Energy optimal data propagation in wireless sensor networks

Powell, Olivier; Leone, Pierre; Rolim, Jose

How to cite

POWELL, Olivier, LEONE, Pierre, ROLIM, Jose. Energy optimal data propagation in wireless sensor networks. In: Journal of parallel and distributed computing, 2007, vol. 67, n° 3, p. 302–317. doi: 10.1016/j.jpdc.2006.10.007

This publication URL: https://archive-ouverte.unige.ch//unige:32491
Publication DOI: 10.1016/j.jpdc.2006.10.007

1

Energy Optimal Data Propagation in Wireless Sensor Networks

Olivier Powell, Pierre Leone, José Rolim

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 centralised algorithm computing 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 centralised algorithm computing 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.

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 constraints 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

*This research was supported in part by Swiss SER Contract No. C05.0030

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 analyse 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 [ENR06]. It allows each sensor responsible of propagating data to choose between sending it to a next hop sensor, a procedure which requires a relatively small amount of energy, or to send 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 [ENR06, LNR05, JLPR06] and the main contributions of this paper are to formally prove the connection between energy balancing and lifespan maximization, define the optimality of data 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 centralised 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 typically 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 to balance the energy consumed between the slices. The division of a WSN in slices is illustrated on figure 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.

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 amongst 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. To our knowledge, a solution to the

premature energy depletion of sensors close to the sink was first proposed in [ENR06]. This problem has since then been investigated in a setting similar to ours, where the network is divided in slices, and energy consumption studied at the slice level, in a series of papers [ENR06, LNR05, SD05, OS06, JLPR06]. In order to avoid or minimize the premature depletion of sensors close to the sink, [ENR06, LNR05, JLPR06] use *ejections,* first introduced in [ENR06]. Independently, [SD05] considers varying the battery levels between slices, called rings, and [OS06] considers varying the emission ranges between slices, called coronas; however, in both cases the data propagation is exclusively multi-hop.

In [ENR06], the problem of finding an energy-balanced solution to data propagation in a Wireless Sensor Network was considered for the first time. The lifespan of the network is maximised 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 has to be propagated by the WSN towards a sink located at the center 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 centralised 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.

We point out that neither [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 [ENR06, LNR05] are useless. Our algorithm is a generalisation 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.

The paper is organized as follows. In section 2, we introduce an appropriate mathematical model of a WSN, together with notational conventions and some preliminary technical results. Section 3 presents the off-line centralised algorithm computing the parameters to adjust the distributed probabilistic data propagation algorithm. Section 4 is a formal proof of the optimality of the probabilistic data propagation algorithm using the parameters computed by the centralised algorithm. Finally, section 6 presents numerical validations of the data propagation algorithm and a simple spreading technique to overcome the unbalanced energy amongst sensors of the same slice, an issue which was not taken into account by the mathematical model of section 2.

2 Model and Notations

The model we study is the same as the one from [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 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 figure 1. When considering

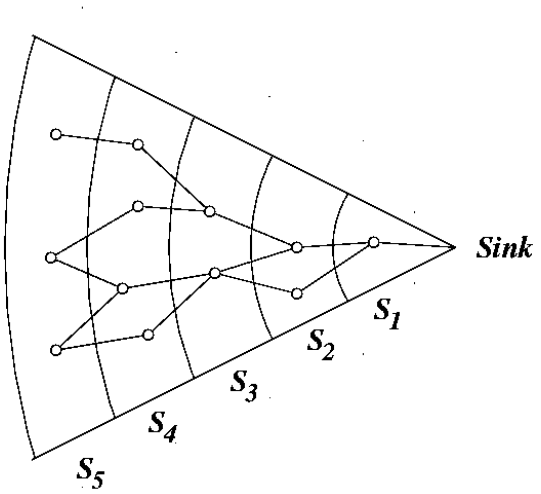

Figure 1: Network division in slices

a slice S_k, we take the convention to call 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 [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 its next slice S_{i-1}, we say that the message is *slid* from S_i, and on the contrary, 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 to be 1 Joule 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 sliding messages* 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 [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 flux 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 $p_i = \frac{f_i}{f_i + j_i}$.

