by the motes. The experiment would have been repeated with the different RDC protocols available in Contiki, with several different parameter settings. During those experiments the considered performance indices would be recorded and finally a multifactorial analysis would be performed to evaluate the influence each selectable factor had on the observed performance indices.

3.2. Preliminary experiments

In order to prepare the planned experimental campaign, experiments to determine the properties of the motes that were available were done. The two varieties of motes were: Zolertia Z1 motes with internal ceramic antenna and the same motes with external whip antenna. These motes were mounted with nylon straps on wooden poles, approximately 120cm long, stuck in the ground. Both varieties of motes are shown in Figure 18 which is a picture of a mote with whip antenna, a mote with internal ceramic antenna and a mote protected against rainfall. The motes ran Contiki Version 2.6.

Figure 18: Three motes used for experiments

From left to right, a mote with an internal ceramic antenna, a mote with a whip antenna and a mote with whip antenna protected against rainfall.

A first series of experiments was intended to estimate the range of the radios and to evaluate the performance indices that are commonly associated with radio links. A second series was intended to check with a few motes the power measurements that had to be performed on all the motes of the RPL network and a third series checked the accuracy of the real-time clock build into the motes. These last measurements were made much later but are reported here among the preliminary experiments for reasons of coherence of the text.

3.2.1 RF power and communications range measurements

3.2.1.1 Performance indices

The range of a radio link is determined by the power generated by the sender, by the "quality" of the sender antenna, by the attenuation of the signal between sender and receiver antenna, by the "quality" of the latter and by the sensitivity of the receiver. When the distance between sender and receiver is much larger than the wavelength, all this can be approximated quantitatively, in free space, by means of the Friis formula [58], which shows that the power of the signal decreases proportionally to the square of the distance.

When interference with the signal itself reflected by the ground is considered, a decrease proportional to the fourth power of the distance is a better approximation for distances that exceed a threshold given by:

$$\mathbf{d > 2*\pi*h_t*h_r/\lambda}$$

where $\mathbf{h_t}$ and $\mathbf{h_r}$ are respectively the height above the ground of the antenna of the sender and the receiver, while $\boldsymbol{\lambda}$ is the wavelength of the radio signal[59].

As the present research is not related to the physical communications layer but rather to the layers managing that physical layer, the purpose of the experiments and measurements that were done was essentially gaining a qualitative insight in the properties of the used radio links to prevent gross experimental errors, rather than studying experimentally the laws of radio propagation. For one experiment (see further) the results were compared with those resulting from the Friis formula for free space: a good matching was observed in the range of interest for this work and a stronger attenuation was visible for distances exceeding the threshold mentioned in the preceding paragraph. The number of measurements points at these distances was however not sufficient for modelling the decrease in power.

In order to estimate the spreading of RF emitted power, a high frequency power meter was connected directly to the antenna connector of those motes equipped with such connector. The measured RF power was for all 20 tested motes within a range of +/- 1 dB around the nominal power programmed through the radio driver (0 dBm). It was concluded that no significant differences in the output power of different motes was to be expected.

All other experiments encompassed the entire path between sender and receiver, including the antennas. To characterize quantitatively this path, three indices were available: The Packet Delivered Ratio (PDR), the Radio Signal Strength Indicator (RSSI) and the Link Quality Indicator (LQI). The two later indices being evaluated by the radio hardware when receiving a packet. The first one gives the percentage of packets correctly received, the second one, for each received packet, the average level of the analog signal in dBm. The third is an estimation of the quality of a received packet.

In the cc2420 radio, used for our experiments, each 4-bit symbol is mapped into a 32 bit (called “chirps”) pseudo-random sequence. The receiver compares the received chirp sequences with the 16 sequences that correspond to a 4-bit symbol and selects the symbol whose hamming

distance is closest to the received sequence. The LQI value (between 0 and 127) is derived from the hamming distance between the received and selected sequence[60]. Different designers of integrated radio chips use slightly different mappings between the hamming distance and the resulting LQI value. Of course, the values of PDR, RSSI and LQI are somewhat correlated and many studies have been published on that topic [61]–[63] but none gives a clear, practically usable, relation between them.

Throughout all preliminary experiments with single hop radio links, the value of the three indices was stored. The analysis of the data relative to some 10 000 packets showed that for RSSI values above -90dBm, PDR was always above 99%, while LQI varied, almost randomly, as shown in Figure 19, between 110 and 90. When the RSSI was below -90dBm, both PDR and LQI values dropped quickly. As the PDR could be influenced by the RDC protocol that would be studied further on, it was not considered as a suitable index to characterize physically the radio links. So, it was finally decided to use mainly the RSSI value to characterize the quality of a radio link.

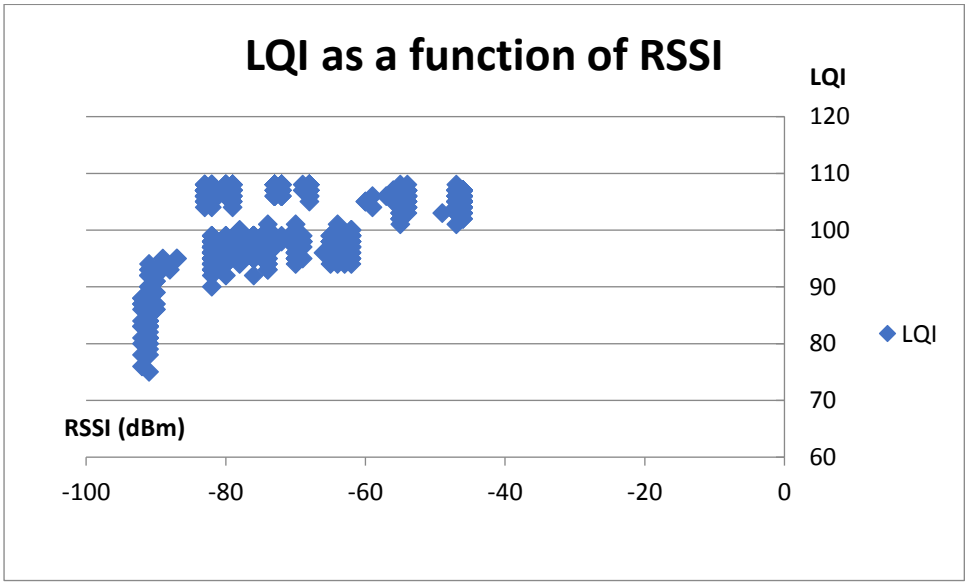

Figure 19: Observed relation between RSSI and LQI on a single hop radio link.
The RSSI and LQI values of some 10000 packets transmitted during various experiments were mapped.

3.2.1.2. The effect of the relative orientation of the motes

Two different motes were used to compare the efficiency of a whip antenna and the build-in ceramic antenna. The transmitting mote was positioned at 4m of a receiving mote, equipped with a whip antenna. The orientation of the transmitting mote with respect to the receiving mote was successively given 8 different angles, with steps of approximately 45 degrees between. Each measurement consisted in transmitting 100 packets and recording the values of the RSSI. The observed Packet Delivery Rate (PDR) was consistently, for all experiments 100%. The average value of the RSSI obtained from each group of 100 transmissions is represented in Figure 20.

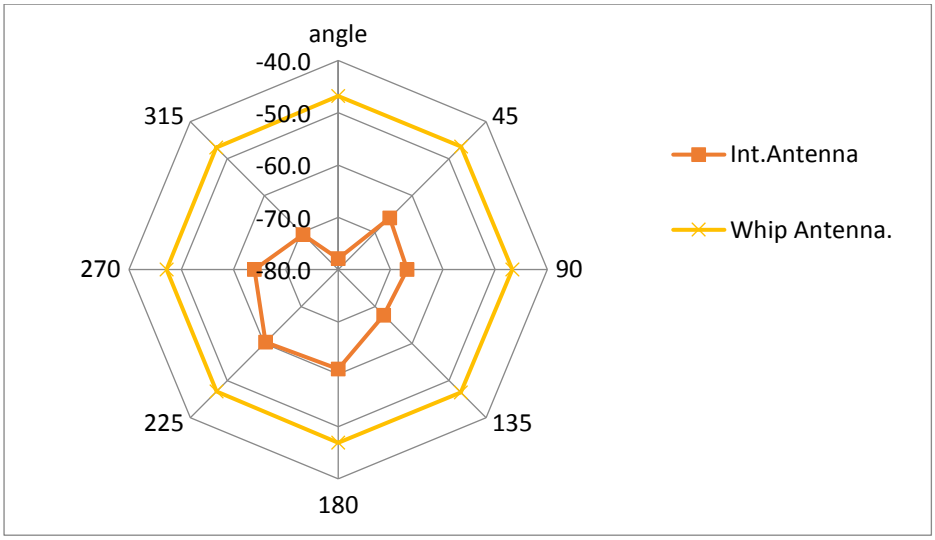

Figure 20: Average RSSI values for a 4m radio link as a function of the orientation of the transmitter and the type of antenna.

These results were obtained from 100 transmissions for each orientation. All RSSI values were within a +/- 1 dB of the average value.

As clearly visible in Figure 20, the internal antenna of the z1 motes is far from omnidirectional and transmission in front of the mote is especially poor. First, the presence of batteries in front of the antenna was suspected but experiments with the battery holder removed from the mote gave only slightly better results. The radiation patterns shown in the datasheet of the build-in antenna indicate that, in fact, the problem resides most probably with that antenna itself [64]. As our experiments were all planned in a relatively flat garden, where differences in altitude between motes did never exceed 2m, we did not study carefully the vertical radiation diagram of our motes, just observing that moving vertically motes between 50cm and 150cm did not significantly affect the RSSI figures.

3.2.1.3. Practical radio range measurements

To estimate the distances that can be covered by single hop radio links, the experiments done for the antenna comparisons were repeated, but instead of varying the orientation of the transmitting mote the distance between sender and receiver was increased by steps of 3m along a few straight lines in the garden where most experiments were done, and by steps of 20m in a large prairie, when the garden proved to be too small for fully testing the radio ranges at full transmitting power. First, experiments in the garden along a straight line between oak trees, called “direction F”, were done with a whip antenna, with the build-in ceramic antenna with the best orientation and with the worst. Figure 21 shows the observed RSSI values.

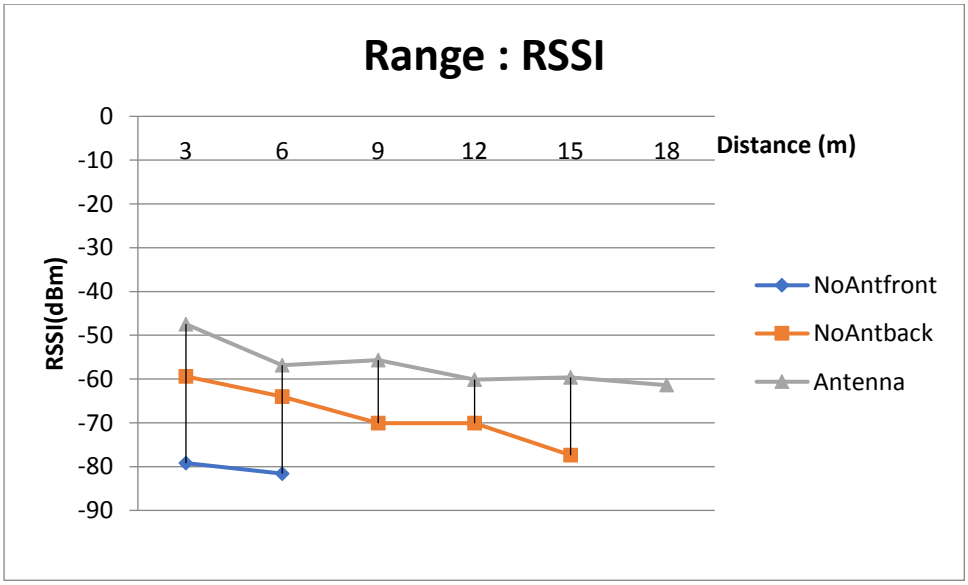

Figure 21: RSSI values as a function of the distance between transmitter and receiver
Three sets of measurements were done: with a whip antenna (antenna), with the build-in ceramic antenna, oriented for the best RSSI value (NoAntback) and with the same antenna, but oriented for the worst RSSI value (NoAntfront)

Additional measurements were made along different straight lines through the garden and with different transmission power settings. These measurements were only done with a whip antenna, as it was too difficult to make accurate and reproducible measurements with the build-in antenna, because of the influence of the mote orientation. The results summarized in Figure 22 for two almost perpendicular directions show that the configuration of the grounds, their vegetation and the proximity of trees plays a non-negligible role in the radio propagation. These measurements were repeated several times and the results were reproducible within +/- 2 dBm.

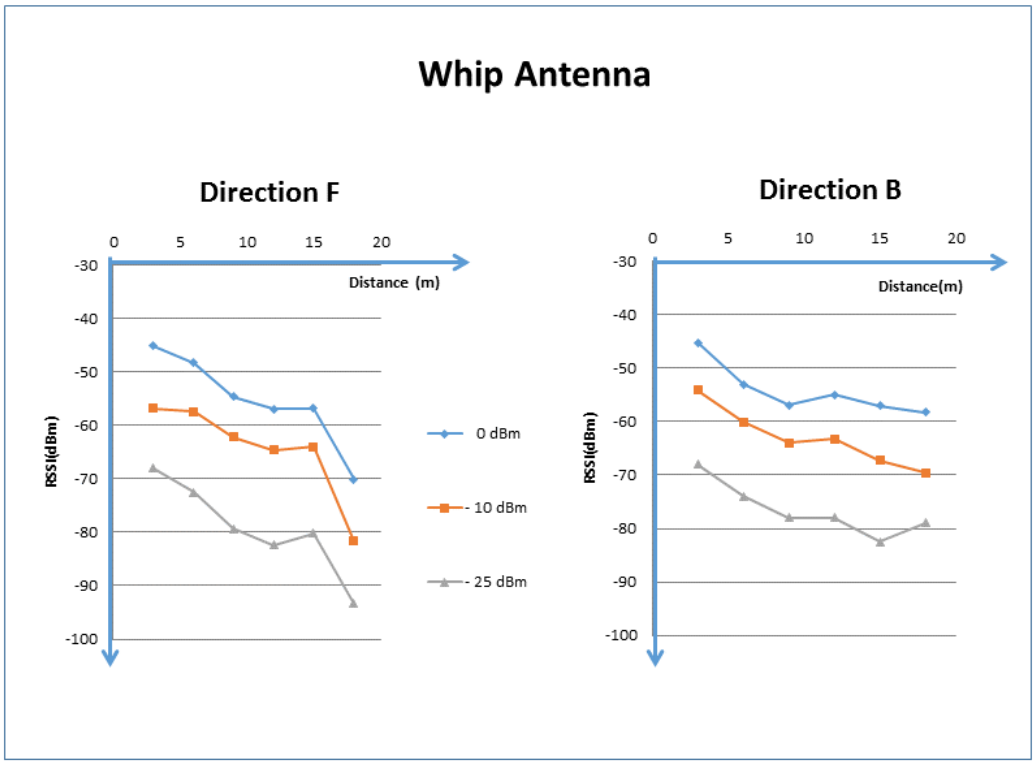

Figure 22: Signal attenuation as a function of the distance in two different directions in a garden

Both directions were obstacle-free, but in direction F, three oak trees were located at approximately one meter from the straight line between sender and receiver, while for direction B, there were no trees at less than 3 meters. The measurements were made with three different transmission powers the same day under equal meteorological conditions.

Finally, a set of measurements was made in a large, open prairie far from any buildings, trees or power-lines, surrounded by dense forests between the villages of Jehonville and Anloy. Both sender and receiver were equipped with a whip antenna, and the maximum transmission power (0dBm) was used.

Figure 23 shows that reliable radio links can operate, in ideal propagation conditions, over distances exceeding 160m and that the measured RSSI matches quite accurately the RSSI computed by means of the Friis formula for free space propagation, except for the longer distances, where the measured values are significantly lower than those predicted by Friis, which is due to the effect of wave reflections on the ground.

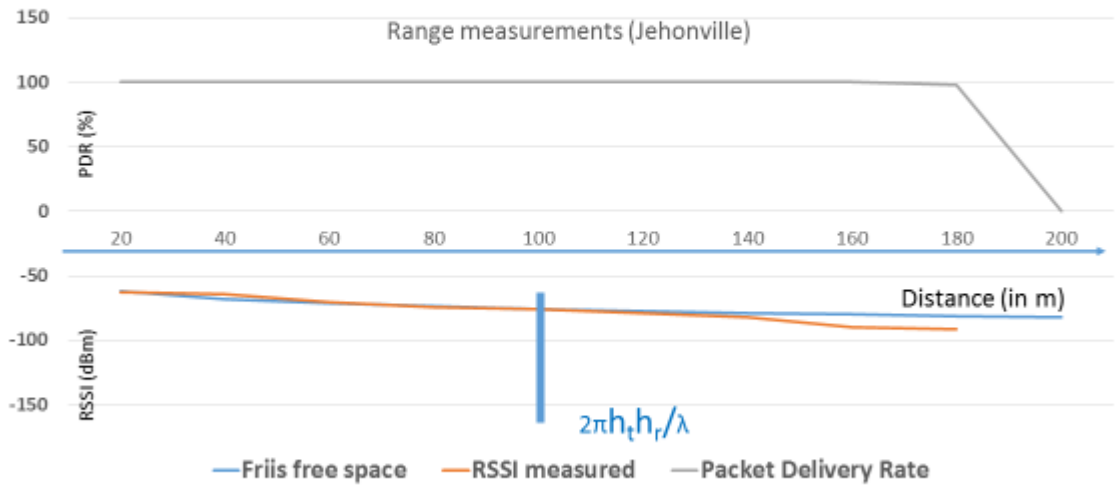

Figure 23: Range test in an open field

The upper graph gives the packet delivery ratio which remains at 100% up to 160m and falls rapidly to 0 at 200m.

The lower graph gives the measured RSSI and the computed RSSI according to the Friis formula for free space propagation. The height of both sender and receiver antenna were approximately 1.5m above the (dry) grass.

3.2.1.4. Influence of meteorological conditions

Comparisons of the results obtained during this preliminary series of measurements and results deduced from other measurements done during the following months allowed to conclude that

- Temperature had little influence on the observed RSSI values.
- Snow on the ground also had little influence on the RSSI.
- The rain protections surrounding the motes had no influence on the radio links
- Moderate rain (5mm per hour) reduced the RSSI values by 3 +/-1 dB (Figure 24)
- Bushes with abundant wet leaves between sender and receiver antennas could cause reductions of 15 +/-3 dB (Figure 25)
- The same bushes, in winter, without leaves, caused a reduction of less than 2 dB.

Rain

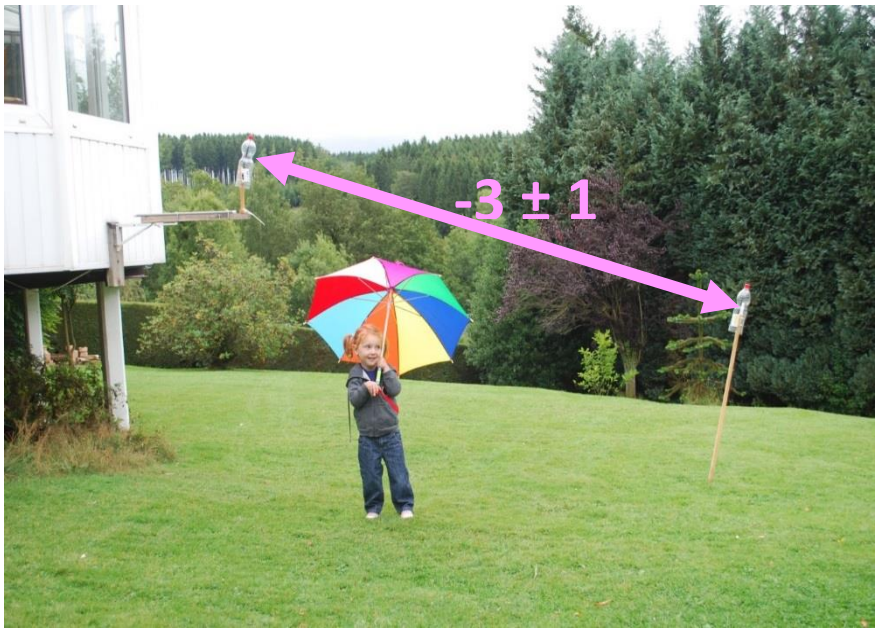

Figure 24: Moderate rain (5mm/hour) causes a RSSI reduction of 3+/-1 dBm

Bushes

Figure 25: Wet leaves on bushes cause important RSSI reductions

3.2.2 Power consumption

To evaluate the power used by a mote, two approaches are possible: the first, hardware oriented, consists in measuring the supply voltage and the current drawn from the power supply and integrating the product over time, the second, software oriented, proposed in 2007, by Dunkels et al. [65] consists in adding to the operating system of the motes four variables that would totalize the times the sender, the receiver and the CPU were active, as well as the time the mote was in sleeping mode. These variables are called the "activity timers". By multiplying the times recorded in the activity timers 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. [66] 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 [67], sampling the mote current at 1000 Hz.

More recently another device, with even better resolution has been presented at the RealWSN workshop in Como [68] for similar purposes. However, with more recent motes such as the Zolertia Z1, these high frequency data loggers are quite useless to calibrate the power requirements of parts of the mote as modern motes contain an internal power management subsystem which draws its energy from a capacitor that is replenished periodically by a built-in DC to DC converter from the external batteries whenever needed. Lengthy observations by means of an oscilloscope (Figure 26 gives a typical example) showed that power drawn by the mote was a poor indicator of the instantaneous activity of the radio and that only current measurements averaged over periods much longer than the time constants due to internal capacitors could give an accurate image of the real power drained by a mote from its batteries.

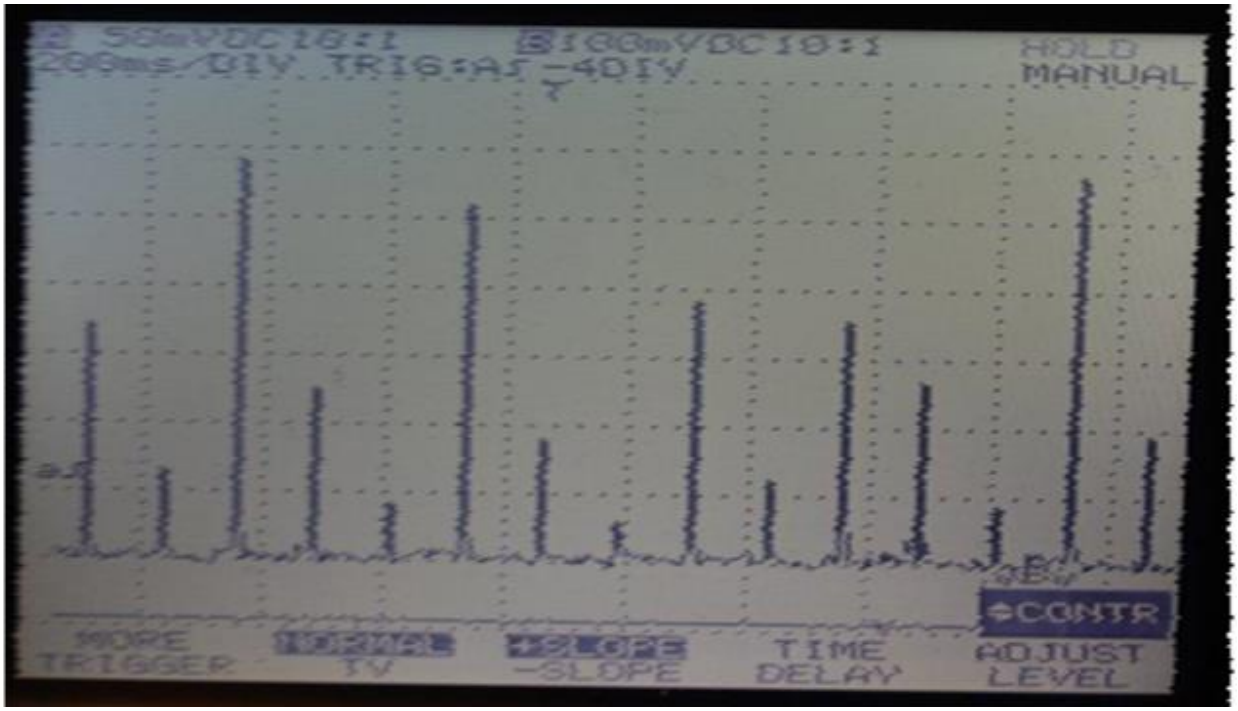

Figure 26: Oscilloscope trace of the power used by an idle mote
The current peaks show clearly that the internal DC to DC converter fetches periodically power from the batteries to keep the (inaccessible) internal power reserve of the radio constant.

Based on these observations, exploratory experiments were done to try to nevertheless calibrate the power consumption of the different parts of the mote and avoid costly current measurements in the field.
As power requirements of electronic components can vary due to tolerances in the manufacturing process, the energy used by 10 randomly selected Z1 motes for sending 100 packets were compared. The power level for transmissions was set to -15 dBm as in all experiments described in this paragraph. The measured power requirements for transmitting 100 packets were quite uniformly spread between 298 and 342 mJ. This justifies why Hurni et al. [66] recommend calibrating 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 (energest variables) were recorded. In fact, each experiment had to be done twice, because a USB connection between receiver and a laptop was necessary to record the values of the activity timers, but such connection imports power from

the laptop into the mote. The difficulty was avoided by running each experiment two times, once with the USB connection to record the activity timers and once without to measure the power used.

