BACHELOR THESIS

RDC protocols in Wireless Sensor Networks running Contiki.

Maite Martincorena Arraiza.

Academic year 2014/2015.

Promoters:

Prof.dr.ir K. Steenhaut (VUB-INDI, VUB-ETRO)

Prof. Jacques Tiberghien (VUB-ETRO)

Supervisor:

Maite Bezunartea (VUB-ETRO)

This final project was realized within the framework of the Erasmus exchange programme between Vrije Universiteit Brussel (VUB) and Universidad Pública de Navarra (UPNA) and submitted in partial fulfilment of the requirements for the bachelor degree in Telecommunication Engineering.

Abstract

This thesis presents a comparison between the different radio duty cycling (RDC) protocols used in wireless sensor networks (WSN) running ContikiMAC as operating system. With that purpose, in order to obtain the most reliable results and avoid the inaccuracy associated with simulations, an experimental study was performed in a real wireless sensor network (WSN) set up using Zolertia Z1 motes. RDC protocols are one of the most important elements of a WSN configuration, being the ones in charge of managing the wake up intervals (WUI) of the radio in order to achieve low power usage, which is one of the main requirements in energy constrained devices such as WSN motes. Moreover, since the performance of the RDC protocols affects higher layers in the protocol stack, it is crucial to find the RDC configuration that gives the best performance, which is the main goal of this bachelor thesis.

Este trabajo fin de grado presenta una comparación entre los diferentes protocolos que definen el ciclo de trabajo de la radio (protocolos RDC), empleados en las redes de sensores inalámbricos que usan Contiki como sistema operativo. Con ese propósito, para obtener los resultados más fiables posibles y para evitar la imprecisión asociada a las simulaciones, se ha realizado un estudio experimental en una verdadera red de sensores inalámbricos usando módulos Zolertia Z1. Los protocolos RDC son uno de los elementos más importantes en la configuración de una red de sensores inalámbricos, ya que están a cargo de la gestión de los intervalos de actividad de la radio, con el fin de conseguir un bajo consumo de energía, el cual es uno de los requerimientos principales de aparatos con restricción energética como los sensores inalámbricos. Además, como el comportamiento de los protocolos RDC afecta a las capas superiores de la pila de protocolos, es crucial encontrar una configuración RDC que proporcione el mejor rendimiento posible, siendo éste el principal objetivo de este trabajo fin de grado.

Deze thesis maakt een vergelijking tussen de verschillende Radio Duty Cycling (RDC) protocollen, die gebruikt worden in draadloze sensornetwerken (wireless sensor networks, WSN) draaiend op het Contiki besturingssysteem. Om de meest betrouwbare resultaten te behalen en om de onnauwkeurigheden geassocieerd met simulaties te vermijden, werd een experimenteel onderzoek uitgevoerd in een echt netwerk, bestaande uit Zolertia Z1 knopen. RDC protocollen zijn één van de meest belangrijke elementen in de configuratie van een WSN, omdat ze de intervallen, waarop de radio ontwaakt, beheren. Dit heeft tot doel het energieverbruik te verlagen, wat één van de belangrijkste vereisten is in apparaten, die beperkt zijn op vlak van energie, zoals knopen in een WSN. Omdat de prestaties van de gebruikte RDC protocollen de hogere lagen in de stack beïnvloeden, is het zeer belangrijk om de best presterende configuratie van de RDC protocollen te vinden, wat dan ook de belangrijkste doelstelling van deze bachelor thesis is.

Key words

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

Acknowledgements

I would like to give a special thank you to all the people that have helped me in the fulfilment of this thesis.

First, I would like to thank all the people in the VUB-INDI and VUB-ETRO departments, for creating such a good environment to work in and for always being ready to help with any problem or doubt I could have had. I would like to thank Prof. Dr. Ir. Kris Steenhaut for offering me this topic that I find so interesting and challenging and for the warm welcome she gave me, always willing to answer all my questions and help with anything. I would also like to thank Maite Bezunartea so much for always having time to help me with infinite patience and good humour, as well as encouraging me when I felt overwhelmed. Without her help and support my journey to the completion of my thesis would have been much harder. A special thanks to Prof. Dr. Ir. Jacques Tiberghien for his complete dedication, for making monotonous tasks entertaining and interesting, and for his patience, kindness, help and lessons. Working with him I have learned a great deal not only about the topic of this thesis but also about the patience and perseverance you need to be a researcher. Without his help I would not have been able to finish this bachelor thesis.

I would like to thank Universidad Pública de Navarra (UPNA) and the ERASMUS program as well, for giving me the opportunity to make my bachelor thesis abroad with all the experiences and learnings that has given me.

A would also like to thank my friends in Brussels for always supporting me and keeping my spirits up when I most needed it. Finally, I would like to thank my parents and brother for their constant love, support, understanding and encouragement during my four years of university career. I owe them everything I had the opportunity to achieve and experience.

Contents

Figures

Tables

List of acronyms

6LoWPAN: IPv6 over Low power Wireless Personal Area Networks.
CCA: Clear Channel Assessment
CPU: Central Processing Unit
CSMA: Carrier Sense Multiple Access
CSMA/CA: Carrier Sense Multiple Access with Collision Avoidance
CSMA/CD: Carrier Sense Multiple Access with Collision Detection
DHCP: Dynamic Host Configuration Protocol
ETX: Expected Transmission Count
FIFO: First In, First Out
FTP: File Transfer Protocol
IEFT: Internet Engineering Task Force
IoT: Internet of Things
ISM: Industrial, Scientific and Medical
LR-WPAN: Low-Rate Wireless Personal Area Network
MAC: Medium Access Control
OF: Objective Function
PCB: Printed Circuit Board
PDR: Packet Delivery Ratio
RDC: Radio Duty Cycle
PHY: Physical layer
PPDU: PHY Protocol Data Unit
PSDU: PHY Service Data Unit
QFN: Quad Flat No-leads
RISC: Reduced Instruction Set Computer
RPL: Routing Protocol for Low-power and Lossy Networks
RSSI: Received Signal Strength Indication
UART: Universal Asynchronous Receiver-Transmitter
UIP: micro IP
USB: Universal Serial Bus
WPAN: Wireless Personal Area Network
WSN: Wireless Sensor Network
WUI: Wake Up Interval.

Chapter 0 – Introduction

0.1 Problem statement.

The “Smart Nets” research group, led by Kris Steenhaut, is dedicated to the study of the performance of wireless sensor networks, searching for and proposing new protocol designs and improvements, as well as debugging certain aspects of their functioning.

A lot of different applications are being developed nowadays in the field of wireless sensor networks. However, the good operation of these applications depends on the whole protocol stack and, currently the best physical and MAC layer configuration is still uncertain. Most importantly, a good performance of the RDC protocol in a WSN is crucial, as the nodes are very energy constrained and the RDC protocol is in charge of the wake up intervals of the radio and, therefore, the power usage.

0.2 Aims and objectives.

The aim of this bachelor thesis is to experimentally evaluate the performance of a wireless sensor network (WSN) running Contiki, using different RDC protocols and different configurations in order to find the best performance under given traffic conditions. To achieve this, several experiments were made in a real network, in order to obtain the most realistic and trustworthy results. This network consisted on a point to point link between two Zolertia Z1 motes and an observing network formed by three monitoring motes, one attached to each one of the two observed motes and a sink which collects the data from the other two. In these experiments, several parameters of each RDC protocol will be changed, to search for the best option, and the performance of the link will be measured by means of packet delivery ratio (PDR), packet latency and power usage.

0.3 Thesis structure.

Once the topic and main objectives of this thesis have been introduced, in Chapter 1 the background of this thesis will be presented, in order to acquire all the knowledge necessary in the topics addressed during the experimental work. This background contains a general

explanation of WSNs, including the protocols and standard used in them; a description of the Zolertia Z1 motes and the Contiki operating system.

All the experiments done with the intention to understand the performance of the different RDC protocols are included in Chapter 2. After explaining the experimental set up and evaluation techniques in the first two sections, the experiments are described in a chronological order. First of all, before immersing ourselves in the analysis of the different RDC protocols and in order to find the best conditions for these experiments, some problems are addressed: in section 2.3, a beforehand encountered problem with ContikiMAC and LPP is presented together with a possible solution; following this, in section 2.4, we include another problematic issue encountered during the experiments performed in section 2.3. After that, in section 2.5, a comparison between different RDC protocols in WSNs running Contiki is made for different levels of perturbation (perturbing traffic conditions). Together with this, to get a deeper understanding of RDC protocols, we make a comparison between three different wake up intervals in ContikiMAC.

Finally, in Chapter 3, the general conclusions and future work are presented.

Chapter 1 - Background.

1.1 Wireless sensor networks.

Nowadays there are multiple applications, such as remote environmental monitoring and target tracking, military and health applications that need to collect sensed data. With the development, in the recent years, of wireless sensors that are becoming smaller, cheaper and more intelligent, this task has become much simpler than in the past when human operators and wired sensors were used. The use of wireless sensors fixes the main problems and restrictions of former means of sensing the environment.

A wireless sensor network (WSN) [1], [2] is a set of autonomous, low-power devices, cooperating, communicating and reporting some sensed phenomenon to a supervisor. Each device is typically composed of a microcontroller, a memory, one or more sensors, a radio transceiver with an internal or external antenna and an energy source.

As they can be used for many different applications, some WSNs might require that the sensors operate unattended for a long time, due to the challenging location of the nodes in the network, while some others might demand the sensor to be able to process a big amount of data in a short period of time. This is the reason why the software and hardware to be used should be chosen very carefully. One choice to make is the RDC protocol, which controls the amount of time the radio is switched on, in order to maximize the lifetime of the batteries (see section 1.3.4.2). The operating system (OS) is another very important part of the software in a WSN.

The operating systems used in Wireless sensor networks [3] are very different from those used in computers or smart-phones. This difference is mainly due to the kind of hardware on which the OS is running. The microcontroller used as CPU in wireless sensor nodes is usually not very powerful, since one of their main goals is to achieve the minimum power consumption as possible.

Even though the operating systems used in Wireless Sensors nodes are much simpler than those used in computers, they still usually have to handle many different operations concurrently. Therefore, they need to have a scheduling system that shares the CPU resources between the different tasks, as a microcontroller can only execute one program at a time.

To summarize, the main requirements for an operating system in WSNs are:

- Limited resources: The hardware platforms in wireless sensor networks offer very limited resources in order to ensure low power consumption, low price and small size, so the operating system should use them efficiently.

- Concurrency: The operating system should be able to handle different tasks at the same time.
- Flexibility: Since the requirements for different applications vary wildly, the operating system should be able to be flexible to handle those.
- Low Power: Energy conservation should be one of the main goals of the operating system.

1.1.1 Protocol stack in WSNs.

The architecture for wireless sensor networks commonly follows the OSI model [4], but with the three upper layers combined into one application layer, leaving 5 layers: Physical layer, Data link layer, Network layer, Transport layer and Application layer. Through these layers, one can distinguish three functional planes: Power management, Mobility management and Task management. This architecture is shown in Figure 1.1.

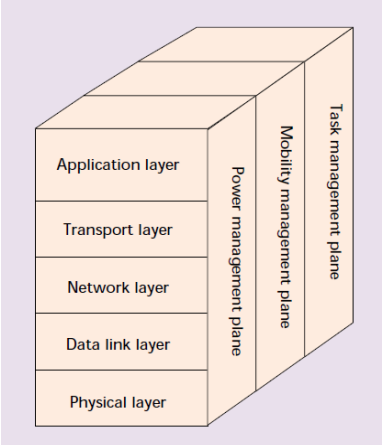


Figure 1.1: protocol stack in WSNs

OSI model layers in WSN.

- **Physical layer:** Is the layer responsible for signal transmission and reception over the selected physical communication medium by performing the frequency, carrier frequency and power selection, signal detection, modulation and data encryption. This layer's main priority in WSNs is low power consumption. Generally the minimum output power level necessary to transmit a signal over a distance of d is proportional to d^n, where $2 \leq n < 4$. For low-lying antennae and near-ground channels, typical in wireless sensor network communication, n is closer to 4.
- **Data link layer:** This layer focuses on the multiplexing of data streams, data frame detection and medium access control (MAC), and error control. One of the most significant functions of the data link layer is medium access control. A WSN must have a MAC protocol that must meet two goals: the first goal is to compose a network infrastructure by establishing communication links between several nodes and providing the network self-organizing capabilities. The second goal is to fairly and efficiently share communication resources between all the nodes in the network to achieve good network performance in terms of energy consumption, network throughput, and delivery latency.

