Available online at www.sciencedirect.com
ScienceDirect
J. Parallel Distrib. Comput. 67 (2007) 302–317
Journal of Parallel and Distributed Computing
www.elsevier.com/locate/jpdc

Energy optimal data propagation in wireless sensor networks☆

Olivier Powell*, Pierre Leone, José Rolim

Department of Computer Science, Centre Universitaire d'Informatique, University of Geneva, 24 rue du General Dufour 1205, Geneva, Switzerland

Received 30 September 2005; received in revised form 29 September 2006; accepted 28 October 2006

Abstract

We propose an algorithm to compute the optimal parameters of a probabilistic data propagation algorithm for wireless sensor networks (WSN). The probabilistic data propagation algorithm we consider was introduced in previous work, and it is known that this algorithm, when used with adequate parameters, balances the energy consumption and increases the lifespan of the WSN. However, we show that in the general case achieving energy balance may not be possible. We propose a centralized algorithm to compute the optimal parameters of the probabilistic data propagation algorithm, and prove that these parameters maximize the lifespan of the network even when it is not possible to achieve energy balance. Compared to previous work, our contribution is the following: (a) we give a formal definition of an optimal data propagation algorithm: an algorithm maximizing the lifespan of the network. (b) We find a simple necessary and sufficient condition for the data propagation algorithm to be optimal. (c) We constructively prove that there exists a choice of parameters optimizing the probabilistic data propagation algorithm. (d) We provide a centralized algorithm to compute these optimal parameters, thus enabling their use in a WSN. (e) We extend previous work by considering the energy consumption per sensor, instead of the consumption per slice, and propose a spreading technique to balance the energy among sensors of a same slice. The technique is numerically validated by simulating a WSN accomplishing a data monitoring task and propagating data using the probabilistic data propagation algorithm with optimal parameters.
© 2006 Elsevier Inc. All rights reserved.

Keywords: Wireless sensor networks; Ad-hoc networks; Energy-balancing; Data propagation algorithm; Lifespan maximization

1. Introduction

Wireless sensor networks (WSN) are composed of sensor nodes which are small electronic devices equipped with computing resources (CPU), environment sensing capabilities and wireless links used in a multi-hop fashion to build a network structure [RAdS+00, WLLP01, SL05]. Sensor nodes usually have restricted resources and this constrains the design of distributed algorithms running on top of sensor networks. In this paper we specifically address the constraint of energy consumption, motivated by the fact that sensors are usually battery powered. We consider one of the most common scenarios where sensors have to report sensed events to a particular node of the network, called the sink, and we analyze the lifespan of a distributed probabilistic data propagation algorithm. Since sensors propagate data to the sink in a multi-hop fashion [BN04, AKK05, AY05], evenly balancing the energy consumed among the entire set of sensors increases the lifespan of the network. The probabilistic data propagation algorithm we consider for balancing the energy was first introduced in [ENR04]. It allows each sensor responsible for propagating data to choose between sending it to a next hop sensor, a procedure which requires a relatively small amount of energy, and sending the data directly to the sink, a procedure which requires a long hop and hence a relatively large amount of energy. The algorithm we present computes optimal parameters for the probabilistic data propagation algorithm. These parameters control the ratio of data sent directly to the sink and the ratio sent to a next hop neighbour, and depend on the network topology and the distribution of sensed events. The probabilistic data propagation algorithm we consider has already been used in [ENR04, ENR06, LNR05] and the main contributions of this paper are to formally prove the connection between energy balancing and lifespan maximization, define the optimality of data

☆ Research partially funded by Swiss SER Contract No. 05.0030 and the *Swiss National Science Foundation* (SNF).
* Corresponding author. Fax: +4122 3797780.
E-mail addresses: olivier.powell@cui.unige.ch (O. Powell), pierre.leone@cui.unige.ch (P. Leone), jose.rolim@cui.unige.ch (J. Rolim).

0743-7315/$ - see front matter © 2006 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2006.10.007

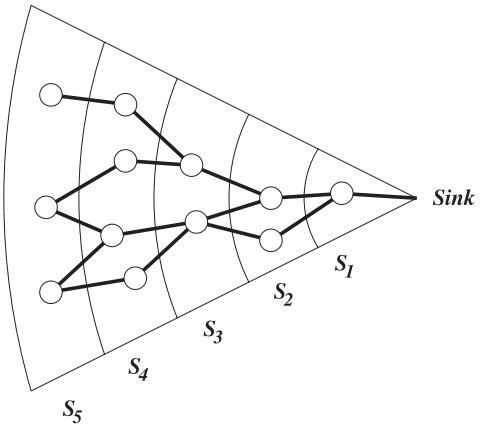

Fig. 1. Network division into slices.

propagation algorithms as maximizing the lifespan of the network and prove that an optimal probabilistic data propagation algorithm always exists. Moreover, we provide an algorithm computing off-line the probability of sending data directly to the sink ensuring the optimality of the probabilistic data propagation algorithm in terms of lifespan of the network.

A typical application is to use the centralized algorithm to compute at the sink level the optimal parameters and to broadcast them in the WSN, thus letting each node fulfill its role in the distributed optimal data propagation algorithm. The input to the algorithm running at the sink level is a description of the network in terms of density of sensors per region and relative frequencies of sensed events per region. These could be estimated statistically or observed dynamically during run-time of the WSN, as proposed in [LNR05]. The formal analysis of the distributed probabilistic propagation algorithm is based on modelling the network as a succession of slices and balancing the energy consumed between the slices. The division of a WSN into slices is illustrated in Fig. 1. The energy balancing among sensors belonging to the same slice is not considered by the algorithm. Actually, numerical experiments discussed in this paper show that the energy is usually not balanced among those sensors and a spreading technique is introduced and numerically validated to ensure energy balancing among all sensors composing the network.

The paper is organized as follows. In Section 2, we give a brief survey of related work. In Section 3, we introduce an appropriate mathematical model of a WSN, together with notational conventions and some preliminary technical results. Section 4 presents the off-line centralized algorithm computing the parameters needed to adjust the distributed probabilistic data propagation algorithm. Section 5 is a formal proof of the optimality of the probabilistic data propagation algorithm when using the parameters computed by the centralized algorithm. Section 6 presents numerical validations of the data propagation algorithm as well as a simple spreading technique to overcome the unbalanced energy among sensors of the same slice, an issue which was not taken into account by the mathematical model of Section 3. Finally, we draw conclusions in Section 7.

2. Related work

Minimizing the energy consumption has been considered under various approaches: multi-hop transmission techniques [IGE00, CNS02], clustering techniques [HCB00], alternating power saving modes [STGS02], varying transmission levels with route selection [CT00], energy replenishment [LSS05], multi-path routing [HGWC02], combination of sleep/awake and probabilistic forwarding techniques [BCN05] are among existing strategies. However, these strategies minimize the energy consumption without taking into account the overuse of some bottleneck regions of the network. These regions will prematurely run out of energy and eventually disconnect the multi-hop network, even if most of the sensors still have enough energy to keep running. In Section 2.1, we give a survey of energy balancing algorithms for WSNs, not all of them being data gathering algorithms. In Section 2.2, we present alternative approaches to energy aware data gathering in WSNs which differ from the approach used in this paper, either because the lifetime metric for the network is different, not based on energy balancing, or because a radically different approach, e.g. by allowing the sink to be mobile, is used.

2.1. Maximizing the lifetime of sensor networks by balancing energy consumption among sensors

The idea of balancing energy consumption in the network can be traced back to [SP03, YP03, GLW03]. To our knowledge, the first solution to the premature energy depletion of sensors close to the sink was proposed in [GLW03]. This problem has since then been investigated in a setting similar to ours, where the network is divided into slices and energy consumption studied at the slice level, in a series of papers [ENR04, ENR06, LNR05, JLPR06, SD05, OS06]. Sichitiu and Dutta [SD05] consider varying battery levels between slices, called rings, while Olariu and Stojmenovic [OS06] allow varying emission ranges between slices, called coronas; however, in both cases the data propagation is exclusively hop-by-hop. On the other hand, in order to avoid or minimize the premature depletion of sensors close to the sink, the authors in [ENR04, ENR06, LNR05, JLPR06] use hop-by-hop transmission combined with *ejections*, first introduced in [ENR04].

In [SP03] the authors consider the particular process of sorting in wireless networks and show (among other things) that sorting can be done in $\mathcal{O}(n \log(n))$ time and energy. During the sorting time period, no node is awake for more than $\mathcal{O}(\log(n))$ time steps. Notice that sorting can actually be done in $\mathcal{O}(n)$ time and energy but no algorithm balancing the energy with this complexity is known.

In [YP03] the authors consider the problem of task allocation by formulating an associated integer linear problem. The tasks have to be allocated periodically to the nodes. Besides the ILP formulation they propose a heuristic to solve the problem and show experimentally that it is efficient on small scale problems. The total energy consumption is the cumulative consumption

due to computations and transmissions which are assumed to occur without collisions (for instance if the network is synchronized and nodes transmit according to a time division (TDMA) scheme). The energy consumption due to computations may be lowered by reducing the running frequency. This approach is called dynamic voltage scaling (DVS) and takes advantage of the fact that energy consumption of an electronic component is proportional to fV^2 with f the frequency and V the voltage power supply (the energy consumption per cycle is proportional to V^2). The heuristic is based on three phases. In the first phase (1) the voltage power supply is assumed maximal and the tasks are grouped into clusters with the objective of minimizing the overall execution time. In the second phase (2) the clusters are assigned to nodes, so that the energy dissipation is proportional to the remaining energy. In the third phase (3) the DVS device is used to maximize the lifetime. It is experimentally shown that results obtained with this heuristic improve by 125–250% the solution of the ILP formulation without DVS. We emphasize that these works solve the energy-balancing problem with centralized computations.

