Decentralized Scattering of Wake-Up Times in Wireless Sensor Networks

Alessandro Giusti[1], Amy L. Murphy[2,3], and Gian Pietro Picco[1,4]

[1] Politecnico di Milano, Italy
giusti@elet.polimi.it
[2] University of Lugano, Switzerland
[3] ITC-IRST, Povo, Italy
murphy@itc.it
[4] University of Trento, Italy
picco@dit.unitn.it

Abstract. Duty-cycling in wireless sensor networks (WSNs) has both beneficial effects on network lifetime and negative effects on application performance due to the inability of a sensor to perform while it is sleeping. In a typical scenario, the active periods of nodes are randomly initialized, leading to unpredictable and often sub-optimal performance. In this paper, we propose a fully decentralized *wake-up scattering* algorithm that uniformly spreads wake-up times of nearby sensors. Interestingly, our approach is complementary and dual to existing approaches that aim at synchronizing (instead of scattering) times, and to those that focus on spatial (instead of temporal) coverage. Wake-up scattering is beneficial in several application scenarios, three of which are considered here: responsiveness to one-hop queries from a mobile base station, sensing coverage for event detection, and latency in multi-hop communication. Our evaluation shows that, w.r.t. a random assignment of wake-up times, wake-up scattering brings improvements in all these measures, along with a positive impact on the network lifetime.

1 Introduction

One of the primary challenges for wireless sensor network (WSN) applications is energy management. A common solution that dramatically increases sensor lifetime is duty-cycling, i.e., periodically switching on and off communication and/or sensing capabilities. The major factors that affect lifetime are the length of the period (the *epoch period*) and the duration of the on-time (the *awake interval*). However, another variable critically affects the *performance* of the network, namely *when* in each period each node activates. We refer to this as the *wake-up time* of the sensor node. Typically, the wake-up time is randomly established, relating only to when the sensor is initially activated.

We first encountered the potential negative performance impact of wake-up times during our work on TinyLIME [1], a middleware that allows applications running on mobile bases stations to access the data (e.g., temperature) of nearby (one-hop) sensors through a tuple space interface. The system is designed to account for sensors that duty-cycle their communication components, therefore each time a base station needs to contact a sensor, it repeats its request until a sensor wakes up, receives the request,

K. Langendoen and T. Voigt (Eds.): EWSN 2007, LNCS 4373, pp. 245–260, 2007.

and replies. In the best case, one of the sensors is awake when the first request is sent and responds immediately. However, in the worst case, all sensors in range have just cycled off and will not reactivate communication until the next epoch. Therefore, the base station repeatedly sends the request until the sensors wake up and one replies. In a system with an epoch of minutes, such delays in response time can be significant.

Our solution is to introduce a calibration phase to deliberately select the time within the epoch in which each sensor wakes up. To improve performance, the wake-up times should be scattered evenly throughout the epoch, minimizing the time that a base station must wait before a sensor becomes active. Scattering is performed such that response to a base station at any location in the system is minimized, thus avoiding the need to track and adjust to a possibly rapidly moving base station.

We achieve this goal with a fully decentralized algorithm that incrementally improves the scattering of wake-up times among neighboring nodes. Essentially, each node periodically determines the wake-up times closest to its own among its one-hop neighbors, then moves its own wake-up time so as to minimally overlap with others. Because decisions are made entirely locally and independently, a single run of this algorithm may result in poor scattering. However, the system quickly stabilizes to a scattered network in very few repetitions.

Interestingly, this *wake-up scattering* approach can be naturally applied in many other situations. For example, in contrast to the aforementioned scenario which duty-cycles *communication*, consider a system that duty-cycles *sensing* activities. Scattering the wake-up time for sensing allows the area to be more effectively covered at all times, as opposed to activating all sensors in a given area at the same time. More importantly, the guarantee of a better scattering in time enables one to design a network with shorter awake intervals, therefore achieving a longer overall network lifetime. Another use is in a system that exploits a tree for communication from sensors to a fixed, centralized sink. To speed up communication from the edge of the network to the base station, parent nodes should be active *after* their children. In this case, proper scattering of wake-up times can decrease the average latency for data traveling on the network towards the root, thus increasing network performance.

Finally, it is worth noting how the problem of *wake-up scattering* is complementary and dual to other problems in WSNs. For instance, the idea of controlling wake-up times to enable communication has been studied in the context of MAC protocols, but with the goal of *synchronizing*, instead of scattering, the wake-up times. Similarly, the problem of covering an area to detect an event has been studied only for what concerns *spatial* coverage of an area and not, to our knowledge, in terms of coverage in time.

Summarizing, in this paper we put forth the following main contributions:

- we introduce for the first time the notion of wake-up scattering;
- we present a fully decentralized algorithm solving this problem;
- we evaluate this algorithm in three application scenarios common in WSNs.

These contributions are presented in Section 2, 3, and 4, respectively. The paper concludes by reporting related work in Section 5, followed by a brief summary.

