Algorithms and Simulation Methods for Topology-Aware Sensor Networks

Alexander Kröller[1], Dennis Pfisterer[2], Sándor P. Fekete[1], and Stefan Fischer[2]

[1] Algorithms Group
Braunschweig Institute of Technology
{kroeller,fekete}@tu-bs.de
http://www.ibr.cs.tu-bs.de/alg/
[2] Institute of Telematics
University of Lübeck, Germany
{pfisterer,fischer}@itm.uni-luebeck.de
http://www.itm.uni-luebeck.de

Abstract. This chapter presents a number of different aspects related to a particular kind of large and complex networks: A Wireless Sensor Network (WSN) consists of a large number of nodes that individually have limited computing power and information; their interaction is strictly local, but their task is to build global structures and pursue global objectives.

Dealing with WSNs requires a mixture of theory and practice, i.e., a combination of algorithmic foundations with simulations and experiments that has been the subject of our project SwarmNet. In the first part, we describe a number of fundamental algorithmic issues: boundary recognition without node coordinates, clustering, routing, and energy-constrained flows. The second part deals with the simulation of large-scale WSNs; we describe the most important challenges and how they can be tackled with our network simulator Shawn.

1 Introduction

Our modern world has grown immensely complex, much beyond what any group of human experts could have designed. A major part in this is played by *decentralized networks*, which form a robust yet dynamic mechanism that is able to handle a large variety of challenges. In the following we give a brief overview of a particular type of decentralized network, as well as the corresponding algorithmic and simulation challenges.

1.1 Decentralized Networks

Traditional computing systems are based on a centralized algorithmic paradigm: Data is gathered, processed, and the result administered by one central authority. On the other hand, in today's communication world, wireless networks such as the mobile phone network GSM in the wide area and WiFi in the local area

J. Lerner, D. Wagner, and K.A. Zweig (Eds.): Algorithmics, LNCS 5515, pp. 380–400, 2009.

range have become ubiquitous. So far, most applications using these networks rely on a thoroughly managed infrastructure, e.g., base stations in GSM or access points in WiFi. Many research activities, however, already go one step further and make use of the fact that more and more mobile devices with radio communication capabilities are available. These devices are not necessarily bound to communication infrastructures, but can instead create a spontaneous network, a so-called ad-hoc network, in order to allow communication among all participating nodes (and not necessarily to the outside). In its sophisticated form, some of the nodes in an ad-hoc network act as relay stations and transport data from a source to a destination that is not in direct radio range (multihop ad-hoc network).

Based on the ad-hoc kind of network setup, another new kind of network is currently in the focus of many research activities: the *sensor network*. Its purpose is the recording, transport and interpretation of phenomena of the environment, such as measuring temperatures, monitoring areas, etc. Many people claim that this new technology is so substantially different from today's networks that from the scientific point of view a paradigm shift for design, implementation, deployment and management of such networks and their nodes is necessary. A number of facts support this view: Sensor networks usually consist of a large number of nodes, while their communication mode is usually ad hoc without infrastructure. It may be necessary that after potentially uncontrolled deployment (e.g., throwing nodes out of a plane), nodes have to organize themselves into a network without the help of an administrator. During the network's lifetime, nodes might move, run out of energy or may even be destroyed. Sensor nodes have very limited computing and storage capacities; on the other hand, algorithms have to be devised with limited energy supply in mind. Finally, sensor networks usually provide enormous amounts of data. In order to get meaningful results from their operation, powerful data aggregation and interpretation facilities have to be created.

1.2 Algorithmic Challenges

From an algorithmic point of view, the characteristics of a sensor network require working under a paradigm that is different from classical models of computation: absence of a central control unit, limited capabilities of nodes, and limited communication between nodes require developing new algorithmic ideas that combine methods of distributed computing and network protocols with traditional centralized network algorithms. In other words: How can we use a limited amount of strictly local information in order to achieve distributed knowledge of global network properties?

This task is much simpler if the exact location of each node is known. Computing node coordinates has received a considerable amount of attention. Unfortunately, computing exact coordinates requires the use of special location hardware like GPS, or alternatively, scanning devices, imposing physical demands on size and structure of sensor nodes. As we demonstrated in our paper [1], current methods for computing coordinates based on anchor points and distance

estimates encounter serious difficulties in the presence of even small inaccuracies, which are unavoidable in practice.

Our work shows that even in the absence of coordinates, there is some underlying geometric information that can be exploited. As we discuss in Section 2, boundary recognition without node coordinates, clustering, routing, and energy-constrained flows allow algorithmic solutions.

1.3 Simulation Challenges

To acquire a deeper understanding of these networks, three fundamentally different approaches exist: Analytical methods, computer simulation, and physical experiments. Designing algorithms for sensor networks can be inherently complex. Many aspects such as energy efficiency, limited resources, decentralized collaboration, fault tolerance, and the study of the global behavior emerging from local interactions have to be tackled.

In principle, experimenting with actual sensor networks would be a good way of demonstrating that a system is able to achieve certain objectives, even under real-world conditions. However, this approach poses a number of practical difficulties. First of all, it is difficult to operate and debug such systems. This may have contributed to the fact that only very few of these networks have yet been deployed [2, 3, 4]. Real-world systems typically consist of roughly a few dozen sensor nodes, whereas future scenarios anticipate networks of several thousands to millions of nodes [5, 6]. Using intricate tools for simulating a multitude of parameters, it may be possible to increase the real-world numbers by a couple of orders of magnitude. However, the difficulty of pursuing this approach obfuscates and misses another, much more crucial issue: The vision of an efficient, decentralized and self-organizing network cannot be achieved by simply putting together a large enough number of sensor nodes. Instead, coming up with the right functional structure is the grand scientific challenge for realizing the vision of sensor networks. Understanding and designing these structures poses a great number of algorithmic tasks, one level above the technical details of individual nodes. As this understanding progresses, new requirements may emerge for the capabilities of individual nodes; moreover, it is to be expected that the technical process and progress of miniaturization may impose new parameters and properties for a micro-simulation.

In Section 3 we discuss a powerful alternative: Our simulator Shawn focuses on the algorithmic effects of interaction within the network, rather than trying to simulate the physical basics. This allows a large-scale simulation of algorithmic approaches to large and complex networks.

2 Topology-Aware Algorithms

An important issue in WSNs is *Localization* or *Positioning*, i.e., letting the nodes know where they are. This essentially adds geometry to what would otherwise be conceivable only as a graph. It is often assumed that location knowledge

can only be established by providing coordinates in 2- or 3-dimensional space to the nodes. Unfortunately, it is not generally possible to equip all nodes with positioning devices such as GPS receivers, due to size and cost constraints, and, most importantly, because GPS receivers only function in environments that allow line-of-sight connections to satellites.

