Randomized Self-stabilizing Algorithms for Wireless Sensor Networks

Volker Turau and Christoph Weyer

Hamburg University of Technology, Institute of Telematics
Schwarzenbergstraße 95, 21073 Hamburg, Germany
turau@tuhh.de

Abstract. Wireless sensor networks (WSNs) pose challenges not present in classical distributed systems: resource limitations, high failure rates, and ad hoc deployment. The lossy nature of wireless communication can lead to situations, where nodes lose synchrony and programs reach arbitrary states. Traditional approaches to fault tolerance like replication or global resets are not feasible. In this work, the concept of self-stabilization is applied to WSNs. The majority of self-stabilizing algorithms found in the literature is based on models not suitable for WSNs: shared memory model, central daemon scheduler, unique processor identifiers, and atomicity. This paper proposes problem-independent transformations for algorithms that stabilize under the central daemon scheduler such that they meet the demands of a WSN. The transformed algorithms use randomization and are probabilistically self-stabilizing. This work allows to utilize many known self-stabilizing algorithms in WSNs. The proposed transformations are evaluated using simulations and a real WSN.

1 Introduction

Wireless sensor networks (WSNs) are networks of small, battery-powered, resource-constrained wireless devices equipped with sensors embedded in a physical environment where they operate unattendedly for long periods of time. WSNs pose challenges not present in classical distributed systems, foremost extreme resource limitations, high failure rates, and ad hoc deployment. These boundary conditions and the high number of nodes preclude dependence on manual configuration and control. Inevitably unattended WSNs must self-organize in response to node failures or addition of new nodes, and must adapt to changing environmental conditions. The dynamic and lossy nature of wireless communication caused by the primitive, low-power radio transceivers found in WSNs can lead to situations, where nodes lose synchrony and their programs reach arbitrary states [1]. Traditional approaches to fault tolerance like replication where the effects of faults are shielded or a shutdown and globally reset of the complete network are not feasible. In this work, the concept of self-stabilization pioneered by Dijkstra [2] is applied to WSNs. A distributed system is self-stabilizing if after transient faults, regardless of their cause, it returns to a legitimate state in a finite number of steps regardless of the initial state, and the system remains in a legitimate state until another fault occurs [3, 4]. Self-stabilizing algorithms tolerate arbitrary transient failures caused by corruptions of local state, or the disruption of message passing channels, or system resets with unknown initialization. They do not try to handle every individual failure separately, but try to capture the commonality of all failure modes.

Self-stabilization provides a generalized non-masking approach to fault tolerance. This implies that the system experiences the effect of transient faults, in contrast to the replication paradigm. As a consequence applications must be prepared to handle or tolerate these situations. A disadvantage of self-stabilizing algorithms is that a node does not know when the algorithm has stabilized. Self-stabilization fits into the unattended operation style of WSNs, where no outside intervention is necessary. Over the last 20 years many self-stabilizing algorithms have been proposed, quite a few of them are of interest for WSNs: graph coloring [5], articulation points [6], dominating sets [7], depth-first trees [8], and spanning trees [9]. However, the majority of these algorithms is based on models not suitable for the constraints of WSNs: shared memory model, central daemon scheduler, unique processor identifiers, and atomicity. To utilize these algorithms in WSNs, transformations from these strict models into the WSN model are needed. The majority of transformations that have been proposed so far appear to be problem-specific (for an exception see [10]). There is a strong need to devise a general method for systematically transforming algorithms into the realm of WSNs while preserving the stabilization character.

The main contribution of the paper consists of problem-independent stabilization preserving transformations for algorithms that stabilize under the central daemon scheduler in a bounded number of moves in anonymous networks. This is a generalization of problem-specific solutions for graph algorithms such as vertex coloring [5] and minimal independent sets [11]. The key concept is to introduce randomization, as a consequence the transformed algorithms are only probabilistically self-stabilizing. This work enables us to execute self-stabilizing algorithms designed for the central daemon scheduler in WSNs. More importantly, it also helps to develop new and more practical self-stabilizing algorithms for WSNs.

The paper is organized as follows: Section 2 introduces the main concepts of self-stabilization and Section 3 discusses the problems of self-stabilizing WSNs. After that the example used in the experiments covered in Section 6 is presented. Section 5 contains the main contribution, the transformations. In Section 6 preliminary results attained by simulations and through an implementation of the transformation using a real sensor network are presented. The paper ends with related work and a conclusion.

2 Self-stabilization

The objective of self-stabilization is to recover from transient faults in a bounded time without any external intervention. The absence of faults is defined by a predicate $\mathcal{P}$ over the global state of the system, $\mathcal{P}$ is defined locally, i. e., based on the local state of each node and the states of their neighboring nodes. More formally, let $N = \{N_1, N_2, \ldots, N_n\}$ be a set of sensor nodes and $E \subseteq N \times N$ be a set of bidirectional communication links in a sensor network. The topology of the system is represented as the undirected graph $G = (N, E)$. A set of local variables defines the local state of a node. By s_i, we denote the local state of node $N_i \in N$. A tuple of local states $(s_1, s_2, \ldots, s_n)$ forms a *configuration* of the sensor network and defines the global state. Let Σ be a set of all configurations. A *system* is a pair $(\Sigma, \rightarrow)$, where $\rightarrow: \Sigma \times \Sigma$ is a transition relation. An *execution* is a maximal sequence $c_0, c_1, c_2, \ldots$ of configurations such that $c_i \rightarrow c_{i+1}$ for each $i \geq 0$.