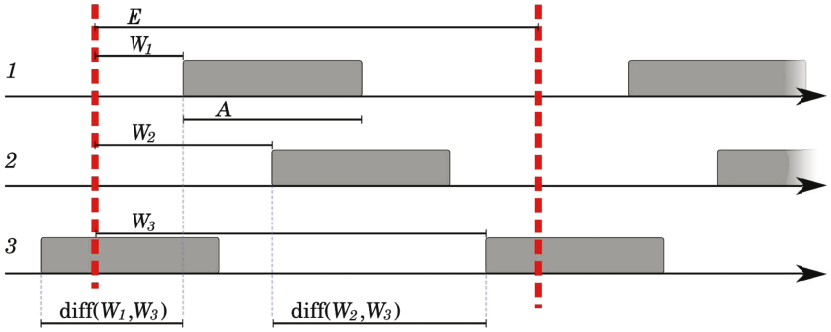


Fig. 1. Key definitions illustrated with three nodes. Time increases left to right. E is the epoch period, A the awake interval, W_i the wake-up time for node i, and $\mathrm{diff}(W_i, W_j)$ the length of time between the wake-up times of nodes i and j.

2 Model and Motivation

Before presenting the details of our protocol, we briefly formulate the wake-up scattering problem and outline three reference scenarios that benefit from our approach.

2.1 Model

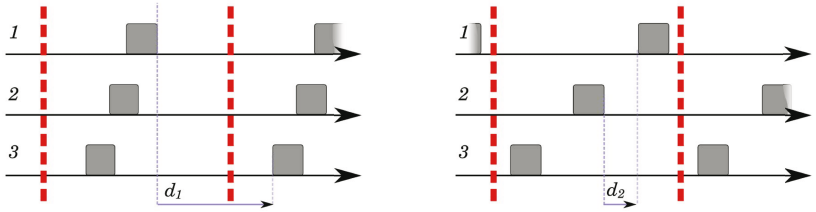
For the purposes of this work, we assume a straightforward network model in which non-mobile sensor nodes with circular communication and sensing radii are deployed in an unobstructed field. We further assume that all nodes have the same communication range. While these assumptions significantly simplify our analysis, they can be altered without affecting the algorithm's fundamental validity. Finally, we assume that the topology is not known and nodes are only aware of the neighboring nodes they can directly communicate with.

The primary premise of our work is that sensors operate in a duty-cycling manner, turning on and off certain capabilities at regular intervals. The length of the interval is the *epoch period* E, while the time during which a node is on is the *awake interval* A. Although it is possible to allow both E and A to vary for each node, in this work we assume they are system-wide, deployment-time parameters, equal for all nodes. These parameters are illustrated in Figure 1. Although this figure shows that the epochs at all three nodes are synchronized, this is not a requirement for our approach.

The figure also represents the *wake-up time* W_i, for each node i, and the interval between wake-up times measured with $\mathrm{diff}(W_i, W_j)$. This operator accounts for the fact that the wake-up patterns repeat across epochs, therefore it measures the difference in the start of the awake intervals between any two nodes, even those waking up at the beginning or end of an epoch.

2.2 Wake-Up Scattering: Objectives and Usage Scenarios

This work illustrates an approach to modify, in a decentralized fashion, the wake-up time W_i of the nodes in a WSN such that they are scattered as much as possible. We



(a) Before scattering. (b) After scattering.

Fig. 2. Three connected nodes before and after scattering. d_i shows the maximum delay for a base station to wait for a response in each setting.

maintain that such *wake-up scattering* improves application performance in many situations. Here, we focus on three representative scenarios with different characteristics, for which we detail the application functionality and goal as well as the improvement expected from wake-up scattering. In Section 4 we use these same scenarios to evaluate the effectiveness of our approach.

Responsiveness to Local Queries. As mentioned in the introduction, our first scenario is motivated by our work on the TinyLIME middleware [1]. In TinyLIME applications, sensor nodes are distributed through a region and a mobile base station queries for sensor data close to it. The goals in this environment are two-fold. First, we wish to minimize the time that a base station needs to wait to receive data. In other words, the goal is to always have at least one node near the base station awake, or about to wake up, in order to respond to queries. Second, we wish to minimize the maximum number of nodes that are awake at the same time, therefore reducing the possibility of collisions among query responses issued simultaneously by awake nodes.

Rather than trying to adapt to the possibly rapidly moving base station, our approach yields a wake-up scattering solution that suffices for any location of the base station. We accomplish this by considering the wake-up times among *nearby* nodes, motivated by the fact that such nodes are likely to be in contact with the base station at the same time, while those farther away will not. In this case the problem can be defined as the search for a global configuration of wake-up times that maximizes the minimum difference in wake-up times for any pair of nearby nodes. Put another way, our goal is that any pair of nearby motes should have their wake-up times as far away from each other as possible.

Figure 2 intuitively demonstrates the benefits of wake-up scattering with three nodes in a fully connected network (i.e., each pair can communicate) both before and after scattering. It can clearly be seen that after scattering, fewer nodes are awake at any given time, therefore reducing the possibility of communication interference. The figure also shows the maximum time, d_i that an unlucky base station needs to wait before receiving a response to a query, with the maximum delay d_1 much larger than d_2. On average, a base station initiating a query at any point during the epoch will have less delay to receive a response after scattering.

Event Coverage. Another popular use for sensor networks is detection of an event occurring somewhere in the field covered by the sensors. In this scenario, the critical parameters are the sensing range of the devices and the amount of time that the sensors

are active. To detect an event, it must occur both *spatially* inside the sensing range of a node and *temporally* during the awake time of the node.

Our goal in this scenario is to maximize the percentage of events detected. By treating sensing analogous to communication, we can reuse the previous wake-up scattering technique to manage the wake-up times of *sensing* devices, thus guaranteeing that the intervals of data acquisition are spread as evenly as possible throughout the epoch thus improving event detection. Figure 2 can be used to visualize the concept, by defining the awake interval as the active time of the sensor devices. Although after scattering there are still gaps in the sensed time, these gaps are smaller than before scattering. To completely eliminate these breaks in sensing coverage, the awake interval of the sensors should be modified dynamically. However, in this paper we do not consider this additional optimization and limit ourselves to the management of wake-up times.

