Technical Report

Open-ZB: an open-source implementation of the IEEE 802.15.4/ZigBee protocol stack on TinyOS

André Cunha

Anis Koubaa

Ricardo Severino

Mário Alves

HURRAY-TR-070409

Version: 1.0

Date: 09-04-2007

Open-ZB: an open-source implementation of the IEEE 802.15.4/ZigBee protocol stack on TinyOS

André Cunha, Anis Koubaa, Ricardo Severino, Mário Alves

IPP-HURRAY!

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail: arec@isep.ipp.pt akoubaa@dei.isep.ipp.pt rars@isep.ipp.pt mjf@isep.ipp.pt

http://www.hurray.isep.ipp.pt

Abstract

The IEEE 802.15.4/ZigBee protocols are gaining increasing interests in both research and industrial communities as candidate technologies for Wireless Sensor Network (WSN) applications. In this paper, we present an open-source implementation of the IEEE 802.15.4/Zigbee protocol stack under the TinyOS operating system for the MICAz motes. This work has been driven by the need for an open-source implementation of the IEEE 802.15.4/ZigBee protocols, filling a gap between some newly released complex C implementations and black-box implementations from different manufacturers. In addition, we share our experience on the challenging problem that we have faced during the implementation of the protocol stack on the MICAz motes. We strongly believe that this open-source implementation will potentiate research works on the IEEE 802.15.4/Zigbee protocols allowing their demonstration and validation through experimentation.

Open-ZB: an open-source implementation of the IEEE 802.15.4/ZigBee protocol stack on TinyOS

André CUNHA[1], Anis KOUBÂA[1,2], Ricardo SEVERINO[1], Mário ALVES[1]

[1] IPP-HURRAY! Research Group, Polytechnic Institute of Porto, Rua António Bernardino de Almeida, 431, 4200-072 Porto, Portugal

[2] Al-Imam Muhammad Ibn Saud University, Computer Science Dept., 11681 Riyadh, Saudi Arabia

arec@isep.ipp.pt, akoubaa@dei.isep.ipp.pt, rars@isep.ipp.pt, mjf@isep.ipp.pt

Abstract **- The IEEE 802.15.4/ZigBee protocols are gaining increasing interests in both research and industrial communities as candidate technologies for Wireless Sensor Network (WSN) applications. In this paper, we present an open-source implementation of the IEEE 802.15.4/Zigbee protocol stack under the TinyOS operating system for the MICAz motes. This work has been driven by the need for an open-source implementation of the IEEE 802.15.4/ZigBee protocols, filling a gap between some newly released complex C implementations and black-box implementations from different manufacturers. In addition, we share our experience on the challenging problem that we have faced during the implementation of the protocol stack on the MICAz motes. We strongly believe that this open-source implementation will potentiate research works on the IEEE 802.15.4/Zigbee protocols allowing their demonstration and validation through experimentation.**

1. Introduction

The IEEE 802.15.4 protocol specifies the Medium Access Control (MAC) sub-layer and the Physical Layer of Low-Rate Wireless Private Area Networks (LR-WPAN) [1]. Although this standard protocol was not specifically developed for Wireless Sensor Networks (WSNs), it provides enough flexibility for fitting different requirements of WSN applications by adequately tuning its parameters. In fact, low-rate, low-power consumption and low-cost wireless networking are the key features of the IEEE 802.15.4 protocol, which typically fit the requirements of WSNs [2]. Moreover, the ZigBee specification [3] relies on the IEEE 802.15.4 Physical and Data Link Layers, building up the Network and Application Layers, thus defining a full protocol stack for LR-WPANs.

The ZigBee Alliance - an organization with more than 150 company members - has been working in conjunction with the IEEE Task Group 15.4 in order to specify a full protocol stack for low cost, low power, low data rate wireless communications, as well as to foster its worldwide use. The ZigBee specification, with a new release in December 2006, aims at the provision of a standard protocol that facilitates the interoperability between multiple hardware and software platforms from different providers. Fig. 1 shows the layered architecture of the IEEE 802.15.4/Zigbee protocol stack.

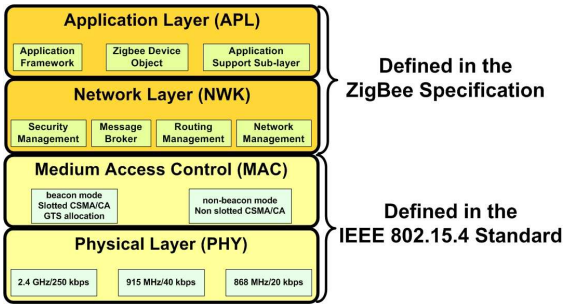


Fig. 1. The IEEE 802.15.4/ZigBee protocol stack architecture

The IEEE 802.15.4/Zigbee protocols have attracted several recent research works (e.g. [4-9]). Most of those research studies have typically focused on the evaluation/improvement of some characteristics of the standard protocols either analytically or by simulation. No experimental work has argued any of those research works due to the lack of a real open-source implementation of the IEEE 802.15.4/Zigbee protocol stack. This lack prevents from experimentally demonstrating the feasibility of the proposed approaches and from the accurate validation of the theoretical results of those studies, since simulation tools are usually not sufficient to evaluate the real behaviour of the protocols due to many abstractions in the simulation models.

There is a tremendous motivation for developing an open-source implementation of IEEE 802.15.4/Zigbee for different sensor network platforms to (1) foster the development of research works focusing on the IEEE 802.15.4/Zigbee protocol stack, (2) provide a means to validate, demonstrate and evaluate the real deployment of IEEE 802.15.4/Zigbee WPANs.

In this paper, we propose Open-ZB [10], an open-source implementation of the protocol stack under the TinyOS operating system. So far, the implementation has been made for the MICAz motes [11]. In addition, for the sake of a comparative evaluation between simulation and experimentation of the IEEE 802.15.4/ZigBee protocol stack, we have also developed a simulation model using the OPNET simulator [12]. This simulation tool implements the Physical and the MAC Layers of the IEEE 802.15.4 protocol standard supporting the physical layer characteristics, the beacon enabled mode, the slotted CSMA/CA, the protocol frame formats and a battery module that computes the

consumed and remaining energy levels for the MICAz motes.

This implementation was developed in the context of the ART-WiSe Framework [13], which consists in providing real-time and reliable communication for WSNs using COTS (Commercial Off The Shelf) technologies. We expect that the line of work we have been following in the assessment, improvement and engineering of IEEE 802.15.4/ZigBee networks will have significant repercussions. IEEE 802.15.4 and ZigBee are emerging technologies with plenty of potentialities for WSN applications. Nevertheless, for these technologies to gain widespread use we believe it is important to provide open source implementations of these protocols, to act as common platforms for the scientific community to discuss, interact and contribute. Moreover, it is important for the scientific community to collaborate with the official working groups from IEEE and with the ZigBee Alliance in a way that our findings can contribute for improving the current protocol standards.