A transition is caused by the execution of a program on a node (all nodes run the same program). Programs consist of rules of the following kind:

$$\begin{array}{c} \textit{precondition}_1 \longrightarrow \textit{statement}_1 \\ \textit{precondition}_2 \longrightarrow \textit{statement}_2 \\ \ldots \end{array}$$

The preconditions are Boolean expressions based on the state of a node and the states of its neighbors only (i. e., no global view of the network). The semantics of a program is that whenever a node executes, it executes the statements corresponding to a rule whose precondition evaluates to true. The statements of a rule can only change the local state. It is assumed that reading the states of the neighbors is atomic. The execution of a selected statement is also assumed to be atomic. If more than one precondition is satisfied, then one of them is chosen non-deterministically. A *move* of a node is the execution of a rule. A rule is called *enabled* if its precondition evaluates to true, otherwise it is called *disabled*. A node is called *enabled* if at least one of its rules is enabled.

A configuration $c \in \Sigma$ is called *legitimate* relative to $\mathcal{P}$ if c satisfies $\mathcal{P}$. Let $\mathcal{L} \subseteq \Sigma$ be the set of all legitimate configurations. A system $(\Sigma, \rightarrow)$ is *self-stabilizing* with respect to $\mathcal{P}$ if the following two conditions hold:

1. If $c \in \mathcal{L}$ and $c \rightarrow c'$ then $c' \in \mathcal{L}$ (closure property).
2. Starting from any configuration $c \in \Sigma$ every execution reaches $\mathcal{L}$ within a finite number of transitions (convergence property).

Self-stabilization models the ability of a system to recover from failures under the assumption that they do not continue to occur forever (*eventual-quiescence*). To model long periods of time during which the system operates without errors, it is assumed that eventually the system enters a last operational interval that is infinitely long in which there are no more faults occur. This interval is called the *final interval*. When a fault occurs the system enters a new configuration and the algorithm restarts from this configuration. The same is true for changes of the topology, i. e., adding or removing node or links.

Designing self-stabilization for anonymous networks under the distributed scheduler is a difficult task and for some problems the non-existence of such algorithms has been proven. A distributed scheduler may select two enabled neighboring nodes to execute at the same step, and as a result both nodes may be enabled thereafter. It is not difficult to see that if any two neighboring nodes never execute at the same step, the computation is equivalent to the centralized scheduler. By local mutual exclusion, execution of two neighboring processes at the same step is disabled (see [12, 10] for example). If the nodes have unique identifiers, then they can often be used for this purpose, e. g., [5]. Another solution is to use randomization. A system is called *randomized*, if the execution of an enabled node depends on the outcome of a random experiment. A randomized system $(\Sigma, \rightarrow)$ is said to be *probabilistically self-stabilizing* with respect to a predicate $\mathcal{P}$ if it satisfies the closure property as defined above and there exists a function $f : \mathbb{N} \rightarrow [0, 1]$ satisfying $\lim_{k\rightarrow\infty} f(k) = 0$, such that the probability of reaching a legitimate configuration, starting from an arbitrary configuration within k transitions, is $1-f(k)$ (probabilistic convergence property).

The execution of the transitions of the enabled nodes is controlled by a scheduler. A *schedule* S is a sequence $S_1, S_2, \ldots$ of non-empty subsets of enabled nodes called *rounds*. All nodes in S_i may execute their moves in parallel, but the first move in S_i can only be executed after the last move of S_{i-1} has finished. At the beginning of every round, all nodes evaluate the preconditions of their rules and a subset of the enabled nodes is selected. Then all selected nodes execute the statement of a single enabled rule. Schedules are an abstraction used to model the semantics of concurrent execution, they are not an implementation requirement. Schedules are restricted to satisfy certain fairness and atomicity properties. A scheduler is *fair* if for any schedule S it selects, for all $p \in N$, for infinitely many values of i, $p \in S_i$ holds. A *central daemon* scheduler is one that satisfies $|S_i| = 1$ for all i; it models the serial activation of one process at each step. A *distributed daemon* scheduler satisfies $|S_i| \leq n$ for all i, i. e., all enabled nodes may execute their statements in parallel.

3 Self-stabilizing WSNs

Wireless sensor networks are inherently fault-prone due to the shared wireless communication medium: message losses and corruptions due to fading, collisions, and hidden-terminal effects are the norm rather than the exception [1]. In many cases nodes can communicate with each other only with a very low probability. Moreover, node failures due to crashes and energy exhaustion are commonplace. These faults can drive a portion of a WSN to be arbitrarily corrupted and hence to become inconsistent with the rest of the network. Since WSNs are deployed in remote locations, in-situ maintenance is not feasible and therefore sensor network applications should be self-healing. Self-healing ensures eventual compliance with a specification upon starting from a corrupted state. A big challenge for fault-tolerance is the energy constraint of the nodes. Applications cannot impose an excessive communication burden on nodes. As a consequence, self-healing of WSNs must be local and communication-efficient.

Self-stabilization is a specific form of self-healing that has many advantages for WSNs. Large scale WSNs will be operating over a longer period of time and additional nodes can be added at any time. Self-stabilizing eliminates the overhead of initialization all nodes in a consistent manner, actually state variables need no initialization at all. The software of the nodes in a long running network needs an upgrade over time, the software may be distributed over the wireless medium. While switching to the new version, it will be impossible for all nodes to simultaneously switch software. A self-stabilizing system guarantees completion of this change in a finite number of operations. Once a node recovers after failing (e. g., after a temporary power outage or due to a memory crash) its state may be inconsistent with the rest of the system. Self-stabilization also guarantees consistency of the nodes in this case. And finally, errors in transmissions leading to corruption of data may be handled by self-stabilizing algorithms as well.

