

# Implementing Energetically Sustainable Routing Algorithms for Autonomous WSNs

Lorenz Cuno Klopfenstein

Emanuele Lattanzi

Alessandro Bogliolo

Information Science and Technology Institute

University of Urbino, 61029 Urbino, Italy

{cuno.klopfenstein, emanuele.lattanzi, alessandro.bogliolo}@uniurb.it

## Abstract

*Energy harvesting technologies make it possible to exploit environmental power to realize autonomous wireless sensor networks with unlimited lifetime. However, the actual autonomy of a wireless network depends on the energetic sustainability of its workload that depends, on its turn, on the algorithms adopted to route packets from the sensor nodes to the sink. A theoretical framework for assessing and optimizing the energetic sustainability of routing algorithms has been recently proposed. In this paper we discuss the actual applicability of optimal energetically sustainable routing algorithms and the issues raised by non-stationary environmental conditions. Furthermore, we present a distributed algorithm, running on sensor nodes, that is able to re-compute optimal routing tables on the field in order to adapt to the actual distribution of environmental power.*

## 1 Introduction

Energy-aware wireless sensor networks (WSNs) are usually designed and deployed in order to maximize battery lifetime [8]. Lifetime, however, is not a suitable metric for steering the design of *environmentally-powered* WSNs, capable of working as long as their workload can be sustained by the power they are able to harvest from the environment [1, 9]. In this new scenario, the key design goal should be maximizing the steady-state energetic sustainability of the network's workload, rather than maximizing its lifetime. In fact, while battery-operated WSNs are subject to energy constraints imposed by the limited capacity of their batteries, autonomous WSNs are subject to power constraints imposed by the limited amount of power provided by the environment.

It has been recently shown that power-constrained WSNs can be represented as flow networks and that the optimization of the energetic sustainability of the workload can be

cast into an instance of maxflow [2]. The solution of the maxflow problem induces non-deterministic routing tables that can be actually applied at the sensor nodes in order to achieve the theoretical optimum.

However, the optimal routing strategy strongly depends on the actual environmental conditions and should be periodically recomputed. Since the sensor nodes are deployed in the environment they take supply power from, they are the best candidates for detecting environmental power changes and possibly adapt their own routing tables. But the information available at each node is not sufficient to take local decisions leading to a global optimum. On the other hand, the whole sensor network could be viewed as a distributed computer system that can be programmed to solve maxflow, thus computing optimal routing strategies on the field.

In this paper we demonstrate the feasibility of this approach, we outline the details of a real-world implementation, we present preliminary results and we discuss practical issues and future directions.

## 2 Theoretical framework

The workload of a sensor network can be expressed as the average number of packets routed from the sensors to the sink in a unit of time. The maximum workload that can be sustained by a node of an autonomous WSN (say  $n$ ) depends on the available power at the node  $P(n)$ , and on the energy needed to process and send each packet across an edge,  $E(n, e)$ , where  $e$  denotes the chosen outgoing edge. In general, processing includes packet generation (is the node is the source of the packet) or packet reception (is the node acts as a routing). Both packet generation and reception require a non-negligible amount of energy. In particular, the energy spent by a sensor node to receive a packet from another node is often higher than the energy spent to re-transmit it.

Whenever a packet is processed, a finite amount of

```

PushRelabel(G,s,t)
1 initialize preflow (G,s)
2 while (there is an applicable operation)
3   perform an applicable operation
4 return overflow(t)

```

(a)

```

RouterPushRelabel()
1 if overflow
2   if (descending edge available)
3     push flow through the edge
4   else
5     height = relabel()
6 return

```

(c)

```

MESW(G,P)
1 add global source (s) and global sink (t)
2 derive node constraints from P
3 from node constraints to edge constraints
4 PushRelabel(G,s,t)
5 return overflow(t)

```

(b)

```

SensorPushRelabel()
1 if (height > N)
2   drop overflow
3 else
4   RouterPushRelabel()
5 return

```

(d)

**Figure 1.** Pseudo-code of (a) PushRelabel and (b) MESW algorithm. Local operations performed by (c) the routers and (d) the sensors of a WSN in a parallel distributed implementation of PushRelabel.