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

Below is a table listing the symbols used in this article, together with their units and interpretation.

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

Table: Summary of symbols

2.1 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

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

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

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

Among this total number of messages, a $(1-\epsilon_i)$ fraction reaches slice S_{i-1} after having been slid from slide 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 $i, i-1, \ldots, 1$

$$(1-\epsilon_i)\frac{b_i}{(1-\epsilon_i) + \epsilon_i d_i^2} \text{ (message)} = \frac{b_{i-1}}{(1-\epsilon_{i-1}) + \epsilon_{i-1} d_{i-1}^2} \text{ (message)} \tag{1}$$

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 $1-\epsilon_i$ and ejected with probability ϵ_i (for $1 \leq i \leq N$). 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 $1 \leq k \leq N$ and that equation (1) is satisfied if $2 \leq i \leq k$. Then for every S_i with $1 \leq i \leq k$, the lifespan of S_i is a constant equal to*

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

Relation (1) can be rewritten in the following useful form:

$$\epsilon_{i+1} = \frac{\frac{b_{i+1}}{b_i}\left(1 - \epsilon_i + \epsilon_i {d_i}^2\right) - 1}{\frac{b_{i+1}}{b_i}\left(1 - \epsilon_i + \epsilon_i {d_i}^2\right) + {d_{i+1}}^2 - 1} \tag{2}$$

Setting $\epsilon_1 = 1$ (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.

3 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 2. 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 same

lengths $\{b_i\}_{1\leq i\leq N}$, $\{d_i^2\}_{1\leq i\leq N}$ and $\{g_i\}_{1\leq i\leq N}$ 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 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 2. The *output* is a sequence $\{p_i\}_{1\leq i\leq N}$ representing the parameters of the probabilistic data propagation algorithm: in order to maximize the lifespan of the network, each slice S_i for $1 \leq i \leq N$ 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 figure 2. In turn, with the energy balancing constraint of equation (2), 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 in 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 from the first slice, this is illustrated by the crossed-out rectangular of the middle picture of figure 2. Then, the remaining messages to be handled are slid to S_1 using the probability ϵ_2 computed in Proposition 2.1, hence equally increasing the power to battery ratio of slices S_1 and S_2, which thus remain balanced, as illustrated by the black rectangular in the middle picture of figure 2. 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 figure 2. Since the b_i's

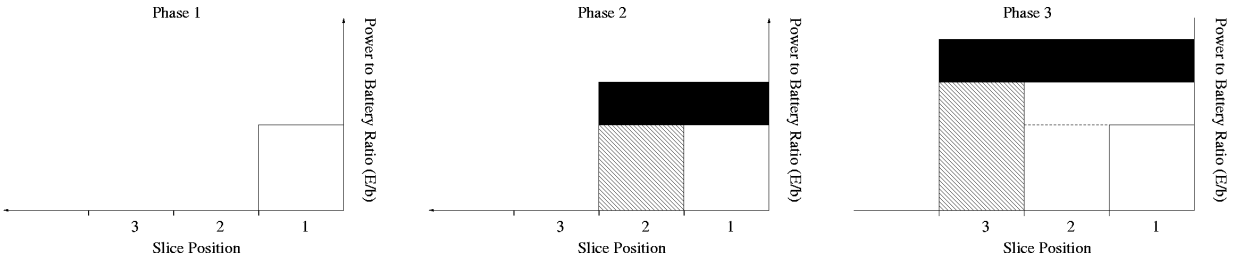

Figure 2: Intuitive idea of the Algorithm

and d_i's are given from the input for $1 \leq i \leq N$, we can compute the ϵ_i's ($1 \leq i \leq N$), the solution to relation (1) with $\epsilon_1 = 1$.