Most research on self-stabilizing algorithms has concentrated on the central daemon scheduler. The main reason is that proving correctness of algorithms is much easier than in the case of a distributed scheduler. But the concept of a central daemon is against the spirit of Distributed Systems since it does not allow for concurrency. Also, it is difficult to implement this scheduler in a WSN. In the shared memory model each process can read the local states of all neighbor processes without delay. This model is not suitable for sensor networks, instead broadcasts, the communication primitive of wireless networks, should be used. Herman introduced the *cached sensornet transformation* (CST) as an alternative model for WSNs [13]. Let each node $N_i \in N$ in the WSN have a single variable s_i that completely represents the local state of the node. Let N_i have for each neighbor N_j a variable $\nabla_i s_j$, which denotes a cached version of s_j. Atomically, whenever N_i assigns a new value to s_i, node N_i also broadcasts the new value to its neighbors. Whenever a node N_j receives a new value for s_i, it immediately (and atomically) updates $\nabla_j s_i$. Because sending and receiving operations are exclusive in the nodes, we suppose that receiving a cache update message cannot interfere with concurrent assignment and broadcast by the receiving node. To use a self-stabilizing algorithm under this transformation, the rules have to be changed: at every node N_i each reference to s_j is replaced with $\nabla_i s_j$ for all j. The execution of a statement does not modify the cache and receiving a broadcast message only changes the cache and not the state of the node.

A system is called *cache coherent* if $\nabla_j s_i = s_i$ for all $(N_j, N_i) \in E$. Cache coherence is invariant under local broadcast provided that no messages are lost. Let $\mathcal{A}$ be a self-stabilizing algorithm under the central daemon scheduler. Then the CST transformed algorithm is also self-stabilizing under the central daemon scheduler provided the initial state of the execution was cache coherent and no messages are lost or corrupt.

4 Example

Clustering is a useful technique to control the topology of WSNs. Clustering algorithms should satisfy two properties: In order to allow efficient communication between nodes, every node should have at least one clusterhead in its neighborhood and no two clusterheads should be within each others mutual transmission range. The latter property greatly facilitates the task of establishing an efficient MAC layer, because clusterheads will not face interference. These properties lead to the concept of a maximal independent set in a graph: An independent set (IS) I of G is a subset of N such that no two nodes in I are neighbors. I is a maximal independent set (MIS) if any node v not in I has a neighbor in I. The following self-stabilizing algorithm has been proposed by several authors. Each node has a Boolean variable *in*. A state is called legitimate if the set of nodes v with $v.in = true$ forms a MIS of G. The rules of the algorithm are:

```
if $(in = \textbf{\textit{false}} \wedge \forall\ neighbors\ v : (v.in = \textbf{\textit{false}}\ ))$  $\longrightarrow$  $in := \textbf{\textit{true}}$
if $(in = \textbf{\textit{true}} \wedge \exists\ neighbor\ v : (v.in = \textbf{\textit{true}}\ ))$  $\longrightarrow$  $in := \textbf{\textit{false}}$
```

It was been proved in [14] that this algorithm self-stabilizes under the central daemon scheduler after at most $2n$ moves. There is no guarantee about the quality of the produced MIS, i. e., there may exist another MIS containing more nodes. Clearly this algorithm does not stabilize under the distributed daemon scheduler. Suppose the variable *in* of all nodes initially has the value *true*. Then the first rule of all nodes is enabled. If all nodes execute, then the value changes to *false* for every node and all nodes are enabled again. This process continues forever. Applying the transformation described in the next section to this algorithm yields a randomized algorithm that is probabilistic self-stabilizing under the distributed daemon scheduler.

5 Transformation of self-stabilizing algorithms

This section presents techniques to transform algorithms that self-stabilize under the central daemon scheduler such that they can be used in WSNs under the distributed daemon scheduler. The WSNs under consideration are anonymous networks, i. e., nodes have no globally unique identifiers. Nodes must be able to distinguish their neighbors. It is assumed that messages are not corrupted (e. g., by using error correcting codes). For the time being it is also assumed that no messages are lost. This restriction will be removed in the following part.

Let $\mathcal{A}$ be a self-stabilizing algorithm that stabilizes under the central daemon scheduler in a finite number of moves. The main issue with the distributed daemon scheduler is to enforce the separation of the executions in consecutive rounds. If all nodes have a common understanding of time, then the rounds can be organized under the assumption that the execution times of the statements are bounded by a finite constant. There are several proposals for time synchronization in WSN, e. g., [15]. The first step is to apply the cached sensornet transformation to $\mathcal{A}$, call the resulting algorithm $\mathcal{A}^{\mathcal{C}}$. The main issue with $\mathcal{A}^{\mathcal{C}}$ in WSNs is to guarantee the atomicity of the moves as described above. To deal with concurrent moves within a round, a random element is introduced. It is assumed that each node is equipped with a random number generator *rand*. Furthermore, all nodes have agreed on a constant $p \in (0, 1)$. Let $Rule_{\mathcal{A}^{\mathcal{C}}}$ be the set of rules of algorithm $\mathcal{A}^{\mathcal{C}}$, then for each rule

$$\textit{precondition} \longrightarrow \textit{statement}$$

from $Rule_{AC}$ construct a new rule:

$$\textit{precondition} \longrightarrow \textbf{if } (\text{rand}() < p) \textbf{ then } \textit{statement}$$

Call the randomized algorithm for this new set of rules $\mathcal{A}^{\mathcal{CR}}$. Note that the execution of a statement now involves a call to the random number generator and that a statement of an enabled rule is not necessarily executed in the current round. Algorithm $\mathcal{A}^{\mathcal{CR}}$ has the following property.

