Power Management Extensions for Tagus-SensorNet

Jorge M. Soares†, Bruno Gonçalves†, Rui M.Rocha*

†Instituto Superior Técnico
Technical University of Lisbon
Lisbon, Portugal
{jorgesoares,brunojfgoncalves}@ist.utl.pt

*Instituto de Telecomunicações, Instituto Superior Técnico
Technical University of Lisbon
Lisbon, Portugal
rui.rocha@lx.it.pt

***Abstract* — A flexible Wireless Sensor Network platform for easier implementation of diverse applications has been developed and deployed at one of the Instituto Superior Técnico - Technical University of Lisbon (IST-TUL) campus. Since its initial deployment in 2007, this test-bed has grown steadily, supporting new nodes, applications and experiments. However, some initial problems, which were solved on an ad hoc basis, are becoming more serious as the network spans throughout the campus. Major issues, like global power management, have to be tackled not only with traditional protocol level approaches but also from a system's viewpoint, providing solutions that are capable of guaranteeing a consistent operational test-bed. We discuss the main issues related with the development of power management solutions, at different levels, presenting our architecture, design choices and implementation. We also address the lessons learnt from its integration in the platform. Results of the experimental evaluation of our solution have shown considerable energy savings (extending the network lifetime up to 9 times) even in the presence of demanding applications.**

Keywords - Power Management; WSN; Test-bed; Radio Cycling; Time Synchronization

I. INTRODUCTION

Over the last few years, Wireless Sensor Networks (WSN) have been gradually moving from a mostly theoretical, academic approach to real world roles in industrial and commercial applications. The availability of better and increasingly cheaper sensor platforms have decisively contributed to foster WSN deployment worldwide.

Recognizing this trend, and taking advantage of the experience gained through the participation in several internal and external projects in this area, IST-TUL decided to develop and deploy a WSN test-bed at one of its campus [1].

Since then, Tagus-SensorNet, as the test-bed was later named, has grown to support additional applications and experiments. New nodes, also based on Crossbow's MICAz motes [2], were added (for a current total of 29) and the whole network has been migrated to TinyOS 2.1, a new major version of the operating system[1]. Several students worked on the project, developing new hardware and software components.

However, as with any other deployment of its kind, there are several operational problems affecting Tagus-SensorNet, of which the rapid decay of energy levels, and consequent short network lifetime, was one of the most serious. Even taking advantage of Lower Power Listening (LPL) modes [3], low routing traffic activity, and periodic monitoring rounds for collecting environmental data, compressed through in-network processing techniques, without a power management solution batteries had to be replaced approximately once a week. Considering the number of nodes and the difficulty in accessing some of their locations, such frequent replacement was not very practical. This caused the network to be unavailable most of the time, requiring activation every time a specific experiment was to be performed.

Clearly this situation was far from optimal, as one could not tap the otherwise continuous stream of data made available by the deployed applications. Aiming to enable always-on operation of Tagus-SensorNet, the decision of investing in the development of an overlay solution for this energy problem was taken. The solution – Tagus-SensorNetPM or TagusPM for short – involving both node-level and network-wide power management schemes, comprises two basic components:

- hardware – consisting of higher capacity batteries, energy harvesting schemes and quicker and easier recharge methods;
- software – meant to reduce energy consumption and allow easy monitoring of nodes' energy status.

The hardware component uses environmental energy harvesting, either based on solar or vibration sources, to charge a lithium rechargeable battery through a dedicated interface circuit that also drives a supercapacitor, the secondary energy buffer of the system. The supercapacitor is also used to rapidly replenish the energy supply of sensor nodes in situations where there is no easy way to install the harvesting technology (e.g. there are no light or vibration sources available) and there is easy access to the node's hardware. In such a case, the use of a small size lead battery can be an interesting solution to quickly recharge the motes. This intelligent power supply is managed by an ultra low-power microcontroller that interfaces with the sensor node main board through an I^2C interface, providing information about energy resource levels that can be used for monitoring purposes.

While the hardware is still under development, the required software has already been implemented and deployed, and will be our focus for the rest of this paper.