Remark. *Notice that although it although equation (1) implies that $\epsilon_i \leq 1$, nothing guarantees that $\epsilon_i \geq 0$. For simplicity, let us make the temporary assumption that $\epsilon_i \geq 0$. We treat the case with negative ϵ's in section 3.1.2.*

For running, our algorithm needs the following *variables*, for $1 \leq i \leq N$:

- 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 $1 \leq i \leq N$. The algorithm then treats each slice one at a time from S_1 towards S_N. First, S_1 is being 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 $J_i := \frac{b_i}{b_{i-1} {d_i}^2} P_{i-1}$, which is equivalent to

$$\frac{b_i}{J_i {d_i}^2}(s) = \frac{b_{i-1}}{P_{i-1}}(s), \tag{3}$$

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 on figure 2. 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 sufficiently many messages to be treated at slice S_i, i.e. it should be that the following holds:

$$J_i = \frac{b_i}{b_{i-1} {d_i}^2} P_{i-1} \leq G_i. \tag{4}$$

We overcome this limitation in section 3.1.1. For the time being, we consider only the case where the initial G_i's are large enough to ensure that equation (4) holds. So far energy balance is achieved for slices S_i to S_1 (because of equation (3) and by induction), which is represented by the crossed-out rectangles of figure 2 . 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 figure 2. The strategy is the following: a fraction $(1 - \epsilon_i)$ 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:

$$\begin{aligned} F_i &:= (1 - \epsilon_i)\, G_i \\ J_i &:= J_i + \epsilon_i G_i \\ G_i &:= 0 \end{aligned}$$

Among the $G_i\,(1 - \epsilon_i)$ messages slid from S_i towards S_{i-1}, a fraction $1 - \epsilon_{i-1}$ 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 number of slid and ejected messages have to be updated. The algorithm implementing this idea needs to do the following:

$$\begin{aligned}
F_{i-1} &:= F_{i-1} + (1-\epsilon_{i-1})(1-\epsilon_i) G_i, \\
J_{i-1} &:= J_{i-1} + \epsilon_{i-1}(1-\epsilon_i) G_i, \\
F_{i-2} &:= F_{i-2} + (1-\epsilon_{i-2})(1-\epsilon_{i-1})(1-\epsilon_i) G_i, \\
J_{i-2} &:= J_{i-2} + \epsilon_{i-2}(1-\epsilon_{i-1})(1-\epsilon_i) G_i, \\
F_{i-3} &:= F_{i-3} + (1-\epsilon_{i-3})(1-\epsilon_{i-2})(1-\epsilon_{i-1})(1-\epsilon_i) G_i, \\
J_{i-3} &:= J_{i-3} + \epsilon_{i-3}(1-\epsilon_{i-2})(1-\epsilon_{i-1})(1-\epsilon_i) G_i, \\
&\dots
\end{aligned}$$

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

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

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

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

and so forth for slices S_{i-2}, S_{i-3}, up to S_1. But because the ϵ_i's satisfy equation (2), the $\frac{b_j}{m_j}(s)$'s $(1 \leq j \leq i)$ have the same value, as follows from proposition 2.1. So when we finish treating slice S_i the average times before running out of energy in $S_1, S_2, \dots, S_i$ are equal. Again, this is illustrated by the black rectangles of figure 2. 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 $1 \leq i \leq N$ slice S_i treats a total of $F_i + J_i$ $(messages/s)$, of which F_i are being slid and J_i are being 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: $\left\{\frac{F_i}{F_i+J_i}\right\}_{1\leq i\leq N}$

3.1 Special cases

In this section, we lift the assumption that all ϵ's are positive (remark 3), and from the assumption that equation (4) holds, starting with the former.