In [GLW03] the problem considered is data gathering with a single sink while balancing the energy between sensors. This work is extended by the same authors in [LXG05] in which they consider a division of the network into slices, see Fig. 1, composed of sensors at nearly equal distance to the sink and able to send the data to sensors belonging to the next slice (towards the sink) or directly to the sink. They assume that the energy used to send data directly to the sink is proportional to the square of the distance. In order to balance the energy between sensors, they divide time into two different phases: during the first one sensors send data to the sink and during the other one data is sent to the next slice, the ratio between the two periods of time depends on the slice and needs to be found in order to balance energy. In their work the optimal ratio, ensuring the balance of energy consumption, is found by simulation of the process.

In [ENR04] the problem of finding an energy-balanced solution to data propagation in WSNs using a probabilistic algorithm was considered for the first time. The lifespan of the network is maximized by ensuring that the energy consumption in each slice is the same. Sensors are assumed to be randomly distributed with uniform distribution in a circular region or, more generally, the sector of a disk. Data have to be propagated by the WSN towards a sink located at the centre of the disk, and it is shown that energy balance can be achieved if a recurrence relation between the *probabilities that a slice ejects a message to the sink* is satisfied.

In [LNR05], a more general case is studied: events may not happen according to a uniform distribution, and the sensors may not be distributed uniformly over the area to be monitored. Moreover, it is assumed that the distribution of events is unknown. A solution to the energy-balancing problem is then computed on-line by a centralized algorithm while the distribution is inferred from observations of the events. The idea is then to broadcast periodically the updated parameters, and it is shown that the algorithm converges to the energy-balanced solution when this solution exists.

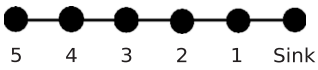

Fig. 2. Line model of the sliced sensor network.

We point out that neither [ENR04, ENR06] nor [LNR05] consider the case where an energy-balanced solution does not exist. Results obtained in the present paper show that an energy-balanced solution does not always exist and, in such a case, the algorithms of [ENR04, ENR06, LNR05] are useless. Our algorithm is a generalization of these two approaches since it finds the same optimal and energy-balanced solution when it exists, and finds an optimal non-energy-balanced solution otherwise.

In [JLPR06], a distributed algorithm is proposed. Each sensor is allowed to transmit data directly to the sink or to neighbour nodes. The algorithm is inspired by the *gradient based routing* family of protocols from [SS01] and uses a local heuristic to balance energy consumption between neighbour nodes. It is shown experimentally that the algorithm converges to an energy-balanced solution in the case of uniform distribution of events and sensors. An analytical proof of the convergence of the algorithm is also given using Markov chain theory in the restricted line model which is illustrated in Fig. 2.

2.2. Other strategies to maximize the lifetime of sensor networks

The strategies described in Section 2.1 are closely related to the strategy for maximizing the lifetime of WSNs proposed in this paper. However, different approaches for maximizing the lifetime of a WSN accomplishing a data gathering task exist and are summarized in this section. We focus on the main ideas and do not try to track every contribution from an historical point of view. Although we do not claim to be exhaustive, we try to cover the contributions which are relevant to the results presented in this paper. Moreover, we do not incorporate contributions which are mainly incremental in order to emphasize the fundamental ideas. The interested reader is referred to survey books or articles such as [Bou05, AKK05, AY05, BN04].

The LEACH protocol (low energy adaptive clustering hierarchy) is presented in [HCB02] as a protocol to achieve good performance in terms of sensor lifetime. The protocol is constrained to (a) run on an easily deployed sensor network, (b) increase the lifetime of the network, (c) be respectful of the latency. The protocol is based on self-organization of the network in clusters with cluster head election. In order to consume energy evenly the cluster head changes through time and distributed signal processing is used in order to minimize the data traffic.

In [OS06] the authors consider the data gathering problem and the sliced network as introduced in the preceding section, see Fig. 1. From the energy point of view, they consider the total amount of energy needed to convey data from a source node to the sink. They consider the problem of network design and assume that sensor parameters can be set according to the position of the sensor. More precisely they allow for the range

of emission to vary depending on the slice the sensor belongs to. The sliced network thus constructed has slices of varying sizes and the aim is to find the appropriate values for the ranges of emission in order to maximize the lifetime of the network. The assumptions on which the computations are based are (a) each sensor is equally likely to be the source of data and (b) each sensor in a given slice is equally likely to be required to handle data from a previous sensor (the model is similar to the one in Fig. 2).

With this model they prove that in order to minimize the total energy spent on routing along a path, every slice must have the same size, and point out that this result is not satisfactory since uneven energy depletion is observed in this situation. Hence, they specifically consider the problem of balancing the energy among slices. However, it turns out that in the situation where the power attenuation factor is 2, uneven energy depletion is intrinsic to the system and cannot be avoided with this strategy.

As sensor networks are deployed in order to monitor an area, one can make the lifetime of the network longer by avoiding monitoring the entire region at all times. This leads to the concept of α-lifetime which is the time duration during which at least an α portion of the region is monitored by the WSN. Stated in this way the problem to be solved is to schedule the sensors' activity in order to maximize the lifetime while still satisfying the constraint on the size of the region to be monitored at any time [ZH04, ZH05]. In [ZH04] the authors provide an upper bound of α-lifetime for large sensor networks. In [ZH05] a centralized algorithm is proposed and numerically validated.

Another new metric of the network lifetime is the *lifetime per unit cost* [CCZ05] which is the average network lifetime divided by the number of sensors enabled. This measure is based on the observation that the lifetime of networks increases monotonically with the number of sensors involved in the network. However, the lifetime per unit cost decreases for large values of the total number of sensors after reaching a maximal value. This shows that efficient network design would need to take into consideration the tradeoff between the total number of sensors and the desired lifetime. Moreover, a longer lifetime is achieved by dividing sensors in small groups and switching on only one in each group each time.

Classical optimization methods are relevant for lifetime maximization. In [GK05], it is assumed that sensors have to send data periodically towards the sink and the following linear program is to be solved in order to minimize the energy required for each period of time:

$$\text{minimize} \quad z \tag{1}$$

$$\text{subject to} \tag{2}$$

$$\frac{1}{E_i}\left(\sum_{0\leqslant j\leqslant n}\lambda_{ij}f(d(i,j))+\sum_{1\leqslant j\leqslant n}\lambda ij\right)\leqslant z,$$

$$1\leqslant i\leqslant n, \tag{3}$$

$$\sum_{0\leqslant j\leqslant n}\lambda_{ij}-\sum_{1\leqslant j\leqslant n}\lambda_{ji}=b_i,\quad 1\leqslant i\leqslant n, \tag{4}$$

$$\lambda_{ij}\geqslant 0,\quad 1\leqslant i,j\leqslant n, \tag{5}$$

where b_i is the amount of data to be transmitted, n the number of sensors, λ_{ij} the fraction of data sent from node i to j, f a function describing the energy consumption as a function of the distance of emission and f_R the energy consumption for receiving data. The first constraint expresses the minimization of the energy consumption and the second one is a flow preservation constraint. A closed solution of this linear program is exhibited for various particular topologies of the network. In particular, they show that for the line model, see Fig. 2, the routing strategy ensuring maximum lifetime consists only of choosing between sending data directly to the next slice or directly to the sink, as is done in [ENR04, ENR06, LNR05] as well as the present paper. Actually, we independently proved this result in [JLPR06] using different technical tools.

The mobility of the sink [Luo06, LH05, LH06] is also an alternative to increase the lifetime of a sensor network accomplishing a data gathering task. In [LH06] the authors consider different approaches to apply mobility and suggest a theoretical framework in which formal analysis can be done. Given the flow of data the problem can be stated as a linear program. However, it is proved in [Luo06] that the problem is NP-hard. Hence, based on duality theory, approximation algorithms are investigated in [LH06].

3. Model and notations

The model we study is the same as the one from [ENR04, ENR06, LNR05, JLPR06] and resembles the models of [SD05, OS06]. It is based on the division of a *sensor network* in *slices*. To define slices, we first consider the unit disc graph built upon the sensor network with a vertex for each sensor, including the sink, and an edge between any pair of sensors which are at a distance less than 1 from one another. The first slice, S_1, is composed of all the sensors located in the unit disc around the sink. The kth slice S_k is defined to be the union of sensors located k hops away from the sink, as illustrated on Fig. 1. When considering a slice S_k, we use the convention of calling S_{k+1} the *previous slice* to S_k, while S_{k-1} is the *next slice*. By convention, S_1 is the *first slice*, and the letter N is used for the *last slice* S_N, thus the slices range from S_1 to S_N. When a slice, for example S_i, needs to send a message to the sink, it can send it to the next slice, S_{i-1}. However, as already mentioned in the introduction, to ensure energy balance we follow [ENR04, ENR06, LNR05, JLPR06] and assume that a slice also has the option of sending a message directly to the sink. This long hop is significantly different from the hops which are used to send a message from a slice to the next slice, since it implies a larger amount of energy consumption. Indeed, the energy consumption required to send a message at distance d is usually taken to be d^α, where α is an attenuation factor depending on the environment with typical values between 2 and 6. In this paper, we take $\alpha = 2$ but the results are similar for other values of α. When a slice S_i sends a message to the next slice S_{i-1}, we say that the message is *slid* from S_i, and when the message is sent directly to the sink, we say it is *ejected* from S_i. In our model, the amount of energy consumed by S_i to slide a message is arbitrarily chosen

Table 1
Summary of symbols

Symbol	Units	Interpretation
f_i	(messages/s)	Sliding rate from S_i to S_{i-1}
j_i	(messages/s)	Ejection rate from S_i to the sink
p_i		Sliding probability
g_i	(messages/s)	Rate of events detection for slice S_i
d_i^2	(J/messages)	Cost for ejecting a message from S_i
b_i	(J)	Energy initially available in slice S_i
P_i	(J/s)	Power of slice S_i

to be 1 J per message (J/message), and the energy required to eject a message is d_i^2 (J/message), where d_i is a constant depending on the network topology, proportional to the distance between S_i and the sink. Each time a slice detects an event, it needs to report to the sink by sending a message. g_i is the *rate of events detection* in slice S_i, the unit being messages per second (message/s). b_i is the total amount of energy available in slice S_i, the unit being Joules (J). Assuming that each sensor has the same amount of energy available, b_i is proportional to the number of sensors in S_i. f_i is the *rate of messages sliding* from S_i to S_{i-1}, the unit being (message/s). j_i is the *rate of messages ejected* from S_i to the sink, the unit being (message/s). P_i is the *power* of slice S_i, it is the expected energy consumption per slice, defined in J/s or Watts (W). P_i satisfies the following equation: $P_i := f_i + j_i d_i^2$ (J/s). It was shown in [ENR04, ENR06] that introducing a cost for receiving messages does not change results and is thus not relevant for energy balancing. However, it increases the complexity of equations and diminishes clarity, we therefore follow the common convention of not including in our model the cost of receiving a message. This is also justified in a setting where the energy costs of waiting for a message, or listening, and receiving a message are close. Notice that the following recurrence relation holds:

$$f_i + j_i \text{ (message/s)} = f_{i+1} + g_i \text{ (message/s)}.$$

These are flow equations, which account for the fact that messages are propagated along the network slices, where each slice is a source with input rate g_i. We define p_i to be the *sliding probability*, with $p_i = f_i/(f_i + j_i)$.

