Self-Stabilizing Structures for Data Gathering in Wireless Sensor Networks

Sandra Beyer*, Stefan Lohs*, Jörg Nolte*, Reinhardt Karnapke† and Gerry Siegemund‡

*Distributed Systems/Operating Systems Group, BTU Cottbus-Senftenberg
email: sandra.beyer.sb@gmail.com, {slohs,jon}@informatik.tu-cottbus.de
†Communication and Operating Systems Group, Technische Universität Berlin
email: karnapke@tu-berlin.de
‡Institute of Telematics, Hamburg University of Technology
email: gerry.siegemund@tu-harburg.de

***Abstract*—Wireless Sensor Networks (WSN) enable a number of applications, with monitoring of habitats, office buildings, or restricted areas most prominent among them. All of these applications have one thing in common: the need to communicate. However, the nature of the wireless medium results in quite a few problems. Lossy communication links with transient faults require acknowledgments, retransmissions, and route repair mechanisms. Tree- or similar structures for data gathering scenarios lead to increased load closer to the sink, with congestion, higher buffer space requirements, and energy drain as results. The second problem is often addressed by aggregation and reduction schemes. These schemes are bound to fail, however, when the underlying structure is compromised due to changes in the connectivity between nodes. Therefore, it is necessary to focus on the structures first of all. We address the problem of transient faults by using the inherent fault tolerance of self-stabilizing algorithms when building and using tree- or tiers (communication-) structures. In this paper we show that self-stabilizing structures are suitable for data gathering scenarios in WSN by comparison of the connectivity achieved by our self-stabilizing tiers algorithm and the tree algorithm from Dolev with that of Collection Tree Protocol (CTP), the standard data-gathering protocol for TinyOS.**

Keywords–Wireless Sensor Networks; Self-Stabilization; Routing Structures

I. INTRODUCTION

Wireless Sensor Networks have gained a lot of attention in the research community in the last decade. Application scenarios include for example monitoring of habitats, intrusion detection and house control. This strong interest in sensor networks stems from the fact that they are inexpensive, autonomous systems that have to adapt to ever changing conditions. Due to the fact that sensor networks should be inexpensive, the individual sensor nodes are usually not very powerful but can cooperate to solve complex tasks. In some applications, the nodes need to operate autonomously for many years, because they are deployed in hard to reach areas and human interaction is restricted to the absolute minimum. The restrictions imposed by pricing and form factor result in the need for special protocols, as standard protocols induce too much (computational, communication or memory) overhead.

Sensor nodes are usually powered by batteries, energy harvesting is only rarely possible. Therefore, energy is an important resource and must be conserved as much as possible. In combination with the price factor, nodes should often be as cheap as possible, this leads to the usage of transceivers that require only little energy but can also only transmit over short distances. This in turn makes the usage of multi-hop communication protocols necessary. Radio communication is always error prone due to the shared medium, collisions and environmental influences. The fact that most sensor nodes only feature a cheap transceiver intensifies this problem. To enable the sensor networks to perform their duties in spite of these errors, repair mechanisms are included. These usually encompass forward error correction, acknowledgments, and retransmissions. However, detecting the errors and storing messages for retransmission increases memory consumption (flash and RAM). The amount required can be arbitrarily large, depending on the number of different kinds of errors that should be detected and the correction mechanisms. Also, the error handling code itself might introduce a new source of errors. To overcome this problem, we propose the usage of self-stabilization, which describes only the 'good' states of the sensor nodes. Any other state is an error state and needs to be changed. This way, there is no need for the programmer to examine all possible error-states. We show that our self-stabilizing protocols are less complex and the memory consumption is reduced when compared to traditional protocols.

The most common applications for sensor networks feature data gathering scenarios, in which all nodes gather their data and transmit it to a central sink from which a user can retrieve it. In a multi-hop environment, the nodes need to forward more messages the closer they are to the sink. This increased communication load leads to congestion, a higher rate of collisions and increased RAM consumption on the nodes close to the sink as they need to store more messages. Also, the need to communicate more results in a change in duty cycle, as those nodes need to stay awake longer, which in turn leads to a higher energy consumption and early node failure.

