ZIGBEE OVER TINYOS: IMPLEMENTATION AND EXPERIMENTAL CHALLENGES

André Cunha[1], Ricardo Severino[1], Nuno Pereira[1], Anis Koubâa[1,2], Mário Alves[1]

[1] *IPP-HURRAY Research Group, Polytechnic Institute of Porto (ISEP/IPP),Porto, Portugal*
[2] *Al-Imam Muhammad Ibn Saud University, Computer Science Dept., Riyadh, Saudi Arabia*

{arec, rars, nap, aska, mjf}@isep.ipp.pt

Abstract: The IEEE 802.15.4/Zigbee protocols are a promising technology for Wireless Sensor Networks (WSNs). This paper shares our experience on the implementation and use of these protocols and related technologies in WSNs. We present problems and challenges we have been facing in implementing an IEEE 802.15.4/ZigBee stack for TinyOS in a two-folded perspective: IEEE 802.15.4/ZigBee protocol standards limitations (ambiguities and open issues) and technological limitations (hardware and software). Concerning the former, we address challenges for building scalable and synchronized multi-cluster ZigBee networks, providing a trade-off between timeliness and energy-efficiency. On the latter issue, we highlight implementation problems in terms of hardware, timer handling and operating system limitations. We also report on our experience from experimental test-beds, namely on physical layer aspects such as coexistence problems between IEEE 802.15.4 and 802.11 radio channels.

Keywords: Wireless Sensor Networks, ZigBee, IEEE 802.15.4, TinyOS

1. INTRODUTION

IEEE 802.15.4/ZigBee [1,2] and TinyOS [3] are currently buzzwords, since these technologies have been playing and important role in leveraging a new generation of large-scale networked embedded systems. The IEEE 802.15.4/ZigBee protocol stack has several interesting technical features for serving as a federating communication technology for Wireless Sensor Networks (WSN). TinyOS is the most widespread operating system for embedded resource-constrained systems.

Within this context, we have been investigating the potentiality of the IEEE 802.15.4/ZigBee protocols for time-critical WSN applications. We have developed the open-ZB [4] open source toolset, encompassing a simulation model [5] and a protocol stack over TinyOS [11], for the MICAz/TelosB [6] motes. This toolset has been mainly used to test, validate and demonstrate our theoretical proposals through a number of experimental test-beds.

This paper describes the most important problems encountered in the implementation of the IEEE 802.15.4/ZigBee protocol stack over TinyOS and also in our experimental test-beds, identifying some of the most relevant challenges to be addressed.

The remainder of the paper is organized as follows: Section 2 overviews relevant aspects of the IEEE 802.15.4/ZigBee protocols, focusing on the Cluster-Tree topology. The major problems related to the open-ZB protocol stack implementation are presented in Section 3. Section 4 reports some experience from experimental test-beds. Finally, Section 5 provides some concluding remarks.

2. ON THE IEEE 802.15.4/ZIGBEE PROTOCOLS

2.1 Protocols Overview

The IEEE 802.15.4 protocol [1] specifies the Medium Access Control (MAC) sub-layer and the Physical Layer of Low-Rate Wireless Private Area Networks (LR-WPANs). The ZigBee protocol [2] relies on the IEEE 802.15.4 layers, building up the Network and Application Layers.

ZigBee defines three types of devices: (1) *ZigBee Coordinator* (ZC): one for each PAN, initiates and configures the network formation; (2) *ZigBee Router* (ZR): associated (as a child node) with the ZC or with a previously associated ZR, participates in multi-hop message routing; (3) *ZigBee End Device* (ZED): device with sensing/actuating capabilities but that does not allow other devices to associate with it and does not participate in routing.

The IEEE 802.15.4 Physical Layer is responsible for data transmission and reception using a certain radio channel. It offers three operational frequency bands: 2.4 GHz, 915 MHz and 868 MHz. There is one channel between 868 and 868.6 MHz, ten channels

[1] This work was partially funded by FCT under the CISTER Research Unit (FCT UI 608), PLURALITY and CMU-PT projects, and by the ARTIST2/ARTISTDesign NoEs.

between 902 and 928 MHz, and sixteen channels between 2.4 and 2.4835 GHz. Direct Sequence Spread Spectrum (DSSS) modulation is used.