Table 1Table 1 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 MSP430 processor [69] and the cc2420 radio[60]. 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 mode might be caused by Contiki not accounting the devices that are permanently on such as the 32768 Hz real time clock. The small fraction of the time (less than 0.5%) the transmitter was active during all the calibration experiments explains why the error on the corresponding value can be large. 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, it was decided to restrict the usage of the activity timings to comparisons of different protocols on real or simulated motes and to use real motes and hardware measurement techniques when actual power values need to be evaluated.

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

Table 1: Power coefficients per activity

3.2.3 Latency measurements

Throughout this work, latency is always considered between application layers. A message is sent when the **unicast_send** or **broadcast_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 Figure 27:

- 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.

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.

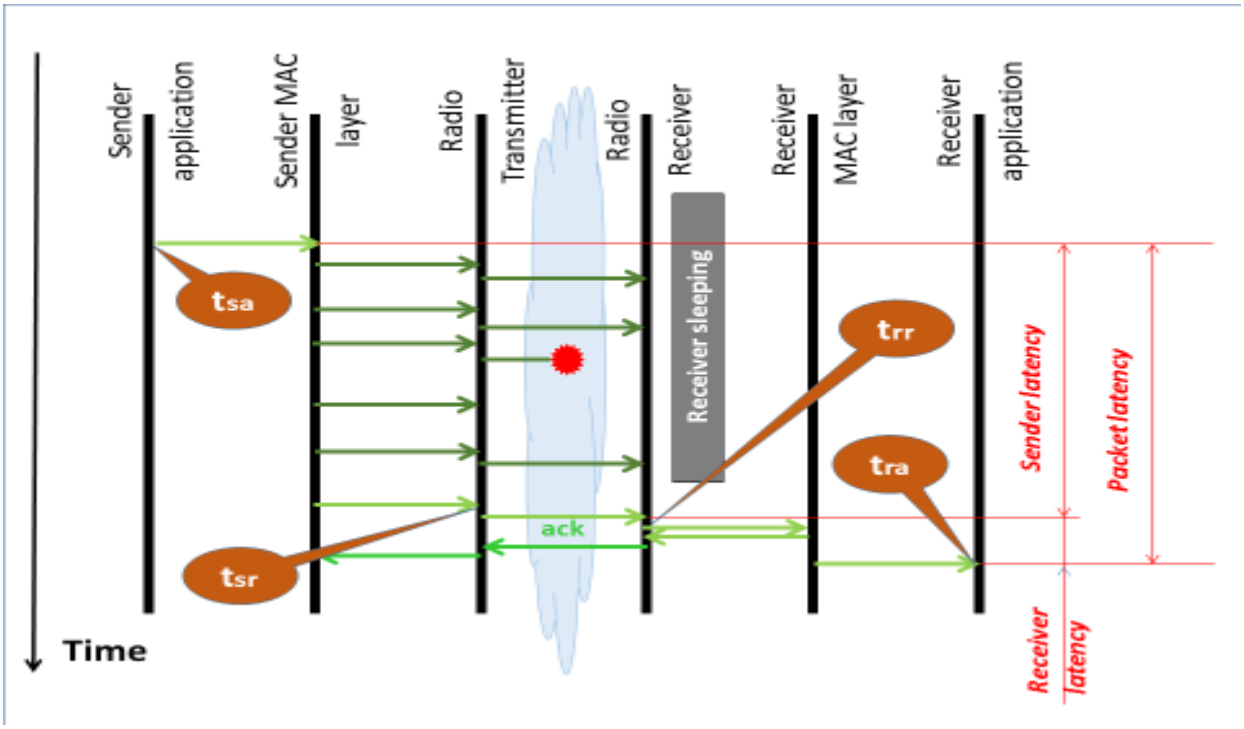

Figure 27:Definition of sender latency, receiver latency and packet latency

The red dot in the middle of the transmission cloud represents the loss of a packet, due to an external cause.

The relevant time stamps are:

tsa: the moment the application layer in the sender calls the send function.

tsr: the moment the radio of the sender transmits the Start Of Frame (SOF) byte.

trr: the moment the receiver radio recognizes the SOF byte

tra: the moment the receiver application gets the entire packet from the lower layers.

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 however would require synchronized real-time clocks in sender and receiver. If the system under study does not include such synchronization, adding it would create communications overhead that could jeopardize the accuracy of the observations [42], [43], [60]. 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.

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 real-time clock at the moment the radio sends the Start of Frame byte. The second, for the receiver, copies in a variable the reading t_{rr} of the receiver real-time clock at the moment the radio receives the Start of Frame. 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 and one additional radio interrupt in the sender and in the receiver. This technique, however, cannot be used when the application layer verifies the correctness of its messages or encrypts them as the radio driver modifies the contents of the payload.

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 for the preliminary experiments: 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 3.2.4) 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 12 ms 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 the implemented latency measurements, but as in these preliminary experiments the interval between packets was chosen to be 1 s, the resulting error could probably be neglected.

3.2.4 Hardware circuits for power and latency measurements

To measure the current drawn by a 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 (Velleman instruments PCS10, connected to a laptop). A low pass filter constituted by a 1000 µF capacitor and the 2Ω resistor, ensures that no significant aliasing error will result from the 100Hz sampling. Figure 28 shows the block diagram of the data acquisition system and how it is connected to unmodified Z1 motes.

As, to measure the power used by the sender and receiver mote, a hardware accessory and a commercial data logger are needed, the same equipment can be used to measure accurately the latency of packets, between the sending and the receiving applications. For that purpose, the sending and receiving application software has been extended by a few instructions, setting and resetting one of the parallel I/O pins of the mote whenever a packet is sent or received. As the pulses on this I/O pin are too short (15µs) to be reliably observed by the data logger sampling its inputs at 100Hz a monostable pulse shaper has been included to widen the pulse to some 12ms, ensuring reliable recording of the pulses by the data logger.

The power is obtained from a 4.5V battery and reduced to 3.3V by means of two silicon diodes causing each a voltage drop of 0.6V. This allows using the same battery for the differential amplifier, the pulse shaper and the mote.

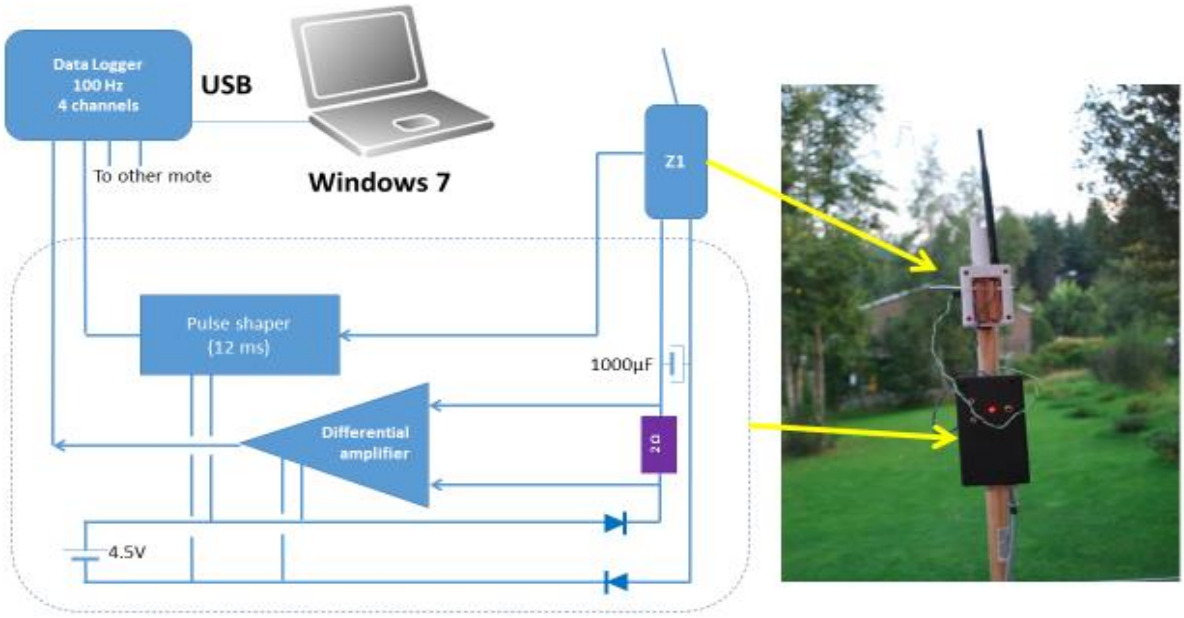

Figure 28: Hardware circuitry for measuring power usage and packet latency on a single hop radio link

Table 2 gives an example of the output of the data logger when a packet is successfully transmitted. In line 10, CH1 shows that the sender has started sending a packet. In line 14, CH3 shows that the packet has been received. Its latency was 40ms +/- 10ms as it could have been sent anytime between 90 and 100ms and received anytime between 130 and 140ms.
CH2 and CH4 show that the power used by sender and receiver is increased due to the transmission of the packet, but this increase is spread over almost 100ms.

N	**CH1**	**CH2**	**CH3**	**CH4**	**Time**
	V	**mA**	**V**	**mA**	**s**
7	0	0.7	0	0.7	0,07
8	0	0.7	0	1.3	0,08
9	0	1.2	0	1.2	0,09
10	3	1.3	0	1.1	0,10
11	3	1.2	0	1.0	0,11
12	0	1.3	0	1.0	0,12
13	0	2.4	0	0.9	0,13
14	0	4.3	3	1.6	0,14
15	0	3.6	3	1.8	0,15
16	0	2.9	0	1.6	0,16
17	0	2.5	0	1.4	0,17
18	0	2.2	0	1.3	0,18
19	0	1.9	0	1.2	0,19

Table 2: Partial view of the output of the data logger showing the successful transmission of one packet

N is the sample number and Time gives, in seconds, the time elapsed since the start of the logging.

CH1 is the output of the pulse shaper on the sender mote.

CH2 is the current drawn by the sender.

CH3 and CH4 have the same roles as CH1 and CH2 but for the receiver.

3.2.5 Real-time clock frequency comparisons

The Zolertia Z1 motes contain a 32768Hz real-time clock, similar to those used in watches, which is used for all timings that are not less than one tick of that clock, or 30.5µs. As will be shown in next chapter, even if the absolute frequency of that clock is of little practical importance, the relative difference between the clocks of communicating motes might be of great importance. In order to evaluate the order of magnitude of the relative frequency differences to be expected, 21 motes were selected at random and their real-time clocks compared.

As it proved physically difficult to connect directly the real-time clock of a mote with a high-precision frequency meter, it was decided to use the real-time clock of a laptop, running Linux, as a reference for comparisons. The mote being tested ran for 10 minutes a program that wrote every minute to the serialdump file of the linux laptop, the number of minutes that the experiment had been running. As each entry in the serialdump file is timestamped with a millisecond resolution, it was easy to read the difference in duration between a minute according to the clock of the mote and a minute according to the clock of the laptop. Table 3 shows such a serialdump file, from which one can conclude that the mote clock was fast with respect to the laptop clock by 133 ppm.

Figure 29 shows the relative differences between the clocks of the 21 selected motes and the clock of the laptop used for the experiment. The three first motes shown belonged to another batch, purchased two years earlier than the 18 others and had another brand of real-time clock. To check the influence of temperature on these results, the experiments were repeated outside on a sunny day, first around noon and then around midnight. No significant differences were noticed.

20160921 00:47:30.860 Rime started with address 136.0
…
20160921 00:47:30.883 Starting 'Printing timing messages'
20160921 00:47:30.883 Timer check
20160921 00:47:30.884 Constant delay between messages = 7680 etimer clock ticks
20160921 00:48:30.880 1
20160921 00:49:30.880 2
20160921 00:50:30.879 3
20160921 00:51:30.878 4
20160921 00:52:30.877 5
20160921 00:53:30.876 6
20160921 00:54:30.876 7
20160921 00:55:30.875 8
20160921 00:56:30.874 9
20160921 00:57:30.873 10
20160921 00:58:30.872 11
clock difference = (880-872) / 60000 = 133 ppm

Table 3: Example of a serialdump used to compare the real-time clock of a mote with the clock of a laptop

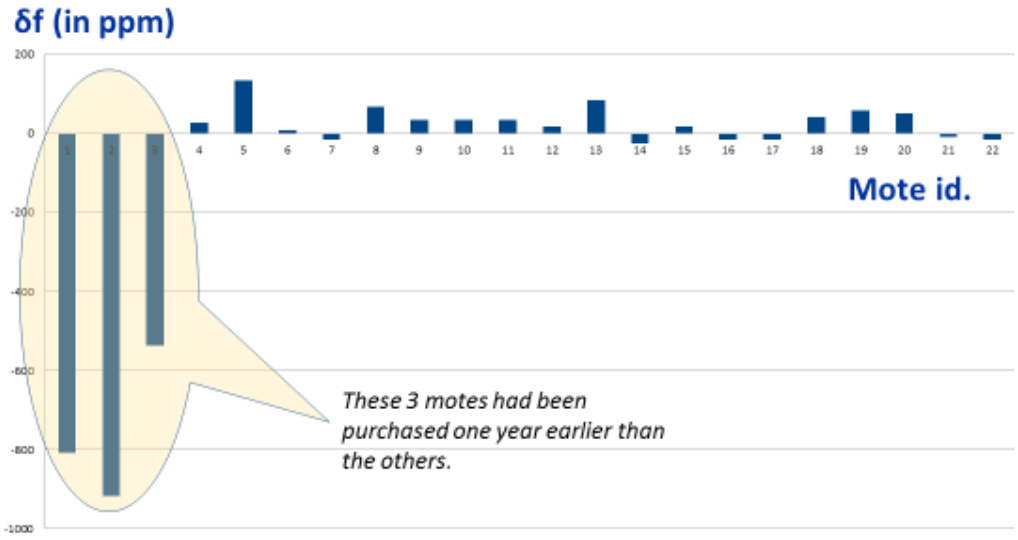

Figure 29: Relative differences between the frequencies of the real-time clocks of different motes and the real-time clock of a Hewlett Packard laptop running Linux

From these experiments on needs to conclude that important differences in the real-time clock frequencies of motes from the same manufacturer can exist. Even if motes from different manufacturing batches are separated, differences of 100ppm are common.

3.3. First exploration of Radio Duty Cycling (RDC) protocols

In parallel with the experiments described above, other team members had studied the RPL protocol and tried to improve it by adding priority classes in the routing algorithm [70]. Their experiments gave good results when the COOJA simulator was used, but, with real motes, the RPL protocol failed to build, even after hours, a stable DODAG and the overall PDR was unacceptably low. However, by replacing the usual RDC protocol ContikiMac by Nullrdc, a protocol that leaves the radio always on, the real network behaved well, just like the simulated one. This was a strong incentive to use the tools described in section 3.3.2 to study in detail the behavior of the different RDC protocols available in Contiki.

3.3.1. The hardware set-up for studying RDC protocols

The two augmented motes described in section 3.2.4 were used to study the behaviour of a single hop radio link running the different RDC protocols implemented in Contiki, version 2.6. As both perturbations by external radio sources, such as messages exchanged between other motes in the RPL network, and protocol bugs were suspected culprits for the observed malfunctions, the experimental set-up was extended with two devices:

- A second radio link, independent of the one under study, but able to generate perturbing well defined radio traffic in the same frequency band.
- A so called “sniffer” composed of a dongle (TI, CC2531DK) and a free sniffer software (TI, SmartRF) running on a separate laptop to record, without influencing it, all radio traffic between the four motes involved in the experiment.

Figure 30 shows the experimental set-up used for investigating the RDC protocols.

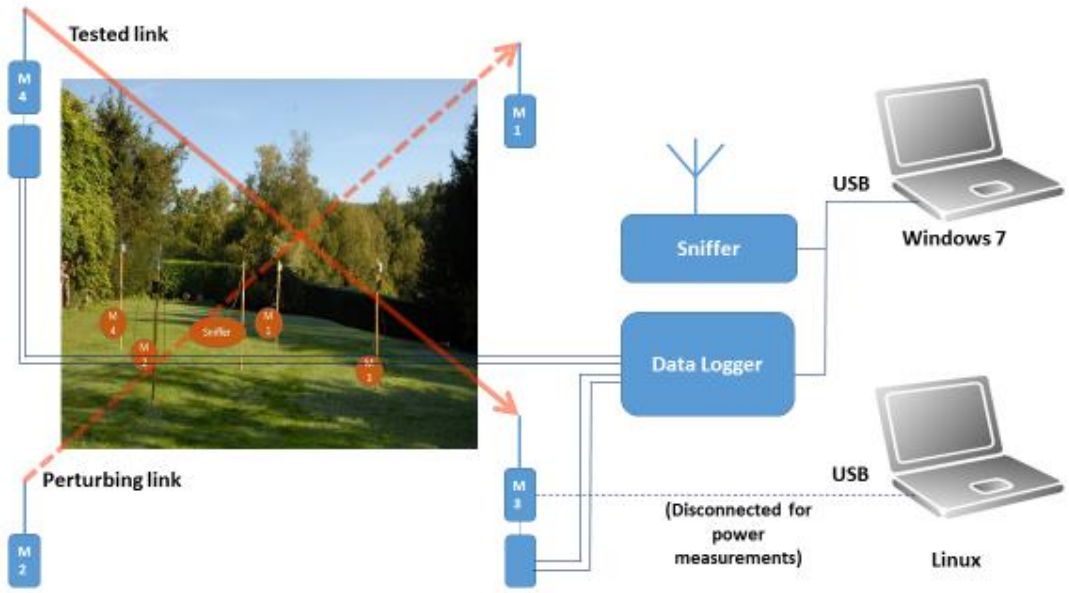

Figure 30: The experimental set-up for an initial exploration of the RDC protocols

3.3.2. Contents and organization of the tests

Three RDC protocols, readily available in Contiki version 2.6, were tested, and their performance evaluated. They were ContikiMac [47], XMac [71] and lpp [48]. The same tests were performed with nullrdc for comparison purposes. The RDC protocols were successively tested with three different Channel Check Frequencies (CCF), 4, 8 and 16Hz. The actual CCF of nullrdc was 128Hz. Each test consisted in transmitting from M4 to M3, at an average rate of 1 packet per second, 1200 application level packets containing 50 bytes each as payload. Each test was repeated three times, once with no perturbing traffic between M2 and M1, once with the same application level traffic between M2 and M1 as the one between M4 and M3 and finally once with M2 generating the same application level traffic as M4, but in vain, as M1 was switched off. The last test was intended to create a more perturbing traffic, as the MAC layer would make three attempts to send the packet and the ContikiMac and XMac protocols would try to send their packet repeatedly during a full wake-up period.

When duty cycling is used in a radio link, the packet latency has two parts: one is caused by a multitude of delays in the handling of the packets, the other is the time between the sending of a packet and the next wake-up of the receiver.

To measure the average packet latency, the sending of the test packets should be totally desynchronized from the receiver wake-ups. For that reason, the interval between sending successive test packets has been programmed to have a random value, uniformly distributed between 10ms and the double of the desired average interval minus 10ms.
Collecting from each of the 1200 transmitted packets values for each observed index would allow to compute statistically valid mean values and even the standard deviation on the mean values, as 1200 measurements can be considered as 34 sets of 34 measurements and that 30 sets of 30 are generally considered sufficient to obtain meaningful statistics.

3.3.3. The results of the tests

The three tests required one hour of testing per RDC and CCF combination, or 10 hours of testing for the three RDC protocols and nullrdc. They could be done in one day of experiments and were repeated several times, with different levels of sending power.
The analysis of the collected data gave results in line with the expectations for the used power, but it appeared almost impossible to extract reliable quantitative data for the PDR and latency indices.
As described in section 3.2.4 every packet generated by mote M4 and every packet received by mote M3 results in a pulse recorded by the data logger. When the PDR is close to 100% and the latencies are stable, as was always the case with nullrdc, it is easy to associate a sender pulse with the corresponding receiver pulse but, when the PDR is very different from 100% or when latencies exceed largely the time intervals between packets, this association becomes almost impossible guesswork. Various heuristics have been imagined and tested for associating sender and receiver pulses. The best one did build a table with two columns, the first containing the times packets have been passed by the sender application to the MAC layer (t_{sa}) and the second the times these same packets have been received by the receiver application (t_{ra}) (see Figure 8 for the definition of the symbols t_{sa} and t_{ra}). It works as follows:

1. All recorded t_{sa} values are written down in the first column in the order the corresponding packets were sent.
2. The t_{ra} values 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_{sa} with no corresponding t_{ra}, implying that packets have been lost.
3. If a t_{ra} cannot be added due to causality,

3.1 If one of the five previous sent packets was considered lost, all t_{ra} after the lost packet are shifted back in time, creating space for the t_{ra} that could not be added

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

This heuristic was very helpful to explore the lengthy log files (12000 lines per test!) but did not yield convincing values for the PDR and latency indices due to frequent abnormal situations in the log files.

However, by exploring manually the files, some threads of a few hundred packets that could easily be analyzed were found. The resulting indices could not be considered as statistically representative of the behavior of the radio link but became an incentive to develop better means to study in depth that behavior. Figure 31, for instance, shows clearly the trade of energy for latency between nullrdc and the tested RDC protocols and Figure 32 shows a comparison between latency and energy figures between ContikiMac and XMac for various traffic profiles. The figure shows that the traffic does not influence the power consumption but strongly influences the latency.

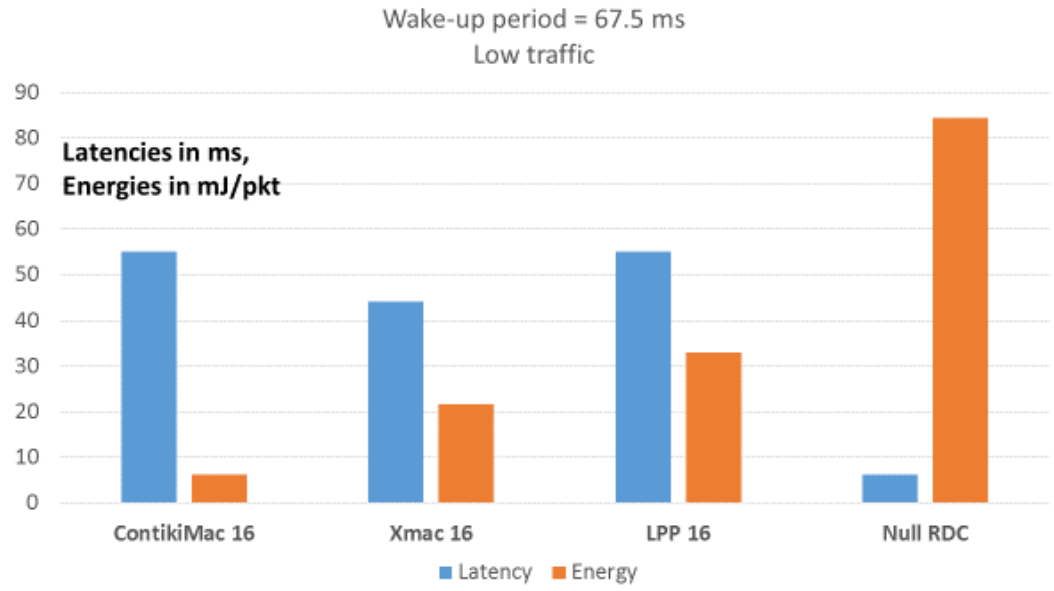

Figure 31: Latency vs. power compared for three RDC protocols and no RDC

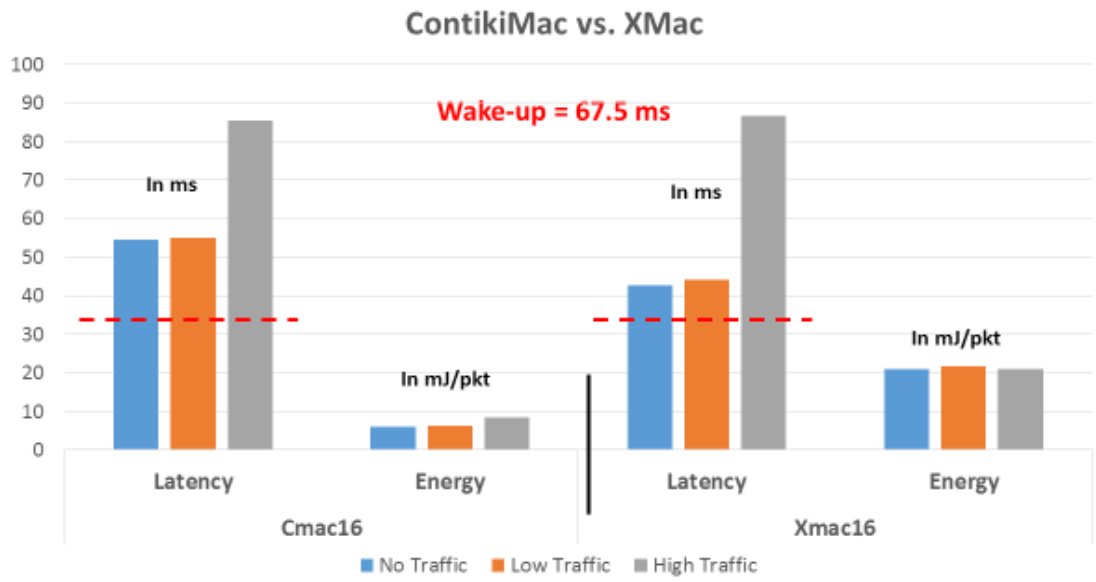

Figure 32: Latency and power comparisons between ContikiMac and XMac under various traffic conditions

The red horizontal discontinued lines represent half the wake-up period. Due to the random interval between packets, the average latency cannot be lower.