Practically all relevant localization scenarios boil down to the following setting: Given some (or even zero) nodes that know their position, and a graph following geometric constraints, find consistent positions for the other nodes. Sometimes, inter-node distance or angles are given. It is well known that these problems are NP-hard if there is any inaccuracy in the data [7]. In a prestudy [1] we confirmed that available heuristics produce unusable results also in realistic scenarios.

This motivated us to seek location knowledge that does not consist of coordinates, but topological and geometric structures instead.

2.1 Boundary Recognition

One topological problem is *Boundary Recognition*: Given a sensor network, find the boundary of the covered area. There is no formal definition of the problem, as there are many different meaningful definitions for the boundary. We are especially interested in networks without available position information, i.e., we are given a graph defined by a geometric embedding, but the embedding itself is unknown. It is easy to construct a graph that can be defined via different embeddings, each of which has a different boundary.

We consider the following setup: The network populates a geometric region $A \subseteq \mathbb{R}^2$, i.e., no "large" piece of A is unoccupied. Now we identify nodes that are located close to the region's boundary ∂A. We assume that the nodes follow the *d-Quasi Unit Disk Graph* model [8, Chapter 2], where nodes can always communicate when their distance is at most $d < 1$, can never communicate when their distance is more than 1, and nothing is assumed for distances in between. This model is a more realistic generalization of Unit Disk Graphs (which arise as the special case $d = 1$), as it allows for non-circular communication ranges.

Randomized Settings: A straightforward approach is the following: Assume the nodes are distributed on A following a uniform distribution. Nodes close to ∂A will have fewer neighbors than the average, as parts of their communication range lies outside of A. We developed an algorithm that detects nodes close to ∂A with high probability, provided sufficient density [9]. The algorithm works well in very high densities (over 100 neighbors on average), which are unlikely to appear in practice. We improved the algorithm with a set of heuristics in the same spirit [10], exploiting measures from social network analysis that are sensitive to how close a node is to the boundary. In experiments we confirmed that they work sufficiently in networks with as few as 20–30 neighbors.

Deterministic Settings: The major issue with randomized approaches such as the previous ones are that they assume a specific distribution of the nodes.

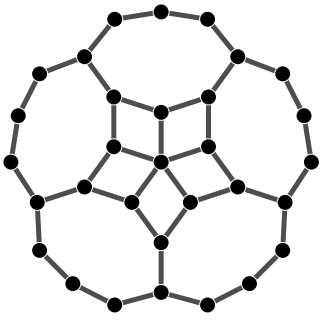


Fig. 1. A 5-flower

To overcome this, we proposed a deterministic algorithm that sidesteps all difficulties. Consider the graph shown in Figure 1, which is an example of a class of graphs called *flowers*. Flowers are beautiful because of the following fact [11]:

Theorem 1. *In every embedding as a d-Quasi Unit Disk Graph for* $\sqrt{2}/2 < d \leqslant 1$*, the outer cycle of a flower describes a polygon that contains the flower's central node on the inside.*

Proof sketch. Any embedding will have a number of properties: First, taking a cycle in the graph and all cycle neighbors produces a cut that cannot be crossed by any edge in the graph. This property vanishes when $d \leqslant \sqrt{2}/2$, which is the reason why d is constrained in the theorem. Consider a connected set U in the graph and such a cycle C with neighbors $N(C)$, where $U \cap (C \cup N(C)) = \emptyset$. It is obvious that either all of U is located outside of the polygon defined by C, or all of U is on the inside. At the same time, a set of independent nodes $I \subseteq U$ requires a certain area for the embedding, as the nodes cannot be placed close to each other. So, if $|I|$ is larger than a specific threshold defined by $|C|$, it can be concluded that U is located outside of the polygon. This is independent of the actual embedding. By applying this argument to cycles connecting the central node of a flower with one of the outer paths, one can prove the claim. □

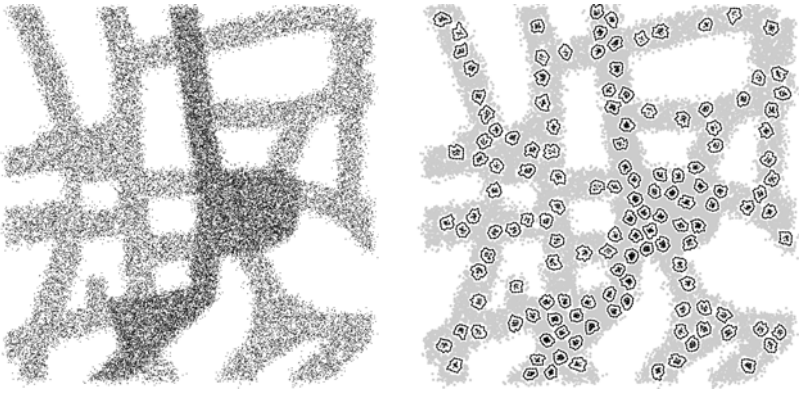
Theorem 1 provides a new way to detect boundaries: While a flower does not indicate anything about the network boundary, it does identify a subnetwork, and for it a meaningful and provably correct boundary.

We extend this idea to a global boundary recognition algorithm. We start by identifying one or more flowers, establishing a local "explored" region. From there, we successively enlarge the region by augmenting the flowers with small appendices. These also have the property that there is a cycle that provably surrounds some nodes. Thereby the algorithm extends the explored area until no further augmentation is possible. In the end, the algorithm produces one or more boundary cycles (if there are holes) and a set of "explored" nodes.

Our algorithm has two important properties that make it stand out:

- It is able to recognize the region for which it can identify the boundary; if the network has parts of very low density, the algorithm will only fail there.
- The boundary is described by a set of connected cycles instead of as a unordered set of nodes. This links the discrete representation to the topological shape of the explored region's boundary.

Figure 2 visualizes our algorithm when applied to a network in an urban scenario. Figure 2(a) shows the network, consisting of 50000 nodes with an average neighborhood size of 20 in the lighter and 30 in the darker regions. The following three pictures show the identified flowers, an intermediate state where explored regions are growing and merging, and finally the resulting boundary and the explored area.



