Self-Synchronized Duty-Cycling For Sensor Networks With Energy Harvesting Capabilities: Implementation in Wiselib

H. Hernández [1], T. Baumgartner [2], C. Blum [1], M. J. Blesa [1], S. P. Fekete [2] and A. Kröller [2]

[1] *ALBCOM Research Group, Universitat Politècnica de Catalunya, Barcelona, Spain*
{hhernandez,cblum,mjblesa}@lsi.upc.edu

[2] *Algorithms Group, Braunschweig Institute of Technology, Braunschweig, Germany*
{t.baumgartner,a.kroeller,s.fekete}@tubs.de

***Abstract*— In this work we present a protocol for a self-synchronized duty-cycling mechanism in wireless sensor networks with energy harvesting capabilities. The protocol is implemented in Wiselib, a library of generic algorithms for sensor networks. Simulations are conducted with the sensor network simulator Shawn. They are based on the specifications of real hardware known as iSense sensor nodes. The experimental results show that the proposed mechanism is able to adapt to changing energy availabilities. Moreover, it is shown that the system is very robust against packet loss.**

I. Introduction

Sensor networks [27], [1], [2] consist of a set of small, autonomous devices which may be used, for example, for monitoring large areas and for analyzing complex phenomena for extended periods of time. Currently available hardware includes sensors for a wide range of physical data such as light intensity, humidity, temperature and the oxygen level, as well as for some characteristics of objects such as direction and speed. Thanks to these sensing capabilities sensor networks can be used for a variety of different tasks including environmental monitoring, patient monitoring in health care, and industrial machinery surveillance. Some of these applications may require the nodes of a sensor network to be distributed within wide areas without power supplies. Moreover, sensor nodes might be actively or passively mobile. Therefore, they are generally equipped with batteries, which makes energy a scarce resource.

A basic idea for saving energy is to periodically switch off the sensor nodes. This can, of course, only be done if permited by the targeted application. For instance, consider the case of environmental data monitoring where measurements must be taken in (more or less) regular time intervals. In such a case, sensor nodes can become inactive after each measurement, and they are activated again when the next measurement is scheduled. Not switching off sensor nodes in such a scenario is clearly a waste of energy. The mechanism that establishes the alternation between the active and inactive states is generally called *duty-cycling* (see, for example, [25]). If sensor nodes are equipped with energy harvesting capabilities, the duty-cycling mechanism may be adaptive based on the available energy. In the context of the example mentioned above, and considering the case of sensor nodes equipped with solar panels, this may mean that measurements may be taken more frequently on sunny days than on cloudy days. The literature offers some works on energy-aware duty-cycling in wireless sensor networks (see, for example, [18], [21], [19]). However, the main disadvantage of these approaches is that they require a quite regular pattern for the availability of energy from the environment.

In previous work [17], [16], [15] we introduced and studied a possible technique for energy-aware duty-cycling in (mobile) sensor networks with energy harvesting capabilities. This system was inspired by self-synchronized sleeping patterns of natural ant colonies [24], [14], [11], [10]. In nature, some species of ants rest quite large fractions of their time. Interestingly, not only single ants show this behavior, but whole ant colonies exhibit synchronized activity phases resulting from self-organization ([12], [20], [9]). Apart from achieving duty-cycling in a self-synchronized way, the system that we proposed is able to adatp to changing energy conditions of the individual nodes. The focus of these first studies was purely on the swarm intelligence aspects of the proposed system. The experiments were performed without considering any network constraints such as packet loss, collisions and network failures. Before we outline the contributions of this work, we would like to introduce already a glimpse of the basic behaviour of the system introduced in [17], [16], [15]; see Figure 1. The solid line shows the fraction of active nodes over time, whereas the slashed line shows the average battery level of the nodes over time. Finally, the dotted line represents the sun power that is used to establish the amount of energy which can be harvested by the sensor nodes at each time step. Note that all three measures are scaled to $[0,1]$. At this point we would like the interested reader to understand the following two aspects. First, self-synchronized duty-cycling is indicated by the repetitive appearance of activity peaks over time (see solid line). Second, the adaptation to changing energy conditions is indicated by the changing height of the activity peaks. At times of lower battery levels, activity peaks are lower as well. This is the mechanism used by the sensor nodes to adapt to varying energy conditions.

Contribution of this Work. As mentioned above, the focus of our previous work has been purely on the swarm intelli-

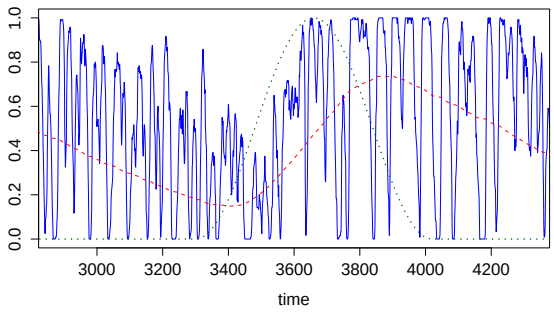


Fig. 1. A first glimpse of the working of energy-aware, self-synchronized duty-cycling.

gence properties of the proposed system. Experiments have been executed with a discrete event simulator without considering packet loss, collisions, etc. The first contribution of this work is therefore the design of a protocol for wireless sensor networks that captures the essential aspects of our swarm intelligence system previously proposed in [17], [16], [15]. The second contribution consists in the implementation of this protocol in Wiselib [4], which is a library of basic algorithms for wireless sensor networks. Algorithms in the Wiselib cover several categories such as routing, localization, clustering, data dissemination, time-synchronization algorithms, target tracking and topology control. In our opinion, it is important to contribute to such libraries, because they assist other researchers in rapidly developing prototype applications for their own purposes without the need of implementing everything from scratch. Finally, we experimentally test our duty-cycling protocol in a real scenario, simulating iSense sensor nodes from Coalesense GmbH [8] with the network simulator Shawn [22].

