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, School of Engineering (ISEP/IPP), 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 and TelosB 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 problems that we have faced during the implementation of the protocol stack. 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-WPANs) [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 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.

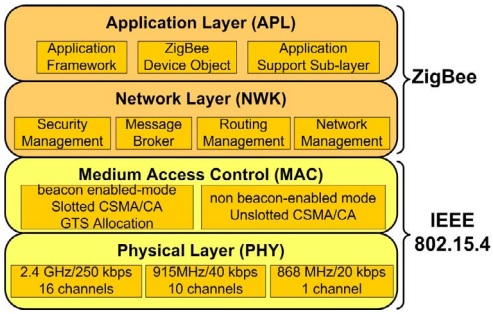


Fig. 1. The IEEE 802.15.4/ZigBee protocol stack architecture

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 results 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 these theoretical results, since simulation tools are usually not sufficient to evaluate the real behaviour of the protocols due to many abstractions in the simulation models.

Therefore, 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 networks.

In this paper, we propose Open-ZB [10], an open-source implementation of the IEEE 802.15.4/ZigBee protocol stack under the TinyOS operating system. Currently, the implementation supports the MICAz [11] and TelosB 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 [12,13] using the OPNET tool [14]. This simulation model implements the Physical and the Data Link 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.

The Open-ZB implementation was developed in the context of the ART-WiSe research framework [15], which consists in providing real-time and reliable communications 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 the IEEE and with the ZigBee Alliance in a way that our findings can contribute for improving the current protocol standards.

The main contribution of this 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 presented in Section 3, namely general aspects of our development environment, a short overview of TinyOS [16] and nesC [17] programming language, the implementation structure and future challenges. In Section 4 we outline some research achievements based on our implementation. Finally, in Section 5, we draw some concluding remarks.

2. IEEE 802.15.4/ZigBee Overview

2.1. IEEE 802.15.4 Physical and Data Link Layers

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 must be associated to a PAN Coordinator and does not create its own network; (3) the End-Devices: do not implement the previous functionalities and must 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. A RFD is intended for supporting simple tasks, such as a light switch or a passive infra-red sensor; they do not have the need to send large amounts of data and can 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 1 channel between 868 and 868.6 MHz, 10 channels between 902 and 928 MHz, and 16 channels between 2.4 and 2.4835 GHz. 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 these frequency bands are based on the Direct Sequence Spread Spectrum (DSSS) spreading technique.

The Physical Layer also allows dynamic channel selection, a channel scan, receiver energy detection, link quality indication and channel switching.

The IEEE 802.15.4 protocol supports two operational modes that may be selected by the PAN Coordinator: (1) the *non beacon-enabled* mode, in which the Medium Access Control (MAC) is simply ruled by Unslotted CSMA/CA, (2) the *beacon-enabled* mode, in which beacons are periodically sent by the Coordinators to synchronize nodes that are associated with them, 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 of the *BI*, being divided into 16 equally-sized time slots, during which frame transmissions are allowed. Optionally, an inactive period can be 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:

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

where *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, during the Contention-Access Period (CAP).

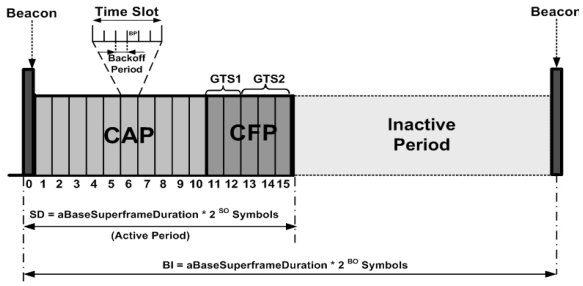
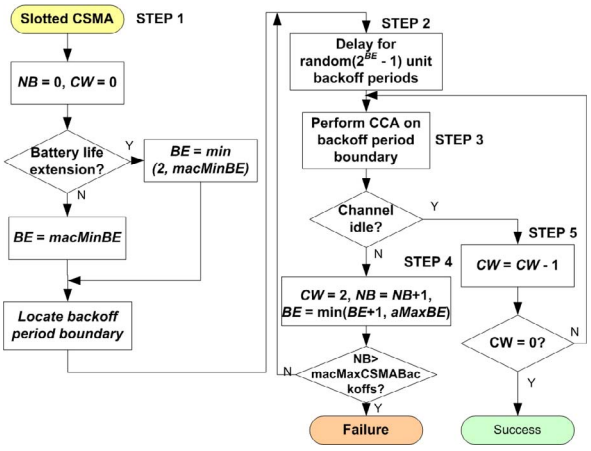


Fig. 2. IEEE 802.15.4 Superframe Structure

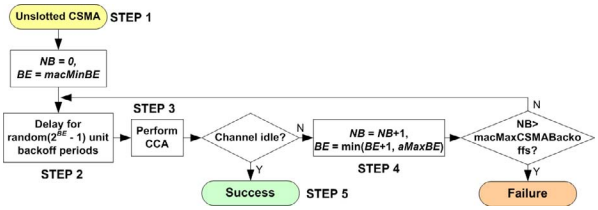
As depicted in Fig. 2, low duty cycles can be configured by setting small values of *SO* as compared to *BO*, resulting in greater inactive periods. Additionally, the IEEE 802.15.4 protocol also provides real-time guarantees by using the Guaranteed-Time Slot (GTS) mechanism. This feature is quite attractive for time-sensitive WSN applications. In fact, 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.