Theorem 1. *Let $\mathcal{A}$ be a self-stabilizing algorithm that stabilizes under the central daemon scheduler after a finite number of moves with respect to a predicate $\mathcal{P}$. If the initial configuration is cache coherent, then algorithm $\mathcal{A}^{\mathcal{CR}}$ is probabilistic self-stabilizing with respect to $\mathcal{P}$ under the distributed daemon scheduler provided that all broadcasts are reliable.*

Proof. Suppose that algorithm $\mathcal{A}$ stabilizes after at most M moves. Consider the execution of algorithm $\mathcal{A}^{\mathcal{CR}}$. Since the initial configuration is cache coherent all following configurations are also cache coherent since all messages are sent successfully. Let (S_i) be a schedule under the distributed daemon scheduler. Assume that $S_i \neq \emptyset$ for all $i > 0$ (otherwise $\mathcal{A}^{\mathcal{CR}}$ stabilizes). The probability that exactly one node is executed in round S_i is equal to $\beta_i = c_i p(1-p)^{c_i - 1}$ where $c_i = |S_i| \leq n$. Let (b_i) be a binary sequence where $b_i = 1$ if exactly one node executes during round i and 0 otherwise. Note that $\text{Prob}(b_i = 1) = \beta_i$ and $\text{Prob}(b_i = 0) = 1 - \beta_i$ and that $|\{\beta_i \,|\, i \in \mathbb{N}\}| \leq n$. If the sequence (b_i) has a subsequence $b_{j+1}, b_{j+2}, \ldots, b_{j+M}$ of length M where all elements have the value 1, then $\mathcal{A}^{\mathcal{CR}}$ stabilizes after round $j + M$, because this sub-schedule is equivalent to a schedule under the central daemon scheduler. Let $f(k)$ be the probability that such a subsequence is not contained in $b_1, b_2, \ldots, b_k$. Then by Theorem 3 (see Appendix A)

$$\lim_{k\to\infty} f(k) = \lim_{k\to\infty} P_M(k) = 0$$

and hence $\mathcal{A}^{\mathcal{CR}}$ is probabilistic self-stabilizing under the distributed daemon scheduler. □

Note that the requirement that the initial configuration is cache coherent cannot be dropped. The problem is that if there are nodes with non-coherent caches then there can be situations where no nodes are enabled and the predicate $\mathcal{P}$ is not satisfied at the same time. Theorem 1 does not make a statement about the rate of stabilization, e. g., about the probability that the algorithm stabilizes in k moves. The expression $1 - f(k)$ is a lower bound for this probability, but we have no explicit expression for $f(k)$. Also the values of c_i are specific to the algorithm under consideration. The lower bound $1 - f(k)$ can be improved using the following observation. A sub-schedule $S_j, S_{j+1}, \ldots, S_t$ is equivalent to a schedule under the central daemon scheduler if the nodes of each round of that sub-schedule that execute form an independent set. The proof of Theorem 1 covers the special case of only a single node executing. Note that if two non-neighboring nodes, that have a common neighbor, execute, then the probability that their broadcasts during the sensornet transformation collide at the common neighbor is high.

Unreliable Communication In the following the assumption about the reliability of broadcasts is dropped, i. e., not all neighbors receive a broadcasted message. Let q be the probability that a message is successfully transmitted from one node to another. We make the assumption that all transmissions are independent. Note that the loss of a message is not regarded as a transient fault, it is a possible behavior of the system that is tolerated by the algorithm. Hence, messages may also be lost during the final interval. The argument of the proof of Theorem 1 can also be applied in this case. The probability that exactly one node is executing in round i and that at the same time all broadcasts of this node succeed is equal to

$$\beta_i = p(1-p)^{c_i-1} \sum_{j \in S_i} q^{d_j}$$

where d_j is the degree of node j. The set of all β_i is again a finite set (n^{Δ} is an upper bound). From Theorem 3 it follows that the probability that this algorithm halts after k rounds converges to 1 with increasing k. But that does not necessarily mean that the algorithm is probabilistic self-stabilizing under the distributed daemon scheduler. The problem is that the algorithm may reach a non-cache coherent configuration in which no node is enabled. To overcome this problem each node broadcasts the values of its public variables to its neighbors periodically at the beginning of every round, call this algorithm $\mathcal{A}^{\mathcal{CRP}}$. The probability that exactly one node is executing in round i and that at the same time all broadcasts succeed is equal to $\beta_i = q^m c_i p(1-p)^{c_i-1}$ where m is the number of links in the network. The proof of the following theorem is similar to the proof of Theorem 1. Note that the initial configuration does not need to be cache coherent.

Theorem 2. *Let $\mathcal{A}$ be a self-stabilizing algorithm that stabilizes under the central daemon scheduler after a finite number of moves with respect to a predicate $\mathcal{P}$. Let the probability that a message is successfully transmitted from one node to another be fixed and assume that these events are independent. Then algorithm $\mathcal{A}^{\mathcal{CRP}}$ is probabilistic self-stabilizing with respect to $\mathcal{P}$ under the distributed daemon scheduler.*

Broadcasting the public variables at the beginning of every round causes two problems: It increases the total energy consumption and if all nodes make their broadcast at the beginning of a round, many collisions will occur, probably leading to a prolonged stabilization time. The second problem can be mitigated if the nodes broadcast their data after a random waiting period. Another solution is that nodes do not broadcast their data in each round, but make this decision dependent on the outcome of a random experiment. Call this new algorithm $\mathcal{A}^{\mathcal{CRPP}}$. Let r be the probability that a node makes a broadcast. Then the probability that exactly one node is executed in round i and that at the same time all broadcasted messages are successfully received is equal to

$$\beta_i = r^n q^m c_i p(1-p)^{c_i - 1}$$

where n is the number of nodes in the network. Using Theorem 3 it can be shown that Theorem 2 also holds for algorithm $\mathcal{A}^{\mathcal{CRPP}}$. To further reduce the number of messages sent, the probability of a broadcast could be decreased in every round after a state change, e. g., by reducing the probability to 50%. But then it is no longer possible to prove the probabilistic self-stabilization behavior.