Definition. We say that the network is energy-balanced if the power to battery ratio is a constant, i.e. if for $1 \leqslant i \leqslant N$ it holds that $P_i/b_i = P_1/b_1$, and the lifespan of the network is defined as $\min\{b_i/P_i\}_{1 \leqslant i \leqslant N}(s)$.

That is, the lifespan of the network is determined by the time until one of the slices depletes all its energy and the network is energy-balanced if the total energy available in every slice is consumed during the same time period, namely b_i/P_i (s).

Table 1 lists the symbols used in this article, together with their units and interpretation.

3.1. Preliminary result

Suppose S_i ejects a message directly to the sink with probability ε_i. The mean energy consumption per message handled in slice S_i is equal to

$$\left[(1-\varepsilon_i) + \varepsilon_i d_i^2\right] \text{ (J/message)}.$$

Hence, the mean number of messages which can be handled by slice S_i before it runs out of energy is given by

$$\frac{b_i}{(1-\varepsilon_i) + \varepsilon_i d_i^2} \text{ (message)}.$$

Among this total number of messages, a $(1-\varepsilon_i)$ fraction reaches slice S_{i-1} after having been slid from slice S_i (the rest being ejected directly to the sink). If the ε_i's are chosen to ensure that the following equality is satisfied for $i, i-1, \ldots, 1$:

$$(1-\varepsilon_i)\frac{b_i}{(1-\varepsilon_i) + \varepsilon_i d_i^2} \text{ (message)} = \frac{b_{i-1}}{(1-\varepsilon_{i-1}) + \varepsilon_{i-1} d_{i-1}^2} \text{ (message)} \quad (6)$$

and if we assume that messages are only generated in slice S_i, it results that the expected lifespan of slices $S_i, S_{i-1}, \ldots, S_1$ is the same. More formally

Proposition. *Suppose that messages are slid by S_i towards S_{i-1} with probability $1-\varepsilon_i$ and ejected with probability ε_i (for $1 \leqslant i \leqslant N$). Suppose that the rates of event detections $\{g_i\}_{i=k}^N$ satisfy $g_i = 0$ if $i \neq k$ and $g_k > 0$ for some $1 \leqslant k \leqslant N$ and that Eq.* (6) *is satisfied if $2 \leqslant i \leqslant k$. Then for every S_i with $1 \leqslant i \leqslant k$, the lifespan of S_i is a constant equal to*

$$\frac{b_k/g_k}{(1-\varepsilon_k) + \varepsilon_k d_k^2}(s).$$

Relation (6) can be rewritten in the following useful form:

$$\varepsilon_{i+1} = \frac{((b_{i+1})/b_i)\left(1 - \varepsilon_i + \varepsilon_i {d_i}^2\right) - 1}{((b_{i+1})/b_i)\left(1 - \varepsilon_i + \varepsilon_i {d_i}^2\right) + {d_{i+1}}^2 - 1}. \quad (7)$$

Setting $\varepsilon_1 = 1$ (which is natural since the first slice can only send messages directly to the sink), we can directly compute the ε_i's satisfying the recurrence relation for fixed b_i's and d_i's.

4. Computation of an optimal solution

We propose an algorithm to compute the sliding probabilities p_i which ensure that a probabilistic data propagation algorithm is optimal in the sense that it maximizes the lifespan of the network, as defined in Section 3. The *input* is a description of the network and a statistical description of the data to be propagated in the form of three sequences of the same lengths $\{b_i\}_{1 \leqslant i \leqslant N}$, $\{d_i^2\}_{1 \leqslant i \leqslant N}$ and $\{g_i\}_{1 \leqslant i \leqslant N}$ where the b_i's describe the energy available in each slice in Joules (J), the d_i^2's are the energy necessary to eject a message from slice S_i to the

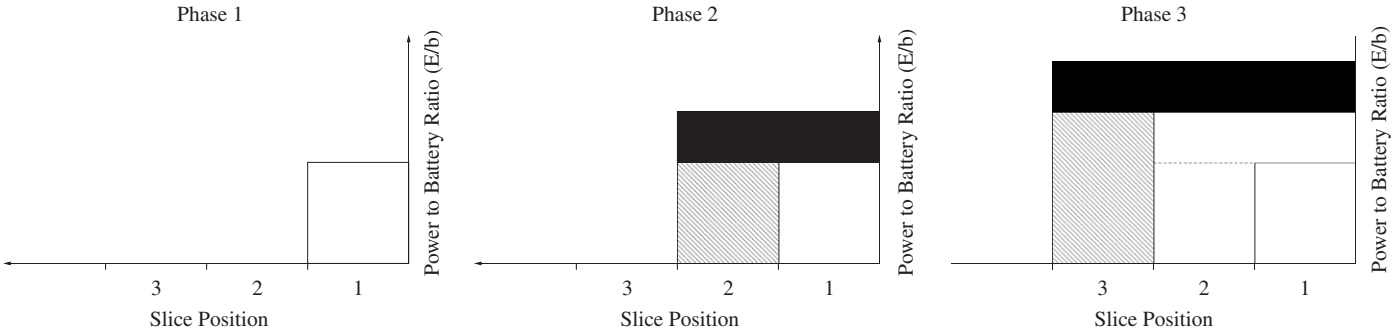

Fig. 3. Intuitive idea of the algorithm.

sink (J/message) and the g_i's are the distribution of events generating data to be propagated in the network (message/s), as in Section 3. The *output* is a sequence $\{p_i\}_{1 \leqslant i \leqslant N}$ representing the parameters of the probabilistic data propagation algorithm: in order to maximize the lifespan of the network, each slice S_i for $1 \leqslant i \leqslant N$ should send data directly to the sink with probability $1 - p_i$ and slide it to the next slice with probability p_i.

The heuristic for balancing the energy consumed among slices relies on the observation that the power of the first slice S_1 induced by considering only the rate of events g_1 generated in the first slice equals $P_1 = f_1 + j_1 d_1^2 = g_1$ (J/s), where the last equality holds because by convention, $d_1 = 1$, and because we only consider g_1. This is illustrated on the leftmost picture of Fig. 3. In turn, with the energy-balancing constraint of Eq. (3), this will determine the power consumption P_2 in slice S_2 when taking into consideration the rate of events g_2 generated in S_2. g_1 was already considered in the previous step, so a strategy to balance the energy consumed in the second slice S_2 with S_1 consists of first ejecting the right number of messages in order to ensure that the power to battery ratio P_2/b_2 (s^{-1}) in the second slice is equal to the power to battery ratio P_1/b_1 in the first slice. This is illustrated by the crossed-out rectangle of the middle picture of Fig. 3. Then, the remaining messages to be handled are slid to S_1 using the probability ε_2 computed in Proposition 3.1, hence equally increasing the power to battery ratio of slices S_1 and S_2 which thus remain balanced, as illustrated by the black rectangle in the middle picture of Fig. 3. The heuristic must be inductively applied considering slices $S_3, S_4, \ldots$ up to S_N, the case of S_3 is illustrated on the rightmost picture of Fig. 3. Since the b_i's and d_i's are given from the input for $1 \leqslant i \leqslant N$, we can compute the ε_i's ($1 \leqslant i \leqslant N$), the solution to relation (6) with $\varepsilon_1 = 1$.

Remark. Notice that although Eq. (6) implies that $\varepsilon_i \leqslant 1$, nothing guarantees that $\varepsilon_i \geqslant 0$. For simplicity, let us make the temporary assumption that $\varepsilon_i \geqslant 0$. We treat the case with negative ε's in Section 4.1.2.

During its execution, our algorithm needs the following *variables*, for $1 \leqslant i \leqslant N$:

- G_i is the rate of messages to be treated at slice S_i (message/s).
- F_i is the rate of messages forwarded from slice S_i towards slice S_{i-1} (message/s).
- J_i is the rate of messages ejected from slice S_i directly to the sink (message/s).
- P_i is the power consumed by slice S_i, which is equal to $F_i + J_i {d_i}^2$ (J/s).

Using the g_i's from the input we initialize $F_i = J_i = 0$ and $G_i = g_i$ for every $1 \leqslant i \leqslant N$. The algorithm then treats each slice one at a time from S_1 towards S_N. First, S_1 is treated according to the heuristic described at the beginning of this section: we let $J_1 = G_1$ to account for the fact that all the messages generated at S_1 are ejected to the sink. This means an average power consumption in slice S_1 of $P_1 = G_1 d_1^2 = J_1 d_1^2$. Since all messages have been ejected, there are no more messages to treat for S_1 and we update the value of G_1 to $G_1 = 0$.