The GTS mechanism allows a device to access the medium without contention, during the Contention-Free Period (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 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.



a) Slotted CSMA/CA



b) Unslotted CSMA/CA

Fig. 3. The CSMA/CA mechanism

IEEE 802.15.4 supports two contention-access MAC mechanisms (Fig. 3) based on Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA): Slotted CSMA/CA (Fig. 3a) in the beacon-enabled mode and Unslotted CSMA/CA (Fig. 3b) in the 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.

The Slotted CSMA/CA algorithm (Fig. 3a) 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 sub-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 Assessment (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.

The unslotted 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.

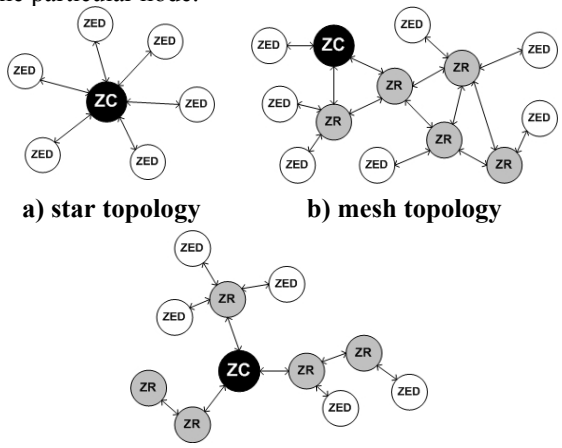
2.2. The ZigBee Network Layer

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 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 and acronyms of the devices are used interchangeably.

IEEE 802.15.4/ZigBee enable three network topologies – star, mesh and cluster-tree. In all cases, 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. In the *star topology* (Fig.4a), the communication paradigm 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 WSN 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 leading to a scalability problem.

In the *mesh topology* (Fig. 4b) the communication paradigm 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 hop routing from any node to any other node. The mesh topology may be more power-efficient than the star topology since communications do not rely on one particular node.



a) star topology **b) mesh topology**

c) cluster-tree topology

Fig. 4. IEEE 802.15.4 network topologies

The *cluster-tree 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 (Coordinator per cluster. Any FFD can act as a ZR and provide synchronization services to other devices and ZRs.

3. Open-ZB Protocol Stack

3.1. Mote Platforms

The Open-ZB [10] implementation was developed under the TinyOS operating system version 1.1.15.

The first version was developed for the MICAz [11] mote platform. The current version also supports the TelosB [11] mote platform. The TelosB architecture is slightly different from the one of the MICAz, especially due to the 16-bits MSP430 microcontroller [18] as compared to the MICAz 8-bits Atmega128 microcontroller [19]. This triggers the need for a selection of the hardware files (or drivers) already provided in TinyOS and to an adaptation of the previous version of the implementation to support the 16-bits memory block of the MSP430. Both platforms use the same 2.4 GHz Chipcon CC2420 radio transceiver [20].

The MICAz mote needs an interface to program it (the MIB510), while the TelosB mote features an USB

interface. Both motes provide a debug mechanism by sending data through the serial (COM/USB) port and reading it in a communication listener (e.g. ListenRaw, provided with the TinyOS distribution, or Windows HyperTerminal). This debugging 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 usually causes synchronization problems.

In order to overcome the COM debugging problems, we have used network/protocol analysers to track and display the packets being transmitted, which provide a better debugging mechanism by transmitting debugging data in the packet payloads.

We have used an IEEE 802.15.4/ZigBee packet sniffer provided by Chipcon - the CC2420 Packet Sniffer for IEEE 802.15.4 v1.0 [21] 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 [22] that provides additional functionalities, such as graphical topology of the network, statistics, message flows, PAN information and association details.

3.2. TinyOS and nesC

TinyOS [16] is an operating system for embedded systems with an event-driven execution model. TinyOS is developed in nesC [17], a language for programming structured component-based applications. nesC has a C-like syntax and is designed to express the structuring concepts of TinyOS. This includes the concurrency model, mechanisms for structuring, naming and linking together software components into embedded system applications. The component-based application structure provides flexibility to the application design and development.

nesC applications are built out of components and interfaces. The components define two target areas: (1) *the specification*, a code block that declares the functions it provides (*implements*), and the functions that it uses (*calls*); (2) the implementation of the functions provided. The interfaces are bidirectional collections of functions provided or used by a component. The interfaces commands are implemented by the providing component, and the interface events are implemented by the component using it. The components are "wired" together by means of interfaces, forming an application.

TinyOS defines a concurrency model based on tasks and hardware events handlers/interrupts. 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.

3.3. Software architecture

The Open-ZB protocol stack implementation has three main blocks: (1) the hardware abstraction layer, including the IEEE 802.15.4 physical layer and the timer module supporting both MICAz and TelosB mote platforms; (2) the IEEE 802.15.4 MAC sub-layer; and (3) the ZigBee Network Layer.

The functionalities supported by the IEEE 802.15.4 implementation include the slotted version of the CSMA/CA algorithm, allowing the testing and parameterization of its variables, the different types of transmission scenarios (e.g. direct, indirect and GTS transmissions), association of the devices, channel scans (e.g. energy detection and passive scan) and beacon management. Other IEEE 802.15.4 features were left out of this implementation version because they were not needed for the works reported in Section 4. Features that are not currently supported include the unslotted version of the CSMA/CA, the active and orphan channel scan and the use of extended addressing fields in normal data transmissions.

In the ZigBee Network Layer, the currently supported features comprise the data transfer between the Network Layer and the MAC sub-layer, the association mechanisms and the network topology management (e.g. cluster-tree support by the ZigBee Addressing schemes) and routing (e.g. neighbour routing and tree-routing). Security is not supported yet.

The Open-ZB implementation has three main TinyOS components: the *Phy*, the *Mac* and the *NWL* components (Fig. 5).