Our goal is not to present a revolutionary solution for energy saving in WSNs, but rather to recount our experiences

This work was fully sponsored by Instituto de Telecomunicações and Instituto Superior Técnico, Technical University of Lisbon

[1] http://www.tinyos.net/scoop/section/Releases

with development and real-life deployment. Of the numerous previously proposed approaches to solving this problem, most take the form of power-aware single-layer protocols. Popular examples come from the MAC (e.g. S-MAC, Sift, WiseMAC) [4] and routing (e.g. PEGASIS, TEEN, MECN) [5] camps. There are also some, albeit fewer, more encompassing approaches, such as the one presented by UPMA [6]. A real application, featuring both software techniques and energy harvesting hardware, can be seen in the ZebraNet Project [7].

The remainder of this article is organized as follows: in Section II we discuss the requirements and initial choices related to the system design; in Section III we provide an overview of the system architecture and the reasons that led to it; in Section IV we detail the implementation, the development process and the problems we faced; in Section V we show some experimental results; finally, in Section VI we draw some conclusions and finish by suggesting some future work.

II. System Requirements

When first approaching the problem, we defined a set of basic requirements for the TagusPM solution:

- It had to provide a significant extension of network lifetime.
- Existing applications should not be seriously affected, i.e. it should not force applications to be rewritten.
- It should be easy to use by applications that wish to take advantage of the full functionality to integrate with it.
- It should provide easy remote access to each node's energy level.

Considering these requirements, and the general constraints posed by the used platform, we opted for a simple solution: controlling only the radio power state. While this choice may be seen as limiting, the radio is the largest energy consumer on a MICAz mote [8], and, for a typical usage pattern, most of this energy is wasted on unnecessary idle listening.

There are two typical strategies for radio power management:

- Asynchronous switching, in which each node turns its radio on at a self-chosen time, independently of the nearby nodes, using fixed or variable intervals.
- Synchronous cycling, in which all nodes switch their radios' power state at approximately the same time, with some (generally) fixed round period.

The asynchronous sleep mode can be used in the TinyOS 2.x platform provided LPL is turned on. In this mode, every node wakes periodically to check if there is another node with a message addressed to it. However, as the nodes are not synchronized, they wake up at different times forcing the sender to transmit long preambles to ensure that all receivers are ready to receive a message it wants to send [9]. Moreover, as soon as a receiver senses the preamble, it has to stay awake waiting for the upcoming message. Besides the overhearing inefficiency involved, the lack of synchronization implies a larger radio power consumption of both receivers and transmitters.

Synchronous approaches tend to be more efficient, as a node meaning to transmit a message does not need to keep its radio on while waiting for the destination to wake up. In particular, a synchronous MAC layer, of which there are several implementations for TinyOS, could provide a more than satisfactory solution for this energy efficiency problem. However, such a single-layer mechanism, having its own duty-cycle rounds to cope with, hardly matches the application timings, naturally leading to some increase on buffering needs and latency.

On the other hand, the existing Tagus-SensorNet software framework already provided a time synchronization service, used by several applications to perform round-based in-network data aggregation and processing. Thus, a cross-layer approach, integrating the application rounds with radio control procedures seemed to be the best option.

Our basic working model so far consisted of turning the radio on and off synchronously across the entire network. Then, in order to minimize the impact of the power management system on applications developed under the assumption of an always-on radio, the natural solution was to implement a message queuing system. Finally, monitoring was virtually separated from the rest of the functionality and could be freely implemented as a separate component or application.

III. System Architecture

With the initial decisions already made, we set out to draft a pluggable component model capable of achieving our energy-saving goal while still fulfilling the other requirements, namely easy integration and low impact on existing applications.

The first approach was centred on the insertion of a new layer on the CC2420 Radio Stack [10]. This layer included not only the radio control logic, but also a common global queue for all applications. We soon found out that the interface contract for the message sending interfaces imposes a limit of a single pending message per sender [11], forcing us to extract the queuing functionality to an external component. The resulting architecture is shown in Fig. 1.