The main contribution of the paper is the provision of a comprehensive description of our IEEE 802.15.4/Zigbee implementation and demonstrating its importance in fostering the development of research work based on these standard protocols.

The rest of the paper is organized as follows. Section 2 highlights some relevant aspects of the IEEE 802.15.4/ZigBee protocols. Some implementation details are shown in Section 3, namely general aspects of our development environment, a short overview of TinyOS[14] and nesC [15] programming language, the implementation structure and future challenges. In Section 4 we present some research achievements based on our implementation.

2. Overview - IEEE 802.15.4/ZigBee protocols

2.1 IEEE 802.15.4 Phy and Mac

The IEEE 802.15.4 specification defines two different types of devices: the Full Function Devices (FFDs) that implement the full protocol stack and the Reduced Function Devices (RFDs) that only implement a subset of the protocol stack. The FFDs can have three different roles in the network: (1) the Personal Area Network (PAN) Coordinator: the principal controller of the PAN, identifying the network and its configurations; (2) the Coordinator: provides synchronization services through the transmission of beacons; this device should be associated to a PAN coordinator and does not create its own network; (3) the End Devices: do not implement the previous functionalities and should associate with a Coordinator before interacting with the network.

The RFD is an end device operating with the minimal implementation of the IEEE 802.15.4 protocol. An RFD is intended for applications that are extremely simple, such as a light switch or a passive infrared sensor; they do not have the need to send large amounts of data and may only associate with a single FFD at a time.

The IEEE 802.15.4 physical layer is responsible for data transmission and reception using a certain radio channel and according to a specific modulation and spreading technique. It offers three operational frequency bands: 2.4 GHz, 915 MHz and 868 MHz. There is a single channel between 868 and 868.6 MHz, 10 channels between 902 and 928 MHz, and 16 channels between 2.4 and 2.4835 GHz. The protocol also allows dynamic channel selection, a channel scan function in search of a beacon, receiver energy detection, link quality indication and channel switching.

Lower frequencies are more suitable for longer transmission ranges due to lower propagation losses. Low rate transmissions provide better sensitivity and larger coverage area. Higher rate means higher throughput, lower latency or lower duty cycles. All of these frequency bands are based on the Direct Sequence Spread Spectrum (DSSS) spreading technique.

The IEEE 802.15.4 MAC protocol supports two operational modes that may be selected by the PAN Coordinator: (1) the non beacon-enabled mode, in which the MAC is simply ruled by non-slotted CSMA/CA, (2) the beacon enabled mode, in which beacons are periodically sent by the Coordinators to synchronize nodes that are associated with it, and to identify the PAN.

In beacon-enabled mode, the PAN Coordinator defines a superframe structure (Fig. 2) which is constructed based on (1) the Beacon Interval (*BI*), defining the time between two consecutive beacon frames, (2) the Superframe Duration (*SD*), defining the active portion in the BI, being divided into 16 equally-sized time slots, during which frame transmissions are allowed. Optionally, an inactive period is defined if *BI* > *SD*. During the inactive period (if it exists), all nodes may enter in a sleep mode (to save energy). *BI* and *SD* are determined by two parameters, the Beacon Order (*BO*) and the Superframe Order (*SO*), respectively, as follows:

$$\left.\begin{aligned} BI &= aBaseSuperframeDuration \cdot 2^{BO} \\ SD &= aBaseSuperframeDuration \cdot 2^{SO} \end{aligned}\right\} for\ 0 \le SO \le BO \le 14 \qquad (1)$$

aBaseSuperframeDuration = 15.36 ms (assuming 250 kbps in the 2.4 GHz frequency band) denotes the minimum duration of the superframe, corresponding to *SO=0*. During the superframe duration, nodes compete for medium access using slotted CSMA/CA in the Contention Access Period (CAP).

As depicted in Fig. 2, low duty cycles can be configured by setting small values of *SO* as compared to *BO*, resulting in greater inactive periods. Additionally,

the IEEE 802.15.4 protocol also provides real-time guarantees by using the Guaranteed-Time Slot (GTS) mechanism. This feature is quite attractive for time-sensitive WSN applications. In fact, when operating in beacon-enabled mode, the IEEE 802.15.4 protocol allows the allocation/deallocation of GTSs in a superframe for nodes that require real-time guarantees.

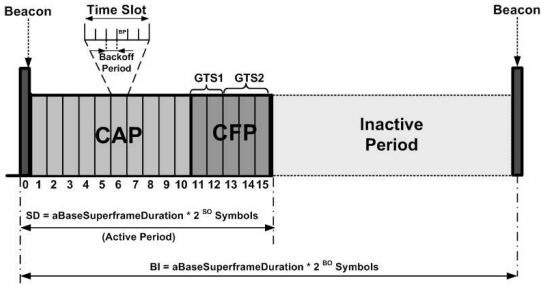


Fig. 2. IEEE 802.15.4 Superframe Structure

The GTS allows the corresponding device to access the medium without contention in the CFP. The GTS is allocated by the Coordinator and is used only for communications between the Coordinator and a device. A single GTS may extend over one or more time slots. The Coordinator may allocate up to seven GTSs at the same time, provided that there is sufficient capacity in the superframe. Each GTS has only one direction: from the device to the coordinator (transmit) or from the coordinator to the device (receive).

The GTS can be deallocated at any time at the discretion of the Coordinator or the device that originally requested the GTS. A device to which a GTS has been allocated can also transmit during the CAP. The Coordinator is the responsible for performing the GTS management; for each GTS, it stores the starting slot, length, direction, and associated device address. All these parameters are embedded in the GTS request command. Only one transmit and/or one receive GTS are allowed for each superframe.

The IEEE 802.15.4 defines two modes of medium access. The slotted CSMA/CA in beacon enabled mode and the non slotted CSMA/CA in non-beacon enabled mode. The CSMA/CA mechanism is based on backoff periods (with the duration of 20 symbols). Three variables are used to schedule the access to the medium: (1) Number of Backoffs (*NB*), representing the number of failed attempts to access the medium; (2) Contention Window (*CW*), representing the number of backoff needed to be clear before starting transmission; (3) Backoff Exponent (*BE*), enabling the computation of the number of wait backoffs before attempting to access the medium again.