3.3.4. Qualitative observations

The manual analysis of the log files and the sniffer records yielded several important symptoms of malfunctions of the different RDC protocols. They are listed here for each of the protocols.

3.3.4.1. ContikiMac malfunctions

- While the PDR was close to 100% sometimes total black-outs, lasting several minutes, occurred. The recovery was spontaneous.
- The PDR for packets with a RSSI below the CCA threshold was non-zero in presence of perturbing traffic.
- Many hardware generated acknowledgments where not recognized by the sender mote, causing useless packet retransmissions by the CSMA MAC layer when the RSSI was below the CCA threshold.
- The encounter optimization mechanism was frequently reset without any apparent reason.

3.3.4.2. Xmac malfunctions

- The encounter optimization mechanism was sometimes reset without any apparent reason.

3.3.4.3. LPP malfunctions

- Packets with a low RSSI were triplicated unless the debug option was enabled in the LPP driver.
- When the receiver of the perturbing link is active, long, self-recovering, black-outs occur occasionally in the tested link.

3.4. Conclusions

An affordable technique has been developed for studying experimentally the RDC protocols and this technique has uncovered several abnormal behaviors of the three asynchronous RDC protocols available in Contiki.

The developed technique, however, was found to yield experimental data that were very hard to interpret through efficient, automated, procedures and, moreover, that technique was not scalable to entire RPL networks deployed in the field as each observed mote had to be connected to a central data logger via wires.

The range of the motes equipped with a whip antenna was too large for building a multi-hop RPL test network in the available garden.

Chapter 4: A testbed for exploring experimentally RDC protocols

4.1. Introduction

The preliminary experiments described in chapter 3 brought to light several malfunctions in the three asynchronous RDC protocols implemented in Contiki versions 2 and 3. These malfunctions were not observed when the same experiments were done with the Cooja simulator rather than with real motes. As these malfunctions were likely to affect seriously the operation of a multihop network such as RPL, it was decided they should be understood and eliminated before the interactions between the RPL and the RDC protocols should be further explored.

When packet latencies grew higher than the interval between successive packets or when many packets got lost or duplicated, the measurement tools described in chapter 3 proved unpractical because it was not possible to associate unique packet sequence numbers with the sending and receiving events recorded by the data logger. Moreover, due to the wires required between the data logger and every monitored mote, these tools could not practically be scaled up to monitor, in a later phase of the research, a RPL network with at least twenty motes, deployed outdoors. A new approach to observing the behavior of the motes, integrating the know-how acquired during the preliminary experiments, was clearly needed.

4.2. Choice of a testbed

From the preliminary experiments it became clear that a better experimental testbed was needed for exploring further the various observed malfunctions of RDC protocols and for studying, after these malfunctions would have been eliminated, how RDC protocols could affect the performance of RPL networks. The main requirements for this testbed were listed as follows:

- Use unmodified, commercially available motes, preferably the Zolertia Z1 motes, as a lot of know-how about them had already been accumulated in the research team.
- Run an unmodified version of the Contiki operating system.
- Support multi-hop networks with at least 25 motes.

- Allow an accurate (+/- 10ms) measurement of individual packet latencies, measured between sender application and receiver application, on single or multi-hop radio links.
- Allow an accurate (+/- 5%) measurement of the average power used by each mote that would, in normal operation, obtain its power from local batteries.
- Can easily be deployed in an environment free of perturbing radio activity.

Using testbeds developed by other research groups and made available through the Internet seemed to be an attractive idea that could save time and money. Examples of the first widely advertised testbeds, appearing as early as 2005, were Motelab [72] and TWIST [73]. In the first one, popular in north-American universities, each mote was connected to a Stargate single board computer manufactured by Intel Research and running Linux. All the Stargate computers were interconnected via Ethernet with a system management computer, allowing to load and run software in each of the motes, to monitor the execution and even to debug the software. No provisions were made to monitor the power used by the motes and, due to the intimate coupling of the motes with the Stargate computers, changing the type of mote to experiment with was difficult.

The TWIST approach was more versatile and made possible by the availability of motes equipped with a USB interface for loading software and sending serial data to a monitoring computer. Through USB hubs many motes (up to several hundreds) could be connected to a central development and testing workstation and the motes could receive their power through the USB connection as well but power consumption could not be measured. Moreover, such measurement would not be very useful to predict lifetimes of batteries because the USB interface itself is quite power hungry. As USB 2.0 has provisions to switch on and off the power delivered via USB ports, motes could be powered from their own batteries or from a central power supply and the lifetime of these batteries verified experimentally, if the lifetimes were in the order of hours or days. Verifying lifetimes in the order of years, as is often required for wireless sensors inaccessible for battery replacement, would require impossibly long experiments or unreliable extrapolations. The Indriya testbed [74] deployed in 2009 in Singapore was quite similar to TWIST but used less costly USB hubs.

A more ambitious and more promising development was SensLAB [75], a set of large scale well instrumented wireless sensor networks implemented in several French universities (Grenoble, Strasbourg, Lille) and coordinated by INRIA, the French national research institute

for digital technologies. The SensLAB facilities offered networks with several hundreds of identical motes that could be used to run a wide variety of experiments, entirely controlled via the web. Even mobility was a possible research topic as motes mounted on programmable miniature trains were available. Performance indices such as PDR, message latency and power usage could easily be monitored and excellent software tools for organizing experiments and interpreting their results are widely available as open source software [76]. Unfortunately, for reaching such a versatility, specific motes had been developed for the project and several popular operating systems for WSNs had been tailored for the SensLAB motes. The hard- and software differences between products widely available for real-world applications and those that could be tested on SensLAB made the latter inadequate for testing in depth specific versions of operating systems and RDC algorithms.

Another large-scale testbed was created through the federation of various testbeds available in European universities and the design of an abstraction layer that allowed to use these resources regardless of their technical properties, creating a kind of virtual testbed that could encompass true physical testbeds with real motes, but also simulators and hardware emulators [77]. Of course, such a high level of abstraction was inappropriate for diagnosing low level protocols.
As a consequence, after several discussions with SensLAB users, it was decided to design and build a testbed inspired by the one used for the preliminary experiments but freed from its serious limitations. Several recent surveys of testbeds available all over the world [78]–[80] have confirmed, a posteriori, that this was the right decision as all surveyed testbeds are designed for testing applications and, to a lesser extent protocols above the MAC layer.

4.3. A testbed with dual motes

An engineering student, and later PhD researcher, Maite Bezunartea, played an important role in the design and the realization of this testbed. She presented it at the International Internet of Things Summit in Rome [81].

4.3.1. Overall architecture

A small (20 motes at present) but very versatile new testbed, that can be deployed anywhere indoors or outdoors, was designed and constructed featuring two WSNs. One runs the applications and protocols under study, the other observes the first one and transmits these observations to a sink mote in which they are recorded. Figure 33 shows a mote of the dual

network as deployed in a garden, far from sources of interferences commonly encountered in university laboratories. The *black* box contains a Z1 mote that belongs to the observed network. It has a built-in ceramic antenna, limiting its radio range to a few meters. The *white* box is a Z1 mote that belongs to the observer network. It uses an external antenna, allowing single hop communications with the sink mote(s). From now on, the observed network will be named the *black* network while the observer network will be called the *white* network. *Black* and *white* networks use different frequency bands, typically channel 26 for the *black* motes and channel 16 for the *white* ones. These frequencies were chosen to avoid occasional interferences with WiFi from smart phones and laptops.

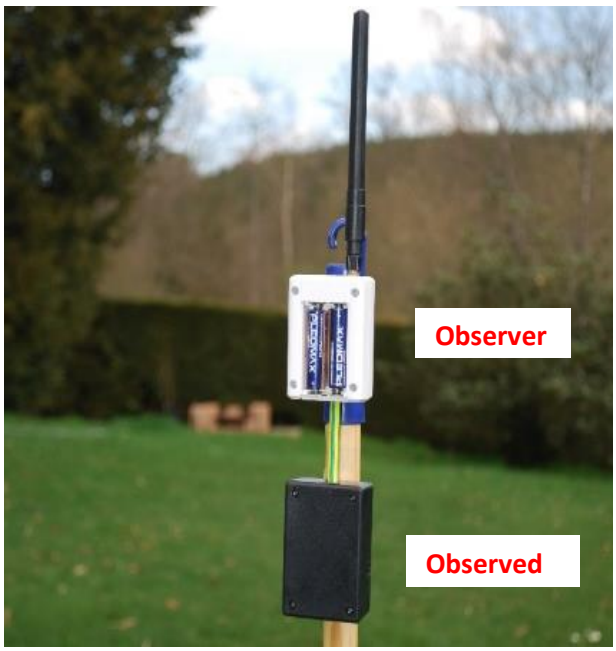

Figure 33: A dual mote from the testbed deployed in a garden

The motes of the *black* network run application programs that send short messages to each other (for instance, slightly modified versions of the Contiki UDPsender and CollectView programs can be used for testing RPL networks). These messages are uniquely identified by the address of the mote where they were created, and a local sequence number generated by the mote. Whenever a *black* application program sends or receives a packet, it communicates the sequence number to the *white* observer application running in the associated *white* mote. This application sends then to the *white* sink a packet containing the sequence number as well as the power used by the *black* mote since the last packet was transmitted or received. A Linux computer connected to the *white* sink stores all the received packets, together with a timestamp

that gives the moment the message arrives, according to the clock of the Linux computer with a 1ms resolution.

Black motes are identified by a number between 201 and 299, white motes by a number between 100 and 199. Associated black and white motes always have the same two last digits in their identifier. Dual mote 11 is composed of white mote 111 and black mote 211. Mote 100 is the white sink.

The recorded data, (supposing no messages are lost in the *white* network) allows a full inventory of the data packets transmitted and received by applications running on the *black* motes, from which it is easy to compute the packet delivery ratio, the per packet latency and the average power used in the initial sender and the final receiver.

4.3.2. The Link between the Black and White Motes

Transferring data between two motes can be done through the serial USB port or via some of the parallel GPIO pins available on the Z1 motes. As transferring data through the serial port requires approximately 80µs per character, which would disturb beyond reasonable limits the normal operation of the *black* network and affect significantly, because of the USB interface, its power requirements, the parallel pins were chosen. A Zolertia designer recommended to use GPIO pins 1.0, 1.6, 1.7, 2.3, 4.0, 4.2 and 4.3 as those are used by none of the built-in features of the Z1 motes. The 7 mentioned pins of the *black* and *white* motes were directly connected with each other.

Setting these 7 bits to 0 or 1 requires approximately 25 instructions in the *black* mote, which is a negligible perturbation for most application protocols. Six pins are used to transfer the six least significant bits of the sequence number of a packet sent by a *black* mote, allowing to reorder up to 63 out of order received packets. Pin 1.0 is used as trigger for the *white* mote. To avoid that the power measured for a black mote should be augmented by current flowing between the two motes via the GPIO pins, the pull-up and pull-down resistors available in the GPIO interface should be disabled when the white GPIO are configured. At the very moment the toggling of pin 1.0 by the *black* mote gets detected by the *white* mote, this last one records this moment by reading its local 32768Hz real-time clock and sends a report to the *white* sink.

4.3.3. Evaluating the Packet Latency

The latency for a packet travelling from mote A to mote B in the *black* network (possibly via other motes) is defined (see Figure 34) as the difference between the moment (t_{AB}) when the application in A transfers the packet to the lower communications layer and the moment (t_{BB}) when the application in B has received it. In the chapter about the preliminary experiments, the need of external observing devices to measure such latencies without perturbing the communicating process has been explained. In this new design, instead of using a data logger, the *white* motes are the observing devices. They are informed of any transmission or reception of a packet by an application running on a *black* mote, through the interconnected trigger pins. Toggling these pins causes the *white* mote to send via a single hop radio link a packet to the *white* sink that transmits it via a USB cable to the Linux computer for timestamping and recording.

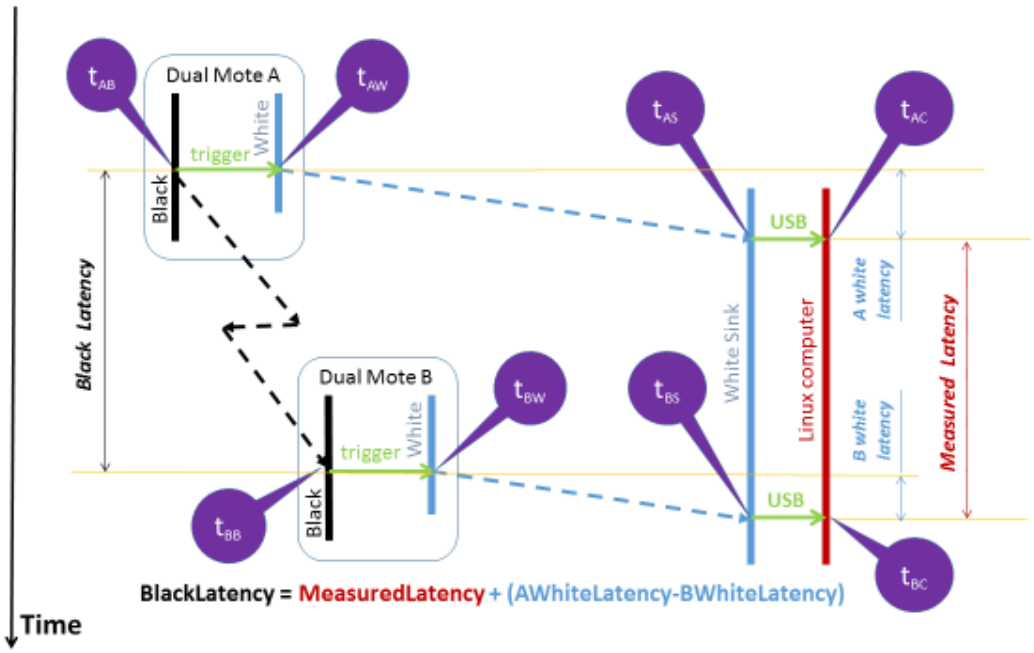

Figure 34: Timing of the latency measurements by means of the dual network

The relevant time stamps are:
tAB: the moment Black mote A sends a packet and communicates its sequence number to White A mote.
tAW: the moment White mote A is triggered by its associated Black mote..
tAS: the moment the White Sink has received the packet sent by White mote A
tAC: the moment the Linux computer has stored and time-stamped the packet coming from white mote A.
tBB: the moment Black mote B has received the packet sent by Black mote A and communicates its sequence number to White mote B.
the meaning of tBW,tBS and TBC is analogous to the meanings of tAW,tAS and TAC

The difference between the timestamps $t_{BC} - t_{AC}$ is a good estimation of the *black* packet latency, on condition that the difference between the delays ($t_{AC} - t_{AB}$) and (t_{BC} - t_{BB}) is negligible.

The delays caused by the triggering of the white mote by the black one ($t_{AW} - t_{AB}$) and (t_{BW} - t_{BB}) can be subdivided in two parts: first, the execution of the additional code in the black application to set the GPIO pins which is essentially the same in motes A and B and, therefore, doesn't need to be considered here as it will be cancelled in the subtraction $t_{BC} - t_{AC}$; the second part is the time needed by the white mote to notice the change of the state of its GPIO pins and generate the white packet reporting this event.
As the trigger pin causes an interrupt in the white mote and that the processor of the white mote is idle most of the time, the generation and initiating of the sending of the white packet will require only some tens of microseconds, resulting in a negligible difference between the delays in the A and B channels.

The difference between the delays caused by the transfer, via the USB interfaces, of the received packets between the white sink and the monitoring computer ($t_{AC} - t_{AS}$ and t_{BC} - t_{BS}) can also be neglected because they are proportional to the length of the packets, and all packets in the white network have the same length. The "speed" of the interrupt system of the monitoring Linux computer also guarantees a 1ms accuracy of the timestamps attached by the serialdump program to the records of the received packets.

The only important differences appear in the latency of the white messages ($t_{AW} - t_{AS}$ and t_{BW} - t_{BS}). The white radio links use no RDC protocol (nullrdc), because power consumption in the white network is not an issue, but they use the CSMA MAC protocol, with, up to two retransmissions of unacknowledged packets, in order to minimize white packet loss due to collisions, and, of course, such retransmissions have a big impact on packet latency. In the previous chapter, three different methods to measure the latency of packets over a single hop radio link have been introduced. The second method, based upon instrumentation of the radio driver, can be used to measure the latency of the white packets, as there is room in the payload for the required clock readings and no end to end verification of the integrity of the payload needs to be performed. For the reader's comfort, the explanations from the previous chapter, restricted to the requirements of the white links, will be repeated here.

The different steps, with their timing, of the transfer of a packet from the application layer of a white sender to the application layer of a white receiver, in absence of any RDC protocol, is shown in Figure 35.

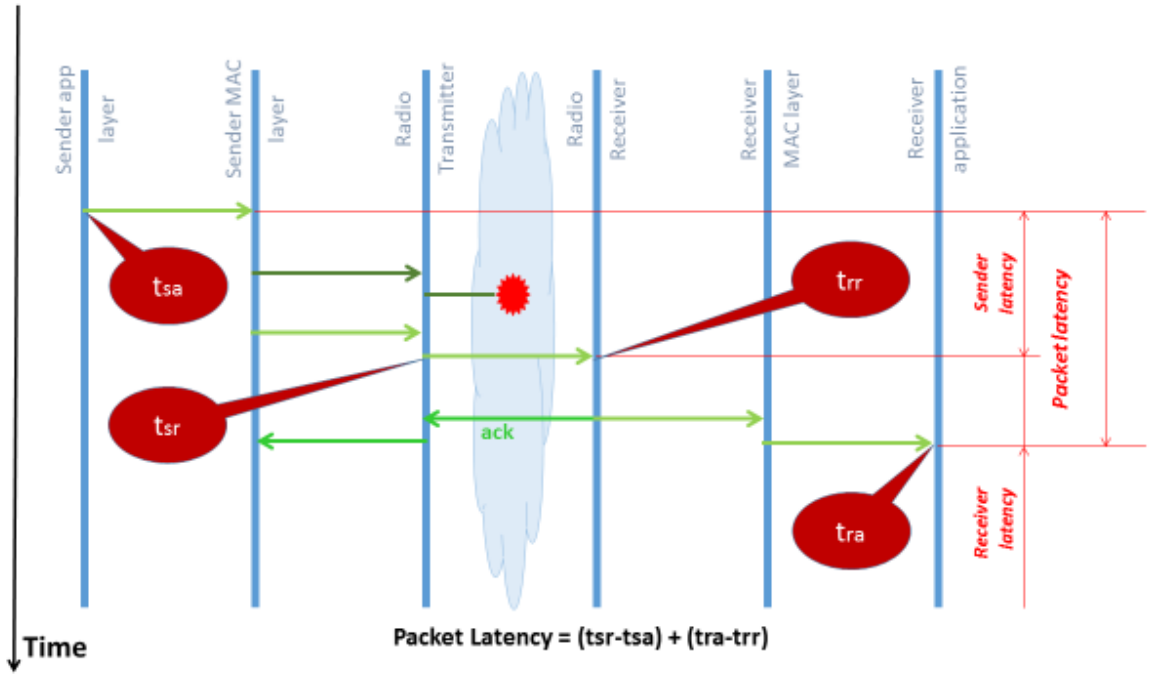

Figure 35: Timing of the transfer of a packet between sender and receiver, in absence of any RDC protocol

The red dot in the middle of the transmission cloud represents the loss of a packet, due to an external cause.
The relevant time stamps are:
tsa: the moment the application layer in the sender calls the send function.
tsr: the moment the radio of the sender transmits the Start Of Frame (SOF) byte.
trr: the moment the receiver radio recognizes the SOF byte
tra: the moment the receiver application gets the entire packet from the lower layers.

The moment called here t_{sa}, when the packet is generated in the application layer of the white sender, corresponds to the moments called t_{AW} or t_{BW} in Figure 34, while t_{ra}, when the packet is available in the application layer of the white sink, corresponds to t_{AS} or t_{BS}. The total white packet latency t_{ra} - t_{sa} is composed of three parts: the time the packet spends in the white sender mote before being transmitted with success, the propagation time between sender and receiver,

which is negligible because of the small distances, and the time the packet stays in the white sink mote. As is shown in Figure 35, the non-negligible components of the packet latency are called “sender latency” and “receiver latency”.

To measure the sender latency, the application layer in the sender mote, when triggered by the black mote, includes the reading of the local real-time clock at moment t_{sa} in the payload of the reporting packet before transferring the packet to the MAC layer, where it is copied in a FIFO buffer and send in the air after the absence of any other radio activity has been checked. At the moment the fifth byte of the packet (the so called “Start of Frame”) has been transmitted, the radio generates an interrupt, handled by the radio driver who, without stopping the radio, copies the reading of the real-time clock in the two last bytes of the FIFO buffer.

This way, the transmitted packet contains in its payload the readings of the local real-time clock at the moment it was created and at the moment it was send out. In the example illustrated in Figure 35, the packet collides with another radio signal and is lost. The MAC layer, in absence of an acknowledgement, retries the transmission of the contents of the sender FIFO, but, through the “Start of Frame Detected” interrupt, the last two bytes of the FIFO buffer are updated. Through this mechanism, the successfully received packet contains both the time the packet was generated in the application layer (t_{sa}) and the time it was successfully transmitted (t_{sr}), both derived from the local real-time clock of the sender, allowing to compute, a posteriori, the sender latency.

A similar scheme, involving two readings of the 32768Hz real time clock of the white sink, is used to calculate the “receiver latency” or the time elapsed between the moment the “Start of Frame” was detected and the moment the packet was delivered to the application layer of the white sink. This is less important because the receiver latency results almost entirely from the clocking-in of the successive bytes of the packet and as the packets transmitted in the white network have the same format, the “receiver latency” will be constant, unless the application software in the white sink was busy transmitting over the USB link the previous packet.
By these techniques, the link delays $t_{AS} - t_{AW}$ and $t_{BS} - t_{BW}$ can be computed for both the packets reporting transmission and reception of a black packet and the latency of that packet is given by:

$$t_{BB} - t_{AB} = (t_{BC} - t_{AC}) - (t_{BS} - t_{BW}) + (t_{AS} - t_{AW})$$

4.3.4. Measuring technique for Packet Delivery Ratio

As already discussed in section 3.3.2, six GPIOs are used for carrying the six least significant bits of the *black* packet sequence numbers. By matching the sequence numbers of transmitted and received packet reports received and stored by the Linux computer connected to the *white* sink it is easy to determine for each *black* end to end link the PDR. This however supposes that no messages are lost in the *white* network. Packet loss in the *white* network can be detected by numbering also the *white* packets and can be almost totally avoided by using a MAC protocol. An important cause of packet loss could however be the congestion of the *white* sink. A simple way to avoid this would consist in using more than one *white* network, operating at different frequencies. However, the need of this simple extension has not yet been observed when up to 24 dual motes transmitting one packet per second were used concurrently and less than 1/10000 *white* messages got lost.

4.3.5. Measuring technique for Power Usage

Instead of using, as was done in the preliminary experiments, a data logger for measuring the power used by the motes, the build-in analog to digital converter of the white motes was used for that purpose. The *black* motes are powered from the batteries of the *white* motes via a 1Ω series resistor. A 4700 µF capacitor in parallel with the power input of the black mote, ensures that no significant aliasing error will occur if the voltage across the resistor is sampled 128 times per second (at each tick of the real-time clock of the white motes) to record the current used by the *black* mote. Figure 36 shows how the *black* mote is powered and its current amplified by a differential instrumentation amplifier before being sampled at 128Hz and digitized by the built-in AtoD converter with sample and hold of the *white* mote.

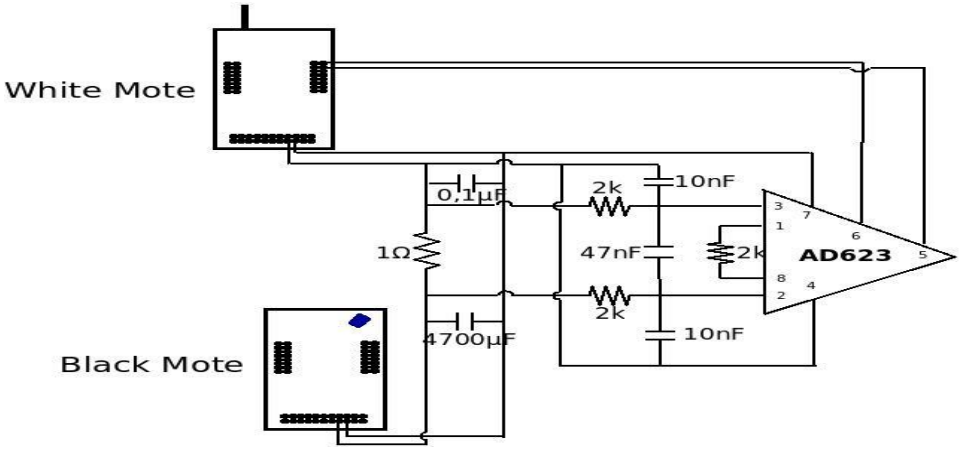

Figure 36: Measuring the power used by the black mote

The black mote is powered by the batteries of the white mote, Between the two, a 1Ω resistor is inserted to measure the current. The voltage over that resistor is amplified by an instrumentation amplifier before being sampled and digitized in the white mote. The 4700 µF capacitor avoids aliasing errors when the current is sampled at 128Hz.