- **Network layer:** It handles the routing of the sensed data from the sensors nodes to the sink or sinks, leading the process of selecting paths through which to send data in the network.
- **Transport layer:** This layer encompasses the different networks connecting the sensor with the application that handles the collected data. As the end-to-end communication schemes in sensor networks are not based on global addressing, new schemes that split the end-to-end communication (probably at the sinks) may be needed.
- **Application layer:** Is the layer responsible for traffic management and software provision for different applications that translate the data in an understandable way or send queries to obtain information.

1.1.2 WSN Standards.

As mentioned before, wireless sensors require low power consumption, which is the key requisite in the design of the communication standards used in wireless sensor networks. These standards define the protocols and functions to be used so that wireless sensor nodes can communicate with other networks. Some of the mentioned standards include IEEE 802.15.4, ZigBee, IETF 6LoWPAN, etc. However, in this thesis, we will focus on the lower layers of the protocol stack defined by IEEE 802.15.4.

1.1.1.1 IEEE 802.15.4

IEEE 802.15.4 [5] was developed in 2000 by the Task Group 4 under the IEEE 802 Working Group 15, with the aim to design a standard which provides low power consumption, low cost of deployment and little complexity. Nowadays, IEEE 802.15.4 is the standard proposed for low rate wireless personal area networks (LR-WPAN), specifying both physical and medium access control (MAC) layers. Geographically, IEEE 802.15.4 operates globally in the 2.4 GHz industrial, scientific and medical (ISM) band, but it also operates in the 868-868.6 band in Europe and in the 902-928 band in the USA as shown in Table 1.1.

Frequency	Channels	Region	Data Rate	Baud Rate
868-868.6 MHz	0	Europe	20 Kbit/s	20 KBaud
902-928 MHz	1-10	USA	40 Kbit/s	40 KBaud
2400-2483.5 MHz	11-26	Global	250 Kbit/s	62.5 KBaud

Table 1.1: IEEE 802.15.4 frequency bands

The IEEE 802.15.4 standard allows the network to implement the two topologies shown in Figure 1.2: the star topology, where the communication is performed between network devices and a single central controller; or the peer-to-peer topology, which allows the

implementation of much more complex network formations such as ad hoc or self-configuring networks.

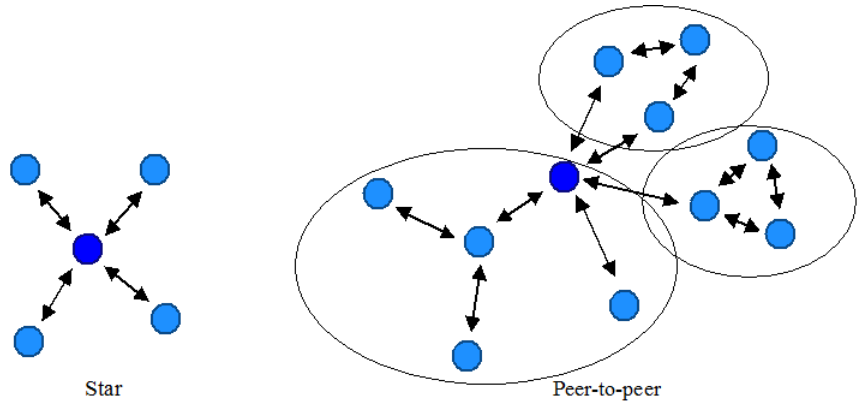


Figure 1.2: Topologies in IEEE 802.15.4

An IEEE 802.15.4 LR-WPAN device includes a physical (PHY) layer and a medium access control (MAC) layer that provides access to the physical channel for all types of transfer and ensures the reliable transfer of frames.

- **Physical (PHY) layer.** IEEE 802.15.4 supports two PHY layers, one for the 868/915 MHz band (or low-band) and the other for the 2.4 MHz band (or high-band). The characteristics of each band are shown in Table 1.1.

  Both PHY layers use the same packet structure, described in Table 1.2, making possible the definition of a common MAC interface. Each packet, called PHY protocol data unit (PPDU), includes a preamble, a start of packet delimiter, a packet length, and a payload field, or PHY service data unit (PSDU). The preamble of 32-bit is designed to enable the acquisition of symbol and chip timing. The IEEE 802.15.4 payload length can vary from 2 to 127 bytes.

PHY protocol data unit (PPDU)			
Preamble	Start of packet delimiter	Length field	PHY layer payload PHY service data unit (PSDU)
4 bytes	1 byte	1 byte	2-127 bytes

Table 1.2: IEEE 802.15.4 PHY layer packet structure

- **MAC layer.** The IEEE 802.15.4 medium access control (MAC) layer controls the access to the radio channel using the CSMA/CA mechanism. When a node wants to transmit, it

first listens to the medium in order to check if another node is transmitting at that moment. If there are no transmissions, the node transmits immediately; otherwise, the node postpones the transmission for a random amount of time and keeps on monitoring the channel until it is idle.

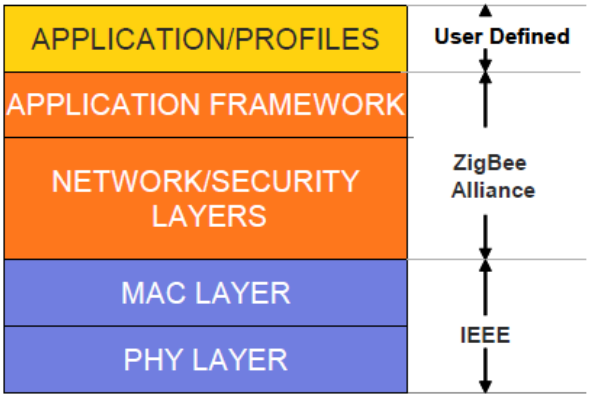


Figure 1.3: IEEE 802.15.4 protocol stack (Extracted from [6])

In the case of Contiki, this MAC layer is conceptually divided in three "sublayers": Framer, RDC and MAC, as it can be seen in Figure 1.4.

Figure 1.4: IEEE 802.15.4 protocol stack in ContikiOS (Extracted from http://anrg.usc.edu/contiki/index.php/MAC_protocols_in_ContikiOS)

The framer is not a typical layer implementation. Instead, it is a set of auxiliary functions that are called in order to create the data frame to be transmitted and to perform the parsing of data frames being received.

The RDC sublayer is the one responsible of the sleep periods of nodes, deciding when to send a packet, depending on the wake-up times of the receiver. There are several RDC protocols that can be implemented in wireless sensor networks, but in this thesis, the attention will be focused on the four protocols used in Contiki: ContikiMAC, LPP, CXMAC and NullRDC.

Finally, the MAC sublayer, as explained before, takes care of the retransmission of lost packets.

1.1.1.2 ZigBee.

ZigBee [6] is an industry alliance to develop the upper layers of interoperable WSN applications over IEEE 802.15.4 (see Figure 1.3). It emphasizes low cost and low power consumption, targeting battery life of many years.

There are three kinds of ZigBee devices: ZigBee coordinator, ZigBee router and ZigBee end device. The ZigBee coordinator is the one that initiates the network formation, stores information, and is able to bridge networks together. ZigBee routers connect groups of devices with each other and provide multi-hop communication across devices. ZigBee end devices are the sensors, actuators, and controllers that collect data and only communicate with the router.

1.1.1.3 6LoWPAN.

Direct communication with traditional IP networks requires many protocols, which often demands an operating system to handle the complexity and maintainability of these protocols. This is the reason why, up until now, Internet of Things (IoT) has only been within reach of devices with a powerful processor, an operating system with a full TCP/IP stack and an IP-capable communication link. As wireless sensors do not meet those requirements, the IETF 6LoWPAN working group was created, which, nowadays, develops and maintains all core Internet standards and architecture work, to enable IPv6 to be used with wireless embedded devices and networks. This way 6LoWPAN was created.

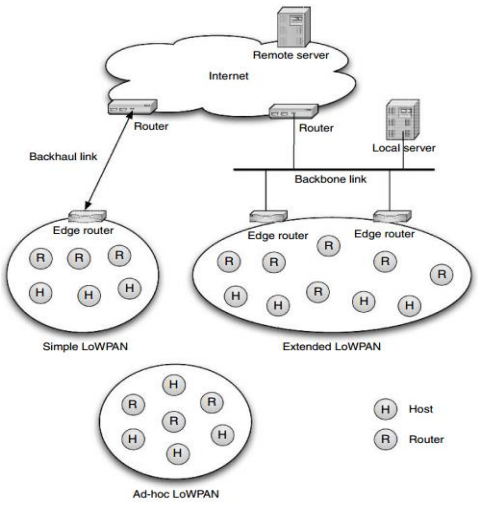


Figure 1.5: 6LoWPAN architecture (Extracted from [26])

6LoWPAN standards [7] enable the efficient use of IPv6 over low-power, low-rate wireless sensor networks on simple embedded devices through an adaptation layer and the optimization of related protocols.

The 6LoWPAN architecture, shown in Figure 1.5, is formed by three different types of IPv6 subnetworks called low-power wireless personal area networks (LoWPANs): simple LoWPANs, extended LoWPANs, and Ad hoc LoWPANs. A LoWPAN is the group of 6LoWPAN nodes which share a common IPv6 address prefix (the first 64 bits of an IPv6 address), meaning that no matter where a node is in a LoWPAN its IPv6 address remains unchanged. LoWPANs are connected to other IP networks through edge routers, which route traffic in and out of the LoWPAN and, at the same time handle 6LoWPAN compression and Neighbour Discovery.

1.2 Zolertia Z1 motes

There are several WSN platforms available at the moment, such as Sky and Sentilla, but the one that will be used in this thesis is Zolertia.

The Z1 module from Zolertia [8], shown in Figure 1.6 is a general purpose development platform for WSNs. It is equipped with two on board sensors: a digital programmable accelerometer (ADXL345) and a programmable temperature sensor (TMP102). It is also compatible with other analogue and digital sensors.

The Z1 module is equipped with a second generation MSP430F2617 low power microcontroller, which features a powerful 16-bit RISC CPU at 16MHz clock speed, built-in clock factory calibration, 8KB RAM and a 92KB Flash memory. For radio communication, Z1 motes come with an integrated ceramic antenna from Yageo/Phycomp connected to the CC2420 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver or an external λ/2 antenna. The CC2420 transceiver has an effective data rate of 250Kbps.

The CC2420 RF transceiver [9] is a low-cost, highly integrated solution for robust wireless communication in the 2.4 GHz unlicensed ISM band. It follows worldwide regulations covered by ETSI EN 300 328 and EN 300 440 class 2 (Europe), FCC CFR47 Part 15 (US) and ARIB STD-T66 (Japan). It is designed for low power and low voltage wireless applications and it provides extensive hardware support for packet handling, data buffering, burst transmissions, data encryption, data authentication, clear channel assessment, link quality indication and packet timing information. These features reduce the load on the host controller and allow CC2420 to interface low-cost microcontrollers. The transmitter has an adjustable output power of maximum 1mW (0dBm) and the receiver a sensitivity that goes down to -93dBm.

Figure 1.6: Zolertia Z1 mote

When it comes to programming, Zolertia Z1 motes do not require any external hardware, since built-in full USB capability allows quick developing of WSN applications and fast integration with multiple systems.

The CP2102 is a highly-integrated USB-to-UART Bridge Controller providing a simple solution for updating RS-232 designs to USB using a minimum of components and PCB space. The CP2102 includes a USB 2.0 full-speed function controller, USB transceiver, oscillator, EEPROM, and asynchronous serial data bus (UART) with full modem control signals in a compact 5 x 5 mm QFN-28 package. No other external USB components are required.

The ADXL345 is a small, thin, low power, 3-axis accelerometer with high resolution (13-bit) measurement at up to ±16 g. Digital output data is formatted as 16-bit twos complement and is accessible through either a SPI (3- or 4-wire) or I2C digital interface.

The TMP102 is ideal for extended temperature measurement in a variety of communication, computer, consumer, environmental, industrial, and instrumentation applications. The device is specified for operation over a temperature range of -40°C to +125°C.

As operating system, Z1 motes currently support TinyOS and Contiki. But in this thesis only Contiki will be used.

1.2 Contiki.

1.3.1 Introduction

When considering a wireless sensor network, the application should be the main goal and most of the resources for development should be dedicated to it. However, the implementation of almost any application requires a lot of software that is not specific to any particular application and that requires a lot of programming and debugging efforts. For this reason, most wireless sensor network developers prefer to find means to share these efforts among many applications. A first possible approach consists in buying such software, typically from companies that sell so called “real time operating systems”. These systems are however targeted at hardware with much more resources 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 [10], initially developed at the University of California at Berkeley by David Culler et al. as early as 1999. Contiki [11], 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).