Periodic Broadcasting with implicit Acknowledgments. Once all neighbors of a node know the current state of the node, the node can suspend broadcasting until the state of the node changes again. In order to implement this technique a node needs the information that all neighbors know its current state. To realize this task, nodes include in their broadcasts the latest received states of all neighbors. This way a node can find out whether its current state is known to all neighbors and may then stop the periodic broadcasting. The node still needs to perform broadcasts to signal other nodes that it received their current state. The following code illustrates this procedure. Nodes are in one of two modes: `Broadcast` and `Suspend`, initially the mode is `Broadcast`. Also every time the state of the node changes, the mode immediately changes to mode `Broadcast`.

```
▷ Mode Broadcast
while not all neighbors have acknowledged current state do
    broadcast
end while
mode ← Suspend

▷ Mode Suspend
if received broadcast from neighbor then
    broadcast
end if
```

Call this new algorithm $\mathcal{A}^{\mathcal{CRPPA}}$. Theorem 2 still holds for this algorithm since the modifications have no influence on the stabilization behavior. An increase in packet size is the price for reducing the number of messages. Larger packets result in larger transmission times and may lead to more collisions, which may slow down the stabilization process. The main advantage of this approach is that after the system has reached a legitimate state, no broadcast messages are needed until the next transient fault.

Unidirectional links. WSNs suffer from unidirectional links where one sensor can communicate with another with a high probability although the probability of the reverse communication is very low. Unidirectional links are not considered useful in the context of WSNs, the reason is the lack of efficient MAC layer protocols that work with unidirectional links (e. g., both RTS-CTS and ACK based schemes cannot be used directly). Almost all self-stabilization algorithms are defined for bidirectional links only. As an example, consider the algorithm presented in Section 4. If this algorithm is executed on a system with unidirectional links then the result is no longer a MIS. We therefore propose to exclude unidirectional links from being used by self-stabilization algorithms. To meet this end the neighborhood protocol in use should discard such links. The protocol described in [1] can be used for this purpose, it can also be combined with the periodic broadcasts in order to reduce the total number of messages sent. A link that continuously shows a low quality is discarded by this protocol, at the same time new links are accepted as they appear. These events have to be regarded as faults and therefore cannot occur during the final interval. In order to avoid links to appear and disappear frequently over time, neighborhood protocols have to find a balance between agility and stability. If the intervals between faults of this kind are long enough, the proposed algorithms may stabilize during an interval, otherwise the stabilization process will be disrupted considerably. Since the different algorithms have different stabilize times a general statement about this kind of stability is impossible.

Failing nodes. In the following the case of completely failing nodes is considered. If a node fails it stops broadcasting its state, but neighboring nodes continue to regard this node as a neighbor. The remedy is to associate with each cache value $\nabla_i s_j$ of a node N_i a *time to live* (TTL) value. The TTL value is renewed every time the node receives a message with the state of neighbor N_j and it is decreased every round in which no such message is received. If a cache value $\nabla_i s_j$ is not confirmed within TTL rounds it is discarded and N_i is no longer regarded as a neighbor of N_j. As a consequence a disabled node might get enabled. Thus, the situation is comparable to a discarded link and the discussion from the last section applies.

The concept of TTL values cannot be used in combination with the above described technique to limit the periodic broadcasts with implicit acknowledgments, i. e., algorithm $\mathcal{A}^{\mathcal{CRPPA}}$. The problem is that a node can no longer distinguish a node that suspended broadcasting from a failed node. As a consequence, the failed node could stay permanently in the neighborhood list of a node. This may in turn lead to a non-coherent cache. But TTL values can be combined with algorithm $\mathcal{A}^{\mathcal{CRP}}$. The value of TTL should be large enough to distinguish the case of a failed node from a node that is temporary not able to communicate with its neighbor. Otherwise nodes with an instable link may disappear and appear repeatedly in the neighbor list of a node with negative consequences for the stabilization process. The handling of newly introduced nodes requires no special treatment. In case the join and leave rates are low, the algorithm may stabilize during intervals with fixed topology. This kind of stabilization behavior depends on the number of moves required to stabilize the system after the failure/introduction of a node. The technique of TTL values can also be combined with algorithm $\mathcal{A}^{\mathcal{CRPP}}$. In this case the relationship between TTL and r, the probability that a node makes a broadcast, needs to be carefully tuned. Otherwise, the algorithm will not stabilize during intervals with fixed topology. A detailed analysis of these situations is outside the scope of this paper.

6 Experiments

The stabilization behavior of the proposed transformations were further analyzed in a series of experiments with a MIS algorithm. The experiments are based on simulations and on a real WSN. Let $\mathcal{A}$ be the MIS algorithm based on the two rules presented in Section 4. The stabilization behavior of algorithm $\mathcal{A}^{\mathcal{CR}}$ was analyzed in a simulation. At the beginning of each round the set N_{en} of enabled nodes was determined and after this step all nodes in N_{en} executed the statements of the enabled rule (without checking whether the node was still enabled). As a consequence a node may execute a rule, even though it is no longer enabled. The reason for this mode of operation is to simulate the interleaving execution of the nodes. The algorithm was run on several graph classes:

$G_{n,q}$:	Graphs with n nodes and any pair of nodes is connected by an edge with probability $q \in [0, 1]$.
K_n:	Complete graphs with n nodes.
$U_{n,d}$:	Unit disc graphs with n nodes where the locations of the nodes are randomly selected in a square of side length d.