The IEEE 802.15.4 MAC protocol supports two operational modes that may be selected by the ZC: (1) the *non beacon-enabled* mode, in which the MAC is simply ruled by *non-slotted CSMA/CA*; (2) the *beacon-enabled mode*, ruled by *slotted CSMA/CA*, in which beacons are periodically sent by the ZC to synchronize nodes that are associated with it, and to identify the PAN. In beacon-enabled mode, the ZC defines a Superframe structure (Fig. 1) which is constructed based on: (1) the *Beacon Interval* (*BI*), which defines the time between two consecutive beacon frames; (2) the *Superframe Duration* (*SD*), which defines the active portion in the *BI*, and is 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*):

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

aBaseSuperframeDuration = 15.36 ms (assuming 250 kbps in the 2.4 GHz band) denotes the minimum Superframe Duration, corresponding to *SO = 0*.

During the *SD*, nodes compete for medium access using slotted CSMA/CA, in the Contention Access Period (CAP). IEEE 802.15.4 also supports a Contention-Free Period (CFP) within the *SD*, by the allocation of Guaranteed Time Slots (GTS).

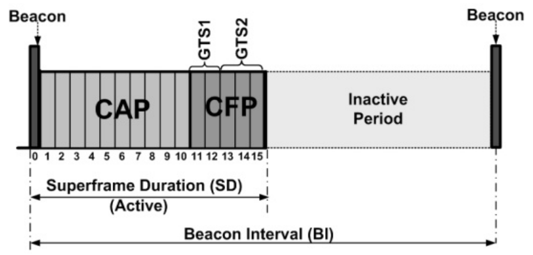
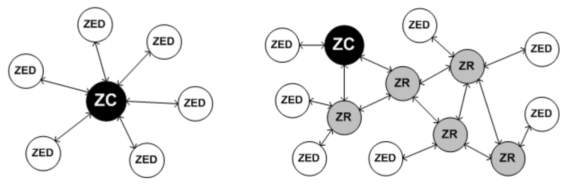


Fig. 1 Superframe structure [1]

It can be easily observed in Fig. 1 that low duty-cycles can be configured by setting small *SO* values as compared to *BO*, resulting in greater sleep (inactive) periods.

ZigBee supports three network topologies – star, mesh and cluster-tree, illustrated in Figs. 2 and 3.



a) star topology b) mesh topology

Fig 2. ZigBee star and mesh network topologies

In the *star topology* (Fig. 2a) communications must always be relayed through the ZC. In the *mesh topology* (Fig. 2b) each node can directly communicate with any other node within its radio range or through multi-hop. The *cluster-tree topology* (Fig. 3) is a special case of a mesh network where there is a single routing path between any pair of nodes and a distributed synchronization mechanism (operates in beacon-enabled mode).

2.2 Cluster-tree network model

The Cluster-Tree network concept is outlined in the ZigBee specification and exemplified in Fig. 3. A unique ZC identifies the entire network and each ZR assumes the role of cluster-head, allowing the association of other ZRs and ZEDs in a parent-child relationship. Inside each cluster, messages must be relayed by the cluster-head, i.e. as in star topologies. However, the IEEE 802.15.4/ZigBee specifications do no clearly describe how the cluster-tree model can be implemented, providing just a broad view on how the cluster-tree network should operate and on the tree routing algorithm.

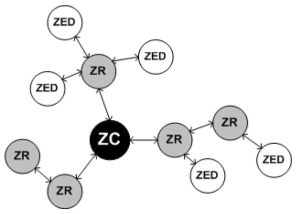


Fig. 3 Cluster-tree network topology

More specifically, the Cluster-Tree model includes more than one ZigBee Router that periodically generates beacons to synchronize nodes (or clusters of nodes) in their neighbourhood. If these periodic beacon frames are sent in an unorganized fashion, without any particular schedule, they will collide with each other or with other frames. These collisions result in the loss of synchronization between a parent ZigBee Router and their child devices, which prevents them to communicate.

Only some basic approaches dealing with this problem were proposed for discussion by the Task Group 15.4b [7], which is a group aiming to improve some inconsistencies of the original specification. Therefore, beacon scheduling mechanisms such as the one proposed in [10] are required to avoid beacon frame collisions in ZigBee cluster-tree networks.