(a) Input network (b) Identified flowers

(c) Growing the explored region (d) Final boundary

Fig. 2. Stages of the boundary recognition algorithm

2.2 Clustering

In the following we show how to utilize the detected boundary to provide location knowledge. Our goal is to segment $A \subseteq \mathbb{R}^2$ into regions $A_1, A_2, \ldots, A_k$. We then construct a cluster graph whose nodes refer to the region. There is an edge whenever the corresponding regions meet. Now each sensor node stores the cluster graph, which is of much smaller size than the sensor network, as well as the information to which cluster it belongs. If the clusters are constructed carefully, the resulting location knowledge accurately describes the network topology and can be applied in a number of applications.

Our clustering scheme is based on the *medial axis* $\mathbf{MA}(A)$ of A, which is the set of all centers of maximal circles contained in A. See Figure 3 for an example. The medial axis is a finite geometric graph under very mild assumptions on A [12]. Furthermore, it describes the topology of A accurately [13].

We construct two types of clusters:

- For every vertex x of $\mathbf{MA}(A)$, we construct a *vertex cluster*. Let $C(x)$ be the set of all contact points of x, that is $C(x) = S \cap \partial A$, where $S \subseteq A$ is a maximal circle centered at x. The set $\mathrm{conv}C(x)$ becomes one cluster. See Figure 3 for a visualization.
- After all vertex clusters $A_1, \ldots, A_{k'}$ are constructed, we consider the connected components of $A \backslash (A_1 \cup \ldots \cup A_{k'})$. For each component B, we construct a *tunnel cluster* from its closure $\overline{B}$.

We can prove that our clusters have certain beneficial shapes [14]. Essentially, when thinking about A as the streets of a city, i.e., a highly complex topological shapes, vertex clusters take the role of intersections, and tunnel clusters reflect the streets. This makes the cluster graph equivalent to the street map of the city.

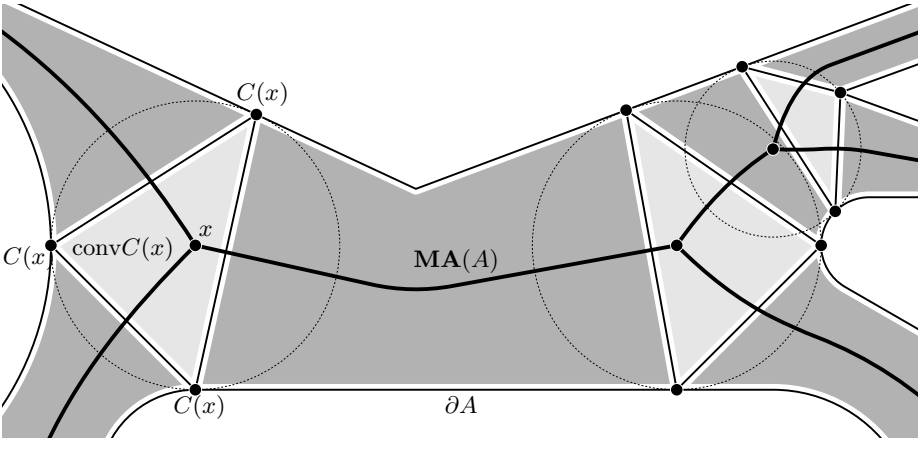


Fig. 3. Medial axis and vertex clusters for network region A

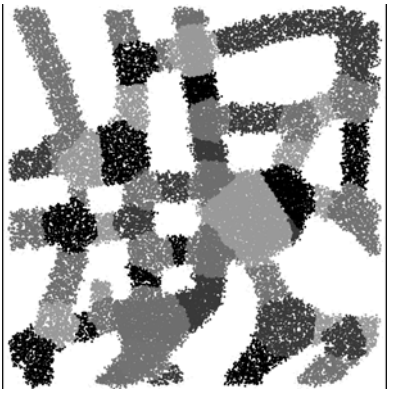


Fig. 4. Result of the clustering algorithm

Theorem 2. *The clustering adheres to the following properties:*

- *The clusters have pairwise disjoint interiors.*
- *Vertex clusters are convex.*
- *The number of clusters is finite.*
- *The boundary of a vertex cluster* $\mathrm{conv}C(x)$ *is a finite alternating sequence of*
  1. *contact components of* x *and*
  2. *chords of the maximal circle around* x*.*
- *Let* T *be a tunnel cluster. Then one of the following holds:*
  1. *Either* $T = A$*, or*
  2. ∂T *consists of one exit of some vertex cluster and one continuous piece of a boundary curve, or*
  3. ∂T *consists of two vertex cluster exits (possibly two exits of the same cluster) and two continuous pieces of boundary curves.*

To apply the scheme in a discrete WSN, the definition has to be translated from geometric to network terms. This is easily done by using graph distances ("hop counts") as a replacement for Euclidean distances. The only problem lies in the definition of vertex clusters, where $\mathrm{conv}C(x)$ has to be computed without available coordinates. Fortunately, such a cluster has a known shape. Our algorithm first constructs a cycle O representing $\partial\mathrm{conv}C(x)$, consisting of shortest paths and boundary pieces. It then marks the interior by taking all nodes in the medial axis ball around x, and removes all those that can reach the outside without crossing O.

A sample output of our algorithm is depicted in Figure 4. It is clearly visible how some clusters exactly describe intersections, while others follow the streets.

2.3 Low-Overhead Routing

A straightforward application of a clustering scheme is message routing. To send a message to a node that cannot be reached directly, it is necessary to construct

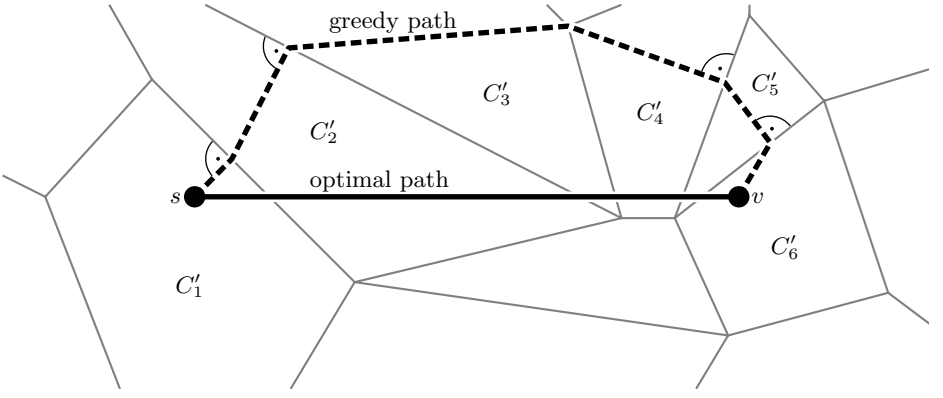


Fig. 5. Optimal vs. greedy path

a routing path through the network. A simple approach is that every node stores a successor for every possible destination, resulting in messages travelling along shortest paths. Unfortunately, such an approach requires a large amount of data to be stored in every node, if the number of possible destinations is unbounded.