Under this architecture, the system was composed of the following components:

- TagusPmManager, which had the function of coordinating the other modules.
- TagusPmLayer, which was responsible for controlling the radio power state.
- TagusPmMonitor, which obtained and sent the energy readings.
- TagusPmQueue, which stood between the applications and the sending interface and implemented a common queue for the messages handled.

We quickly began to notice the disadvantages of this model: it was tied to the CC2420 stack, involved modifying TinyOS files and was extremely difficult to debug. Although inadequate, it did pave the way for a second, cleaner approach.

Using the lessons learnt in our first try, we designed a new architecture that did not require the use of a layer, or any other modification to the TinyOS core system. The result is presented in Fig. 2, with the power management functionality being split into 3 separate modules:

- TagusPM Controller, responsible for controlling the entire system, keeping the system state and interacting with external services and applications.
- TagusPM Monitor, an entirely separate, application-level component that includes the battery level monitoring functionality.
- TagusPM Queuing, a set of multi-instanced components that may be used by applications to buffer data during radio-off times.

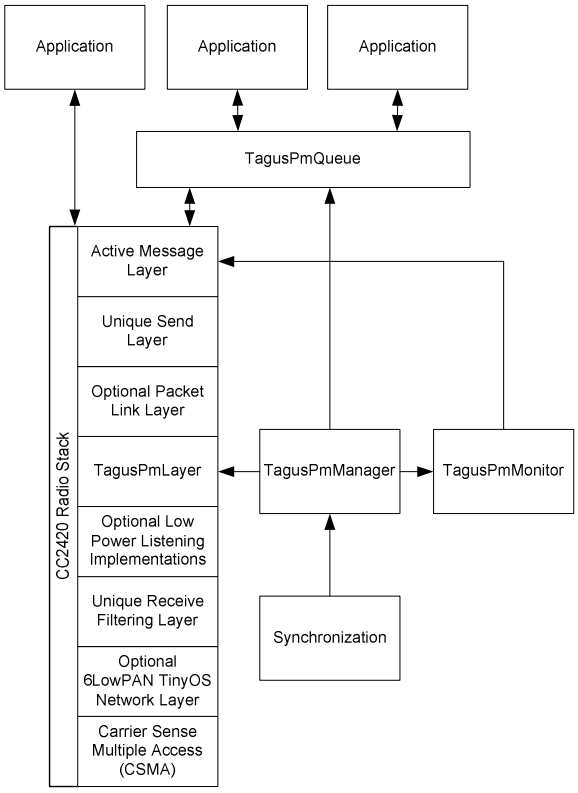

Figure 1. First approach to the TagusPM system architecture

The queuing and monitoring components are completely optional, with the core functionality being contained in the controller. This allows for some flexibility regarding resource usage, as not all applications and scenarios require the extra features, and memory is usually a scarce resource.

The queues were implemented as wrapper components for the sending interface, and are instantiated and used by each application separately. The adoption of individual queues brings a significant advantage over the previously proposed global solution, as it lets applications dimension their queues according to their real needs. Applications that choose not to use the queuing components, in order to spare memory or because they require *deterministic* message dispatching, are still able to subscribe to power management events, being notified every time the radio is turned on. This allows application to time their messages (or, in some cases, their full behaviour) as to not lose any data, even in the absence of queuing.

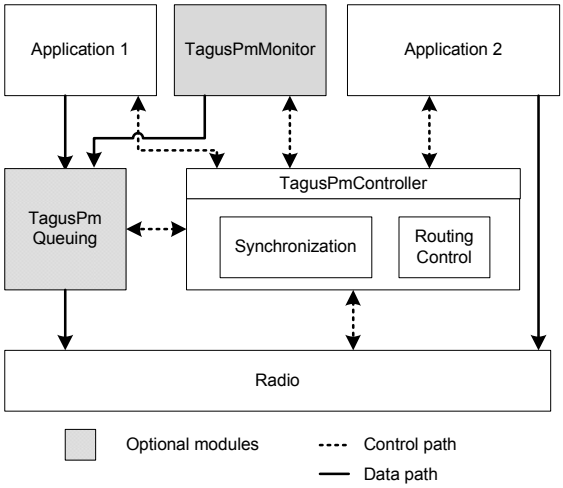

Figure 2. Final TagusPM system architecture

As for the Synchronization block, it is actually a simplified representation of a set composed by the FTSP synchronization components and our own module that generate the synchronous rounds from the time reference, and which will be described further ahead. The block referred to as Routing Control is the control interface for the routing protocol, which allows us to manually send route discovery messages at will.