The organization in components enables an easy and fast development of adaptations/extensions to the current implementation. Each of these components makes use of auxiliary files to implement some generic functions (e.g. functions for bit aggregation into variable blocks), declaration of protocol constants, enumerations (e.g. data types, frame types, response status) and data structure definitions (e.g. frames).

The interface files (Fig. 5a right side) are used to bind the components and represent one Service Access Point (SAP), providing functions that are called from the higher layer components and are executed/implemented in the lower layer module. The interfaces also provide functions used by the lower layer components to signal functions that are executed/implemented in the higher layer components. For example, 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.

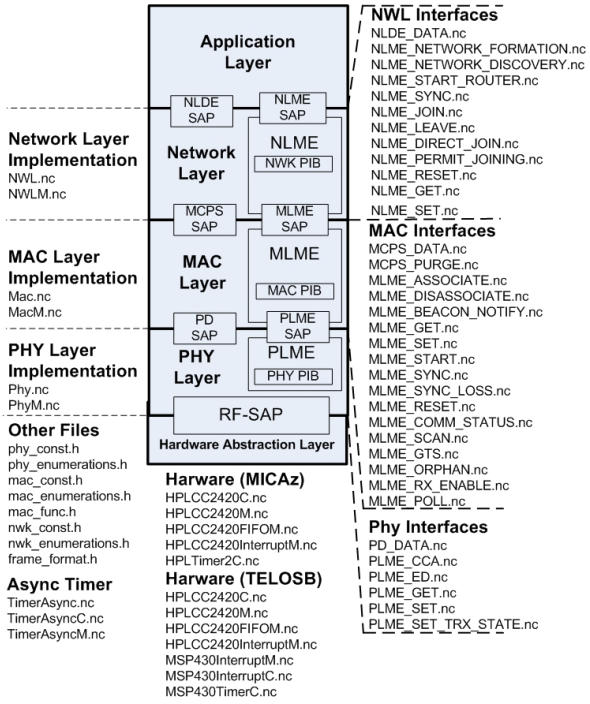
Fig. 5b depicts the relations between the different components of the protocol stack implementation. In this implementation, there is no direct interaction with the hardware, since TinyOS already provides hardware drivers forging a hardware abstraction layer used by the *Phy* component. In Fig. 5b, observe that the components filled in white are hardware components already provided by the TinyOS operating system (e.g. the HPL<…>.nc and the MSP430<…>.nc modules).

The *Mac* and the *NWL* components are shared by the two platforms (MICAz and TelosB). However, there are two different *Phy* components, one for each platform. At compilation time, the *Phy* component is selected according to the envisaged platform. The need of two different *Phy* components is due to the fact that the TinyOS hardware specific modules are different for each platform.

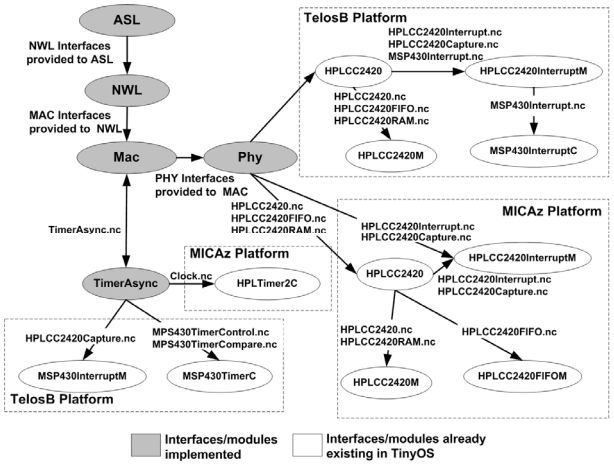
In addition, both platforms differ in the hardware timers they provide, leading to two different timer modules (the *TimerAsync*) with the purpose of managing all asynchronous timer events of the MAC sub-layer (e.g. Beacon Interval, Superframe Duration, time slots and backoff periods). For the synchronous timers, used in non time critical operations (e.g. application layer events), we use the standard *TimerC* module already provided by TinyOS. Nevertheless, the software architecture is the same for both platforms.

The MICAz hardware clock (provides a frequency of 7.3728 MHz) is implemented in the *HPLTimer2C* (as seen in Fig. 5b) component, already provided in TinyOS, and is defined by two parameters: the *SCALE* that defines the scale division of the timer frequency and the *INTERVAL* defining the number of ticks per clock firing.

The clock tick granularity of the MICAz mote that best fits the implementation requirements is equal to 69.44 µs, which approximately corresponds to four symbols (configuration with *SCALE* equal to 4 and *INTERVAL* equal to 1), assuming a 250 kbps bit rate. The 69.44 µs is achieved by dividing the clock frequency by 256 (*SCALE* of 4) resulting in a frequency of 28.8 kHz (approximately 34.72 µs) and 2 interval counts (*INTERVAL* of 1) resulting in a clock tick every 69.44 µs. This value corresponds to the duration of four symbols (16 bits) and is a fair approximation to the theoretical value of 64 µs. However, it still leads to a cumulative effect on the discrepancy with the theoretical values of Beacon Intervals, Superframe Durations and time slot durations, especially for high Superframe and Beacon Orders. For instance, the theoretical Superframe duration with *SO = 3* is equal to 122.88 ms, while it is equal to 133.36 ms using the MICAz motes and the TinyOS time management of the clock granularity.



a) Protocol stack architecture



b) TinyOS implementation diagram

Fig. 5. Protocol Stack Software Architecture

The hardware timer available in the TelosB is based on a 32768 Hz clock that fires approximately every 30.5 µs. Comparing with the MICAz timer, this does not allow the set of a scale or interval parameters. Instead, this is a continuous timer that counts from *0* to *0xFFFF* and when it overflows it triggers an interrupt and starts again from *0*. The only parameterization allowed is the number of overflow count before

issuing the interrupt. The TelosB hardware clock is implemented in the *MSP430TimerC* module (as seen in Fig. 5b), already provided in TinyOS. The *HPLCC2420InterruptM* module implements the timer fired interrupt as well as all the other hardware interrupts. The solution that best fits ours requirements is to trigger the timer on every backoff. The IEEE 802.15.4 defines that one backoff is 20 symbols, that theoretically corresponds to 16 µs. With this timer granularity, the value obtained for each symbol is approximately 16.775 µs, leading to a backoff period duration of 335.5 µs instead of the theoretical 320 µs.

