Mission Statement: Applying Self-Stabilization to Wireless Sensor Networks

Andreas Lagemann, Jörg Nolte
Distributed Systems/Operating Systems
Brandenburg University of Technology Cottbus
Cottbus, Germany
Email: {ae,jon}@informatik.tu-cottbus.de

Christoph Weyer, Volker Turau
Institute of Telematics
Hamburg University of Technology
Hamburg, Germany
Email: {c.weyer, turau}@tu-harburg.de

***Abstract*—Long living and unattended deployments of wireless sensor networks requires fault-tolerant solutions. Self-stabilizing algorithms are providing these properties in an elegant and verifiable way. Recently, a lot of research has been performed to determine appropriate means to apply these promising technique to wireless sensor networks. In this paper the current state of the art in this field is given. Additionally, three major challenges are presented for achieving self-stabilizing sensor networks.**

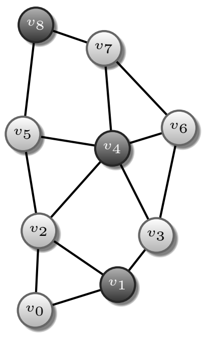
I. Introduction

Wireless sensor networks (WSNs) are operating under inherently instable conditions. The notoriously unreliable wireless communication facilities and environmental influences lead to highly dynamic conditions that in turn lead to frequent communication topology changes and other disturbances. The fact that WSNs usually are intended to run unattended for several months or years necessitates facilities that handle these dynamics in a self-acting manner. The concept of self-stabilization, introduced by Dijkstra in 1974 [1], comprises many properties that are helpful in this context. In fact a self-stabilizing system has embedded mechanisms to react to disturbances and faults in a self-controlled way. These mechanisms provide for the ability of the system to move from any faulty state to a safe one in bounded time. This property is called *convergence* and is augmented by *closure*, which additionally guarantees, that no move made by a self-stabilizing algorithm may lead to a faulty state. For an existing algorithm it can be shown formally, that it possesses both properties and thus is self-stabilizing. Self-stabilization can be regarded as a completely novel approach to fault-tolerance. Instead of specifying faults that may occur and creating algorithms that are robust against these particular faults, simply the set of desired system states is specified and algorithms are formulated such that these states are eventually reached after a fault occurred. This kind of fault-tolerance is also called non-masking fault tolerance, because faults are not (and need not be) detected as such and thus no measures to mask their effects can be taken.

In the following section we will introduce the concept of self-stabilizing algorithms and their application in WSNs. After that we will present three major challenges still to be met for applying self-stabilizing algorithms to WSNs. Finally, we will conclude by giving an outlook of work to be done.

II. Self-Stabilization in WSNs

Figure 1 shows a simple example for a self-stabilizing algorithm: it constructs a maximal independent set (MIS) as described in [2], which can be used as a basis for clustering. The membership of a node in the independent set is indicated by setting the variable `in` to true. Such an algorithm typically consists of a set of rules, which in turn consist of a *guard* (left of $\rightarrow$) and a *statement* (right of $\rightarrow$) that shall be executed when the guard evaluates to true.



$$\begin{gathered} v.in = false \land \forall\, w \in N(v):\ (w.in = false) \\ \rightarrow v.in := true \\ v.in = true \land \exists\, w \in N(v):\ (w.in = true) \\ \rightarrow v.in := false \end{gathered}$$

Fig. 1. Maximal Independent Set [2]

Obviously it is necessary to define the exact semantics of these rules. There are two main questions that come to mind when looking at the algorithm in Fig. 1. The first thing to notice is, that the rules access variables of neighboring nodes. For the formal model often a special shared memory model is used. First of all the variables can be divided in public and private ones. Private variables can solely be accessed by the corresponding node. It is assumed, that each node can read the public variables of its direct neighbors but only the owner of a public variable can write it. To achieve this in WSNs, it is mandatory to provide the nodes with a reasonably stable view of their neighborhood. Therefore, this view has to hide the continuous fluctuations of the wireless link's quality from the algorithm. Mahalle [3] is a neighborhood protocol that achieves this and was developed especially for self-stabilizing algorithms. When this view is established, nodes can exchange the contents of their public variables on a regular basis with their neighbors and maintain a cache for each neighbor. The guards are then evaluated based on the cached values. This communication model was introduced by Ted Herman [4] who called it *cached sensornet transformation*.