We then repeat the following for each of the slices S_i for i from 2 to N: first, let $J_i := (b_i / b_{i-1} {d_i}^2) P_{i-1}$, which is equivalent to

$$\frac{b_i}{J_i {d_i}^2}(s) = \frac{b_{i-1}}{P_{i-1}}(s). \tag{8}$$

This ensures that the time needed to exhaust the available energy b_{i-1} in slice S_{i-1} equals the time needed to exhaust the energy b_i in slice S_i while considering only ejection, which is represented by the crossed-out regions in Fig. 3. Notice that $J_i > G_i$ means that we are trying to eject more messages than the total amount of messages available to be treated, which is not physically possible. Therefore, our approach requires the presence of a sufficiently number of messages to be treated at slice S_i, i.e. it should be that the following holds:

$$J_i = \frac{b_i}{b_{i-1} {d_i}^2} P_{i-1} \leqslant G_i. \tag{9}$$

We overcome this limitation in Section 4.1.1. For the time being, we consider only the case where the initial G_i's are large enough to ensure that Eq. (9) holds. So far energy balance is achieved for slices S_i to S_1 (because of Eq. (8) and by induction), which is represented by the crossed-out rectangles of Fig. 3. Since J_i messages have been ejected, we update G_i to $G_i := G_i - J_i$. The remaining G_i messages to be treated will be handled in such a way that they will increase the power to

battery ratio in each of the slices S_i to S_1 by exactly the same amount, as illustrated by the black rectangles of Fig. 3. The strategy is the following: a fraction $(1-\varepsilon_i)$ of the G_i messages yet to be treated is forwarded to the next slice S_{i-1}, while the rest is ejected directly to the sink, thus increasing the power of slice S_i. Formally, this means setting the following:

$$F_i := (1-\varepsilon_i)\, G_i, \tag{10}$$

$$J_i := J_i + \varepsilon_i G_i, \tag{11}$$

$$G_i := 0. \tag{12}$$

Among the $G_i\,(1-\varepsilon_i)$ messages slid from S_i towards S_{i-1}, a fraction $1-\varepsilon_{i-1}$ will be further slid from S_{i-1} towards S_{i-2}, while the rest is ejected directly to the sink from S_{i-1}, thus increasing the power of S_{i-1}. This process goes down to the first slice, and the numbers of slid and ejected messages have to be updated. The algorithm implementing this idea needs to do the following:

$$F_{i-1} := F_{i-1} + (1-\varepsilon_{i-1})\,(1-\varepsilon_i)\, G_i, \tag{13}$$

$$J_{i-1} := J_{i-1} + \varepsilon_{i-1}\,(1-\varepsilon_i)\, G_i, \tag{14}$$

$$F_{i-2} := F_{i-2} + (1-\varepsilon_{i-2})\,(1-\varepsilon_{i-1})\,(1-\varepsilon_i)\, G_i, \tag{15}$$

$$J_{i-2} := J_{i-2} + \varepsilon_{i-2}\,(1-\varepsilon_{i-1})\,(1-\varepsilon_i)\, G_i, \tag{16}$$

$$F_{i-3} := F_{i-3} + (1-\varepsilon_{i-3})\,(1-\varepsilon_{i-2})\,(1-\varepsilon_{i-1}) \times (1-\varepsilon_i)\, G_i, \tag{17}$$

$$J_{i-3} := J_{i-3} + \varepsilon_{i-3}\,(1-\varepsilon_{i-2})\,(1-\varepsilon_{i-1})\,(1-\varepsilon_i)\, G_i, \tag{18}$$

$$\vdots \tag{19}$$

The messages handled increase the power of slice S_i by an amount equal to

$$m_i := G_i \left[(1-\varepsilon_i) + \varepsilon_i d_i^2\right] (\text{J/s})$$

while the increase in power for slice S_{i-1} equals

$$m_{i-1} := G_i\,(1-\varepsilon_i)\left[\left(1-\varepsilon_{i-1}+\varepsilon_{i-1}d_{i-1}^2\right)\right] (\text{J/s})$$

and so forth for slices S_{i-2}, S_{i-3}, up to S_1. But because the ε_i's satisfy Eq. (7), the $(b_j/m_j)(s)$'s $(1 \leqslant j \leqslant i)$ have the same value, as follows from Proposition 3.1. So when we finish treating slice S_i the average time before running out of energy in $S_1, S_2, \ldots, S_i$ is equal. Again, this is illustrated by the black rectangles of Fig. 3. We then go on to treat the next slice (S_{i+1}) until we reach the last slice, S_N.

At this point and for each $1 \leqslant i \leqslant N$ slice S_i treats a total of $F_i + J_i$ (*messages/s*), of which F_i are slid and J_i are ejected, and the network is energy-balanced. The output of the algorithm representing the optimal parameters for the probabilistic data propagation algorithm is the following ordered sequence: $\{F_i/(F_i+J_i)\}_{1 \leqslant i \leqslant N}$.

4.1. Special cases

In this section, we lift the assumption that all ε's are positive (Remark 4), and the assumption that Eq. (9) holds, starting with the former.

4.1.1. First case: too many sensors or too few messages

Suppose that while executing the algorithm from the previous section, Eq. (9) does not hold for some i, meaning that while treating slice S_i, even if all the g_i generated messages are ejected to the sink, the power to battery ratio P_i/b_i for S_i will not be as high as for S_{i-1}. In Fig. 3, this means that the crossed-out rectangle is not as high as the white rectangle on its right.

In essence, the solution is to get slices previous to S_i (i.e. S_{i+1}, S_{i+2}, etc.) to forward some of their generated messages towards S_i, so that S_i can "catch-up" with the power to battery ratio of S_{i-1}. To do so, we recursively use the algorithm described earlier in the following way: since the problem is that S_i has not got enough messages to eject, we recompute new values of ε's satisfying Eq. (7), only this time we force $\varepsilon_i = 1$, which means that slice S_i will now eject every message which is slid from S_{i+1}. This will eventually enable S_i to catch-up with S_{i-1}, if sufficient messages are slid from S_{i+1}. If the power to battery ratio P_i/b_i of S_i catches up with the power to battery ratio of S_{i-1}, we can lift the constraint of having $\varepsilon_i = 1$ and go on with the previous values of ε's.

When S_i needs to catch-up with S_{i-1} and we need to force $\varepsilon_i = 1$, we say the algorithm *goes down* one-level in the recursion. This brings the need to stack some values, which we will be able to unstack when *coming up* one level in the recursion, and to compute some new values. The following list explains how and what to stack.

- Stack the current value of a variable *start*, which remembers at what position the last recursion started. (If this is the first level of recursion, we stack the value $start = 1$). The new value of *start* is set to $start = i$, which is the position of the slice which is trying to catch up.
- Stack the current value of a variable *max*. If this is the first level of recursion, we stack the value $max = \infty$. The new value of *max* is set to

$$max = \frac{1}{b_{\text{start}-1}}\left(F_{\text{start}-1} + J_{\text{start}-1}d_{\text{start}-1}^2\right)\ (s^{-1})$$

  which is the power to battery ratio increase needed for S_{start} to "catch up" with $S_{\text{start}-1}$.
- Stack the value of the previous ε's. Set new current values for ε_j's for $start \leqslant j \leqslant N$:
  - $\varepsilon_{\text{start}}$ is set to 1, so that slice S_{start} ejects every sliding messages it sees, in order to try to catch up with $S_{\text{start}-1}$.
  - The other ε_k's with $start < k \leqslant N$ are computed using Eq. (7).

Once this is done, we can go down one level in the recursion, which essentially means redoing the algorithm from Section 4, but using the above newly computed ε's, and considering the

possibility of either going further down one recursion level, or on the contrary coming back up one recursion level. That is, once the algorithm has gone down one level of recursion and while treating slices $S_{\text{start}+1}$, $S_{\text{start}+2}$, and so on, we have to take into account that three different possible cases may occur (for the sake of comprehensiveness, suppose we are treating slice S_k for some $k > 0$):

- S_k may in turn not be able to increase its power to battery ratio P_k/b_k to the level of S_{k-1}, in which case we need to go down one further recursion level.
- Slice S_{start} may be able to catch up (using messages previously slid from slices $S_{\text{start}+1}$ to S_{k-1} and the messages newly slid from S_k), i.e. sufficient messages will have been slid from the previous slices. In this case, we shall have to go back up one recursion level.
- Finally, it may be that neither of the two above cases happens: slices $S_k, S_{k-1}, \ldots, S_{\text{start}}$ have so far the same power to battery ratio (but less than $S_{\text{start}-1}$), and we start treating the next slice, S_{k+1}.

Keeping in mind these three possible cases, a description of how the algorithm of Section 4 should be adapted follows, assuming it has gone down some levels of recursion and slice S_k is being treated for some $k > start$ while slice S_{start} is trying to catch up with slice $S_{\text{start}-1}$.

1. *If* there are not enough messages for S_k to increase its power to battery ratio to the level of S_{k-1} and hence, even if all the G_k messages are ejected from slice S_k, the lifespan of S_{k-1} remains smaller than the lifespan of S_k, i.e. if $b_k/G_k d_k^2\,(s) > b_{k-1}/(F_{k-1} + J_{k-1}d_{k-1}^2)\,(s)$ we just try to diminish this unbalanced lifespan as much as possible by ejecting all the G_k messages, setting $J_K = G_K$, $G_k = 0$ and going further down one recursion level. Now S_k will have to try to catch up with S_{k-1}.
2. *Else*, we eject $J_k = b_k/(d_k^2 b_{k-1})\left(F_{k-1} + J_{k-1}d_{k-1}^2\right)$ messages from S_k and set $G_k = G_k - J_k$. S_k now has the same lifespan as S_{k-1}, but there are still G_k messages to be treated. Essentially, we now want to slide the remaining G_k messages *along the network*, from S_k to S_{start}, but with some precaution:
   - First of all, we still have to take into the account the ε's and eject "ε fractions" of the messages sliding along the network from S_k to S_{start} as was explained in the previous section. This was explained in detail in Eqs. (10)–(18).
   - If we have enough remaining messages, i.e. if G_k is still sufficiently large, we will be able to let slice S_{start} (and slices S_k to $S_{\text{start}+1}$) catch up with $S_{\text{start}-1}$. If this is so, we want to slide *just enough* messages to catch up, then go back up one recursion level and unstack the previous ε's. The remaining messages will then be slid along the network, this time using the ε's that have just been unstacked.