3.1.1 First Case: Too Many Sensors or Too few Messages

Suppose that while executing the algorithm from the previous section, equation (4) 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 figure 2, this means that the crossed-out rectangle is not as high as 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 equation 2, only this time we force $\epsilon_i = 1$, 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 sufficiently 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, we can lift the constraint of having $\epsilon_i = 1$ 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 $\epsilon_i = 1$, 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

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

  which is the power to battery ratio increase needed for S_{start} to "catch up" with $S_{start-1}$.

- Stack the value of the previous ϵ's. Set new current values for ϵ_j's for $start \leq j \leq N$:

  - ϵ_{start} is set to 1, so that slice S_{start} ejects every sliding messages it sees, in order to try to catch up with $S_{start-1}$.
  - The other ϵ_k's with $start < k \leq N$ are computed using equation (2).

Once this is done, we can go down one level in the recursion, which essentially means redoing the algorithm from section 3, 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 on level of recursion and while treating slices $S_{start+1}, S_{start+2}$, 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 level.

- Slice S_{start} may be able to catch up (using messages previously slid from slices $S_{start+1}$ to S_{k-1} and the newly messages slid from S_k), i.e. sufficient messages will have been slid from its 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 happen: slices $S_k, S_{k-1}, \ldots, S_{start}$ have so far the same power to battery ratio (but less than $S_{start-1}$), 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 3 should be adapted follows, assuming it has gone down some level of recursion and slice S_k is being treated for some $k > start$ while slice S_{start} is trying to catch up with slice $S_{start-1}$.

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 $\frac{b_k}{G_k d_k^2}(s) > \frac{b_{k-1}}{F_{k-1}+J_{k-1}d_{k-1}^2}(s)$ we just try do 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 $J_k = \frac{b_k}{d_k^2 b_{k-1}}\left(F_{k-1} + J_{k-1}d_{k-1}^2\right)$ 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 equations
   - 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 $S_{start+1}$) catch up with $S_{start-1}$. 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 $\Delta_t = (max - \frac{1}{b_k}(F_k + J_k d_k^2))^{-1}$ (s) and $msgToGoUp = \frac{b_k}{\Delta_t\left(\epsilon_k d_k^2 + (1-\epsilon_k)\right)}$. Let $\Phi = \max\{G_k,\ msgToGoUp\}$. From the remaining G_k messages, we further slide and eject respectively $F = (1-\epsilon_k)\cdot\Phi$ and $J = \epsilon_k \cdot \Phi$ 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 $S_{start-1}$. We thus need to make the following adjustments: $F_k = F_k + F$, $J_k = J_{start} + J$ and $G_k = G_k - F - J$. The F_j's and J_j's for $start \leq j < k$

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 $S_{start-1}$, i.e. if Φ was 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 lifespans (from slice S_1 towards S_N), and which is "locally" energy-balanced, for example, we could have:

$$\frac{b_1}{P_1} = \frac{b_2}{P_2} = \frac{b_3}{P_3} < \frac{b_4}{P_4} < \frac{b_5}{P_5} = \frac{b_6}{P_6} \leq \dots$$

Although not reaching energy balance, we shall prove in section 4 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: $\frac{P_i}{b_i} = \frac{P_{i-1}}{b_{i-1}}$
- The algorithm does not return from this recursive call and the solution is *not* energy-balanced: $\frac{P_i}{b_i} < \frac{P_{i-1}}{b_{i-1}}$. Furthermore, since S_i was trying to "catch up" with S_{i-1} and since we set $\epsilon_i = 1$ (point (3.1.1) of the algorithm), it holds that $F_i = 0$, and thus that

$$p_i = 0 \tag{5}$$

In section 4, we use equation (5) to show that this solution is always optimal.