Fig 3 depicts a flowchart describing the two modes of the CSMA/CA mechanism.The slotted CSMA/CA can be summarized in five steps: (1) initialization of the algorithm variables: *NB* equal to 0; *CW* equals to 2 and *BE* is set to the minimum value between 2 and a MAC layer constant definition (*macMinBE*); (2) after locating a backoff boundary, the algorithm waits for a random defined number of backoff before attempting to access the medium; (3) Clear Channel Accessment (CCA) to verify if the medium is idle or not. (4) The CCA returned a busy channel, the *NB* is incremented by 1 and the algorithm must start again in Step 2; (5) The CCA returned an idle channel, the *CW* is decremented by 1 and if it reaches 0 then the message is transmitted otherwise the algorithm jumps to Step 3.

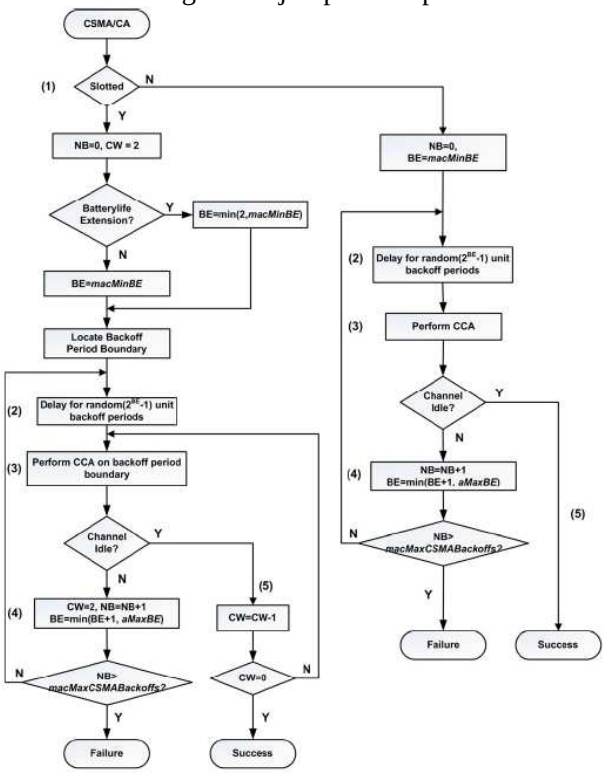


Fig. 3. The CSMA/CA mechanism

The non slotted mode of the CSMA/CA is very similar to the slotted version except the algorithm does not need to rerun (*CW* number of times) when the channel is idle.

2.2 ZigBee Network Layer

In ZigBee networks there are 3 types of devices: (1) ZigBee Coordinator (ZC): FFD, one for each ZigBee Network, initiates and configures the network formation, acts as an IEEE 802.15.4 PAN Coordinator and also as a ZigBee Router (ZR) once the network is formed; (2) ZigBee Router (ZR): FFD, associated with the ZC or with a previously associated ZR, acts a an IEEE 802.15.4 PAN Coordinator, participates in multi-hop routing of messages; (3) ZigBee End Device (ZED): does not allow other devices to associate with it, does not participate in routing and may implement a reduced subset of the protocol stack (RFD).

Throughout this document the names of the devices and the acronyms are used interchangeably

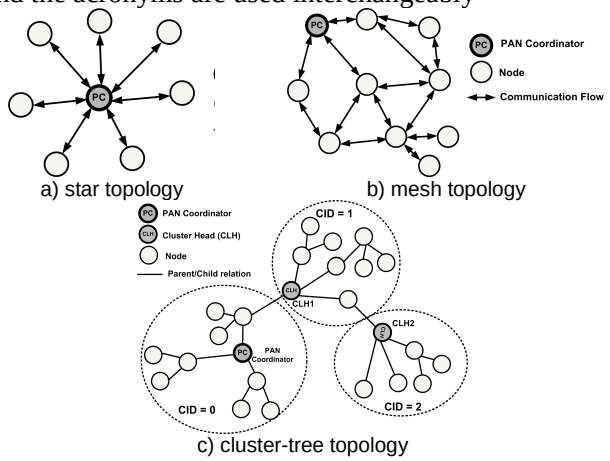


Fig. 4. IEEE 802.15.4 network topologies

The IEEE 802.15.4/ZigBee enables three network topologies – star, mesh and cluster-tree.

In the star topology (Fig.4a), a unique node operates as a ZC. The ZC chooses a PAN identifier, which must not be used by any other ZigBee network in the vicinity. The communication paradigm of the star topology is centralized i.e., each device (FFD or RFD) joining the network and willing to communicate with other devices must send the data to the ZC, which dispatches it to the adequate destination. The star topology may not be adequate for traditional wireless sensor networks for two reasons. First, the sensor node selected as a PAN Coordinator will get its battery resources rapidly ruined. Second, the coverage of an IEEE 802.15.4 cluster is very limited while addressing a large-scale WSN, leading to a scalability problem.

The mesh topology (Fig. 4b) also includes a ZC that identifies the entire network. However, the communication paradigm in this topology is decentralized - each node can directly communicate with any other node within its radio range. The mesh topology enables enhanced networking flexibility, but it induces an additional complexity for providing end-to-end connectivity between all nodes in the network. Basically, the mesh topology operates in an ad-hoc fashion and allows multiple hops to route data from any node to any other node. In contrast with the star topology, the peer-to-peer topology may be more power-efficient and the battery resource usage is fairer, since the communication process does not rely on one particular node.

The cluster-tree network topology (Fig. 4c) is a special case of a mesh network where there is a single routing path between any pair of nodes and there is a distributed synchronization mechanism (beacon-enabled mode). There is only one ZC which identifies the entire network and one ZR per cluster. Any of the FFD can act as a coordinator and provide synchronization services to other devices and coordinators. The nomination of new Coordinators is the role of the PAN Coordinator.

3. Implementation Details

3.1 General Aspects

This implementation was developed under the TinyOS operating system version 1.1.15, for the MICAz [11] motes (16 MHz Atmel ATMega128L and a 2.4 GHz Chipcon CC2420 radio transceiver [16]). The MIB510 was used to program the motes. This programming board is able to upload the applications to motes through the serial port (or COM port) and provides a debug mechanism by sending data through the COM port and reading it in a software listener (e.g. ListenRaw, provided with the TinyOS distribution, or Windows HyperTerminal). This debug mechanism raises a problem concerning the hardware operation because the relaying of data through the COM port blocks all the other mote operations, while this data is being sent. This can usually cause synchronization problems. In order to overcome the COM debug problems we use packet sniffers to track and display the packets being transmitted, which provides a better debugging mechanism by transmitting debug data in the packet payloads. We have used two different packet sniffer applications. The first is an IEEE 802.15.4/ZigBee packet sniffer provided by Chipcon - the CC2420 Packet Sniffer for IEEE 802.15.4 v1.0 [17] that provides a raw list of the packets transmitted. This application works in conjunction with a CC2400EB evaluation board and a CC2420 radio transceiver. We have also used the Daintree IEEE 802.15.4/ZigBee Network/Protocol Analyser [18] that provides more functionalities (e.g. graphical topology of the network, statistics, message flows, PAN information, association details, etc.).