   Here is how we propose to implement the above remark.
3. Set $\Delta_t = (max - (1/b_k)\left(F_k + J_k d_k^2\right))^{-1}$ (s) and $msgTo\,GoUp = b_k/\left(\Delta_t\left(\varepsilon_k d_k^2 + (1-\varepsilon_k)\right)\right)$. Let $\Phi = \max\{G_k, msgToGoUp\}$. From the remaining G_k messages, we further slide and eject, respectively, $F = (1-\varepsilon_k)\cdot\Phi$ and $J = \varepsilon_k \cdot \Phi$ messages. This ensures that slices S_k to S_{start} still have the same lifespan and that this is bounded from under by the lifespan of slice $S_{\text{start}-1}$. We thus need to make the following adjustments: $F_k = F_k + F$, $J_k = J_{\text{start}} + J$ and $G_k = G_k - F - J$. The F_j's and J_j's for $start \leqslant j < k$ also have to be adjusted, acknowledging the fact that F new messages are slid along the network from S_k to S_{k-1}, and using as usual the ε's to compute the ratio which is slid and ejected by each slice.
4. *If* there are enough messages for S_{start} to catch up with $S_{\text{start}-1}$, i.e. if Φ is equal to $MsgToGoUp$, we can go back up one recursion level, which means unstacking the previous ε's, unstacking the previous value of *max* and unstacking the previous value of *start*. Otherwise, we do not unstack any variables, and keep them as they are. Finally, *if there are no more messages to treat* for slice S_k, i.e. if $G_k = 0$, we can start to treat the next slice, S_{k+1}. This means jumping to point 1 above, but this time with $k = k + 1$, which also means we can stop the algorithm if $k + 1 > N$. Otherwise, *we need to treat the remaining G_k messages*. This is done by jumping to point 3 above.

In the end, if the algorithm returns from all the recursive calls to the main algorithm, it is easily seen that energy balance is reached. Otherwise, we have a solution with increasing lifespan (from slice S_1 towards S_N), and which is "locally" energy-balanced, for example, we could have

$$\frac{b_1}{P_1} = \frac{b_2}{P_2} = \frac{b_3}{P_3} < \frac{b_4}{P_4} < \frac{b_5}{P_5} = \frac{b_6}{P_6} \leqslant \cdots .$$

Although not reaching energy balance, we shall prove in Section 5 that this solution is optimal in the sense that it maximizes the lifespan. An important thing to observe is that if a recursion starts at slice S_i, either one of the two cases happens:

- The algorithm returns from this recursive call and the solution is locally energy-balanced: $P_i/b_i = P_{i-1}/b_{i-1}$.
- The algorithm does not return from this recursive call and the solution is *not* energy-balanced: $P_i/b_i < P_{i-1}/b_{i-1}$. Furthermore, since S_i was trying to "catch up" with S_{i-1} and since we set $\varepsilon_i = 1$ (point (4.1.1) of the algorithm), it holds that $F_i = 0$, and thus that

$$p_i = 0. \tag{20}$$

In Section 5, we use Eq. (20) to show that this solution is always optimal.

4.1.2. Second case: too few sensors or not enough battery

The second problem which may occur is when the assumption that all ε's are positive, (i.e. the assumption from the remark of p. 11) does not hold. From Eq. (7), we can see that this occurs only if some of the slices have little b_i's (thus the title of this subsection, since b_i's are proportional to the amount of sensors). Let us first analyze what it means for an ε, say ε_i, to be *negative*. Suppose slice i has, so far, j_i ejected messages

(per second) and f_i forwarded messages (per second). When it receives k sliding message from S_{i+1}, it should eject an ε_i fraction to the sink and pass on the $1-\varepsilon_i$ rest to the next slice. After this, there are $j_i + k\varepsilon_i$ ejected messages and $f_i + k(1-\varepsilon_i)$ slid messages. The fact that the ε's satisfy Eq. (7) ensures that energy balance is conserved (at least locally if we are already into a recursive call as described in Section 4.1.1). A difficulty follows from the fact that since ε_i is negative, j_i becomes negative if

$$k \cdot \varepsilon_i + j_i < 0 \quad \Leftrightarrow \quad k > j_i/\varepsilon_i \tag{21}$$

and thus the solution is not physical (a negative amount of messages cannot be ejected from S_i). The solution to this problem has some similarity with the previous one. Whenever a slice (say the ith) is about to slide k messages along the network, it should ensure that no slice will find itself in a non-physical state afterward by bounding the number of messages it allows itself to slide along the network. Suppose that, for some fixed k, a slice S_k wants to slide messages along the network. We call *maxSlide* the maximum number of messages S_k may slide along the network without putting any of its following slices in a non-physical state. In order to compute *maxSlide*, we should remember what happens when k messages are slid along the network by slice S_l: some (or part) of them are ejected by each of the slices sliding the message, according to the ε's, and therefore only a $k_i = k \cdot \Pi_{j=i+1}^{l}(1-\varepsilon_i)$ fraction of the k initial messages reaches slice S_i. *maxSlide* is defined as the maximum value k such that $k_i\varepsilon_i + j_i \geqslant 0$ for $1 \leqslant i \leqslant l$, or equivalently, the maximum value such that $k_i \leqslant j_i/|\varepsilon_i|$ for every $1 \leqslant i \leqslant l$ such that $\varepsilon_i < 0$, and it can be computed by the following algorithm.

Algorithm 4.1. COMPUTEMAXSLIDE(i).

Input: i, a slice number.
Output: *max*, the max number of messages S_i can slide.
global ε[ ] **comment:** ε[ ] is an array storing the values of the ε's
local $F \leftarrow 1$
local *ejected*[ ]
for $(k \leftarrow i; k \geqslant 1; k \leftarrow k-1)$
do { *ejected*$[k] \leftarrow F * \varepsilon[k]$; $F \leftarrow F * (1-\varepsilon[k])$
local $max \leftarrow \infty$
for $(k \leftarrow i;\ k \geqslant 1;\ k \leftarrow k-1)$
do {
  local $j \leftarrow$ *ejected*$[k]$
  if $j < 0$
    then
      do {
        $j \leftarrow -j$
        if $max = \infty$
          then $max \leftarrow j[k]/j$
          else
            do {
              local $tmp\_max \leftarrow j[k]/j$
              if $tmp\_max < max$
                then $max \leftarrow tmp\_max$
            }
      }
}
return (*max*)

Once we have computed *maxSlide*, we can decide what to do when slice S_i wants to slide k messages. If $k \leqslant maxSlide$, then we can simply slide the k messages, but if $k > maxSlide$, we have to be more careful. First, we can partially fulfill the aspiration of S_i by allowing it to slide *maxSlide* messages. At this stage, S_i still wants to slide $k - maxSlide$ messages and one of the previous following slices (say S_k) has a negative ε_k and $j_k = 0$. If any more messages are slid, S_k will be in a "non-physical" state. So what we do is that we recompute all ε_k's for $1 \leqslant k \leqslant j$. Notice that if we are inside a recursion of the type described in Section 4.1.1, we do not recompute all ε_k's for $1 \leqslant k \leqslant N$, but we rather set $\varepsilon_{\text{start}} = 1$ and recompute all the ε_i's for $start < i \leqslant N$, where *start* is the place where the last recursion took place (cf. the first buller of the first enumeration in Section 4.1.1). While recomputing the ε's, whenever $\varepsilon_k < 0$ and $j_k = 0$, we force ε_k to 0. What this does is, first of all, to force the solution to be physical. Second, it breaks the relation from Eq. (7), since for some k's ε_k is forced to 0. The fact of breaking this relation prevents slices from ejecting a *negative* amount of messages (and thus in a sense save some energy), when this would lead them to be in a non-physical state. Thus slice S_k will spend more energy than the (locally) energy-balanced solution would require, and on the other hand, slices following S_i (that is S_{i-1} to S_1) will spend less since the negative amount of messages which have been prevented from being ejected were supposed to be slid along the network. We are therefore confronted with a "local peak", in the sense that

$$\frac{P_k}{b_k} > \frac{P_{k-1}}{b_{k-1}}.$$

It should be observed that for the rest, energy balance is conserved (at least locally), and furthermore whenever such a "peak" appears at S_k, it holds that

$$p_k = 1 \tag{22}$$

which is an important fact we shall use to prove that the solution obtained is optimal.

4.2. Time complexity

On input $(\{b_i\}_{i=1}^N, \{d_i\}_{i=1}^N, \{g_i\}_{i=1}^N)$, the algorithm presented in the previous section returns a list $\{p_i\}_{i=1}^N$ of values. We explain at a high level of abstraction why the runtime of the algorithm has a worst-case complexity of $\mathcal{O}(N^3)$, and omit the tedious details. To see this, first observe that the algorithm runs in a top-down fashion from slice S_1 to slice S_N, so there are N slices to be treated. Each time a slice is treated, e.g. when slice S_k is treated for some $1 \leqslant k \leqslant N$, new values of J_i and F_i have to be computed for $1 \leqslant i \leqslant k$, as follows from Eqs. (10) to (18). This already implies a $\mathcal{O}(N^2)$ complexity. The reason why the final complexity is not $\mathcal{O}(N^2)$ but rather $\mathcal{O}(N^3)$ is a bit more subtle. What happens is that for each slice, say for slice S_k, the values of J_i and F_i may, in the worst case, have to be updated up to k times, for some $k \leqslant N$. The reason why the F_i and J_i

may have to be updated more than once is that each of the slices S_1 to S_{k-1} may force the algorithm to update the F_i's and J_i's twice, by limiting the number of messages to be slid from S_k to the sink on the first occasion. This may happen for two distinct reasons: either because one of these slices was "catching up" and the algorithm "comes back up one level of recursion", as explained in Section 4.1.1, thus forcing the update of the J_i's and F_i's to happen twice. Alternatively this may happen because one of the slices has a negative ε value, as explained in Section 4.1.2, and it limits the maximum number of messages allowed to be sent during the first step to *maxSlide*, the value which is computed by Algorithm 4.1, the rest being sent in a second step. In the worst case, slice S_k thus has to update the values of $\{F_i\}_{i=1}^{k}$ and $\{J_i\}_{i=1}^{k}$ for every slice from 1 to $k-1$. When $k=N$, this means a total of $(N-1)$ updates of $\{J_i\}_{i=1}^{N}$ and $\{F_i\}_{i=1}^{N}$ (i.e. $\mathcal{O}(N^2)$ updates). This brings the worst-case runtime complexity of the algorithm to $\mathcal{O}(N^3)$ since there are N slices. It may be observed that when neither of the two special cases from Section 4.1 occurs, i.e. when Eq. (9) holds or if condition (21) is never satisfied, then the algorithm runs in $\mathcal{O}(N^2)$.