Data Latency in Tree-Based Networks. The third scenario we consider is the construction of a tree to funnel data from the edges of the network to a centralized base station, a common approach for applications such as TinyDB [2]. When considering nodes that duty-cycle their communication capabilities, we want to guarantee that when a child has data to send, either its parent in the tree is immediately active or it will be soon, thus minimizing the latency of a data packet on the path to the sink.

If we ignore propagation delays, the trivial solution is to force all nodes to turn on their communication at the same time. Not only does this solution exhibit a high collision probability, if sensors are active only at the same time as communication, sensing will also be synchronized and the potential benefits of scattering for event detection will not be achieved. Therefore, our goal is to maintain good event coverage while simultaneously reducing data latency.

To achieve this, we require that all nodes keep track of which of their neighbors can serve as parents towards the sink. We assume this is accomplished by broadcasting a message from the sink, that allows nodes to know their distance from it, and to infer that *all* neighbors with distances less than their own are *potential* parents.

When a node has a message to propagate to the sink (i.e., data collected from one of its own sensors or a message from one of its children), it must forward this to one of its (potential) parents, determined as above. We assume that the node tracks the wake-up time of its parents, and thus knows which parent will be awake next. If one is currently awake, the packet can be immediately sent; if not, the node must wait until a parent is awake, then forward the message. If the parent's wake-up time is much after the end of the awake interval of the child, rather than stay awake waiting, energy-savings may be achieved by putting the child to sleep and waking it up later when the parent is awake.

Intuitively, this gap between the readiness of the data at the child and the readiness of the parent to receive the data is minimized if the parent node is the next node to wake up in the sequence of wake-up times. If, instead, the next node is a sibling or child, data cannot make quick progress. Therefore, our goal is to introduce *into the scattering algorithm* the constraint that the next node should be a parent. Clearly this cannot be a strict requirement, because a node that must serve as a parent for multiple children can only have one previous to it in the wake-up order. However, we enforce the constraint that every node has either a parent or a sibling next in the wake-up order.

3 A Decentralized Wake-Up Scattering Algorithm

With the previous application targets in mind, we now present the details of our wake-up scattering approach. The algorithm is designed to generate low communication overhead and is easily implementable in a real WSN system. Our presentation first addresses the general scattering necessary for the first two application scenarios, then introduces extensions for multi-hop tree-based communication.

3.1 Overview

To simplify the explanation of the algorithm, we assume that the epoch starting times are synchronized at all nodes and that all start the scattering algorithm at the same time. Section 3.4 outlines the minimal changes required to remove these constraints.

Our wake-up scattering algorithm is inherently distributed among the sensor nodes, with each node making decisions with information only about nodes it can directly communicate with. Wake-up scattering iteratively refines the wake-up times of nodes in a series of calibration rounds. Each calibration round lasts one epoch, after which each node selects its new wake-up time. Our experiments, presented in the next section, demonstrate that after few rounds (on the order of 3-5) the network stabilizes to a well-scattered configuration.

It is worth noting that calibration rounds need not occur in successive epochs, but can be spread out over time, further limiting the already minimal impact of wake-up scattering on the normal operation of the WSN. Furthermore, after calibration stabilizes, it is meaningful to periodically repeat a calibration round to account for the insertion or removal of sensors.

The processing on a node inside a calibration phase is as follows:

1. Each node must learn the wake-up times of *some of* its neighbors. In a synchronized system, the nodes exchange their W_i values at the beginning of the calibration epoch. Each node is only interested in the wake-up times of the neighbor that wakes up immediately before it, `prev`, and immediately after it, `next`.

   The computation of $W_{\texttt{prev}}$ and $W_{\texttt{next}}$ must take into account the fact that the scheduling of wake-up times is repeated across epochs. Therefore, if a mote is the first to wake up in its neighborhood, $W_{\texttt{prev}} = W_{\texttt{last}} - E$, i.e., the wake-up time of the last node that wakes up in the epoch, minus the duration of the epoch E. Similarly, if the node is the last to wake up in the epoch, $W_{\texttt{next}} = W_{\texttt{first}} + E$.
2. Based on the collected information, each node selects a new wake-up time *near* the center of the time interval between $W_{\texttt{prev}}$ and $W_{\texttt{next}}$. The motivations for not moving to the precise center are discussed later. Also, the target is calculated modulo E, forcing the wake-up time to fall inside the epoch.

Figure 3 shows a single calibration round for node 1 and the four nodes in communication with it. In this example, only nodes 1, 2, and 3 change their wake-up times, as indicated by the shift from the light to dark wake-up intervals, while nodes 4 and 5 are already well scattered w.r.t. all their neighbors and therefore do not adjust their wake-up times. The figure shows the key control information for node 1, namely the wake-up time of its `prev` (node 3) and `next` (node 2). $W_{\texttt{target}}$ is the midpoint between these values, although node 1 does not move exactly to this time.