The organization of the paper is as follows. In Section II the Wiselib is shortly outlined. In Section III the extension of the Wiselib for duty-cycling algorithms is introduced, followed by the description of the protocol for self-synchronized duty-cycling. Finally, experimental results are presented in Section IV, and conclusions and an outlook to future work is given in Section V.

II. Wiselib

The Wiselib [4], [29] is a generic algorithm library for heterogeneous wireless sensor networks. The main design goal is the possibility to run the same implementation of an algorithm on different hardware and software platforms. Not only real sensor hardware such as MicaZ or TelosB nodes are supported by Wiselib, but also simulation environments such as Shawn and TOSSIM. Wiselib intents to be a library of algorithms for heterogeneous wireless sensor networks comparable to some well-known centralized algorithm libraries such as LEDA, CGAL or BOOST. This aim has strong requirements:

- The Wiselib must run on various sensor nodes, from tiny nodes equipped with a MSP430 to powerful nodes running an Intel XScale. It should also easily be ported to additional devices.
- Its algorithms should utilize the capabilities of the device they are compiled for.
- Its memory overhead should be as low as possible compared to a native implementation for a specific device.
- Its algorithms should be highly efficient considering the capabilities of the devices.
- Its algorithm implementations should not explicitly deal with platform specific dependencies.

The library can be used by any user application that needs any of the algorithms implemented in it or, alternatively, that needs to implement a new algorithm using the components that the library offers.

The algorithms included in the Wiselib itself are organized in topics according to their functionality. In order to abstract the algorithms from the particularities of the physical platform of sensors and the operating system administrating that platform, a set of connectors exists that defines and fixes an interface for interacting with them. A connector is also defined to interact with wireless sensor networks simulators such as Shawn [22]. Those connectors are defined in a way that the same algorithm can be run on a physical platform or on the simulator. Details on how this is achieved are provided in the following.

A. C++ Templates

Wiselib is implemented in C++ [23]. This decision allows the use of modern programming techniques via object-oriented design, and provides mostly type-safe development. Moreover, the language offers important features such as const-correctness and templates [26], [3]. Wiselib massively uses templates, especially to develop very efficient and flexible applications. The basic functionality of templates is to allow the use of generic code that is resolved by the compiler when specific types are given. Thereby, only the code that is really needed is generated, and methods and parameters as template parameter can be directly accessed. Object-oriented concepts are implemented using template specializations. Using templates and member templates provides the Wiselib with several advantages, such as early binding, inline optimizations, code pruning, extensibility, and a layered structure. It uses the well-established technique of template-based “concepts” and “models”, where the former are not specified as actual code, but rather as formal specifications in documentation. Models are implementations of concepts. Both concepts and models allow for polymorphism, including multiple inheritance. These techniques are used successfully in standard C++ libraries, such as the STL, BOOST [5], and CGAL [6]. The Wiselib employs these methods in the same manner, i.e., using standard compiler features without custom additions.

B. Wiselib Architecture

In programming terms, an algorithm model implementing a concept is basically a template expecting various parame-

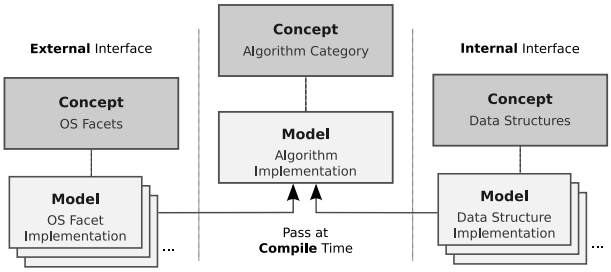


Fig. 2. Wiselib software architecture [4]

ters. These parameters can be both Operation System (OS) facets and data structures. OS facets represent the connection to the underlying operating system or firmware (e.g., the sensor radio or the timers), thus being an abstraction layer to the OS, and thus also to the hardware platform. The concepts and models of this abstraction layer form the so-called *external interface* (see Fig. 2). OS facets are passed to an algorithm as template arguments and the compiler later resolves such calls to the OS. An OS facet can be implemented by different models, that may have different advantages or purposes. The user can pass any of those models to an algorithm at compile time, and no extra overhead is payed for that. Additionally, the so-named *internal interface* is formed by data structure models and concepts, which make the algorithms independent from specific implementations for their required data structures.

In general, this separation of concepts and their implementation through a template-based approach leads to a highly flexible and powerful design, because the necessities, strength and weaknesses of different hardware platforms or different data structures can easily be utilized just by changing the parameters, or even simply by changing makefile targets.