3.2 TinyOS and nesC

TinyOS [14] is an operating system for embedded systems with an event-driven execution model. TinyOS is developed in nesC [15], a language for programming structured component-based applications. nesC has a C-like syntax and is designed to express the structures of TinyOS. This includes the concurrency model and mechanisms for structuring, naming and linking together software components into embedded system applications. The component-based application structure provides a good flexibility in the process of the application design and development. nesC applications are built out of components and interfaces. The components define two areas: (1) the specification, a code block that declares the functions it provides (implements) and the functions that it uses (calls); (2) the implementation, a collection of the functions

provided. The interfaces are a bidirectional collection of functions provided or used by a component. The interface commands are implemented by the providing component and the interface events are implemented by the component using them. The components are binded together by the means of interfaces and the overall set constitutes an application.

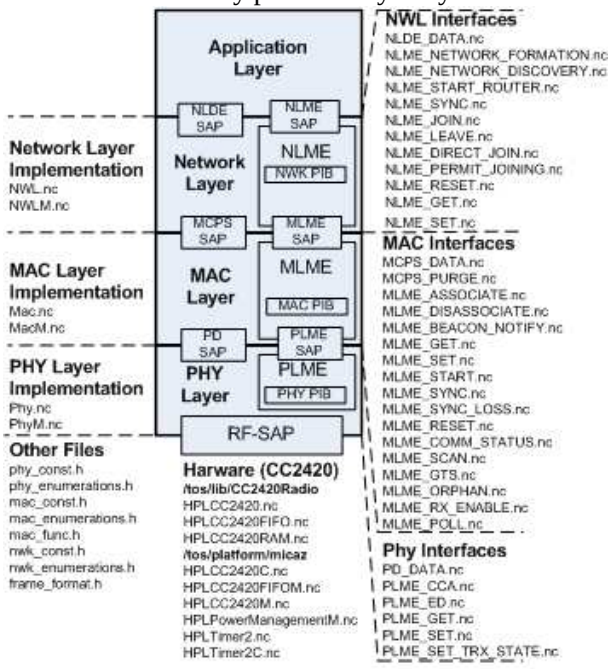
TinyOS defines a concurrency model based on tasks and hardware event handlers/interrupts. The TinyOS tasks are synchronous functions that run without preemption until completion and their execution is postponed until they can execute. Hardware events are asynchronous events that are executed in response to a hardware interrupt and also run to completion. All the asynchronous tasks and events may preempt running/synchronous non-atomic code.

3.3 Software architecture

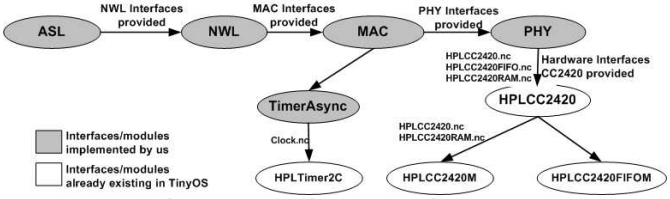
The Open-ZB implementation has three main TinyOS components: the Phy, the Mac and the NWL. The Phy component implements the following Physical Layer tasks: (1) activation and deactivation of the radio transceiver; (2) energy detection within the current channel; (3) transceiver data management, Received Signal Strength Indication (RSSI) readings and channel frequency selection; (4) Clear Channel Assessment (CCA) procedure for the CSMA/CA mechanism; (5) data transmission and reception management. The Mac component provides the following functionalities: (1) beacon generation if the device is a coordinator; (2) synchronization services; (3) PAN association and disassociation procedures; (4) CSMA/CA as a contention access mechanism; (5) the GTS management mechanism. The NWL component provides the following functionalities: (1) definition of the network topology (by enabling the device operation as a ZC, ZR or ZED); (2) association mechanisms; (3) ZigBee addressing schemes; (4) maintenance of neighbour tables; (5) tree-routing.

Fig. 5.a presents the layered view of the different TinyOS components and interfaces of our IEEE 802.15.4/Zigbee protocol stack implementation. We have opted for a modular implementation i.e. the implementation is organized in different modules (NWLM, MacM and PhyM) where each module implements a protocol layer. The purpose of this modularity is to enable fast and easy extensions of our implementation by adding or updating new functionalities. Each of these modules makes use of auxiliary files used to implement some generic functions (e.g. functions for bit aggregation into variable blocks), constants declaration (e.g. layer constants), enumerations (e.g. data types, frame types, response status) and data structure definitions (e.g. frame construction data structures).

In addition, we have developed an auxiliary module - the TimerAsync module - for the implementation of an asynchronous timer based on the hardware clock (used for the implementation of the beacon interval, superframe duration, time slots and backoffs). For the synchronous timers, used in non time critical operations (e.g application layer events), we use the standard TimerC module already provided by TinyOS.



a) Protocol stack architecture



b) TinyOS implementation diagram

Fig. 5. Protocol Stack Software Architecture

The interface files (Fig. 5.a right side) are used to bind the components and represent one Service Access Point (SAP). Each of these interfaces provides functions that are called from the higher layer module and are executed/implemented in the lower layer module. The interfaces also provide functions used by the lower layer modules to signal functions that are executed/implemented in the higher layer modules (e.g. the *PD_DATA.nc* interface is used by the MacM module to transfer data to the PhyM module, that is going to be transmitted, and also enables the signalling by the PhyM in the MacM of received data).

Fig. 5b depicts the relations between different components of our IEEE 802.15.4/Zigbee protocol stack implementation. Note that some components used in our

implementation are already part of the TinyOS operating system, namely the hardware components (e.g. the HPL<…>.nc modules).
In our implementation, we did not interact directly with the hardware, in fact, TinyOS already provides hardware drivers forging a hardware abstraction layer used by our Phy component. In Fig. 5b, observe that the components highlighted in white are hardware components already provided by the TinyOS operating system.
Refer to [19] for a detailed description of the implementation functions and protocol mechanisms.

3.4 Implementation Challenges

The main problems encountered while implementing the IEEE 802.15.4/Zigbee protocol stack are related to the hardware constraints. We believe that the MICAz motes that we are using (with 8 bits microcontroller and a Chipcon CC2420 transceiver) do not provide enough processing power and radio performance for an implementation that fully complies with the IEEE 802.15.4 standard timing constraints, especially for small beacon orders (*BO<2*) and superframe orders (*SO<2*). In addition, the MICAz available memory size is rather scarce. Nevertheless, it is possible to achieve a reasonable operational behaviour with higher superframe configurations allowing the experimentation of several features of the protocol (e.g. tuning the CSMA/CA variables and other protocol parameters) and to implement new ones.