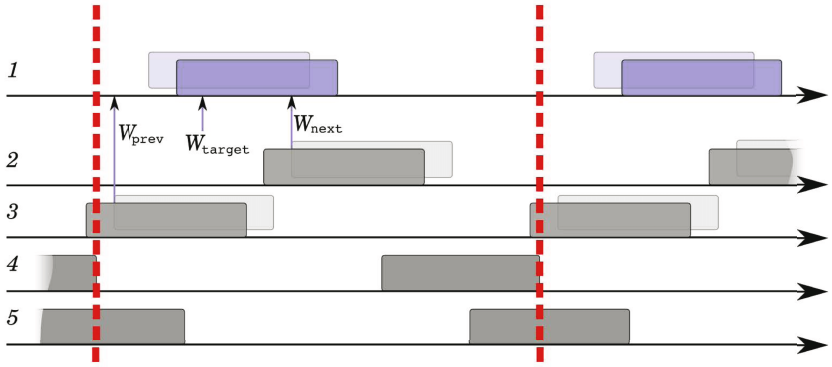


Fig. 3. A scenario with five nodes in the same 1-hop neighborhood, scattering their wake-up times. Light shading indicates the old awake interval, and dark shading the new one. Arrows indicate the $W_{\texttt{prev}}$, $W_{\texttt{next}}$, and the computed target wake-up time for node 1, $W_{\texttt{target}}$.

Indeed, the actual new wake-up time is calculated as a weighted average of $W_{\texttt{target}}$ and the current wake-up time W_1. This weight, $0 < \alpha \leq 1$, is a parameter to the algorithm whose primary effect is to regulate the speed at which the algorithm stabilizes, i.e., the number of required calibration rounds. Small values reduce the convergence speed, while larger values generally converge faster. However, if the value is too large, the evolution to new configuration is not stable. Moreover, if we consider unreliable communication in which packets containing information about $W_{\texttt{next}}$ and $W_{\texttt{prev}}$ are lost, a smaller α value improves robustness.

Finally, it is worth noting that each calibration phase requires only a single, one-hop broadcast message from each node. Therefore, a calibration epoch can overlap with an epoch in which data is also exchanged.

3.2 Extensions for Tree-Based Communication

As noted in Section 2.2, a good wake-up configuration for multi-hop tree-based communication is one in which every node has either a sibling or a parent as its `next` in the wake-up sequence. Further, latency can be reduced by moving the wake-up time of the child close to the wake-up time of the parent. In this section we introduce two extensions to achieve these goals, namely *jumping* and *waving*.

"Jumping" Wake-up Times to Allow a Parent to Follow a Child. While ideally a parent should wake up after a child to forward data upstream as quickly as possible, the previously presented scattering algorithm never changes the sequence that nodes wake up inside an epoch. In other words, if a parent wakes up before its child, our scattering algorithm will never swap this order. Therefore, to reach a *better* configuration for tree-based communication, we introduce a new behavior in the scattering algorithm, namely the ability to *jump* over a neighbor w.r.t. wake-up time.

Specifically, if a node detects that its `next` node is one of its children, it sets its new wake-up time in the middle of the wake-up times of $W_{\texttt{next}}$ and $W_{\texttt{afternext}}$, where $W_{\texttt{afternext}}$ is the node to wake up immediately following `next`. To avoid unstable behaviors such as nodes continuously jumping back and forth, jumping is

applied only with a given probability β, a parameter of our algorithm. In experiments with $\beta = 0.6$, shown in Section 4, all topologies converged quickly to "tree-friendly" configurations. The time to convergence depends mainly on the depth of the tree. In fact, a faster convergence could be achieved by using a high value of β for nodes close to the sink, and decreasing it for nodes farther away. In this way, the nodes closer to the root stabilize their configuration quickly, allowing incremental stabilization of downstream nodes.

"Waving" Wake-up Times to Reduce Latency. While jumping generally retains both short intervals between wake-up times and good event detection, the latency for a message to move from the edge of the network to the sink is not fully optimized. In fact, the optimal wake-up configuration is achieved when the time between a node and its parent's wake-up time is fixed to the message passing time. Enforcing such a wake-up schedule is relatively trivial and results in a wake-up pattern in which nodes one hop from the sink wake up together, those two hops away wake up immediately before, and so on to the edge of the network. The result is a "wave" of wake-up times originating with nodes at the edges and ending at the sink. Any scattering away from this wave pattern negatively affects the optimal latency for a packet traveling to the sink.

To counteract this, we introduce controlled *waving*, i.e., setting wake-up times to reduce the interval between a node's wake-up time and $W_{\texttt{next}}$. However, our goal is not simply to form a tree, but also to retain the benefits of wake-up scattering, namely a short time to communication and a good event coverage. Such a compromise is reached by enforcing a maximum wake-up distance between a sensor and its `next`. This distance, γ, is another parameter of our protocol allowing the application to tune the trade-off between a good scattering and the latency to reach the sink.

3.3 Pseudocode

Figure 4 shows the full pseudocode for a *single* calibration round of our scattering algorithm, including the extensions for managing a tree. For convenience, we use the notation W_i^j to indicate the wake-up time of node i in calibration round j. Initially, $j = 0$ and the wake-up time for each node, W_i^0, is assumed to be random. The key parameters already mentioned are α, affecting the convergence speed, β, establishing the jumping probability, and γ determining the waving behavior. When $\beta = 0$, no jumping optimization is performed. The γ parameter has meaning only if jumping is on, i.e., $\beta \neq 0$. $\gamma = 0$ represents perfect synchronization of wake-up times; meaning all nodes wake up at the same time.

Notably, the first steps of the algorithm establish the values of $W_{\texttt{next}}$ and $W_{\texttt{prev}}$ by assuming that all nodes receive the full configuration R of wake-up times before the round starts. Next we consider an alternate approach that does not require synchronization among the nodes.