2.3 Cluster-tree vs mesh topologies

Table 1 summarizes some of the differences between ZigBee mesh and cluster-tree topologies.

Table 1 – Mesh vs. Cluster-Tree

	Mesh	Cluster-Tree
Synchronization	No	Yes
Inactive Periods	ZEDs	All Nodes
Real-Time Communications	No	Yes (GTS)
Redundant Paths	Yes	No
Routing Protocol Overhead	Yes	No
Commercially Available	Yes	No

The synchronization (beacon-enabled mode) feature of the cluster-tree model may be seen both as an advantage and as a disadvantage, as reasoned next.

On one hand, synchronization enables dynamic duty-cycle management in a per cluster basis, allowing nodes (ZEDs and ZRs) to save their energy by entering the sleep mode. In contrast, in the mesh topology (as in [1]), only the ZEDs can have inactive periods; note that as the mesh network model supports no synchronization, ZRs are forced to be always active for complying with their routing duties. These energy saving periods enable the extension of the network lifetime, which is one of the most important requirements of WSNs. In addition, synchronization allows the dynamic reservation of guaranteed bandwidth in a per-cluster basis, through the allocation of GTS in the CFP. This enables the worst-case dimensioning of cluster-tree ZigBee networks, namely to compute worst-case message end-to-end delays and ZR buffer requirements [8].

On the other hand, managing the synchronization mechanism all over a cluster-tree network is a very challenging task. Even if we can cope with minor synchronization drifts between ZRs, this problem can grow for larger cluster-tree networks (higher depths). As previously mentioned, the de-synchronization of a cluster-tree network leads to collision problems due to beacon frames and Superframes overlap. For instance, the CAP of one cluster can overlap with the CFP of another cluster, which is not tolerable.

Regarding the routing protocols, tree routing (cluster-tree) is lighter than the AODV-like [9] protocol (mesh) in terms of memory and processing requirements. The routing overhead, as compared with AODV, is reduced. Note that the tree routing protocol considers just one path from any source to any destination, thus it does not consider redundant paths, in contrast to AODV. Therefore, the tree routing protocol is prone to the single point of failure problem, while that can be avoided in mesh networks if alternative routing paths are available - ZRs have more than one ZR within radio coverage.

Note that if there is a faulty ZigBee Router, network inaccessibility times may be inadmissible for applications with critical timing and reliability requirements. Thus, designing and engineering energy and time-efficient fault-tolerance mechanisms to avoid/minimize the single point of failure problem in ZigBee cluster-tree networks is crucial.

Besides the Beacon/Superframe scheduling and the single-point-of-failure problems, there are other challenges to effectively engineer ZigBee cluster-tree networks, namely: (1) the dynamic network resynchronization, for instance in case of a new ZR joining or leaving the network; (2) the dynamic rearrangement of the all the duty cycles in the case of a ZR failure; (3) a new router association or even rearranging the Superframe duration of some routers to adapt the bandwidth allocated to that branch of the tree; (4) the rearrangement of the addressing space allocated to each router; and (5) supporting mobility of ZEDs, ZRs or even whole clusters.

From our perspective, all these impairments have lead to the lack of commercial or even academic solutions based on the ZigBee cluster-tree model. Nevertheless, we consider this model as a promising and adequate solution for WSN applications with real-time and energy-efficiency requirements.

3. OPEN-ZB STACK: PROBLEMS/CHALLENGES

3.1 Introduction

The main problems we have encountered during the implementation of the IEEE 802.15.4/ZigBee protocol stack [4,11] are related to both hardware constraints and the lack of details in the standard specifications regarding important aspects of the beacon-enabled mode and of the cluster-tree model.

We have implemented the beacon-enabled mode of the IEEE 802.15.4 MAC sub-layer and the required functionalities in the ZigBee Network Layer to support cluster-tree topologies. TinyOS v1 was used over the MICAz and TelosB motes. More recently, though still under testing, we ported our stack to TinyOS v2.0, keeping the same software architecture [11], as a result from our collaboration with the TinyOS Network Protocol Working Group [12] to implement a ZigBee compliant stack for TinyOS 2.0.