The timing requirements of the IEEE 802.15.4 protocol are quite demanding. In the beacon-enabled mode, all the devices must synchronize with the PAN Coordinator by receiving and decoding the beacon frames in order to align their superframes. If a device looses synchronization it will not be able to operate in the PAN. On the other hand, if it is not accurately synchronized with the entire PAN there is a possibility of collisions in the GTS, resulting from the overlap of the CAP with the CFP. From our experience in the implementation, the de-synchronisation can be caused by multiple factors: (1) the processing duration of beacon frames for high duty cycles, (2) the mote stack overflow that results in a freeze or a hard reset, (3) the unpredictable delays of the wireless communications, and (4) the low processing power of the microcontroller in conducting some of the protocol management tasks (e.g. creating the beacon frame, the management of GTS expiration and the indirect transmissions).

The implementation of the CSMA/CA algorithms is also demanding concerning the timer precision. In fact, the IEEE 802.15.4 protocol imposes that each backoff corresponds to 20 symbols (one symbol is equal to 4 bits), which is equivalent to 320 µs. A first difficulty in the implementation of the beacon-enabled mode was related to the TinyOS management of the hardware timers provided by the MICAz motes, which does not allow having the exact values as specified by the IEEE 802.15.4 standard.

To accomplish an accurate synchronization, we have developed a timer module (*TimerAsync*) based on the hardware clock (TinyOS *HLPTimer2C* component) with an asynchronous behaviour regarding the code execution that implements the events depicted in Fig 6. Two different types of timers have been implemented: (1) the synchronous timers, which are used in the implementation for events that do not need accuracy (2) and the asynchronous timers that are more precise due to their asynchronous behaviour.

Fig 6 depicts the asynchronous timer events that we have implemented (*TimerAsync* Module).

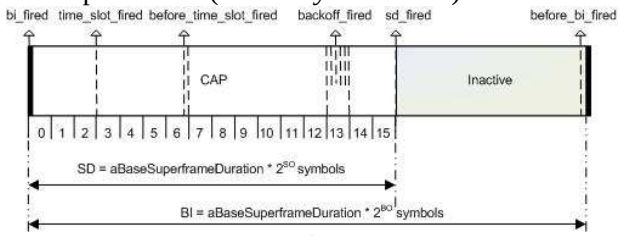


Fig. 6. Asynchronous events

The clock tick granularity (the minimum time unit of the clock) of the MICAz mote that best fits our requirements is equal to 69.54 ms, which approximately corresponds to four symbols (16 bits with 250 kbps). In fact, the four-symbol duration has a theoretical value of 64 ms which leads to a cumulative effect on the discrepancy between the experimental and the theoretically values of the beacon interval, superframe durations and time slot duration for high superframe and beacon orders. For instance, a beacon interval *BI=8* corresponds to 245760 symbols, which theoretically corresponds to 3932.160 ms, but experimentally corresponds to 4266.588 ms, based on the MICAz clock granularity. This discrepancy, however, does not impact the correct behaviour of the implemented protocol. In fact, since we are using the same mote platform for every node, we experience a coherent network behaviour.

The frequency of the asynchronous software events (as seen in Fig. 6) in addition to the hardware events, with precedence in their execution, and the low processing ability of the microprocessor, may lead to an insufficient processing power left to execute the remaining higher layer protocol tasks.

The IEEE 802.15.4 protocol does not provide any reference regarding the implementation of the buffer mechanisms (e.g. receive, transmit, GTS, indirect transmissions). The way buffer are implemented impacts the performance of the protocol implementation. On the one hand, the protocol implementation must avoid

excessive memory copy operations because it can jeopardize the synchronization (since these operations are very time consuming). On other hand, the buffers have to be small and efficiently managed because of the limitation of the device memory (the MICAz only has approximately 4 kbytes of RAM memory available and the maximum packet length is about 127 bytes; if we increase the buffer size the free memory will decrease rapidly).

Another constraint of the IEEE 802.15.4 Physical Layer is the turnaround time of 12 symbols (192 µs), the time that the transceiver takes to switch from receive mode to transmit mode, and vice-versa, to acknowledge messages. Unfortunately, this is not possible to achieve in most IEEE 802.15.4-compliant radio transceivers including the Chipcon CC2420, which can take up to 192 µs to switch between transmit and receive modes, leaving no time for data transitions between the MAC, the PHY layer and the chip transmit memory space.

Moreover, TinyOS imposes some overheads [20] in the primitive operations (e.g. posting tasks, calling commands) that is considerable in order to comply with the most demanding operational modes of the IEEE 802.15.4 protocol.

4. Research work

We have been characterizing the IEEE 802.15.4 behaviour in several research works, both via analytical and simulation tools. In this section we overview our research work in which we use our Open-ZB implementation to validate our proposals and to assess some of the current functionalities proposed in the standards. We start by evaluating the CSMA/CA mechanism of the IEEE 802.15.4 comparing the practical experiment result with theoretical result from our IEEE 802.15.4 simulation model. Also in the GTS management we have implemented an implicit Guaranteed Time Slot allocation mechanism (i-GAME) proposed in [5]. Finally we have implemented a mechanism to overcome the problem of beacon collision in cluster-tree topologies.

4.1 Evaluation of the CSMA/CA

The performance of the IEEE 802.15.4 CSMA/CA protocol was recently evaluated in [21-23], however the impact of Beacon Order (*BO*), Superframe Order (*SO*) and Backoff Exponent (*BE*) was not addressed. In order to carry out this task, we have developed a simulation model for the IEEE 802.15.4 slotted CSMA/CA mechanism using the OPNET simulator [12]. Using this model, we have analyzed the performance limits of the slotted CSMA/CA mechanism for broadcast transmissions (e.g. without acknowledgements). This was done for different network settings, in order to understand the impact of the protocol attributes (superframe order, beacon order and backoff exponent) on the network performance, namely in terms of Throughput (*S*), Average Delay (*D*) and Probability of Success (*Ps*) as presented in [24]. The evaluation of the saturation throughput and the impact of the number of nodes and frame size on the performance of slotted CSMA/CA were also addressed and the simulation results are presented in [25].

Currently, we have been using the Open-ZB implementation in the MICAz motes with the purpose of analysing the performance of the slotted CSMA/CA and comparing it with the simulation results. In general, both the simulation and experimental scenarios consist of several nodes (MICAz) generating traffic at pre-programmed inter-arrival times at the application layer and a packet analyzer capturing all the data for later processing and analysis. The packet analyzer used in the experimental evaluation process has been the Chipcon CC2420 Packet Sniffer [17]. It generates a text file with all the received packets and corresponding timestamps enabling us to retrieve all the necessary data with a parser application.