Consider the case of an arbitrary node v communicating with a sink s of a WSN, that is a base station collecting data: If there is just a limited number of sinks, it is easy to send messages from v to s over a shortest path P_{vs}, using the aforementioned approach. However, routing responses (i.e., messages from s to v) is not as easy. We do not want the nodes on P_{vs} to store anything about messages they pass, because there could possibly be many simultaneous communications of nodes with the sink. Instead, we want to collect path information in the first packet from v to s, and use that for routing the responses. Simply collecting all nodes of P_{vs} in the packet is infeasible due to packet size limitations.

Our algorithm exploits a clustering. Assume the Euclidean plane to be partitioned into a finite set of convex clusters $C_1, \ldots, C_k \subseteq \mathbb{R}$. Let $v, s \in \mathbb{R}^2$. The optimal, i.e., shortest path from s to v is a straight line segment. The path visits one or more clusters, let these be $C'_1, C'_2, \ldots, C'_\ell$, in this order. Now we construct another v-s-path, called *greedy path*, as follows: Start at $s \in C'_1$. Connect s to the closest point in C'_2 using a straight line. Repeat connecting to the closest point in $C'_3, C'_4, \ldots$, until the path enters C'_ℓ. Finally connect to v directly. See Figure 5.

We were able to prove that the greedy path is an approximation [15,16]:

Theorem 3. *The greedy path is a c-approximation to the optimal path, where $\pi + 1 \leqslant c \leqslant 5$.*

This directly translates into an efficient routing scheme: On the route from v to s, collect the visited clusters in the packet. Every node stores a successor on a shortest path for every neighboring cluster. In the cluster of v, either employ a recursive application of the scheme or temporarily store shortest path routes to v.

Therefore, establishing the greedy path for the sink's responses is straightforward and memory-efficient while producing a short message route.

2.4 Energy-Constrained Flows

Next consider the situation where there is urgent data to be sent from one or more sources to a sink. For example, there could be an event requiring immediate attention, such as a natural disaster. Nodes generate data about it, but there is only limited time until the sink has to react. We model this as the following objective: Maximize the total information reaching the sink before a known point in time. Packets arriving later than this point cannot be considered in the decision process happening at the sink and are therefore useless. Furthermore, the nodes are equipped with non-rechargeable batteries. When a node runs out of energy due to excessive communication, it will shut down, thereby leaving the network.

To model the timing behavior of the sensor network accurately, we introduce an explicit notion of time: With each edge e, we associate a *transit time* $\tau_e \in \mathbb{N}$, defining how much time it takes a packet to travel e. This puts our problem into the context of Dynamic Network Flows [17]. We assume that a *time horizon* $T \in \mathbb{N}$ is given. Each edge $e \in E$ can be used in both directions. The flow on an edge is bounded by an *edge capacity* u_e, which applies to the sum of both directions. Let s be the source and t the sink. Let $\mathcal{P}^{st}$ be the set of all directed, simple s-t-paths in the network. The total transit time of a path $P \in \mathcal{P}^{st}$ is denoted by $\tau(P) := \sum_{e \in P} \tau_e$. We define $\tau_e(P)$ as the time a packet traveling P needs until it reaches the edge $e \in P$. We assume that sending one unit of flow from u to v over $uv \in E$ reduces C_u, the initial energy level of u, by c^{s}_{uv}, and C_v by c^{r}_{uv}. The aggregated cost $c^*_{v,P}$ denotes the total energy cost for a node $v \in V$ when one flow unit is sent over path P.

The *Energy-Constrained Dynamic Flow* problem (ECDF) can be formulated as follows, where $x_P(\theta)$ denotes the flow starting to travel P at time θ:

$$\max \sum_{P \in \mathcal{P}^{st}} \sum_{\theta=0}^{T-\tau(P)} x_P(\theta) \tag{1}$$

$$\text{s.t.} \sum_{\substack{P \ni e: \\ 0 \leqslant \theta - \tau_e(P) \leqslant T - \tau(P)}} x_P(\theta - \tau_e(P)) \leqslant u_e \quad \forall e \in E, \theta = 0, \ldots, T \tag{2}$$

$$\sum_{P \ni v} \sum_{\theta=0}^{T-\tau(P)} c^*_{v,P} x_P(\theta) \leqslant C_v \qquad \forall v \in V \tag{3}$$

$$x_P(\theta) \geqslant 0 \qquad \forall P \in \mathcal{P}^{st}, \theta = 0, \ldots, T - \tau(P)\ . \tag{4}$$

It should be noted that our treatment [18] of the problem is the only one addressing energy-constrained dynamic flows so far, so there are many problem variants left unsolved.

An important special case of the problem is 1-ECDF, where we assume all transit times to be 1. This case reflects the store-and-forward message passing in

WSNs accurately. We showed that two important problem variants are hard [18]:

Theorem 4. *Finding an integral solution for 1-ECDF is* NP*-hard. The original ECDF with arbitrary transit times* $\tau \in \mathbb{N}^E$ *is also* NP*-hard.*

We were not able to decide whether the fractional 1-ECDF problem is hard. We presented some evidence that there may not be a polynomial-size encoding scheme for optimal solutions [18]. If this could be verified, it would prove that the problem is indeed PSPACE-hard. Unfortunately, this remains an open problem.

Our most important result on this problem is the following:

Theorem 5. *1-ECDF admits a distributed FPTAS.*

Proof sketch. Consider a solution to 1-ECDF, where for given P, $x_P(\theta)$ is constant over all feasible θ. Such a solution is called *temporally repeated*. It can be shown that temporally repeated flows are $(1-\varepsilon)$-approximations to optimal flows, when the time horizon T is larger than $\frac{1}{\varepsilon}|V|$. We consider LP formulations for temporally repeated flows and large time horizons as well as for 1-ECDF with small time horizons. Both use an exponential number of variables, yet only a polynomial number of constraints. Furthermore, both formulations describe fractional packing problems. Such problems can be solved using the algorithmic framework by Garg and Könemann [19], which provides a $(1-\varepsilon)^2$-approximation. This algorithm does a number of flow augmentation steps that are controlled by solving the dual separation problem for the LP formulation, where the important twist is to use exponentially growing dual weights. It can be shown that each such step corresponds to finding a shortest path $P \in \mathcal{P}^{st}$ w.r.t. a nonlinear path cost function. Finding such a shortest path can be $(1-\varepsilon)$-approximated by using simplified linear path costs. Our algorithm therefore provides a $(1-\varepsilon)^4$-approximation in polynomial time, by iteratively augmenting flow over shortest paths. The algorithm is easily distributed and requires little more storage at the nodes than the solution itself. □

3 Shawn: A Customizable Sensor Network Simulator

Software for WSNs must be thoroughly tested prior to real-world deployments, because sensor nodes do not offer convenient debugging interfaces and are typically inaccessible after deployment. Furthermore, successfully designing algorithms and protocols for WSNs requires a deep understanding of these complex distributed networks. To achieve these goals, three different approaches are commonly used: analytic methods, real-world experiments, and computer simulations.