3.1.2 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 page 8) does not hold. From equation (2), 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 $1 - \epsilon_i$ rest to the next slice. After

this, there are $j_i + k\epsilon_i$ ejected messages and $f_i + k\,(1 - \epsilon_i)$ slid messages. The fact that the ϵ's satisfy equation (2) ensures that energy balance is conserved (at least locally if we are already into a recursive call as described in section 3.1.1). A difficulty follows from the fact that since ϵ_i is negative, j_i becomes negative if

$$k \cdot \epsilon_i + j_i < 0 \Leftrightarrow k > j_i/\epsilon_i \tag{6}$$

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 $k_i = k \cdot \Pi_{j=i+1}^{l}\,(1 - \epsilon_i)$ fraction of the k initial messages reaches slice S_i. $maxSlide$ is defined as the maximum value k such that $k_i\epsilon_i + j_i \geq 0$ for $1 \leq i \leq l$, or equivalently, the maximum value such that $k_i \leq \frac{j_i}{|\epsilon_i|}$ for every $1 \leq i \leq l$ such that $\epsilon_i < 0$, and it can be computed by the following procedure.

Algorithm 3.1: COMPUTEMAXSLIDE(i)

Input: i, a slice number.
Output: max, the max number of messages S_i can slide.
global $\epsilon[\,]$**comment:** This is an array storing the values of the ϵ's

local $F \leftarrow 1$
local $ejected[\,]$
for $(k \leftarrow i; k \geq 1; k \leftarrow k-1)$
do $\begin{cases} ejected[k] \leftarrow F * \epsilon[k] \\ F \leftarrow F * (1 - \epsilon[k]) \end{cases}$
local $max \leftarrow \infty$
for $(k \leftarrow i;\ k \geq 1;\ k \leftarrow k-1)$
do $\begin{cases} \textbf{local } j \leftarrow ejected[k] \\ \textbf{if } j < 0 \\ \quad \textbf{then} \\ \quad \textbf{do} \begin{cases} j \leftarrow -j \\ \textbf{if } max = \infty \\ \quad \textbf{then } max \leftarrow j[k]/j \\ \quad \textbf{else} \\ \quad \textbf{do} \begin{cases} \textbf{local } tmp\_max \leftarrow j[k]/j \\ \textbf{if } tmp\_max < max \\ \quad \textbf{then } max \leftarrow tmp\_max \end{cases} \end{cases} \end{cases}$
return (max)

Once we have computed $maxSlide$, we can decide what to do when slice S_i wants to slide k messages. If $k \leq maxSlide$, 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 $1 \leq k \leq j$,Notice that if we are inside a recursion of the type described in section 3.1.1, we do not recompute all ϵ_k's for $1 \leq k \leq N$, but we rather set $\epsilon_{start} = 1$ and recompute all the ϵ_i's for $start < i \leq N$, where $start$ is the place where the last recursion took place (c.f. point (3.1.1) of the enumeration on section 3.1.1). While recomputing the ϵ's, whenever an $\epsilon_k < 0$ 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 equation (2), 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:

$$p_k = 1 \tag{7}$$

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

3.2 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 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 being treated for some $1 \leq k \leq N$), new values of J_i and F_i have to be computed for $1 \leq i \leq k$ (as follows from equations 5 to 5). 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 $k \leq N$). The reason why the F_i and J_i may have to be updated more then 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 time. 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 3.1.1, 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 3.1.2), and it limits the maximum number of messages allowed to be sent during the first step to $maxSlide$, the value which is computed by the algorithm 3.1, the rest being sent in a second step. In the worst case, the 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 3.1 occur (i.e. when equation (4) holds or if condition (6) is never satisfied), then the algorithm runs in $\mathcal{O}(N^2)$.

4 Proof of Optimality

In this section, we prove that our algorithm produces an optimal solution, in the sense that it maximizes the *lifespan* (c.f. definition 2).