A common way to deal with this problem is to use aggregation and/or reduction mechanisms on the sensor nodes. Aggregations reduces the number of messages, while reductions reduce their size. This can be useful even for applications with long intervals between message generation where congestion is not a problem, as it also reduces the consumed transmission energy. In order for the aggregation and reduction to be effective, the underlying communication structure needs to operate correctly. Without aggregation and reduction, a lost message results in a single lost value. With aggregation and reduction, this could mean the loss of the data from a whole subnet. In this paper we focus on mechanisms that can be used to build structures that lie underneath the aggregations and

 ISBN: 978-1-61208-425-1

reductions. To build and, most of all, sustain these structures, we rely on self-stabilizing algorithms.

This paper is structured as follows: Related work is presented in Section II, followed by an introduction to self-stabilization in Section III. Our self-stabilizing tiers algorithm (SelfTIER) and the self-stabilizing tree algorithm from Dolev [1] (SelfTREE) are discussed in Section IV and evaluated in comparison with CTP in Section V. We finish with a conclusion in Section VI.

II. Aggregation in WSN

In literature, several approaches for in-network aggregation are presented. All have in common that they first establish a special communication structure and afterward use it for a certain time. The structure is responsible for deciding which node needs to aggregate the data from which other nodes and which path the aggregated data takes on its way to the sink. Also, the times for aggregations are defined. When a node wants to forward the aggregated values, it first needs to wait for all its children to transmit their data. The structure can also include information about deadlines and whether or not all messages from children have arrived. Most aggregation algorithms focus on the error free case and describe the aggregation itself, without offering too much details about the underlying structure and its repair mechanisms. In the following we take a look at three categories of aggregation structures, namely tree based, cluster based, and multi path. Others approaches, which fall into neither or multiple of the categories mentioned above, are also discussed briefly.

A. Tree Based

A tree is the most commonly used routing scheme for aggregation in sensor networks. The data is routed from source nodes to a sink node. Intermediate nodes (parent) are able to collect the data of well defined children and apply a fusion function before forwarding data to the next hop. Using the well organized aggregation structure, each node only forwards one packet per aggregation round, which avoids high network loads and conserves energy. The drawback is that if one packet is lost, the data of the whole sub-tree is lost.

An aggregation scheme for monitoring applications is Tiny AGgregation (TAG) [2]. TAG uses a two phase approach. First, the sink sends a query message to build an aggregation tree. All nodes receiving this message adapt their level (the distance to the sink) and select the sender as parent node. In the second phase, the collection phase, the tree is used to aggregate data and forward it to the sink node. TAG uses a *per-hop aggregation* approach [3], each parent has to wait until all children have sent their data before forwarding the aggregated data. Each round is divided into slots, the number of slots equals the height of the tree. This way, timeouts can be calculated, in case a parent does not receive the messages from all its children. The query message is sent periodically to recover the tree after dynamic link changes.

There are several other approaches that can be used to build aggregation trees, which focus on, e.g., energy consumption, path reliability and/or mobility of nodes [4]-[8].

B. Cluster Based

The second group of schemes are cluster based aggregation schemes. Like tree based algorithms, cluster based algorithms also organize the network in a hierarchical manner. Each selected cluster head is responsible for aggregating the data of its cluster members and forwarding it to a sink node.

LEACH [9] is one member of this group. LEACH is divided into a setup phase to organize the clusters and a steady-state phase to send the data to the sink. The algorithm is distributed. At first, cluster heads are selected by a probabilistic approach. After that, the cluster head computes a TDMA (Time Division Multiple Access) scheme to avoid collisions in its cluster. Cluster heads send their aggregated data with one single transmission to the sink.

Other members of this group include COUGAR [10] and DRINA [11]. They use different mechanisms to elect the cluster head, to reduce communication overhead or select nodes by different suitable metrics.

C. Multi Path

The idea of multi path approaches is to increase the tolerance against link changes by using multiple paths to the sink node. If a link breaks or one packet is lost because of a collision, the data can be restored using packets that traveled an alternative path.