Analytic methods are typically not well-suited to support the development of complete WSN applications. Despite their expressiveness and generality, it is difficult to grasp all details of such complex, distributed applications in a purely formal manner.

Real-world experiments are an attractive option as they are a convincing means to demonstrate that an application is able to accomplish a specific task

in practice—if the technology is already available. However, due to the unpredictable environmental influences it is hard to reproduce results or to isolate sources of errors.

Computer simulations are a promising means to tackle the task of algorithm and protocol engineering for WSNs. Existing simulation tools such as Ns-2 [20], OMNeT++ [21], TOSSIM [22], or SENSE [23], reproduce real-world effects inside a simulation environment and therefore mitigate required efforts for real-world deployments. However, the high level of detail provided by these tools obfuscates and misses another, much more crucial issue: The large number of factors that influence the behavior of the whole network renders it nearly impossible to isolate a specific parameter of interest.

For example, consider the development of a routing protocol. The cause for low throughput is not clear at first sight, as the sources for the error are manifold: the MAC layer might be faulty; cross-traffic could limit the bandwidth; or the routing protocol's algorithm is not yet optimal. Therefore, it is not only sufficient to simulate a high number of nodes with a high level of detail. Instead, developers require the ability to focus on the actual research problem.

When designing algorithms and protocols for WSN it is important to understand the underlying structure of the network—a task that is often one level above the technical details of individual nodes and low-level effects. There is certainly some influence of communication characteristics. From the algorithm's point of view, there is no difference between a simulation of low-level networking protocols and the alternative approach of using well-chosen random distributions on message delay and loss. Thus, using a detailed simulation may lead to the situation where the simulator spends much processing time on producing results that are of no interest at all.

To improve this situation, we propose a novel simulation tool: Shawn [24, 25, 26]. The central idea of Shawn is to replace low-level effects with abstract and exchangeable models so that simulations can be used for huge networks in reasonable time, while keeping the focus on the actual research problem. In the following, Section 3.1 discusses fundamental design goals of Shawn while Section 3.2 shows how these goals reflect themselves in Shawn's architecture. Finally, Section 3.3 compares Shawn's performance with two other prominent simulation tools (Ns-2 and TOSSIM).

3.1 Design Goals

Shawn differs in various ways from the above-mentioned simulation tools, while the most notable difference is its focus. It does not compete with these simulators in the area of network stack simulation. Instead, Shawn emerged from an algorithmic background. Its primary design goals are:

- Simulate the effect caused by a phenomenon, not the phenomenon itself.
- Scalability and support for extremely large networks.
- Free choice of the implementation model.

Simulate the effects. As discussed above, existing simulation tools perform a complete simulation of the MAC layer, including radio propagation properties such as attenuation, collision, fading and multi-path propagation. A central design guideline of Shawn is to *simulate the effect caused by a phenomenon, and not the phenomenon itself.* Shawn therefore only models the effects of a MAC layer for the application (e.g., packet loss, corruption and delay).

This has several implications on the simulations performed with Shawn. On the one hand, they are more predictable and there is a performance gain, because such a model can be implemented very efficiently. On the other hand, this means that Shawn is unable to provide the same detail level that, e.g., Ns-2 provides with regard to physical-layer or packet-level phenomena. However, if the model is chosen well, the effects for the application are virtually the same.

It must be mentioned though that the interpretation of obtained results must take the properties of the individual models into account. If, for instance, a simplified communication model is used to benchmark the results of a localization algorithm, the quality of the obtained solution remains unaffected. However, the actual running time of the algorithm is not representative for real-world environments, because no delay or loss occurs.

Scalability. One central goal of Shawn is to support numbers of nodes that are orders of magnitudes larger than the currently existing simulators. By simplifying the structure of several low-level parameters, their time-consuming computation can be replaced by fast substitutes, as long as the interest in the large-scale behavior of the system focuses on unaffected properties. A direct benefit of this paradigm is that Shawn can *simulate vast networks.*

To enable a fast simulation of these scenarios, Shawn can be custom-tailored to the problem at hand by selecting appropriate configuration options. This enables developers to optimize the performance of Shawn specifically for each single scenario. For example, a scenario without any mobility can be treated differently than a scenario where sensor nodes are moving.

Free model choice. Shawn supports a multi-stage development cycle, where developers can *freely choose the implementation model.* Using Shawn, they are not limited to the implementation of distributed protocols. The rationale behind this approach is that—given a first idea for a novel algorithm—the next natural step is not the design of a fully distributed protocol. In fact, it is more likely to perform a structural analysis of the problem at hand. To get a better understanding of the problem in this phase, it may be helpful to look at some exemplary networks to analyze their structure and the underlying graph representation. This allows developers to start from an initial idea and gradually leads to a fully distributed protocol.

3.2 Architecture

Shawn's architecture comprises three major parts (see Figure 6): *Models*, *Sequencer* and *Simulation Environment.*

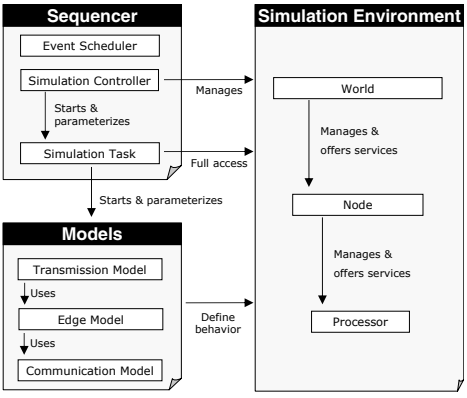


Fig. 6. High-level architecture of Shawn and overview of its core components

Every aspect of Shawn is influenced by one or more *Models*, which are the key to its flexibility and scalability. The *Sequencer* is the central coordinating unit in Shawn as it configures the simulation, executes tasks sequentially and controls the simulation. The *Simulation Environment* is the home for the virtual world in which the simulated sensor nodes reside. In the following, the functionality of these core components is described in detail.