Convention. *In this section, we consider a fixed sensor network of size N, with fixed event distribution $\{g_i\}_{1\leq i\leq N}$ and fixed battery levels $\{b_i\}_{1\leq i\leq N}$. A* configuration *C of the network is the choice of a sliding probability assignment $\{p_i\}_{1\leq i\leq N}$ 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 parameter: 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:*

$$\frac{P_i}{b_i} \geq \frac{\tilde{P}_i}{b_i}$$

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

$$\forall i \ \frac{P_i}{b_i} \leq \frac{\tilde{P}_i}{b_i}$$

and for at least one of the i's

$$\frac{P_i}{b_i} < \frac{\tilde{P}_i}{b_i} \tag{8}$$

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}, $\dots p_{N-i+1}$) 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 $1 \leq i \leq N$, if we use ${}^iE$ and ${}^ib$ to designate the power and battery of configuration C_i, the following holds:

$$\begin{aligned} \frac{{}^iP_k}{{}^ib_k} &= \tfrac{P_k}{b_k} \quad \forall N \geq k > N-i \\ \frac{{}^iP_i}{{}^ib_i} &\geq \tfrac{P_i}{b_i} \end{aligned}$$

Where equation (9) follows from the definition of C_i and where equation (9) follows from the easy observation that $\frac{{}^iP_i}{{}^ib_i} \geq \frac{\tilde{P}_i}{b_i}$ combined with equation (4). Next, let $k = \max\left\{i \mid \frac{P_i}{b_i} < \frac{\tilde{P}_i}{b_i}\right\}_{1\leq i\leq N}$, which exists by (8). Then for every $i \leq k$ the inequality in (9) becomes strict. In particular, for $i = 0$ (and using the fact that $C_0 = C$) it becomes $\frac{P_0}{b_0} > \frac{P_0}{b_0}$, 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 [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. $\forall i \ \frac{P_i}{b_i} = \frac{P_{i+1}}{b_{i+1}}$), then for every other configuration $\tilde{C}$, we have the following inequality, with equality if and only if $C = \tilde{C}$: $\min\left\{\frac{b_i}{P_i}\right\} \geq \min\left\{\frac{\tilde{b}_i}{\tilde{P}_i}\right\}$, that is, C maximizes the lifespan amongst all possible configurations.*

Next we generalize corollary 4.

Lemma. *Let C be a configuration of our network. Let $max = \max\left\{\frac{P_i}{b_i}\right\}$. Let $k = max\left\{i \mid \frac{P_i}{b_i} = max\right\}$, and let $l = min\left\{i \mid \frac{P_i}{b_i} < max\right\}$, 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 (c.f. figure 2). 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 page 17. □

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 power to battery ratio maximum 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 4. To see that this is the case, the main ingredients are equations (5) and (7). We leave the easy details to the reader. □

5 Simulations

In this section, we present numerical validation of our algorithm. The approach consists in randomly and uniformly scattering sensor nodes in a sector of the plane, with a sink placed at the center of the sector, c.f. figure 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 $1 \leq i \leq N$, 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 3, we compute p_i the sliding probability which are predicted to balance the energy between slices. The simulations we present show that this is indeed the case.

5.1 First simulation

We divide the time in 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 amongst 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 the 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 further 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 figure 3, we plot the radius of each of 6280 nodes scattered in a 20 meter radius and 90 degrees sector against the energy spent by each of these nodes while reporting events to the sink according to the strategy described here above 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 amongst sensors.

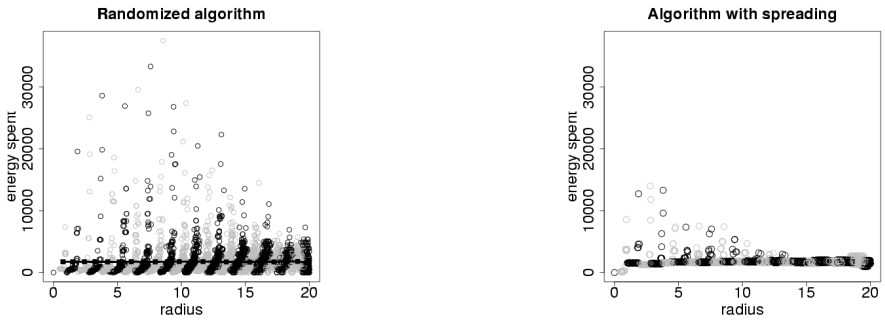

Figure 3: Simulations

5.2 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 techniques* (c.f. [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 less 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 5.1, nodes of each slice 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 figure 3, we see that the use of the spreading technique induces a well balanced energy consumption not only amongst slices but also amongst 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 simulations results are comparable to those presented in this section.