3.2 Hardware platforms and debugging

The TelosB and MICAz architectures are slightly different, NAMELY due to their 16-bits MSP430 [13] and 8-bits Atmega128 [14] microcontrollers. This requires selecting the corresponding driver modules already provided in TinyOS and the adaptation of our first implementation version, which only supported the MICAz , to encompass the 16-bits memory block of the MSP430. Both platforms use the 2.4 GHz Chipcon CC2420 radio transceiver [15].

The MICAz requires the use of a programming interface (the MIB510), while the TelosB features an USB interface, enabling programming via the PC. Both motes provide a debug mechanism by sending data through the serial (COM/USB) port and reading it via a communication listener (e.g. ListenRaw, provided with the TinyOS distribution, or Windows HyperTerminal). This debugging mechanism raises a problem, since the transmission through the COM port blocks all the other mote operations, which usually causes synchronization problems.

In order to overcome these local debugging issues and to have a total control over the network behaviour and of all transmitted packets, we have been using two different network/protocol analysers [16,17]. The CC2420 Packet Sniffer for IEEE 802.15.4 v1.0 [16] provides a raw list of the packets transmitted. The software application works in conjunction with a CC2400EB evaluation board and a CC2420 radio transceiver. We have also been using the Daintree Sensor Network Analyser [17] that provides some additional functionality, such as geographically distributed sniffing, graphical topology of the network, statistics, message flows, PAN information and association details.

3.3 Memory constraints

The MICAz and TelosbB are very limited in terms of random access memory (RAM) – roughly 4 kB for the former and 10 kB for the latter. The RAM must be sufficient to fulfil the requirements of the TinyOS operating system, of the protocol stack and of the high level application. In this aspect, the MICAz motes are more constrained than the TelosB.

Take the example of two TinyOS 2.0 demo applications in order to demonstrate the variation in RAM memory usage – the *Blink* and *MultihopOscilloscopeApp* applications, compiled for both platforms. The first uses approximately 55 bytes and the second 3348 bytes of RAM. Besides the RAM memory allocated at compilation time, the devices need to have enough free memory for the operating system stack. In our TinyOS 2.0 implementation, the memory needed by an application that only uses the IEEE 802.15.4 beacon-enabled mode needs approximately 2678 bytes of RAM while an application using the ZigBee network layer with the cluster-tree topology needs approximately 3224 bytes.

Note that these demo applications are very simple and just used for testing purposes and that the different buffers used are very small. If the ZigBee Application Layer and more complex user-level applications are to be fit into these motes, memory limitations will surely come to the scene.

3.4 *CC2420 transceiver limitations*

The CC2420 transceiver (used by the MICAz and TelosB motes) also has some limitations in terms of turnaround time, i.e. the time that it takes to switch from receive mode to transmit mode and vice-versa.

According to the IEEE 802.15.4 standard, the transceiver's turnaround time must be 12 symbols (192 μs). This is the maximum time bound required to acknowledge messages. In fact, the CC2420 has the hardware configuration of auto-acknowledge messages but, besides generating several false acknowledgments (messages that are acknowledge but not received by the protocol stack), it needs to have the address decode functionalities activated.

Unfortunately, similarly to several IEEE 802.15.4 compliant transceivers, it is not possible to achieve the specified turnaround time. For instance, the Chipcon CC2420 can take up to 192 μs just 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.

In addition, the processing power available in the motes microcontroller revealed to be quite limited to comply with the most demanding IEEE 802.14.5 timing constrains, especially for small Beacon orders (*BO* < 3) and Superframe orders (*SO* < 3). This turns these Superframe configurations impossible to deploy, considering that the motes must also have availability for processing other tasks. It is reasonable to assume that the processing limitations can be easily overcome in the near future with the development of new and faster microcontrollers or by a hardware implementation of the protocol stack.