3.4 Removing Synchronization Requirements

In some cases, it may not be reasonable to assume that nodes share a clock and therefore that epochs are synchronized among the nodes. This makes a direct comparison among the wake-up times impossible, and demands an alternate mechanism to learn $W_{\texttt{prev}}$, $W_{\texttt{next}}$ and, if necessary, $W_{\texttt{afternext}}$.

```
// Exchange wake-up times with neighbors
Broadcast W_i^j
Receive R = {W_n^j : n is a neighbor of i}
if R is empty then exit                                  // i is alone

// Initialization of W_next and W_prev
W_first ⇐ min(R)
W_last ⇐ max(R)
if W_i^j > W_last then W_next ⇐ W_first + E               // i is the last
else W_next ⇐ min(W_n^j ∈ R : W_n^j > W_i^j)
if W_i^j < W_first then W_prev ⇐ W_last − E               // i is the first
else W_prev ⇐ max(W_n^j ∈ R : W_n^j < W_i^j)

// Scattering for next round
target ⇐ (W_prev + W_next)/2
W_i^{j+1} ⇐ (W_i^j · (1 − α) + target · α) mod E

// Extensions for the tree scenario
if i belongs to a tree and is not the root then
  // Waving
  if W_i^{j+1} < W_next − γ then W_i^{j+1} ⇐ W_next − γ
  // Jumping
  if next is a child then
    with probability (1 − β) exit                        // abort jumping
    if i is the last or second to last then
      W_afternext ⇐ min(W_n^j ∈ R : W_n^j + E > W_next) + E
    else W_afternext ⇐ min(W_n^j ∈ R : W_n^j > W_next)
    W_i^{j+1} ⇐ ((W_next + W_afternext)/2) mod E         // jump
```

Fig. 4. A single calibration round j of the wake-up scattering for node i

A simple solution is to gather these wake-up times *relative* to a nodes own epoch. This can be achieved by requiring each node to broadcast a message upon waking up, and having all nodes listen for two epochs. The latter constraint ensures that a node receives the wake-up time of nodes that were already awake when its epoch started. The algorithm in Figure 4 remains fundamentally unchanged after the wake-up times are collected. One additional advantage of this mechanism is that it reduces the probability of collisions among calibration messages, as they are sent at the wake-up times of the nodes and are naturally scattered.

Another practical concern is the initiation of the calibration round. While we have so far considered that all nodes are aware of the calibration at the same moment, it is trivial to extend the algorithm to allow a node to start a calibration round when it receives a calibration message from one of its neighbors. In this manner, knowledge of calibration will propagate to the whole system regardless of the node where it originates.

4 Evaluation

Our wake-up scattering algorithm potentially supports many applications. Our evaluation shows the achievable benefits for three, distinct scenarios as we vary the key parameters. Table 1 shows the most important parameters along with the default values used during simulation.

Table 1. Key simulation parameters and their default values. Those in italics remain constant throughout the evaluation.

Parameter	Value
Number of Nodes	200
Size of area	1000 x 660
Epoch period	1

Parameter	Value
Awake interval	0.25
Radio range	70
Sensing range	70

Parameter	Value
α, *scattering weight*	0.5
β, jumping probability	0
γ, maximum waving distance	–

The evaluation was performed with a custom Java simulator.[1] As previously mentioned, the simulator assumes a straightforward communication model where nodes have configurable, circular communication and sensing ranges. The results presented here assume messages are never lost, although we also performed experiments (not presented here for space reasons) showing that even with up to 10% loss, the system is still able to quickly converge to a good wake-up configuration.

Most of the reported results are calculated as averages over 10 different random topologies, each with 5 different initial wake-up configurations. Our baseline for comparison are the initial, random configurations.

Responsiveness to Local Queries. In our scenario with a mobile base station and many scattered sensors, the primary goal to minimize the time that the base station waits for a query response. This corresponds to the average response delay between when a base station first requests data, until a sensor in range is awake and sends the data. To measure this quantitatively, we use a Monte-Carlo sampling technique that randomly selects and point (x, y) and time t such that (x, y) is within communication range of at least one sensor and t is inside the epoch. The response delay is 0 if at least one sensor in radio range is awake at t, otherwise it is calculated as the difference between t and the closest wake-up time of an in-range sensor. Thousands of samples are taken and averaged to compute the overall average response delay.

One of the major factors affecting radio responsiveness is the duration of the awake interval for each of the nodes. The longer the nodes are active, the greater the chance that a node will be available to respond immediately. In our default setting with 200 nodes and radio range of 70, a node has on average 4.2 neighbors. Therefore, with a wake-up time near to $1/4$ of the epoch, we expect very short response delays, as it is likely that at least one node is awake when a query is made. However, as the awake interval shrinks, the probability of finding a gap between the query and the first wake-up time increases. Figure 5 shows the average response delay in both absolute (left) and relative (right) terms. The absolute response delay is in terms of the fraction of an epoch while the relative is in percentage improvement over the initial wake-up configuration. From the left side of the figure, it is important to notice that in all cases, our scattering approach converges in few configuration rounds, with most improvements taking place in the first three rounds.