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

1.3.2 Overall organization of Contiki.

Basically, Contiki is a huge collection of macros written in plain C and a set of makefiles that allow assembling 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 more than 10 years of additions of new functionalities. However, as most of the work reported in this thesis has been done with version 2.6, the older structure will be described here. At the top level of the Contiki directory one finds following directories:

- **platform**: this directory contains mainly specific sub-directories associated with 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, choosing 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 behaviour of the simulated system cannot be explored by simulation.

- **cpu**: The sub-directories of this directory contain, for the different CPUs, 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 multi-hop 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, webbrowsers and webservers can also be considered part of this first category. In the much more recent second and third categories, one finds implementations of current developments in wireless sensor networks such as antelope [12] (a database) and erbium [13] (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 behaviour of a 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.

1.3.3 Event management in Contiki.

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

1.3.3.1 Concurrent processes.

The most common of these abstractions is the sequential process [14]: 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 1.7)

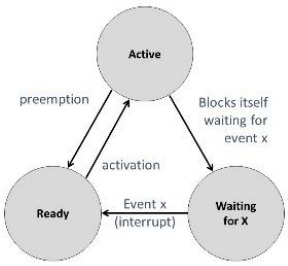


Figure 1.7: Process states

In the Active state, the process is executing its sequential code, which can eventually contain an explicit wait statement, requiring that the process waits until 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. 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.

1.3.3.2 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.

1.3.3.3 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 accordingly its state. 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 applications programmer point of view, the finite state machine approach has some severe drawbacks: the different logical threads of the application share the same code space and no syntactical rules prevent mixing them up and, as no explicit blocking wait statement exists, 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 programming.

1.3.3.4 Protothreads.

Contiki offers a new programming abstraction, called protothreads [15] 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 therefor 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 [16].

1.3.4 Communications facilities in Contiki.

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

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

1.3.4.1 The physical (PHY) layer

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

1.3.4.2 The Radio Duty Cycling (RDC) layer

To save energy in WSNs it is necessary to limit the time the radio is on as this is the most energy-consuming part of a mote. The problem resides mostly with the receiver as the transmitter can be switched on whenever something needs to be transmitted while the receiver, not knowing when a message is arriving, should continuously be kept on [8]. Instead, the receiver is kept in a low power sleeping mode most of the time but awakes periodically. RDC protocols try to control the timing of the transmissions so that they take place when the receiver is awake. Two approaches are commonly used: the synchronous and the asynchronous approach. With synchronous protocols sender and receiver keep synchronized real time clocks and time slots for communication are predefined. This requires, however, to keep clocks synchronized, which is a complex task that can cost a non-negligible amount of energy [17], [18]. With asynchronous protocols, a sender initiates a simple synchronization protocol for each message to transmit. Some protocols are hybrids as they start in an asynchronous way but keep timing information about successful transmissions so that subsequent transmissions will require less synchronization overhead.

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

1.3.4.2.1 Contikimac (CM)

Contikimac is the RDC protocol invented by the main authors of Contiki [19]. Its functioning is summarized in Figure 1.8.

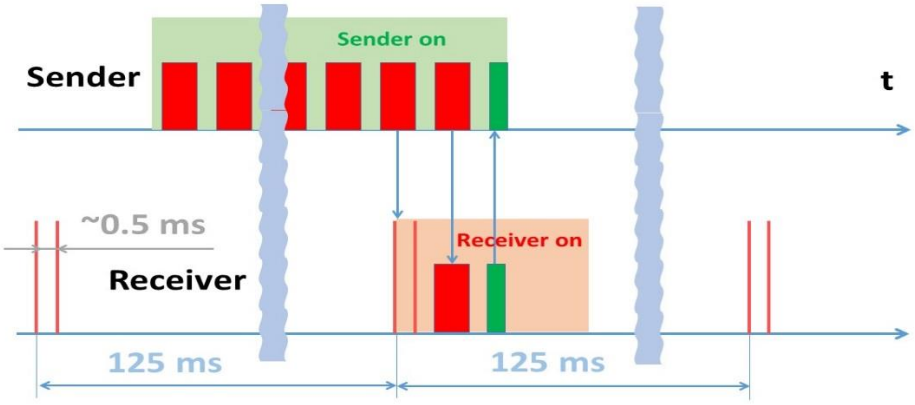


Figure 1.8: Principle of operation of ContikiMAC.

Most of the time the receiver sleeps, but a few times per second the receiver wakes up to perform two consecutive Clear Channel Assessments (CCA). The time interval between two consecutive CCA pairs is called the “Wake Up Interval” (WUI) and is set to 125 ms by default. If a pair of CCAs observes no radio-activity above a predetermined threshold the receiver returns to sleep.

The sender, when it has a frame to transmit, does it repeatedly, for a duration longer than the WUI.

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

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

When a sender receives an ACK, it knows that the receiver was awake just before the transmission of the acknowledged frame. As the wake-ups occur strictly periodically with a period of WUI, the sender can set up a table giving for each destination the optimal time to start transmitting a frame (phase locking the sender and receiver). This is represented schematically in Figure 1.9. One can observe that, once the phase lock has been established, instead of retransmitting a large number of times each frame, the sender has to retransmit

them only a few times as retransmission starts only when the receiver is going to wake up. By updating the timing when receiving each ACK, the phase lock can continue to operate over a long period, even if the sender and receiver clocks have slightly different frequencies.

When a sender repeatedly does not receive an ACK, it resumes sending frames for the whole duration of the WUI in order to re-establish a phase lock. (this is not shown in Figure 1.9)

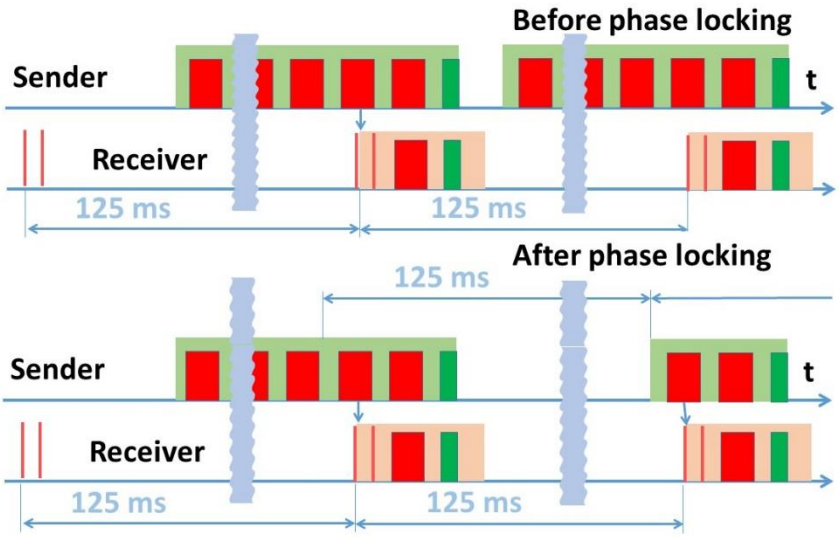


Figure 1.9: The benefits of phase locking in ContikiMAC

1.3.4.2.2 Low Power Probing (LPP)

Low Power Probing is a RDC protocol where the receiver announces by a broadcast when it is awake. A sender that has a frame to transmit switches on its receiver until it hears that the receiver is awake [20]. Operation of LPP is summarized in Figure 1.10.

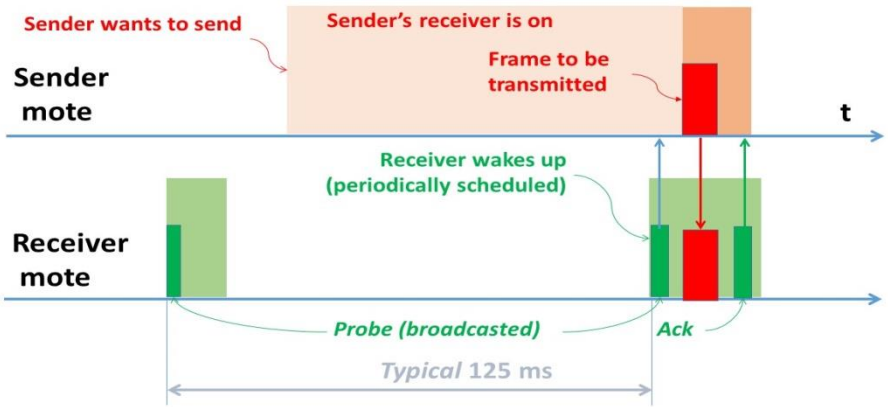


Figure 1.10: The Low Power Probing RDC protocol

The original description of LPP did not include any phase locking, but the Contiki implementation of the algorithm has provisions for it, although, they are disabled by default. Keeping track when a specified receiver wakes up allows a sender to reduce considerably the time it has to listen for a wake-up broadcast from that specified receiver.

1.3.4.2.3 The XMAC protocol.

This protocol was originally described in [21] and is the most commonly used RDC protocol in TinyOS.

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

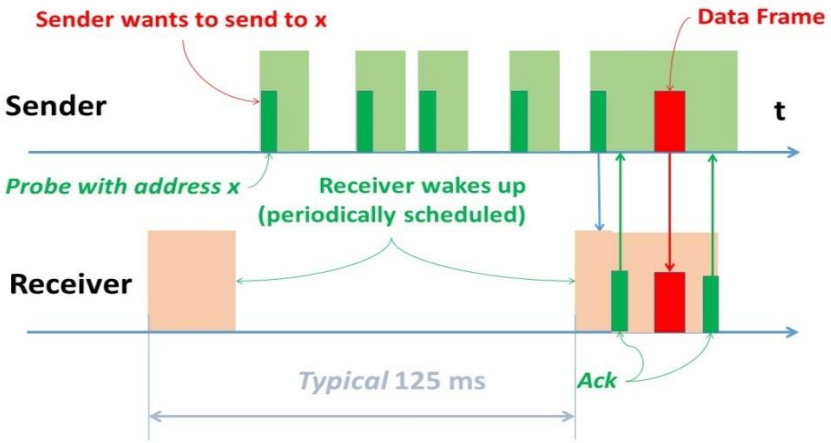


Figure 1.11: The XMAC RDC protocol

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

1.3.4.2.4 NullRDC.

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

1.3.4.2 The Medium Access Control (MAC) layer.

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

1.3.4.3.1 CSMA

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

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

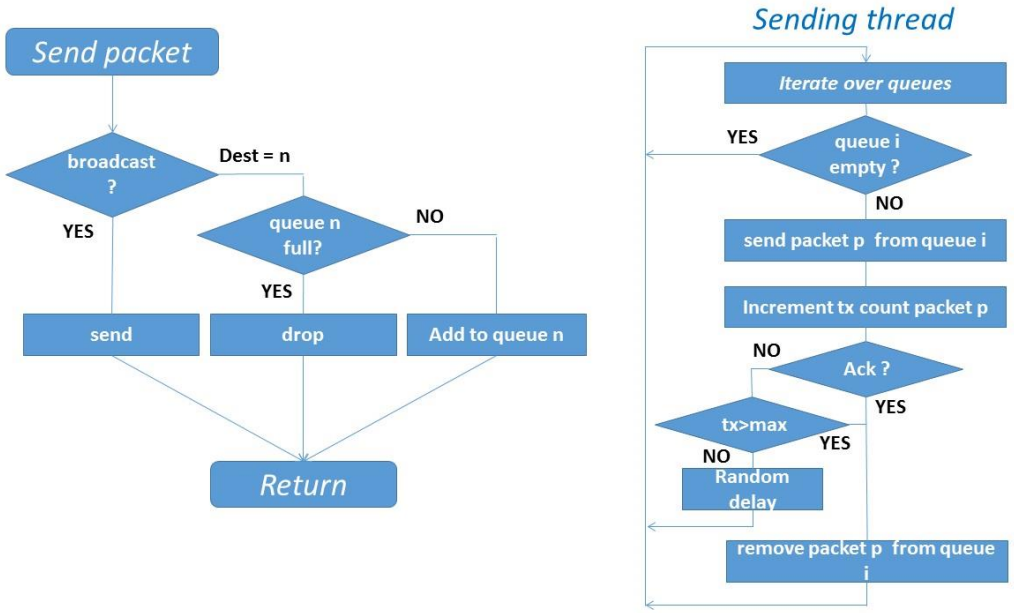


Figure 1.12: Simplified Control-flow chart of the Contiki implementation of the CSMA MAC protocol.