IV. Design and Implementation

Once the system architecture was defined, a prototype was designed and implemented. While parts of the implementation were straightforward conversions of the architecture, some aspects required careful thought and consideration. Over the next subsections, we will present some of the most relevant, including the system core, the synchronous rounds component, the message queuing implementation and the monitoring functionality.

A. TagusPmController

The TagusPmController component assumes a central role in the system, encompassing all the logic and controlling the remaining modules and services, including the radio, the synchronization mechanisms and the routing messages. Its implementation is small, spanning just over 300 lines of nesC code, but its development required some care, as it deals with a relatively complex state machine, shown in Fig. 3.

The state diagram is divided into two parts:

- On the right, with transitions in full lines, are the states corresponding to the stable regime, i.e. in normal operation, the system is synchronized and cycles between these states with a defined period.
- On the left, with transitions in dashed lines, are the special states, used when the system is activated or deactivated, as well as when it is still synchronizing.

For clarity, the events leading to the state transitions were omitted from the diagram. The three labeled events correspond to external interventions: either by other modules, such as applications or framework components, or, in the case of *WatchdogTimer.fired()*, by the synchronization watchdog, whose function is to reset the system if it loses a correct time reference. In the absence of this watchdog, it would be possible for a node to fall out of synchronization with the rest of the network, never to be able to communicate again. The remaining transitions are triggered by timers, callbacks from split control interfaces or signals from the synchronous rounds generator. A more detailed description of each state can be found in Table I.

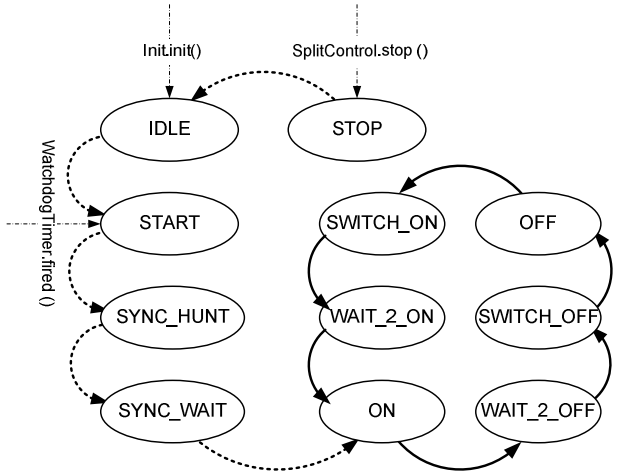

Figure 3. STATE DIAGRAM FOR THE TAGUSPMCONTROLLER MODULE

In a normal situation, the following sequence of events takes place:

- The node is powered on and the system is loaded in the IDLE state.
- The application or the OS call *Init.init()*, placing the system into state START. The system then initializes the routing algorithm, starts the watchdog timer and asks for the radio to be turned on. When the radio comes up (or if it already was), the system moves to state SYNC_HUNT and configures the synchronization module to send periodic beacons.
- The acquisition of a time reference signals the system, which moves to state SYNC_WAIT, and starts a timer, in order to allow global time to stabilize before beginning radio cycling.
- The firing of this timer means that the system is ready to enter normal operation. The state is set to ON, the synchronization module is set to manual control and a timer is started. After T_{round}, the timer fires and the system enters the WAIT_2_OFF state, starting another timer. After a brief delay, the timer fires, causing the system to go into state SWITCH_OFF and request the radio to be turned off. When the radio call-back is received, the system moves to state OFF.
- A generally similar process occurs when a new round is signalled. First the system is moved into state SWITCH_ON, while waiting for the radio to come up, and then to WAIT_2_ON, which has a small random delay added in order to reduce collisions. Finally, the system sets the ON state, starts a timer with T_{round}, triggers the sending of the necessary control messages, and, after a brief delay, alerts queues and applications that the radio is available. From then on, it continues the cycle already described.