3.5 *Timing and synchronization requirements*

The timing requirements of the IEEE 802.15.4 protocol are very demanding. In the beacon-enabled mode, all devices (ZRs and ZEDs) must synchronize with their parents (ZR or ZC) through beacon frames. If a device loses synchronization it cannot operate in the PAN. Moreover, if a node is not properly synchronized, there is a possibility of collisions in the GTS slots (when the CAP overlaps the CFP). From our experience, synchronization losses can be caused by multiple factors, such as: (1) the processing time of the beacon frame for low *BO/SO* configurations; (2) the mote stack overflow, that results in a processing block or a hard reset; and (3) the reduced processing capability of the microcontroller in conducting some fundamental protocol tasks (e.g. creating beacon frames, managing GTS and indirect transmissions).

The implementation of the slotted CSMA/CA algorithm is also quite demanding in terms of timer accuracy, since the IEEE 802.15.4 protocol defines that each backoff period 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. These timers do not allow having the exact theoretical values of the *BI*, *SD*, time slot and backoff period durations as specified by the IEEE 802.15.4 standard. This discrepancy, however, does not impact the correct behaviour of the implemented protocol, provided that the same mote platforms are used in the experiments (at least as ZC and ZRs).

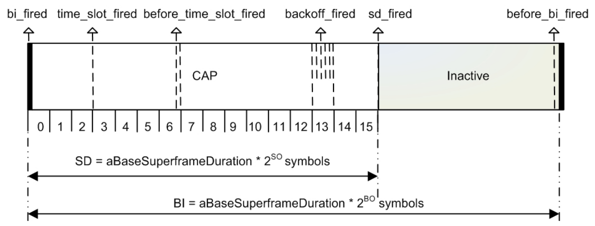


Fig. 4. Asynchronous events

The frequency of the asynchronous software events (Fig. 4), the hardware events and the low microprocessor processing ability may lead to an insufficient processing time left to execute the remaining protocol and higher-layer application tasks, as a great amount of interrupts have to be processed in short periods of time.

One hardware aspect that we had to take into account while implementing the timers needed to run the beacon-enabled mode of the IEEE 802.15.4 was the different hardware timer granularities of the MICAz and TelosB mote platforms. The timer granularities were approximated as much as possible to the theoretical values defined in the standard [11].

3.6 *TinyOS task scheduler*

The default TinyOS scheduler does not support tasks prioritization and is non pre-emptive. Typically, there are two different kinds of interrupt events in TinyOS: (1) the timers and (2) the radio. These events are captured by event handlers that normally post a task to the FIFO task queue, which significantly impacts the behaviour of the protocol stack.

In fact, sharing the microcontroller between all protocols tasks, such as processing and generation of the beacon frame or other messages, performing protocol maintenance or routing, is very demanding, specially for high duty cycles.

For example, processing and transmitting the beacon frame is a critical task for the protocol, and should take precedence over others. The problem is that the TinyOS scheduler does not support this. Nevertheless, there are several proposals [18,19] to introduce prioritization in the TinyOS task scheduler.

Operating system support for priority-driven multitasking is one relevant asset that could improve our implementation. In that direction, our future/ongoing work is to implement our protocol stack [11] in ERIKA [20], which is a real-time operating system for WSN platforms. We expect that ERIKA can enable reliable synchronization, even under high duty cycles. This will allow the comparison between two different embedded operating systems and assess all the features required for the correct behaviour of the IEEE 802.15.4/ZigBee protocols.

However, even with the aforementioned problems, the protocol stack behaves steadily for *beacon* and *Superframe orders* higher than 3.

4. PHYSICAL LAYER-RELATED PROBLEMS

4.1 *Introduction*

In the course of our research work, several experimental scenarios were built for testing and validating our theoretical proposals. During these implementation and experimental efforts, some difficulties were encountered namely in what concerns the behaviour of the hardware platforms – the MICAz and TelosB motes. In this section, we summarize some relevant problems we faced and how they have been tackled.

4.2 *Interference between radio channels*

In order to experimentally analyse the behaviour of the protocol, we devised several scenarios that enabled us to evaluate different network metrics, such as the *Network Throughput* and *Probability of Success* as a function of the *Network Load*. In general lines, these scenarios consisted of several nodes programmed to generate packets at the application layer with preset inter-arrival times, enabling us to push the necessary traffic load into the network. We used the previously referred IEEE 802.15.4 protocol analyzer [16] to log the received packets and developed an application to parse the message payload, which embedded relevant performance information retrieved from the nodes in order to compute the required metrics.