1.3.4.3.2 NullMAC.

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

1.3.4.4 The routing and application layers.

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

1.3.4.4.1 Rime.

A comprehensive description of the objectives and the underlying software architecture of Rime is given in [22]. The authors describe the functionality of Rime as follows: “The Rime protocol stack provides a set of communication primitives, ranging from best-effort local

neighbour broadcast and reliable local neighbour unicast, to best-effort network flooding and hop-by-hop reliable multi-hop unicast. Applications or protocols running on top of the Rime stack may use one or more of the communication primitives provided by the Rime stack".

1.3.4.4.2 uIP.

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

- First, the addressing space: as WSNs and the IoT are conceived for large numbers of devices, the almost exhausted IPv4 address space is obviously inadequate. UIP uses the IPv6 addressing schemes.

- Next, the internet protocols optimize throughput and responsiveness, at the cost of memory and energy usage, while WSNs and the IoT seldom have significant throughput but have often severe hardware restrictions. Even the smallest IPv6 packets are larger than the maximum size IEEE 802.15.4 frames. Header compression and packet fragmentation have to be added to the traditional TCP/IP stack. 6LoWPAN is a sub-layer designed to be inserted between the MAC layer and the IP layer to satisfy these needs. It is implemented in Contiki under the name **sicslowpan**.

- Finally, it seems logical to consider each WSN and each physically integrated part of the IoT as a subnet in the Internet addressing scheme, but most internet protocols suppose that every subnet is a single broadcast domain, while communication between motes is generally based upon multi-hop links. A routing protocol with suitable algorithms applicable to a single multi-hop subnet is needed. RPL has been designed for that purpose by the Internet Engineering Task Force [23] and is implemented in Contiki.

RPL is a fairly sophisticated variety of the Distance Vector routing algorithm. It leaves to the user (the application implementer) the choice of the function that defines the cost of each link and each node. This is the so called "Objective Function" (OF). In the Contiki implementation of RPL, an OF based upon ETX [24] parameters of the different links is available. The ETX value of a link is obtained from the CSMA MAC layer which keeps track of the average number of attempts that are needed to successfully transfer a packet over each link.

Chapter 2 - Experimental evaluation of the performance of WSN running Contiki.

2.1 Experimental setup.

To analyse the performance of a unicast link between two Zolertia motes, a small testbed was built in a garden in order to avoid interferences. This setup consists of two WSNs: the first one, the observed network, runs the protocols and applications under evaluation; the second one, the observing network, monitors the first one and transmits the monitored information to a sink node which records that information. Figure 2.1 shows one of the dual motes that were used in the experiments. This dual mote is composed of two motes disposed inside plastic boxes and connected to each other. The black box contains the observed Z1 mote which has a built in ceramic antenna that limits its radio range to a few meters. The white box contains the observing Z1 mote equipped with an external antenna which has a bigger radio range. In order to avoid interferences between these two networks they will work in different channels: channel 26 for the observed network, being the channel least affected by interferences, and channel 16 for the observing network.

To simplify, for now on, we will refer to the observed network as the black network and to the observing network as the white network.

Figure 2.1: Zolertia Z1 dual mote in the garden

The black sender will send packets to the black receiver, first at a constant rate of one packet per second and then at a random rate between 10 and 1990ms with a mean of one second between packets.

Since in real applications of wireless sensor networks the network is usually exposed to additional traffic from other devices outside the network, two other motes where added to the setup in order to generate this additional traffic. To simulate the perturbing effect of a network where each mote is reachable from more than two others, the additional sender, when active, sends twice more messages than the sender under evaluation. To produce light additional traffic, both extra motes where switched on so that the sender will send packets to the receiver and receive the corresponding ACKs. Heavier additional traffic was obtained by switching off the receiver, so that the sender will continuously send messages to the non-replying receiver. The whole experimental setup for unicast transmission test is shown in Figure 2.2.

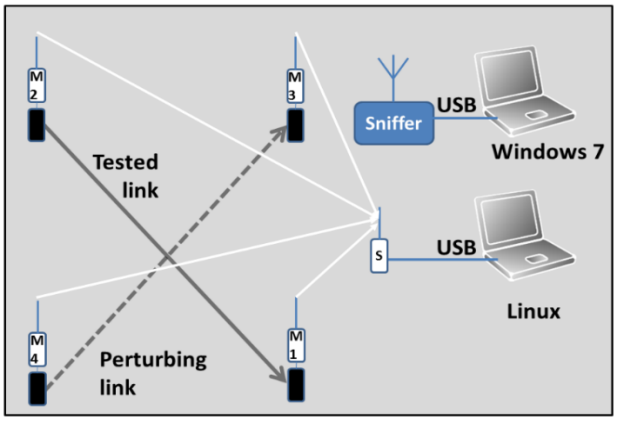


Figure 2.2: Experimental setup for unicast transmission with perturbing traffic

General operation:

The nodes of the black network send short messages to each other. In this case, the software used is RRPsender and RRPreceiver in *contiki-2.6/examples/z1.* These messages are uniquely identified by the address of the node where they were created and a local sequence number generated by this same node. In each dual mote, the black and white motes are interconnected via hardware. When the application program of the black mote sends or receives a message, it reports the sequence number of this packet to the observer application running on the white mote. Then, this application in the white mote sends a packet to the white sink, containing the sequence number together with the power used by the black node since the last packet was sent or received. A computer connected to the sink stores all the data from the white sink, which contains all the packets with the address of the sender and their timestamp according to the clock in the sink. This data, supposing that, ideally, no packets are lost in the observing network, provides an inventory of the packets

transmitted and received by the applications running in the observed network. From this inventory it is simple to compute the packet delivery ratio (PDR), packet latency and power usage in order to analyse the performance of the network. To obtain trustworthy and statistically valid results, more than 1000 packets where recorded for each experiment.

Figure 2.3 shows a screenshot of part of a serialdump showing the data provided by each packet.

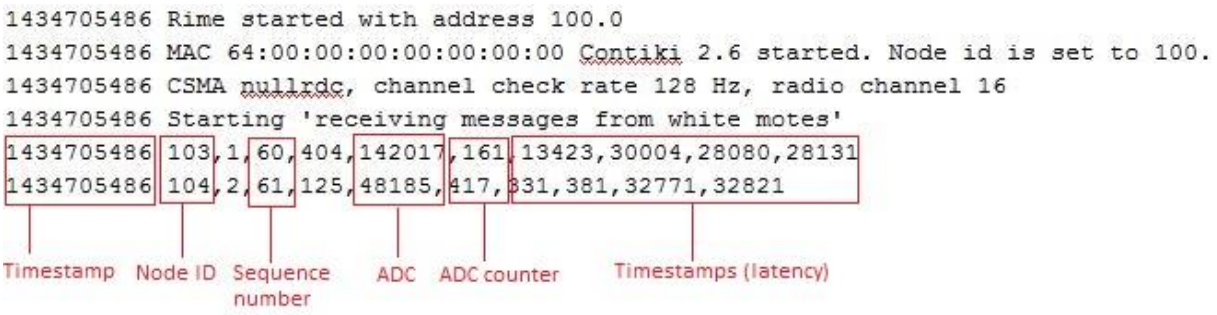


Figure 2.3: Example of a serialdump

The parts indicated are the ones used to compute the PDR, packet latency and power usage. The first timestamp corresponds to the instant the packet arrives in the sink, measured in seconds since 01/01/1980; The node ID univocally identifies the node, while the sequence number identifies the packet; ADC and ADC counter are the measures necessary to calculate the power usage and the last four values correspond to the timestamps used to compute the latency. They refer to the local real time clock which has a period of 1/32768 s or approximately 30.5 microseconds. The role of these timestamps and ADC values will be described in more depth in section 2.2 in which the evaluation techniques are described.

Link between the black and white motes.

Data transmission between two motes can be done in two ways: through the serial USB port or using some of the parallel GPIO pins available on the z1 motes. Since transmitting data through the serial USB port takes about 80µs per packet, which will affect the normal operation of the network, the second option was chosen. As the GPIO pins 1.0, 1.6, 1.7, 2.3, 4.0, 4.2 and 4.3 are not used by any of the built in features of the Zolertia z1 motes, those were the ones chosen to form the link between both nodes. These seven pins of each mote were connected directly with each other. Pin 1.0 is used as clock to trigger the data transfers. As uncertainties about the sequencing of received packets exceeding 64 are not expected, the 6 remaining pins are used for carrying the 6 least significant bits of the sequence numbers.

2.2 Evaluation criteria.

In order to experimentally evaluate the performance of a wireless sensor network (WSN) running Contiki and using different RDC protocols, it is necessary to minimize the perturbations induced by the observing process. One possible way to do it would be to use Contiki's simulator COOJA, but some simplifications inherent to simulations limit the accuracy of the results when studying low level protocols. For instance, COOJA uses a common clock for all simulated motes, which hides possible consequences of clock drifts between motes, and its quite simple definitions of simulated radio ranges hide the complexity of actual radio propagation. Another more accurate and low cost way to achieve low perturbations is to perform the necessary measurements in an open field in the countryside (in this case the Ardennes), where there are almost no disturbing radio transmitters.

In this thesis three main evaluation criteria will be used: Packet Delivery Ratio (PDR), packet latency and power usage.

2.2.1 Packet Delivery Ratio (PDR).

It is the percentage of correctly received packets. To obtain this ratio the sender includes a sequence number in each packet it sends and the receivers keeps count of the number of received packets. This way, the PDR is obtained dividing the number of received packets by the number of sent packets, derived from the sequence numbers.

2.2.2 Packet latency.

Latency is a measurement of the time it takes a packet to go from the sender to the receiver, done at application level. As it can be seen in Figure 2.4 latency is composed of three parts:

1. The first and largest part is the time the packet spends in the sender after it is transmitted from the sender's application layer until it reaches the MAC/RDC layer. Here, the packet is sent repeatedly until an acknowledgement is received from the receiver. We will call this interval "sender latency" (tsr-tsa).

2. Then, the packet is transmitted through the air via radio. In most cases, the distances between motes are small compared to the speed of light, so this part of the latency can be neglected for all single-hop radio links.

3. When the message is received in the receiver's radio, it is decoded and analysed to check its correctness and destination and then sent to the application layer. This time interval will be called "receiver latency" (tra-trr). This interval depends mainly on the length of the packet and the data rate.

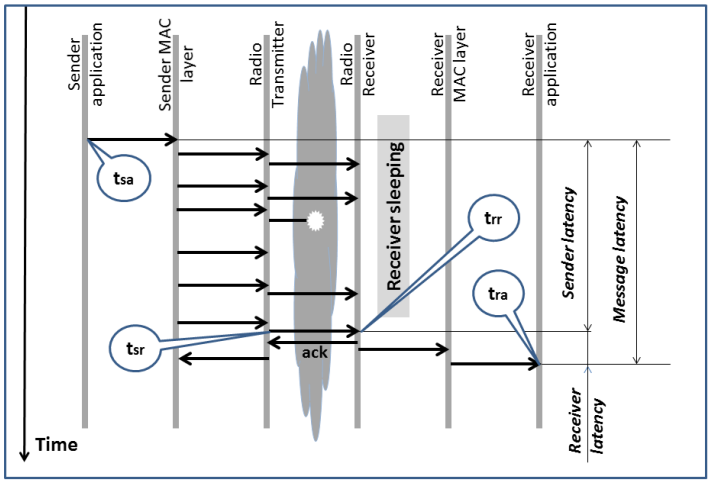


Figure 2.4: Packet latency.

There are several ways to measure the packet latency. These methods should not influence the behaviour of the black motes, so the measurement of the latency should be done by the white motes. In the ideal situation where the clocks in the sender and the receiver are synchronized, the packet latency can be obtained by computing the difference between the instant the receiver receives the packet and the instant the sender transmits it, which can be done using the timestamp included in each packet. This is the case in the Cooja simulator, which uses a common clock for all the motes in a simulation. But in a real life set up each node in a wireless sensor network has a different notion of time based on a clock situated in the node's hardware.

One way to measure the latency in the dual network is to record timestamps at different moments of the packet's journey from sender to receiver, taking also into account the delays in the white network. These timestamps (see Figure 2.5) are stored and sent in the packets generated by the white motes, using the real time clock in each mote:

- The moment the packet was generated at the application level in the black sender recorded by the white sender attached to it, tsa.
- The moment the packet was successfully sent by the white sender at the radio level, tsr.
- The moment the packet was received at the application level in the black receiver recorded by the white sender attached to it, tra.
- The moment the packet was successfully sent by the white sender attached to the black receiver at the radio level, trr.