time is needed to recover the energy spent to process and transmit it. The *recovery time*, expressed as the ratio between  $E(n, e)$  and  $P(n)$ , imposes an upper bound to the average inter-arrival time of the packets that compose an energetically-sustainable workload. Such an upper bound can be viewed as a capacity constraint imposed to the outgoing edge:

$$C(e) = P(n)/E(n, e)$$

This simple observation suggests that environmentally-powered WSNs can be modelled as flow networks whose maxflow is the *maximum energetically-sustainable workload* (MESW). However, each sensor node may have multiple outgoing edges sharing the same power budget, so that capacity constraints cannot be independently associated with edges, as in standard flow networks, but must be associated with nodes. The local sustainability constraint must thus be expressed as:

$$\sum_{e \text{ exiting from } n} F(e)E(n, e) \leq P(n)$$

where  $F(e)$  is the flow (i.e., the packet rate) across edge  $e$ . If the transmission power is not dynamically adapted to the actual distance of the target node, the energy per packet [10] can be considered as independent from the edge of choice. This simplifying assumption, that holds for many real-world networks, allows us to transform a node-constrained flow network into an equivalent edge-constrained flow network where each original node is split into an input node (destination of all incoming edges) and an output node (source of all outgoing edges), connected by an internal edge with capacity  $P(n)/E(n)$ . Hence, all the algorithms developed for studying and optimizing edge-constrained flow networks can be suitably applied to power constrained WSNs. A framework for determining the MESW of an arbitrary WSN based on a modified version of

*Ford Fulkerson's* MaxFlow algorithm [5] (FF from now on) was recently proposed [2].

### 3 Implementation

Given a WSN and a power map (representing environmental power conditions), a non deterministic routing algorithm that achieves the MESW can be easily derived from the solution of the maxflow problem: each edge can be chosen by the routing node with a probability proportional to the amount of flow crossing it in the maxflow solution. Edge probabilities are stored as routing tables at each node and used at runtime to take pseudo-random decisions.

Unfortunately, however, power conditions may vary in time, making it necessary to recompute the routing tables on the field. Ideally, sensor nodes should be able to adapt their own routing tables to environmental changes by implementing a distributed version of maxflow algorithm.

Generally speaking, a graph-traversal algorithm can be implemented by a network of elementary processing units having the same topology of the graph under study. Each unit directly implements the local operations of the corresponding node, while function calls and returns are implemented as messages exchanged through the edges of the graph. As such, a distributed version of FF algorithm could be directly implemented on the sensor network by exploiting the computational and communication resources of the nodes. However, FF is not the best choice to this purpose since it is inherently sequential in nature, so that its distributed implementation would not exploit parallelism and would require a significant synchronization overhead. Moreover, FF is not the most efficient algorithm available to solve MaxFlow, so that its run-time execution would require too much time and energy.

### 3.1 The Push-Relabel Algorithm

The above-mentioned drawbacks are overcome by the *Push-Relabel* (PR) algorithm [3], that exploits a fluid-flow analogy by modelling network edges as pipes and network nodes as junctions among them. Each junction has a *height* attribute, reflecting its virtual elevation in relation to the other nodes, and an arbitrarily large *reservoir*, which can temporarily accommodate excess flow.

As its name implies, the algorithm makes use of two basic operations:

- **push**: applied when a node has excess flow in its reservoir and a non-saturated *downhill* edge (i.e., an edge with non-null residual capacity that leads to a lower node) to push (part of) the excess flow across the edge. The result is a transfer of excess flow from a node to another.
- **relabel**: applied when a node has excess flow but no available downhill edges to push it across. The height of the overflowing node is set just above the lowest of its neighbor nodes connected by a non-saturated edge.

The generic PR algorithm uses a subroutine to generate an initial *preflow* (actually saturating all outgoing edges of the main source node) and to set the height of all nodes to 0 (except the source, whose height is set to the number of nodes  $N$ ). The algorithm then proceeds by applying push and relabel operations until no more operations are allowed (see Fig. 1.a). All possible flow is eventually pushed either to the sink or back to the source. The algorithm terminates when no nodes are left overflowing, except the sink and - possibly - the source. The MaxFlow corresponds to the excess flow found in the sink's reservoir at the end of execution.

Notice that MaxFlow algorithms determine the maximum flow from a single source to a single sink. In a WSN there are usually many sensors acting as primary sources and, possibly, many sinks that collect sensed data. In order to use MaxFlow to determine the global MESW, dummy nodes have to be added to the network: a main source (say  $s$ ) with infinite capacity edges leading to all sensors, and a main sink (say  $t$ ) with infinite capacity edges coming from all sinks. The pseudo-code of the MESW algorithm based on PR is shown in Fig. 1.b.