Models. Shawn distinguishes between *models* and their respective implementations. A model is the interface used by Shawn to control the simulation without any knowledge on what a specific implementation may look like. Shawn maintains a repository of model implementations that can be used to compose simulation setups by selecting the desired behaviors. The implementation of a model may be simplified and fast, or it could provide close approximations to reality. This enables the user to select the most appropriate implementation of each model to fine-tune Shawn's behavior for a particular simulation task.

As depicted in Figure 6, three models form the foundation of Shawn: *Communication Model*, *Edge Model* and *Transmission Model*. In the following, these models and their already included implementations are explained in detail. Other models of minor importance are briefly introduced at the end of this section.

Communication Model. Whenever a simulated sensor node in Shawn transmits a message, the potential receivers of this message must be identified by the simulator. Note that this does not determine the properties of individual transmissions but defines whether two nodes can communicate at all. This question is answered by implementations of the *Communication Model*.

By implementing an instance of such a model, arbitrary communication patterns can be realized. Shawn comes with a set of different *Communication Model* implementations, such as the Unit Disk Graph Model, in which two nodes can communicate bidirectionally if the Euclidean distance d between the nodes is less than r_{max}; or the *Radio Irregularity Model* (RIM, [27, 28]), which is based on real-world experiments.

The *Edge Model* provides a graph representation of the network. The simulated nodes are the vertices of the graph, and an edge between two nodes is added whenever the *Communication Model* returns `true`. To assemble this graph representation, the *Edge Model* repeatedly queries the *Communication Model*. It is therefore possible to access the direct neighbors of a node, the neighbors of the neighbors, and so on. This is used by Shawn to determine the potential recipients of a message by iterating over the neighbors of the sending node. Simple centralized algorithms that need information on the communication graph can be implemented very efficiently as in contrast to other simulation tools, no messages must be exchanged that serve as probes for neighboring nodes.

Depending on the application's requirements and its properties, different storage models for these graphs are needed. For instance, mobile scenarios require different storage models than static scenarios. In addition, simulations of relatively small networks may allow storing the complete neighborhood of each node in memory. Conversely, huge networks will impose impractical demands for memory and hence supplementary edge models trade memory for runtime, e.g., by recalculating the neighborhood on each request or by caching a certain amount of neighborhoods. Accordingly, Shawn provides different *Edge Model* implementations.

Transmission Model. Whenever a node transmits a message, the behavior of the transmission channel may be completely different than for any other message transmitted earlier. For instance, cross traffic from other nodes may block the wireless channel or interference may degrade the channel's quality. To model these transient characteristics inside Shawn, the *Transmission Model* determines the properties of an individual message transmission. It can arbitrarily delay, drop or alter messages. This means that a message may not reach its destination even if the *Communication Model* states that two nodes can communicate and the *Edge Model* lists these two nodes as neighbors.

Again, the choice of an implementation strongly depends on the simulation goal. In case that the runtime of an algorithm is not of interest but only its quality, a simple transmission model without delay, loss or message corruption is sufficient. Models that are more sophisticated could take contention, transmission time and errors into account at the cost of performance.

Shawn's built-in transmission models cover both abstract and close-to-reality implementations. The *Reliable* transmission model delivers all messages immediately, without loss or corruption to all neighboring nodes. *Random drop* discards messages with a given probability but it neither delays nor alters messages. Additionally, an implementation is available that models the effect of the well-known CSMA/CA [29,30] medium access scheme.

Sequencer. The sequencer is the control center of the simulation: It prepares the world in which the simulated nodes live, instantiates and parameterizes the implementations of the models as designated by the configuration input and controls the simulation. It consists of *Simulation Tasks*, the *Simulation Controller* and the *Event Scheduler*.

Simulation Tasks are pieces of code that are invoked from the configuration of the simulation supplied by the user. They are not directly related to the simulated application, but they have access to the whole simulation environment and are thus able to perform a wide range of tasks. Example uses are managing simulations, gathering data from individual nodes or running centralized algorithms.

Shawn exclusively uses tasks to expose its internal features to the user. A variety of tasks is included in Shawn that supports the creation and parameterization of new simulation worlds, nodes, routing protocols, random variables, etc. Even the actual simulation is triggered using a task.

The *Simulation Controller* acts as the central repository for all available model implementations and runs the simulation by transforming the configuration input into parameterized invocations of *Simulation Tasks*. In doing so, it mediates between Shawn's simulation kernel and the user. In line with most other components of Shawn, the *Simulation Controller* can be customized by a developer to realize an arbitrary control over the simulation. The default implementation reads the configuration commands from a text file or the standard input stream, while a second one allows the use of Java scripts to steer the simulation. While providing the same functionality, it allows more complex constructs and evaluations already in the configuration file.

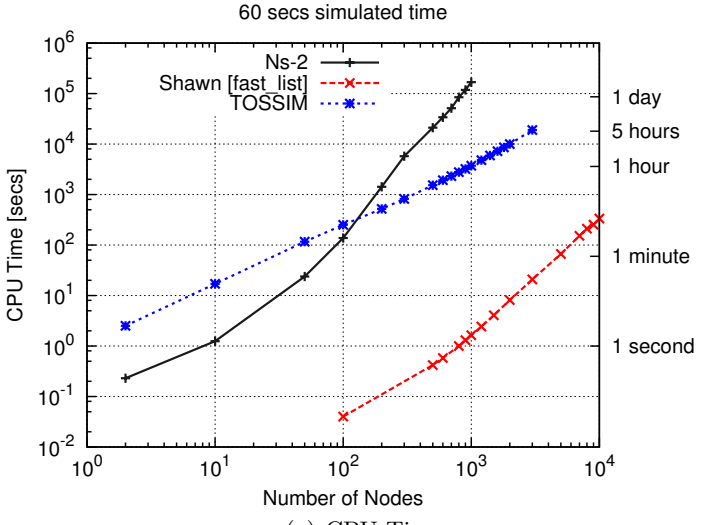
Event Scheduler. Shawn uses a discrete event scheduler to model time. The *Event Scheduler* is Shawn's time-keeping instance. Objects that need the notion of time can register with the Event Scheduler to be notified at an arbitrary point in time. The simulation always skips to the next event time and notifies the registered handlers. This process continues until all nodes signal either that they have powered down or until the maximum configured time has elapsed.