TABLE I. EXPLANATION OF TAGUSPMCONTROLLER STATES

	State	Description
Normal behaviour	ON	Radio on, messages flowing freely.
	OFF	Radio off, no radio traffic.
	WAIT_2_ON	Radio just turned on, waiting for sufficient time to guarantee all nodes activate their radios.
	WAIT_2_OFF	Radio on but about to be turned off, finishing transmission of pending messages.
	SWITCH_ON	Radio off, but request to turn on already sent to the stack.
	SWITCH_OFF	Radio on, but request to turn off already sent to the stack.
Other states	IDLE	System disabled, either not yet started or already stopped.
	START	System currently being enabled, as a consequence of an external function call.
	STOP	System currently being disabled, as a consequence of an external function call.
	SYNC_HUNT	System enabled, radio on, waiting to acquire a time reference.
	SYNC_WAIT	System enabled, radio on, acquired a time reference, waiting to validate it.

B. Round Synchronization

The synchronization reference for the TagusPM system is provided by the Flooding Time Synchronization Protocol (FTSP) [12] implementation supplied in TinyOS and used in our software framework. Oversimplifying, this protocol establishes a synchronization tree, propagating the time reference from the root to the leaves. This time reference is a single integer counter, separate from the local clock, and valued in milliseconds.

The value, as it is, is useful as a way to timestamp events. For our purpose, however, we need to convert this reference into a synchronous alarm at all nodes. We created a component, RoundSyncC, which uses a set of one-shot timers to achieve this transformation.

FTSP had been in use in Tagus-SensorNet for two years, showing good performance – this was still while running TinyOS 1.x. The shift towards TinyOS 2.x and the new implementation was actually concurrent with the development of TagusPM, and it was assumed the system would perform just as well – an assumption that turned out to be wrong. While there were no major problems (just some infrequent synchronization losses) during our lab evaluation, on deployment to the full test-bed we noticed the nodes never managed to get a stable reference, dropping out so often that the system seldom got past the SYNC_WAIT state. Worse, sometimes FTSP acquired an invalid reference but did not become aware of it, leading to long periods in which nodes had

their rounds triggered at different times, and could not listen to other nodes' synchronization broadcasts. The same problems were later detected in the network even without TagusPM activated, even though its critical role in our system caused it to be much more obvious. Further investigation allowed the linking of this behaviour to a bug report on the TinyOS website, dealing with the implementation of low level packet times stamps, as of yet unfixed[2].

C. Queuing

What shows up in Fig. 2 as TagusPmQueuing is, in reality, a set of two components, called QueuedSender and QueuedAMSender. As previously stated, these serve as wrapper components for the native Sender and AMSender, exposing the same interface but buffering the messages they receive.

These components do not, in fact, entirely comply with their interface specifications, as they do not enforce the existence of a single pending message. While this is not elegant, it is fully intended and documented. Our goal while designing the queuing modules was to keep the necessary adaptations to existing applications to a bare minimum. This way, and considering applications should not be counting on a reliable channel to start with, we are able to reduce required changes to only two lines, thereby decreasing the adaptation cost.

D. Monitoring

The monitoring functionality was implemented as a separate module that, while conceptually a service, is in practice an application. It periodically collects battery level information, using the built-in *sensor*, creates a message and sends it to the operating center, through the multi-hop Collection protocol [13].

On reception, these messages are parsed and the battery values are updated on a table featuring all of the nodes, allowing the operator to check the energy status of the whole network at a glance. In the future, we expect the system to collect additional information provided by the hardware currently being developed.

V. EVALUATION

In order to ascertain the real impact of our system, some evaluation tests were conducted. First, we had to determine the real energy usage under normal operating conditions, with the radio on and off. The values presented in Table II were obtained from a MICAz node, running a simple application that constantly collects and sends light sensor readings (somewhat of a worst-case scenario for power management), and are an average of 10 minutes of continuous measurement.