A member of this group is Synopsis Diffusion [12]. Synopsis Diffusion uses a tier structure as the underlying topology, where each node forwards all messages it receives from nodes on a higher tier. The advantage of this approach is that data is forwarded on multiple paths to the sink node but it also introduces the problem of *duplicate sensitive* aggregation. Nodes may receive duplicates of data which may affect the result of the aggregation. This problem must be solved by the use of suitable aggregation functions. Nevertheless, this approach is more appropriate in harsh environments with a high rate of link breaks and message loss.

D. Other Approaches

Several other approaches exist, whose underlying topologies do not match the previous categories. One Example is the chain based structure of PEGASIS [13], which is focused on conserving energy. Another example is the hybrid approach of Tributaries and Deltas [14], which combines the advantages of tree- and multi path based schemes.

E. Summary

All schemes have in common that they are divided into a setup- and a collection phase. Often the underlying algorithm that is used to establish a structure is exchangeable or can be extended by additional repair mechanisms to increase fault tolerance, periodic rebuilds are the most often suggested approach.

III. Self-Stabilization

The concept of self-stabilizing algorithms was first introduced by Dijkstra in his paper "Self-Stabilizing Systems in Spite of Distributed Control" [15]. He described a network of several processors having a set of registers. Each processor has a so called local view, which consist of the registers of its direct neighbors.

A self-stabilizing algorithm consists of a set of rules in the form $guard \rightarrow assignment$. If the guard predicate of a rule is resolved to *true*, the rule is called *enabled* and

 ISBN: 978-1-61208-425-1

```
1 NodeID parent;
2 Integer level;
```

Figure 1. State of the SelfTREE algorithm

```
1 Integer level;
```

Figure 2. State of the SelfTIER algorithm

the assignment part *may* be executed. Every processor checks (locally) whether any of the rules is enabled, based on its local view. There are two properties necessary for an algorithm to be self-stabilizing: *Convergence* and *Closure*. *Convergence* means that the system will reach a defined (stable) state within finite time, while *Closure* means the execution of any rule will never take the system from a stable state to an unstable one.

In wireless sensor networks, each node represents one of Dijkstra's processors. The local view of a processor then corresponds to the information from that node and all its neighboring nodes, meaning that the state of a node needs to be communicated. Nodes that receive such a state message may change their own state due to the execution of rules, based on the changes in the state of the neighbor. As state changes propagate through the network, the whole system eventually reaches a stable state, making the system inherently tolerant against transient faults. Exchanging the state messages can make good use of the broadcast character of the medium. Another advantage of the self-stabilizing approach is that the algorithm (eventually) turns the system into a stable state from any given state, meaning that no code for initialization is required.

IV. SELFTREE AND SELFTIER

In this section we describe two self-stabilizing algorithms for a data gathering scenario. Each node of the sensor network has a unique identfier. The goal is to build and maintain a routing topology where all nodes can send their measured data to one distinct node, i.e., the sink. Of course, nodes that are separated from the network will not find a route. The first algorithm builds a minimum spanning tree (SelfTREE) published by Dolev [1] in 2000. The second algorithm is our self-stabilizing tiers algorithm (SelfTIER). Both algorithms consist of only two rules. The first rule of each algorithm is executed by the node which requested the aggregated data (the sink). The second rule is run by all other nodes.

A. SelfTREE

In the SelfTREE algorithm, the state of a node consists of the ID of its parent node and its distance from the sink (Listing 1). Please note that both values can start with any arbitrary value. The state of a node is propagated to its neighbors on a regular basis, and may result in enabled rules on the receiving nodes, which might also lead to state changes.

The first rule of SelfTREE applies only to the sink node. It sets the parent ID to an invalid value and the level as 0. The second rule is applied to all non-sink nodes. Based on the local view, the node checks the distance from the sink of all its neighbors. If the lowest distance from the sink is lower then its own distance minus one, a shorter path to the sink seems to exist. Therefore, the node sets its parent ID to the node with the minimum distance and its own distance as that nodes distance incremented by one. Please note that if there are multiple candidates with the minimum distance, the new parent can be any arbitrary one of them.

A node is in a stable state if it has correct knowledge of the distance to the sink and a correct parent node. A global stable state/ stable system state is reached when all nodes are in a stable state.