This has some performance advantages compared with traditional approaches that use fixed time intervals (such as a clock-tick every 1ms): First, handlers are notified only at the precise time that they have requested avoiding unnecessary calls to idle or waiting nodes that have no demand for processing.

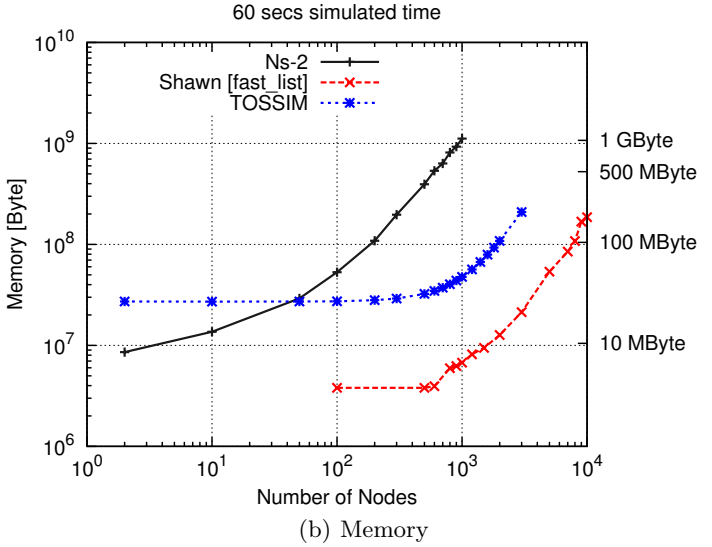
Simulation Environment. The simulation environment is the home of the virtual world in which the simulation objects reside. As shown in Figure 6, the simulated *Nodes* reside in a single *World* instance. The Nodes themselves serve as a container for so-called *Processors*. Developers using Shawn implement their application logic as instances of these Processors. By decoupling the application inside a Processor from the Node, multiple applications can easily be combined in a single simulation run, without changing their implementations. For instance, one processor could implement an application-specific protocol while another processor gathers statistics data.

It is often the case that an algorithm requires input that is produced by multiple (potentially very complex) other algorithms. To avoid waiting for the same results of these previous steps repeatedly in every simulation run, Shawn offers the ability to attach type-safe information to Nodes, the World and the Simulation Environment. Simulation Tasks provide the ability to load tags from and

to save tags to XML documents. A benefit of this concept is that it allows decoupling state variables from member variables in the user's simulation code. By this means, parts of a potentially complicated protocol can be replaced without code modification, because the internal state is stored in tags and not in member variables of a special implementation.



(a) CPU Time



(b) Memory

Fig. 7. Comparison of Shawn, Ns-2 and TOSSIM

3.3 Evaluation

This section evaluates the performance and adaptability of Shawn by comparing Shawn with Ns-2 and TOSSIM. Because the exchange of wireless messages is the key ingredient in wireless sensor networks, a simulator's ability to dispatch messages to their recipients determines the speed of simulations. In the following, measurements are presented that show the amount of memory and CPU time required to simulate a simple application that broadcasts a message every 250ms of simulated time.

The communication range of the sensor nodes is set to 50 length units and each simulation runs for 60 simulated time units. The size of the simulated area is 500x500 length units. A number of simulations with increasing node count were performed. Therefore, the network's density increases steadily as more nodes are added to the scenario. This application has been implemented for Ns-2, TOSSIM and Shawn. All simulation tools were used as supplied by the source repositories with maximal compiler optimization enabled. The simulations were run on standard, state-of-the-art i686 PCs.

Figure 7 depicts the required CPU time in seconds and the maximally used amount of RAM for the three simulation tools at different node counts. It should be noted that this kind of comparison is biased in favor of Shawn, because the two other simulators perform much more detailed computations to arrive at the same results; it should be seen primarily as an indication how application developers can benefit from using Shawn when these detailed results are not in the focus of interest. (Note that these and the following figures use a logarithmic scale on both axes.)

The first thing to notice is that Shawn outperforms both other simulation tools by orders of magnitude. Ns-2 hits the one-day barrier where Shawn is still finishing in less than one minute with a considerably smaller memory footprint. As mentioned above, this is because Ns-2 performs a very detailed simulation of lower layers such as the physical and the data-link layer, while Shawn simply dispatches the messages using a simplified model. Nearly the same situation applies to TOSSIM that simulates an underlying TinyOS-supported hardware platform. This clearly shows that Shawn excels in its specialty: the simulation of large-scale sensor networks with a focus on abstract, algorithmic considerations and high-level protocol development.

3.4 Conclusions and Future Work

The above measurements show that Shawn's central design goal (simulate the effect caused by a phenomenon, and not the phenomenon itself) indeed leads to a high scalability and performance when compared to traditional approaches to simulation. Therefore, developers must carefully select the simulation tool, depending on the application area. When detailed simulations of issues such as radio propagation properties or low-layer issues should be considered, Shawn is obviously not the perfect choice. This is where Ns-2 and TOSSIM offer the desired granularity. However, when developing algorithms and high-level protocols

for WSNs, this level of detail often limits the expressiveness of simulations and blurs the view on the actual research problem. This is where Shawn provides the required abstractions and performance.

Shawn is currently in active development and used by several universities and companies to simulate wireless (sensor) networks. It was and continues to be an invaluable tool for over 20 research publications and more than 30 bachelor and master theses. Shawn is also the enabling means behind the development of algorithms and protocols for WSNs in the SWARMS [31], the SwarmNet [32] and the EU-funded FRONTS [33] and WISEBED [34] projects. It supports recent versions of Microsoft Windows and most Unix-like operating systems, such as Linux and Mac OS X. In addition, Shawn is easily portable to other systems with a decent C++-compiler. It is licensed under the liberal *BSD License*[1] and the full source code is available for download at `http://www.sourceforge.net/projects/shawn`.