Looking at Table II, we can see that there is a relevant savings potential, the energy consumption with the radio on being approximately 9 times higher. It should be noted that these values are dependent on the behaviour of each application, especially the energy consumption with the radio powered off. Its value can exhibit extreme variations depending on the fraction of time the microcontroller spends in its low power states, as well as on how the application uses the remaining hardware: a single LED, for instance, can require up to 2.5mA [8].

TABLE II. AVERAGE POWER DEPENDING ON POWER STATE

Radio state	Average Power (mW)
On	76.42
Off	8.33

We then proceeded to quantify the expected average power (1) and power saving factor (2), which is also an estimate of the network lifetime extension factor. In both formulae, P_{off} and P_{on} refer to the average power with the radio respectively off and on, T_{round} is the round time and T_{on} measures the time the radio is on in each round (it is therefore included in the round time).

$$P_{avg} = \frac{P_{off}(T_{round} - T_{on}) + P_{on}(T_{on})}{T_{round}} \quad (1)$$

$$r = \frac{P_{on}\, T_{round}}{P_{off}(T_{round} - T_{on}) + P_{on}\, T_{on}} \quad (2)$$

We then plotted (1), considering the power values from Table II, as well as a T_{on} value of 2 seconds, which, we believe, provides as good balance between the needs of the test application, the wish for a short on time and the reduction of wasted energy on the on/off guard times. Fig. 4 shows, as expected, that average power decreases exponentially as a function of the round time.

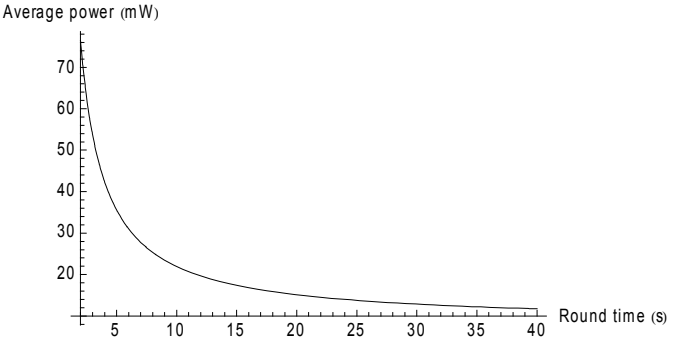

Figure 4. Impact of round time on energy consumption, for a fixed radio on time of 2 seconds

The choice of round time depends on the exact applications, as it must consider both the acceptable delay and the desired energy savings. For our application, we decided to use a round time of 10 seconds, and performed real-life tests to validate both the system and our expected power reductions. The results of one such test, consisting of a 60 minutes long measurement of a node's average power, can be seen in Table III.

Even when using these admittedly conservative settings, we achieve a network lifetime of approximately 3.5x the one we had before. Further tests showed that, with these times and adequate queue sizes, there was no change in the packet loss rate when using TagusPM. It is however important to note that these measurements refer to an always-on application with a high sampling rate, in which there are high data rates, short

[2] http://docs.tinyos.net/index.php/PacketTimeStamp_CC2420_bug

CPU sleep times and low potential for energy saving. In a less demanding application, not only could the duty cycle be reduced (equivalent to the round time being increased), but, thanks to the TinyOS microcontroller power management subsystem, the average power for the radio off situation would also be lower, leading to a steep decrease on energy consumption.

TABLE III. PREDICTED AND MEASURED VALUES FOR OUR TEST SETTING

Parameter	Value
P_{avg} (predicted)	21.95 mW
P_{avg} (measured)	22.17 mW
r (measured)	3.447
Duty cycle	20%

VI. CONCLUSIONS AND FUTURE WORK

The developed system, TagusPM, was able to considerably reduce energy consumption and extend the network lifetime, as shown on the course of our evaluation. The ability to monitor the battery status can also aid network operators in their task, allowing them to know which nodes need replacement, instead of having to check each and every one or wait for them to run out of energy. It presents a significant contribution to the future of Tagus-SensorNet, increasing the network availability and reducing the maintenance effort.