5. Proof of optimality

In this section, we prove that our algorithm produces an optimal solution, in the sense that it maximizes the *lifespan*, cf. Definition 3.

Convention. *In this section, we consider a fixed sensor network of size N, with fixed event distribution* $\{g_i\}_{1\leqslant i\leqslant N}$ *and fixed battery levels* $\{b_i\}_{1\leqslant i\leqslant N}$. *A configuration C of the network is the choice of a sliding probability assignment* $\{p_i\}_{1\leqslant i\leqslant N}$ *for each slice. If C and* $\tilde{C}$ *are two configurations, we use the letters* f_i *and* $\tilde{f}_i$ *to denote the slid messages under configurations C and* $\tilde{C}$, *respectively. We do the same for the other parameters*: j_i'*s*, p_i'*s*, ε_i'*s and* P_i'*s*.

Lemma (*No win–win modification*). *No configuration is strictly better in terms of lifespan than another configuration*: *if C and* $\tilde{C}$ *are two configurations, then there exists an i such that*:

$$\frac{P_i}{b_i}\geqslant\frac{\tilde{P}_i}{b_i}.$$

Proof. Suppose (absurd) this is not true. Therefore, there exist two configurations C and $\tilde{C}$ such that

$$\forall i\ \frac{P_i}{b_i}\leqslant\frac{\tilde{P}_i}{b_i} \tag{23}$$

and for at least one of the i's

$$\frac{P_i}{b_i}<\frac{\tilde{P}_i}{b_i}. \tag{24}$$

We now define the following configurations $C_0=\tilde{C}$ and $C_N=C$, and more generally, C_i is the configuration where the i last p_i's (i.e. p_N, p_{N-1}, $\ldots p_{N-i+1}$) are the same as the p_i's from C, whereas the $N-i$ first p_i's are the same as the p_i's from $\tilde{C}$. Then for each $1\leqslant i\leqslant N$, if we use ${}^iE$ and ${}^ib$ to designate the power and battery of configuration C_i, the following holds:

$$\frac{{}^iP_k}{{}^ib_k}=\frac{P_k}{b_k}\quad\forall N\geqslant k>N-i,$$

$$\frac{{}^iP_i}{{}^ib_i}\geqslant\frac{P_i}{b_i}, \tag{25}$$

where the first equation follows from the definition of C_i and the second follows from the easy observation that ${}^iP_i/{}^ib_i\geqslant\tilde{P}_i/\tilde{b}_i$ combined with Eq. (23). Next, let $k=\max\{i|P_i/b_i<\tilde{P}_i/\tilde{b}_i\}_{1\leqslant i\leqslant N}$, which exists by (24). Then for every $i\leqslant k$ the inequality in (25) becomes strict. In particular, for $i=0$ (and using the fact that $C_0=C$) it becomes $P_0/b_0>P_0/b_0$, which is the contradiction we need. □

The reason we give this lemma the name of *no win–win modification lemma* is that a principle can be derived from it, the *no win–win modification principle*, which is the following: *if a configuration is modified to increase the lifespan in some parts of the network, then necessarily the lifespan is decreased in another part of the network*.

In [ENR04, ENR06], the authors point out that looking at the numerical solutions, one observes that an energy-balanced solution mostly uses single-hop data propagation, and only with little probability propagates data directly to the sink. The authors then suggest that this is an important finding implying that the energy-balanced solution is also energy efficient, since it only rarely uses the costly single-hop direct ejection of messages to the sink. Our previous lemma enables us to easily formalize this intuition:

Corollary. *Any energy-balanced solution is optimal in terms of lifespan*: *if C is an energy-balanced configuration* (*i.e.* $\forall i\ P_i/b_i=P_{i+1}/b_{i+1}$), *then for every other configuration* $\tilde{C}$, *we have the following inequality, with equality if and only if* $C=\tilde{C}$: $\min\{b_i/P_i\}\geqslant\min\{\tilde{b}_i/\tilde{P}_i\}$, *that is, C maximizes the lifespan among all possible configurations*.

Next we generalize the previous corollary.

Lemma. *Let C be a configuration of our network. Let* $max=\max\{P_i/b_i\}$. *Let* $k=max\{i|P_i/b_i=max\}$, *and let* $l=min\{i|P_i/b_i<max\}$, *if such an l exists. The configuration is optimal if and only if the conjunction of the following holds*:

- ($k<N$ *and* $p_{k+1}=0$) *or* $k=N$;
- ($l<N$ *and* $p_{l+1}=1$) *or* $l=N$ *or l was not well defined*.

Proof. We only give the main ideas of the proof. First, notice that slices S_k to S_l form a tabletop-like maximum of the plotting of slice position against power to battery ratio (cf. Fig. 3). Since $p_{k+1} = 0$, nothing can be done on the left-hand side of the tabletop to lower it. Second, since $p_{l+1} = 1$, the tabletop cannot rely on the slices on its right to take on a larger part of the message sliding towards the sink. The only solution to produce a better solution than C (i.e. if C was not an optimal solution) would therefore be to modify the probabilities from p_k to p_{l+1}, i.e. to reorganize the configuration "inside the tabletop". The final point is to notice that this will break the energy balance of the tabletop, increasing the maximum, using the no win–win modifications lemma of p. 19. □

Theorem. *Our algorithm always produces an optimal solution.*

Proof. The demonstration would be complete if we could prove that our algorithm always produces a solution where the maximum power to battery ratio is reached at a tabletop with $p_i = 0$ on the left and $p_i = 1$ on the right, since this enables to use Lemma 5. To see that this is the case, the main ingredients are Eqs. (20) and (22). We leave the easy details to the reader. □

6. Simulations

In this section, we present numerical validation of our algorithm. The approach consists of randomly and uniformly scattering sensor nodes in a sector of the plane, with a sink placed at the centre of the sector, cf. Fig. 1. We consider the *unit disc graph* constructed upon the sensor nodes: we place an edge between two sensor nodes or the sink if and only if they are at distance at most 1 from one another. The shortest path from a node to the sink determines in which slice it is. For example, if the shortest path from a node to the sink is a 3-hop path, the node is considered to be in the third slice S_3. Let N be the maximum slice number. For each $1 \leqslant i \leqslant N$, b_i is the number of sensor nodes in S_i. Assuming a uniform distribution of events, the expected number of events in S_i is equal to the expected number of sensors in S_i, and thus we set $g_i = b_i$. Finally, in a pessimistic approximation, we let $d_i = i$. Using these b_i, g_i and d_i as an input to the algorithm described in Section 4, we compute p_i the sliding probabilities which are predicted to balance the energy between slices. The simulations we present show that this is indeed the case.

6.1. First simulation

We divide the time into rounds, and during each round we let a randomly and uniformly chosen sensor node detect an event. The sensor node which has detected an event adds a message to be sent to its message queue. Also, during each round, each sensor node which has a non-empty message queue sends one message according to the following strategy: suppose that a sensor node n which is in the ith slice S_i needs to send a message, it sends the message directly to the sink (the message is ejected) with probability $1 - p_i$, thus spending i^2 (J), and with probability p_i it slides the message to one of its neighbours in the unit disk graph, thus spending 1 (J). In the case where the message is slid, the receiving node is chosen among all neighbours of n which are in the slice S_{i-1}. More precisely, let R be the set of neighbours of n which are in S_{i-1}, which is the next slice towards the sink. n sends the message to the node of R which has the highest remaining energy. If this node is not unique, a random decision is made. The implicit assumption that sensor nodes are aware of the remaining battery level of their neighbours can be implemented in a real WSN by adding information on the remaining battery level of emitting nodes in a small header to the messages.

This routing protocol is inspired by the *gradient based routing* (GBR) family of routing algorithms, see [SS01, HKK04, YZLZ05]. GBR was introduced in [SS01] and is inspired by [IGE00]. In our simulations, the gradient is determined by the slice number and the remaining battery level: a node is lower than another when its slice number is smaller, and when the slice number is the same, a node is lower than another if is has spent less energy.

Simulations show that, as expected, the average energy consumption in each slice is balanced. However, inside of each slice, the energy consumption is *not* well balanced. In particular, we observe that in each slice, the nodes which are further away from the sink spend more energy than the nodes close to the sink. We understand that this phenomenon happens because in slice S_i the nodes which are the furthest away from the sink have a lot of neighbours in S_{i+1} and just a few neighbours in S_{i-1} while the contrary happens for the nodes of S_i which are close to the sink. As a consequence, nodes on the "far from the sink" side of S_i receive more messages from S_{i+1} than nodes on the "close to the sink" side of S_i.

On the left-hand side of Fig. 4, we plot the radius of each of 6280 nodes scattered in a 20 m radius and 90° sector against the energy spent by each of these nodes while reporting events to the sink according to the strategy before mentioned for a total of 118 000 rounds. For readability, the nodes which belong to a slice S_i with i an odd number are in grey and black is used when i is even. For each slice, we also compute the average radius and the average energy spent, and plot this as the squares joined by a line. The figure shows that the mean energy consumption from sensors of the same slice is almost the same for every slice, but inside of each slice the energy consumption is not balanced among sensors.