The Unit disc graph model is included, because it is regularly used in theoretical models of WSNs. For each graph algorithm $\mathcal{A}^{\mathcal{CR}}$ with $p = 0.5$ was executed 500 times.

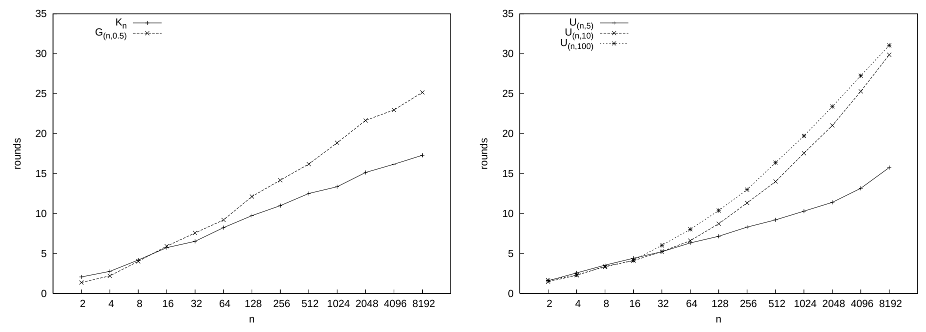

Fig. 1. Average number of rounds of algorithm $\mathcal{A}^{\mathcal{CR}}$ with $p = 0.5$ before stabilization for various classes of graphs

Figure 1 shows the average number of rounds the algorithm needed to stabilize for the different classes of graphs (note the logarithmic scale of the x-axis).

The data shows that the number of rounds for classes K_n and $G_{n,q}$ is roughly proportional to $\log_2 n$. For Unit disc graphs the number of rounds grows slightly faster. Figure 2 shows the average number of the sum of moves executed by all nodes before stabilization. The data shows that roughly $n/2$ moves were needed on the average independently of the class of graphs. Interestingly our experiments also showed that the average number of moves for $\mathcal{A}^{\mathcal{CR}}$ is only slightly higher than in the case of a central daemon scheduler.

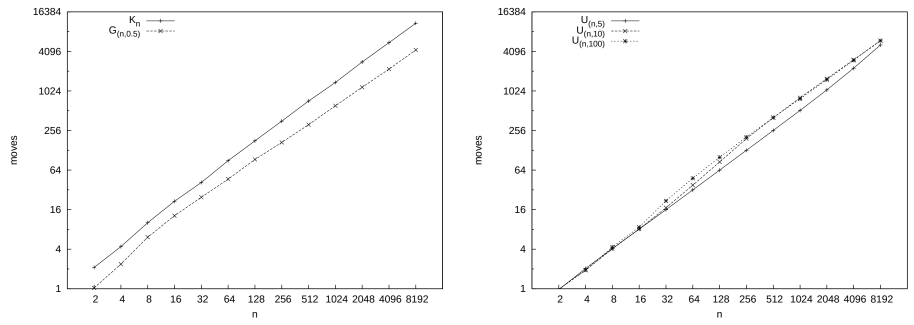

Fig. 2. Average total number of moves of algorithm $\mathcal{A}^{\mathcal{CR}}$ with $p = 0.5$ before stabilization for various classes of graphs

The main weakness of the transformations described above with respect to WSNs are the assumptions about the atomicity of the cached sensornet transformation and about the reliability of message delivery. To analyze the impact of these assumptions an experiment with a real WSN was carried out. The experiment was based on algorithm $\mathcal{A}^{\mathcal{CRP}}$. The sensor network consisted of 25 nodes of type ESB, a sensor node platform developed by the Free University Berlin [16]. A node consists of the micro controller MSP 430 from Texas Instruments, the transceiver TR1001, which operates at 868 MHz at a data rate of 19.2 kbit/s, some sensors, and a RS232 serial interface. Each node has 2 KB RAM and 64 KB EEPROM. The lowest layer of the implementation is a synchronization protocol that is used to force the nodes to operate in rounds, each round had a duration of 10 seconds. In order to reduce the possibility of collisions, the *cached sensornet transformation* was implemented such that each node randomly selected an instant during each round and broadcasted its state. At the end of each round the enabled nodes executed an enabled rule with fixed probability p. To analyze the influence of the value of p on the stabilization time, the experiment was repeated four times with $p = 0.25, 0.5, 0.75$ and 1.0. The sensor nodes were distributed in a grid-style inside a large lecture hall. Each node could communicate with its immediate neighbors, depending on the position in the rectangular grid a node had between 3 and 8 neighbors. Messages from nodes further away were discarded. The transmissions power of the nodes was reduced such that this topology was realized and no avoidable interference was caused.