4.3.6. The white packets

To summarize the operation of the (white) observation network, the contents of the packets that are sent by radio to the white sink each time a white mote is triggered through its GPIO port by the associated black mote is given here. The white sink collects the packets from all white motes and transfers them to a laptop. Each packet contains:

- the identity of the sending white mote,
- the sequence number assigned by the sending white mote to the packet,
- the sequence number obtained from the black mote,
- the power used by the latter since the launch of the previous packet,
- the number of 128Hz clock ticks elapsed since the launch of the previous packet,
- the reading t_1 of the white mote real time clock (RTIMER, 32768 ticks per second), at the moment the white mote was triggered by the black one,
- the reading t_2 of the same clock when the packet was sent by the white radio; the difference (t_2- t_1) is the time the packet has been delayed in the white MAC layer

4.4. Preprocessing of the collected data

Table 4 shows a sample of data collected for three seconds. It is intended to clarify by means of an example the observation by the white network of the events occurring in the black network and their reporting.

A	B				C	D	E	F	G	H	I	J	K	L
20160701	17	6	50	274	104	6817	62	2558	157714	295	9727	9777	55359	55409
20160701	17	6	50	479	102	6818	33	1250	57083	528	19392	19440	62096	62147
20160701	17	6	50	523	104	6819	63	2559	5839	6	17911	17956	63537	63587
20160701	17	6	50	572	111	6820	33	1246	12842	185	22098	22198	65132	65183
20160701	17	6	50	609	104	6821	0	2560	42475	47	20734	20779	824	874
20160701	17	6	50	652	103	6822	62	2484	64364	133	23596	23727	2208	2258
20160701	17	6	50	675	102	6823	34	1251	69598	111	25803	25846	2965	3016
20160701	17	6	50	822	111	6824	34	1247	24316	52	30307	30407	7805	7855
20160701	17	6	51	305	104	6825	1	2561	108117	393	43523	43572	23615	23665
20160701	17	6	51	402	103	6826	0	2485	78831	133	48199	48316	26794	26845
20160701	17	6	51	409	103	6827	1	2486	0	0	48450	48567	27045	27096
20160701	17	6	51	438	104	6828	2	2562	57558	73	47873	47922	27965	28015
20160701	17	6	51	525	103	6829	2	2487	9866	19	52221	52356	30834	30885
20160701	17	6	51	687	104	6830	3	2563	75372	113	56056	56100	36142	36193
20160701	17	6	51	776	103	6831	3	2488	23663	44	60447	60579	39057	39107
20160701	17	6	52	105	102	6832	35	1252	421295	826	7114	7162	49814	49865
20160701	17	6	52	197	111	6833	35	1248	139269	292	9798	9911	52841	52892
20160701	17	6	52	425	102	6834	36	1253	171465	183	17600	17643	60295	60345
20160701	17	6	52	571	104	6835	4	2564	456871	499	19457	19500	65076	65127
20160701	17	6	52	651	103	6836	4	2489	141146	156	23589	23706	2181	2232
20160701	17	6	52	764	111	6837	36	1249	67426	79	22084	28496	5890	5940

Table 4: Raw format of reporting packets obtained via the white network

A: date of the recording.
B: timestamp, with a millisecond resolution, assigned by the Linux computer.
C: number of the reporting dual mote (the white mote number is given here).
D: sequence number of the report as assigned by the white sink.
E: sequence number, modulo 64, of the packet send or received by the reporting black mote.
F: sequence number assigned by the reporting white mote.
G: power consumed by the reporting black mote since the previous report.
H: number of 128 Hz clock ticks since the previous report counted by the reporting white mote.
I: fast real time clock (32768Hz) of the reporting white mote when it was triggered by the associated black mote.
J: fast real time clock of the reporting white mote when the SFD character of the report was successfully launched.
K: fast real time clock of the white sink when the SFD character of the report was received.
L: fast real time clock of the white sink when the report was successfully transferred to the Linux computer.

The observed experiment consists in sending, at an average rhythm of one per second, black packets from mote 202 to mote 211, while mote 204 is sending, at the same average rhythm, perturbing packets to mote 203. The interval between packet creations in both senders is a random variable, uniformly distributed between 10ms and 1990ms.

Columns A and B give the date and hour the reporting packets were received by the computer connected to the white sink (between 17:06:50:274 and 17:06:52:764).

Column C gives the number of the white mote that generated the report. Packets are being received from white motes 102, 103, 104 and 111, respectively associated with black motes 202, 203, 204 and 211.
Column D contains the sequence number, given by the white sink, to each received report.

Column E contains the sequence number, modulo 64, of the black packet that triggered the white report. Observing the black sequence numbers of the packets send out by mote 204, one can clearly see sequence number 63 being followed by sequence numbers 0, 1 and 2. One can also observe that mote 203 receives packet 62 and then packets 0, 1 and 2, which is an indication that packet 63 from 204 was lost or seriously delayed.

Column F contains a sequence number assigned by each white mote to the reports it sends out. This makes possible to verify for each white mote the continuity of its sequence numbers and discover possible loss of white packets.

Column G gives for each white mote the power used by the associated black mote since the previous report was send. The duration of the interval between successive reports is given, in 128Hz clock ticks, in column H. It is worthwhile to observe that the power used is not at all proportional to the elapsed time. This is due to the storage of energy in the mote that was already mentioned in chapter 3.

Columns I and J are the sender's 32768Hz real time clock readings whose difference is the sender latency of the white packet. The receiver latency is obtained by subtracting the sink's clock readings from columns K and L. Of course, attention must be paid, when computing latencies, to the modulo 2^{16} format of all readings of real-time clocks in the motes.

4.4.1. Determining the time of reported black events

For analyzing the collected data, they are copied in EXCEL spreadsheets. Table 5 shows the data presented in Table 4 after their insertion in an EXCEL spreadsheet.

The names A, B, C, ... given to different columns have been kept but for clarity all columns not relevant for calculating the time of a black event according to the clock of the Linux computer have been hidden. Columns SL, RL and WL have been added. They give, for each report, the white sender latency, the white receiver latency and the sum of the two, the total white latency, that should be subtracted from the recorded timestamp to find the "true" time of the reported black event. Finally, the four columns, entitled "B corrected" give the corrected value of the timestamp recorded in the four B columns.

It is clear that, almost always, both the sender and the receiver latencies are small, in the order of a few milliseconds. However, the last line of the table shows that exceptions exist: the white sender latency for this report is 196ms, which indicates that it has been necessary to retransmit the report because the first transmission had failed.

For detecting and understanding collisions in experiments where one packet flow is perturbed by another (202 and 204 both sending in this example) it is useful to replace the original timestamps by the corrected ones and to sort the reports according to the corrected timestamps. Table 6 shows the effect of such reordering on the 8 last reports shown in Table 5. The alternation of send and receive reports is clearly visible while, at the end of Table 4, it was harder to see due to the large white latency.

B				C	E	I	J	K	L	S.L.	R.L.	W.L.	B corrected			
H	m	s	ms	sndr	seq#	snd.lat		rcv.lat		ms	ms	ms	h	m	s	Ms
17	6	50	274	104	62	9727	9777	55359	55409	1.5	1.5	3	17	6	50	271
17	6	50	479	102	33	19392	19440	62096	62147	1.5	1.6	3	17	6	50	476
17	6	50	523	104	63	17911	17956	63537	63587	1.4	1.5	3	17	6	50	520
17	6	50	572	111	33	22098	22198	65132	65183	3.1	1.6	5	17	6	50	567
17	6	50	609	104	0	20734	20779	824	874	1.4	1.5	3	17	6	50	606
17	6	50	652	103	62	23596	23727	2208	2258	4.0	1.5	6	17	6	50	646
17	6	50	675	102	34	25803	25846	2965	3016	1.3	1.6	3	17	6	50	672
17	6	50	822	111	34	30307	30407	7805	7855	3.1	1.5	5	17	6	50	817
17	6	51	305	104	1	43523	43572	23615	23665	1.5	1.5	3	17	6	51	302
17	6	51	402	103	0	48199	48316	26794	26845	3.6	1.6	5	17	6	51	397
17	6	51	409	103	1	48450	48567	27045	27096	3.6	1.6	5	17	6	51	404
17	6	51	438	104	2	47873	47922	27965	28015	1.5	1.5	3	17	6	51	435
17	6	51	525	103	2	52221	52356	30834	30885	4.1	1.6	6	17	6	51	519
17	6	51	687	104	3	56056	56100	36142	36193	1.3	1.6	3	17	6	51	684
17	6	51	776	103	3	60447	60579	39057	39107	4.0	1.5	6	17	6	51	770
17	6	52	105	102	35	7114	7162	49814	49865	1.5	1.6	3	17	6	52	102
17	6	52	197	111	35	9798	9911	52841	52892	3.4	1.6	5	17	6	52	192
17	6	52	425	102	36	17600	17643	60295	60345	1.3	1.5	3	17	6	52	422
17	6	52	571	104	4	19457	19500	65076	65127	1.3	1.6	3	17	6	52	568
17	6	52	651	103	4	23589	23706	2181	2232	3.6	1.6	5	17	6	52	646
17	6	52	764	111	36	22084	28496	5890	5940	195.7	1.5	197	17	6	52	567

Table 5: Example of timestamp corrections by means of subtraction of white latencies

B				C	E	I	J	K	L	S.L.	R.L.	W.L.	B corrected			
h	M	s	ms	sndr	seq#	snd.lat		rcv.lat		ms	ms	ms	h	m	s	Ms
17	6	51	687	104	3	56056	56100	36142	36193	1.3	1.6	3	17	6	51	684
17	6	51	776	103	3	60447	60579	39057	39107	4.0	1.5	6	17	6	51	770
17	6	52	105	102	35	7114	7162	49814	49865	1.5	1.6	3	17	6	52	102
17	6	52	197	111	35	9798	9911	52841	52892	3.4	1.6	5	17	6	52	192
17	6	52	425	102	36	17600	17643	60295	60345	1.3	1.5	3	17	6	52	422
17	6	52	764	111	36	22084	28496	5890	5940	195.7	1.5	197	17	6	52	567
17	6	52	571	104	4	19457	19500	65076	65127	1.3	1.6	3	17	6	52	568
17	6	52	651	103	4	23589	23706	2181	2232	3.6	1.6	5	17	6	52	646

Table 6: The last 8 lines of table 5, reordered in function of the corrected timestamps

4.4.2. Determining black packet delivery rates and latencies

By sorting the reports according to the number of the sending mote, one can obtain separate ordered lists of the packets send and received on a specified link. By matching the black sequence numbers from sender and receiver one can see the packets that got lost and determine the latency of those that were correctly received. Table 7, again derived from Table 4, shows the relevant columns from the reports generated in white motes 104 and 103, describing the traffic from black mote 204 to 203.

One report from 104, carrying sequence number 63 could not be matched with a report from the receiver side, meaning that packet 63 has been lost. The large latencies of the packets surrounding the lost one suggests that the two-radio links where seriously perturbing each other at that time and causing multiple MAC layer retransmissions. In this example, the PDR is:

PDR = received packets / send packets = 6/7 = 86%

This is statistically meaningless because measured over far too few packets but given as an explanatory example.

C	E	B corrected					C	E	B corrected					Packet lat.
sndr	seq#	h	m	s	ms		sndr	seq#	h	m	s	ms		ms
104	62	17	6	50	271		103	62	17	6	50	646		375
104	63	17	6	50	520									
104	0	17	6	50	606		103	0	17	6	51	397		791
104	1	17	6	51	302		103	1	17	6	51	404		102
104	2	17	6	51	435		103	2	17	6	51	519		84
104	3	17	6	51	684		103	3	17	6	51	770		86
104	4	17	6	52	568		103	4	17	6	52	646		78

Table 7: Spreadsheet used for computing latencies and PDR over a black link

4.4.3. Determining the power consumption

To compute the power used by a black mote, the supply voltage is multiplied by the current. The supply voltage is provided by two lithium/iron disulfide AA batteries. These batteries were chosen because, among AA 1.5V batteries, they have a constant voltage of 3V (+/- 0.1V) over almost their entire lifetime [82].

The current is converted in a voltage by means of a 1Ω series resistor and an instrumentation differential amplifier with gain 50. This amplified voltage is sampled at 128Hz and converted in 12-bit numbers included in the reports transmitted by the associated white motes. Calibration of the current measuring chain shows that one digital unit corresponds to a current of 11.5µA, or, after multiplication by 3V, 34.5µW. Considering the (lack of) accuracy of the different components in the power measurement chain, the true power consumption is estimated to be within a +/-5% range of the measured values. Measurements with multimeters on individual motes confirmed these estimations.

C	E	F	G	H	interval	B corrected				Energy used	
sndr	seq#	#	power	int	s	h	m	s	ms	mJ	
104	62	2558	1761140	67	0.523	17	6	50	271	907	
104	63	2559	60336	31	0.242	17	6	50	520	67	
104	0	2560	467225	11	0.086	17	6	50	606	1465	
104	1	2561	1202802	89	0.695	17	6	51	302	466	
104	2	2562	978486	17	0.133	17	6	51	435	1986	
104	3	2563	778844	31	0.242	17	6	51	684	867	
104	4	2564	4693311	113	0.883	17	6	52	568	1433	
Total energy used =										7191	mJ
Total duration =				359							128Hz ticks
Average power used =										20.0	mW

Table 8: Estimation of the average power used by mote 204

Table 8 gives the data necessary to compute the power consumption of mote 204 in the example from Table 4. One column has been added to show the energy (expressed in mJ) used by the black mote since the previous report, obtained by multiplying the contents of column G by 34.5 µW and by dividing by the number of times this has been summed (column H). The sum of all values in the columns H and "Energy used" gives respectively the total duration of the experiment and the total amount of energy used. The quotient of the two gives the average power used by mote 204 during that experiment.

Table 9, similar to Table 8, but for mote 203, a receiver, shows a particularity of the measuring technique when the RDC protocol is ContikiMac. The power reported in white report 2560 is 0.

C	E	F	G	H	interval	B corrected				Energy used	
sndr	seq#	#	power	int	s	h	m	s	ms	mJ	
103	62	2558	64364	133	1.039	17	6	50	646	17	
103	63	2559	78831	133	1.039	17	6	51	397	20	
103	0	2560	0	0	0.000	17	6	51	404		
103	1	2561	9866	19	0.148	17	6	51	519	18	
103	2	2562	23663	44	0.344	17	6	51	770	19	
103	3	2563	141146	156	1.219	17	6	52	646	31	
Total energy used =										105	mJ
Total duration =				485							128Hz ticks
Average power used =										0.2	mW

Table 9: estimation of the average power used by mote 203

This is due to the very short interval (7ms, less than the sampling interval of the ADC converter) between the reception of black packets 63 and 0. Indeed, in ContikiMac, when a packet has been received, the receiver checks whether no other packets are coming in. When successful transmission of a packet has been delayed due to radio traffic, the next packet can be waiting in the transmitter buffer and two packets can follow each other. Table 9 shows also that, with a good working ContikiMac, a receiving mote uses much less energy than a sending one.

4.5. Conclusions

In this chapter the design of a simple versatile but low-cost testbed has been presented and the techniques to analyze the measurements by means of excel spreadsheets has been explained. This testbed permitted to compare systematically the performances of the different RDC protocols implemented in Contiki and can, later, be used to evaluate the performances (PDR, packet latency and power usage) of RPL networks using these, or other, RDC protocols.

Chapter 5: RDC malfunctions

5.1. Introduction

By means of the testbed described in chapter 4, a systematic comparison of the performances of three Radio Duty Cycling (RDC) protocols implemented in Contiki was made. During these measurements, several undocumented malfunctions of the protocols were observed. These malfunctions were investigated, and it appeared that, except for one that that was caused by a simple software bug, they were all consequences of two properties of real motes that were overlooked by the designers of the RDC protocols and the Cooja simulator. In this chapter, the observed malfunctions of each of the RDC protocols will first be described, their causes analyzed, and work-arounds proposed.

Some of the observations presented in this chapter were investigated with the collaboration of PhD candidate Maite Bezunartea, particularly those presented in paragraph 5.2.1.4-The case of LPP and in paragraph 5.2.3-ContikiMAC receives packets with RSSI below the CCA threshold.

5.2. Observed malfunctions

5.2.1. Black-outs in ContikiMac and LPP

5.2.1.1.Symptoms

In experiments consisting in sending one packet per second over a link using ContikiMac as RDC protocol, periodically a few consecutive packets were lost. The period between losses and the number of packets lost was stable but did change considerably when other, apparently identical, motes were used. With motes 202 and 211, commonly used for the performance measurements described in the following chapter, the interval between black outs was approximately 60s and some 7 to 10 packets got lost. A similar malfunction occurred with LPP as RDC protocol, but with a period of some 40 minutes and a loss of some 25 packets.

5.2.1.2.Investigation

These black-outs were first investigated in a ContikiMac RDC environment mainly because incidents were more frequent than those occurring with LPP. The traffic between sender and receiver was observed by means of a Texas Instruments cc2531 radio sniffer which showed clearly that the black-outs were simply caused by the receiver ignoring incoming packets. The CSMA MAC layer of the sender made the normal three attempts to transmit the packets, but to no avail.

Observation of the radio traffic by means of the sniffer revealed however a serious difference between the timing specifications of the ContikiMac protocol [47] and its actual operation. The specifications, explained in detail in chapter 2, are repeated in Figure 37, while the actual timings can be deduced from Figure 38 obtained from the sniffer. The time interval measured between transmission of repeated data packets is in the order of 3 ms, but the time for transmitting the entire packet (61 bytes+4 bytes for the preamble) is only 65/31250 = 2.08ms, the difference between the 3ms and the 2.08ms is the interval between successive retransmissions, but it exceeds almost by 600 µs the interval t_i specified in Figure 37.

As t_i must be smaller than t_c, the interval between the two CCA tests, to prevent packets staying undetected when the two CCA tests fit in the interval t_i between repetitions of the same packet, this timing anomaly became a good candidate to explain the observed blackouts.

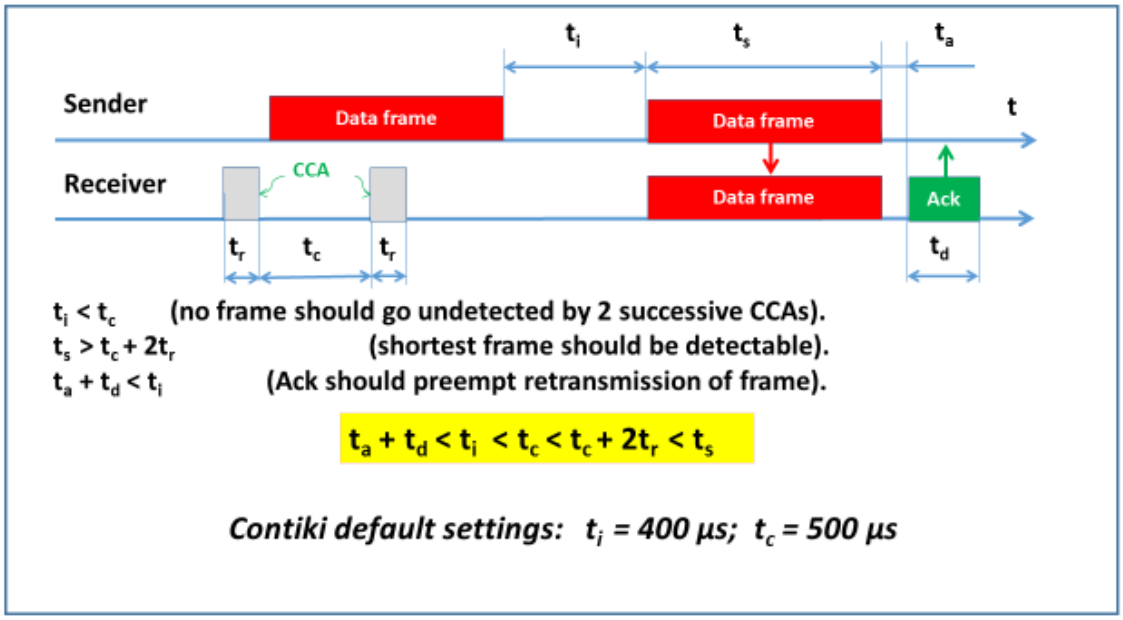

Figure 37: ContikiMac timing specifications *(*Repeated from chapter 2*)*

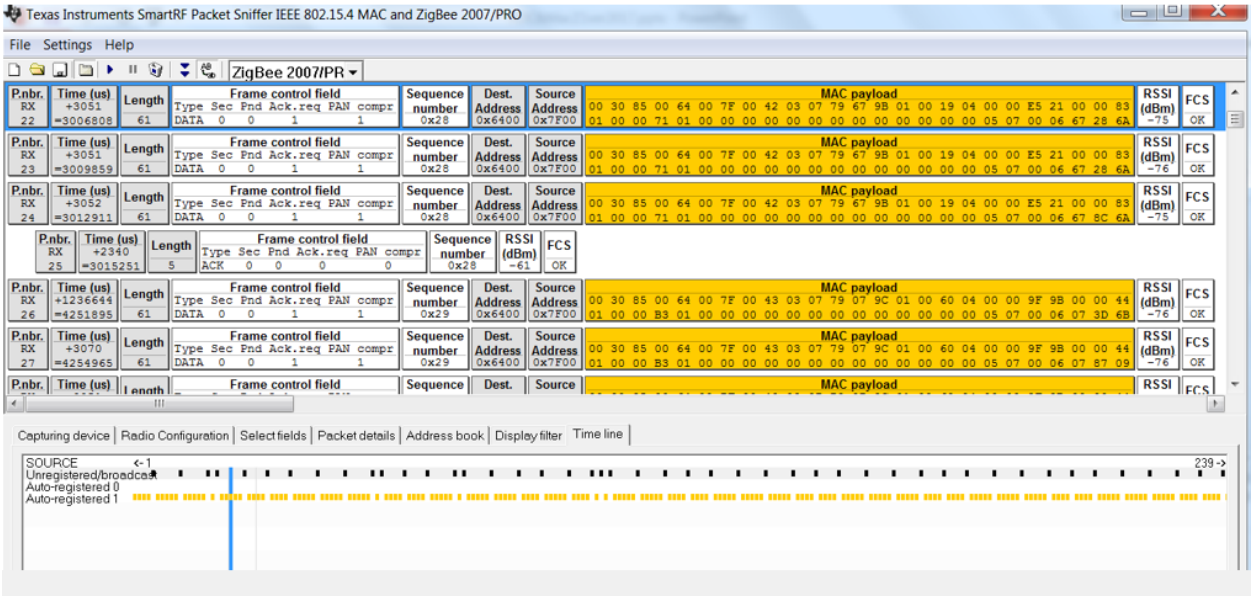

Figure 38: Recording by the TI 2531 sniffer of 6 packets exchanged between two motes running the ContikiMac RDC protocol

The time elapsed between successive packets can be read in the "Time" column while the length of the packet is available in the "Length" column. The time line at the bottom of the figure shows in yellow the successive packet transmissions and in black the corresponding acknowledgments. This time line is graduated in events, not in elapsed time

The periodicity of the blackouts and the spontaneous recovery can be explained by considering the small frequency differences between the real-time clocks of the different motes: due to such differences the relative timing of packet transmission and wake-up of the receiver with the associated double CCA test will shift in time. Figure 39 shows the effect of such shift on the detection of incoming packets by the double CCA when t_i is larger than t_c.

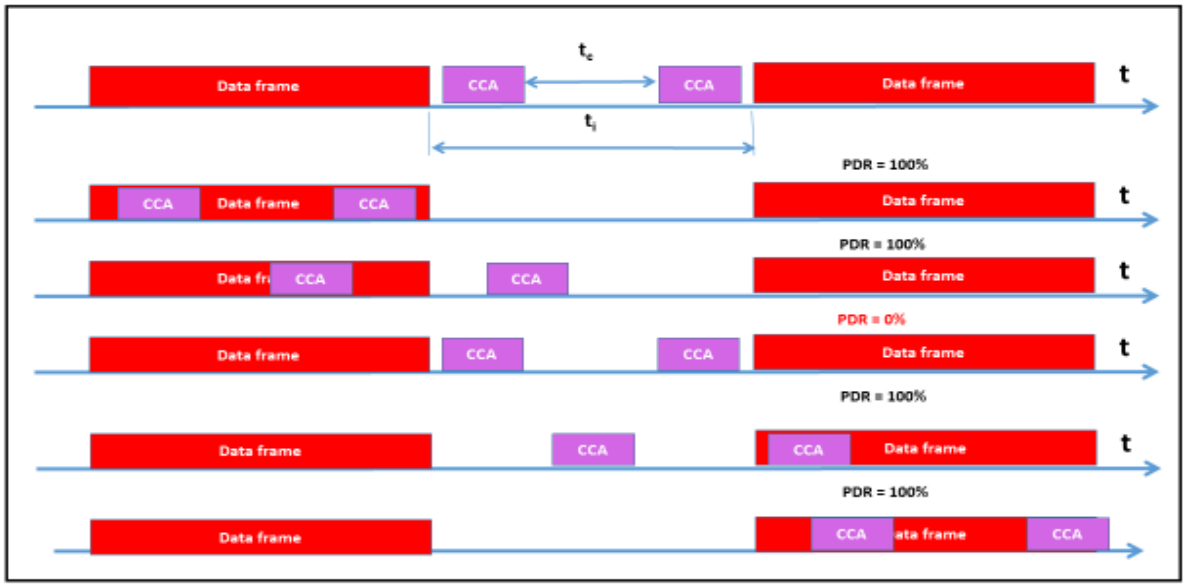

Figure 39: Effect of an excessive ti and a small difference between sender and receiver clocks on the PDR

This results in the CCAs moving slowly forward or backward between the repeated packets.