When data transmission starts, each node knows its parent node and addresses its message to that parent. Nodes only forward messages in which they are listed as next hop.

B. SelfTIER

In the SelfTIER algorithm, only the distance to the sink is required and included in the state of a node (Listing 2). As in all self-stabilizing algorithms, the state of a node is shared with direct neighbors on a regular basis and may lead to changes on the receiving nodes.

The two rules of the SelfTIER algorithm are fairly similar to those of the SelfTREE algorithm. Rule one is only applied to the sink and sets its level to 0. The second rule is only applied to non-sink nodes. Each node checks the levels of its neighbors and sets its own level to the minimum incremented by one.

In such a tier based structure, only the distance to the sink is relevant for routing decisions. When a node receives a data message, it checks from which tier the message was transmitted. If the tier is higher, i.e., the distance to the sink was higher, the message is forwarded. Otherwise, the message is either from a node on the same level and will be forwarded on the next lower level, or it is from a node that is closer to the sink. Either way, it must not be forwarded. This way, all nodes that are closer to the sink than the transmitting node forward the message, resulting in multiple redundant paths taken by the same message. This increases robustness, but also increases network load and gives rise to a potential problem with duplicates. Depending on the aggregation/reduction scheme that is used on top of the tier structure, this can be ignored (e.g. reduction function: maximum) or must be detected (e.g. reduction function: add).

The big advantage of the tier based algorithm is its redundancy. If the link between a node and its parent breaks in the tree structure, data from this node and all its children will be unavailable until the structure is rebuilt. In the tiers structure, an arbitrary number of links may break without affecting the forwarding of messages, as long as there is still at least one node available in a lower tier for each node.

V. EVALUATION

Both self-stabilizing algorithms have been integrated into the TOLERANCEZONE middleware. It provides a neighborhood discovery protocol and chooses bidirectional neighbors for each node. Also, it takes care of the regular propagation of a nodes state to its neighbors. To reduce the network load, the state and neighborhood messages are transmitted together. The speed with which an algorithm can react to changes in

 ISBN: 978-1-61208-425-1

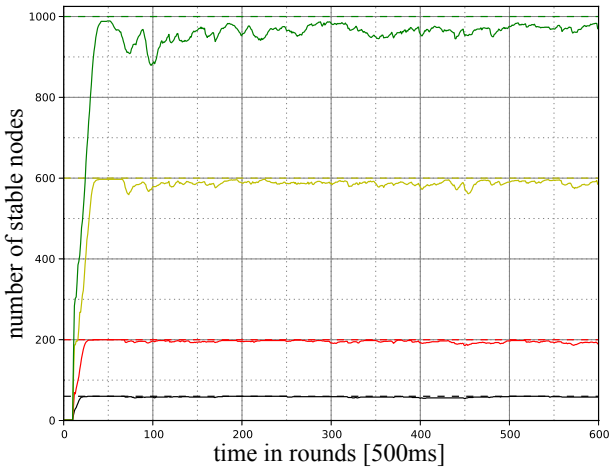

Figure 3. Structure state over time. SelfTIER algorithm with varying number of nodes (60, 200, 600, 1000).

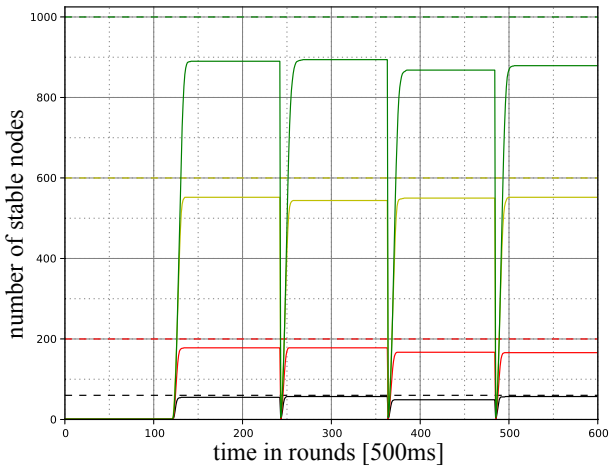

Figure 4. Structure state over time. Ordinary tiers algorithm with varying number of nodes (60, 200, 600, 1000).

a nodes neighborhood depends on the frequency of message transmission. As changes may need to propagate through the whole network, reaching a stable state after a change occurred may take $\mathcal{O}(maxLevel)$ rounds.