C. The Duty-Cycling Concept

The design of a new algorithm class in the Wiselib requires the definition of a concept, that is, a documentation of how an algorithm looks and behaves. Within the scope of this paper, we provide a generic concept for the class of energy-aware duty-cycling algorithms, and provide one model for this concept: A duty-cycling algorithm based on the behavior of ants, as described in Section III, for the Wiselib.

In general, Wiselib algorithms are particularly used by higher-level applications to simplify the development process. This means, such a duty-cycling algorithm must assist sensor nodes in their decision of being active or inactive. This is handled via a callback to the sensor node when it is supposed to change its mode. Based on this callback, the sensor node is then responsible for the corresponding task, e.g. changing to sleep-mode for a given time interval in case of inactivity, or performing application-oriented tasks in case of being active.

Therefore a duty-cycling algorithm has basically two functionalities: It can be enabled and disabled, and a callback can be (un)registered to indicate changes in activity. The concept looks as follows:

```
1  concept DutyCycling {
2    enum DutyCyclingActivity {
3      DC_ACTIVE, DC_INACTIVE
4    };
5    void enable(void);
6    void disable(void);
7    template<class Callee, void (Callee::*TMethod)(int)>
8      int reg_changed_callback( Callee *obj_pnt )
9    void unreg_recv_callback(int);
10 };
```

With the aid of this generic concept, it is possible to cover a broad range of duty-cycling algorithms. As long as the algorithm is disabled, it does not consume any energy, hence the sensor node is not allowed to use the radio or to do any calculations. After being enabled, it calls each registered callback method whenever the activity state changes.

The exact behavior of a potential duty-cycling model is not mandatory. It can be asynchroneous or synchronized, it may rely on exact time-synchronization or do not have any requiremnts. The important aspect is that the method signatures from the concept are implemented, so that it can be passed to other algorithms as a template parameter.

III. Proposed Duty-Cycling Model

As mentioned before, in [17], [16], [15] some of the authors of this paper have introduced a swarm intelligence technique with a potential application for self-synchronized duty-cycling in (mobile) wireless sensor networks with energy harvesting capabilities. In the following we outline the protocol for the integration of duty-cycling with user applications that we designed on the basis of this work. This protocol was implemented as a *duty-cyling model* in Wiselib. The current version of the protocol assumes that there is a time synchronization algorithm executed by a lower layer of the network. The protocol works in periods. Each period has a length of Δ time units (say, seconds). Each period is divided in two phases: the first phase is dedicated to actions concerning the management of duty-cycling, whereas the second phase is dedicated to application-specific tasks that sensor nodes must perform (see Figure 3). The first phase of each period is very short. In this phase all nodes may receive transmissions from neighboring nodes and themselves they execute one duty-cycling event. The outcome of the first phase decides if a node will be *active* or *inactive* for the rest of the corresponding period. In case of being active a node is available for user-defined applications (environmental data monitoring, tracking, etc). However, if th state of a sensor node is set to inactive the node will turn off the radio and will sleep until the start of the following period.

In the following we focus on the description of the duty-cycling algorithm executed in the first phase of each period. This algorithm consists in a so-called *duty-cyling event* that is executed by each sensor node i exactly once. The time of executing this event is, at the moment, randomly chosen by each sensor node within the first phase of each period. Each sensor node i maintains a state variable S_i. The value of this state variable is initially set to S_{act}, which is a parameter

Fig. 3. Division of time between the duty-cycling mechanism and user applications. The protocol works in a sequence of time periods of length Δ. In each period, the first phase is dedicated to duty-cycling (DC), and the second phase to the user application.

of the mechanism. Moreover, a_i is a binary variable whose value determines if the sensor node is *active* or *inactive* in phase of two of the corresponding period. More specifically, if $a_i = 1$ sensor node i is active, and inactive otherwise. The value of the variable a_i is determined as follows:

$$a_i := \Phi(S_i - \theta_{\text{act}}) \ , \tag{1}$$

where θ_{act} is the so-called activation threshold, and $\Phi(x) = 1$ if $x \geq 0$, and $\Phi(x) = 0$ otherwise. Note that an inactive sensor node can return to the active state either due to local interactions (as explained below in Eq. 2) or spontaneously with a probability p_a and an activity level S_a.

In addition, each sensor node maintains a queue Q_i for incoming duty-cycling messages from neighboring sensor nodes. Each message $m \in Q_i$ contains a single value m_{activity}, which contains the activity S_j of the sensor node j that has sent the message. When sensor node i executes its duty-cycling event, the value of state variable S_i is updated depending on the messages in Q_i. Subsequently sensor node i sends a duty-cycling message, containing the new value of S_i, using a certain transmission power level. Note that the choice of the transmission power level is a crucial compoment for the working of our duty-cycling technique. More specifically, the value of state variable S_i of a sensor node i is computed as follows:

$$S_i := tanh\left[g \cdot \left(S_i + \sum_{m \in Q_i} m_{\text{activity}}\right)\right] \ , \tag{2}$$

where g is a gain parameter whose value determines how fast the value of variable S_i diminishes. After this update, all messages are deleted from Q_i, that is $Q_i := \emptyset$. Note that with this update the value S_i of an inactive sensor may increase sufficiently enough in order to be greater than the activity threshold S_{act}.