One requirement of the performance evaluation was to achieve high traffic loads in the network, even above 100% of the network capacity. We immediately observed that it was not only difficult to get a consistent behaviour of the Throughput metric but also to get high offered loads. Moreover, it was hard to ensure the stability of the network when the nodes were generating packets with very low inter-arrival times.

After performing several assessments, we reached the conclusion that this behaviour was mostly related to three factors: (1) the interference caused by close Wi-Fi equipment; (2) TinyOS-related constraints; and (3) others related to the node's scarce processing capability.

The interference between IEEE 802.11 and 802.15.4 radio channels, confirmed using a spectrum analyser, had unpredictable effects on the results. We observed that the interference of IEEE 802.11 networks often generated collisions with data/beacon frames. This effect, lead to data corruption and network de-synchronization. Moreover, it also had implications on the amount of traffic sent to the network because in the IEEE 802.15.4 slotted CSMA/CA protocol, the medium was often sensed as busy (during the Clear Channel Assessment (CCA)), causing deference and failed transmissions. This obviously affected the behaviour of the network since it did not allow reaching high traffic loads. We overcame the interference problem by using the only IEEE 802.15.4 channel (Channel 26 in the 2480 MHz frequency band) that is completely outside the IEEE 802.11 frequency spectrum (Fig. 5).

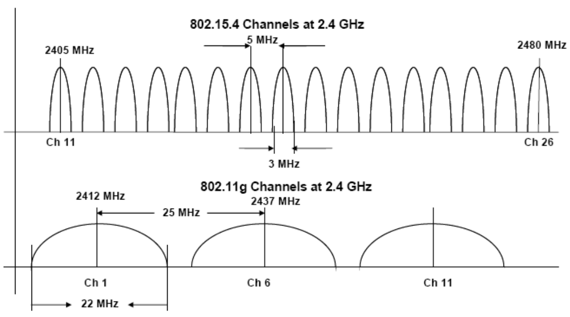


Fig. 5. IEEE 802.15.4/02.11 spectra [21]

Note that this interference must be really taken into consideration for the reliable deployment of ZigBee networks operating in the 2.4 GHz frequency band. Nevertheless, besides the interference problem, we have also identified other sources of inconsistencies.

As already discussed in Section 3, TinyOS imposes several limitations that influence the behaviour of the protocol stack, namely on the synchronization. We have observed that when packet transmission followed a very small inter-arrival time (in the order of 50 packets per second) the de-synchronization was a concern, mainly due to the high amount of tasks posted to generate the required offered load. To mitigate this problem, we programmed the nodes to generate packets only during the active portion of the Superframe, trying to guarantee that the beacon frame would be parsed immediately upon the reception. Nevertheless, when using a full duty cycle the problem remained. Thus, we used a new timer that fires a few milliseconds before the end of the Superframe, stopping all the packet generation and leaving the nodes ready to process the beacon frame.

4.3 *RSSI-based localization inaccuracy*

Another test-bed scenario consisted of a Search&Rescue application to demonstrate the ART-WiSe architecture [22]. A rescuer robot is supposed to track and reach, in the minimum amount of time, a steady or moving target (person or robot), using a wireless sensor network for tracking and localization. A first approach to this application was reported in [23]. In this context we wanted to develop a simple but effective localization mechanism, relying as much as possible on COTS technologies and taking advantage of the Received Signal Strength Indicator (RSSI), available directly in the CC2420 transceiver, for estimating distances between source and destination nodes.

We immediately observed that these measurements were highly sensitive to ambient conditions. The proximity to metal and walls highly increased the number of reflections leading to non-consistent RSSI readings. Moreover, the RSSI value was not linear with the distance and it varied with different mote antenna orientations. This means that it was probable to find different RSSI readings for the same distance.

To overcome this problem, several experiments were carried out for different distances, transmission powers and antenna orientations, in an attempt to get a consistent set of values for different distances [24,25]. After these experiments, it became possible to establish a correspondence between distance range levels and the window of RSSI values encountered for that same range. This enabled us to engineer a simple RSSI-based localization mechanism that is characterized by an inaccuracy of roughly 60 cm, which we consider acceptable for the envisaged applications.