References

1. Buschmann, C., Fekete, S.P., Fischer, S., Kröller, A., Pfisterer, D.: Koordinatenfreies Lokationsbewusstsein. IT- Information Technology 47, 70–78 (2005)
2. Mainwaring, A., Polastre, J., Szewczyk, R., Culler, D., Anderson, J.: Wireless sensor networks for habitat monitoring. In: ACM International Workshop on Wireless Sensor Networks and Applications (WSNA 2002), Atlanta, GA (2002)
3. Szewczyk, R., Mainwaring, A., Polastre, J., Anderson, J., Culler, D.: An analysis of a large scale habitat monitoring application. In: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys 2004), pp. 214–226. ACM Press, New York (2004)
4. Zhang, P., Sadler, C.M., Lyon, S.A., Martonosi, M.: Hardware design experiences in ZebraNet. In: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys 2004), pp. 227–238. ACM Press, New York (2004)
5. Estrin, D., Govindan, R., Heidemann, J.: Embedding the Internet: Introduction. Commun. ACM 43(5), 38–41 (2000)
6. Kumar, V.: Sensor: the atomic computing particle. SIGMOD Rec. 32(4), 16–21 (2003)
7. Basu, A., Gao, J., Mitchell, J.S., Sabhnani, G.: Distributed localization using noisy distance and angle information. In: Proceedings of the 7th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC 2006), pp. 262–273. ACM Press, New York (2006)
8. Wagner, D., Wattenhofer, R. (eds.): Algorithms for Sensor and Ad Hoc Networks. LNCS, vol. 4621. Springer, Heidelberg (2007)
9. Fekete, S.P., Kröller, A., Pfisterer, D., Fischer, S., Buschmann, C.: Neighborhood-based topology recognition in sensor networks. In: Nikoletseas, S.E., Rolim, J.D.P. (eds.) ALGOSENSORS 2004. LNCS, vol. 3121, pp. 123–136. Springer, Heidelberg (2004)
10. Fekete, S.P., Kaufmann, M., Kröller, A., Zweig, K.A.: A new approach for boundary recognition in geometric sensor networks. In: Proceedings of the 17th Canadian Conference on Computational Geometry (CCCG 2005), pp. 82–85 (2005)

[1] http://www.opensource.org/licenses/bsd-license.php

11. Kröller, A., Fekete, S.P., Pfisterer, D., Fischer, S.: Deterministic boundary recognition and topology extraction for large sensor networks. In: Proceedings of the 17th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA 2006), pp. 1000–1009 (2006)
12. Choi, H.I., Choi, S.W., Moon, H.P.: Mathematical theory of medial axis transform. Pacific Journal of Mathematics 181, 57–88 (1997)
13. Sherbrooke, E.C., Patrikalakis, N.M., Wolter, F.E.: Differential and topological properties of medial axis transforms. Graphical Models and Image Processing 58(6), 574–592 (1996)
14. Kröller, A.: Algorithms for Topology-Aware Sensor Networks. Ph.D thesis, Braunschweig Institute of Technology (2008)
15. Förster, K.T.: Clusterbasierte Objektüberwachung in drahtlosen Sensornetzwerken: Lokal optimale Wege in der Ebene und ihre Anwendung in Sensornetzwerken. Diploma thesis, Braunschweig Institute of Technology (2007)
16. Fekete, S.P., Förster, K.T., Kröller, A.: Local routing in geometric cluster graphs (2009)
17. Ford, L.R., Fulkerson, D.R.: Constructing maximal dynamic flows from static flows. Operations Research 6, 419–433 (1958)
18. Fekete, S.P., Hall, A., Köhler, E., Kröller, A.: The maximum energy-constrained dynamic flow problem. In: Gudmundsson, J. (ed.) SWAT 2008. LNCS, vol. 5124, pp. 114–126. Springer, Heidelberg (2008)
19. Garg, N., Könemann, J.: Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In: Proc. FOCS, p. 300 (1998)
20. University of Southern California, Information Sciences Institute (ISI): Ns-2: Network simulator-2 (1995), `http://www.isi.edu/nsnam/ns/`
21. Varga, A.: OMNeT++: Objective modular network testbed in C++ (2007), `http://www.omnetpp.org`
22. Levis, P., Lee, N., Welsh, M., Culler, D.: TOSSIM: Accurate and scalable simulation of entire TinyOS applications. In: Proceedings of the First ACM Conference on Embedded Networked Sensor Systems, SenSys 2003 (2003), `http://www.cs.berkeley.edu/~pal/research/tossim.html`
23. Szymanski, B.K., Chen, G., Branch, J.W., Zhu, L.: SENSE: Sensor network simulator and emulator (2007), `http://www.cs.rpi.edu/~cheng3/sense/`
24. Kröller, A., Pfisterer, D., Buschmann, C., Fekete, S.P., Fischer, S.: Shawn: A new approach to simulating wireless sensor networks. In: Design, Analysis, and Simulation of Distributed Systems 2005 (DASD 2005), pp. 117–124 (2005)
25. Pfisterer, D., Fischer, S., Kröller, A., Fekete, S.: Shawn: Ein alternativer Ansatz zur Simulation von Sensornetzwerken. Technical report, 4. Fachgespräch Drahtlose Sensornetze der GI/ITG-Fachgruppe Kommunikation und Verteilte Systeme (2005)
26. Fekete, S.P., Kröller, A., Fischer, S., Pfisterer, D.: Shawn: The fast, highly customizable sensor network simulator. In: Proceedings of the Fourth International Conference on Networked Sensing Systems, INSS 2007 (2007)
27. Zhou, G., He, T., Krishnamurthy, S., Stankovic, J.A.: Impact of radio irregularity on wireless sensor networks. In: MobiSys 2004: Proceedings of the 2nd international conference on Mobile systems, applications, and services, pp. 125–138 (2004)
28. Zhou, G., He, T., Krishnamurthy, S., Stankovic, J.A.: Models and solutions for radio irregularity in wireless sensor networks. ACM Trans. Sen. Netw. 2(2), 221–262 (2006)
29. Coulouris, G., Dollimore, J., Kindberg, T.: Distributed Systems: Concepts and Design, 4th edn. Addison Wesley, Reading (2005)

30. Tanenbaum, A.S., Steen, M.V.: Distributed Systems: Principles and Paradigms. Prentice Hall PTR, Englewood Cliffs (2001)
31. Fischer, S., Luttenberger, N., Buschmann, C., Koberstein, J.: SWARMS: Soft-Ware Architecture for Radio-based Mobile Self-organizing Systems, DFG SPP 1140 (2002), http://www.swarms.de
32. Fekete, S.P., Fischer, S., Kröller, A., Pfisterer, D.: SwarmNet: Algorithmen und Protokolle für Vernetzung und Betrieb großer Schwärme autonomer Kleinstprozessoren, DFG SPP 1126 (2003), http://www.swarmnet.de
33. Spirakis, P., et al.: Foundations of Adaptive Networked Societies of Tiny Artefacts (FRONTS), Research Academic Computer Technology Institute, 7th Framework Programme on Research, Technological Development and Demonstration (2007)
34. Fischer, S., et al.: Wireless sensornet test beds (WISEBED), Universität zu Lübeck, 7th Framework Programme on Research, Technological Development and Demonstration (2007)