The goal of our self-stabilizing algorithms is to increase the tolerance against transient faults. When a transient fault occurs, the system is in a non-stable state and the underlying routing topology must be repaired. In case of self-stabilization, this is done autonomously, in the other cases we have to reconstruct or repair our structure.

Stability

In our first evaluation we compared the SelfTIER algorithm with an ordinary algorithm as proposed in TAG. In the ordinary approach, the sink periodically floods a beacon packet to establish a valid tier structure. Both algorithms were implemented in C++ for the REFLEX operating system [16]. The SelfTIER algorithm is part of our TOLERANCEZONE middleware.

In the first part of the evaluation we measured the stability of the network. In simulations using the discrete event simulator OMNeT++ [17], we ran both algorithms in a grid topology with 60, 200, 600, and 1000 nodes. Each second, a snapshot of the network was taken and the amount of stable nodes was counted. For the SelfTIER algorithm, a node was counted as stable if no rule was enabled. In case of the ordinary tier algorithm, it is counted as stable if it received and forwarded the beacon. To simulate transient faults we introduced link breaks. In case of the network consisting of 1000 nodes, one node lost connections for 30 seconds every second, after this time the connection was reestablished. The probability of a fault at a specific node was the same for all simulated network sizes.

Figure 3 and Figure 4 show the result of our simulations for SelfTIER and the ordinary algorithm respectively. As Figure 3 shows, the SelfTIER algorithm reaches nearly 95% stability for all four different network sizes in spite of the transient faults, while the ordinary algorithm reaches at most 90%.

In the case of the ordinary algorithm, a link break during the construction of the structure results in loss of the construction beacon. Then, the corresponding node is not part of

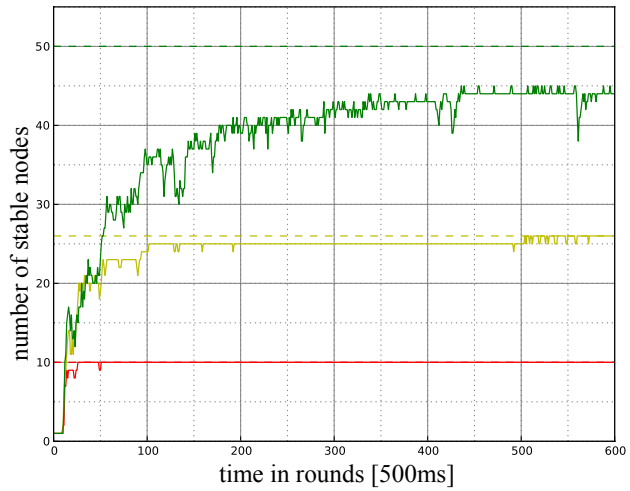

Figure 5. Real world deployment. Structure state over time. SelfTIER algorithm with varying number of nodes (10,26,50).

the structure until it receives the next beacon. In the case of the self-stabilizing algorithm, the node automatically rejoins the tiers structure once the link break is over and the node is connected to its neighbor(s), again.

After the simulations, we also ran experiments with real sensor nodes. We used EZ-Chronos 430 sensor nodes from Texas Instruments and placed them in grids of different sizes on the ground with a distance of 3 meters between nodes. The grids always consisted of a square of nodes plus the sink node, resulting in 10 (3x3 +1), 26 (5x5 +1) and 50 (7x7 +1) nodes used in the experiments. In real world experiments, inducing errors is not easy but also not necessary, as the wireless channel is lossy enough on its own. We measured the stability of the structures generated by both algorithms, using the same nodes and the same placement of nodes to keep the results comparable.