Refer to extended technical reports in [23, 24] for a detailed description of the implementation functions, variables and protocol mechanisms.

3.4. Implementation challenges

The main challenges encountered while implementing the IEEE 802.15.4/ZigBee protocol stack were related to hardware specificities and constraints. In that aspect, the MICAz motes revealed to be more limited that the TelosB. Nevertheless, none of them provides 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 ($SO < 2$). Additionally, the available memory size is rather scarce. However, it is possible to achieve a reasonable operational behaviour for higher beacon orders.

The timing requirements of the IEEE 802.15.4 protocol are very demanding. In the beacon-enabled mode, all the devices must synchronize with a Coordinator beacon frame in order to align their Superframe. If a device loses synchronization it cannot operate in the PAN; and if it is not correctly synchronized there is a possibility of collisions in the GTS slots (if the CAP overlaps the CFP). As experienced during this implementation, the loss of synchronization can be caused by multiple factors: (1) the processing of the beacon frame for small *SO/BO* configurations; (2) the mote stack overflow that results in a block or a hard reset; (3) the unpredictable delay of the wireless communications; and (4) the low processing power of the microcontroller in conducting some of the protocol maintenance tasks (e.g. creating the beacon frame, the maintenance of GTS expiration and the indirect transmissions).

The implementation of the CSMA/CA algorithm is also very demanding in terms of timers precision, since the IEEE 802.15.4 protocol defines that each backoff corresponds to 20 symbols (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 motes, which does not allow having the exact values of the Beacon Interval, Superframe, time slots and backoffs durations as specified by the IEEE 802.15.4 standard. This discrepancy, however, does not impact the correct behaviour of the implemented protocol. If the same mote platforms are used in the experiments (at least as ZC and ZRs), it is possible to experience a coherent network behaviour.

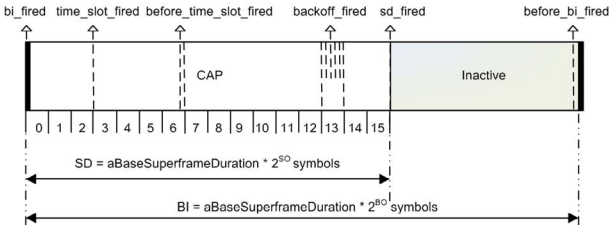


Fig. 6. Asynchronous events

The frequency of the asynchronous software events (Fig. 6) together with the hardware events and the microprocessor processing capacity may lead to an insufficient remaining processing power to execute protocol and high level application tasks as a great amount of interrupts have to be processed in short periods of time.

The IEEE 802.15.4 protocol has no reference concerning the implementation of the buffer mechanisms, which impacts on the correct behaviour of the protocol. On the one hand, the protocol must avoid excessive memory copy operations because they may cause synchronization problems and are very time consuming. On the other hand, the buffers have to be small and very well managed due to the devices memory constraints.

Another constraint of the IEEE 802.15.4 Physical Layer is the turnaround time of 12 symbols (192 µs), the time that the CC2420 radio 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. For instance, the Chipcon CC2420 can take up to 192 µs to switch between these two modes, leaving no time for data transitions between the MAC sub-layer, the PHY layer and the chip transmit memory space.

Also, TinyOS imposes some overhead [25] in the primitive operations (e.g. posting tasks, calling commands) that may be considerable, taking into account the need to comply with the most demanding operational modes of the IEEE 802.15.4 protocol.

In spite of having no comparison between this TinyOS implementation and others, it is possible to

assume that an implementation without any base operating system (OS) could have better performance results, since TinyOS can introduce some unnecessary processing overhead in its internal operations. There is an obvious trade-off between the benefits of using an OS, bringing in several functionalities that enable a faster development of high end applications and the processing overhead introduced. Considering that the embedded devices have limited resources, it is reasonable to assume that a non-OS based implementation can be more optimized but not so flexible.

Nevertheless, this implementation still has space for extensions and improvements, as it it is envisaged to implement the full functionalities of the IEEE 802.15.4 and the ZigBee Network Layer. Also, we aim at the migration of our protocol stack from TinyOS v1.15 to v2.0, in collaboration with the TinyOS Network Protocol Working Group [26].

As a final remark about the evolution between the 2004 and 2006 ZigBee specification, several issues were corrected while others were added introducing more complexity but with the advantage of adding more flexibility. The mesh network topology evolved with the addition of new functionalities, such as the possibility of multicast transmissions and a source route routing protocol, while the cluster-tree synchronized topology was left behind. Hence, there are still many open-issues in the ZigBee standard that leaves room for improvement, especially for cluster-tree networks.