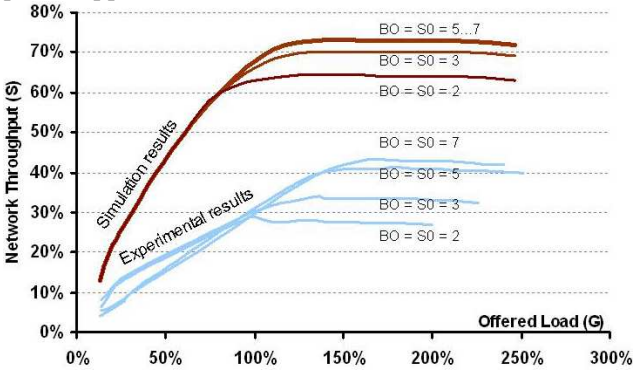


Fig. 7. The Network Throughput as a function of the Offered Load obtained through simulation and experimental work

As an example of what has already been achieved, Fig.7 presents the results obtained by simulation and experimental evaluation for the Throughput as a function of the Offered Load. The network Throughput metric represents the fraction of traffic correctly received by the network analyzer normalized to the overall capacity of the network (250 kbps). The Offered Load, represents the amount of traffic passed to the MAC layer, again normalized to the overall network capacity. Note that low *SO* values lead to lower network Throughput. This is basically due to two factors. First, the overhead of the beacon frame is more significant for lower *SO* values, since beacons are more frequent. Second, CCA deference is also more frequent in case of lower *SO* values, leading to more collisions at the start of each superframe. This behaviour is observed both in the simulation and experimental analysis. Nevertheless, the Throughput values obtained through experiment are

lower. We believe these differences are somewhat related to hardware constrains of the MICAz and some efforts are being carried out to minimize its impact in the results.

4.2 i-GAME

The IEEE 802.15.4 supports a GTS allocation, where a node explicitly allocates a number of time slots in each superframe for its exclusive use. The limitation of this mechanism is inherent to the maximum number of seven available GTS that can be allocated in each superframe, preventing other nodes to benefit from guaranteed service and resulting in a wasted bandwidth if the GTS is underutilized. The i-GAME approach is based on implicit GTS allocation requests, taking into account the traffic specifications and the delay requirements of the flows, therefore enabling the use of one GTS by several nodes, still guaranteeing that all their requirements (delay, bandwidth) are satisfied. In [5] the authors propose an admission control algorithm that decides whether to accept or reject a new GTS allocation.

The i-GAME mechanism was implemented in the MAC and Network Layers defining a new service access point between these two layers, the *MLME-iGAME*. A detailed standard-like description of the interfaces added to the Network layer and the enhancements to the MAC layer for supporting the i-GAME mechanism is presented in [26].

Comparing with the standard IEEE 802.15.4, the i-GAME mechanism in the MAC layer just needs to change the management of the beacon GTS descriptors, which have to be included in the beacon in a round robin sequence. The implicit GTS descriptors are managed by the i-GAME Admission Control by issuing the *MLME_iGAME.response*. This primitive is implemented in the MAC layer by updating the GTS descriptors (either by removing or adding). The MAC layer maintains a list with the descriptors characteristics.

The i-GAME mechanism assumes that when a node wishes to allocate a time slot, it sends an implicit GTS request command (similar to the IEEE 802.15.4 GTS request command) that besides the current IEEE 802.15.4 GTS characteristics (length, direction and type) also includes the desired flow specification, including the burst size, arrival rate and the delay requirements. The PAN Coordinator evaluates the acceptance of the GTS allocation by running the Admission Control algorithm with the requested flow specifications. The i-GAME Admission Control algorithm manages the number of necessary GTS time slots in order to comply with the requests, and accepted, flow specifications. This is accomplished by managing the GTS descriptors of the beacon frame transmitted by the PAN Coordinator allowing the nodes that allocated a GTS to use them.

Fig. 8 depicts an example of the usage of the GTS allocated time slots and the optimization of bandwidth that can be achieved with the i-GAME mechanism.

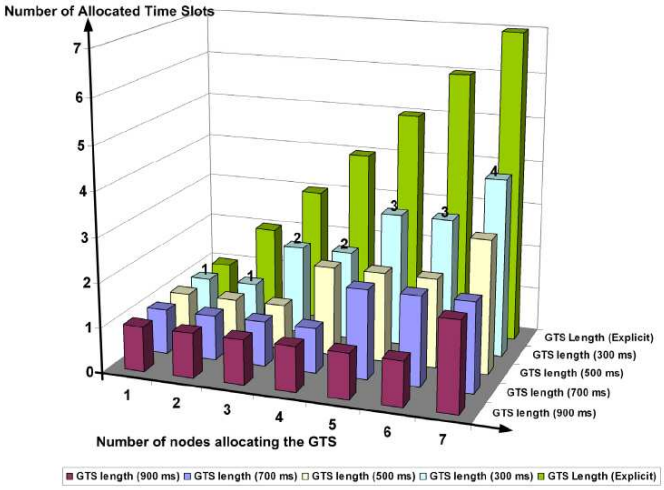


Fig. 8. Number of nodes allocating a GTS with i-GAME versus the GTS length

4.3 Time Division Beacon Scheduling

The current IEEE 802.15.4/Zigbee specifications restrict the synchronization in the beacon-enabled mode to star-based networks, while it supports multi-hop networking using the peer-to-peer mesh topology, but with no synchronization. Even though both specifications mention the possible use of cluster-tree topologies, which combine multi-hop and synchronization features, the description on how to effectively construct such a network topology is missing.

The Time Division Beacon Scheduling (TDBS) mechanism (without coordinator grouping), proposed in [27], can be implemented in a simple manner, with only minor add-ons to the protocol. In our implementation, the ZigBee Network Layer supports the network management mechanisms (e.g. association and disassociation) and the tree-routing protocol. The tree-routing relies on a distributed address assignment mechanism that provides to each potential parent (ZC and ZRs) a finite sub-block of unique network addresses based on the maximum number of children, depth and the number of routers in the PAN. The ZC is the first node in the WSN to come to life and to broadcast beacons. Every ZigBee Router (ZR), after its association to the network, temporarily acts as a ZED and must be granted permission by the ZC before assuming ZR functionality and starting sending beacon frames. All the ZRs and ZC use the same Beacon Interval (BI). Each ZR must be active both during its Superframe Duration (in the cluster under its control) and also during the active period of its parent.