Once the packets from the white observers have been successfully sent, the white sink records the following timings:

- Tsor: moment at which the packet containing the information related to the black sender arrives at the radio level in the white sink.
- Tsoa: moment at which the packet containing the information related to the black sender arrives at the application level in the white sink.
- Tror: moment at which the packet containing the information related to the black receiver arrives at the radio level in the white sink.
- Troa: moment at which the packet containing the information related to the black receiver arrives at the application level in the white sink.

For example, once the packet has reached the white sink, we can compute the sender latency by subtracting the time the packet was passed to the sender's MAC layer (tsa) from the time it was transmitted (tsr). As these two timestamps are taken from the same clock, there is no need for clock synchronization.

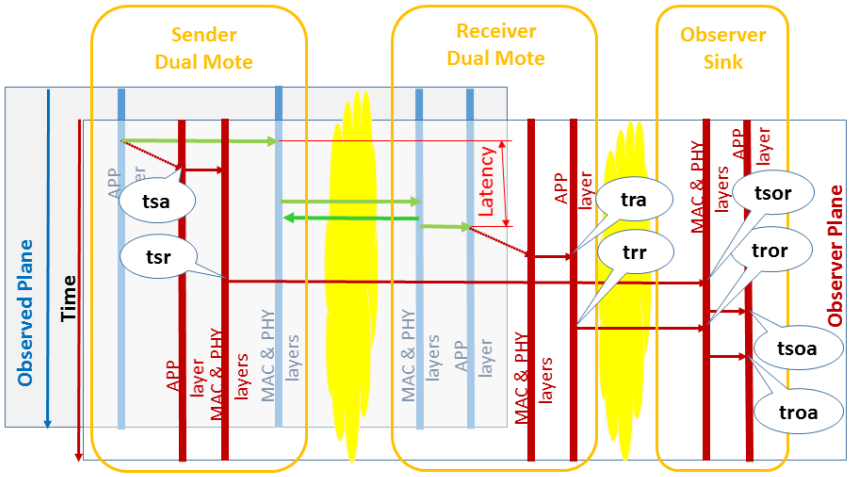


Figure 2.5: Packet latency in the dual network

With this method, the latency can be measured using the following formula:

Latency = (tsoa-troa)-(troa-tror)-(trr-tra)+(tsoa-tsor)+(tsr-tsa)

The location in the serialdump of each timestamp used to obtain the latency as described in the previous formula is shown in Figure 2.6. One should not forget that all timestamps are 16 bit unsigned integers, so that modulo 65536 arithmetic should be used for calculating the packet latency. This remains simple as long as none of the computed differences exceeds 2 seconds, which should be the case in our experiments.

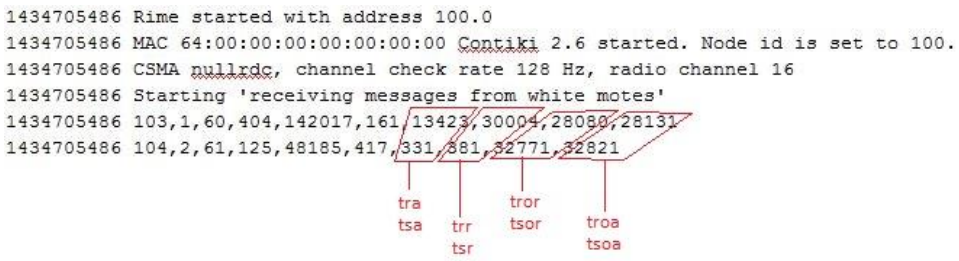


Figure 2.6: Latency timestamps in a serialdump

2.2.3 Power usage.

A team on the Swedish Institute of Computer Science lead by Adam Dunkels came up with a software-based method to measure the power used by the sender and receiver motes in a WSN [25].

Basically, they record how long each component of the sensor is on (CPU, LPM, Tx, Rx). Multiplying these times by the power required by the different components of a mote, one could compute the energy required for a given task. However, this method relies on theoretical values of required power, obtained from the datasheet of each component. Experiments done last year in our department [26] show that power figures obtained that way were highly inaccurate. Therefore we decided to measure the value for the average current absorbed by the black motes. This means that, in this thesis the power will be measured by hardware instead of software. To achieve that, we used the 12-bit analogue to digital converter (ADC) built in the white motes. By means of 1ohm series resistor and an instrumentation amplifier, a voltage proportional to the current taken up by the black mote is measured by the ADC. An ADC reading of 4095 corresponds to a current of approximately 47mA. The ADC is sampled with a frequency of 100Hz. The readings are recorded and the cumulative value between packets is sent in the white packets, together with the number of ADC readings. The location of these values in the serialdump is shown in Figure 2.3.

The average power used by a mote can be calculated with the following expression,

$$Power(W) = \frac{(\sum ADC) \times 11.5 \cdot 10^{-6} \times 3\text{V}}{N_{ADC}}$$

where the coefficient $11.5 \cdot 10^{-6}$ was obtained experimentally by members of our team, N_{ADC} is the total number of samples, $\sum ADC$ is the cumulative power value between packets and the 3V corresponds to the feeding source.

2.3 Bug in ContikiMAC8 and LPP8.

In previous experiments performed at ETRO department in the VUB, a bug was detected in ContikiMAC and LPP8 protocols when analysing the resulting PDR. As it can be seen in Figure 2.7, while NullRDC and CXMAC behave normally, in LPP8, each packet is sent 3 times, which is the default number of attempts to send a message when not receiving an ACK and, ContikiMAC shows low and unstable PDR for low RSSI values. In ContikiMAC we see no duplicates since it has a code to discard them. This shows that, for some reason, in both cases there are some ACKs lost or ignored, which can be observed by a sniffer.

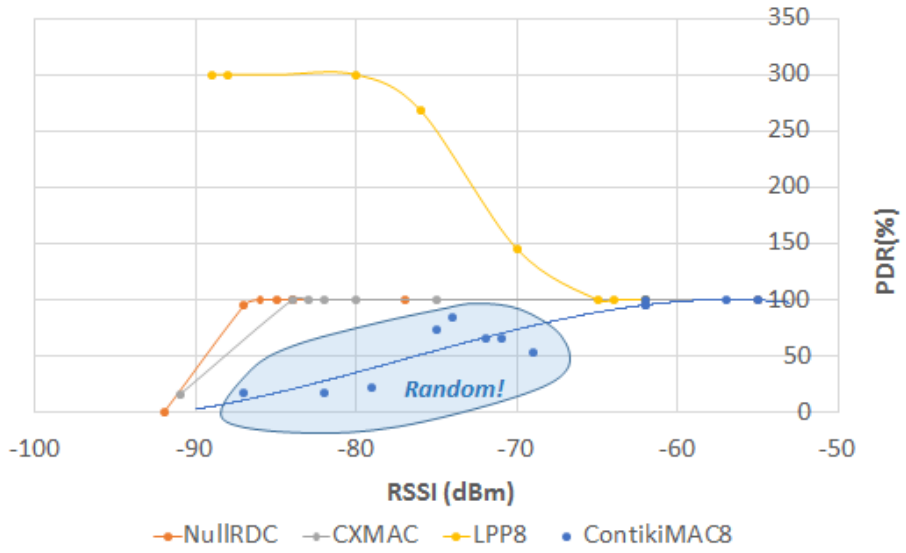


Figure 2.7: bug in ContikiMAC8 and LPP8

It was discovered that the problem was the amount of time the sender waits before checking for an ACK after sending a packet.

The three ored Boolean functions in charge of the detection of an acknowledgement (see Figure 2.8) are:

- channel_clear: It is based upon the receiver cca function.
- receiving_packet: It detects the start of frame byte and remains high until the end of the frame.
- pending_packet: it detects a frame with the adequate destination address and remains high until the length of the received packet exceeds a predefined threshold or the end of the frame is received.

Channel_clear only detects packets with RSSIs higher than -77dBm, while receiving_packet and pending_packet detect packets with RSSIs higher than -93dBm, which is the limit of the radio in the Zolertia Z1 motes.

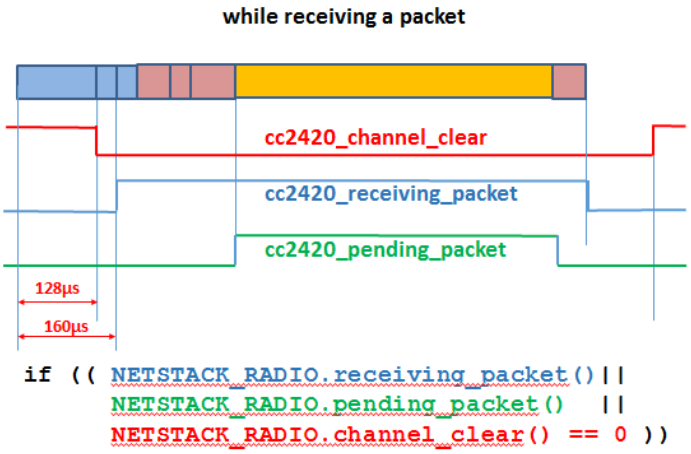


Figure 2.8: functions used for checking for ACKs

Some experiments where done to see when the checking for an ACK was performed and which of the previous functions was in charge of detecting the ACK. These experiments demonstrated that channel_clear was the function being used, which was the source of the problem since, as mentioned before, this function does not detect packets with RSSIs lower than -77 dBm. The best way to fix this problem would consist in using receiving_packet function for checking for an ACK rather than the channel clear function. Indeed the receiving packet function is independent of the signal level. This requires that checking for an acknowledgment should be done at least 352 us later after the end of transmission.

Knowing that, the solution for LPP8 was as simple as delaying the moment of checking for an acknowledgement. Packet duplication was, indeed, caused by the MAC protocol that requests retransmission of unacknowledged packets. The solution for ContikiMAC was not that simple because that extra time would not only delay the moment of the check for an ACK, but also enlarge the interval between packets, causing another protocol malfunction. The specifications of ContikiMAC state that the inert packet interval should be 400µs.

The actual time between the end of one packet and the beginning of the next one (t_i) was calculated using the data recorded by the sniffer. The time between successive packets, measured from the Start of Frame, was 2756µs. Since the duration of the frame, derived from the frame length, is 1888µs and there are five bytes (160µs) preceding the Start of Frame detection, the time between successive frames is 2756µs - 1888µs - 160µs = 708µs, as shown in Figure 2.9.

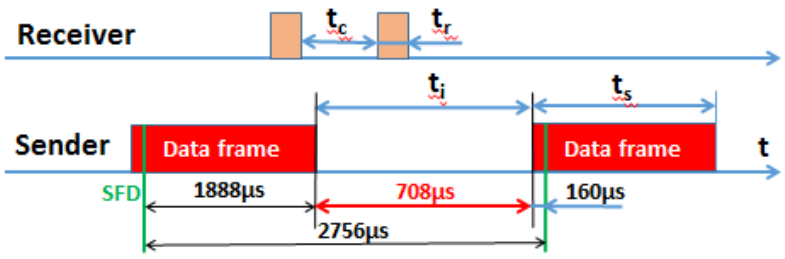


Figure 2.9: inter packet interval

This time is larger than the default time between two CCA assessments (500µs), meaning that if by chance one of the CCAs detects a frame, everything works fine, but when the two CCAs fall just between two frames, the receiver goes back to sleep. If the sender's and receiver's clocks are identical, either all the packets are detected or none of them is. On the other hand, if both clocks are not synchronized, the CCAs move from one frame to the next one and, in between, packets are lost. The better the accuracy of the clocks, the longer the duration of the lost packet bursts. This issue is illustrated in Figure 2.10.

There are three sources of delay (necessary or unnecessary) that make t_i too large.

1. Calibration of the transmitter before the start of a transmission: this takes 12 symbol times or 192µs, but it is possible to configure the transmitter in such a way that it takes only 8 symbol times or 128µs. By default, in the cc2420 driver of Contiki, the code to select 8 symbol times is included but commented out, which adds 64µs of unnecessary delay.

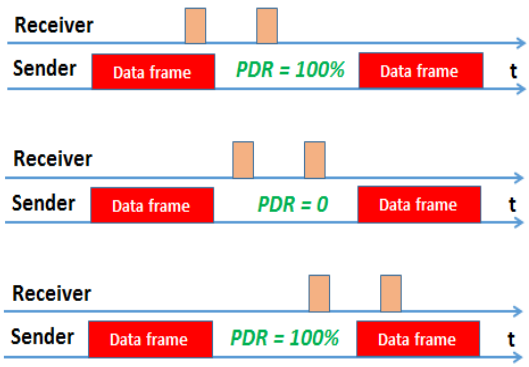


Figure 2.10: Packet loss due to long inter packet interval

2. Additional CCA performed before each frame is retransmitted. This option is enabled by setting WITH_SEND_CCA to 1 and it adds about 260µs of unnecessary delay. This option should be disabled (WITH_SEND_CCA=0) since it is not compatible with the proposed timing for ContikiMAC.