If δf is the relative difference of the frequencies of both clocks and $t_s + t_i$ the total duration of a packet transmission, a blackout will occur every $(t_s + t_i)/\delta f$ seconds because this is the time between occurrences of an identical relative position of the data frame and the double CCA. The blackout will last $(t_i - t_c)/\delta f$ seconds because this is the time between the one CCA no longer overlapping with a data frame and the other CCA overlapping with the previous or the next data frame (the duration of the CCA test is not taken into consideration for this rough estimation because the minimum overlap between a CCA test and a data frame needed for detection depends on the RSSI of the packet).

These estimations have been checked for the motes used in this investigation and, as shown in Table 10, results confirm the order of magnitude of the observations.

Sender = 127	Receiver = 100	$\delta f = 5.10^{-5}$
$t_s + t_i = 3$ ms	$t_i = 1$ ms	$t_c = 0.5$ ms
	Computed	*Observed*
Blackout period	60s	Between 55s and 70s
Blackout duration	10s	Between 7s and 10s

Table 10: Blackout periodicity with ContikiMac

The next, and most important issue to investigate about the ContikiMac blackouts is why t_i has such a high value, clearly incompatible with the ContikiMac specifications. In the source code of ContikiMac, one can find the definition:

/ INTER_PACKET_INTERVAL is the interval between two successive packet transmissions */*

…

#define INTER_PACKET_INTERVAL RTIMER_ARCH_SECOND / 2500

This definition, which seems to implement the ContikiMac specifications, is misleading, because a careful analysis of the subsequent code shows that the true t_i is much larger than INTER_PACKET_INTERVAL.

Figure 40 gives a detailed timing diagram of what happens between two retransmissions of a packet.

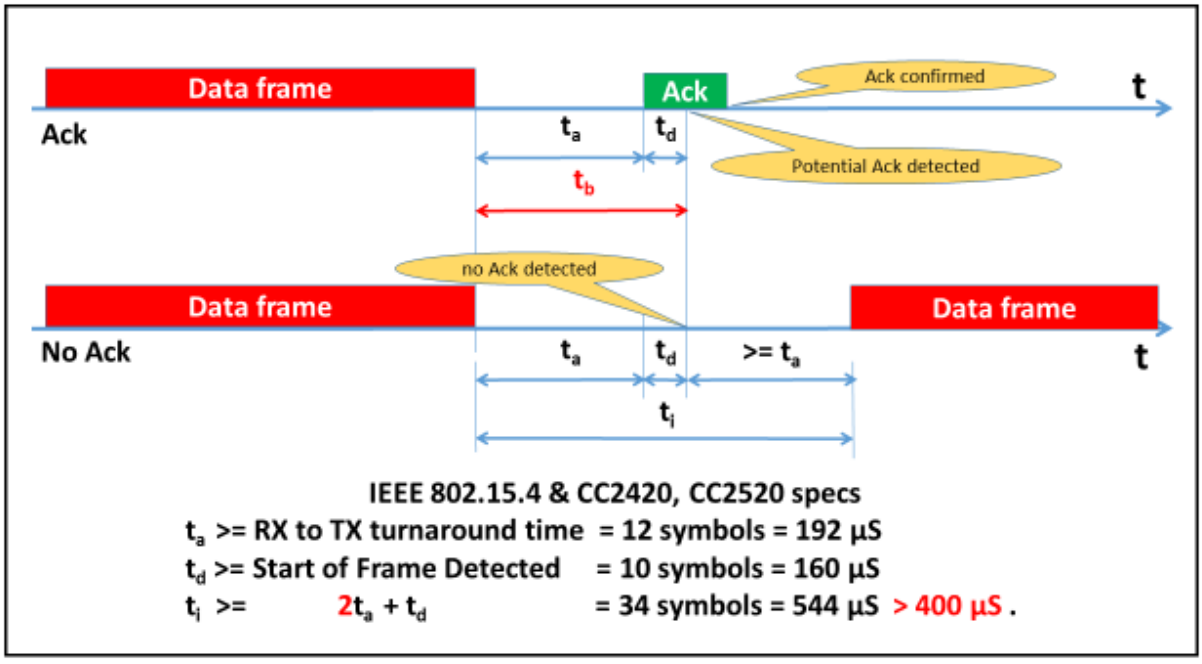

Figure 40: Timing of the events between two retransmissions of a packet by the sender running ContikiMac

The minimum values for t_a and t_d are specified in the IEEE 802.15.4 standard.

When the transmission of a data frame is complete, the radio in the sender has to change its state from "sending" to "receiving" and the radio in the receiver has to perform the opposite state change to send the ACK. According to the IEEE 802.15.4 standard and the data sheet of the cc2420 radio, used for these experiments, both state changes require a time equivalent to the transmission of 12 symbols (a sequence of 4 bits) or 192µs. This time interval is called t_a in Figure 40. When using hardware generated ACKs, as is the case in all the reported experiments, the ACK is send, if the packet was correctly received, t_a after the end of the transmission. The possible presence of an ACK is detected by means of an OR function of three status bits of the radio receiver:

```
if(!is_broadcast && (NETSTACK_RADIO.receiving_packet()||
        NETSTACK_RADIO.pending_packet()  ||
        NETSTACK_RADIO.channel_clear()== 0 )) {
```

Such a combination of three status bits is used to maximize the chances of detecting the ACK in presence of jitter in the timing of the ACK transmission. Testing these bits however doesn't make sense until, at least the 10 first symbols of the ACK have been received.

This delay is called t_d and is at least 160 µs. The INTER_PACKET_INTERVAL defined in ContikiMac and shown in Figure 40 as t_b is, in fact, the sum of t_a and t_d.

If no potential ACK has been detected after a delay t_b, the packet needs to be retransmitted, but, for that, the state of the sender radio has to be switched back to the "sending" state, which requires again a delay t_a.

From the diagram shown in Figure 40, it is clear that the interval between retransmitted packets t_i cannot be less than $2*t_a + t_d$ which is, at least 544 µs, or 144 more than the ContikiMac specification.

This major discrepancy between specifications and implementation is a consequence of the progress in semiconductor technology: when ContikiMAC was initially designed many motes used separate transmitter and receiver chips, so that no delay to switch between send and receive states was required. When combined sender/receiver chips, standardized in IEEE 802.15.4 became available, the ContikiMAC timing specifications should have been revised, but it was not done.

If a potential ACK has been detected after the delay t_b, the nature of the ACK needs to be confirmed, which is done by checking the length of the received packet after an additional delay defined by:

/* AFTER_ACK_DETECTECT_WAIT_TIME is the time to wait after a potential ACK packet has been detected until we can read it out from the radio. */

…

#define AFTER_ACK_DETECTECT_WAIT_TIME RTIMER_ARCH_SECOND / 1500

If, after these additional 666 µs, it appears that the detected packet was not an ACK, t_i will be augmented by these 666µs. None of the remedies proposed in the following paragraph is adequate to make sure that the subsequent CCA test will detect the unacknowledged data frame, but it is unlikely that such incidents will occur so frequently that they would cause noticeable malfunctions.

5.2.1.3.Remedies

The most obvious remedy to ensure that t_i is smaller than t_c consists in increasing sufficiently t_c. Unfortunately, this would raise other issues because small packets could then stay undetected between two CCAs.

By carefully tuning all the delays at the cost of safety margins for jitter and by using a non-standard feature of the cc2420 radio to reduce from 192 to 128µs the time required to switch from the receiving to the sending state, it was possible to bring t_i down to 490µs and this appeared to be sufficient to prevent periodic blackouts. This remedy however was considered unreliable and certainly not applicable in other IEEE802.15.4 based systems, because of the cc2420 specific reduction of the state switching delay.

A third remedy was explored and appeared to solve the problem without creating new issues. It consists in using three successive CCAs instead of two. This replaces the condition $t_i < t_c$ by $t_i < 2*t_c$, which is easy to satisfy within the boundaries of IEEE802.15.4 without sacrificing safety margins or undue restrictions on packet lengths. The performance studies reported in chapter 6 were mostly done with 3 CCAs, only a few were done with the minimized t_i version for comparison purposes.

For clarity purposes of the Contiki code, it was also proposed that the time defined by INTER_PACKET_INTERVAL should be renamed as BEFORE_ACK_DETECT_WAIT_TIME

5.2.1.4.The case of LPP

Once the phase shifts between algorithms in communicating motes were clearly understood, the cause of the observed blackouts in LPP became clear: in LPP a mote announces that it is awake by broadcasting a short message. If two motes send their wake-up announcements simultaneously they will collide, and no one will be aware that the motes are awake, causing a communications blackout for these motes.

At a wake-up rate of 8Hz and a relative frequency error δf of 5.10^{-5} such a blackout will occur every 2500s or approximately every 42 minutes and last approximately 28s as the duration of a wake-up announcement is approximatively 700 µs. Delaying the wake-up announcement by a randomly generated delay of 0,1 or 2 times the duration of the wake-up announcement was considered, but not tested in practice by lack of time and a relative lower importance of LPP in the literature about asynchronous RDC protocols.

5.2.2. Periodic duplication of broadcasted packets with the CXMac RDC protocol

5.2.2.1.Symptoms

In experiments consisting in sending one broadcast packet per second over a link using CXMac as RDC protocol, periodically two copies of the same packet were received. The period between duplications was stable but did change considerably when other, apparently identical, motes were used. With motes 202 and 211, commonly used for the performance measurements described in the following chapter, the interval between duplications was approximately 40 minutes. Despite the fact that broadcast messages are handled in an identical way in ContikiMac and in CXMac, no packet duplications were ever observed with ContikiMac.

5.2.2.2.Investigation

In both ContikiMac and CXMac broadcasts are implemented by sending repeatedly the broadcast packet during a full CCI, augmented by a small safety margin to make sure that all motes, whatever the moment they wake up, will detect the broadcast. Measured by means of the sniffer, with a CCI of 125ms, broadcasted packets were transmitted repeatedly during 139ms.

If, by chance, a receiver catches the first packet of a broadcast, it will also catch the last packet of the same broadcast when it wakes again up, one CCI later. Due to the small difference

between the clock frequencies of sender and receiver the situation where the receiver can catch the first and the last packet of a same broadcast will occur periodically, with a period equal to CCI/δf. With a δf of 5.10^{-5} and a wake-up frequency of 8Hz, duplication of broadcasted packet should occur every 40 minutes, which is indeed what was observed.

The difference in behavior of CXMac and ContikiMac was investigated next as both protocols handle broadcast the same way but only CXMac delivers sometimes duplicated packets. The explanation for this was simple: the ContikiMac software contains a filter that eliminates packets that have the same sequence number as a recently received packet:

```
/* Check for duplicate packet by comparing the sequence number
   of the incoming packet with the last few ones we saw. */
{
  int i;
  for(i = 0; i < MAX_SEQNOS; ++i) {
    if(packetbuf_attr(PACKETBUF_ATTR_PACKET_ID)==
received_seqnos[i].seqno &&
       rimeaddr_cmp(packetbuf_addr(PACKETBUF_ADDR_SENDER),
                    &received_seqnos[i].sender)) {
      /* Drop the packet. */
      /* printf("Drop duplicate ContikiMAC layer packet\n");*/
      return;
    }
  }
  for(i = MAX_SEQNOS - 1; i > 0; --i) {
    memcpy(&received_seqnos[i], &received_seqnos[i - 1],
           sizeof(struct seqno));
  }
  received_seqnos[0].seqno = packetbuf_attr(PACKETBUF_ATTR_PACKET_ID);
  rimeaddr_copy(&received_seqnos[0].sender,
                packetbuf_addr(PACKETBUF_ADDR_SENDER));
}
```

Temporarily bypassing that filter caused ContikiMac to generate duplicated packets just as CXMac did.

5.2.2.3.Remedies

The issue of duplicated broadcasted packet was entirely solved by including in CXMac the same anti-duplicates filter as exists in ContikiMac.

5.2.3. ContikiMac receives packets with a RSSI below the CCA threshold

5.2.3.1.Symptoms

According to the specifications of ContikiMac, incoming packets with a RSSI below the CCA threshold should not wake up the receiver and therefor remain undetected. This was verified experimentally in the test site in the Ardennes, but not when experiments were done in the university laboratories. Figure 41 shows the measured PDR as a function of the RSSI of the incoming packets both in Brussels and in the Ardennes.

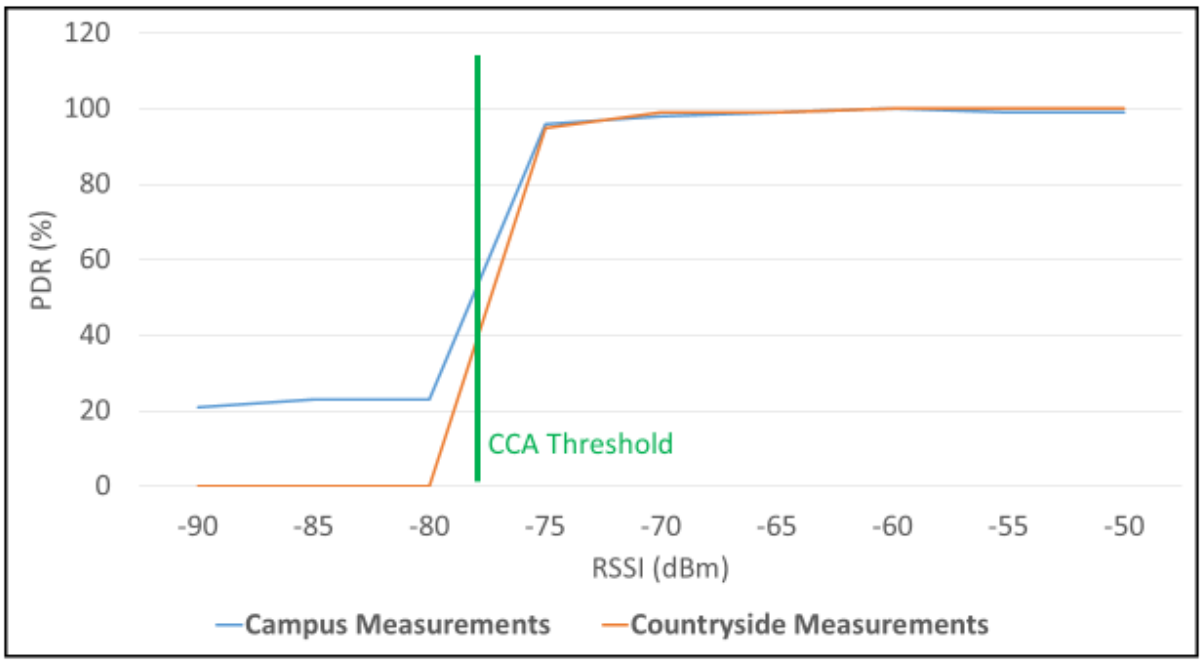

Figure 41: ContikiMac PDR as a function of the RSSI of incoming packets

5.2.3.2.Investigation

The default value of the CCA threshold is -77dBm but the cc2420 receiver can receive correctly packets with a RSSI as low as -91dBm. In ContikiMac, the receiver has to be woken-up before it can receive packets. Normally this is done by the incoming packet, which should have an RSSI well above the CCA threshold but, of course, it is possible that unrelated powerful radio traffic wakes up the receiver which, shortly after the wake-up, is capable of receiving any

packet. This was verified experimentally by reducing the power level of the tested link for an RSSI of -85dBm and leaving the perturbing link at full power. The observed PDR in the countryside was 0% when the perturbing link was disabled and fluctuated between 20 and 30% when the perturbing link was active.

One can wonder if receiving correctly packets with a RSSI below the CCA threshold is an issue but, indeed, it could be when setting up a RPL DODAG because weak DIO messages from far away motes could cause these motes to be erroneously considered as parents of a mote that was woken up by nearby traffic.

5.2.3.3. Remedies

By including in ContikiMac a filter that eliminates all packets received with a RSSI lower than the CCA threshold, the randomness of the radio range caused by the difference in sensivity of the receiver and the CCA threshold can be eliminated, which, accordingly to qualitative tests with RPL improves the stability and the convergence rate of the DODAG building. The proposed filter to be included in ContikiMac is:

```
/* MPU: Check for packet received with rssi below the cca threshold */
  int8_t rssi_val = ( cc2420_last_rssi ) - 45;
  int8_t cca_threshold = ( cc2420_get_cca_threshold());
  if(rssi_val < cca_threshold) {
        /* Drop the packet. */
    return;
  }
```

5.2.4. The encounter optimization algorithm is suboptimal in ContikiMac

5.2.4.1.Symptoms

All RDC protocols included in Contiki have an optional built-in mechanism to synchronize packet transmission with the wake-ups of their destination: by recording for recently used destinations the times ACKs are received and by knowing the wake-up frequency, the transmitter knows when the receiver at a specific destination will be awake and it can then schedule packet sending so that the packets reach their destination precisely when the destination is awake. Such encounter synchronization is essential for saving power by means of radio duty cycling.

As the ACK is not a very accurate (see next paragraph) indicator of the wake-up time of the destination, ContikiMac starts transmitting repeatedly packets before the computed wake-up of the destination. This safety margin is called “GUARD_TIME” and is defined by the following statement:

/* GUARD_TIME is the time before the expected phase of a neighbor that a transmitter should begin transmitting packets. */
*#define GUARD_TIME 10 * CHECK_TIME + CHECK_TIME_TX*

The constants CHECK_TIME and CHECK_TIME_TX are specific for the radio hardware but the somewhat arbitrary factor 10 is critical for the proper operation of ContikiMac: a too large value results in multiple repetitions of the packet before it is acknowledged and a waste of power in the transmitter while a too small value results in frequent packet losses and renewed initializations of the encounter optimization algorithm.

5.2.4.2.Investigation

Mathieu Michel et al. [52] have extensively studied the relation between wake-up and ACK timings in the original version of Contiki, with two CCAs. As shown in Figure 42, there is an uncertainty equal to the duration of one entire packet plus the time between the two successive CCAs about the interval between wake-up and ACK. They established a probabilistic model of the delay and proposed to replace hardware ACKs by software, user defined, ACKs containing the measured value of the delay between wake-up and ACK. This would allow the sender to accurately determine the true wake-up time and schedule its transmissions accordingly but would increase the overhead of the various radio drivers.

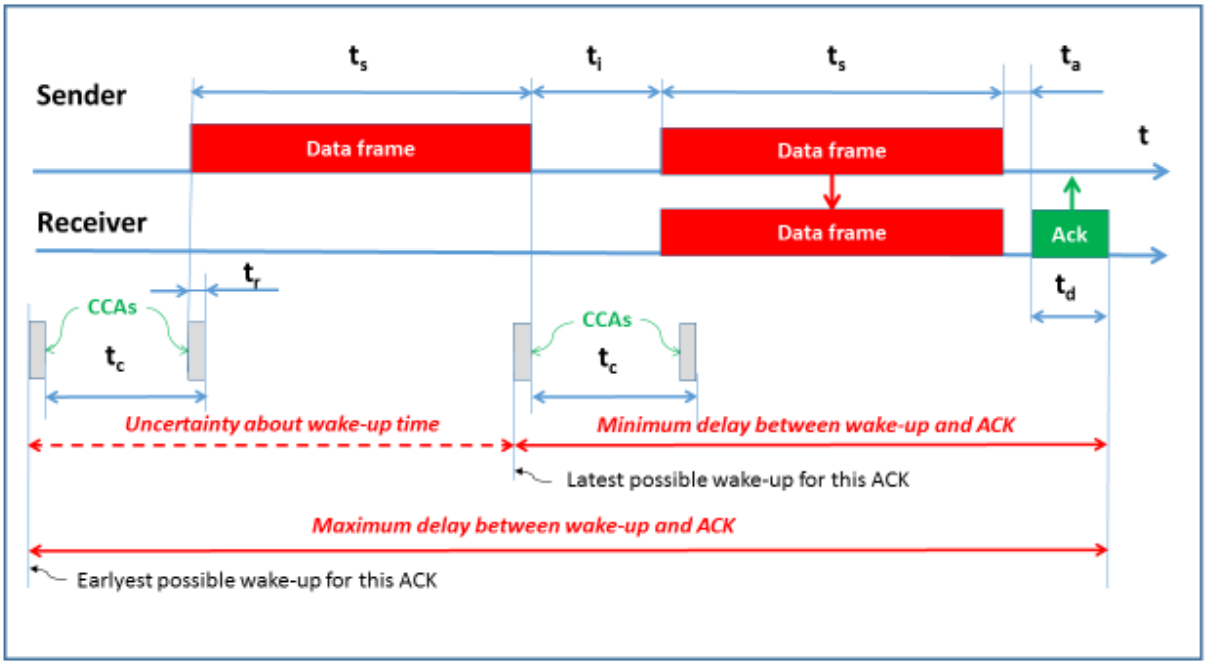

Figure 42: Delay between receiver wake-up and sending of the ACK

For a specific acknowledgment, radio activity could have been detected anytime between the detection by the first CCA of the first byte of the packet preceding the acknowledged one or the detection by the second CCA of the last byte of the same packet.

They claim that their technique allows to reduce the guard time so that each packet only needs to be retransmitted once, which is the minimum possible. However, their model does not consider the difference between the clock frequencies of sender and receiver.

Figure 43 shows sniffer files recorded with different encounter optimization settings, but without the proposed improvement to the ACKs. The top record is obtained with the default Contiki settings, where the factor in the definition of GUARD_TIME is 10. One can observe that most packets need to be retransmitted four to five times, which is certainly suboptimal. The second record corresponds to a factor 5 in the definition of GUARD_TIME. A periodic loss of synchronization causing the loss of many packets is visible. As nothing in the model proposed by M.Michel et al. leads to periodic black-outs the relative shift in time of sender initiated and receiver-initiated events is a likely cause of this periodic abnormal behavior.

A further reduction of the GUARD_TIME results in a total lack of synchronization and a very low PDR as shown in the third record in Figure 43.

The groups of four packets that are correctly received after a renewed initialization of the encounter optimization are four queued packets transmitted consecutively within a single receiver wake-up.

5.2.4.3. Remedies

The fourth record of Figure 43 shows that, when 3 CCAs are used instead of 2, the factor giving the GUARD_TIME can be reduced to 5 without causing synchronization losses and reducing significantly the number of packets that need to be transmitted more than 2 times. For this reason, this malfunction was not explored any further.

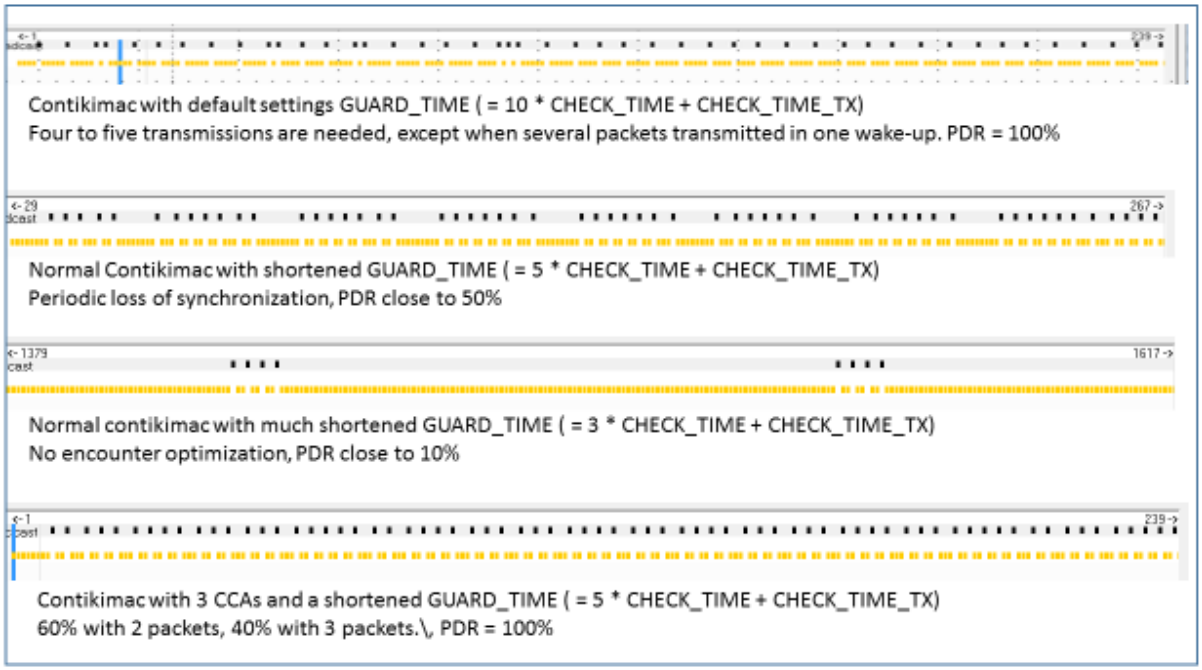

Figure 43: Sniffer records, showing attempts to transmit a packet

The transmitted packet is represented by the yellow dots and ACKs by the black dots

The horizontal scale is the number of the observed packets, regardless of the timing

5.2.5. The encounter optimization is periodically reset without good reason in CXMac

5.2.5.1.Symptoms

As shown in Figure 44, the smooth transmission of packets by the CXMac protocol is interrupted by a spurious reinitialization of the encounter optimization data. These incidents occur sporadically, sometimes two or three times consecutively.