4. Research Work

We have been characterizing and improving the IEEE 802.15.4 behaviour in several research works, via analytical simulation and experimental work. In this section, we present a brief overview on our research work where we have used our Open-ZB implementation to validate our proposals and to assess some of the current functionalities proposed in the standards. In Section 4.1, we start by evaluating the Slotted CSMA/CA mechanism comparing experimental and simulation results from the IEEE 802.15.4 simulation model [12, 13]. In addition, concerning GTS management, we show how we have implemented an implicit Guaranteed Time Slot allocation mechanism (i-GAME) proposed in [5], and some general results (Section 4.2). Finally, we outline how we have implemented and validated a mechanism to overcome the problem of beacon frame collisions in cluster-tree topologies (Section 4.3).

4.1 Evaluation of Slotted CSMA/CA

The performance of the IEEE 802.15.4 Slotted CSMA/CA mechanism is evaluated with the purpose of testing and validating the effectiveness of the hardware devices and the implemented CSMA/CA mechanism.

In order to accomplish this evaluation an OPNET [14] simulation model [12,13] for the IEEE 802.15.4 supporting the slotted CSMA/CA mechanism was used as a means to compare experimental and simulation results, for the same scenarios. Using this model, it was possible to analyse the performance limits of the Slotted CSMA/CA mechanism for broadcast transmissions (without acknowledgements). The analysis was done for different network settings, in order to understand the impact of some protocol parameters on the network performance, namely in terms of Network Throughput (*S*) and Probability of Successful transmissions (*Ps*), given a certain Offered Load (*G*). The performance metrics analysed are based on an extensive study of the Slotted CSMA/CA presented in [27, 28].

Recently, we have been using the Open-ZB implementation in the MICAz motes with the purpose of analysing the performance of the Slotted CSMA/CA mechanism and comparing it with the simulation results. In general, both the simulation and experimental scenarios (with 100 seconds duration) consisted in 10 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 was the Chipcon CC2420 Packet Sniffer [21]. It generates a text file with all the received packets and the corresponding timestamps, enabling to retrieve all the necessary data, (embedded in the packets payload), with a parser application, in order to avoid serial communications.

As an example of what has already been achieved, Fig.7 presents some elucidative results obtained by simulation and experimental evaluation for the Network Throughput (*S*) and Success Probability (*Ps*) as a function of the Offered Load (*G*), considering the case of one experiment where $SO = BO = 5$.

The Network Throughput (*S*) represents the fraction of traffic correctly received by the network analyzer normalized to the overall capacity of the network (250 kbps). The Success Probability (*Ps*) reflects the degree of reliability achieved by the network for successful transmissions. This metric is computed as the throughput *S* divided by *G,* representing the amount of

traffic passed to the MAC sub-layer, again normalized to the overall network capacity.

Although Fig. 7 only presents the experiments for $S0 = BO = 5$, several SO configurations have been analysed.

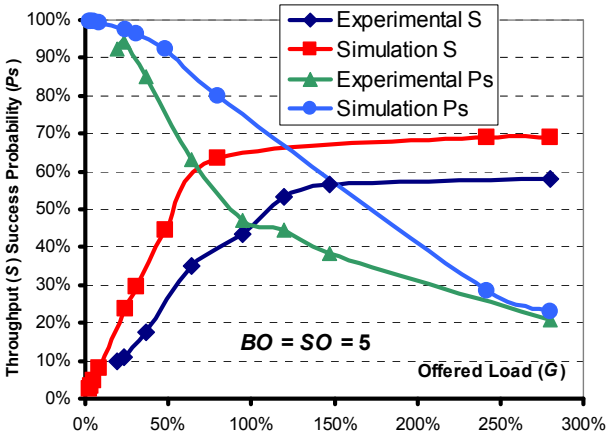


Fig. 7. Network Throughput function of Offered Load obtained (simulation & experimentation)

Simulation and experimental results allowed observing similar behaviours, which consolidates the consistency of the implemented version of the Slotted CSMA/CA mechanism.

As it could be expected, the simulation results for Throughput and Probability of Success are higher that the experimental results. We believe that this is mainly because the simulation model does not consider some of the physical constraints of the MICAz mote, especially the processing power, the internal delays due to TinyOS overheads and the normal interferences of a real wireless medium.

One of the reasons for a lower performance with lower *SO* is due to an increased probability of transmission deference (e.g. frames that are deferred to the next Superframe because the device is not able to send them in the current one). The transmission deference problem is more frequent with lower *SO*, since the *SD* is smaller. Another factor for the lower performance is the overhead of the beacon frame transmission, which is more significant for lower *SO* values.

4.2. Implicit GTS allocation mechanism

The IEEE 802.15.4 supports a GTS allocation mechanism, 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 also to a wasted bandwidth, if GTSs are underutilized.

The i-GAME approach [5] 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 each GTS by several nodes, still guaranteeing that all their requirements (delay, bandwidth) are satisfied. In [5], we have proposed an admission control algorithm that decides whether to accept or reject a new GTS allocation based on its requirement and traffic specifications.

The i-GAME mechanism was implemented in the MAC and Network Layers, defining a new service access point between these two layers - the *MLME-i-GAME*. A detailed standard-like description of the interfaces added to the Network layer and the enhancements to the MAC sub-layer for supporting the i-GAME mechanism is presented in [29].

Comparing with the standard IEEE 802.15.4, the i-GAME mechanism 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 procedure by issuing the *MLME_iGAME.response*, which updates the GTS descriptors.