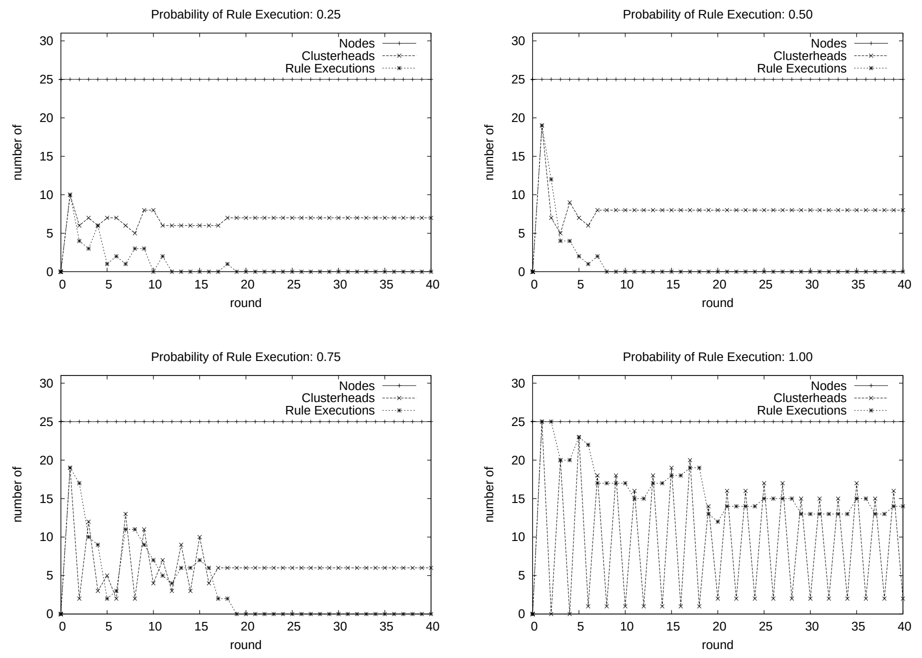

Fig. 3. Wireless sensor network with 25 nodes executing algorithm $\mathcal{A}^{\mathcal{CRP}}$ to compute a minimum independent set. The experiment was repeated with four different values of p. The diagrams depict the number of nodes, clusterheads and rule executions during the first 40 rounds.

The duration of each experiment was 400 seconds, i. e., 40 rounds. Initially no node was a clusterhead (i. e., the variables `in` had the value `false`). The algorithm reached a legitimate configuration for the probabilities $p = 0.25, 0.5$ and 0.75 within the first 19 rounds. Without randomization (e. g., $p = 1.0$) the algorithm did not reach a legitimate configuration within 40 rounds. In this case in every round more than 50% of all nodes made a move, as a consequence the number of clusterheads alternated between a few nodes (0,1 or 2) and about 15-20 nodes. The particular style of initialization of the variable `in` caused this pattern to emerge. In the first round all nodes were enabled, these nodes turned into clusterheads enabling all nodes in round two. At the end of this round no node was a clusterhead. This pattern would have been repeated if no messages had been lost. When a messages is lost after a node has changed its state, the caches at the neighbors of this node are no longer coherent. Because of this phenomenon only 20 nodes made a move in round three.

The other three experiments suggest that the optimal stabilization time is achieved for $p = 0.5$. In this case, the system stabilized after 7 rounds making a total of 44 moves. In case $p = 0.25$ the system could have stabilized after round 12, but the execution of the corresponding rule was deferred for another six rounds until round 18. The reason is that with $p \to 0$ the behavior of the transformation converges to that of the central daemon. The sequence of experiments also shows that the number of moves decreases with the value of p: 136, 44, and 36. A detailed analysis is part of future work.

7 Related Work

Self-stabilization as a tool to achieve fault tolerance in WSNs was first investigated by Herman [13]. He surveys standard models of self-stabilization and relates these to WSNs. In particular, the *cached sensornet transformation* – a construction that transforms a sensor network to a central daemon model – is introduced. Based on the assumption that all links are bidirectional and that nodes possess a CSMA/CA capability, probabilistic stabilization of an algorithm for maximal independent sets is proven.

The work of Herman is extended by Kulkari et al. [17]. To overcome the read/write model – which is used in many existing algorithms, but not applicable to WSNs – the *write all with collision* model (WAC) is introduced. WAC captures the computations of sensor networks. Intuitively, in one atomic action, a node can update its own state and the state of all its neighbors. If two nodes simultaneously try to update the state of a node, then the state of this node is unchanged. Transformations from existing models to the WAC model and vice versa are presented. To obtain the transformed program that is correct in the WAC model, the nodes are organized in a ring. Such a ring can be statically embedded in any arbitrary graph by first embedding a spanning tree in it and then using an appropriate traversal mechanism to ensure that each process appears at least once in the ring. The nodes execute one after the other as they appear on the ring. For timed systems a transformation using a collision-free time-slot based protocol is presented. Under some assumptions it is shown that if the given algorithm is self-stabilizing then the transformed algorithm is also self-stabilizing for a fixed topology. It is also argued that in some cases for untimed systems self-stabilization preserving transformations into the WAC model are not possible. A distributed algorithm for TDMA slot assignment in WSNs that is self-stabilizing to transient faults and dynamic topology change is presented in [18]. The work of Römer et al. on role assignment in WSNs is in parts related to self-stabilizing algorithms [19]. The authors present heuristics to maintain local cache tables at the nodes. They employ randomized delays in order to avoid temporary inconsistencies due to the lack of sequentialization.

Refining self-stabilizing algorithms using tight scheduling constraints into corresponding algorithms for weaker scheduling constraints, while preserving the stabilization property, has been subject of serious research in the last decade. In most cases the core of the transformations is a self-stabilizing local mutual exclusion algorithm based on unique node identifiers [10]. Kakugawa et al. have developed an algorithm that transforms a serial model program to a distributed model [20]. A timestamp based self-stabilizing concurrency control (CC) protocol is incorporated in the transformed program. After the CC protocol stabilizes, it is guaranteed that for each execution of the transformed distributed model program, there always exists an equivalent execution of the original serial model program. Therefore, if the original program is correct with respect to self-stabilization, the transformed program is also self-stabilizing. A drawback of this approach is the efficiency of transformed algorithms, they require more messages than the ones coded from scratch. Furthermore, these transformations cannot be easily adopted to the case of unreliable broadcasts.

Mizuno and Nesterenko considered distributed systems where all processors have unique identifiers. They propose a procedure to transform a self-stabilizing algorithm under the central daemon scheduler to an equivalent self-stabilizing algorithm that runs on an asynchronous shared memory parallel computing system [12]. Timestamps are used to guarantee mutually exclusive execution of guarded commands among neighbor processes.