The remaining issue is the choice of the power level for the transmission of the duty-cycling messages. Here we assume a standard antenna model which allows sensor nodes—for each transmission—to choose from a finite set $P = \{P^1, \ldots, P^n\}$ of different transmission power levels.[1] More specifically, the choice of a sensor node i depends on its battery level, which is denoted by $b_i \in [0,1]$. Hereby, $b_i = 1$ corresponds to a full battery. In the following we first outline the determination of a so-called *ideal transmission power level*, which then leads to the choice of the real transmission power level. The ideal transmission power level (p_i) of a sensor i depends on the current battery level in the following way:

$$p_i := p_{\min} \cdot (1 - b_i(t)) + p_{\max} \cdot b_i(t) \tag{3}$$

[1]Popular sensor hardware such as iSense nodes or SunSPOTs are equipped with such antennas.

where $p_{\min}$, respectively $p_{\max}$, are parameters that fix the minimum, respectively maximum, transmission power levels. Only when batteries are fully charged the ideal transmission power level may reach $p_{\max}$. The ideal transmission power level is then translated into the *real transmission power level* (T_i) as follows:

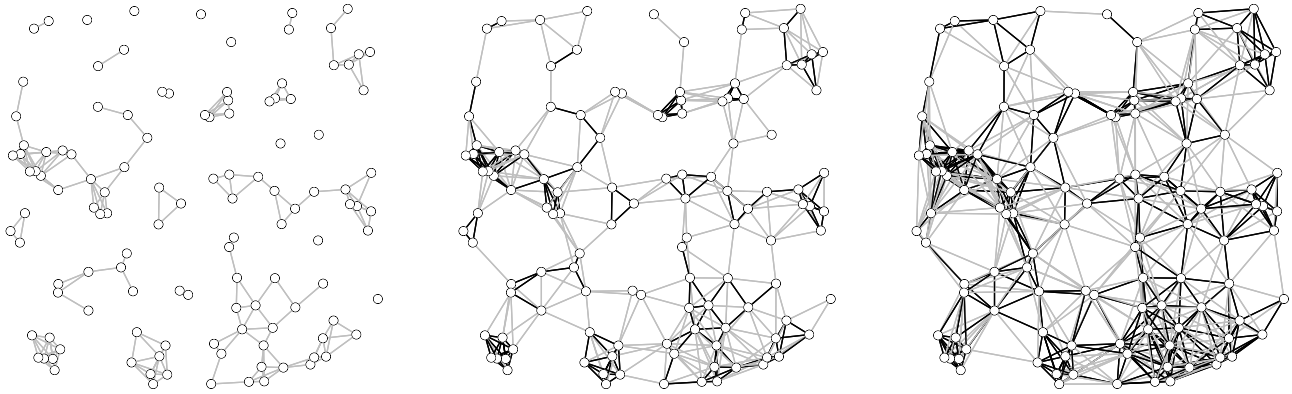
$$T_i := P^k \in P \text{ such that } p_i \in \left(P^k - \frac{P^k - P^{k-1}}{2}, P^k - \frac{P^{k+1} + P^k}{2}\right] \tag{4}$$

At this point it is important to realize that the transmission power level T_i is used only for sending the duty-cycling message. For other messages during the second phase of each period, the user application is responsible for choosing transmission power levels. The duty-cycling event described above is summarized in Algorithm 1.

Algorithm 1 Duty-cycling event of a sensor node i

1: Calculate a_i (see Eq. 1)
2: **if** $a_i = 0$ **then**
3: Draw a random number $p \in [0,1]$
4: **if** $p \leq p_a$ **then** $S_i := S_a$ and $a_i := 1$ **endif**
5: **end if**
6: Determine transmission power level T_i (see Eq. 4)
7: Compute new value for state variable S_i (see Eq. 2)
8: Send duty cycling message m with $m_{\text{activity}} := S_i$ with transmission power level T_i

As mentioned above, the battery level of the sensor nodes is responsible for their choice of a transmission power level for sending the duty-cycling message. Therefore, the battery level of course affects the communication topology in the context of the duty-cycling mechanism. As an example, consider a network of 120 nodes randomly located with static positions in a region of one square kilometer. Suppose that the minimum transmission power level P_1 reaches a distance of 100 m and the maximum transmission power level reaches a distance of 200 m. In Figure 4(c) we show the communication topologies that result from fully charged batteries. Over time nodes will consume energy and, especially during night, battery levels will start to decrease. This decrease in the amount of available energy causes changes in the communication topology due to the effect of Equation 3. The topology obtained with the same node locations as in the previous example but with batteries filled just to half of their maximum capacity is shown in Figure 4(b). Finally, Figure 4(a) shows the induced topology when batteries are nearly empty.



(a) Edges in gray show the communication topology when the transmission power levels are such that distances of up to 100 m are reached.

(b) Edges in black are the edges that were shown in gray in (a). Edges in gray show the new communication links obtained when the transmission power level is increased such that distances of up to 150 m are reached.

(c) Edges in black are all edges from (b). Edges in gray show the new communication links obtained when the transmission power level is increased such that distances of up to 200 m are reached.

Fig. 4. Different communication topologies in a network with 120 randomly located nodes when different transmission power levels are considered.