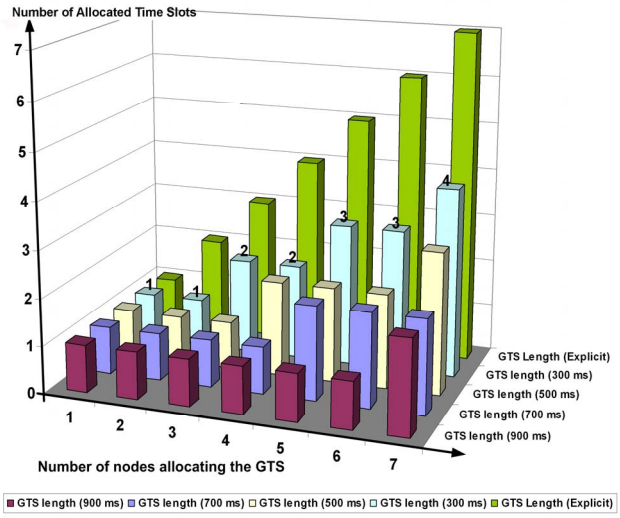


Fig. 8. Number of nodes allocating a GTS with i-GAME versus the GTS length

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 includes the desired flow specification (the burst size, arrival rate and the delay requirements) in addition to the standard IEEE 802.15.4 GTS characteristics (direction and type). 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 requested 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 [5] 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. To observe the impact of the delay requirement on the improvement of the GTS efficiency, we have run the experimental scenario with delay requirements of 900 ms, 700 ms, 500 ms and 300 ms (Fig. 8). Observe that relaxing the delay bound of 7 nodes (to 900 ms) requesting GTS allocation enables to save up to 5 time slots as compared to explicit allocation, while still satisfying the delay bounds. This (saved) time can be used to extend the Contention-Access Period, thus improving the utilization of the network.

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 supporting multi-hop networking using the 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 [30], 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 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 *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 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.

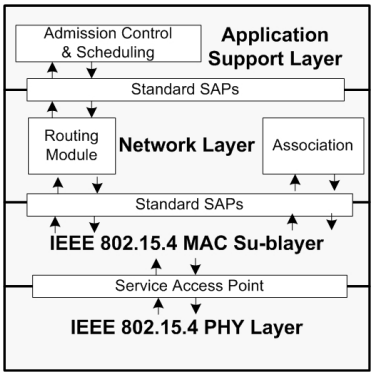


Fig. 9. TDBS Implementation Architecture

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.

In this example scenario, the cluster-tree network contains 15 clusters that consist of one ZC and 14 ZRs (all TelosB motes). The *BO* is set to 8 for all Coordinators, which gives a *BI* of 245760 symbols (≅ 4122.624 ms). Since the, we must have at least 16 Beacon/Superframe time windows, each with duration of 15360 symbols (≅ 257.664 ms). This restricts the (maximum) *SO* to 4 (i.e. Superframe Duration (*SD*) = 15360 symbols). In our experimentation, we choose a *SO* = 4 (*SD* = 15360 symbols (≅ 257.664 ms)). The cluster-tree network parameters (for setting up the tree routing mechanism) consist in a maximum depth equal to *Lm=3*, a maximum number of child nodes per parent router equal to *Cm=6*, and a maximum number of child routers per parent router equal to *Rm=4*. As shown in Fig. 10, the network comprises the ZC at depth 0, two ZRs at depth 1, four ZRs at Depth 2 and

eight ZRs at depth 3. The ZED (0x0400) was also considered for performing out a message routing test.

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* (4150 ms) between beacons represents the Beacon Interval.

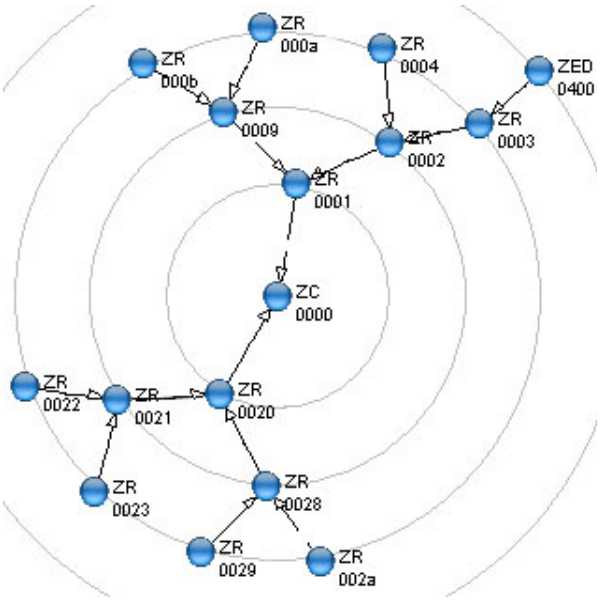


Fig. 10. Experimental network configuration

The sequence of messages marked as 2 (Fig.11) represents the association procedure. The ZR with the extended address of 0x0000000200000002 sends an association request to the ZC (0x0000), which acknowledges the reception and informs the ZR of pending data. Then, the ZR sends a data request command frame requesting the pending data and the ZC replies with the association response command frame containing the status of the association, where the ZR is assigned the short address 0x0001.