3. After sending a packet, ContikyMAC checks if the packet has been acknowledged before sending the next one. This is done in two steps: After the return from the cc2420_transmit function ContikiMAC waits for a period called erroneously INTER_PACKET_INTERVAL until the start of frame byte of an ACK should be detected by the receiver. This is checked by means of three ored different Boolean functions obtained from the cc2420 status registers (see Figure 2.8). If the transmitted packet is a broadcast (never acknowledged) or if no signal is detected by the receiver, ContikiMAC just loops back to transmit the next strobe. Otherwise ContikiMAC waits

for an additional AFTER_ACK_DETECTED_WAIT_TIME to be sure that the entire ACK packet has been received and then verifies if it was indeed an ACK by checking the length of the packet. If it was, retransmitting is stopped and ContikiMAC returns with success to the MAC layer. Otherwise, a collision is reported as some unexpected signal has been received.

Taking into account the unavoidable delays and the ContikiMAC requirements, one finds that the time available for detecting an incoming ACK cannot exceed 272us, while reliable detection of the ACK cannot be done sooner than 352 µs.

Before presenting the experiments it is important to remark that the INTER_PACKET_INTERVAL defined in the original code of ContikiMAC is not the actual time between two consecutive packets (t_i) but the time between sending a packet and checking for its ACK. In fact, as it can be seen in Figure 2.11, the INTER_PACKET_INTERVAL is only a part of the true interval between successive packets in ContikiMAC. The time necessary to calibrate the sender (128 or 192µs) and the duration of the CCA test performed when WITH_SEND_CCA is on (260µs) should be added to INTER_PACKET_INTERVAL to find the true inter packet interval (ti). That is why INTER_PACKET_INTERVAL will be renamed as BEFORE_ACK_DETECT_WAIT_TIME.

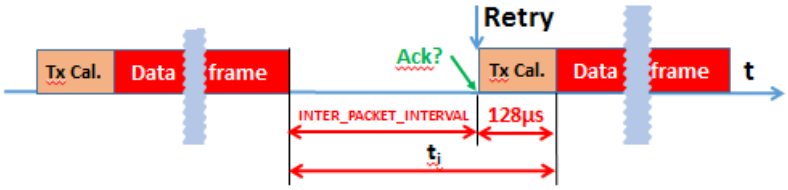


Figure 2.11: difference between INTER_PACKET_INTERVAL and t_i

To find a solution to this bug, it was necessary to evaluate the performance of the protocol using different values for the time the sender waits for an ACK in order to find the one that delivers the best PDR possible. These experiments where performed in the same testbed described previously in section 2.1.

The measurements were made first with a constant data rate of 1 packet per second and then a random rate of 0 to 2 seconds between packets, with a mean of 1 second.

Constant data rate:

Figure 2.12 shows that the network works perfectly when the RSSI is higher than -65dBm for all the tested values of BEFORE_ACK_DETECT_WAIT_TIME. On the other hand, when the RSSI is lower than -80dBm, the PDR is very low for the highest values of BEFORE_ACK_DETECT_WAIT_TIME (-457.5us and 488us). This experiment shows us that in an environment without interferences, the time the sender waits to receive an ACK from the

sender after sending a packet does not affect the PDR when the RSSI is high. On the other hand, with a low RSSI the PDR drops for long BEFORE_ACK_DETECT_WAIT_TIME.

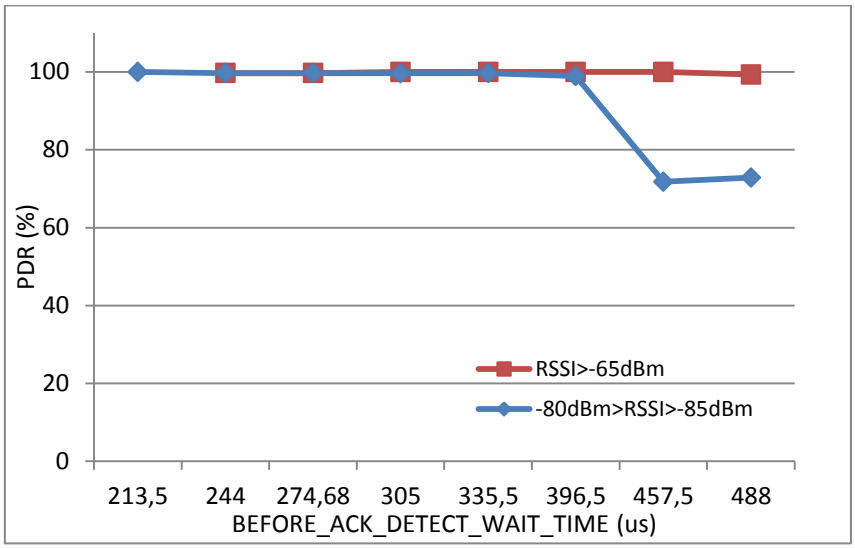


Figure 2.12: PDR for different delays for the check of an ACK with constant data rate

Random data rate:

For this experiment only the case of RSSI lower than -80dBm was tested, being that the case when the problems of packet loss are encountered when using ContikiMAC.

We can see in Figure 2.13 that for random data rates, when the RSSI is lower than -80dBm, the PDR is very low for the highest values of BEFORE_ACK_DETECT_WAIT_TIME (-457.5us and 488us). In some cases the PDRs higher than 100% were computed. This could be because some packets from the sender-observer did not reach the sink. The reason for this to happen could be because, as we were using a random data rate from 0 to 2 packets per second, the time between two successive packets could be too small for the sink to have time to process them. We also see that the results in this case are quite similar to those obtained with constant data rate and low RSSI, so that it is possible that de randomness of the data rate does not affect the PDR. For this reason and in order to avoid packet loss in the white network, in the following experiments only random data rate will be used, ranging from 10ms to 1990ms, maintaining a mean of 1s between packets. Moreover, NullRDC with acknowledgements will be used in the white network.

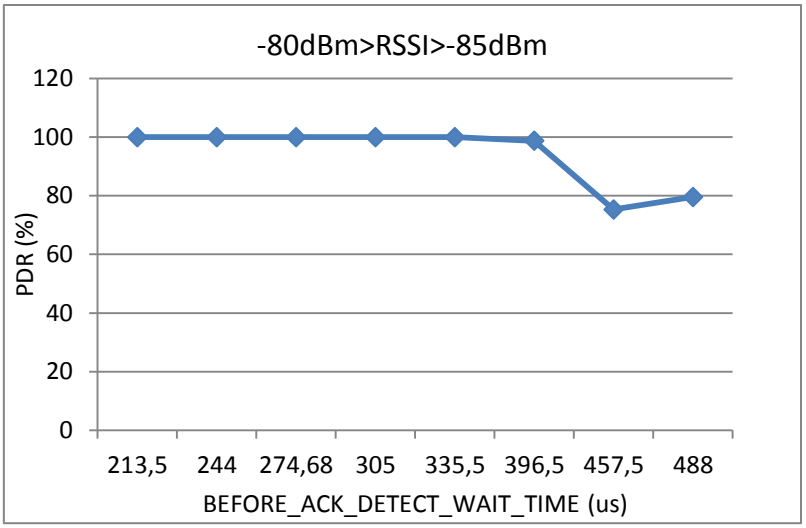


Figure 2.13: PDR for different delays for the check of an ACK with random data rate

Looking at the results obtained from these experiments, we see that for values between 270µs and 390µs the RSSI is 100%, so it was concluded that the best value for the BEFORE_ACK_DETECT_WAIT_TIME to assure a reliable ACK detection was around 360µs.

Apart from this, in order to see how interferences and obstacles affect the communication between these sensor nodes, these same tests were repeated in an office in Brussels.

The comparison between an interference free environment (the Ardennes) and an urban environment with obstacles and interferences from other devices (the office) is shown in Figure 2.14. In the case of the Ardennes, as explained before, for the right BEFORE_ACK_DETECT_WAIT_TIME values, the PDR is constant and almost always 100%. On the other hand, it can be clearly seen that the PDR values obtained in the office are very unstable and, therefore, not trustworthy. This is mainly due to the fact that the radio of the sensor node is very sensitive to interferences, topic that will be analysed more in depth in section 2.4. For this reason, for the main experiments of this thesis, developed in section 2.5, the measurements will be done in the Ardennes where there is almost no risk of having interferences.

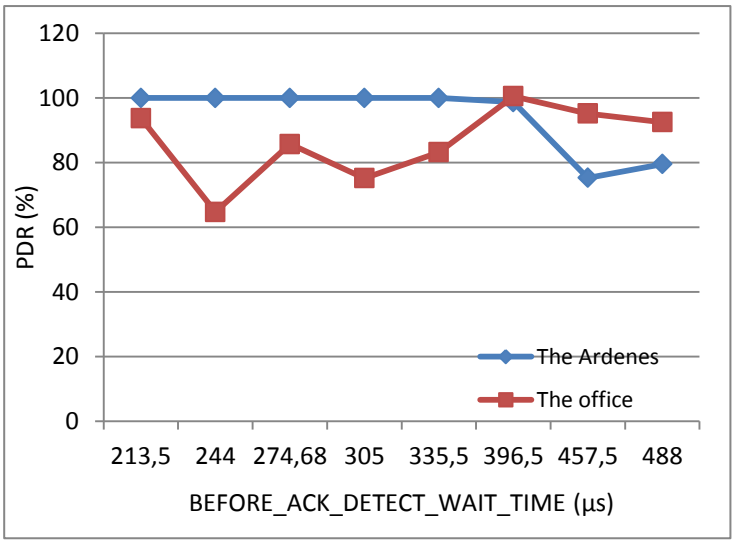


Figure 2.14: effect of interferences in PDR with different BEFORE_ACK_DETECT_WAIT_TIMES

2.4 Analysis of the effect of the CCA power threshold in a unicast link

During the previous experiments (section 2.3) an estrange behaviour was detected in the reception of packets below the CCA threshold when the channel_clear function was in charge of detecting the reception of an acknowledgement. It was observed that a few packets where received when the CCA should not detect any traffic and therefore the radio should not be awake.

The CCA threshold sets the value of RSSI below which packets (ACK) are not detected. In order to evaluate the effect of this value on the PDR of a unicast transmission, different experiments where made changing the threshold of the CCA from its default value -77dBm to, first, -85dBm, then -65dBm and, finally, -90dBm, which is more or less the limit of the cc2420 radio transceiver.

In order to change the CCA threshold, a function call should be added to the application to be run. This function call is cc2420_set_cca_threshold(*value*), where *value* corresponds to the value of the CCA threshold in dBm. The function cc2420_set_cca_threshold is defined in cc2420.c and, in fact, the *value* originally used in this function is not the actual value we introduce in the function call, but *value* + RSSI_offset. Looking at the cc2420 datasheet we see that the value of RSSI_offset, found empirically during system development from the front end gain, is, approximately, -45dBm. This means that, when reading a value of –32 from the RSSI register, the RF input power is approximately –77 dBm.

The results of these experiments are shown in Figure 2.15, where it can clearly be seen that, for all cases, for values above the CCA threshold the PDR is practically always around 100%, while below this threshold it drops significantly down to values around 25%. These results agree with what was explained before, since below the CCA threshold ACKs are ignored and a lot of packets are lost.