### 3.2 Distributed implementation

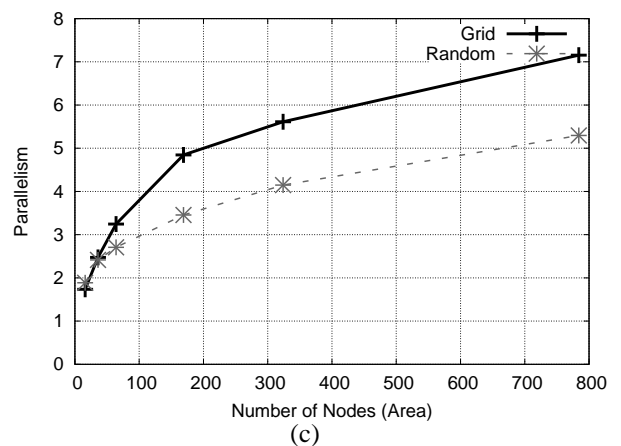
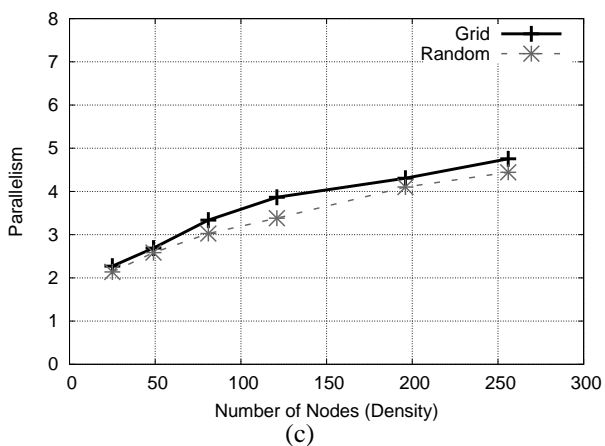
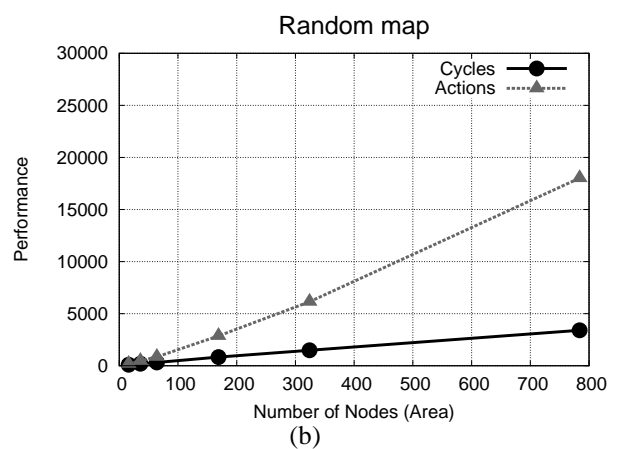
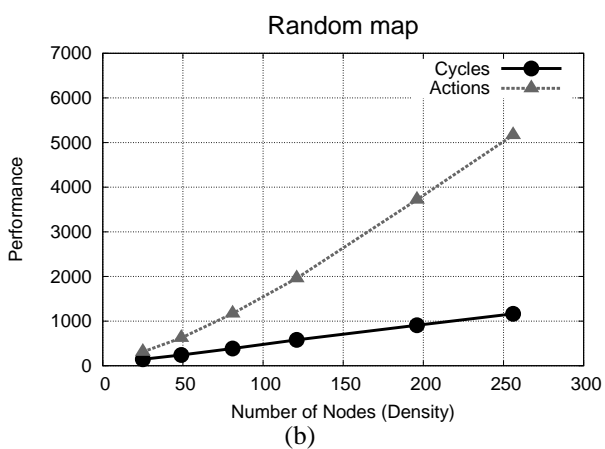
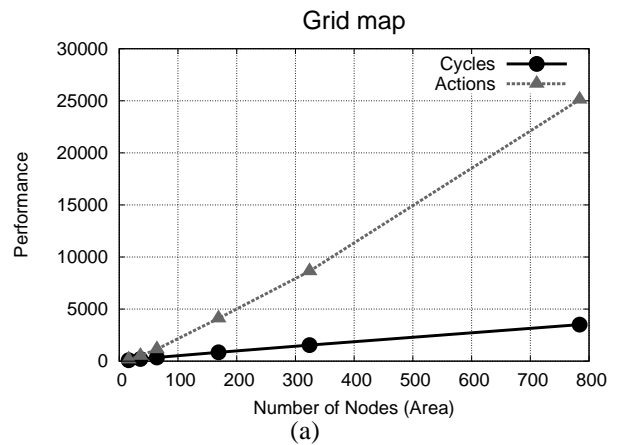
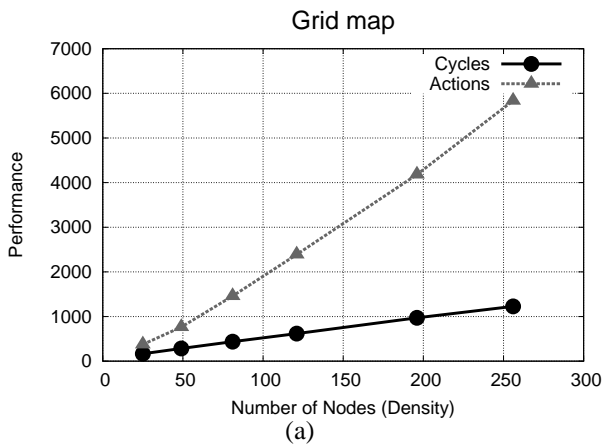
The MESW algorithm of Fig. 1.b can be conveniently implemented in a distributed network of computational units for two main reasons. First, the operations performed at each node require only local information about the state