4. CONCLUDING REMARKS

In this paper, we have reported several problems and challenges emerging from our experimental work on the IEEE 802.15.4/ZigBee protocol stack [4,11]. We have presented some issues that are either ambiguous or actually left open in the 802.15.4/ZigBee protocol specifications. We have also reported on our experience concerning the implementation and use of the open-ZB protocol stack.

Some open and ambiguous issues in the standard specifications require some protocol add-ons and appropriate system planning and network dimensioning. This is particularly true for supporting scalable, reliable, energy-efficient and time-sensitive applications with ZigBee cluster-tree networks.

The hardware platforms under use – MICAz and TelosB – seem to be too limited for the demanding requirements of ZigBee cluster-tree networks, where synchronization depends on the distributed transmission of beacon frames. This also results from the limitations of TinyOS to tackle this demanding protocol behaviour. Therefore, we intend to migrate the open-ZB protocol stack to real-time operating systems (e.g. ERIKA) and to more powerful mote platforms, as well as extending it for supporting mesh topologies.

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] Zigbee-Alliance, "ZigBee specification," http://www.zigbee.org/, Dec 2006.

[3] TinyOS, www.tinyos.net, 2008.

[4] Open-ZB, "Open-source toolset for IEEE 802.15.4 and ZigBee" web site: www.open-zb.net.

[5] P. Jurcik, A. Koubaa, M. Alves, E. Tovar, Z. Hanzalek, "A Simulation Model for the IEEE 802.15.4 Protocol: Delay/Throughput Evaluation of the GTS Mechanism", In Proc. IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS´07), Istanbul, Turkey, October 2007.

[6] Crossbow web site: http://www.xbow.com, 2007.

[7] IEEE 802.15.4b Task Group, web site: http://grouper.ieee.org/groups/802/15/pub/TG4b.html

[8] A. Koubaa, M. Alves, E. Tovar, "Modelling and Worst-Case Dimensioning of Cluster-Tree Wireless Sensor Networks", 27th IEEE Real-time Systems Symposium (RTSS´06), Rio de Janeiro, Brazil, December 2006.

[9] IETF, RFC 3561 Ad hoc On-Demand Distance Vector (AODV) Routing, 2003.

[10] 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), Pisa, Italy, July 2007.

[11] A. Cunha, A. Koubâa, R. Severino, M. Alves, "Open-ZB: an open-source implementation of the IEEE 802.15.4/ZigBee protocol stack on TinyOS", in Proc. of the 4th IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS´07), Pisa, Italy, October 2007.

[12] TinyOS Network Protocol Working Group web site: http://tinyos.stanford.edu:8000/Net2WG

[13] Texas Instruments, "MSP430x1xx Family User's Guide", 2004.

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

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

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

[17] Daintree Networks, http://www.daintree.net, 2006.

[18] C. Duffy, U. Roedig, J. Herbert, C. Sreenan, "Adding Preemption to TinyOS", Fourth Workshop on Embedded Networked Sensors (EmNets 2007), University College Cork, Ireland, ACM Digital Library, June 2007.

[19] V. Subramonian, H. Huang, S. Datar, "Priority Scheduling in TinyOS : A Case Study", Washington University Technical Report WUCSE-2003-74.

[20] E.R.I.K.A., http://erika.sssup.it, 2008.

[21] Crossbow Technology Inc., "Avoiding RF Interference Between WiFi and Zigbee", available at http://www.xbow.com.

[22] The ART-WiSe research framework web site: http://www.hurray.isep.ipp.pt/art-wise

[23] M. Alves, A. Koubaa, A. Cunha, R. Severino, E. Lomba, "On the Development of a Test-Bed Application for the ART-WiSe Architecture", Work-in-Progress Session of the 18th Euromicro Conference on Real-Time Systems (ECRTS'06), Dresden, Germany, July 2006.

[24] R. Severino, M. Alves, "On a Test-bed Application for the ART-WiSe Framework", HURRAY-TR-061103, November 2006.

[25] R. Severino, M. Alves, "Engineering a Search and Rescue Application with a Wireless Sensor Network - based Localization Mechanism", Poster Session of the IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM'07), Helsinki, Finland, June 2007.