	Time Delta	Source	Destination	Packet Type
1	+00:00:04.150	0x0000	0xffff	Beacon: BO: 8, SO: 4, PC: 1, AP: 1
	+00:00:04.150	0x0000	0xffff	Beacon: BO: 8, SO: 4, PC: 1, AP: 1
2	+00:00:00.084	0x0000000200000002	0x0000	Command: Association Request
	+00:00:00.008			Acknowledgment
	+00:00:00.008	0x0000000200000002		Command: Data Request
	+00:00:00.008			Acknowledgment
	+00:00:00.002	0x0000000100000001	0x0000000200000002	Command: Association Response
	+00:00:00.869			Acknowledgment
	+00:00:04.050	0x0000	0xffff	Beacon: BO: 8, SO: 4, PC: 1, AP: 1
	+00:00:04.150	0x0000	0xffff	Beacon: BO: 8, SO: 4, PC: 1, AP: 1
	+00:00:04.150	0x0000	0xffff	Beacon: BO: 8, SO: 4, PC: 1, AP: 1
3	+00:00:00.028	0x0001	0x0000	NWK Data
	+00:00:00.008			Acknowledgment
	+00:00:00.095	0x0000	0x0001	NWK Data
	+00:00:00.008			Acknowledgment
	+00:00:00.355			Acknowledgment
4	+00:00:03.766	0x0000	0xffff	Beacon: BO: 8, SO: 4, PC: 1, AP: 1
	+00:00:00.293	0x0001	0xffff	Beacon: BO: 8, SO: 4, PC: 1, AP: 1
	+00:00:03.863	0x0000	0xffff	Beacon: BO: 8, SO: 4, PC: 1, AP: 1
	+00:00:00.292	0x0001	0xffff	Beacon: BO: 8, SO: 4, PC: 1, AP: 1
	+00:00:03.863	0x0000	0xffff	Beacon: BO: 8, SO: 4, PC: 1, AP: 1
	+00:00:00.292	0x0001	0xffff	Beacon: BO: 8, SO: 4, PC: 1, AP: 1
	+00:00:03.863	0x0000	0xffff	Beacon: BO: 8, SO: 4, PC: 1, AP: 1

Fig. 11. Association and negotiation Example

Now, the ZigBee Router is associated as a ZigBee End Device and can therefore communicate in the network, but it still needs to request the ZigBee Coordinator for a beacon broadcast transmission permit and a time window (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.

Time	Time Delta	Source	Destination	NWK Source	NWK Destination	Protocol	Packet Details	
16:53:43.940	+00:00:00.287	0x002a	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
16:53:44.136	+00:00:00.196	0x0000	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
16:53:44.428	+00:00:00.292	0x0001	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
16:53:44.720	+00:00:00.292	0x0002	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
16:53:44.977	+00:00:00.256	0x0400	0x0003	0x0400	0x0029	Zigbee NWK	NWK Data	
16:53:44.985	+00:00:00.009					IEEE 802.15.4	Acknowledgment	A1
16:53:45.076	+00:00:00.090	0x0003	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
16:53:45.311	+00:00:00.235	0x0004	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
16:53:45.594	+00:00:00.283	0x0009	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
16:53:45.886	+00:00:00.292	0x000a	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
16:53:46.173	+00:00:00.287	0x000b	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
16:53:46.450	+00:00:00.277	0x0020	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
16:53:46.742	+00:00:00.292	0x0021	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
16:53:47.034	+00:00:00.293	0x0022	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
16:53:47.321	+00:00:00.287	0x0023	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
16:53:47.603	+00:00:00.282	0x0028	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
16:53:47.895	+00:00:00.292	0x0029	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
16:53:48.183	+00:00:00.287	0x002a	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
16:53:48.378	+00:00:00.196	0x0000	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
16:53:48.670	+00:00:00.292	0x0001	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
16:53:48.963	+00:00:00.292	0x0002	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
16:53:48.975	+00:00:00.013	0x0003	0x0002	0x0400	0x0029	Zigbee NWK	NWK Data	
16:53:48.984	+00:00:00.008					IEEE 802.15.4	Acknowledgment	A2
16:53:49.261	+00:00:00.277	0x0003	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
				...				
16:53:52.912	+00:00:00.292	0x0001	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
16:53:52.925	+00:00:00.013	0x0002	0x0001	0x0400	0x0029	Zigbee NWK	NWK Data	
16:53:52.933	+00:00:00.008					IEEE 802.15.4	Acknowledgment	A3
16:53:53.211	+00:00:00.277	0x0002	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
16:53:53.483	+00:00:00.273	0x0400	0x0003	0x0400	0x0029	Zigbee NWK	NWK Data	
16:53:53.492	+00:00:00.009					IEEE 802.15.4	Acknowledgment	B1
16:53:53.566	+00:00:00.075	0x0003	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
				...				
16:53:56.874	+00:00:00.196	0x0000	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
16:53:56.887	+00:00:00.013	0x0001	0x0000	0x0400	0x0029	Zigbee NWK	NWK Data	
16:53:56.895	+00:00:00.008					IEEE 802.15.4	Acknowledgment	A4
16:53:56.929	+00:00:00.034	0x0000	0x0020	0x0400	0x0029	Zigbee NWK	NWK Data	
16:53:56.937	+00:00:00.008					IEEE 802.15.4	Acknowledgment	
16:53:57.184	+00:00:00.247	0x0001	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
16:53:57.476	+00:00:00.292	0x0002	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
16:53:57.489	+00:00:00.013	0x0003	0x0002	0x0400	0x0029	Zigbee NWK	NWK Data	
16:53:57.498	+00:00:00.009					IEEE 802.15.4	Acknowledgment	B2
16:53:57.775	+00:00:00.277	0x0003	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
16:53:58.067	+00:00:00.293	0x0004	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
16:53:58.350	+00:00:00.282	0x0009	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
16:53:58.642	+00:00:00.292	0x000a	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
16:53:58.929	+00:00:00.287	0x000b	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
16:53:59.275	+00:00:00.347	0x0020	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
16:53:59.278	+00:00:00.002	0x0020	0x0028	0x0400	0x0029	Zigbee NWK	NWK Data	
16:53:59.292	+00:00:00.014					IEEE 802.15.4	Acknowledgment	A5
16:53:59.579	+00:00:00.287	0x0021	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
16:53:59.871	+00:00:00.292	0x0022	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
16:54:00.158	+00:00:00.287	0x0023	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
16:54:00.504	+00:00:00.346	0x0028	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
16:54:00.506	+00:00:00.002	0x0028	0x0029	0x0400	0x0029	Zigbee NWK	NWK Data	
16:54:00.520	+00:00:00.014					IEEE 802.15.4	Acknowledgment	A6
16:54:00.829	+00:00:00.309	0x0029	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
16:54:01.095	+00:00:00.265	0x002a	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
16:54:01.183	+00:00:00.089	0x0000	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
16:54:01.476	+00:00:00.292	0x0001	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
16:54:01.488	+00:00:00.013	0x0002	0x0001	0x0400	0x0029	Zigbee NWK	NWK Data	
16:54:01.497	+00:00:00.008					IEEE 802.15.4	Acknowledgment	B3
16:54:01.774	+00:00:00.277	0x0002	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
16:54:01.975	+00:00:00.202	0x0400	0x0003	0x0400	0x0029	Zigbee NWK	NWK Data	
16:54:01.984	+00:00:00.008					IEEE 802.15.4	Acknowledgment	C1
16:54:02.129	+00:00:00.145	0x0003	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
16:54:02.365	+00:00:00.236	0x0004	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	
16:54:02.652	+00:00:00.288	0x0009	0xffff			IEEE 802.15.4	Beacon: BO: 8, SO: 4, PC: 1, AP: 1	