of the node and the heights of its neighbors. Second, the correctness of the solution is independent of the order of the push-relabel operations performed at each node, thus enabling concurrency with limited synchronization requirements.

The concurrent implementation requires, however, several modifications. First, dummy sensor nodes are not real nodes, so that they need to be implicitly modelled by the sensor network. All sensors initialize their own reservoirs at an initial value conservatively higher than the maximum possible flow. Such an initial value replaces the incoming flow pushed by the dummy primary input node into the sources in the sequential implementation. Push back of excess flow from sensor nodes to the dummy is implicitly implemented by allowing sensors higher than  $N$  to drop their excess flow. Second, each node must be able to detect the amount of environmental power available and to derive numerical capacity constraints from it, applying it to an inner edge ideally connecting the node's incoming edges to the outgoing edges (and thus limiting the amount of flow that can be pushed). Third, push operations result in the exchange of a unicast message between the involved nodes. Fourth, relabel operations entail the transmission of a broadcast message to notify all neighbors of the changed height. Fifth, a broadcast interest message needs to be generated by the sink and back-propagated by all nodes in order to trigger the beginning of execution. Sixth, the flow pushed across each edge has to be stored in the local routing tables of each node.

Fig. 1.c shows the code executed by each routing node upon incoming events (either interest packets or pushed flow notifications). Figure 1.d shows the variant of the code implemented by sensor nodes. The only difference is the implicit push back of the excess flow to the dummy node when the height of the sensor exceeds the total number of nodes  $N$ .

Notice that each node decides which operation to perform based on the information locally available at decision time, but the decision takes a finite amount of time to be notified to its neighbors. Hence, neighboring nodes could concurrently take conflicting decisions. Consider, for instance, two overflowing nodes with the same height (say  $h$ ). Both of them could decide at the same time to perform a relabel (reaching height  $h + 1$ ) and to push the excess flow to the other node. Thanks to the order-independent correctness of the PR algorithm, concurrent decisions do not impair algorithm convergence. In our example, each node will accept the excess flow coming from its neighbor and eventually push it towards lower nodes or raise its height.



**Figure 2.** Computational complexity (Actions) and execution time (Cycles) of the concurrent PR algorithm as function of the density of sensor nodes, expressed as number of sensors distributed over a fixed-size 10x10 area, assuming each antenna has a transmission range of 3 units and that all packets have to be routed to a common sink placed on a corner of the target region. (a) Sensor nodes arranged on a regular grid. (b) Sensor nodes scattered randomly. (c) Degree of parallelism of the algorithm in terms of number of actions executed in each cycle.

**Figure 3.** Computational complexity (Actions) and execution time (Cycles) of the concurrent PR algorithm as function of the area covered by the sensor network, expressed as number of sensors deployed at a fixed density over a square region of increasing area with a unique sink placed on a corner. Each antenna is assumed to cover a range of 3 units. (a) Sensor nodes arranged on a regular grid. (b) Sensor nodes scattered randomly. (c) Degree of parallelism of the algorithm in terms of number of actions executed in each cycle.