Figures 5 and 6 show the stability of the routing structure measured for 3 minutes after the (re-)start of the network. When the two Figures are compared, it can be seen that the

 ISBN: 978-1-61208-425-1

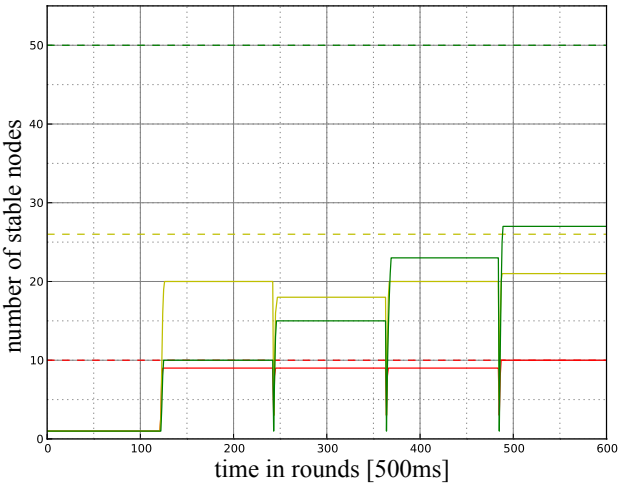

Figure 6. Real world deployment. Structure state over time. Ordinary tier algorithm with varying number of nodes (10,26,50).

size of the network has a strong influence, especially on the ordinary algorithm. Both algorithms perform similar in the network consisting of only ten nodes. SelfTIER can reach a stable structure for all ten nodes almost immediately but the ordinary algorithm reaches nine stable nodes after a rebuild and ten near the end of the three minutes. This could be compensated by allowing for a certain setup time. However, the differences increase when the network size is increased. In the network consisting of 26 nodes, SelfTIER still reaches 95% stability while the ordinary algorithm reaches only about 84%. For the network consisting of 50 nodes it even falls below 60%, while SelfTIER still reaches nearly 90%. As the number of additional nodes that are stable does not rise much for the ordinary algorithm, we expect that there is a certain threshold, above which adding more nodes to the network will not increase the number of stable nodes anymore, for the ordinary algorithm. For the self-stabilizing SelfTIER, however, adding nodes will delay the point in time when most nodes are stable, but eventually this point will be reached, making SelfTIER the algorithm of choice for larger networks.

Connectivity

The second part of our evaluation concerns the quality of the structure. To achieve good aggregation results, as many nodes as possible must know a valid route to the sink. In a data gathering scenario, the data is forwarded hop by hop from all nodes to the single sink. To evaluate the connectivity of the routing structure, we use snapshots of the state of the routing protocol, in addition to the entries in the neighbor tables and the physical layer. In the simulations using OMNeT++, we use the current links between the NIC-modules.

As competitor to the self-stabilizing algorithms, we used a REFLEX implementation of the standard TinyOS data gathering protocol CTP.

CTP uses beacons, which are transmitted by each node regularly, to build a tree structure. After an initial buildup phase the frequency of beacons is reduced. Instead, the quality of connections is measured using acknowledgment messages. To compensate the lack of application messages

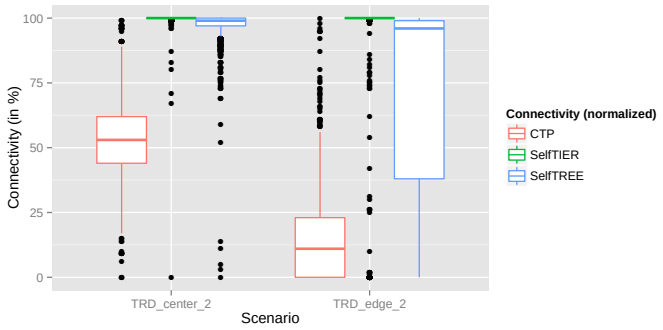

Figure 7. Connectivity of 226 nodes grid. Link disconnect fault injection. Sink node at center (left) and edge (right).

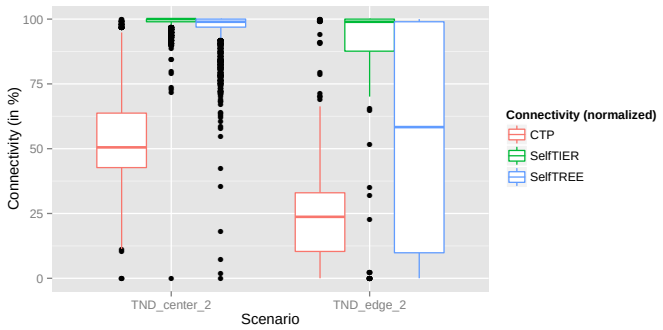

Figure 8. Connectivity of 226 nodes grid. Node disconnect fault injection. Sink node at center (left) and edge (right).

and acknowledgments in our simulations, we kept a constant beacon interval of 500ms, which equals the frequency of status message transmission for the self-stabilizing algorithms. We ran OMNeT simulations with up to 226 (15x15 +1) nodes in a grid topology.