The second question is: when are the guards evaluated and the statements executed? First of all execution is assumed to take place in rounds, which seems to imply a synchronous system (which was indeed assumed at first) but the concept of rounds can also be extended to asynchronous systems, when the algorithm does not depend on knowledge about these rounds. Many approaches of self-stabilizing algorithms require, that no two neighboring nodes can execute a statement concurrently. That greatly simplifies the algorithm analysis and the proofs of the self-stabilizing property. The requirements are embodied by an abstract entity called *daemon*. The so called *central daemon* selects exactly one node per round and thus trivially achieves mutual exclusive execution of neighbors. This execution model is of course too restrictive to be applied in a realistic environment. Therefore, a more flexible model, the *distributed daemon* is introduced. Here in each round a subset of size N is selected to make a step concurrently. This model covers the *central daemon* (for $N = 1$) as well as the *synchronous* model where all nodes execute in each round (when N is set to the total number of nodes in the network). A node is called *enabled*, if one of its guards resolves to *true*. When an enabled node is selected, it executes the corresponding statement.

Algorithms developed for a central daemon often do not stabilize under a distributed or synchronous daemon due the concurrent execution within the neighborhood. The MIS algorithm depicted in Fig. 1 is an example of such an algorithm. To solve this so called *transformations* have been proposed [4]–[7]. They convert algorithms designed for such abstract models into semantically equivalent algorithms that stabilize under weaker assumptions.

III. Major Challenges

Applying self-stabilizing algorithms in the field of WSNs to increase the fault-tolerance is currently an active research area [4], [7]–[11]. In this section we will present the most essential challenges for utilizing the benefits of self-stabilization for WSNs.

A. Appropriate Programming Abstractions

One major concern is an appropriate programming abstraction that preserves the simplicity of the algorithms as well as the self-stabilizing properties. It is thereby very desirable to keep the algorithm *description language* independent of WSN specific details like the model transformation applied or the neighborhood protocol used. One major step in this direction is SelfWISE a programming abstraction designed for applying self-stabilizing algorithm in WSNs. It consists of the SelfWISE framework that is the runtime environment for executing self-stabilizing algorithm and a language to express those algorithms. Figure 2 depicts the MIS algorithm presented above in the SelfWISE language. For a more complete description of the SelfWISE framework and language see [10].

SelfWISE allows the application of self-stabilizing algorithms to WSNs by generating appropriate C/C++ code that can be run in a framework. Additionally, transformations that allow to conserve the self-stabilizing properties when communication takes place in a wireless ad-hoc network can be integrated. To fully guarantee preservation of the self-stabilizing properties it will be also necessary to investigate the influence of the compiler and the operating system. Here the development of techniques that preserve self-stabilization is mandatory for accomplishing the goal of completely fault-tolerant WSNs.

Self-stabilizing algorithms use some notion of neighborhood, which is not always merely the 1-hop communication neighborhood but may also span 2-hop communication distance or be defined by other means than communication neighborhood. A programming language for self-stabilizing algorithms must provide appropriate abstractions for such neighborhood notions. These abstraction must be designed such that they allow for efficient implementations with a low memory footprint. One could imagine communication models that provide access to the state of nodes in such alternative neighborhoods. The transformation that implements such an abstraction has to make a trade off between flexibility and energy consumption. To give developers control over this crucial performance aspects the programming abstraction must provide means to specify the desired trade off.