6 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 balance solution does not exist. The main idea is to divide the WSN in 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 3. 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 amongst slices, but also amongst all sensors of the network. Probably other spreading techniques could be used, 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 transmission 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.

References

[AKK05] Jamal N. Al-Karaki and Ahmed E. Kamal. A taxonomy of routing techniques in wireless sensor networks. In Mohammad Ilyas and Imad Mahgoub, editors, *Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems*, pages 6.1–6.24. CRC Press, 2005.

[AY05] Kemal Akkaya and Mohamed Younis. A survey on routing protocols for wireless sensor networks. *Ad Hoc Network Journal*, 3/3:325–349, 2005.

[BCN05] A. Boukerche, I. Chatzigiannakis, and S. Nikoletseas. Power-Efficient Data Propagation Protocols for Wireless Sensor Networks. *SIMULATION*, 81(6):399–411, 2005.

[BN04] A. Boukerche and S. Nikoletseas. *Wireless Communications Systems and Networks*, chapter Protocols for Data Propagation in Wireless Sensor Networks: A Survey, pages 23–51. Kluwer Academic Publishers, 2004.

[CNS02] I. Chatzigiannakis, S. Nikoletseas, and P. Spirakis. Smart dust protocols for local detection and propagation. In *2nd Workshop on Principles of Mobile Computing (POMC)*, pages 9–16. ACM, ACM Press, 2002.

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

[ENR06] C. Efthymiou, S. Nikoletseas, and J. Rolim. Energy balanced data propagation in wireless sensor networks. *Wireless Networks (WINET) Journal*, 2006. Best papers of the 4th Workshop on Algorithms for Wireless, Mobile, Ad Hoc and Sensor Networks (WMAN 2004).

[HCB00] W.R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy efficient communication protocol for wireless microsensor networks. In *Hawaii International Conference on Sytem Sciences (HICSS)*, number 33, 2000.

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

[HKK04] Kook-Hee Han, Young-Bae Ko, and Jai-Hoon Kim. A novel gradient approach for efficient data dissemination in wireless sensor networks. In *International Conference on Vehicular Technology Conference (VTC)*. IEEE, September 2004. To appear.

[IGE00] C. Intanagowiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *International Conference on Mobile Computing (MOBICOM)*, number 6. ACM/IEEE, 2000.

[JLPR06] Aubin Jarry, Pierre Leone, Olivier Powell, and Jose Rolim. An optimal data propagation algorithm for maximizing the lifespan of sensor networks. In *To appear in proceedings of DCOSS'06*, 2006.

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

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

[OS06] S. Olariu and 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, April 2006.

[RAdS+00] Jan M. Rabaey, M. Josie Ammer, Julio L. da Silva, Danny Patel, and Shad Roundy. Picoradio supports ad hoc ultra-low power wireless networking. *Computer*, 33(7):42–48, 2000.

[SD05] M.L. Sichitiu and R. Dutta. Benefits of Multiple Battery Levels for the Lifetime of Large Wireless Sensor Networks. In *NETWORKING 2005: 4th International IFIP-TC6 Networking Conference*, Lecture Notes in Computer Science, pages 1440–1444. Springer Berlin/Heidelberg, May 2005.

[SL05] Michael J. Sailor and Jamie R. Link. "smart dust": Nanostructures devices in a grain of sand. *Chemical Communication*, (11):1375–1383, 2005.