Fig. 12. Message Flow and Beacon Frames

The ZED (0x0400) has associated with ZR 0x0003 with the purpose of periodically transmit data frames through the cluster-tree in order to test the topology and the tree-routing mechanism. In Fig. 12, message flows are marked with capital characters (e.g. A, B, C) and the hop count with indexes (e.g. A1, A2, A3). The first transmission from the ZED (0x0400) to its parent (ZR 0x0003) is shown in A1. Note that this transmission is carried out during ZR 0x0003 Superframe. The routing of the data frame from ZR 0x0003 to its parent in the cluster-tree (ZR 0x0002) is marked as A2. The multi-hop continues with the routing of the frame from ZR 0x0002 to ZR 0x0001 (A3). In B1, a new message flow is initiated by the ZED (0x0400). Then, in A4, the message is relayed from ZR 0x0001 to ZC (0x0000) and to ZR 0x0020. This transmission sequence is carried out during the ZC Superframe. The multi-hop continues in A5 between ZR 0x0020 and ZR 0x0028. The last hop is carried out in A6 with ZR 0x0028 relaying it to its final destination, ZR 0x0028.

5. Concluding Remarks

The IEEE 802.15.4/ZigBee protocols 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 embedded computing applications.

In this context, we have triggered the ART-WiSe research line [15], 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 [10], 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 presented an overview of the most important aspects of the software architecture and implementation challenges, as well as a number of research works that build on its use.

Acknowledgment

This work was funded by FCT under the CISTER Research Unit (FCT UI 608) and PLURALITY (CONCREEQ/900/2001) projects, and by the ARTIST2 NoE (IST-2001-34820).

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", Sensor Networks and Configurations: Fundamentals, Techniques, Platforms, and Experiments, Springer-Verlag, Germany, pp. 19-49, 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", 14th International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS 2006), 2006.

[5] A. Koubâa, M. Alves, and E. Tovar, "i-GAME: An Implicit GTS Allocation Mechanism in IEEE 802.15.4", 18th Euromicro Conf. on Real-Time Systems (ECRTS'06), 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, 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", 1st International Conference on Wireless Internet (WICON'05), 2005.

[9] A. Koubaa, M. Alves, E. Tovar, "Modeling and Worst-Case Dimensioning of Cluster-Tree Wireless Sensor Networks", 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, http://www.xbow.com, 2007

[12] P. Jurčík, A. Koubâa, "The IEEE 802.15.4 OPNET Simulation Model: Reference Guide v2.0", www.open-zb.net, HURRAY-TR-070509, May 2007.

[13] P. Jurcik, A. Koubâa, M. Alves, E. Tovar, Z. Hanzalek, "A Simulation Model for the IEEE 802.15.4 protocol: Delay/Throughput Evaluation of the GTS Mechanism", to be published in the 15th IEEE International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS´07), 2007.

[14] OPNET Simulator v11, http://www.opnet.com.

[15] The ART-WiSe Research Framework, http://www.hurray.isep.ipp.pt/art-wise/

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

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

[18] Texas Instruments, "MSP430x21x1 Datasheet", 2004.

[19] ATmega128L 8-bit AVR Microcontroller Datasheet, Atmel ref: 2467MAVR-11/04, http://www.atmel.com

[20] Chipcon, "CC2420 transceiver datasheet", 2004.

[21] Chipcon Packet Sniffer for IEEE 802.15.4 v1.0, 2006.

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

[23] A. Cunha, M. Alves, A. Koubaa, "An IEEE 802.15.4 protocol implementation (in nesC/TinyOS): Reference Guide v1.2", HURRAY-TR-061106, http://www.open-zb.net 2007.

[24] A. Cunha, M. Alves, A. Koubâa, "Implementation of the ZigBee Network Layer with Cluster-tree Support", HURRAY-TR-070510, May 2007.

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

[26] TinyOS Network Protocol Working Group, http:/tinyos.stanford.edu:8000/Net2WG

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

[28] 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", HURRAY-TR-060202, Feb. 2006.

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

[30] A. Koubâa, A. Cunha, M. Alves, "A Time Division Beacon Scheduling Mechanism for IEEE 802.15.4/ZigBee Cluster-Tree Wireless Sensor Networks". 19th Euromicro Conf. on Real-Time Systems (ECRTS 2007), July 2007