The TDBS approach relies on a negotiation for beacon broadcasting. Upon success of the association to the network, the ZR (behaving as a ZED) sends a negotiation message to the ZC (routed along the tree) embedding the envisaged (BO, SO) pair requesting a beacon broadcast permit. Then, in the case of a successfully negotiation, the ZC replies with a negotiation response message containing a beacon transmission offset (the instant when the ZR starts transmitting the beacon). In case of rejection, the ZR must disassociate from the network.

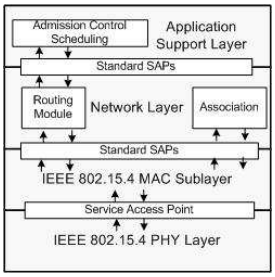


Fig. 9. TDBS Implementation Architecture

Fig. 9 depicts the architecture of the TDBS implementation in the IEEE 802.15.4/ZigBee protocol stack. The admission control algorithm is implemented in the Application Support Layer behaving as a service module of this layer. The TDBS requires minor changes to the Network Layer. Thus, it is necessary to add a *StartTime* argument in the *MLME-START.request* primitive, as already proposed in the ZigBee Specification [2], and to the *NLME-START-ROUTER.request* primitive.

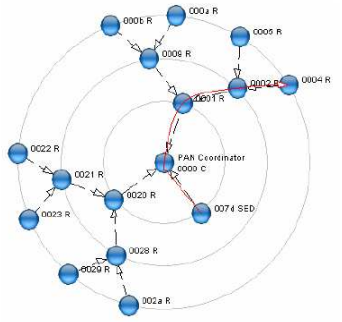


Fig. 10. Experimental network configuration

In our experimental work, we have considered the network scenario presented in Fig. 10. The cluster-tree network contains 15 cluster heads that consist of one ZC and 14 ZR. The Beacon Order (*BO*) is set to 8 for all coordinators, which gives a Beacon Interval of 245760 symbols (4266.885 ms). Hence, we must have at least $2^4=16$ Beacon/Superframe time windows, each with duration of 15360 symbols (266.680 ms). This restricts the (maximum) Superframe Order (*SO*) to 4 (i.e. Superframe Duration (*SD*) = 15360 symbols). In our experimentation, we choose a *SO=3* (*SD=7680* symbols (133.340 ms)). The cluster-tree network parameters (for setting up the tree routing mechanism) consist in a maximum depth equal to *maxDepth=3*, a maximum number of child nodes per parent router equal to *Nchild=6*, and a maximum number of child routers per parent router equal to *Nrouter=4*. As shown in Fig. 10, the network comprises the ZC at depth 0, two ZR at depth 1, four ZR at Depth 2 and eight ZR at depth 3. A ZED (0x007d) was also considered for carrying out a message routing test.

	Time Delta	Source	Destination	Packet Type
1	+00:00:04.266	0x0000	0xffff	Beacon: BO: 8, SO: 4, PC: 1, AP: 1
	+00:00:04.266	0x0000	0xffff	Beacon: BO: 8, SO: 4, PC: 1, AP: 1
2	+00:00:00.002	0x0000000200000002	0x0000	Command: Association Request
	+00:00:00.002			Acknowledgment
	+00:00:00.004	0x0000000200000002		Command: Data Request
	+00:00:00.001			Acknowledgment
	+00:00:00.002	0x0000000100000001	0x0000000200000002	Command: Association Response
	+00:00:00.002			Acknowledgment
	+00:00:04.255	0x0000	0xffff	Beacon: BO: 8, SO: 4, PC: 1, AP: 1
	+00:00:04.266	0x0000	0xffff	Beacon: BO: 8, SO: 4, PC: 1, AP: 1
	+00:00:04.266	0x0000	0xffff	Beacon: BO: 8, SO: 4, PC: 1, AP: 1
3	+00:00:00.002	0x0001	0x0000	Data
	+00:00:00.002			Acknowledgment
	+00:00:00.002	0x0000	0x0001	Data
	+00:00:00.002			Acknowledgment
	+00:00:00.269			Acknowledgment
4	+00:00:03.990	0x0000	0xffff	Beacon: BO: 8, SO: 4, PC: 1, AP: 1
	+00:00:00.276	0x0001	0xffff	Beacon: BO: 8, SO: 4, PC: 1, AP: 1
	+00:00:03.990	0x0000	0xffff	Beacon: BO: 8, SO: 4, PC: 1, AP: 1
	+00:00:00.276	0x0001	0xffff	Beacon: BO: 8, SO: 4, PC: 1, AP: 1
	+00:00:03.990	0x0000	0xffff	Beacon: BO: 8, SO: 4, PC: 1, AP: 1
	+00:00:00.275	0x0001	0xffff	Beacon: BO: 8, SO: 4, PC: 1, AP: 1

Fig. 11. Association and negotiation Example

In Fig 11, marked as 1, is the beacon broadcast of the ZC containing the network configuration BO and SO, as seen in the Packet Type field. Note that the Time Delta (4266 ms) between beacons represents the beacon interval. The sequence of messages marked as 2 represents the association procedure. The ZR with the extended address of 0x0000000200000002 sends an association request to the ZC (0x0000). The ZC acknowledges the reception of the request and informs the ZR that there is pending data (using the pending data field in the acknowledge frame). Then, the ZR sends a data request command frame requesting the pending data. The ZC replies with the association response command frame containing the status of the association (that in this case is successful) and the ZR is assigned the short address 0x0001.

Now, the ZR is associated as a ZED and can therefore communicate in the network, but it still needs to request the ZC for a beacon broadcast transmission permit and a time window slot (transmission offset). The negotiation procedure is marked as 3. Until this point, and after the network association, the ZR behaves as a normal ZED. When the negotiation for beacon transmission finishes, the ZR starts to broadcast beacons in its assigned time window, as seen in Fig 11 marked as 4. Note that both the association and negotiation for beacon transmission took place during the ZC superframe.

In Fig 12, marked as 1, the first transmission of the packet from the ZED 0x002d to its parent (ZR 0x0028) is shown. Note that this transmission is carried out during ZR 0x0028 superframe. The routing of the data frame from the ZR (0x0028) to its parent in the cluster-tree (ZR 0x0020) is marked as 2. The multi-hop continues (Fig.12-3) with the routing of the frame from the ZR 0x0020 to the ZC (0x0000) and to ZR 0x005e.

This transmission sequence is carried out during the ZC superframe. Then, ZR 0x005e routes the frame to its final destination, the ZR 0x0066 (Fig.12-4). The retransmission of the data frame (Fig.12-4) is due to the failure of the acknowledge transmission of ZR 0x0066.