[SS01] Curt Schurgers and Mani B. Srivastava. Energy efficient routing in wireless sensor networks. In *Military Communications Conference (MILCOM)*, pages 357–361, October 2001.

[STGS02] C. Schurgers, V. Tsiatsis, S. Ganeriwal, and M. Srivastava. Topology management for sensor networks: Exploiting latency and density. In *International Conference on Mobile Computing (MOBICOM)*, number 8. ACM/IEEE, 2002.

[WLLP01] Brett Warneke, Matt Last, Brian Liebowitz, and Kristofer S.J. Pfister. Smart dust: Communicating with a cubic-millimeter. *computer*, 34(1):44–51, 2001.

[YZLZ05] Fan Ye, Gary Zhong, Songwu Lu, and Lixia Zhang. Gradient broadcast: A robust data delivery protocol for large scale sensor networks. *Wireless Networks (WINET)*, 2005. To appear.

A Pseudo code of the Algorithm

We provide hereunder 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($eject$)
 $j[i] \leftarrow eject$
 $g[i] \leftarrow g[i] - j[i]$

procedure SLIDE(F)
 SLIDECAREFUL(F)

procedure SLIDECARELESS(F)
 $g[i] \leftarrow g[i] - F$
 for $(k \leftarrow i; k \geq 1; k \leftarrow k - 1)$
 do $\begin{cases} f[k] \leftarrow f[k] + F * (1 - e[k]) \\ j[k] \leftarrow j[k] + F * e[k] \\ F \leftarrow F * (1 - e[k]) \end{cases}$

procedure SLIDECAREFUL(F)
 local $maxCarelessSlide \leftarrow$ COMPUTEMAXSLIDE()
 if $maxCarelessSlide = \infty$ **or** $F <= maxCarelessSlide$
 then
 do $\{$SLIDECARELESS(F)
 else
 do $\begin{cases} \text{SLIDECARELESS}(maxCarelessSlide) \\ F \leftarrow F - maxCarelessSlide \\ \epsilon[\,] \leftarrow \text{EPSILONS}(startPosition, "withCaution") \\ \text{SLIDE}(F) \end{cases}$

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

procedure UPONELEVEL()
 local $max \leftarrow$ POP($maxs[\,]$)
 $startPosition \leftarrow$ POP($startPositions[\,]$)
 local $tmp[\,] \leftarrow$ POP($epsilons[\,]$)
 $\epsilon[\,] \leftarrow tmp[\,]$
 $recLevel \leftarrow recLevel - 1$

procedure DOWNONELEVEL()
 PUSH($maxs[\,], max$) **comment:** store old max
 $max \leftarrow$ AVGNRJ($i-1$)
 PUSH($startPositions[\,], startPosition$)
 $startPosition \leftarrow i$
 local $tmpArray[\,] = \epsilon[\,]$
 PUSH($epsilons[\,], tmpArray[\,]$) **comment:** store old epsilons
 $e[\,] \leftarrow$ EPSILONS(i)
 $recLevel \leftarrow recLevel + 1$

Algorithm A.4: COMPUTATION OF THE EPSILONS()

procedure EPSILONS($first$; $option$)
 local $\epsilon[\,]$
 for $k \leftarrow 1$ **to** $first$
  do $\{\epsilon[k] = 1$
 for $k \leftarrow first + 1$ **to** N
  do
   local $A \leftarrow \frac{d[k]^2-1}{b[k]}$
   local $B \leftarrow (\epsilon[k-1] * (d[k-1]^2 - 1) + 1)/b[k-1]$
   $\epsilon[k] \leftarrow \frac{B-1/b[k]}{A+B}$
   if $option =$ "$withCaution$"
    then
     do
      if $\epsilon[k] \leq 0$ **and** $j[k] = 0$ **and** $k \leq i$
       then
       do $\{\epsilon[k] = 0$
 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)
      { 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)
```