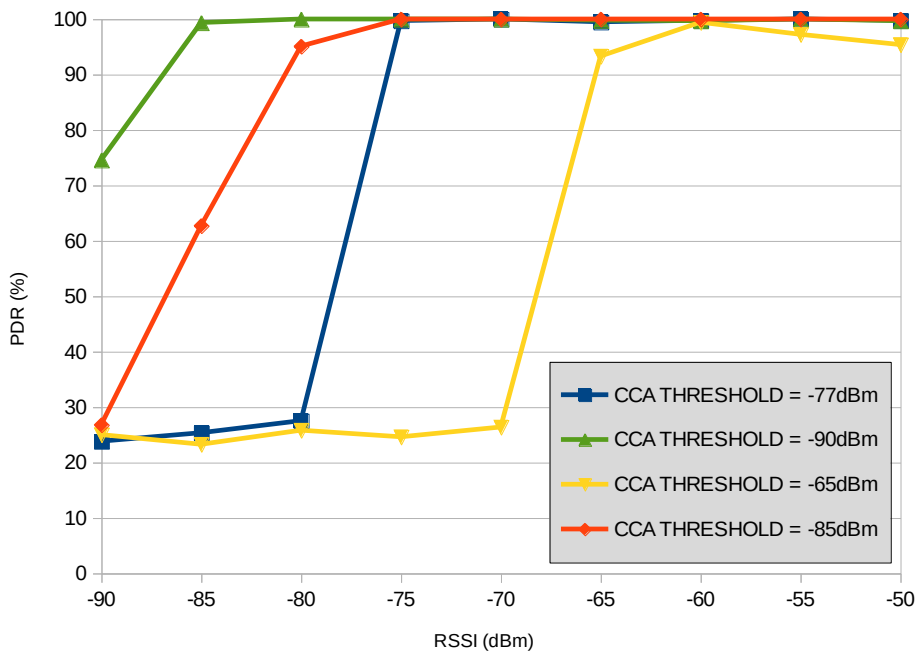
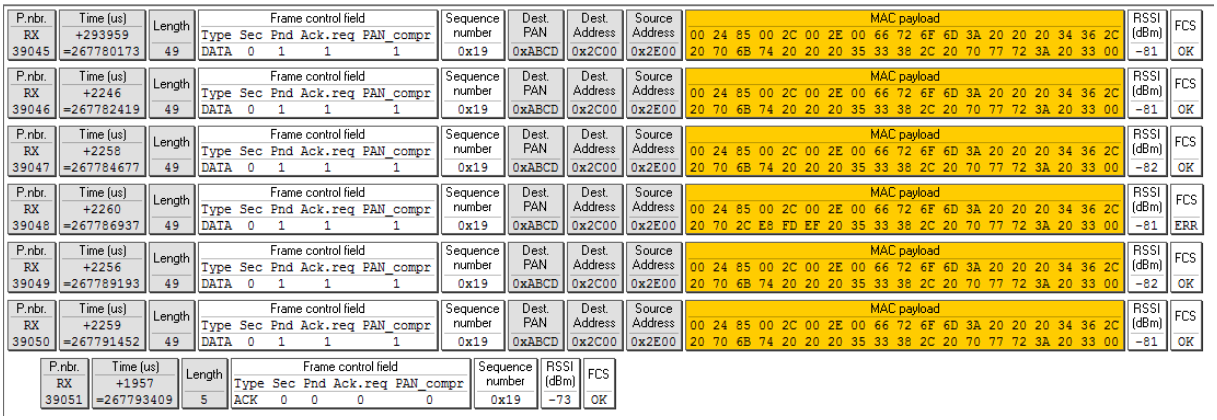


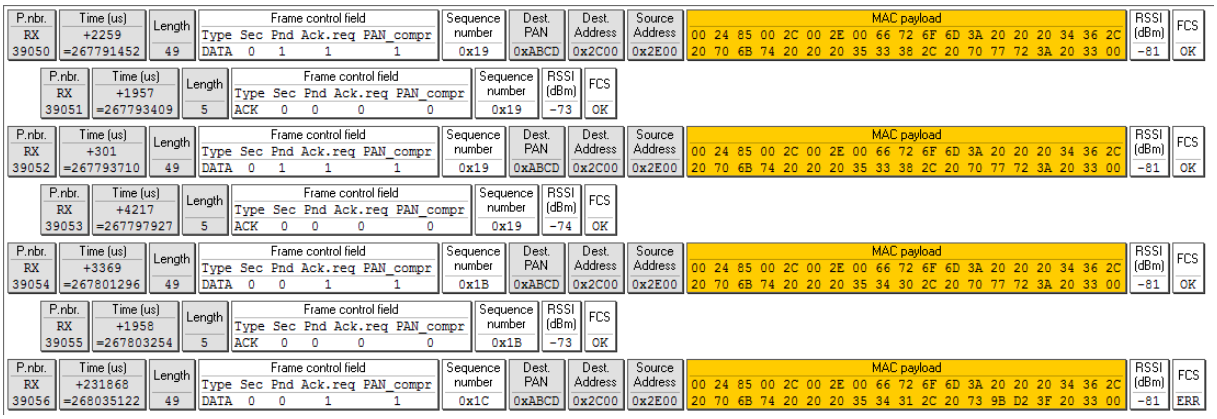
Figure 2.15: effect of CCA threshold on packet reception

However, it can also be seen that bellow the CCA threshold the PDR is not 0% as it should be theoretically, but around 25%, even for RSSIs of -90dBm, which is almost the limit of the radio. Some experiments were done in order to discover why some packets are received when the CCA assessment should not detect any radio activity and, therefore, the radio should have gone to sleep. The CCA threshold was set back to -77dBm and the measurements were taken with RSSI values between -78dBm and -90dBm. To analyse the behaviour of the link, we recorded serialdumps in the sender and the receiver together with sniffer files. To know when the radio was awaken, a printf was inserted in the part of the function powercycle where this happened. This way, we were able to see that, when the RSSI was 80dBm or lower, the radio awakes once and in a short period of time (around tens of miliseconds) a burst of packets is received.

Looking at the data recorded by the sniffer we saw that with low RSSI the packet was sent repeatedly by the sender until it received an ACK and, after that we could see a burst of packets sent and acknowledged in a short period of time. This would mean that, for some reason, the receiver's radio is awoken at one point and acknowledges a packet so, after that, the sender sends all the packets it had stored in the buffer. This behaviour is represented in the Figure 2.16. Figure 2.16.a shows the continuous transmission of a packet until an acknowledgement of said packet is received; Figure 2.16.b represents the burst of received packets and their corresponding ACKs that come after that first ACK, while the radio of the receiver is still awake.



(a)



(b)

Figure 2.16: packet reception below CCA threshold (serialdumps)

After changing the transmission rate in the sender from 1 second to 0.5 seconds we could see that the amount of packets received at once under 80dBm was bigger, so we analysed the sniffer files and we saw that, generally, in cases of low RSSI the time passed since the receiver started sending a packet until it was received was more or less a second, since, for low RSSis, the packets are sent a lot of times until they are acknowledged. This means that, with a data transmission rate of 0.5 seconds, the application generates messages faster than the radio sends them and, therefore, this packets are stored in the buffer until they can be sent. This way, when the radio is awake the sender transmits all the packets from the buffer, generating PDRs higher than expected. The next objective was to discover why the radio awakes at times when the channel_clear assessment should not detect any radio activity since the RSSI is below the CCA threshold. These tests were done in an office, so our theory was that the traffic from other devices outside of the link under test affects the CCA check in a way that it detects a stronger signal than it should and, therefore, it interprets that the level is higher than the threshold. To prove this theory, these same experiments were repeated in the Ardennes, far from any radio sources, and the resulting PDR below the CCA threshold was, indeed, almost 0%.

2.5 Performance of different RDC protocols in Contiki.

To examine the performance of the four different RDC protocols used in ContikiMAC an experimental study, comparing packet delivery ratio (PDR), packet latency and power consumption for each one of them was made using the Zolertia Z1 motes.

The testbed used for these experiments is described in section 2.1, and the ways to analyse the performance of the network are the ones described in section 2.2.

The Contiki Rime software is used for managing the unicast transmissions and CSMA is used as MAC protocol. Four different RDC protocols, available on Contiki, have been used for the tests in order to obtain clearly defined, but different traffic and power usage conditions. These protocols, in the black motes, are ContikiMAC, XMAC, LPP and, as a reference, NullRDC, which leaves the radio always on. In the white network, the RDC protocol used was NullRDC since the main concern is to have low latency, while the power usage is not an issue.

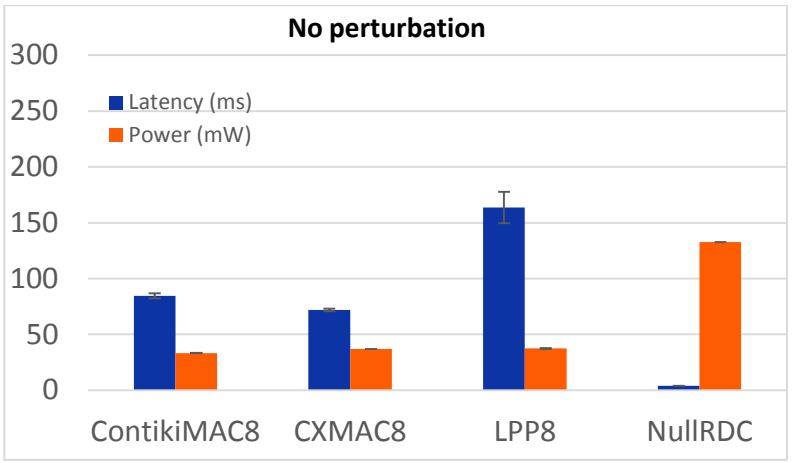
All the experiments in this section were done in the best conditions derived from sections 2.3 and 2.4. The RSSI was always higher than -77 dBm in order to avoid the problems detected in section 2.4 and, following the conclusions of section 2.3, the value for BEFORE_ACK_DETECT_WAIT_TIME was set to 12 clock ticks, corresponding to 366µs.

2.5.1 Comparison between the four main RDC protocols in Contiki.

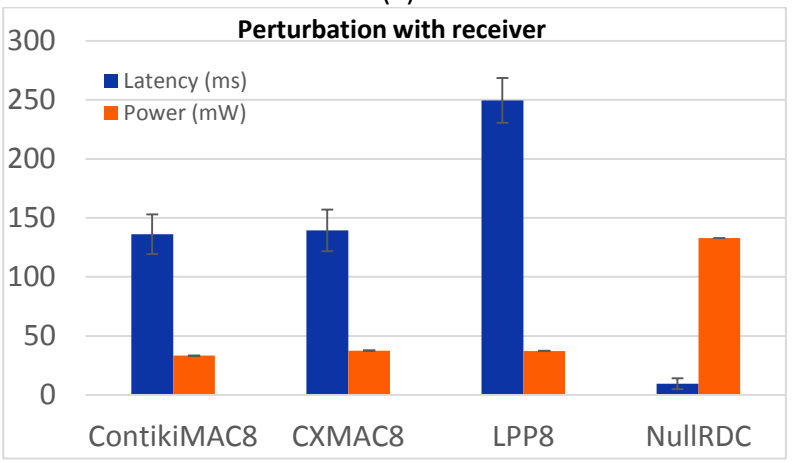
As it can be seen in Table 2.1, in general, for all the RDC protocols the PDR is around 100%, which is the expected result since the RSSI is always higher than -77dBm.

	PDR(%) mean		
	No perturbation	**Perturbation with receiver**	**Perturbation without receiver**
ContikiMAC8	100	99.69	99.75
CXMAC8	100	99.84	97.98
LPP8	100	100	
NullRDC	100	100	100

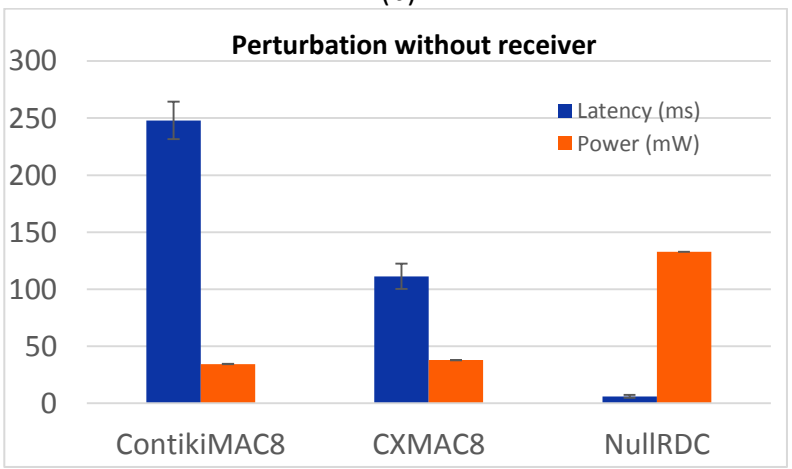
Table 2.1: PDR of the different RDC protocols



(a)



(b)



(c)

Figure 2.17: latency and power usage of the different RDC protocols available in Contiki for the three different perturbation scenarios: No perturbation in (a), perturbation with receiver in (b) and perturbation without receiver in (c). (Lines at the top of the bars correspond to the 95% confidence levels)

In general, as we can see in Figure 2.17, the latency is higher in the cases where there is perturbation. This is most notable in the case of ContikiMAC with perturbation without

receiver. In this case the latency is much bigger, being the difference with the other two perturbation scenarios bigger than in the rest of the RDC protocols. This is due to the fact that ContikiMAC sends the same packet repeatedly until it receives an ACK so that, if in the perturbing link there is no receiver and being the data rate twice higher, in this case the perturbing link will generate the biggest amount of traffic of all cases. This additional traffic will occupy the channel and make the sender in the main link wait to transmit the packets, resulting in bigger latencies. The standard deviation is also bigger in the cases with perturbation since the additional traffic provokes random packet loss and retransmissions, making the latencies of the different packets very variable. We can see this variability more clearly in Table 2.2 where the mean latency together with its standard deviation for each case are represented.