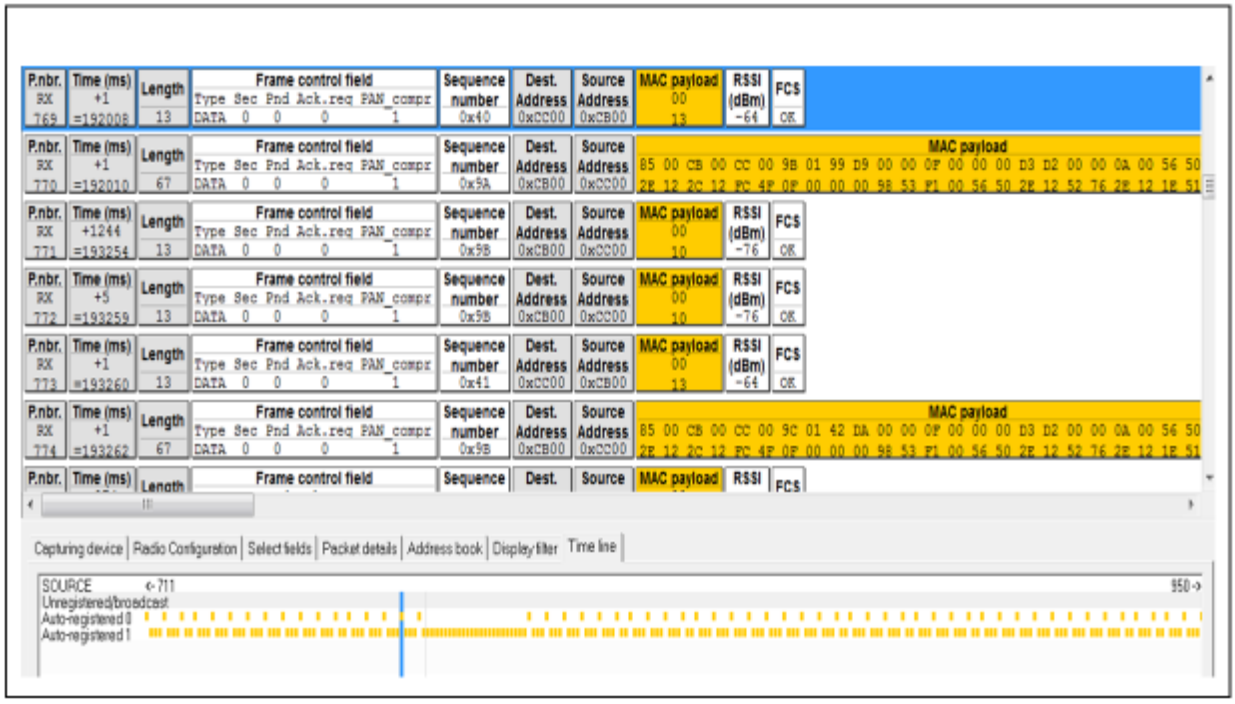

Figure 44: Spurious reset of the encounter optimization with CXMac.
The packet following the blue cursor is still transmitted correctly, just before the encounter optimization reset

5.2.5.2.Investigation

The calculation of the moment the next packet should be sent in order to arrive when the receiver would just have woken-up could, when the interval between successive packets is short, result in a negative delay, which can obviously not be generated. In that case, the next wake-up needs to be targeted.

5.2.5.3.Remedies

The code computing the desired delay has been adapted (the added statement is preceded by a comment starting with MPU):

/ We expect encounters to happen every DEFAULT_PERIOD time units. The next expected encounter is at time e->time +DEFAULT_PERIOD. To compute a relative offset, we subtract with clock_time().*

*Because we are only interested in turning on the radio within the DEFAULT_PERIOD period, we compute the waiting time with modulo DEFAULT_PERIOD. */*

```
now = RTIMER_NOW();
wait = ((rtimer_clock_t)(e->time - now)) % (DEFAULT_PERIOD);
/* MPU: if statement to avoid expected being in the past */
if(wait < 2*DEFAULT_ON_TIME){ wait = wait + DEFAULT_PERIOD; }
expected = now + wait - 2 * DEFAULT_ON_TIME;
```

5.2.6. With LPP, some packets are triplicated

5.2.6.1.Symptoms

When the send power is lowered in a unicast link with LPP as RDC protocol, packets arriving with a RSSI below -80 dBm are systematically triplicated. The malfunction disappears if the DEBUG mode is enabled in the LPP code.

5.2.6.2.Investigation

Observation by means of the sniffer made clear that packets with a RSSI below -80dBm were acknowledged but that these ACKs were not detected and consequently the packets were retransmitted by the CSMA MAC layer of the transmitter. The detection of an ACK was already discussed in paragraph 2.1.2 of this chapter in relation with ContikiMac. Exactly the same code is used in LPP. Figure 45, which is an extension of Figure 40 from paragraph 2.1.2, shows the timing of the 3 different radio status bits that are used to detect the ACK. These bits are tested after a delay t_b following the end of the transmission of the packet. In fact, t_b is the sum of t_a, the time necessary to switch the radio from the “send” state to the “receive” state, and t_d, an additional delay needed to make sure that the presence of the ACK is already reflected in the status bits.

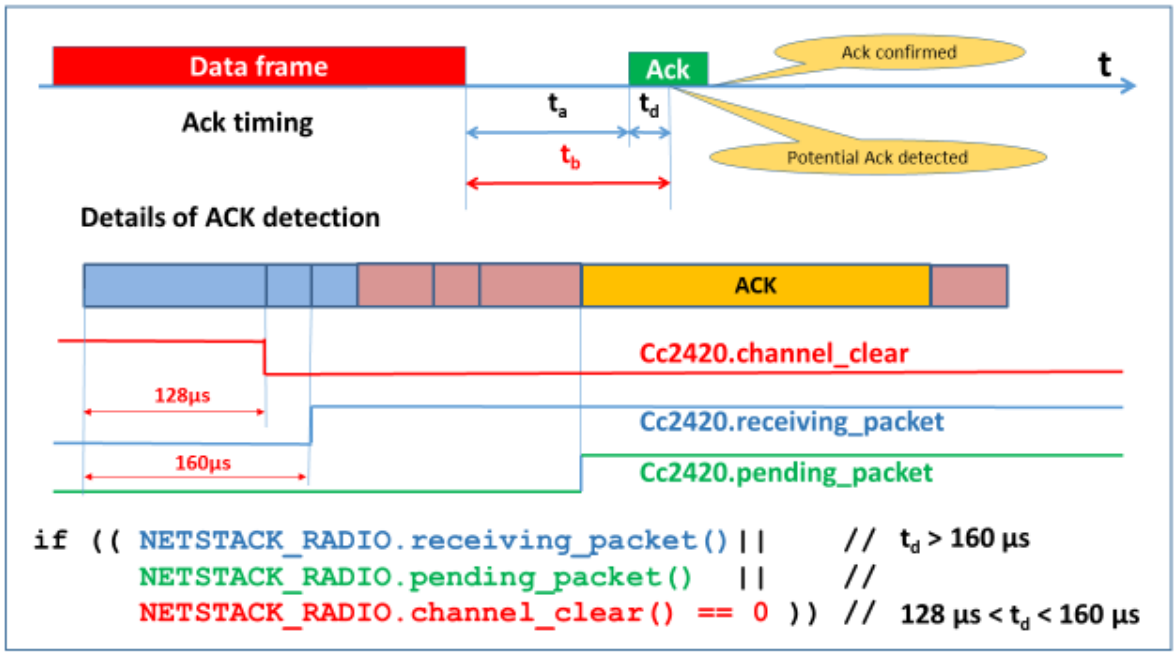

Figure 45: Timing of the detection of an ACK in LPP and ContikiMac

Depending on the timing of the test, the decision is based upon different status bits of the receiver. The first, represented in red, requires a higher RSSI than the two others.

The first bit to detect the incoming packet is the channel_clear bit, mainly used by the carrier sense function of the CSMA MAC protocol. The state of this bit is changed 128µs after the arrival of the first bit of the ACK. The second bit that detects the incoming ACK is the receiving_packet bit, which changes its state 32 µs after the channel_clear bit. There is, however, an important difference between the clear_channel and the packet_received bits: the latter detects all incoming packets while the first detects only incoming packets with a RSSI that exceeds the CCA threshold value set by the CSMA protocol. This means that with values of t_d between 128 and 160µs ACKs with a RSSI below the CCA threshold (by default -77dBm) will not be detected. Enabling the DEBUG mode increases the value of t_d considerably, so that the ACK will be detected by the packet_received rather than the clear_channel bit.

As the detection of ACKs is identical in ContikiMac and in LPP, one can wonder why no triplication occurs in ContikiMac. A first reason is that, as explained in paragraph 2.3 of this chapter, ContikiMac should not work at power levels below the CCA threshold and, in addition, ContikiMac has a filter that eliminates packets with an already received sequence number.

5.2.6.3.Remedies

Increasing t_d, which is a configuration constant, is simple but then, when the DEBUG mode is enabled, the testing occurs after the ACK has been received and the three bits reset. The best solution consists probably to add to the LPP code the filter that eliminate duplicated packets.

5.3. Conclusions

Except for the minor bug mentioned in paragraph 2.5 of this chapter, all other malfunctions are directly related to two physical properties of the motes that have been overlooked by the designers of the RDC protocols in Contiki. The first and most important fact that was overlooked is that motes all have slightly different clock frequencies which can cause black-outs, packet duplications and waste of power. The consequences of these differences could not be discovered by those who tested their protocols by means of the Cooja simulator, because this simulator uses the same clock for all motes.

The clever use of the CCA for other purposes than those it was designed for is also source of some, easy to correct, malfunctions, but again, these malfunctions could not be detected when testing with Cooja because the emulation of a radio link by Cooja is so idealized that CCA thresholds are simply ignored.
For all the observed malfunctions work-arounds have been proposed and, for most of them, implemented and extensively tested but it is clear that further testing, in totally different environments with real applications is required.

The main conclusion of this chapter is that simulation studies and formal software validation techniques are certainly precious and necessary tools to further develop Contiki, but that extensive testing with real hardware is unavoidable to make wireless sensor and actuator networks trustworthy.

Chapter 6: RDC performance comparisons[2]

6.1. Introduction

Once the malfunctions described in the previous chapter had been corrected, it became possible to compare the performances of the different RDC protocols available in Contiki.

The work presented in this chapter was carried out in collaboration with PhD candidate Maite Bezunartea, who contributed equally to the real-world experiments, as well as to the analysis of the data collected during the experiments. The results of these comparisons have recently been published in the IEEE Sensors Journal [6].

6.2. The experimental set-up

All experiments have been performed with the dual network described in chapter 4, black motes use the built-in ceramic antenna for communication with other black motes. All radio channels can be used, except one reserved for the white motes. No modification of the behavior of tested programs in black motes was detected, except for an increase of 2 mJ per packet of the power used, due to the communication with the white motes. White motes use an external antenna, and enough power to reach the white sink through a single hop. The white network uses NullRDC and CSMA to handle possible collisions. Figure 46 depicts the experimental set-up that contains four dual motes denoted M1 to M4. The black network uses channel 26, the white network, channel 16. The power is set to 0dBm in both networks. For unicast experiments the link from black mote M2 to M1 is under study, the traffic between the black motes of M3 and M4 can disturb this communication. For experiments involving broadcast protocols, M1 remains the receiver, M2, a unicast sender and M3 and/or M4 broadcasters.

[2] This chapter is partially based on two publications by Uwase, M. Paule et al. :

1) "Experimental evaluation of message latency and power usage in WSNs" published in the proceedings of the 2014 IEEE International Black Sea Conference on Communications and Networking, BlackSeaCom 2014
2) Experimental Comparison of Radio Duty Cycling Protocols for Wireless Sensor Networks published in the IEEE Sensors Journal, 2017.

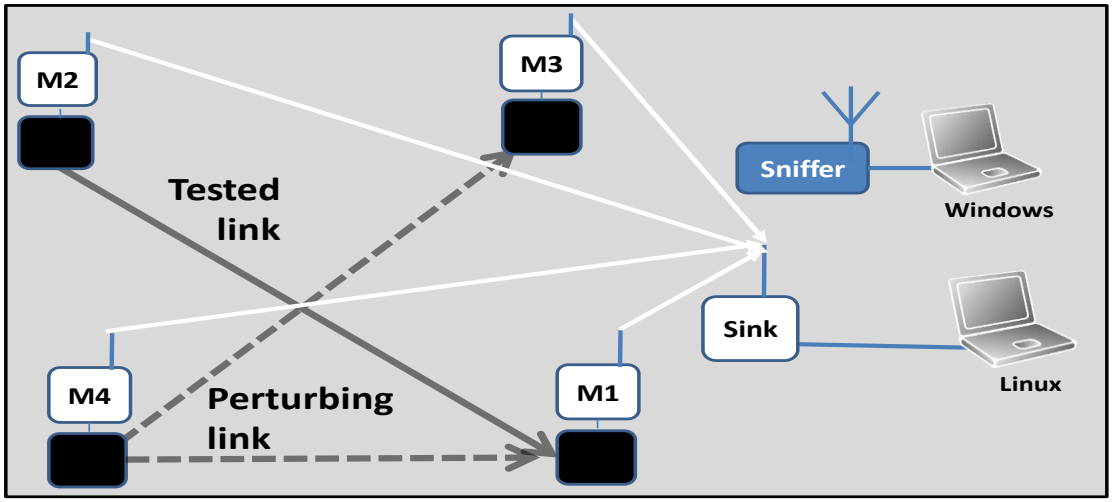

Figure 46: Experimental set-up with 4 dual motes for unicast testing

The communication link between M2 and M1 is under study, while the link between M4 and M3 can be used to create interfering traffic
An additional laptop, running the Texas Instruments sniffer software with a cc2531 radio is used to observe the radio traffic between the black motes and to diagnose any major radio malfunction or configuration errors.

6.3. Traffic profiles for the RDC study

Three main traffic profiles are defined for these RDC studies:

- **No traffic:** Only a RDC driver runs in a black mote.

- **Unicast traffic:** The link under study carries data packets from M2 to M1. Three different traffic conditions are defined:
  - **NP**: No external Perturbations.
  - **PR**: (Perturbation with Receiver) The M2>M1 link is perturbed by an active radio link between M3 and M4.
  - **PNR**: (Perturbation with No Receiver) The M2>M1 link is perturbed by M3 trying, to send packets to M4, which is switched off. Protocols sending until an ACK is received (ContikiMac and CXMac), increase the perturbing effect caused by M3. Indeed, M3 will not receive any ACK and will continue to send packets over entire channel check intervals.
- **Broadcast traffic:** M1 expects packets from M2, M3 and M4. Again, three different traffic conditions are defined:

- **B**: Broadcast by M3, no external perturbation.
- **BB**: Broadcast by both M3 and M4.
- **BU**: Broadcast by M3 and unicast by M2.

	Power			***PDR & Latency***					
Traffic Load / ***RDC Variants***	No traffic	Unicast (NP)	Broadcast (B)	Unicast (NP)	Unicast (PR)	Unicast (PNR)	Broadcast (B)	Broadcast (BB)	Broadcast (BU)
ContikiMAC 2CCAs	4,8,16,32Hz	4,8,16Hz	4,8,16Hz	8Hz	8Hz	8Hz	8Hz	8Hz	8Hz
ContikiMAC 3CCAs	4,8,16,32Hz	4,8,16Hz	4,8,16Hz	8Hz	8Hz	8Hz	8Hz	8Hz	8Hz
CXMAC	4,8,16,32Hz	4,8,16Hz	4,8,16Hz	8Hz	8Hz	8Hz	8Hz	8Hz	8Hz
LPP	4,8,16,32Hz	4,8,16Hz	4,8,16Hz	8Hz	8Hz	8Hz	8Hz	8Hz	8Hz
LPP with Pending Broadcast							8Hz	8Hz	8Hz
NullRDC	128Hz	128Hz	128Hz	128Hz	128Hz	128Hz	128Hz	128Hz	128Hz

Table 11: Overview of the performance measurements

For power measurements, different wake-up frequencies have been tested, for other tests only the default 8Hz wake-up frequency has been used. For NullRDC the radio channel is checked with a frequency of 128 Hz.

In all the previous traffic profiles, packets have a MAC payload of 59 bytes. This means that transmitting one packet lasts ± 2,3 ms. On all links, test packets are sent at an average rate of 1 packet/s (the inter-packet interval is uniformly distributed between 10 and 1990 ms) except on the unicast perturbing link between M4 and M3 which sends at twice that rate.

The experiments lasted for at least 20 minutes, resulting in a minimum of 1200 packets sent over each tested link. Average values were computed for PDR, power and latency as well as 80% percentiles for latency (See section 6.6 for details of latency calculations). By subdividing the 1200 measurements in 34 groups of 34 measurements we checked that we had enough measurements for asserting with a 95% confidence that the mean values obtained for power and latency were accurate within +/- 5% whereas the PDR figures close to 100% had an accuracy better than +/- 1%. Table 11 shows the different measurements made by means of the different RDC protocols studied. The wake-up frequencies used for the experiments are given in the table. Not all performed measurements are discussed here, only the most interesting results are shown in graphs grouping the results of several experiments in order to facilitate comparisons.

6.4. Experimental results

6.4.1. Average power consumption

As saving energy is the main purpose of RDC protocols, an estimation of the average power consumption was made. Figure 47 shows the average power consumed by the black mote of M2 with no traffic as a function of the wake-up frequency for each protocol variant. The much higher power consumption of NullRDC with a fixed channel check rate of 128 Hz, could not easily be included in the graph.

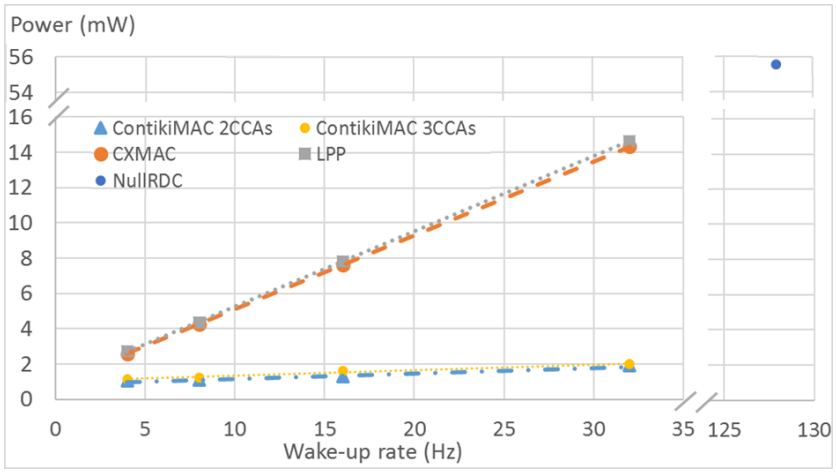

Figure 47: Power usage without traffic as a function of RDC wake-up frequency.

The power consumption observed for NullRDC is consistent with the observations made for CXMac and LPP if one considers a wake-up rate of 128Hz.

The average power consumption of a mote as a function of the RDC protocol's wake-up rate is expected to follow the linear function for all protocol variants:

$$Power = P_0 + (Wake\text{-}up\ Frequency) * (Energy\ per\ wake\text{-}up) \qquad (1)$$

Where P_0 is the power used by the mote with no traffic and no wake-up. From the linear regressions we can calculate an estimate for the coefficients for each variant of ContikiMac, for CXMac and LPP. The results of the calculations are shown in Table 12. The *Energy per wake-up* for CXMac and LPP evaluates to ±0.42 mJ. For ContikiMac performing 2 or 3 CCAs per wake-up it evaluates to ±0,03 mJ. A CCA indeed consumes fewer energy compared to sending probes and/or listening. The measurement for NullRDC follows formula *(1)* for the *Wake-up frequency* set to 128Hz.

Protocol	**Power(mW)**	**Energy per wake-up(mJ)**	**R^2**
ContikiMAC 2 CCAs	0.85	0.03	0.98
ContikiMAC 3CCAs	1.03	0.03	0.97
CXMAC	0.91	0.42	1
LPP	0.98	0.43	0.99

Table 12: Idle power figures from regression calculations

The next comparison shown in Figure 48 considers the average power needed to transmit one unicast message per second without perturbation (NP). This average power is represented by the total height of the bar. The bar is composed of two parts: one represents the average power spent by the execution of the RDC with no traffic (blue part of bar) and the other one is the average extra power needed for sending the unicast packet (red part of bar). The red part of the bar has been obtained by subtracting the measured average no-traffic power consumption from the measured total average consumption. The results are shown for wake-up frequencies of 4, 8 and 16 Hz for all protocol variants.

One can observe that the traffic-dependent part of the average power consumption does not change a lot with the RDC protocol type, neither with the wake-up frequency. The sender mote for CXMac and LPP spends more power in executing the RDC for growing wake-up frequencies, a number which gets significantly larger than the average power for sending packets at 1 packet/s for wake-up frequencies above 4Hz.

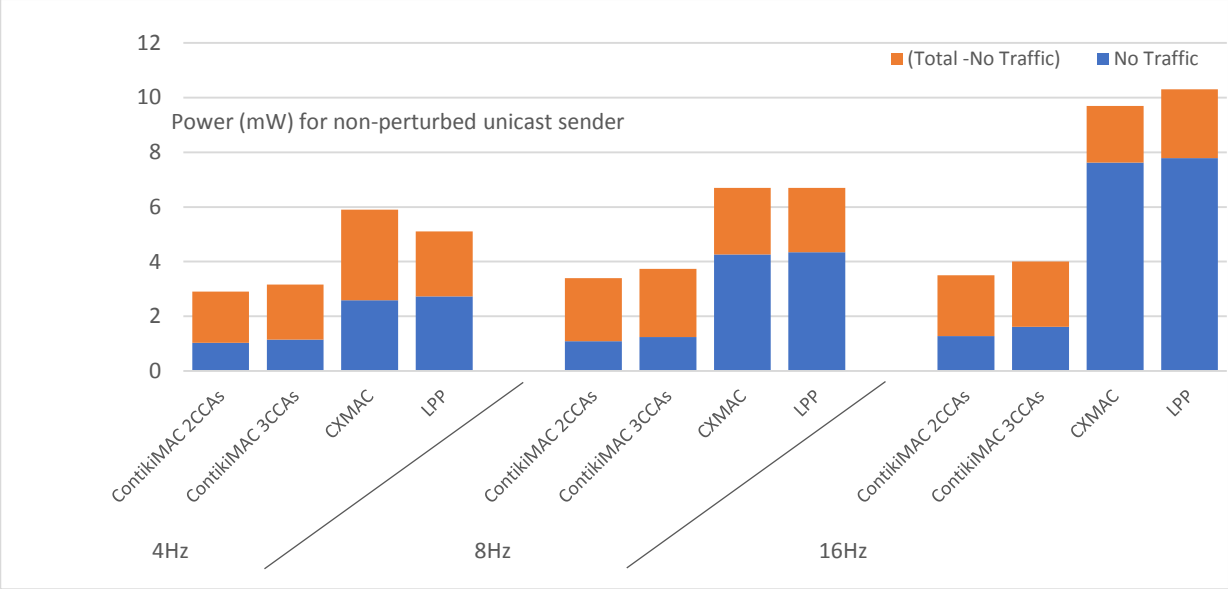

Figure 48: Unicast average power consumption at the sender

The lower blue part of each bar is the average power when no traffic is present whereas the total height of a bar represents the average total power consumption. The wake-up frequencies are given underneath the RDC protocol names. For NullRDC, the average total power consumption is 55mW.

A last comparison considers single point to point broadcasts (B) for all protocol variants, for wake-up rates 4, 8 and 16 Hz. The results are shown in Figure 49 which presents results in the same way as Figure 48. We can compare the average power for unicast versus broadcast by examining Figure 48 and Figure 49.