IV. Experimental Evaluation

In the following we first describe the experimental setup and the experimentation environment before we present the obtained results. The implementation of the presented protocol in the Wiselib provides us with options of executing it on real testbeds but also to perform simulations with sensor network simulators. In the context of this paper we decided for the second option. More specifically we used the sensor network simulator Shawn [22], which is a discrete event simulator with a very high level of parametrization. In particular, it is designed to execute algorithms from the Wiselib. The user can easily run experiments simulating the behavior of different sensor nodes and also add own sensor node specifications. A peculiarity of Shawn is the fact that packet collisions are not explicitly considered. Instead Shawn simulates these collisions and the consequent packet loss under different constraints and in different scenarios.

We decided to experiment with *iSense* sensor node hardware from Coalesenses GmbH [8]. For this purpose we added the specification of *iSense* nodes to Shawn. These sensor nodes use a Jennic JN5139 chip, a solution that combines the controller and the wireless communication transceiver in a single chip. The controller has a 32-bit RISC architecture and runs at 16Mhz. It is equipped with 96kb of RAM and 128kb of serial flash. The maximum transmission power level of iSense nodes reaches a distance of about 500m in all directions in open air conditions. The most important reason for choosing iSense nodes is the fact that they are being used by two of the currently largest European projects on sensor networks: WISEBED [7], [28] and FRONTS [13]. In our simulations, iSense nodes are equipped with solar panels from Coalesenses GmbH. According to their documentation, iSense nodes require $0.025mA$ to work without using any additional peripheral such as the radio or the sensing devices. The state in which the radio is also turned on requires a power supply of $12.8mA$. Additionally, to receive or send a message with 4 bytes of information, as required by duty-cycling messages, implies a consumption of $7.43\mu C$. The batteries have a maximum capacity of $2600\mu C$. Energy harvesting by solar panels can reach a maximum nominal value of 1.6W. This information is summarized in Table I. Finally, let us mention that iSense nodes offer 6 possible transmission power levels, in addition to the state in which the radio is turned off. The five transmission power levels other than the maximum one are obtained by reducing the maximum transmission power level by $\frac{1}{6}$, $\frac{2}{6}$, $\frac{3}{6}$, $\frac{4}{6}$, and $\frac{5}{6}$.

TABLE I

Description of the power devices and parameters for the energy model.

Data	Device specifications
Tx/Rx (4 bytes)	$7.43\mu C$
Radio On	12.8mA
Radio Off	0.025mA
Battery capacity	$2600\mu C$
Energy harvesting (f)	1.6W
Max. Tx Power	500m

One of the aspects that has not been described so far is the simulation of the light source for energy harvesting. This was done as follows. The light source at time $z > 0$ has an intensity of $s(z) \in [0,1]$. Hereby, $s(z) = 0$ corresponds to absolute darkness. In [17] we described a model for the evolution of the sun light intensities, that is, for the evolution of $s(z)$ over time. Here we consider exactly the same model. Additionally, we assume a variable cloud density $c(z) \in [0,1]$. Depending on the technical characteristics of the solar panels used, sensor nodes can transform a fraction f of the available light intensity into energy:

$$e_i^{\text{harv}}((t-\Delta,t]) := f \cdot \int_{t-\Delta}^{t} s(z) \cdot (1-c(z)) \qquad (5)$$

In the experiments presented in this article we do not consider any specific user application, that is, the energy consumption of phase two of the proposed protocol must be simulated. This is done by removing an amount of e_{app}

TABLE II

PARAMETER SETTINGS FOR THE SIMULATION OF DUTY-CYCLING.

$p_{\min}$	$p_{\max}$	g	p_a	e_{app}	f
0.07	0.14	0.1	0.001	0.001	0.0027

of energy from the battery for each execution of phase two. Finally, Table II presents all parameter values that we choose for the simulations. It is important to note that the information which refers to the power profile of the iSense nodes is obtained by properly rescaling the values from Table I to the $[0,1]$ range that is used by the description of duty-cycling given in Section III.

A. Experiments

Assuming that Δ—that is, the length of one period—corresponds to 60 seconds, the simulations that we conducted span 30 days (each day consists of 1440 periods). The first phase of each perido, which is reserved for the duty-cycling events, was given 0.05 seconds. Information about the state of the sensor nodes (active versus inactive) is collected at the start of each period. The most important measure taken is the *mean activity* of the sensor network, which is measured—at any time—as follows:

$$A := \frac{1}{k}\sum_{i=1}^{k} a_i \in [0,1] \qquad (6)$$

Note that, the greater A the more sensors are active at the specific time at which A was determined. Self-synchronization behavior is characterized by an oscillating value of A over time. This was shown already for a mobile sensor network with $k = 120$ sensor nodes in Figure 1 of the introduction (see solid line). However, the results from this figure were obtained in a *perfect environment* with no collisions or transmission failures and no propagation times. Moreover, the energy model that was used had no relation to real sensor node hardware.

The experiments that we present in this section aim at proving the applicability of the proposed mechanism in real sensor networks. All experiments are done on the basis of a static network of $k = 120$ iSense nodes as simulated by the Shawn sensor network simulator. For the first experiment that we conducted we assumed a cero probability for packet collisions. Moreover, we assume a cloud density of zero, that is, $c(z) = 0$ for $z \geq 0$. Figure 5 shows the obtained duty-cycling behavior. Again, the solid line shows the fraction of active sensor nodes over time, whereas the slashed line shows the average battery level of the sensor nodes over time. Finally, the dotted line represents the sun power that is used to establish the amount of energy which can be harvested by the sensor nodes at each time step. The graphic shows the behavior for one day of simulation, that is, 1440 periods. Self-synchronized duty-cycling is indicated by the appearance of activity peaks over time. It is remarkable how the system adapts to the available energy resources, reducing the height of the peaks when the battery level of the nodes is reduced. Note that when a lot of energy is available the system can even prescind from switching off sensor nodes. This can be seen by the existence of a large activity peak of about 200 periods of length located around period 14000. Note that for this experiment the average fraction of nodes that are active at each period is approximately 0.6. This measure will henceforth be called the *mean system activity*.