## 4 Experimental results and discussion

The concurrent MESW algorithm has been implemented on the TinyOS framework using the nesC programming language [6]. TinyOS applications can be natively compiled and run on real sensor networks (e.g., using Tmote Sky nodes [4]), as well as analyzed using the TOSSIM simulator [7], which enables network profiling while ensuring correctness, accuracy and realism.

The functional correctness of the concurrent implementation has been tested by comparing the results with those provided by the optimization and analysis framework based on FF [2], while computational performance has been tested by monitoring the number of push-relabel operations required and the degree of concurrency reached by the algorithm.

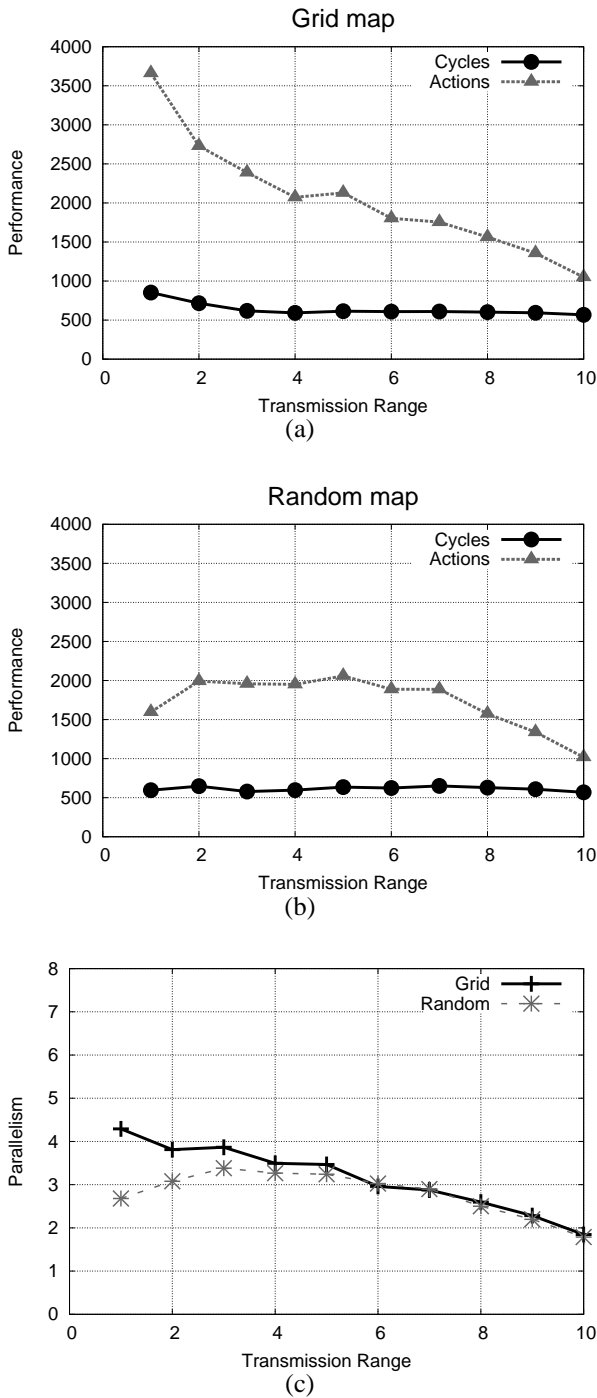
Experimental results are reported in Figures 2, 3 and 4, that show the computational complexity and execution time as functions of the density of sensor nodes (Fig. 2), of the area of the target region (Fig. 3) and of the transmission range of each antenna (Fig. 4). All the experiments refer to networks of sensors distributed over a square region with only one sink placed in a corner.

Execution time is shown by the curves labeled “cycles”, while computational complexity (i.e., the number of push and relabel actions required) is shown by the curves labeled “actions”. Thanks to the concurrent implementation, the algorithm is able to perform multiple actions per cycle, attaining higher performance than the sequential PR.

Each figure contains three charts, showing: (a) the performance of the algorithm when applied to a network of sensors regularly distributed on a grid; (b) the performance of the algorithm when applied to a network of sensors scattered randomly over the target region, and (c) the degree of parallelism computed as the ratio between computational complexity and computation time.