8 Conclusion

The transformations presented in this paper allow to utilize self-stabilizing algorithms developed for the central daemon scheduler in WSNs. Foremost these are algorithms for coloring, spanning trees, independent and dominating sets [5–9]. Furthermore, this work simplifies the design of self-stabilizing for WSNs, developers can work with the central daemon scheduler and do not have to take into considerations the imponderabilities of wireless communication. The proposed transformations are easy to implement and do not pose much overhead. The result of our simulations and experiments demonstrate for a self-stabilizing MIS algorithm that the transformed algorithm stabilizes quickly. The main drawback of self-stabilizing algorithms for WSNs is the dynamic nature of the communication topology. We conjecture that the transformed algorithms have a very good stabilization behavior if used in conjunction with a stable neighborhood protocol to deal with connections that are susceptible to interference. It remains to analyze the influence of the probability threshold used in the transformations. In future we will investigate criteria for self-stabilizing algorithms operating in dynamic topologies.

References

1. Turau, V., Weyer, C., Witt, M.: Analysis of a Real Multi-hop Sensor Network Deployment: The Heathland Experiment. In: 3^{rd} Int. Conf. on Networked Sensing Systems (INSS06). (2006) 6–13
2. Dijkstra, E.: Self stabilizing systems in spite of distributed control. Communications of the ACM **17**(11) (1974) 643–644
3. Dolev, S.: Self-Stabilization. MIT Press (2000)
4. Schneider, M.: Self-stabilization. ACM Comput. Surv. **25**(1) (1993) 45–67
5. Gradinariu, M., Tixeuil, S.: Self-stabilizing vertex coloration and arbitrary graphs. In Butelle, F., ed.: OPODIS. (2000) 55–70
6. Karaata, M.: A self-stabilizing algorithm for finding articulation points. Theoretical and Mathematical Aspects of Computer Science **10**(1) (1999) 33–46

7. Goddard, W., Hedetniemi, S.T., Jacobs, D.P., Srimani, P.K.: A self-stabilizing distributed algorithm for minimal total domination in an arbitrary system graph. In: IPDPS, IEEE Computer Society (2003) 240–243
8. Chaudhuri, P.: A self-stabilizing algorithm for minimum-depth search of graphs. Information Sciences **118**(1-4) (1999) 241–249
9. Higham, L., Liang, Z.: Self-stabilizing minimum spanning tree construction on message-passing systems. In: Distributed Computing 15th Int. Symposium, Springer LNCS:2180. (2001) 194–208
10. Beauquier, J., Datta, A.K., Gradinariu, M., Magniette, F.: Self-stabilizing local mutual exclusion and daemon refinement. Chicago Journal of Theoretical Computer Science **2002**(1) (2002)
11. Shukla, S., Rosenkrantz, D., Ravi, S.: Observations on self-stabilizing graph algorithms for anonymous networks. In: 2nd Workshop on Self-Stabilizing Systems. (1995) 7.1–7.15
12. Mizuno, M., Nesterenko, M.: A transformation of self-stabilizing serial model programs for asynchronous parallel computing environments. Inf. Process. Lett. **66**(6) (1998) 285–290
13. Herman, T.: Models of self-stabilization and sensor networks. In Das, S.R., Das, S.K., eds.: IWDC. Volume 2918 of LNCS., Springer (2003) 205–214
14. Hedetniemi, S., Hedetniemi, S., Jacobs, D., Srimani, P.: Self-stabilizing algorithms for minimal dominating sets and maximal independent sets. Comput. Math. Appl. **46**(5–6) (2003) 805–811
15. Römer, K., Blum, P., Meier, L.: Time synchronization and calibration in wireless sensor networks. In Stojmenovic, I., ed.: Handbook of Sensor Networks: Algorithms and Architectures. John Wiley & Sons (2005) 199–237
16. ScatterWeb. `http://www.scatterweb.net` (2006)
17. Kulkarni, S.S., Arumugam, U.: Transformations for Write-All-with-Collision Model. In Papatriantafilou, M., Hunel, P., eds.: OPODIS. Volume 3144 of LNCS., Springer (2003) 184–197
18. Herman, T., Tixeuil, S.: A Distributed TDMA Slot Assignment Algorithm for Wireless Sensor Networks. In: ALGOSENSORS. Volume 3121 of LNCS., Springer (2004) 45–58
19. Frank, C., Römer, K.: Algorithms for generic role assignment in wireless sensor networks. In: 3rd ACM Conf. on Embedded Networked Sensor Systems. (2005)
20. Kakugawa, H., Mizuno, M., Nesterenko, M.: Development of self-stabilizing distributed algorithms using transformation: case studies. In: 3rd Workshop on Self-Stabilizing Systems. (1997) 16–30

A Appendix

The proof of the following theorem is omitted due to space limitations.

Theorem 3. *Let D be a set of real numbers strictly between 0 and 1 and (β_i) an infinite sequence with $\beta_i \in D$. Furthermore, let (b_i) be an infinite sequence with $b_i \in \{0,1\}$ and $\mathrm{Prob}(b_i = 1) = \beta_i$ for all $i \geq 0$. For $m \in \mathbb{N}$ let $(1)_m$ be the finite sequence $1, 1, \ldots, 1$, i. e., 1 is repeated m times. Let $P_m(k)$ be the probability that $(1)_m$ is not a subsequence of $b_1, b_2, \ldots, b_k$. If D is a finite set or $\lim_{i\to\infty} \beta_i > 0$ then $\lim_{k\to\infty} P_m(k) = 0$ for all $m \in \mathbb{N}$.*