TABLE III

DISTRIBUTION OF THE ENERGY CONSUMPTION IN THE DUTY-CYCLING PROTOCOL

	Task	Energy (%)
Duty-cycling	Tx	0.757
	Rx	18.591
	Idle	0.001
	Active	0.035
User application		80.616

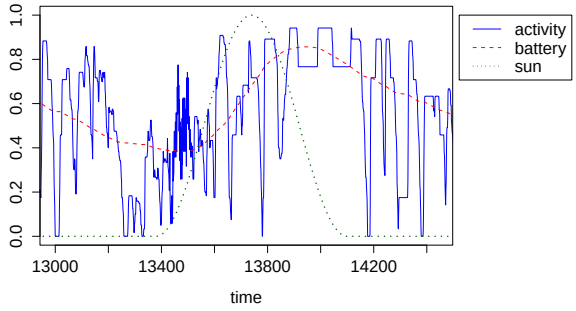


Fig. 5. Simulation results of the duty-cycling protocol obtained from a network with 120 sensor nodes during the 11th day of simulation. The solid line shows the fraction of active nodes in each period, the dashed line shows the average battery level over time, and the dotted line shows the evolution of the sun light intensity.

In the following we present a study of how the energy of the sensor network is spent. Table III presents the energy consumed by the user application against the energy consumed by the duty-cycling mechanism over a whole simulation of 43200 periods (that is, 30 days). The energy spent by duty-cycling is hereby split into the ”Idle” and ”Active” states as well as the energy spent for transmitting the duty-cycling messages (Tx) and receiving duty-cycling messages (Rx). Concerning duty-cycling, note that message reception is the task which consumes most of the energy. In total, the duty-cycling mechanism consumes approximately 20% of the total amount of spent energy. This may seem quite high at first. However, consider that this percentage is strongly dependent on the value of e_{app}, which we have set to a very moderate value of 0.001. Increasing this value will obviously cause the decrease of the percentage of energy spent by duty-cycling.

After these initial studies we will now test the duty-cycling mechanism in two adversarial scenarios. In first place, it is shown how the system responds to situations with communication failures. In second place, the behavior of the system is studied in scenarios where energy harvesting

is limited, for example, due to cloudy weather. Finally, we present a mechanism for the automatic parameter adaptation of the system for what concerns different network sizes. This is an important aspect for an algorithm included in a generic algorithm library such as Wiselib.

1) Effect of Packet Loss: With the next experiment we aim at studying the robustness of the system with respect to communication failures. The experiment consists in simulating the duty-cycling protocol under different packet loss rates. A packet loss rate of $p_{loss} \in [0,1]$ means that the probability of correctly transmitting any message is $1 - p_{loss}$. We repeated the initial experiment as outlined above for all packet loss rates between 0 and 1, in steps of 0.01. The results are shown in terms of the obtained *mean system activity* for each considered packet loss rate in Figure 6(a). It can be observed that the behavior of the system does not visibly change until a packet loss rate of about 0.3. This means that the proposed system is surprisingly robust against packet loss. Only for packet loss rates greater than about 0.3 the system behavior degrades.

2) Limited Energy Harvesting: Another interesting question concerns the possible change in system behavior when cloud densities greater than zero are considered; in other words: when energy harvesting is limited by bad weather. Therefore, we repeated the initial experiment for a range of different cloud densities between 0 and 1. Figure 6(b) shows the evolution of the obtained mean system activity when moving from low to high cloud densities. As expected, with increasing cloud density the mean system activity decreases. Interestingly, the relation between cloud density and the mean system activity is linear.

3) Adapting to Different Network Sizes: So far we have only studied sensor networks with 120 sensor nodes. However, when changing the size of the network it is intuitively clear that at least some parameter values must be adjusted in order to maintain a functional system. Note that when changing the network size, the node density changes. Hence, it is reasonable to assume that for maintaining the shape of the activity peaks, the choice of the transmission power level and the probability of spontaneous activation should be adapted to the new network size. A way of obtaining the new system parameters is described in the following. With k_{new}, $p^a{}_{\text{new}}$ and t^{new} we refer to the new number of sensor nodes, the probability of spontaneous activation and the transmission power level of the new, differently sized, network. First, in order to obtain the same wake-up rates as in the case of a 120-node network, the following rule can be applied:

$$p^a{}_{\text{new}} := p^a \cdot \frac{k}{k_{\text{new}}} \quad , \tag{7}$$

where p^a and k are the parameters from the original network. Note that this rule increases the probability of spontaneous activation of the nodes when the network population is decreased, and vice-versa when the number of nodes increases. Moreover, the average number of nodes' spontaneous activations per time unit are maintained. Next we introduce a rule for adapting the transmission power level. The basic idea is to have a constant average number of sensors being reached by a transmission. Due to the fact that the sensor nodes form a random topology at any moment in time, the following reasoning was used. In general, the number of nodes that can be reached by the ideal transmission of a sensor can be estimated as follows:

$$\pi \cdot t \cdot 2 \cdot \frac{k}{A} \quad , \tag{8}$$

where k is the number of sensor nodes and A is the space in which the sensor nodes reside. In our case it holds that $A = 1^2 = 1$. Therefore, Eq. (8) reduces to $\pi \cdot t \cdot 2 \cdot k$. As t is known for the case of 120-node networks, an adjusted transmission power level can be calculated for networks of different sizes as follows:

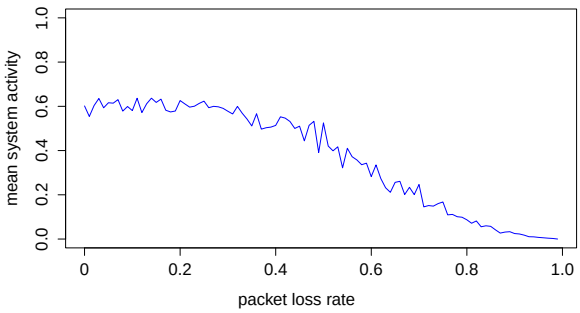
$$t^{\text{new}} = \sqrt{\frac{t \cdot 2 \cdot k}{k_{\text{new}}}} \quad , \tag{9}$$

where k_{new} is the size of the new network, and t^{new} is the transmission power level for the new network. However, remember that the transmission power level is not directly modifiable as an algorithm parameter. The only parameters of our algorithm for what concerns to the transmission power level are $p_{\min}$ and $p_{\max}$. These values are used as the boundaries of the region for the ideal transmission power levels. Therefore, our scaling method consists in using Eq. 9 for obtaining $p^{\text{new}}_{\min}$ and $p^{\text{new}}_{\max}$, which delimit the value of the ideal transmission power level of the new network.

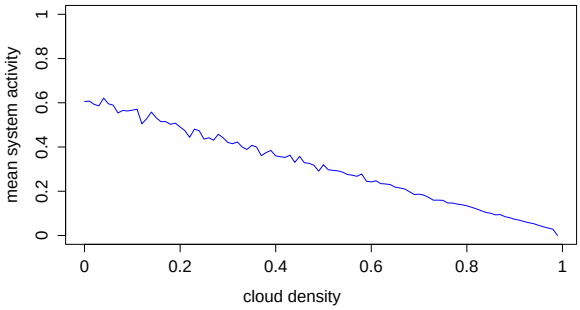
With this method for scaling p^a, $p_{\min}$ and $p_{\max}$ we have repeated the initial experiment for a range of different network sizes $k \in [0,300]$. The graphic in Figure 6(c) shows the evolution of the resulting mean system activity. Ideally we would have expected a more or less straight line of height about 0.6. This would have meant that the introduced parameter scaling method leads to a system that behaves equally for all network sizes. However, as can be seen, the scaling method works for networks with more than 100 nodes. For smaller networks, the system behaviour is changing in the sense that the mean system activity decreases. However, this can be explained by the decreasing connectivity and communication ability when the network size decreases.

V. Conclusions and Future Work

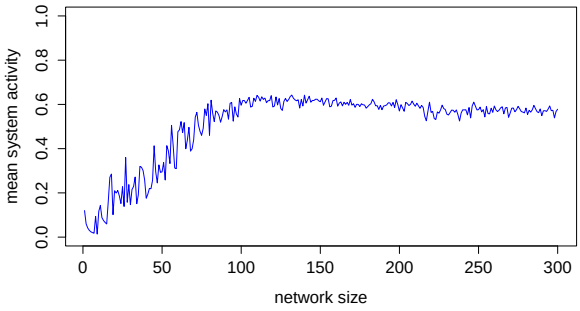
In this paper we have introduced a protocol for self-synchronized duty-cycling in wirless sensor networks with energy harvesting capabilities. The proposed duty-cycling technique was inspired by the working of real ant colonies. The protocol has been implemented in Wiselib, a library of generic algorithms for sensor networks, and experiments have been performed with the network simulator Shawn simulating iSense hardware. The obtained results show that the proposed duty-cycling mechanism is able to adapt to chaning energy conditions. Moreover, the technique is very robuts for what concerns packet loss and changing weather conditions. In the future we plan to combine this protocol with user applications such as monitoring or tracking.



(a) Results for different packet loss rates.



(b) Results for different cloud densities.



(c) Results for different network sizes.

Fig. 6. Behavior of the duty-cycling mechanism under varying conditions

ACKNOWLEDGEMENTS

This work was supported by the EU project FRONTS (FP7-ICT-2007-1) funded by the European Commission under the FET Proactive Initiative *Pervasive Adaptation*. In addition, Christian Blum acknowledges support from the *Ramón y Cajal* program of the Spanish Government of which he is a research fellow, and Hugo Hernández acknowledges support from the Catalan Government through an *FI* grant.

REFERENCES

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393 – 422, 2002.

[2] IF Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *IEEE communications magazine*, 40(8):102–114, 2002.

[3] A. Alexandrescu. *Modern C++ Design: Generic Programming and Design Patterns Applied*. Addision-Wesley, 2001.