6.2. Improvement with spreading techniques

The previous simulation shows that the probabilistic algorithm which makes slice S_i eject with probability $(1 - p_i)$ and slide with probability p_i balances energy well between slices. Notice that this simulation validates our theoretical investigations since they are only concerned with balancing energy among the slices. However, as previously pointed out, the energy consumption is not well balanced among the sensors of a fixed slice. This can be circumvented by using *spreading*

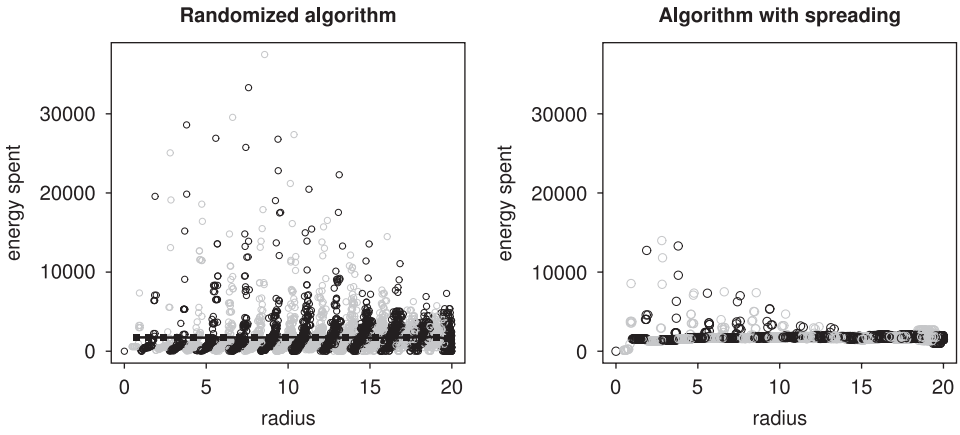

Fig. 4. Simulations.

techniques (cf. [SS01]) on messages: the messages need to be spread more evenly inside of each slice. We propose a simple spreading technique, which is the following. When a message is to be ejected by a node n of slice S_i, the node n does not eject the message straight away. Instead, the message is *marked for ejection*, and sent to a neighbour of m in the same slice as n, i.e. in slice S_i. More precisely, let S be the set of neighbours of n which are in the same slice as n. When n marks a message for ejection, it passes it to the node of S which has spent the least energy so far. A node which receives a message marked for ejection takes care of sending the message directly to the sink. This spreading technique does not change the amount of ejected messages in each slice, but it transfers the charge of ejecting messages to nodes which have more energy and are, as explained in Section 6.1, in each slice, the nodes which are close to the sink.

We show by simulation that this very simple spreading technique balances energy consumption not only between slices, as was the case in the previous simulation, but also between nodes of the same slice. Furthermore, the overhead due to the fact that messages are passed to a node in the same slice before being ejected is low. On the right-hand side of Fig. 4, we see that the use of the spreading technique induces a well balanced energy consumption not only among slices but also among nodes of the same slice. Furthermore, comparison with the left-hand side plot shows that the mean energy consumption while using the spreading technique is almost the same as the mean energy consumption of the previous experiment, without spreading. This means that the overhead introduced by the spreading technique has a minor impact. We conducted many similar experiments (changing the radius, the angle, the density and the distribution of generated events), and the simulation results are comparable to those presented in this section.

7. Conclusion

Previous to this work, data propagation algorithms have been proposed to balance the energy consumption evenly inside of a WSN using a combination of multi-hop short range transmissions and long-range single-hop transmissions, also called ejections. We have shown that ejections can be used to maximize the lifespan of a WSN even when an energy-balanced solution does not exist. The main idea is to divide the WSN into slices and to make sensor nodes eject messages according to a probability depending on the slice in which they are located. The probability, for each slice, is computed off-line by the algorithm described in Section 4. In the simulations of Section 6, we adapt the GBR family of data-propagation algorithms to use the ejection probabilities computed by our algorithm, and show that a simple spreading technique is required and sufficient to make the energy evenly spread not only among slices, but also among all sensors of the network. It would be possible to use other spreading techniques, which could be investigated in future work. Another interesting question would be to find necessary and sufficient conditions for the existence of an energy-balanced solution. Indeed, we show that when an energy-balanced solution exists it is optimal but we did not address the question of finding necessary and sufficient conditions for such a solution to exist. Another important issue would be to find a totally distributed version of our algorithm: one where the computation of the ejection probabilities is made at the sensor node level. The impact of collisions was not taken into account by our model and it would be interesting to study the impact of long-range transmissions on collisions. Finally, in our model, sensor nodes are allowed two sorts of transmissions: long-range ejections directly to the sink and short-range transmissions to a neighbour node. Future work could investigate the possibility of using "medium" range transmissions, for example from a sensor node to a neighbour which is more than one hop away.

Appendix A. Pseudo code of the Algorithm

We give below a pseudo-code of the algorithm presented in this paper. This pseudo code is rigorously based on a Perl implementation of the algorithm which was validated on various inputs.

Algorithm A.1. COMPUTEOPTIMAL(N; $g[\,]$; $b[\,]$; $d[\,]$).

```
global f[ ]; j[ ]; max = ∞; recLevel = 0; startPosition = 1; ε[ ]
comment: Following arrays used as stacks while changing recLevels
global startPositions[ ]; maxs[ ]; epsilons[ ]
main
global i = 0; initialG[ ] = g[ ];
for i ← 1 to N
  do { f[i] = j[i] = 0
ε[ ] ← EPSILONS(1)
while i < N
  do { i ← i + 1
       PUSH(recLevels[ ], recLevel)
       if i = 1
       comment: First step
         then
         do { j[i] ← g[i]
              g[i] ← 0
         else
       comment: From second step to the Nth
         do { local E1 ← f[i − 1] + j[i − 1] ∗ d[i − 1]^2
              local ideal_j ← AVGNRJ(i − 1) ∗ b[i]/d[i]^2
              if ideal_j > g[i]
              comment: Not enough messages to stay at this level
                then
                do { EJECT(g[i])
                     DOWNONELEVEL()
                else
              comment: Enough messages to stay at this level
                do { EJECT(ideal_j)
                     while (g[i] > 0)
                       do { local nrjDelta ← max − AVGNRJ(i)
                            local msgToGoUp ← nrjDelta / ((1/b[i]) ∗ (e[i] ∗ d[i]^2 + (1 − e[i])))
                            if recLevel = 0 or g[i] < msgToGoUp
                            comment: Slide the rest
                              then SLIDE(g[i])
                              else
                            comment: Slide enough to go up one level
                              do { SLIDE(msgToGoUp)
                                   UPONELEVEL()
```

Algorithm A.2. EJECTION AND SLIDING OF MESSAGES().

```
procedure EJECT(eject)
  j[i] ← eject
  g[i] ← g[i] − j[i]

procedure SLIDE(F)
  SLIDECAREFUL(F)

procedure SLIDECARELESS(F)
  g[i] ← g[i] − F
  for (k ← i; k ⩾ 1; k ← k − 1)
    do { f[k] ← f[k] + F ∗ (1 − e[k])
         j[k] ← j[k] + F ∗ e[k]
         F ← F ∗ (1 − e[k])

procedure SLIDECAREFUL(F)
  local maxCarelessSlide ← COMPUTEMAXSLIDE()
  if maxCarelessSlide = ∞ or F <= maxCarelessSlide
    then
    do {SLIDECARELESS(F)
    else
    do { SLIDECARELESS(maxCarelessSlide)
         F ← F − maxCarelessSlide
         ε[ ] ← EPSILONS(startPosition, "withCaution")
         SLIDE(F)
```

Algorithm A.3. GOING UP OR DOWN ONE LEVEL().

```
procedure UPONELEVEL()
  local max ← POP(maxs[ ])
  startPosition ← POP(startPositions[ ])
  local tmp[ ] ← POP(epsilons[ ])
  ε[ ] ← tmp[ ]
  recLevel ← recLevel − 1

procedure DOWNONELEVEL()
  PUSH(maxs[ ], max) comment: store old max
  max ← AVGNRJ(i − 1)
  PUSH(startPositions[ ], startPosition)
  startPosition ← i
  local tmpArray[ ] = ε[ ]
  PUSH(epsilons[ ], tmpArray[ ]) comment: store old epsilons
  e[ ] ← EPSILONS(i)
  recLevel ← recLevel + 1
```

Algorithm A.4. COMPUTATION OF THE EPSILONS().

```
procedure EPSILONS(first; option)
  local ε[ ]
  for k ← 1 to first
    do {ε[k] = 1
  for k ← first + 1 to N
    do { local A ← (d[k]^2 − 1)/b[k]
         local B ← (ε[k−1] ∗ (d[k−1]^2 − 1) + 1)/b[k−1]
         ε[k] ← (B − 1/b[k])/(A + B)
         if option = "withCaution"
           then
           do { if ε[k] ⩽ 0 and j[k] = 0 and k ⩽ i
                then
                do {ε[k] = 0
  return (epsilons[ ])
```

Algorithm A.5. AVERAGE ENERGY AND MAXIMUM NUMBER ().

```
of messages to slide
procedure AVGNRJ(pos)
  return ((1/b[pos])(f[pos] + j[pos] ∗ d[pos]^2))

procedure COMPUTEMAXSLIDE()
  F ← 1 comment: Simulated slide of one packet from the
        current pos ( i )
  local ejected[ ]
  for (k ← i; k ⩾ 1; k ← k − 1)
    do { ejected[k] ← F ∗ e[k]
         F ← F(1 − e[k])
  local max ← ∞
  for (k ← i; k ⩾ 1; k ← k − 1)
    do { local j ← ejected[k]
         if j < 0
           then
           do { j ← −j;
                if (max = ∞)
                  then
                  do max ← j[k]/j
                  else
                  do { local tmp_max ← j[k]/j
                       if tmp_max < max
                         then
                         do max = tmp_max
  return (max)
```

References

[AKK05] J.N. Al-Karaki, A.E. Kamal, A taxonomy of routing techniques in wireless sensor networks, in: M. Ilyas, I. Mahgoub (Eds.), Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems, CRC Press, Boca Raton, FL, 2005, pp. 6.1–6.24.