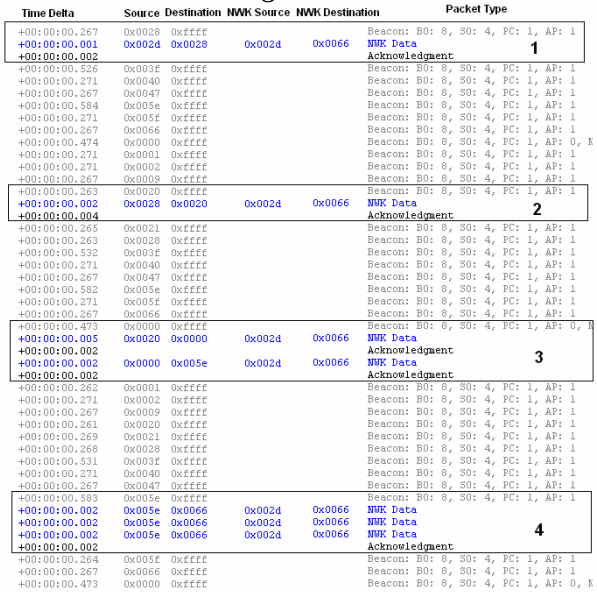


Fig. 12. Message Flow and Beacon Frames

5. Concluding Remarks

IEEE 802.15.4/ZigBee emerge as potential technologies for Wireless Sensor Networks. Thus, it is of paramount importance to analyse their adequateness for fulfilling the requirements of large-scale ubiquitous computing applications. In this context, we have triggered the ART-WiSe research line, which aims at the design of a communication architecture for large-scale critical applications based on COTS technologies, namely IEEE 802.15.4/ZigBee. For that purpose we have developed our own implementation of the protocol stack, which we are making available to the community as open-source. This has already triggered several relevant interactions with world-reputed researchers, companies and normalization bodies.

This paper overviews the most important aspects of the implemented software, as well as a number of research work that builds on its use.

References

[1] IEEE-TG15.4, "Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)," IEEE standard for Information Technology, 2003.

[2] A. Koubâa, M. Alves, and E. Tovar, "IEEE 802.15.4: a Federating Communication Protocol for Time-Sensitive Wireless Sensor Networks," in Sensor Networks and Configurations: Fundamentals, Techniques, Platforms, and Experiments, N. H. Mahalik, Ed., 2007.

[3] ZigBee Specification 2006, http://www.zigbee.org/

[4] A. Koubâa, M. Alves, and E. Tovar, "GTS Allocation Analysis in IEEE 802.15.4 for Real-Time Wireless Sensor Networks," in 14th International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS 2006). Rhodes Island (Greece): IEEE, 2006.

[5] A. Koubâa, M. Alves, and E. Tovar, "i-GAME: An Implicit GTS Allocation Mechanism in IEEE 802.15.4," in Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS 2006), 2006.

[6] J. Misic and V. B. Misic, "Access delay for nodes with finite buffers in IEEE 802.15.4 beacon enabled PAN with uplink transmissions," Computer Communications, vol. 28, pp. 1152-1166, 2005.

[7] J. Misic, S. Shafi, and V. B. Misic, "Modeling a beacon enabled 802.15.4 cluster with bidirectional traffic," Lecture Notes in Computer Science, vol. 3462, pp. 228-239, 2005.

[8] L. Hwang, "Grouping Strategy for Solving Hidden Node Problem in IEEE 802.15.4 LR-WPAN," in 1st International Conference on Wireless Internet (WICON'05). Budapest (HUNGARY): IEEE, 2005.

[9] A. Koubaa, M. Alves, and E. Tovar, "Modeling and Worst-Case Dimensioning of Cluster-Tree Wireless Sensor Networks," in Proceedings of the 27th IEEE Real-Time Systems Symposium (RTSS'06), Rio de Janeiro (Brazil), 2006.

[10] Open-ZB - Open Source Toolset for IEEE 802.15.4 and ZigBee. http://www.open-zb.net

[11] Crossbow, "MICAz datasheet," http://www.xbow.com, 2004.

[12] OPNET, "OPNET Simulator, v 11, http://www.opnet.com."

[13] The ART-WiSe Framework, www.hurray.isep.ipp.pt/art-wise/

[14] TinyOS, http://www.tinyos.net, 2007.

[15] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler, "the nesC Language: A Holistic Approach to Networked Embedded Systems," in Proceedings of the Programming Language Design and Implementation, 2003.

[16] Chipcon, "CC2420 transceiver datasheet," http://www.chipcon.com, 2004.

[17] Chipcon, "Chipcon Packet Sniffer for IEEE 802.15.4," 2006.

[18] Daintree Networks, "Sensor Network Analyser," www.daintree.net, 2006.

[19] A. Cunha, M. Alves, and A. Koubaa, "An IEEE 802.15.4 protocol implementation (in nesC/TinyOS): Reference Guide v1.1," IPP-HURRAY Technical Report, http://www.open-zb.net 2006.

[20] J. Hill, R. Szewczyk, A.Woo, S. Hollar, D. Culler, K. Pister, “System Architecture Directions for Networked Sensors”, ASPLOS 2000, Cambridge, November 2000.

[21] J. Mišic, and V. B. Mišic, "Duty Cycle Management in Sensor Networks Based on 802.15.4 Beacon Enabled MAC", Ad Hoc and Sensor Wireless Networks Journal, Old City Publishing, 1(3):207-233, 2005.

[22] J. Mišic and V. B. Mišic, "Access Delay and Throughput for Uplink Transmissions in IEEE 802.15.4 PAN", Elsevier Computer Communications Journal, 28(10):1152-1166, Jun. 2005.

[23] J. Mišic, S. Shafi, and V. B. Mišic, "The Impact of MAC Parameters on the Performance of 802.15.4 PAN", Elsevier Ad hoc Networks Journal, 3(5):509–528, 2005.

[24]A. Koubaa, M. Alves, E. Tovar, “A Comprehensive Simulation Study of Slotted CSMA/CA for IEEE 802.15.4 Wireless Sensor Networks”, In IEEE WFCS 2006, Torino (Italy), June 2006.

[25] A. Koubâa, M. Alves, E. Tovar, "On the Performance Limits of Slotted CSMA/CA in IEEE 802.15.4 for Broadcast Transmissions in Wireless Sensor Networks", IPP-HURRAY Technical Report, HURRAY-TR-060202, Feb. 2006.

[26] A. Cunha, A. Koubâa, and M. Alves, "Implementation of the i-GAME Mechanism in IEEE 802.15.4 WPANs," IPPHURRAY Technical Report, TR060702, July 2006.

[27] Anis KOUBAA, Andre CUNHA, Mário ALVES, “A Time Division Beacon Scheduling Mechanism for IEEE 802.15.4/Zigbee Cluster-Tree Wireless Sensor Networks”. to be presented in Euromicro Conference on Real-Time Systems (ECRTS 2007), Pisa(Italy), July 2007