Figure 5 also evidences that our wake-up scattering approach can be used to dramatically improve the lifetime of the network. Consider that a randomly initialized network

[1] An online demo is available to test and visualize the effect of wake-up scattering on custom configurations: `http://www.elet.polimi.it/upload/giusti/scattering/`

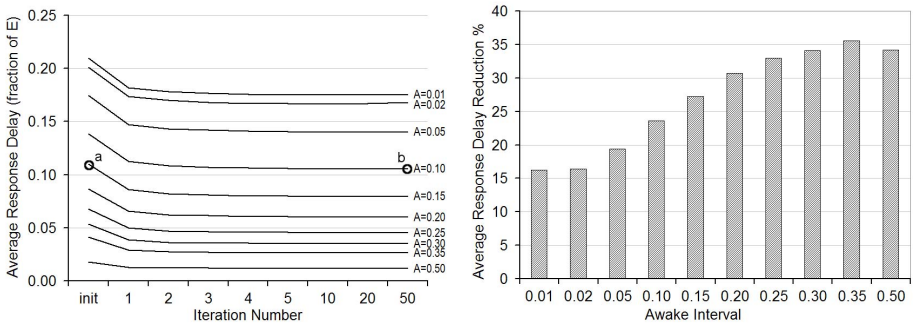


Fig. 5. Response delay with various awake intervals

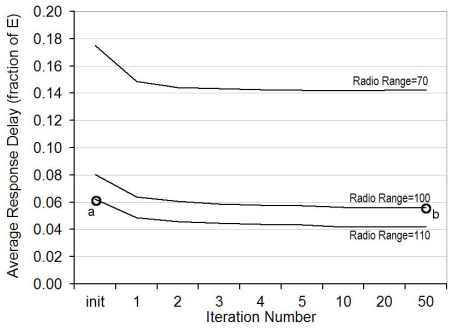


Fig. 6. Response delay with various communication ranges. The three lines correspond to networks with on average 4.3, 8.4 and 10.1 neighbors.

with an awake interval of 0.15 has an initial average response delay of 0.110, labeled a in the figure. A similar response delay of 0.106, b in the figure, can be achieved after wake-up scattering with a shorter awake interval of 0.10. This means that by applying wake-up scattering, the awake interval can be decreased by 33% while the application performance is preserved at what can, on average, be achieved with a random initial configuration. Similar significant results can be observed by comparing the initial response delay with a given awake interval compared to similar response delays achieved after scattering, but with shorter awake intervals.

We also analyzed our approach for different network densities, observing from Figure 6 that with less dense networks, the percentage improvement after scattering tends to be larger. Also, a similar lifetime argument can be made by observing that with a radio range of 100, after scattering, b in the figure, the response delay achieved is that of randomly initialized network with a range of 110, a in the figure. Because higher transmission ranges require more energy, again, wake-up scattering can be exploited to improve network lifetime.

Event Coverage. When applying wake-up scattering to sensor activation, as opposed to communication activation, we consider the probability that an event occurring in the

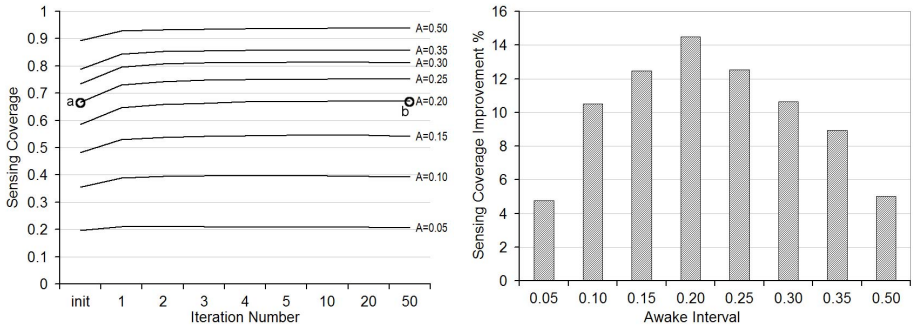


Fig. 7. Sensing coverage for various awake intervals

sensed region is detected. To measure this, we use a sampling technique similar to that used for response delay, selecting a point (x, y) that is covered by at least one sensor, and a time in the epoch t. If a sensor covering this point is active at time t, then the event is detected, otherwise it is not. As before, several thousand samples are taken and the results averaged. It should be noted that perfect event detection is unlikely to be attained, except with contrived topologies. Consider a simple setting with two sensors, each awake half of the epoch, and located just within communication range. Even with perfect scattering, only the events located in the overlapping area will be reliably detected. Instead, those in regions covered only by one sensor will be detected half the time, leading to an average event coverage much less than 1.

In contrast to the scattering for response delay discussed previously, event coverage scenarios are heavily dependent on the length of the awake intervals. With response delay, if a node is inactive when a query is made, the base station simply waits. Instead, if an event occurs and no sensor is active, the event is not detected. Therefore, in Figure 7 we show the effect of awake intervals on the sensing coverage. Although the percentage improvement in the probability of detecting an event is small, less than 15%, we can make similar network lifetime arguments as before by comparing the achieved coverage of a network with an awake interval of 0.20 (0.671, b in the figure) in comparison to the initial coverage of a network with 20% more awake time (0.667, a in the figure). In other words, the lifetime of the network can be increased by 20% while achieving the same coverage as a random initialization with a longer awake interval.

Another important consideration is the relationship between the communication radius and the sensing range. Our wake-up scattering is based only on the ability of two nodes to communicate with one another, ignoring any difference between sensing and communication ranges, however such differences are possible with real sensing devices. Figure 8 shows that if the communication radius remains fixed while the sensing range varies, wake-up scattering still achieves gains.