B. Efficient and Scalable Model Transformations

When concerning the communication model, the aforementioned cached sensornet transformation from Herman is widely regarded as appropriate for WSNs. For the execution model several proposals exist for transforming algorithms written for the central daemon such that they can be run in WSNs while self-stabilization is preserved. Transformations of the execution model ensure the exclusive execution within each neighborhood under the distributed or synchronous daemon. The idea behind these transformations is to break the symmetry by using unique identifiers or randomization. A *strict transformation* converts the algorithms in such a way that the execution of the resulting algorithms is equivalent to an execution under the central daemon. An algorithm $\mathcal{A}$ is transformed into $\mathcal{A}'$ such that only one node in each neighborhood performs a move of $\mathcal{A}$ concurrently. Examples for strict transformations are the deterministic conflict manager (CMD) [5] that uses unique node identifiers and BitToss [6]. The latter elects a neighbor by a Bernoulli trial until solely a single node is enabled. The main drawback of these strict transformations is the limited concurrent activity, exactly one node within each

```
algorithm MaximalIndependentSet;
    public bool in;
rule R1:
    in = false and forall(Neighbors v : v.in = false) -> in := true;
rule R2:
    in = true and exists(Neighbors v : v.in = true) -> in := false;
```

Fig. 2. Maximal Independent Set [2]

neighborhood executes its statement. Often this limitation is too restrictive and a higher degree of concurrency is needed. Algorithms converted by a *weak transformation* produce an execution that may not be possible under a central daemon. The reason for this is the fact that nodes may perform a move within a neighborhood concurrently. The idea is that potential deceptive statement executions are resolved after some time, but with the advantage of a faster convergence. Examples for weak transformations are the randomized conflict manager (CMR) [5] and the randomized transformation introduced by Turau and Weyer [7], which both lead to a probabilistic convergence. The latter reference also proposed a transformation that is even self-stabilizing in the case of occasional message losses.

A good metric for the quality of transformations is the average convergence time they yield for different algorithms. It could be shown that the transformation from [7] performs best with respect to average convergence time [12]. Nevertheless, all transformations regarded in [12] rather impose an abstract model on top of a WSN instead of integrating the algorithm more tightly. A rewarding goal would be a lightweight transformation which utilizes the characteristics of the wireless channel to increase efficiency.

C. Application Field of Self-Stabilization in WSNs

Another major concern is to find fields of application in WSNs where the benefits of self-stabilization show to advantage best. The fault tolerance added by self-stabilizing algorithms is the very first property that comes to mind. This alone is not necessarily a sufficient argument for their use, because the gain in fault tolerance must be carefully compared with other approaches to decide if self-stabilization is the method of choice. Methods for assessing the fault tolerance measure of self-stabilization are currently studied and developed (e. g.,see [13]).

But there is more to self-stabilization than fault tolerance. The inherent flexibility of self-stabilizing algorithms seems to be especially well suited to deal with the dynamics of the wireless medium. Due to this the network topology is bound to the changes over time. The knowledge of the network topology is often needed to achieve efficient message transport. The efficiency suffers greatly from topology changes, since with each change the topology must be built newly. Here self-stabilizing algorithms could help to maintain the topology information even in the presence of changes. It is the inherent locality of these algorithms (only operating on their own and their neighbor's states) that promises fast adaptation.

Another interesting aspect is the convergence property of self-stabilizing algorithms. It is shared by other lightweight approaches that aim for achieving scalability. For instance the concept of eventual consistency is a lightweight consistency model first introduced for distributed databases. It does not give guarantees about consistency of copies at every time instance but merely assures that all copies will eventually be consistent, when the time between updates is long enough again. As Gustavsson and Andler point out [14] this approach has several similarities to self-stabilization.

IV. Conclusion

Application scenarios for wireless sensor networks require long living and unattended deployments. These characteristics necessitate fault-tolerant solutions. Self-stabilizing algorithms are an elegant way to develop such applications. We presented the major challenges for the near future, that need to be tackled rendering their utilization in real sensor networks useful. A suitable programming abstraction is vital for allowing a wide range of developers to create self-stabilizing applications. More efficient and lightweight transformations are needed for integrating those algorithms seamlessly into WSNs. Exploiting self-stabilizing properties in other fields than fault tolerance, for achieving a benefit from this elegant paradigm, is another important issue. The greatest challenge will be the integration of self-stabilization into real applications. Therefore, new ideas and experiences with self-stabilizing algorithms in WSNs are tremendously important.