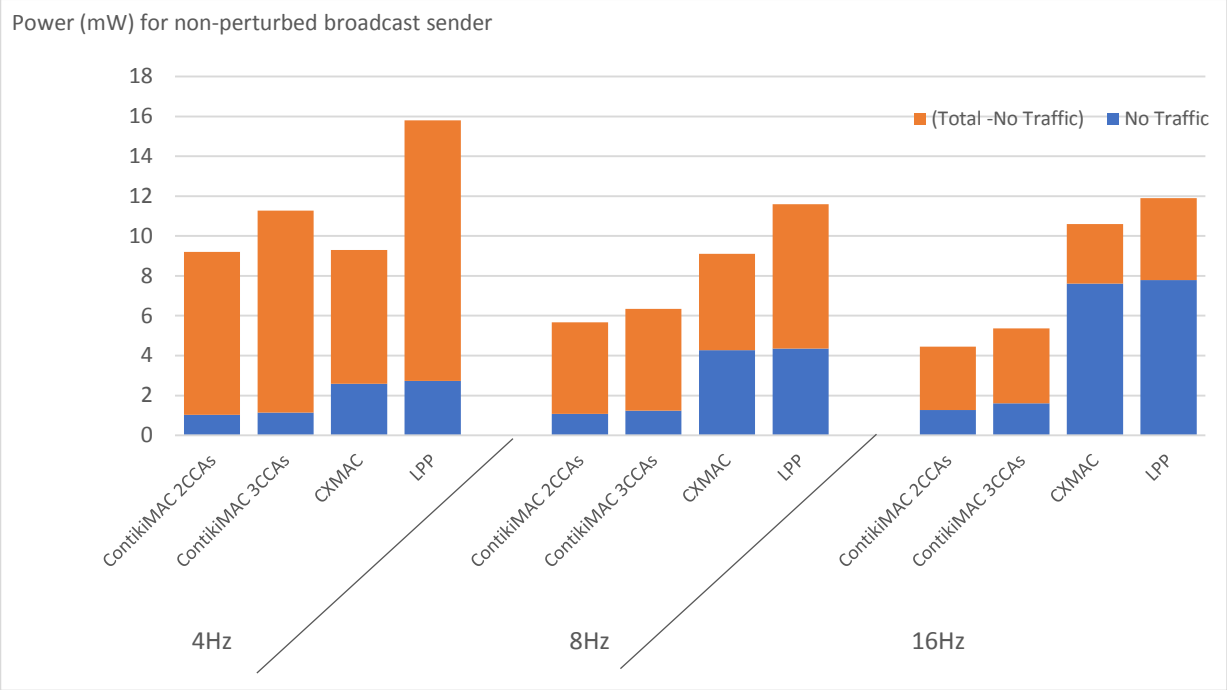

Figure 49: Broadcast power consumption at the sender

The lower part of each bar is the average power in absence of traffic whereas the total height of a bar represents the average total power consumption. The wake-up frequencies are given underneath the RDC protocol names. For NullRDC, the average total power consumption is 55mW.

Two facts deserve special attention: the power required for sending a broadcast packet is much higher than for unicast communications and reducing the wake-up rate does no longer significantly reduce the power consumption. This is due to the fact that the traffic dependent part of the power consumption increases when the wake-up frequency decreases because the channel check interval increases. For ContikiMac and CXMac, the increased power demand is due to the repeated transmission of the packet during a full channel check interval. For LPP, it is due to the sender's radio receiver staying awake during that same interval. Since transmitting broadcasts is handled the same way in ContikiMac and CXMac, one could expect similar power requirements. However, according to sniffer observation, CXMac repeats packet transmission every 5 ms, whereas ContikiMac did it every 2.8 ms, which explains the lower power needs of CXMac in broadcast sending. CXMac can afford a time lapse of 5 ms between broadcast packets because the receiver will listen for a time lapse greater than this 5ms gap. ContikiMac retransmits the packet as soon as possible, to satisfy the timing required for the detection of packets by means of the CCAs.

The NullRDC protocol does not appear on the graphs, because its power requirements are much higher than those represented. They are mentioned in the graph's caption. They are almost constant, whatever the traffic, both for unicast and broadcast. This is normal as the radio is the big power consumer and the radio consumes about the same power in sending/listening or no-traffic mode.

The power consumptions of the receiving mote and of the transmitting motes with traffic conditions PR, PNR, BB and BU are not shown because they are almost independent of the traffic conditions.

6.5. Packet Delivery Ratio (PDR)

A first series of comparisons will be devoted to a study of the PDR for the different variants of the protocols at a wake-up frequency of 8 Hz and also for NullRDC on a point to point link without external perturbation for the unicast NP and broadcast B traffic conditions (scenario NP and B). Figure 50 shows that the PDR is close to 100% for all unicast experiments without external perturbations. This high PDR is due to the excellent radio conditions and the absence of competition for the medium. Under those conditions, broadcast also yields an almost 100% PDR for ContikiMac, CXMac and NullRDC.

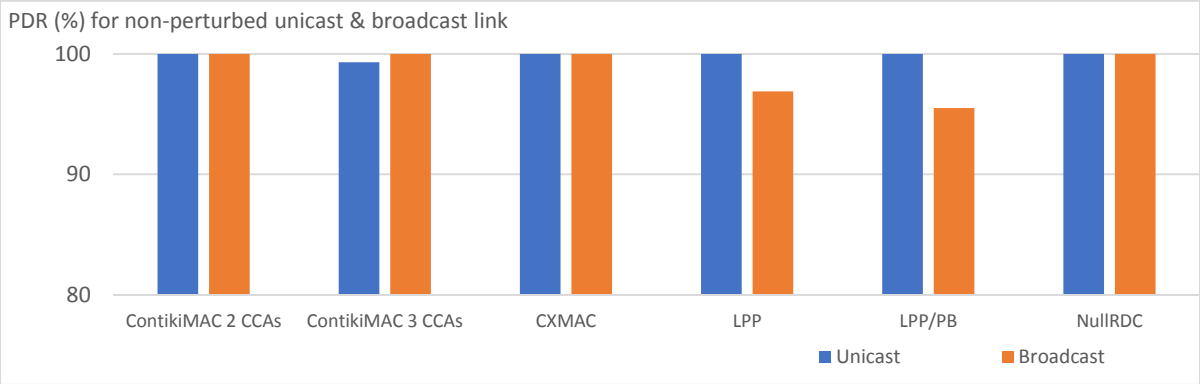

Figure 50: PDR for a unicast and a broadcast transmission both without perturbation

The two tested options for LPP (with and without Pending Broadcast handling (see page 38) are represented separately

The lower PDR observed with both LPP varieties appeared to be due to the fact that the LPP receiver returns immediately to the sleep mode after reception of a broadcast packet. By doing so it ignores subsequent packets transmitted by the sender in the same wake-up slot.

A second study is devoted to the PDR for the different variants of the protocols at a wake-up frequency of 8 Hz and for NullRDC for a point to point with and without external perturbations for the unicast traffic (conditions NP, PR and PNR). The results are depicted in Figure 51 (the unicast NP data are those also shown in Figure 50).

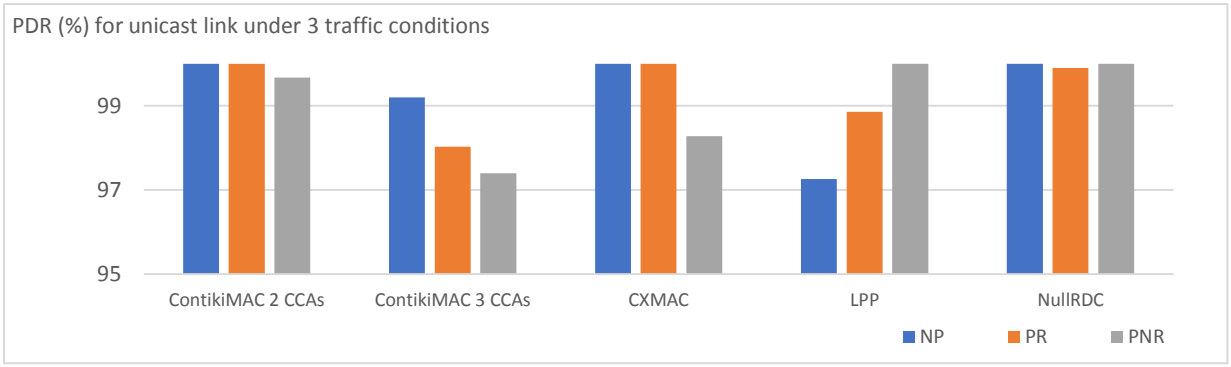

Figure 51: PDR for a non-perturbed (NP), perturbed with receiver (PR) and perturbed with no receiver (PNR) unicast link

The PDR values for both varieties of ContikiMac are above 99%. This shows that the reduction of the interval between repeated packets implemented in the variety with 2 CCAs was effective (See chapter 5, par.2.1.3), however, the same test done with ContikiMac as available from Github, yields a PDR of only 85%, proving that, if one wants to avoid using nonstandard features of the cc2420 radio, the 3 CCAs are necessary. The PDR of ContikiMac with 2 CCAs, CXMac and NullRDC are not affected by a successful (traffic condition PR) perturbing transmission in the neighborhood, whereas LPP suffers somewhat from the additional wake-up probes. ContikiMac and CXMac are more affected by an unsuccessful sender (traffic condition PNR) that makes repeated attempts to reach a non-existing receiver and doing so saturates the radio channel during three (the number of CSMA attempts) entire RDC periods.

Why the 3 CCAs version of ContikiMac suffers more than the version with 2 CCAs remains, at the moment, an open question. The PDR of LPP is improved when the perturbing receiver is switched off, because this reduces the number of wake-up probes in the air. NullRDC performs better because it transmits only one packet per CSMA packet. This also implies that the perturber sends way less packets compared to the perturbers running the other RDC protocols. Global medium occupancy is therefore much lower.

A pre-ultimate facet of the PDR study considered the impact of a mix of unicast and broadcast traffic. Figure 52 shows the PDR when one receiver gets unicast messages from one transmitter and broadcast messages from another one (traffic condition BU).

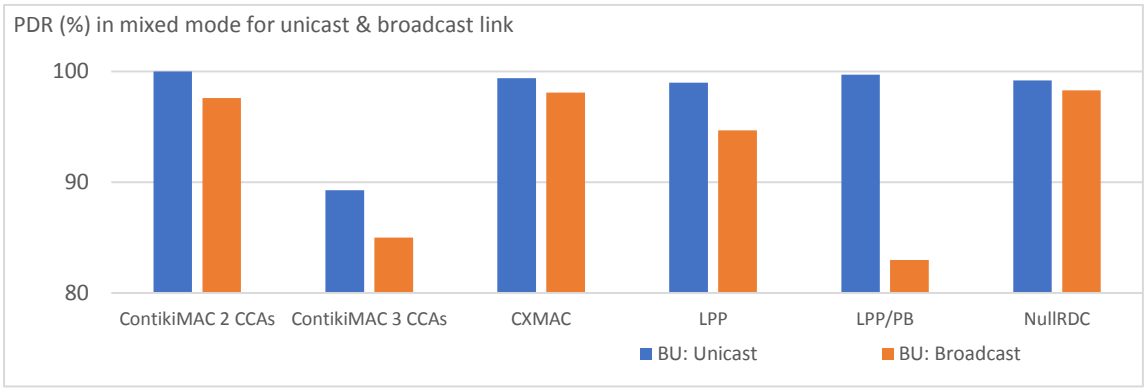

Figure 52: PDR values for simultaneous unicast transmission and broadcast transmission (BU)

The unicast and broadcast traffic reach the same receiver.

From a PDR point of view, except for ContikiMac with 3 CCAs, the unicast traffic is unaffected by the broadcast traffic, but the latter undergoes losses that can reach 20%. A detailed analysis by means of the packet sniffer shows that the difference is due to the CSMA protocol that tries to transmit unicast packets up to three times while making only one attempt for broadcast packets. The lower PDR for broadcast traffic observed with LPP was already explained when pure broadcast traffic was analyzed higher up in this paragraph. The much lower figure obtained when the pending broadcast option is enabled, results from the longer time packets stay in the send buffer, which increases the number of packets transmitted within a single wake-up period. Moreover, as two rather than one probe are involved, the risk of collision of these probes is also larger.

The observed results for ContikiMac with 3 CCAs confirm the, still unexplained, higher sensitivity of this protocol to external perturbations already observed in Figure 52.

A last facet of the PDR study involves two nodes sending broadcast packets to the same receiver. Figure 53 compares, the PDR values for traffic condition B (one broadcast sender without any external perturbation) with the PDR values for traffic condition BB. It shows clearly that the “Pending Broadcast” extension to the LPP protocol significantly improves the PDR in case of multiple broadcast senders.

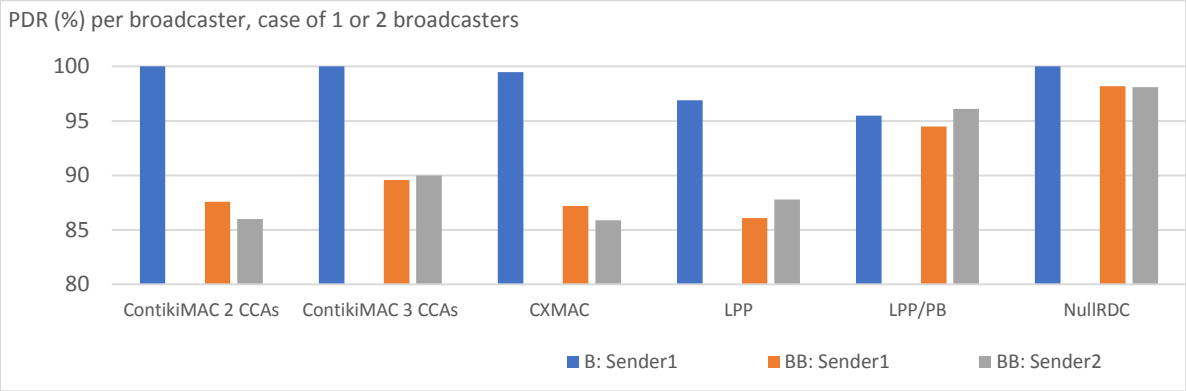

Figure 53: PDR for a broadcast transmission in case of one broadcaster (B) or two simultaneous broadcasters (BB)

For the last case the PDR for each broadcast sender is shown.

6.6. Latency

Figure 54 is a histogram that shows the distribution of latencies measured over a timespan of 12h on a perturbed point to point ContikiMac link with the receiver of the perturbing link switched off (traffic condition PNR) at a wake-up frequency of 8Hz.

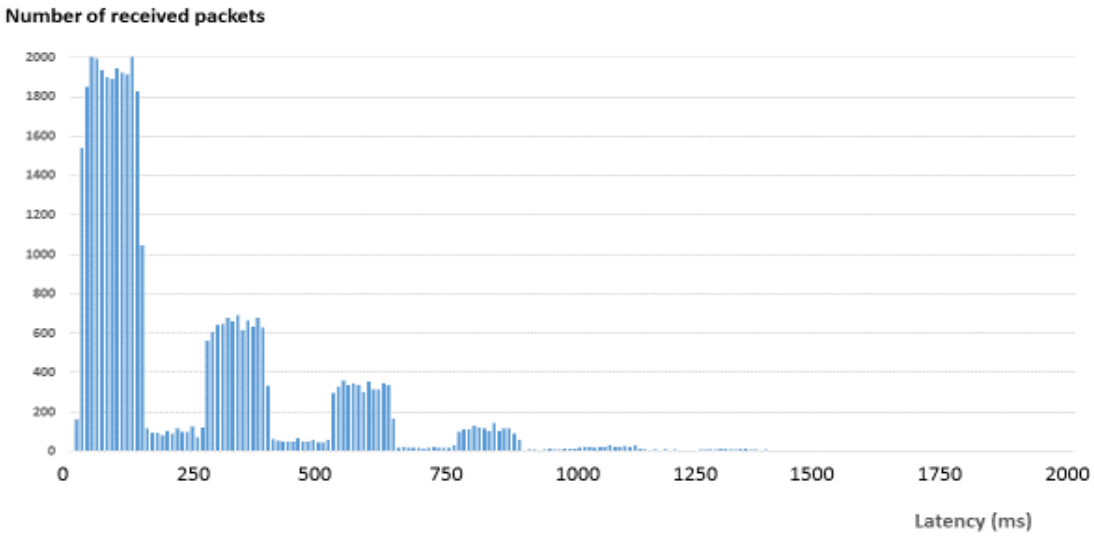

Figure 54: Histogram of latencies for ContikiMac

In a maximally perturbed radio environment (PNR traffic profile). Wake-up frequency = 8Hz, Duration of the experiment = 12h. The observed latencies are distributed between 15 and 1960 ms and clustered in distinct sets whose width is directly related to the time interval between successive retransmission attempts by the MAC layer. One should also observe that aprox.65% of the packets belong to the first cluster, with latencies below 175ms.

To understand this histogram, different delays that a packet undergoes need to be considered: First of all, there is the delay between the moment the application level traffic generator creates packets and the moments the receiver wakes up and can receive these packets. To avoid systematic errors due to this delay, the inter-packet interval is uniformly distributed at random between 10ms and 1990ms to eliminate, in the computed average latency, the influence of the moments packets are generated. For individual packets this creates a random additional delay comprised between 0 and CCI (the wake-up interval), which explains the stable frequencies within the clusters.

The existence of multiple clusters results from the "carrier sense" function of the MAC layer: Before sending a packet, six successive CCAs are done to prevent interference with another transmission. If the issue of this test is negative or if a transmitted packet is not acknowledged, the packet is retransmitted after a random delay, which is a multiple of the CCI.

The non-zero frequency of latencies between the clusters results probably from buffering of packets in the sender after a collision, while waiting for the next optimized encounter with the woken receiver. This results in several packets being transmitted consecutively. For a small number (5) of packets with a latency between 180 and 200ms, this has been verified by means of sniffer records and, indeed, they all were the second or the third of a sequence of two or three packets transmitted in one wake-up cycle.

With such distributions classical indices such as mean and variance have not much significance [83]. Therefore, it was chosen to show the 80% percentile and the average latency of packets that belong to the first class (those that reach their destination without retransmission) to compare the protocols. Figure 55 gives for unicast transmissions the 80% percentile latency figures observed with the three traffic conditions (NP, PR and NPR). The percentiles were based upon the number of transmitted packets to give a true image of the expectations an applications programmer can have about the reliability and the latency of a link.

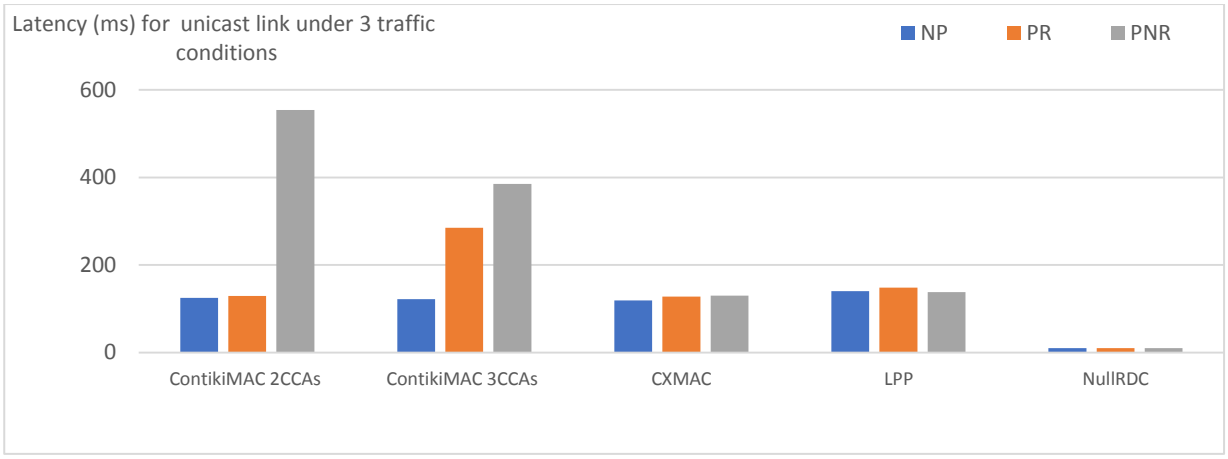

Figure 55: 80% percentile of packet latency for a unicast link that is non-perturbed (NP), perturbed with receiver (PR) and perturbed with no receiver (PNR)

Figure 56 gives the average latencies measured on the unicast packets that were successful on the first transmission attempt and on the broadcast packets that are never retransmitted by the CSMA layer.

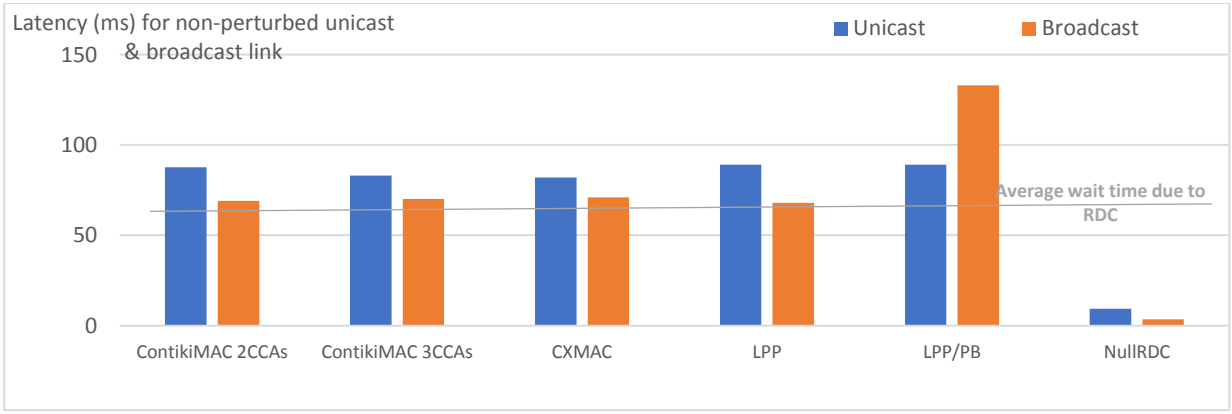

Figure 56: Average packet latency without any external perturbation for a broadcast and a unicast link

The two tested options for LPP (with and without Pending Broadcast handling) are represented separately

As, on average, each packet has to wait for half an RDC wake-up period before it can be transmitted, the difference between the average latency and half the RDC period can be considered as the true average packet latency due to the entire transmission link, from application to application. It is clear that no significant differences between the considered protocols are uncovered by this graph.

6.7. Related Work

To the best of our knowledge, few systematic experimental comparisons of RDC protocols for wireless sensor networks have been published but many specific RDC protocols have been studied, often with suggestions for improvements.

Peter Ruckebush et al. [84] compared the performance of different RDC protocols available in Contiki for RPL-based networks. They observed the same kind of discrepancies between the results obtained from the Cooja simulator and from a testbed with real motes. However, their findings cannot be directly compared to ours because they used the default options provided by Contiki, which tend to favor some RDC protocols over others.

Julien Beaudaux et al. [85] have done an extensive study of the X-Mac protocol. They have identified several flaws in the specifications and evaluated consequences on the performance. They have confirmed their findings by large scale experiments with real motes. However, as they used their own version of X-Mac on motes quite different from ours, direct comparisons are difficult, although similarities are obvious.

François Despaux et al. [86] have published an in-depth analysis of the ContikiMac source code and instrumented that code to build a Markov model of the protocol through process mining techniques. The resulting model does not seem detailed enough to reveal the problems that were identified in the experiments described here.

Mathieu Michel et al. [52] studied in detail the effect of interferences on the performance of ContikiMac and more specifically, on the power consumption. They suggest an improvement that could result in significant economies, even without interference issues. It seems worthwhile to implement these improvements and repeat the former measurements.

Milosh Stolikj et al. [87] discovered how interactions of ContikiMac, CSMA and a Trickle timer can considerably degrade the performance of algorithms based upon a trickle timer, such as maintaining the DODAG in a RPL network. They proposed small changes to the MAC layer to prevent such problems. As the presented tests did not involve Trickle timers, these issues were not explored.

Bogdan Pavkovic et al. [88] explored the interactions between the routing layer and the RPL layer and concluded that the topology of the network itself should be matched with the routes determined by the RPL algorithm because mismatched can seriously impair the global performance.

Based on similar considerations, Duquennoy et al. [89] abandoned altogether the asynchronous RDC protocols and adopted TSCH [41], a multichannel synchronous MAC protocol, to improve the performance and the reliability of RPL networks. Comparing experimentally, in an identical environment, their approach with one based upon on a corrected ContikiMac should be an interesting topic for future PhD research.

Chapter 7: General Conclusions

7.1. Introduction

To summarize in a few words this work, one could say that the implementation in Contiki of what is called “layer 2” in the OSI model has been extensively tested and evaluated and reverse engineered to understand the unexpected results obtained during this investigation. This made us aware of different issues about network layering, software engineering, upward compatibility, testing and performance comparisons that have a scope much broader than this specific research about radio duty cycling for energy savings. Our personal conclusions about what we observed in these varied fields will be summarized in the following sections. Thereafter we will also, in the narrower scope of our research, mention which research topics could contribute to a successful deployment of the Internet of Things.

7.2. Network layering

For almost 50 years, data networks have been designed and build as a stack of independent hardware or software layers, each layer offering specific services to the upper layers based upon the services obtained from the lower layers. This approach results in modular systems that offer a multitude of interoperable choices at each layer. The design and the implementation of each layer can be done by teams specialized in the issues proper to their layer. Without such a layered approach, universal communication systems, such as the Internet, would not be feasible.

However, due to the often-narrow specialization of the teams designing the layers, small mismatches between requirement and offering of services exist. Our work uncovered several such mismatches between the physical and the MAC layer: the designers of all asynchronous RDC protocols had neglected the drift of mote clocks and the designers of ContikiMac had paid insufficient attention to timings and signal levels specified and even standardized, for the physical radio communications. This resulted in mismatches between the physical and link layers that caused rare but unacceptable malfunctions. We conclude from this that software designers working for the lower layers in the communications stack need a better insight in hardware issues than what traditional software education provides.

7.3. Software engineering

No widely accepted consensus about documenting software exists. The approach adopted by the contributors to the Contiki source code consists in giving explanatory names to all variables, functions and macros. Comments in the code are sparse, and usually reserved for very tricky pieces of code. A directory intended for storing documentation is part of Contiki but few contributors to the software add something to that directory. On the other hand, a large number of well written application examples are included to allow beginners to quickly obtain simple working systems that can easily be expanded.