[AY05] K. Akkaya, M. Younis, A survey on routing protocols for wireless sensor networks, Ad Hoc Network J. 3 (3) (2005) 325–349.

[BCN05] A. Boukerche, I. Chatzigiannakis, S. Nikoletseas, Power-efficient data propagation protocols for wireless sensor networks, Simulation 81 (6) (2005) 399–411.

[BN04] A. Boukerche, S. Nikoletseas, Wireless communications systems and networks, in: Protocols for Data Propagation in Wireless Sensor Networks: A Survey, Kluwer Academic Publishers, Dordrecht, 2004, pp. 23–51.

[Bou05] A. Boukerche, Handbook of Algorithms for Wireless Networking and Mobile Computing, Chapman & Hall/CRC, London, Boca Raton, FL, 2005.

[CT00] J. Chang, L. Tassiulas, Energy conserving routing in wireless ad hoc networks, IEEE INFOCOM 1 (2000) 22–31.

[CNS02] I. Chatzigiannakis, S. Nikoletseas, P. Spirakis, Smart dust protocols for local detection and propagation, in: Second Workshop on Principles of Mobile Computing (POMC), ACM, ACM Press, 2002, pp. 9–16.

[CCZ05] Y. Chen, C.-N. Chuah, Q. Zhao, Sensor placement for maximizing lifetime per unit cost in wireless sensor networks, in: MILCOM, October 2005.

[ENR04] C. Efthymiou, S. Nikoletseas, J. Rolim, Energy balanced data propagation in wireless sensor networks, in: Best papers of the Fourth International Workshop on Algorithms for Wireless, Mobile, Ad Hoc and Sensor Networks, 2004.

[ENR06] C. Efthymiou, S. Nikoletseas, J. Rolim, Energy balanced data propagation in wireless sensor networks, Wireless Networks (WINET) J., 2006.

[GK05] A. Giridhar, P.R. Kumar, Maximizing the functional lifetime of sensor networks, in: The Fourth International Conference on Information Processing in Sensor Networks (IPSN), April 2005.

[GLW03] W. Guo, Z. Liu, G. Wu, An energy-balanced transmission scheme for sensor networks, in: Poster Session of the First International Conference on Embedded Networked Sensor Systems (SenSys), ACM, IEEE Computer Society Press, November 2003.

[HKK04] K.-H. Han, Y.-B. Ko, J.-H. Kim, A novel gradient approach for efficient data dissemination in wireless sensor networks, in: International Conference on Vehicular Technology Conference (VTC), IEEE, vol. 4, September 2004.

[HCB00] W.R. Heinzelman, A. Chandrakasan, H. Balakrishnan, Energy efficient communication protocol for wireless microsensor networks, in: 33rd Hawaii International Conference on System Sciences (HICSS), 2000.

[HCB02] W.B. Heinzelman, A.P. Chandrakasan, H. Balakrishnan, An application-specific protocol architecture for wireless microsensor networks, Transactions on Wireless Communications 1 (4) (2002).

[HGWC02] X. Hong, M. Gerla, H. Wang, L. Clare, Load balanced, energy-aware communications for Mars sensor networks, Aerospace Conference Proceedings, 2002, IEEE, vol. 3, 2002, pp. 3–1109.

[IGE00] C. Intanagowiwat, R. Govindan, D. Estrin, Directed diffusion: a scalable and robust communication paradigm for sensor networks, in: Sixth International Conference on Mobile Computing (MOBICOM), ACM/IEEE, New York, 2000.

[JLPR06] A. Jarry, P. Leone, O. Powell, J. Rolim, An optimal data propagation algorithm for maximizing the lifespan of sensor networks, in: Proceedings of Distributed Computing in Sensor Systems (DCOSS) 2006.

[LH05] J. Luo, J.-P. Hubaux, Joint mobility and routing for lifetime elongation in wireless sensor networks, in: 24th INFOCOM, 2005.

[LH06] J. Luo, J.-P. Hubaux, Mobility to improve the lifetime of wireless sensor networks: a theoretical framework, in: Workshops of the Second International Conference on Distributed Computing in Sensor Systems (DCOSS), 2006.

[LNR05] P. Leone, S. Nikoletseas, J. Rolim, An adaptive blind algorithm for energy balanced data propagation in wireless sensor networks, The First International Conference on Distributed Computing in Sensor Systems (DCOSS), Lecture Notes in Computer Science, vol. 3560, Springer, Berlin, June/July 2005.

[LSS05] L. Lin, N.B. Shroff, R. Srikant, Asymptotically optimal power-aware routing for multihop wireless networks with renewable energy sources, in: Proceedings of INFOCOM'05, 2005.

[LXG05] Z. Liu, D. Xiu, W. Guo, An energy-balanced model for data transmission in sensor networks, in: 62nd Semiannual Vehicular Technology Conference, September 2005.

[Luo06] J. Luo, Mobility in wireless networks: friend or foe—network design and control in the age of mobile computing, Ph.D. Thesis, School of Computer and Communications Science, EPFL, Switzerland, 2006.

[OS06] S. Olariu, I. Stojmenovic, Design guidelines for maximizing lifetime and avoiding energy holes in sensor networks with uniform distribution and uniform reporting, in: 25th Conference on Computer Communications (INFOCOM), IEEE Communications Society, IEEE Computer Society Press, Silver Spring, MD, April 2006.

[RAdS+00] J.M. Rabaey, M. Josie Ammer, J.L. da Silva, D. Patel, S. Roundy, Picoradio supports ad hoc ultra-low power wireless networking, Computer 33 (7) (2000) 42–48.

[SL05] M.J. Sailor, J.R. Link, "smart dust": nanostructures devices in a grain of sand, Chem. Commun. 11 (2005) 1375–1383.

[SS01] C. Schurgers, M.B. Srivastava, Energy efficient routing in wireless sensor networks, in: Military Communications Conference (MILCOM), October 2001, pp. 357–361.

[STGS02] C. Schurgers, V. Tsiatsis, S. Ganeriwal, M. Srivastava, Topology management for sensor networks: exploiting latency and density, in: Eighth International Conference on Mobile Computing (MOBICOM), ACM/IEEE, New York, 2002.

[SD05] M.L. Sichitiu, R. Dutta, Benefits of multiple battery levels for the lifetime of large wireless sensor networks, in: NETWORKING 2005: Fourth International IFIP-TC6 Networking Conference, Lecture Notes in Computer Science, Springer, Berlin/Heidelberg, May 2005, pp. 1440–1444.

[SP03] M. Singh, V. Prasanna, Energy-optimal and energy-balanced sorting in a single-hop wireless sensor network, in: First International Conference on Pervasive Computing and Communications (PERCOM), IEEE, 2003.

[WLLP01] B. Warneke, M. Last, B. Liebowitz, K.S.J. Pfister, Smart dust: communicating with a cubic-millimeter, Computer 34 (1) (2001) 44–51.

[YZLZ05] F. Ye, G. Zhong, S. Lu, L. Zhang, Gradient broadcast: a robust data delivery protocol for large scale sensor networks, Wireless Networks (WINET) 11 (3) (2005).

[YP03] Y. Yu, V.K. Prasanna, Energy-balanced task allocation for collaborative processing in networked embedded system, in: Conference on Language, Compilers, and Tools for Embedded Systems (LCTES), June 2003.

[ZH04] H. Zhang, J.C. Hou, On deriving the upper bound of α-lifetime for large sensor network, Technical Report, Department of Computer Science, University of Illinois at Urbana-Champaign, 2004, published in ACM Mobihoc 2004, ACM Transactions on Sensor Networks, 2005.

[ZH05] H. Zhang, J.C. Hou, Maximizing α-lifetime for wireless sensor networks, in: Third International Workshop on Measurement, Modelling, and Performance Analysis of Wireless Sensor Networks (SenMetrics 2005), July 2005.

Pierre Leone is Assistant Professor at the Department of Computer Science of the University of Geneva where he is involved in the European CRESCCO and AEOLUS European projects as well as the Cost 295 action (DYNAMO). He was a scientific researcher at the Engineering School of Geneva where he was involved in various research projects in the fields of collaborative optimization algorithms on distributed systems and hardware development. He received his Ph.D. degree in Mathematics from the University of Geneva and has a background of electrical engineer with orientation in Computer Science. He spent a post-doctoral year as a Visiting Lecturer of the Mathematics, Department of the Auckland University in New-Zealand.

Olivier Powell is a Swiss National Science Foundation research fellow at the Computer Science and Informatics Department of the University of Patras, Greece. He was previously a post-doctoral research associate at the TCSensor lab of the Computer Science Department of the University of Geneva. He received a Ph.D. degree in Computer Science from the University of Geneva in the field of Complexity Theory and a Masters Degree in Mathematics from the same University.

Jose Rolim is Full Professor at the Department of Computer Science of the University of Geneva where he heads the Theoretical Computer Science and Sensor Lab (TCSensor Lab). He received his Ph.D. degree in Computer Science from the University of California, Los Angeles, in the area of formal languages and complexity. He has published many articles in the areas of theoretical computer science, distributed systems, randomization and computational complexity and leads two major national projects in the areas of Power Aware Computing and Games and Complexity. He also participates as a partner in two European Projects in the areas of Dynamic Systems and Foundations of Sensor Networks. Prof. Rolim participates in the editorial board of several journals and conferences and he is the Steering Committee Chair and General Chair of the IEEE Distributed Computing Conference in Sensor Systems. He has been Program Committee Chair of major conferences such as ICALP, IPDPS, RANDOM and served as Program Committee Member of all major conferences in theoretical computer science.