We also conclude that in order to develop an efficient power management solution it is critically important to follow a cross-layer approach, in our case involving the applications, the power management system, the synchronization system and, to a lesser extent, the routing protocol, with which we only interact to guarantee that route discovery messages are only sent when the radio is on. It is however possible to improve this solution by integrating new energy-aware MAC and routing protocols, causing the message flow to take into account the nodes' energy state.

Unfortunately, our efforts fell short of a real world deployment, frustrated by the lack of a mature time synchronization implementation in TinyOS 2.1. While we expect the current implementation to be fixed, and there are others in the works, we expect to start the development of a new time synchronization protocol at IST in the near future.

Although our current solution fulfils our initial requirements, energy saving in sensor networks is a very broad and open research area, and, using TagusPM as the basis, much can still be done. As part of our future work we intend to tackle the following issues:

- The definition of duty cycling parameters in run time, either automatically or by an operator's request. As duty cycling coherence is critical for the nodes to be able to communicate, this should involve the use of some transactional semantics (e.g. Two-phase commit), in order to guarantee simultaneous switchover of all nodes.
- The usage of different round times in each node, namely multiples of a base round time T_r. Methods should be found to dynamically chose these parameters, according not only to internal needs but also to the network load.
- The expansion of the monitoring interface to accommodate calculation of derived measurements, prediction of battery replacement times and notification of the operators.
- The interaction between our system and the Low Power Listening layer distributed with the TinyOS CC2420 stack, as the concurrent usage of both could lead to further reductions on the energy consumption.

ACKNOWLEDGMENTS

We wish to thank our colleagues at the Group of Embedded Networked Systems and Heterogeneous Networks (GEMS) for their input, and especially Luís Pedrosa for the precious help and advice given over the entire course of this project.

REFERENCES

[1] L. D. Pedrosa, P. Melo, R. M. Rocha, and R. Neves, "A flexible approach to wsn deployment," in Proceedings of 17th International Conference on Computer Communications and Networks. ICCCN '08, St. Thomas, U.S. Virgin Islands, 2008.

[2] Crossbow Technology, Inc., "MICAz Datasheet," Document Part Number 6020-0060-04 Rev. A.

[3] D. Moss, J. Hui and K. Klues, "Low Power Listening," TinyOS Core Working Group, TEP 105, 2007.

[4] I. Demirkol, C. Ersoy, and F. Alagoz, "Mac protocols for wireless sensor networks: a survey," in Communications Magazine, IEEE, vol. 44, no. 4, 2006, pp. 115-121.

[5] K. Akkaya and M. Younis, "A survey on routing protocols for wireless sensor networks," in Ad Hoc Networks, vol. 3, no. 3, 2005, pp. 325-349.

[6] G. Xing, M. Sha, G. Hackmann, K. Klues, O. Chipara and C. Lu, "Towards unified radio power management for wireless sensor networks," in Wireless Communications and Mobile Computing, vol. 9, no. 3, 2009, pp. 313-323.

[7] P. Zhang, C. M. Sadler, S. A. Lyon, and M. Martonosi, "Hardware design experiences in zebranet," in Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems, Baltimore, USA, 2004, pp. 227-238.

[8] M. Krämer and A. Geraldy, "Energy measurements for micaz node," University of Kaiserslautern, Kaiserslautern, Germany, Technical Report KrGe06, 2006.

[9] J. Hill, J. Polastre, and D. Culler, "Versatile low power media access for wireless sensor networks," in Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems, Baltimore, USA, 2004, pp. 95-107.

[10] D. Moss, J. Hui, P. Levis and J. Choi, "CC2420 Radio Stack," TinyOS Core Working Group, TEP 126, 2007.

[11] P. Levis, "Packet Protocols," TinyOS Core Working Group, TEP 116, 2007.

[12] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi, "The flooding time synchronization protocol," in Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems, Baltimore, USA, 2004, pp. 39-49.

[13] R. Fonseca, O. Gnawali, K. Jamieson and P. Levis, "Collection," TinyOS Net2 Working Group, TEP 119, 2006.