In our opinion, the use of meaningful names and the availability of many application examples is useful for gaining quickly a superficial insight in the software and for developing applications similar to the examples given. This quick and superficial approach often obscures the code of many applications, because parts of the examples irrelevant for the new application are left in because their role is not clear. This practice is often the source of "subtle bugs". More explanatory comments might encourage attempts to better understand the subtleties of the software and prevent many bugs and also, considerably help those who try to find causes of malfunctions.

7.4. Backward compatibility

Contiki releases 2 and 3 support an amazing broad range of processors and communication interfaces, some of them going back to the early days of Contiki, almost 40 years ago. This has been made possible by implementing common functionalities, such as communication protocols, by means of generic programs with many conditional calls to device specific macros expanded at compile time when needed for a specific device. Which macros need to be expanded is determined by means of configuration files defining the properties of the different platforms supported by Contiki. However, the progress made by the semiconductor technologies results in major architectural differences between the oldest and the newest devices supported by Contiki.

This has three negative consequences: first the number of conditional statements in the generic parts of the software has grown considerably, making it less and less understandable and modifiable; second, some new devices have features that were not foreseen in the generic framework, and therefore not available for new application and third, and that is the most

serious issue, some new devices lack features that were considered universal when the generic software was written, and require serious rewriting, or worse, are incorrectly controlled. As an example, the radios in use when ContikiMac was designed had separate senders and receivers, that could be switched on and off independently. More modern radios combine the sender and receiver circuits in a single chip that can be switched between the sending and receiving states, but that switching takes a non-negligible time, which is ignored in the generic code for ContikiMac. This causes the incompatibility between the timing of the ContikiMac RDC protocol and the specifications of the IEEE 802.15.4 conformant radios we discovered, as described in chapter 5.

Periodically eliminating from the list of supported devices those that are obsolete and no longer in use and refactoring the generic software in order to eliminate the no longer needed requirements seems a necessary condition to keep legacy software attractive. The recent release of Contiki NG with a considerably reduced list of supported devices and protocols shows that this opinion is shared by an active part of the Contiki community.

7.5. Testing

For finding the deep causes of the malfunctions that were observed during initial, straightforward tests by means of simple application level programs, many different testing strategies were considered and developed.

7.5.1. Testing by means of Cooja

The overall correctness of the algorithms and their implementations can be tested by means of Cooja simulations. Cooja itself consists is set of Java programs that emulate a generic mote, so that all the generic software, including the operating system, can be executed on such virtual motes. Of course, device specific parts of the software, and especially those implying timing constraints, cannot be tested trough Cooja. As the generic radio emulated by Cooja implements an ultra-simplified model of radio propagation, testing issues related to RSSI values, CCA thresholds and interferences between senders is almost useless. The same applies to everything related to clock drift, because in Cooja, all motes use a common clock.

7.5.2. Testbeds

As Cooja simulations are insufficient for in depth testing of wireless sensor networks, adequate testbeds are needed. The requirements for such testbeds depend strongly on their intended purpose: for testing new algorithms in large scale networks, possibly even with some mobile motes, publicly accessible testbeds exist and are popular among researchers.

These testbeds, however, are built around special purpose motes instrumented for testing and for evaluating performance indices. They run modified versions of common operating systems and are often surrounded by Faraday cages. They are seldom useful for diagnosing subtle malfunctions occurring when WSNs are being deployed in a real-world environment. For this purpose, we have designed and built networks with dual motes, easily deployable in any environment, indoors or outdoors. In our dual motes, one runs the software under test while the other observes and reports via radio, without causing significant perturbations, the activities of the first one.

7.5.3. Real-time debugging

Most of the generic programs in Contiki have a debug mode. When this mode is enabled, information useful for understanding and eventually debugging the programs is written down at run time on the computer connected to the mote via the USB port. However, writing on the USB port slows down the execution of the programs being debugged, which can totally modify the behavior of time critical threads. We found several functions which required the debugging mode to be enabled to work properly. This made us suspect that these functions were only tested in debugging mode. We concluded that, although precious for understanding algorithmic errors, the debugging mode is almost useless when time critical issues exist but can be adapted by setting flags and timestamps in the time critical sections and translating this information in clear messages when the mote is idle. We used this technique for a few subtle issues such as the ones described in chapter 5, section 5.2.6.

7.6. Performance comparisons

Although the comparison of global network performance indices when different RDC protocols were used underneath the RPL routing protocol was a primary goal of this research, we had to restrict the scope of these comparisons to the RDC protocols alone, because not enough time was left after we had identified and eliminated the numerous protocol malfunctions.

From the comparison of the three RDC protocols we studied intensively, we concluded that, except for energy consumption, where ContikiMac is much better than all its competitors, no significant differences in performance could be brought to light.

7.7. Future work

Two continuations of this work seem worthwhile: One would consist in pursuing the original goal of comparing the performance of the RDC protocols when associated with the RPL network protocol. The other would consist in broadening the study to cover new RDC protocols, specially designed for running underneath RPL. We have developed and implemented the tools necessary for these continuations but upgrading our tools for adapting them to the technology of newer motes is probably necessary but could be done without changing the global approach to testing and measuring.

List of figures

List of tables

List of acronyms

6LOWPAN	IPv6 over Low power Wireless Personal Area Networks
ACK	Acknowledgement
CCA	Channel Clear Assessment
CCI	Channel Check Interval
CoAP	Constrained Application Protocol
CPU	Central Processing Unit
CSMA	Carrier-sense Multiple Access with collision avoidance
DIO	DODAG Information Object
DODAG	Destination Oriented Directed Acyclic Graph
EWSN	Embedded Wireless Systems and Networks
FIFO	First In First Out
GPIO	General-Purpose Input/Output
GPS	Global Positioning System
HTTP	Hypertext Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
INRIA	Institut de Recherche en Informatique et en Automatique
IoT	Internet of Things
J-DSP	JAVA Digital Signal Processing
LQI	Link Quality Indicator
MAC	Multiple Access Control
OSI	Open Systems Interconnection reference model
PDR	Packet Delivery Rate
RDC	Radio Duty Cycle
RPL	Routing Protocol for Low-Power and Lossy networks
RSSI	Received Signal Strength Indicator
TSCH	Time Slotted Channel Hopping
TWIST	TKN WIreless NetworkS Testbed
USB	Universal Serial Bus
WSAN	Wireless Sensor and Actuator Network
WSN	Wireless Sensor Network

References

[1] M.-P. Uwase, J. Tiberghien, and K. Steenhaut, "Interactive Tools for Learnin g Sensor Network Basics," in *Proceedings of the second international conference on Advances in Engineering and technology, Entebbe-Uganda*, 2011, pp. 148–154.

[2] R. Lübke, P. Büschel, D. Schuster, and A. Schill, "Measuring accuracy and performance of network emulators," in *2014 IEEE International Black Sea Conference on Communications and Networking, BlackSeaCom 2014*, 2014.

[3] M. P. Uwase, M. Bezunartea, T. L. Nguyen, J. Tiberghien, K. Steenhaut, and J. M. Dricot, "Experimental evaluation of message latency and power usage in WSNs," in *2014 IEEE International Black Sea Conference on Communications and Networking, BlackSeaCom 2014*, 2014, pp. 69–72.

[4] M.-P. Uwase, M. Bezunartea, J. Tiberghien, J. M. Dricot, and K. Steenhaut, "ContikiMAC, some critical issues with the CC2420 Radio," in *EWSN '16 Proceedings of the 2016 International Conference on Embedded Wireless Systems and Networks*, 2016, pp. 257–258.

[5] M. Bezunartea, B. Sartori, J. Tiberghien, and K. Steenhaut, "Tackling malfunctions caused by Radio Duty Cycling protocols that do not appear in simulation studies," in *15th ACM Conference on Embedded Networked Sensor Systems*, 2017.

[6] M.-P. Uwase, M. Bezunartea, J. Tiberghien, J.-M. Dricot, and K. Steenhaut, "Experimental Comparison of Radio Duty Cycling Protocols for Wireless Sensor Networks," *IEEE Sens. J.*, vol. 17, no. 19, 2017.

[7] K. Romer and F. Mattern, "The design space of wireless sensor networks," *IEEE Wirel. Commun.*, vol. Volume 11, no. Issue 6, pp. 54–61, 2004.

[8] P. Juang, H. Oki, Y. Wang, and D. Rubenstein, "Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with ZebraNet," in *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X)*, 2002, pp. 96–107.

[9] P. Zhang, C. M. Sadler, S. A. Lyon, and M. Martonosi, "Hardware design experiences in ZebraNet," in *SenSys'04 - Proceedings of the Second International Conference on Embedded Networked Sensor Systems*, 2004, pp. 227–238.

[10] K. Branko, G. S. D., B. R. C., and M. M. W., "Design and performance of a wireless sensor network for catchment-scale snow and soil moisture measurements," *Water*

Resour. Res., vol. 48, no. 9, 2012.

[11] K. Martinez, P. Basford, D. De Jager, and J. K. Hart, "Using a heterogeneous sensor network to monitor glacial movement," in *10th European Conference on Wireless Sensor Networks 10th European Conference on Wireless Sensor Networks*, 2013.

[12] K. Martinez and P. Basford, "Robust wireless sensor network performance analysis," in *Proceedings of IEEE Sensors*, 2011, pp. 203–206.

[13] Asanka Sayakkara *et al.*, "Poster: A Low-cost Elephant Localization... (PDF Download Available)," in *EWSN '17 Proceedings of the 2017 International Conference on Embedded Wireless Systems and Networks*, 2017, pp. 226–227.

[14] H. Alemdar and C. Ersoy, "Wireless sensor networks for healthcare: A survey," *Comput. Networks*, vol. 54, no. 15, pp. 2688–2710, Oct. 2010.

[15] "http://nikerunning.nike.com." [Online]. Available: https://store.nike.com/be/en_gb/pw/running-shoes/8yzZoi3?ref=https%253A%252F%252Fwww.google.com%252F&ipp=120. [Accessed: 02-Jul-2010].

[16] A. Bayo, D. Antolín, N. Medrano, B. Calvo, and S. Celma, "Early detection and monitoring of forest fire with a wireless sensor network system," in *Procedia Engineering*, 2010, vol. 5, pp. 248–251.

[17] A. Mangla Amit Kumar Bindal Devendra Prasad, "DISASTER MANAGEMENT IN WIRELESS SENSOR NETWORKS: A SURVEY REPORT," *Int. J. Comput. Corp. Res. ISSN (Online*, vol. 6, pp. 2249–54, 2016.

[18] S. M. Brennan, A. M. Mielke, D. C. Torney, and A. B. Maccabe, "Radiation detection with distributed sensor networks," *Computer (Long. Beach. Calif).*, vol. 37, no. 8, pp. 57–59, Aug. 2004.

[19] M. Maroti, G. Simon, A. Ledeczi, and J. Sztipanovits, "Shooter localization in urban terrain," *Computer (Long. Beach. Calif).*, vol. 37, no. 8, pp. 60–61, Aug. 2004.

[20] K. Selvarajah, B. Arief, A. Tully, and P. Blythe, "Deploying wireless sensor devices in intelligent transportation system applications," in *inTech*, InTech, 2012, pp. 143–168.

[21] D. Zonta *et al.*, "Wireless sensor networks for permanent health monitoring of historic buildings," *Smart Struct. Syst.*, vol. 6, no. 5–6, pp. 595–618, 2010.

[22] B. Hemingway, W. Brunette, T. Anderl, and G. Borriello, "The flock: Mote sensors sing in undergraduate curriculum," *Computer (Long. Beach. Calif).*, vol. 37, no. 8, pp. 72–78, Aug. 2004.

[23] H. Kwon, V. Berisha, V. Atti, and A. Spanias, "Experiments with sensor motes and

Java-DSP," *IEEE Trans. Educ.*, vol. 52, no. 2, pp. 257–262, May 2009.

[24] Y. Verbelen, "Characterization of Self-Powered Autonomous Embedded Systems for Complementary Balanced Energy Harvesting, PhD Thesis," Vrije Universiteit Brussel, Faculty of Engineering, 2018.

[25] H. Yetgin, K. T. K. Cheung, M. El-Hajjar, and L. Hanzo, "A Survey of Network Lifetime Maximization Techniques in Wireless Sensor Networks," *IEEE Communications Surveys and Tutorials*, vol. 19, no. 2. pp. 828–854, 2017.

[26] N. Kushalnagar, G. Montenegro, and C. Schumacher, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals," 2007.

[27] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks," 2007.

[28] E. T. Winter *et al.*, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks."

[29] Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)," Jun. 2014.

[30] F. Corno, L. De Russis, and A. M. Roffarello, "A Semantic Web Approach to Simplifying Trigger-Action Programming in the IoT," *Computer (Long. Beach. Calif).*, vol. 50, no. 11, pp. 18–24, Nov. 2017.

[31] P. Levis *et al.*, "TinyOS: An Operating System for Wireless Sensor Networks," *Ambient Intell.*, vol. 8491, pp. 115–148, 2005.

[32] A. Dunkels, B. Grönvall, and T. Voigt, "Contiki - A lightweight and flexible operating system for tiny networked sensors," in *Proceedings - Conference on Local Computer Networks, LCN*, 2004, pp. 455–462.

[33] A. Ranjan, H. B. Sahu, and P. Misra, "A Survey Report on Operating Systems for Tiny Networked Sensors," *J. Adv. Res. Netw. Commun. Eng.*, vol. 1, no. 1, pp. 1–12, May 2014.

[34] E. Baccelli, O. Hahm, M. Wahlisch, M. Gunes, and T. Schmidt, "RIOT: One OS to Rule Them All in the IoT," no. December, pp. 1--16, 2012.

[35] E. Baccelli, O. Hahm, M. Gunes, M. Wahlisch, and T. Schmidt, "RIOT OS: Towards an OS for the Internet of Things," in *2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2013, pp. 79–80.

[36] "Sentilla programer's guide. Version 1.1.1, Sentilla corporation, 2008." .

[37] F. Aslam, C. Schindelhauer, G. Ernst, D. Spyra, J. Meyer, and M. Zalloom,

"Introducing TakaTuka: a Java virtualmachine for motes," in *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, 2008, p. 399.

[38] M. Kargahi, *Operating System Concepts, 8th Edition*. 2011.

[39] A. Dunkels, O. Schmidt, T. Voigt, and M. Ali, "Protothreads," in *Proceedings of the 4th international conference on Embedded networked sensor systems - SenSys '06*, 2006, p. 29.

[40] N. Tsiftes and A. Dunkels, "A database in every sensor," in *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems - SenSys '11*, 2011, p. 316.

[41] I. Khoufi, P. Minet, E. Livolant, and B. Rmili, "Building an IEEE 802.15.4e TSCH network," in *2016 IEEE 35th International Performance Computing and Communications Conference, IPCCC 2016*, 2017, pp. 1–2.

[42] J. Tiete *et al.*, "Low-overhead time synchronization for schedule-based multi-channel wireless sensor networks," in *2013 19th IEEE Workshop on Local & Metropolitan Area Networks (LANMAN)*, 2013, pp. 1–6.

[43] B. Lemmens, K. Steenhaut, P. Ruckebusch, I. Moerman, and A. Nowe, "Network-wide synchronization in Wireless Sensor Networks," in *2012 19th IEEE Symposium on Communications and Vehicular Technology in the Benelux (SCVT)*, 2012, pp. 1–5.

[44] 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," Vrije Universiteil Brussel, Faculty of Engineering, 2012.

[45] S. Duquennoy, A. Elsts, B. Al Nahas, and G. Oikonomou, "TSCH and 6TiSCH for Contiki: Challenges, Design and Evaluation," *Proc. Int. Conf. Distrib. Comput. Sens. Syst. (IEEE DCOSS)*, pp. 11–18, Jun. 2017.

[46] M. Buettner, G. V. Yee, E. Anderson, and R. Han, "X-MAC: a short preamble MAC protocol for duty-cycled wireless sensor networks," in *Proceedings of the 4th international conference on Embedded networked sensor systems - SenSys '06*, 2006, p. 307.

[47] A. Dunkels, "The ContikiMAC Radio Duty Cycling Protocol," *SICS Tech. Rep. T201113 , ISSN 1100-3154*, pp. 1–11, 2011.

[48] R. Musaloiu-E., C.-J. M. Liang, and A. Terzis, "Koala: Ultra-Low Power Data Retrieval in Wireless Sensor Networks," in *2008 International Conference on Information Processing in Sensor Networks (ipsn 2008)*, 2008, pp. 421–432.

[49] S. Khatarkar and R. Kamble, "Wireless Sensor Network MAC Protocol: SMAC &

TMAC.," *Indian J. Comput. Sci. ...*, vol. 4, no. 4, pp. 304–310, 2013.

[50] A. El-Hoiydi and J.-D. Decotignie, "WiseMAC: an ultra low power MAC protocol for the downlink of infrastructure wireless sensor networks," in *Proceedings. ISCC 2004. Ninth International Symposium on Computers And Communications (IEEE Cat. No.04TH8769)*, vol. 1, pp. 244–251.

[51] A. Burns and A. J. Wellings, *Real-time systems and programming languages*. Addison-Wesley, 2001.

[52] M. Michel, T. Voigt, L. Mottola, N. Tsiftes, and B. Quoitin, "Predictable MAC-level Performance in Low-power Wireless under Interference," in *EWSN '16 Proceedings of the 2016 International Conference on Embedded Wireless Systems and Networks*, 2016, pp. 13–22.

[53] International Telecommunication Union, "X.200: Data Networks and open system communications," vol. 4, p. 59, 1994.

[54] G. Z. Papadopoulos, K. Kritsis, A. Gallais, P. Chatzimisios, and T. Noel, "Performance evaluation methods in ad hoc and wireless sensor networks: A literature study," *IEEE Commun. Mag.*, vol. 54, no. 1, pp. 122–128, Jan. 2016.

[55] "REALWSN 2013 : The fifth Workshop on Real-World Wireless Sensor Networks." [Online]. Available: http://www.wikicfp.com/cfp/servlet/event.showcfp?eventid=29689&copyownerid=2. [Accessed: 05-Jun-2018].

[56] Zolertia, "Z1 Datasheet," pp. 1–20, 2010.

[57] P. Dutta and A. Dunkels, "Operating systems and network protocols for wireless sensor networks," *Philos. Trans. R. Soc. A Math. Phys. Eng. Sci.*, vol. 370, no. 1958, p. 68 LP-84, Jan. 2012.

[58] antenna-theory.com, "Friis Equation - (aka Friis Transmission Formula)," 2011. [Online]. Available: http://www.antenna-theory.com/basics/friis.php. [Accessed: 03-Jun-2018].

[59] L. Van Biesen, "Lecture notes on Physical Communication Vrije Universiteit Brussel Faculty of Engineering - Department of Electricity." Brussels, 2018.

[60] Texas Instruments, "Cc2420 Datasheet | Capacitor | Modulation." [Online]. Available: https://www.scribd.com/document/59211926/Cc2420-Datasheet. [Accessed: 03-Jun-2018].

[61] K. Srinivasan and P. Levis, "RSSI is Under Appreciated," *Proc. Third Work. Embed. Networked Sensors*, vol. 3031, p. 239242, 2006.

[62] M. M. Holland, R. G. Aures, and W. B. Heinzelman, “Experimental Investigation of Radio Performance in Wireless Sensor Networks,” *2006 2nd IEEE Work. Wirel. Mesh Networks*, pp. 5–7, 2006.

[63] T. Liu and A. E. Cerpa, “Foresee (4C): Wireless link prediction using link features,” in *Information Processing in 10th International Conference onSensor Networks (IPSN), 2011*, 2011, pp. 294–305.

[64] “Datasheet of Yageo SMD type antennas for BT/802.11b/g application.”

[65] A. Dunkels, F. Osterlind, N. Tsiftes, and Z. He, “Software-based on-line energy estimation for sensor nodes,” *Proc. 4th Work. Embed. networked sensors - EmNets ’07*, p. 28, 2007.

[66] P. Hurni, B. Nyffenegger, T. Braun, and A. Hergenroeder, “On the accuracy of software-based energy estimation techniques,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2011, vol. 6567 LNCS, pp. 49–64.

[67] A. Hergenroeder, J. Wilke, and D. Meier, “Distributed Energy Measurements in WSN Testbeds with a Sensor Node Management Device (SNMD),” *23rd Int. Conf. Archit. Comput. Syst.*, pp. 1–7, 2010.

[68] M. Buschhoff, C. Günter, and O. Spinczyk, “MIMOSA, a highly sensitive and accurate power measurement technique for low-power systems,” in *Lecture Notes in Electrical Engineering*, 2014, vol. 281 LNEE, pp. 139–151.

[69] U. P. Consumption *et al.*, “Datasheet: MSP430F261x MSP430F241x,” *Production*, no. December, 2011.

[70] N. T. Long, M. P. Uwase, J. Tiberghien, and K. Steenhaut, “QoS-aware cross-layer mechanism for multiple instances RPL,” in *International Conference on Advanced Technologies for Communications*, 2013, pp. 44–49.

[71] M. Buettner, G. V. Yee, E. Anderson, and R. Han, “X-MAC: a short preamble MAC protocol for duty-cycled wireless sensor networks,” in *Proceedings of the 4th international conference on Embedded networked sensor systems - SenSys ’06*, 2006, pp. 307–320.

[72] G. Werner-Allen, P. Swieskowski, and M. Welsh, “MoteLab: a wireless sensor network testbed,” in *IPSN 2005. Fourth International Symposium on Information Processing in Sensor Networks, 2005.*, pp. 483–488.

[73] V. Handziski, A. Köpke, A. Willig, and A. Wolisz, “TWIST: A Scalable and Reconfigurable Testbed for Wireless Indoor Experiments with Sensor Networks

Vlado," *Proc. Second Int. Work. Multi-hop ad hoc networks from theory to Real. - REALMAN '06*, no. May, p. 63, 2006.

[74] M. Doddavenkatappa, M. C. Chan, and A. L. Ananda, "Indriya: A low-cost, 3D wireless sensor network testbed," in *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering*, 2012, vol. 90 LNICST, pp. 302–316.

[75] C. Burin Des Rosiers *et al.*, "SensLAB Very Large Scale Open Wireless Sensor Network Testbed," *Proc. 7th Int. ICST Conf. Testbeds Res. Infrastructures Dev. Networks Communities*, 2011.

[76] R. Leone, J. Leguay, P. Medagliani, and ..., "MakeSense: Managing Reproducible WSNs Experiments," *Fifth Work. Real- ...*, 2013.

[77] G. Coulson *et al.*, "Flexible experimentation in wireless sensor networks," *Commun. ACM*, vol. 55, no. 1, p. 82, Jan. 2012.

[78] M. O. Farooq and T. Kunz, "Wireless Sensor Networks Testbeds and State-of-the-Art Multimedia Sensor Nodes," *Appl. Math. Inf. Sci.*, vol. 8, no. 3, pp. 935–940, 2014.

[79] J. Horneber and A. Hergenroder, "A survey on testbeds and experimentation environments for wireless sensor networks," *IEEE Commun. Surv. Tutorials*, vol. 16, no. 4, pp. 1820–1838, 2014.

[80] A. S. Tonneau, N. Mitton, and J. Vandaele, "How to choose an experimentation platform for wireless sensor networks? A survey on static and mobile wireless sensor network experimentation facilities," *Ad Hoc Networks*, vol. 30, no. C, pp. 115–127, Jul. 2015.

[81] M. Bezunartea, M. P. Uwase, J. Tiberghien, J. M. Dricot, and K. Steenhaut, "Demonstrating the versatility of a low cost measurement testbed for wireless sensor networks with a case study on radio duty cycling protocols," in *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST*, 2016, vol. 169, pp. 222–230.

[82] A. Support@energizer and Com, "Cylindrical Primary Lithium Handbook and Application Manual Lithium/Iron Disulfide (Li/FeS2) Introduction."

[83] "Everything You Know About Latency Is Wrong - DZone Performance." [Online]. Available: https://dzone.com/articles/everything-you-know-about-latency-is-wrong-brave-n. [Accessed: 04-Jun-2018].

[84] P. Ruckebusch, J. Devloo, D. Carels, E. De Poorter, and I. Moerman, "An evaluation of link estimation algorithms for RPL in dynamic wireless sensor networks," in

Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST, 2016, vol. 170, pp. 349–361.

[85] J. Beaudaux, A. Gallais, J. Montavont, T. Noel, D. Roth, and E. Valentin, "Thorough empirical analysis of X-MAC over a large scale internet of things testbed," *IEEE Sens. J.*, vol. 14, no. 2, pp. 383–392, Feb. 2014.

[86] F. Despaux, Y.-Q. Song, and A. Lahmadi, "Modelling and Performance Analysis of Wireless Sensor Networks Using Process Mining Techniques: ContikiMAC Use Case," in *2014 IEEE International Conference on Distributed Computing in Sensor Systems*, 2014, pp. 225–232.

[87] M. Stolikj, T. M. M. Meyfroyt, P. J. L. Cuijpers, and J. J. Lukkien, "Improving the Performance of Trickle-Based Data Dissemination in Low-Power Networks," in *Wireless Sensor Networks*, 2015, pp. 186–201.

[88] B. Pavkovic, A. Duda, W. J. Hwang, and F. Theoleyre, "Efficient topology construction for RPL over IEEE 802.15.4 in wireless sensor networks," *Ad Hoc Networks*, vol. 15, pp. 25–38, Apr. 2014.

[89] S. Duquennoy, B. Al Nahas, O. Landsiedel, and T. Watteyne, "Orchestra: Robust Mesh Networks Through Autonomously Scheduled TSCH," in *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems - SenSys '15*, 2015, pp. 337–350.