The connectivity achieved by all three algorithms when link failures are injected can be seen in Figure 7. In this scenario, one link is disconnected every second, and remains disconnected for 30 seconds. The Figure is divided according to the placement of the sink, either at the center of the grid (left) or at the corner (right). Connectivity is achieved by a node, if it has a path to the sink (routing table), the next hop is in its neighbor table and the physical connection to that neighbor exists. The Figure shows that a placement of the sink in the middle of the network is better for CTP, which then reaches about 50% connectivity. For this placement, there is no big difference between the SelfTREE algorithm from Dolev and our SelfTIER, both reach nearly 100%. When a sink placement at the corner is evaluated, the performance of CTP drops drastically, to roundabout 12%. The results of SelfTREE are spread a little further, but it still achieves more than 95% connectivity. The results from SelfTIER stay at nearly 100% connectivity.

Figure 8 shows the results the three algorithms achieved for the scenario in which complete node disconnection faults were injected. Every 5 seconds, a node is disconnected from the network for 30 seconds. The Figure shows the normalized result with respect to the physical conditions. A node is counted as connected, if there is a valid entry for a path from

 ISBN: 978-1-61208-425-1

the node to the sink in the routing- and neighbor table on the node, and the used physical links exists. The scenario is further divided according to the placement of the sink node: Either at the center of the grid or at the corner. Placing the sink at the corner naturally doubles path length, compared to the placement at the center. When the sink is placed at the center (Figure 8, left side), the median of reached connectivity for CTP is 50%. Both self-stabilizing algorithms, the SelfTREE algorithm from Dolev and our SelfTIER algorithm, reach nearly 100 % connectivity. When the sink is located at the edge of the network, the performance of CTP and SelfTREE decrease to 25% and 55% respectively, while SelfTIER still offers nearly 100% connectivity. This is due to the fact that SelfTIER used multiple paths to deliver messages. While the failure of one node results in lost connectivity for a whole sub-tree in the tree based algorithms, SelfTIER can usually connect the nodes from lower tiers through different nodes, so only the node on which the failure occurs is 'lost'.

In summary, it can be said that self-stabilizing algorithms are a good choice when building structures for aggregation and/or reduction schemes, with our SelfTIER algorithm outperforming CTP and the SelfTREE algorithm from Dolev when not only links break, but nodes can also fail completely for a certain time.

VI. Conclusion

In this paper, we have discussed the prerequisites for a successful aggregation/reduction scheme for data gathering scenarios in wireless sensor networks, namely a robust underlying communication structure. We have argued that self-stabilizing algorithms should be the method of choice, and support our claim with simulations and experiments. In the evaluation we measured stability and connectivity for different structures and network sizes. We compared two different self-stabilizing algorithms with CTP, the standard protocol for data gathering scenarios from TinyOS. The results show that there are performance differences between the two self-stabilizing algorithms that depend on the error scenario and the network diameter, but both protocols always perform better than CTP.

Often, self-stabilizing algorithms are criticized for the higher energy consumption due to the periodic state exchange between nodes. How these exchanges can be reduced without sacrificing too much of the advantages of self-stabilization is currently being evaluated by our group. Sleeping times/duty cycling and other traditional approaches for energy conservation are being discussed, and first evaluations show promising results.

Acknowledgment

This work was partially funded by the Deutsche Forschungsgemeinschaft DFG in the project ToleranceZone (DFG NO 625/6-1).

References

[1] S. Dolev, Self-stabilization. MIT press, 2000.