Data Latency in Tree-Based Networks. When considering a tree, the primary factor to improve is the time it takes a message to reach the collection point, the root. Therefore, we consider the time-to-root averaged over all sensors, assuming the message is sent just

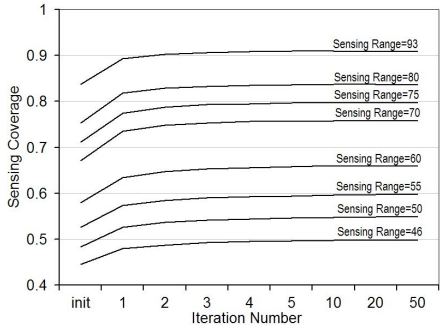


Fig. 8. Effect of sensing range on wake-up scattering

before the end of the sender's awake interval. If the parent is not immediately awake, the sending node wakes up outside of its normal awake interval to forward the message.

The top of Figure 9 gives a first impression of the effect of wake-up scattering on time-to-root, and the tradeoffs introduced with jumping and waving. The chart shows how time-to-root changes from the initial random configuration (labeled init), after the basic wake-up scattering stabilizes (scattering), after the application and stabilization of jumping (jumping), and after applying waving with various γ parameters. It should be noted that this plot represents the *independent* effects of the scattering, jumping, and waving techniques, not a progression of time-to-root values over time.

The first observation is that time-to-root is only marginally worse after scattering. This is because, except with long awake times, scattering removes any overlap of awake intervals between parents and children that occurred in the random initialization. Second, the addition of jumping significantly improves time-to-root, reducing the latency by approximately one third of the total time. Finally, as expected, waving further reduces time-to-root with smaller values of the waving factor, γ, leading to greater improvements because of the reduction of the required gap between the wake-up times of the child and parent.

While the previous plot shows that both jumping and waving have significant, positive impacts on time-to-root, it is reasonable to consider combining the tree scenario with event coverage, for example to send notifications of detected events to a centralized sink. Therefore, the bottom of Figure 9 shows the effect of jumping and waving on the coverage achieved with scattering alone. Most importantly this plot shows that jumping does not significantly affect sensing coverage, however waving does. This is to be expected because jumping maintains the scattering properties while simply changing the order of awake intervals, instead waving causes the wake-up intervals to overlap. Such overlapping intervals lead to an increase in missed events because the nodes tend to be active for event detection at the same times when γ is small. Put another way, when a sensor node is used for event detection where the results are sent to a centralized root, wake-up scattering and jumping provide a solution that has good coverage and time-to-root values. To our knowledge, our work is the first to consider such a combination.

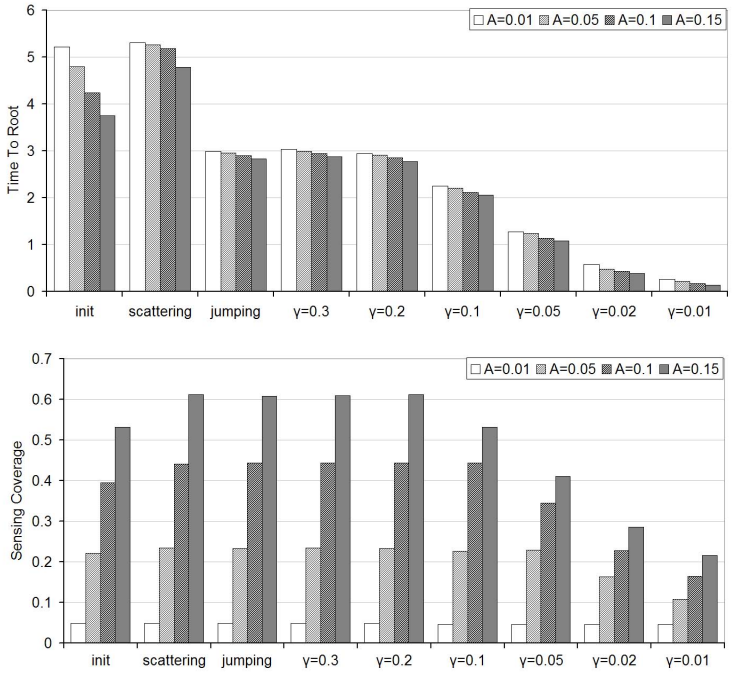


Fig. 9. Average time-to-root and sensing coverage for various awake intervals after enabling the various scattering, jumping, and waving phases of the algorithm. The results are averaged over 50 random topologies, each tested with a single initialization of wake-up times.

We ran similar experiments to evaluate the effects of jumping and waving on response delays in the first scenario. The results are similar to those for coverage, showing that jumping has minimal effect, while waving has negative effects.

5 Related Work

To the best of our knowledge, our approach to scatter the wake-up times of nearby sensors is novel. Nevertheless, it is related to several other directions in the literature, investigated here from the bottom (MAC) up (applications).

Many MAC layer protocols explicitly introduce their own duty cycling as a means to save energy, thus on the surface the two approaches are similar. However, it is important to note that the duration of duty cycles at the MAC-layer and those we consider are orders of magnitude different.

As our approach focuses on assigning wake-up times to nodes, it is interesting to compare the wake-up scattering assignments to those of MAC protocols. For example, SMAC [3] and TMAC [4] attempt to synchronize all nodes to a single awake interval, allowing communication during a short interval, then putting the whole network to sleep for an extended time. While this has the same lifetime extension potential as our approach, it cannot achieve the benefits for average response delay or sensor coverage. In contrast to the whole-network synchronization of SMAC and TMAC, LMAC [5] and

ZMAC [6] are TDMA-like approaches that divide and assign a set of communication slots among all nodes, eliminating overlap as much as possible. While this is related to wake-up scattering as the assigned slots are spread over time, these systems work with an assumption of a fixed number of time slots, eliminating all possible overlapping. Wake-up scattering, on the other hand, scatters as much as possible without discrete, fixed time intervals, allowing overlap if the awake interval is long, but still spreading out over the full epoch even if awake intervals are short.