	Perturbation with receiver		**Perturbation without receiver**	
	Latency mean (ms)	Latency st. dev. (ms)	Latency mean (ms)	Latency st. dev. (ms)
ContikiMAC8	136.2	308.8	247.96	291.2
CXMAC8	139.4	317.6	111.29	218.14
LPP8	249.57	387.53		
NullRDC	9.54	82.43	6.05	24.32

Table 2.2: latency means and standard deviation of the 4 RDC protocols

Apart from that, it is proven that the lowest latency is achieved using NullRDC, which is predictable since it keeps the radio on all the time and, therefore, the messages are sent as soon as they reach the MAC layer if the channel is free. However, for this same reason the power usage is the highest of all cases, making NullRDC the least suitable RDC protocol for power constrained nodes. Analysing the other protocols, we see that with LPP8, even if it provides a low power usage, the latency obtained is very high compared to the other protocols, so it is not the best option either. The reason for these delays is the phase lock mechanism, as it takes longer than in the rest of the RDC protocols. To prove this, LPP8 was tested once more, this time disabling the phase lock mechanism.

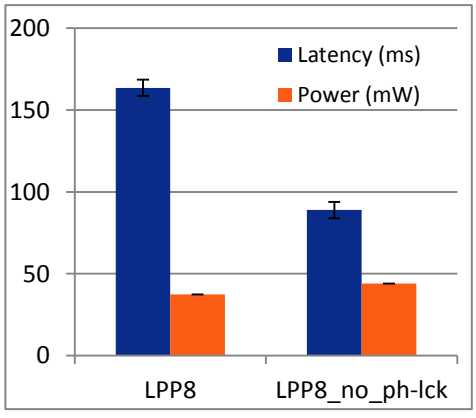


Figure 2.18: effect of phase lock in latency in LPP8. (Lines at the top of the bars correspond to the 95% confidence levels)

Figure 2.18 proves that, certainly, the latency is much lower when the phase lock is disabled and, as expected, the power usage is higher.

From the remaining two RDC protocols, ContikiMAC is the default option in Contiki and, as it can be seen in the graphs, it provides a very low power usage while maintaining the latency moderately low as well. However, it can be observed that, for most of

the cases and most notably for the worst scenario (perturbation without receiver), CXMAC achieves lower latencies than ContikiMAC with a similar power usage. This results show that, even if ContikiMAC is the most widespread RDC protocol for WSNs running Contiki, it might be interesting to use CXMAC instead in order to achieve a faster network.

2.5.2 Comparison between the different wake up intervals (WUI) of ContikiMAC.

To further investigate the RDC protocols in Contiki, some experiments were done to see the differences in the performance of one of an RDC protocol using different wake up intervals. In these experiments three different wake up intervals were analysed: 250ms (the radio wakes up 4 times per second), 125ms (the radio wakes up 8 times per second) and 62.5ms (the radio wakes up 16 times per second). Due to the lack of time, only ContikiMAC was used in these experiments, being currently the default option in Contiki. Theoretically, the conclusion derived from these tests should also be applicable to CXMAC and LPP.

Figure 2.19 depicts the graphs of the latency (a) and the power usage (b) for different types of perturbation for each of the three wake up intervals in ContikiMAC. As for the latency, it can be seen that it is directly proportional to the WUI. This makes sense since, when the radio spends longer periods being asleep, a message has to wait longer to be transmitted, due to the CSMA/CA mechanism used by the MAC layer to access the channel. On the other hand, the exact opposite happens with the power usage, as more power will be consumed if the radio wakes up more frequently.

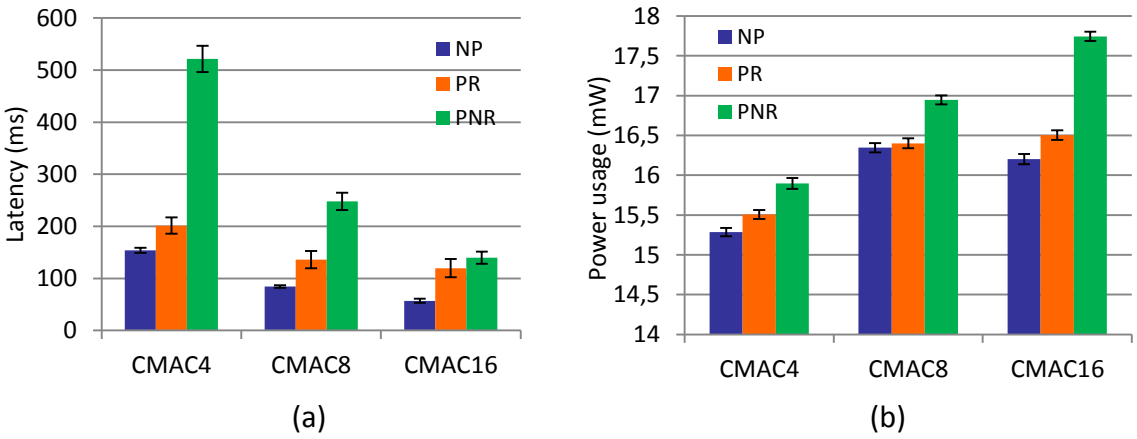


Figure 2.19: Latency and Power Usage in function of Wake up interval in ContikMAC under conditions of No Perturbation (NP), Perturbation with Receiver (PR) and Perturbatuon without Receiver (PNR). (Lines at the top of the bars correspond to the 95% confidence levels)

It is also interesting to notice that for long wake up intervals (ContikiMAC4), the latency is more affected by the additional traffic since we can clearly see that latency in the perturbation without receiver scenario is significantly bigger than in the no perturbation scenario. This is due to the fact that, in the worst case, for this WUI, a packet has to wait up to 250ms for each retransmission, with the addition of the random time between retransmissions that grows exponentially. On the contrary, for short wake up intervals it is the power that is more affected, though the differences between different waking periods and perturbation scenarios are not that big in the case of power usage.

In conclusion, it is clear that it is not possible to improve both latency and power usage by changing the wake up interval in ContikiMAC, so the choice of WUI should depend on the application. For example, if it is required that the sensor operates unattended during long periods of time and the delays are not an issue, low power consumption is the main goal, so ContikiMAC4 will be the best choice. On the other hand, if the application requires fast responses and the power consumption is not that important, it will be wise to use ContikiMAC16. For the rest of the applications, ContikiMAC8 will be the perfect compromise between packet latency and power consumption.

Chapter 3 Conclusions and future work.

3.1 Conclusions.

After the experiments done during the development of this bachelor thesis, it is clear that the communication between the Zolertia Z1 motes is highly affected by interferences, mostly when the RSSI of the received packets is low. This issue reaches the point where, if the experiments are done in an environment with a lot of other radio transmitters, it is not possible to obtain conclusive PDR results, for instance. This is why, even if these sensors can operate in these kind of environments, all experimental evaluations to analyse the performance of these motes should be done in an interference-free environment.

In regards to the main purpose of this thesis, we can conclude that NullRDC and LPP are not advisable for WSN applications using Zolertia Z1 motes, the former because of its high power consumption and the latter because of the big latencies observed. Moreover, a malfunction in the phase lock loop mechanism was detected in LPP, which caused these undesirably high latencies. This problem should be studied in the future in order to find a solution. As for the remaining two RDC protocols, having in mind the better performance observed in packet latency, it could be interesting to try CXMAC in WSN applications instead of the nowadays more used ContikiMAC.

Finally, it was concluded that, with RDC protocols like the ones under test, it is not possible to improve both latency and power usage at the same time. That is why ContikiMAC8, which defines a waking period of 125ms, is the one used in most of the WSN applications, due to the fact that, as we saw in section 2.5.2, it achieves a good trade-off between latency and power usage. Nevertheless, if the application demands a fast communication, as it is the case of earthquake, fire and intrusion detection, it could be interesting to relatively sacrifice the power consumption and use a WUI of 62.5ms (ContikiMAC16). On the other hand, if very low power usage is crucial, like it is the case of water meters, ContikiMAC4, with a WUI of 250ms could be the best option even if the latencies are bigger.

3.2 Future work.

This thesis only presented the comparison between WUIs in ContikiMAC, so, in the following months, it would be interesting to continue analysing the behaviour of LPP and CXMAC using different WUIs, even if similar results to ContikiMAC are expected, in order to make a more

profound and complete study in that topic. Also longer WUIs will be analysed, such as 1 second, as occasionally, some nonlinear behaviour of the performance was observed when changing the WUI.

Also, parallel to this project, the INDI/ETRO department is in the first phases of a project destined to use the LoRa motes with Contiki. The LoRa (Long Range) motes are an improvement to the Zolertia Z1 motes, since they can reach much longer distances in transmission. However, by law, the radio in the LoRa motes can only transmit during 1% of the duty cycle, so that, as LPP is the only RDC protocol in Contiki that fulfils that condition, it is necessary to debug LPP in order to fix the problem with the phase lock mechanism encountered during the experiments of the section 2.5.

References

[1] J. Yick, B. Mukherjee, and D. Ghosal: *Wireless sensor network survey*, Computer Networks, Department of Computer Science, University of California, Davis, CA 95616, United States, 2008.

[2] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, *A survey on sensor networks*, IEEE Communication Magazine, 2002.

[3] T. Reusing: *Comparison of Operating Systems TinyOS and Contiki*, Technische Universität München, 2012.

[4] A. Abed, A. Alkhatib, and G. S. Baicher: *Wireless Sensor Network Architecture*, CNCS 2012, University of Wales Newport, City Campus, Usk Way, NP20 2BP, Newport, U.K, 2012.

[5] I. Howitt and J. Gutierrez: *IEEE 802.15.4 low rate-wireless personal area network coexistence issues*, Charlotte, NC 2003.

[6] W. Craig: *Zigbee: Wireless control that simply works*, Program Manager Wireless Communications ZMD America, Inc., 2004.

[7] Z. Shelby and C. Bormann: *6LoWPAN: The Wireless Embedded Internet*, John Wiley & Sons, UK 2009, pp. 4-8.

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

[9] TEXAS INSTRUMENTS. *CC2420, 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver*, datasheet.

[10] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, D. Culler, W. Weber, J. Rabaey, and E. Aarts, TinyOS: *An Operating System for Wireless Sensor Networks*, obtained from: http://www.cs.berkeley.edu/~culler/cs294-f03/papers/tinyos.pdf, 2005.

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

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

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

[14] A. Silberschatz, G. Cagne, P. B. Galvin: *"Chapter 4 - Processes". Operating system concepts with Java* (Sixth Edition ed.).2004. John Wiley & Sons. ISBN 0-471-48905-0.

[15] A. Dunkels, O. Schmidt, T. Voigt, M. Ali: Protothreads: *Simplifying Event- Driven Programming of Memory-Constrained Embedded Systems*. Proceedings of SenSys'06, November 1-3, 2006, Boulder, Colorado, USA.

[16] A. Dunkels, O. Schmidt, and T. Voigt: *Using protothreads for sensor node programming*. In Proceedings of the Workshop on Real-World Wireless Sensor Networks (REAL WSN'05), Stockholm, Sweden, June 2005.

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

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

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

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

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

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

[23] A. Dunkels, F. Österlind and Zhitao He: *An Adaptive Communication Architecture for Wireless Sensor Networks*, SenSys'07, Sydney, Australia, 2007.

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

[25] A. Dunkels, F. Osterlind, N. Tsiftes, and Z. He, *Software-based on-line energy estimation for sensor nodes*, 2007.

[26] M. Bezunartea: *Experimental Evaluation of Communication Protocols for Wireless Sensor Networks*, Master Thesis, Vrije Universiteit Brussel, ETRO, 2014.

[27] *https://sindarku.wordpress.com/2010/08/13/protocol-stack-for-wireless-sensor-networks-wsns*

[28] *http://anrg.usc.edu/contiki/index.php/MAC_protocols_in_ContikiOS*

[29] *https://github.com/contiki-os/contiki/wiki/Radio-duty-cycling*

[30] *http://zolertia.sourceforge.net/wiki/index.php/Z1*