[2] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "Tag: A tiny aggregation service for ad-hoc sensor networks," SIGOPS Oper. Syst. Rev., vol. 36, no. SI, Dec. 2002, pp. 131–146. [Online]. Available: http://doi.acm.org/10.1145/844128.844142

[3] I. Solis and K. Obraczka, "The impact of timing in data aggregation for sensor networks," in Communications, 2004 IEEE International Conference on, vol. 6, June 2004, pp. 3640–3645 Vol.6.

[4] M. Ding, X. Cheng, and G. Xue, "Aggregation tree construction in sensor networks," in Vehicular Technology Conference, 2003. VTC 2003-Fall. 2003 IEEE 58th, vol. 4, Oct 2003, pp. 2168–2172 Vol.4.

[5] H. S. Kim, T. F. Abdelzaher, and W. H. Kwon, "Minimum-energy asynchronous dissemination to mobile sinks in wireless sensor networks," in Proceedings of the 1st International Conference on Embedded Networked Sensor Systems, ser. SenSys '03. New York, NY, USA: ACM, 2003, pp. 193–204. [Online]. Available: http://doi.acm.org/10.1145/958491.958515

[6] T. Banerjee, K. Chowdhury, and D. Agrawal, "Tree based data aggregation in sensor networks using polynomial regression," in Information Fusion, 2005 8th International Conference on, vol. 2, July 2005, pp. 8 pp.–.

[7] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection tree protocol," in Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems, ser. SenSys '09. New York, NY, USA: ACM, 2009, pp. 1–14. [Online]. Available: http://doi.acm.org/10.1145/1644038.1644040

[8] H. O. Tan and I. Körpeoğlu, "Power efficient data gathering and aggregation in wireless sensor networks," SIGMOD Rec., vol. 32, no. 4, Dec. 2003, pp. 66–71. [Online]. Available: http://doi.acm.org/10.1145/959060.959072

[9] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "An application-specific protocol architecture for wireless microsensor networks," Wireless Communications, IEEE Transactions on, vol. 1, no. 4, Oct 2002, pp. 660–670.

[10] Y. Yao and J. Gehrke, "The cougar approach to in-network query processing in sensor networks," SIGMOD Rec., vol. 31, no. 3, Sep. 2002, pp. 9–18. [Online]. Available: http://doi.acm.org/10.1145/601858.601861

[11] L. Villas, A. Boukerche, H. Ramos, H. de Oliveira, R. de Araujo, and A. Loureiro, "Drina: A lightweight and reliable routing approach for in-network aggregation in wireless sensor networks," Computers, IEEE Transactions on, vol. 62, no. 4, April 2013, pp. 676–689.

[12] S. Nath, P. B. Gibbons, S. Seshan, and Z. R. Anderson, "Synopsis diffusion for robust aggregation in sensor networks," in Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems, ser. SenSys '04. New York, NY, USA: ACM, 2004, pp. 250–262. [Online]. Available: http://doi.acm.org/10.1145/1031495.1031525

[13] S. Lindsey, C. Raghavendra, and K. Sivalingam, "Data gathering algorithms in sensor networks using energy metrics," Parallel and Distributed Systems, IEEE Transactions on, vol. 13, no. 9, Sep 2002, pp. 924–935.

[14] A. Manjhi, S. Nath, and P. B. Gibbons, "Tributaries and deltas: Efficient and robust aggregation in sensor network streams," in Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, ser. SIGMOD '05. New York, NY, USA: ACM, 2005, pp. 287–298. [Online]. Available: http://doi.acm.org/10.1145/1066157.1066191

[15] E. W. Dijkstra, "Self-stabilizing systems in spite of distributed control," Commun. ACM, vol. 17, no. 11, Nov. 1974, pp. 643–644. [Online]. Available: http://doi.acm.org/10.1145/361179.361202

[16] K. Walther, R. Karnapke, and J. Nolte, "An existing complete house control system based on the reflex operating system: Implementation and experiences over a period of 4 years," in Proceedings of 13th IEEE Conference on Emerging Technologies and Factory Automation, 2008.

[17] A. Varga, "The omnet++ discrete event simulation system," in Proceedings of the European Simulation Multiconference (ESM'2001), Prague, Czech Republic, Jun. 2001.

 ISBN: 978-1-61208-425-1