[4] T. Baumgartner, I. Chatzigiannakis, S. Fekete, C. Koninis, A. Kröller, and A. Pyrgelis. Wiselib: A generic algorithm library for heterogeneous sensor networks. In J. Sa Silva, B. Krishnamachari, and F. Boavida, editors, *Proceedings of EWSN 2010 – 7th European Conference on Wireless Sensor Networks.*, volume 5970 of *LNCS*, pages 162–177. Springer Berlin, 2010.

[5] BOOST. http://www.boost.org.

[6] CGAL: Computational Geometry Algorithms Library. http://www.cgal.org.

[7] I. Chatzigiannakis, S. Fischer, C. Koninis, G. Mylonas, and D. Pfisterer. WISEBED: an open large-scale wireless sensor network testbed. In Nikos Komninos, editor, *Proceedings of SENSAPPEAL 2009 – 1st Intl. Conference on Sensor Networks Applications, Experimentation and Logistics*, volume 29 of *Lecture Notes of the Institute for Computer Sciences, Social-Inf*, pages 68–87. Springer, 2009.

[8] Coalsesenses GmbH. http://www.coaelesenses.com.

[9] B. J. Cole. Short-Term Activity Cycles in Ants: Generation of Periodicity by Worker Interaction. *American Naturalist*, 137(2):244, 1991.

[10] B.J. Cole. The social behavior of Leptothorax allardycei (Hymenoptera, Formicidae): time budgets and the evolution of worker reproduction. *Behavioral Ecology and Sociobiology*, 18(3):165–173, 1986.

[11] N. R. Franks and S. Bryant. Rhythmical patterns of activity within the nest of ants. *Chemistry and Biology of Social Insects*, pages 122–123, 1987.

[12] N. R. Franks, S. Bryant, R. Griffiths, and L. Hemerik. Synchronization of the behaviour within nests of the ant Leptothorax acervorum (fabricius)-I. Discovering the phenomenon and its relation to the level of starvation. *Bulletin of Mathematical Biology*, 52(5):597–612, 1990.

[13] FRONTS - Foundations of Adaptive Networked Societies of Tiny Artefacts. http://fronts.cti.gr.

[14] J. M. Herhers. Social organisation in Leptothorax ants: within- and between-species patterns. *Psyche: A Journal of Entomology*, 90(4):361–386, 1983.

[15] H. Hernández and C. Blum. Asynchronous simulation of a self-synchronized duty-cycling mechanism for mobile sensor networks. In *Proceedings of BADS 2009 – Workshop on Bio-inspired Algorithms for Distributed Systems*, pages 61–68. ACM press, New York, NY, 2009.

[16] H. Hernández and C. Blum. Self-synchronized duty-cycling in sensor networks with energy harvesting capabilities: the static network case. In F. Rothlauf et al., editor, *Proceedings of GECCO 2009 – Genetic and Evolutionary Computation Conference*, pages 33–40. ACM press, New York, NY, 2009.

[17] H. Hernández, C. Blum, M. Middendorf, K. Ramsch, and A. Scheidler. Self-synchronized duty-cycling for mobile sensor networks with energy harvesting capabilities: A swarm intelligence study. In *Proceedings of SIS 2009 – IEEE Swarm Intelligence Symposium*, pages 153–159. IEEE press, Piscataway, NJ, 2009.

[18] J. Hsu, S. Zahedi, A. Kansal, M. Srivastava, and V. Raghunathan. Adaptive duty cycling for energy harvesting systems. In *Proceedings of ISLPED 2006 – International Symposium on Low Power Electronics and Design*, pages 180–185. IEEE press, 2006.

[19] Aman Kansal, Jason Hsu, Sadaf Zahedi, and Mani B. Srivastava. Power management in energy harvesting sensor networks. *Transactions on Embedded Computing Sys.*, 6(4):32, 2007.

[20] O. Miramontes. Complexity and Behaviour in Leptothorax Ants. *Master Thesis*, 1992.

[21] V. Raghunathan, A. Kansal, J. Hsu, J. Friedman, and M. Srivastava. Design considerations for solar energy harvesting wireless embedded systems. In *Proceedings of IPSN 2005 – Fourth International Symposium on Information Processing in Sensor Networks, 2005.*, pages 457–462, 2005.

[22] S. Fischer S. Fekete, A. Kröller and D. Pfisterer. Shawn: The fast, highly customizable sensor network simulator. In *4th Intl. Conference on Networked Sensing Systems (INSS)*, page 299, 2007.

[23] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley, 2000.

[24] J.H. Sudd. *An Introduction to the Behaviour of Ants*. Edward Arnold. London, GB, 1967.

[25] D. Tian and N. D. Georganas. A node scheduling scheme for energy conservation in large wireless sensor networks. *Wireless Communications and Mobile Computing*, 3:271–290, 2003.

[26] D. Vandevoorde and N.M. Josuttis. *C++ Templates: The Complete Guide*. Addision-Wesley, 2003.
[27] D. Wagner and R. Wattenhofer, editors. *Algorithms for Sensor and Ad Hoc Networks*, volume 4621 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin, Germany, 2007.
[28] WISEBED - Wireless Sensor Network Testbeds. http://www.wisebed.eu.
[29] WISELIB. http://www.wiselib.org.