The results clearly show that the performance of the algorithm scales nicely with network complexity (Figures 2.a/b and 3.a/b): the computational complexity is slightly super-linear, while the execution time is linear. The benefit of concurrence is shown in Figures 2.a/b and 3.a/b by the gap between computational complexity and execution time, and in Figures 2.c and 3.c by the degree of parallelism achieved by the algorithm. It is worth noting that parallelism grows with the number of nodes. The increased parallelism compensates the superlinear behavior of computational complexity, thus making execution time proportional to the number of nodes in the network.

Finally, Figure 4 (referring to a network of 121 sensors on a 10x10 area) shows that computational complexity decrease when the transmission range of each antenna increases. In fact, the longer the range of each antenna, the easier is routing packets from the sensors to the sink.



**Figure 4.** Computational complexity (Actions) and execution time (Cycles) of the concurrent PR algorithm as function of the transmission range of the antennas, expressed in units of length. Results refer to a network composed of 121 sensors deployed over a square 10x10 region with a sink placed on a corner. (a) Sensor nodes arranged on a regular grid. (b) Sensor nodes scattered randomly. (c) Degree of parallelism of the algorithm in terms of number of actions executed in each cycle.

This benefit is reduced in case of randomly scattered nodes. Interestingly, the degree of parallelism of the algorithm decreases when the range of the antennas increases (Fig. 4.c), so that the execution time of the concurrent implementation is almost independent of the range.

## 5 Discussion

In summary, this paper has shown that autonomous WSNs can be granted the capability of self-adapting their routing strategy to the environment in order to achieve the maximum energetically-sustainable workload (MESW). This can be done by casting MESW into an instance of MaxFlow and by implementing a parallel distributed version of the PushRelabel algorithm, running directly on the sensor nodes, to determine optimal routing tables. The distributed algorithm not only grants the desired self-adapting property to the sensor network, but also exploits parallelism to reduce significantly the execution time.

There are, however, several issues left open:

1. Node constraints cannot be transformed in edge constraints in case of edge-dependent transmission energy;
2. Routing tables are not continuously adapted to the environmental conditions;
3. MESW recomputation takes energy and time that could be spent by the network to perform normal operations, thus making it necessary to trade off routing optimality for recomputation frequency.

Our current research efforts are aimed at addressing these issues in the attempt of making WSNs able to transparently adapt to the environmental power during their normal operation.

## References

- [1] R. Amirtharajah, J. Collier, J. Siebert, B. Zhou, and A. Chandrakasan. Dsps for energy harvesting sensors: applications and architectures. *IEEE Pervasive Computing*, 4(3):72–79, 2005.
- [2] A. Bogliolo, E. Lattanzi, and A. Acquaviva. Energetic sustainability of environmentally powered wireless sensor networks. In *PE-WASUN '06: Proceedings of the 3rd ACM international workshop on Performance evaluation of wireless ad hoc, sensor and ubiquitous networks*, pages 149–152, 2006.
- [3] B. V. Cherkassky. On implementing the push-relabel method for the maximum flow problem. *Algorithmica*, 19(4):390–410, 1997.
- [4] M. Corporation. *Tmote sky - ultra low power IEEE 802.15.4 compliant wireless sensor module*. available at <http://www.moteiv.com/>, 2006.
- [5] L. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.
- [6] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesc language: A holistic approach to networked embedded systems. In *PLDI-03: Proc. of Programming Language Design and Implementation*, 2003.
- [7] P. Levis and N. Lee. *TOSSIM: A Simulator for TinyOS Networks*. Included in TinyOS distribution, 2003.
- [8] V. Mhatre and C. Rosenberg. Energy and cost optimizations in wireless sensor networks: A survey. In A. Girard, B. Sanso, and F. Vazquez-Abad, editors, *Performance Evaluation and Planning Methods for the Next Generation Internet*. Kluwer Academic Publishers, 2005.
- [9] V. Raghunathan, A. Kansal, J. Hsu, J. Friedman, and M. Srivastava. Design considerations for solar energy harvesting wireless embedded systems. In *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*, page 64, Piscataway, NJ, USA, 2005. IEEE Press.
- [10] A. Y. Wang and C. G. Sodini. On the energy efficiency of wireless transceivers. In *Proc. of IEEE Conference on Communications*, pages 3783–3788, 2006.