Finally, DMAC [7] is designed to allow efficient communication along a tree, termed convergecast communication. It performs synchronization similar to SMAC/TMAC, then staggers the wake-up such that messages are forwarded without delay. This spreading of wake-up intervals is analogous to our waving approach that moves wake-up intervals closer together, similarly reducing latency. However, as observed in Section 4, placing wake-up times close together negatively affects all benefits to coverage that can be achieved with scattering.

Above the MAC layer, extensive work exists on managing sensor coverage by intentionally setting the physical sensor locations, as opposed to our approach which deals with temporal coverage. In fact, this field has been studied since the early 1990's in the context of the cellular network. More recent work approaches the same issues in WSNs [8] with a variety of techniques such as Integer Programming [9], greedy heuristics [10,11,12] and Virtual Force Methods [13]. Interestingly, [14] studies sensor placement in combination with varying the ratio of sensing and communication range, supporting our desire to present the evaluation of the same ratio when studying sensing coverage. These spatial techniques can be applied in parallel to our approach, however, it is worth noting that our study measured only the coverage that could be achieved with random placement, ignoring any areas of the field not covered by the sensors.

6 Conclusions

In this paper we presented and evaluated a fully decentralized wake-up scattering algorithm whose goal is to spread uniformly the wake-up times of the nodes in a WSN. We illustrated common application scenarios where this functionality is beneficial, and verified through simulation that indeed our algorithm provides improvements in the WSN performance and lifetime over a random assignment of wake-up times. The algorithm is very simple, and therefore not only is easily implementable on resource-scarce WSN devices, but it also introduces a negligible communication and computational overhead.

Acknowledgments. This work is partially supported by the European Union under the IST-004536 RUNES project and by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322.

References

1. Curino, C., Giani, M., Giorgetta, M., Giusti, A., Murphy, A.L., Picco, G.P.: Mobile data collection in sensor networks: The TinyLIME Middleware. Elsevier Pervasive and Mobile Computing Journal **4**(1) (2005) 446–469

2. Madden, S., M.J. Franklin, J.M. Hellerstein, Hong, W.: TinyDB: An acquisitional query processing system for sensor networks. ACM Trans. Database Syst. **30**(1) (2005) 122–173
3. Ye, W., Heidemann, J., Estrin, D.: An energy-efficient MAC protocol for wireless sensor networks. In: Proceedings of the 21^{st} IEEE INFOCOM. (2002) 1567–1576
4. van Dam, T., Langendoen, K.: An adaptive energy-efficient MAC protocol for wireless sensor networks. In: Proc. of the 1^{st} ACM Conf. on Embedded Networked Sensor Systems (SenSys), Los Angeles, CA, USA (2003) 171–180
5. van Hoesel, L., Havinga, P.: A lightweight medium access protocol (LMAC) for wireless sensor networks. In: Proc. of 1^{st} Int. Wkshp. on Networked Sensing Systems, Tokyo, Japan (2004)
6. Rhee, I., Warrier, A., Aia, M., Min, J.: Z-MAC: A hybrid MAC for wireless sensor networks. In: Proc. of the 3^{rd} ACM Conf. on Embedded Networked Sensor Systems (SenSys), San Diego, CA, USA (2005) 90–101
7. Lu, G., Krishnamachari, B., Raghavendra, C.S.: An adaptive energy-efficient and low-latency MAC for data gathering in sensor networks. In: Proc. of the Int. Wkshp. on Algorithms for Wireless, Mobile, Ad Hoc and Sensor Networks (WMAN), Santa Fe, NM, USA (2004)
8. Meguerdichian, S., Koushanfar, F., Potkonjak, M., Srivastava, M.: Coverage problems in wireless ad-hoc sensor networks. In: Proc. of 20^{th} IEEE INFOCOM, Anchorage, Alaska, USA (2001) 1380–1387
9. Chakrabarty, K., Iyengar, S.S., Qi, H., Cho, E.: Grid coverage for surveillance and target location in distributed sensor networks. IEEE Trans. on Computers **51**(12) (2002) 1448–1453
10. Bulusu, N., Estrin, D., Heidemann, J.: Adaptive beacon placement. In: Proc. of the 21^{st} Int. Conf. on Distributed Computing Systems (ICDCS), Phoenix, AZ, USA (2001) 489–498
11. Howard, A., Mataric, M.J., Sukhatme, G.S.: An incremental self-deployment algorithm for mobile sensor networks. Auton. Robots **13**(2) (2002) 113–126
12. Howard, A., Mataric, M.J., Sukhatme, G.S.: Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem. In: Proc. of the 7^{th} Int. Symp. on Distributed Autonomous Robotic Systems (DARS). (2002)
13. Zou, Y., Chakrabarty, K.: Sensor deployment and target localization in distributed sensor networks. Trans. on Embedded Computing Sys. **3**(1) (2004) 61–91
14. Jourdan, D.B., de Weck, O.L.: Layout optimization for a wireless sensor network using a multi-objective genetic algorithm. In: Proc. of the Vehicular Technology Conf., Los Angeles, CA, USA (2004) 2466–2470