References

[1] E. W. Dijkstra, "Self-stabilizing Systems in Spite of Distributed Control," *Commun. ACM*, vol. 17, no. 11, pp. 643–644, 1974.

[2] S. M. Hedetniemi, S. T. Hedetniemi, D. P. Jacobs, and P. K. Srimani, "Self-Stabilizing Algorithms for Minimal Dominating Sets and Maximal Independent Sets," *Computers and Mathematics with Applications*, vol. 46, no. 5–6, pp. 805–811, Sep. 2003.

[3] C. Weyer, S. Unterschütz, and V. Turau, "Connectivity-aware Neighborhood Management Protocol in Wireless Sensor Networks," in *Proc. 7th GI/ITG KuVS Fachgespräch "Drahtlose Sensornetze" (FGSN'08)*, Berlin, Germany, Jul. 2008.

[4] T. Herman, "Models of Self-Stabilization and Sensor Networks," in *Proc. 5th Int. WS on Distr. Comp. (IWDC'03)*, Dec. 2003.

[5] M. Gradinariu and S. Tixeuil, "Conflict Managers for Self-Stabilization without Fairness Assumption," in *Proc. 27th Int. Conf. on Distr. Comp. Systems (ICDCS'07)*, Jun. 2007.

[6] W. Goddard, S. Hedetniemi, D. Jacobs, and P. K. Srimani, "Anonymous Daemon Conversion in Self-stabilizing Algorithms by Randomization in Constant Space," in *Proc. 9th Int. Conf. on Distr. Comp. and Networking (ICDCN'08)*, Jan. 2008.

[7] V. Turau and C. Weyer, "Fault Tolerance in Wireless Sensor Networks through Self-Stabilization," *Int. Journal of Communication Networks and Distr. Syst.*, vol. 2, no. 1, pp. 78–98, 2009.

[8] H. Kakugawa and T. Masuzawa, "Convergence Time Analysis of Self-Stabilizing Algorithms in Wireless Sensor Networks with Unreliable Links," in *Proc. 10th Int. Symposium on Stabilization, Safety, and Security of Distr. Syst. (SSS'08)*, Nov. 2008.

[9] N. Mitton, E. Fleury, I. Lassous, and B. Sericola, "Fast Convergence in Self-Stabilizing Wireless Networks," in *Proc. 12th Int. Conf. on Parallel and Distr. Syst. (ICPADS'06)*, Jul. 2006.

[10] C. Weyer and V. Turau, "SelfWISE: A Framework for Developing Self-Stabilizing Algorithms," in *Proc. 16th ITG/GI - Fachtg. Komm. in Vert. Syst. (KiVS'09)*, Mar. 2009.

[11] Y. Yamauchi, T. Itou, G. Nishikawa, F. Ooshita, H. Kakugawa, and T. Masuzawa, "Clustering Algorithm for Mobile Ad-Hoc Networks to Improve the Stability of Clusters," in *Proc. IASTED Int. Conf. on Sensor Networks (SN'08)*, Sep. 2008.

[12] C. Weyer, V. Turau, A. Lagemann, and J. Nolte, "Programming Wireless Sensor Networks in a Self-Stabilizing Style," in *Proc. Third Int. Conf. on Sensor Technologies and Applications (SENSORCOMM'09)*, Jun. 2009.

[13] N. Müllner, A. Dhama, and O. Theel, "Derivation of Fault Tolerance Measures of Self-Stabilizing Algorithms by Simulation," in *Proc. 41st Annual Simulation Symposium (ANSS'08)*, April 2008.

[14] S. Gustavsson and F. Andler, Sten, "Self-stabilization and eventual consistency in replicated real-time databases," in *WOSS '02: Proceedings of the first workshop on Self-healing systems*. New York, NY, USA: ACM, 2002, pp. 105–107.