Gossip-Based Self-Management of a Recursive Area Hierarchy for Large Wireless SensorNets

Konrad Iwanicki, *Student Member*, *IEEE*, and Maarten van Steen, *Senior Member*, *IEEE*

Abstract—A recursive multihop area hierarchy has a number of applications in wireless sensor networks, the most common being scalable point-to-point routing, so-called hierarchical routing. In this paper, we consider the problem of maintaining a recursive multihop area hierarchy in large sensor networks. We present a gossip-based protocol, dubbed PL-GOSSIP, in which nodes, by using local-only operations and by periodically gossiping with their neighbors, collaboratively maintain such a hierarchy. Since the hierarchy is a complex distributed structure, PL-GOSSIP introduces special mechanisms for internode coordination and consistency enforcement. Yet, these mechanisms are seamlessly integrated within the basic gossiping framework. Through simulations and experiments with an actual embedded protocol implementation, we demonstrate that PL-GOSSIP maintains the hierarchy in a manner that addresses all the peculiarities of sensor networks. More specifically, it offers excellent opportunities for aggressive energy saving and facilitates provisioning energy harvesting infrastructure. In addition, it bootstraps and recovers the hierarchy after failures relatively fast while also being robust to message loss. Finally, it can seamlessly operate on real sensor node hardware in realistic deployment scenarios and can outperform existing state-of-the-art hierarchy maintenance protocols.

Index Terms—Hierarchical routing, area hierarchy, gossiping, gossip-based algorithms, self-organization, wireless sensor networks.

1 INTRODUCTION

NUMEROUS application proposals of wireless sensor networks (WSNs) assume large numbers of sensor nodes that collaboratively collect and process data from vast geographic regions. Sensor nodes are often severely constrained in terms of resources. It is therefore crucial to organize them in a way that enables scalable addressing and routing, two key features necessary for scalable data collection and querying. Constructing and maintaining this scalable organization should preferably require only minimal human intervention.

A compelling example of such an organization is a recursive area hierarchy [1], [2], [3], [4], [5], [6], [7]. A recursive area hierarchy constitutes a multilevel overlay on the physical network topology in which at subsequent levels nodes are grouped into exponentially larger areas: at level 0, nodes form their own singletons; at level 1, connected singletons are grouped into areas; at level 2, the areas are grouped into superareas, and so forth. Such hierarchical grouping enables addressing and routing that necessitate only polylogarithmic node state, and hence, are highly scalable. Furthermore, the addressing and routing can be employed to build more advanced services such as distributed hash tables [4], [6] or multiresolution in-network aggregation and querying [5], [8], [7]. Finally, the hierarchy can be constructed and maintained autonomously by the nodes, without human intervention, thereby minimizing the deployment and upkeep costs of the network. Because of these merits, a recursive area hierarchy can be a foundation of a plethora of applications proposed for WSNs. Examples of such applications include object tracking [4], [9] for asset management, reactive tasking [10] for "smart" buildings and disaster containment, scalable network monitoring [11], [12] for problem diagnosis, and multiresolution in-network storage [7], [13], [14] for monitoring buildings, microclimate, and crops, to name a few. All in all, a recursive area hierarchy is an important network organization for large WSNs.

For these reasons, the maintenance of an area hierarchy is a fundamental problem. It is challenging due to the following properties of WSNs and the fact that many of them put conflicting requirements on hierarchy maintenance protocols:

- *Tight energy budget.* Since sensor nodes operate on batteries or by harvesting ambient energy, their energy budgets are typically extremely tight. Consequently, a hierarchy maintenance protocol must offer opportunities for aggressive energy saving in order to reduce the network upkeep costs. In addition, in an energy harvesting network, the design of the protocol should facilitate provisioning the energy harvesting infrastructure (e.g., choosing the size of solar cells). This implies that the protocol should not create bursts of energy consumption that could overrun the energy budget for a given time period. Instead, the energy consumption of the protocol should be more or less even and relatively easy to estimate [15].
- *Failures and connectivity changes.* Due to their embedded nature and energy constraints, WSNs experience node failures and connectivity changes. The hierarchy maintenance protocol should provide fast recovery after such events as well as reasonable hierarchy bootstrap times. This is crucial to minimize

• *The authors are with the Department of Computer Science, Vrije Universiteit Amsterdam, De Boelelaan 1081A, 1081 HV Amsterdam, The Netherlands. E-mail: {iwanicki, steen}@few.vu.nl.*

Manuscript received 26 June 2008; revised 24 Mar. 2009; accepted 22 May 2009; published online 28 May 2009.
Recommended for acceptance by M. Oulk-Khaoua.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2008-06-0244.
Digital Object Identifier no. 10.1109/TPDS.2009.89.

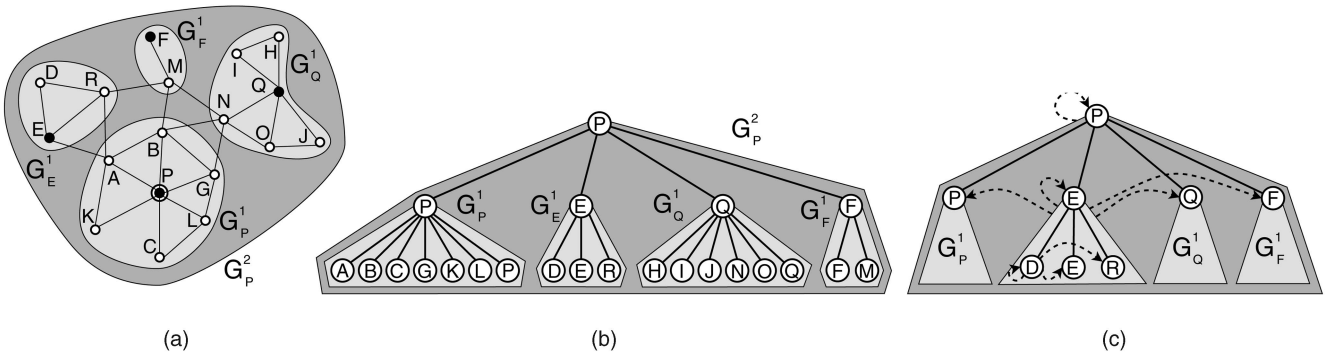

Fig. 1. An example of a group hierarchy. (a) Group hierarchy: the singleton level-0 groups are omitted for clarity. (b) Node labels presented as a tree: a node's label is obtained by concatenating group head identifiers from the leaf representing the node to the root of the tree. (c) Routing table of node D: the routing table contains entries for the node's own groups and for the sibling groups at all hierarchy levels.

the disruption of the application using the hierarchy in the presence of such events. However, fast recovery typically implies higher energy consumption and vice versa.

- *Message loss.* Low-power wireless communication employed by sensor nodes is subject to (sometimes high) message loss, both anticipated, resulting from signal fading with distance, and varying, due to transmission collisions and noise. Consequently, the protocol must inherently assume unreliable communication in its design rather than relying on nearly perfect communication.
- *Severe resource constraints.* To minimize energy consumption and costs, sensor nodes are extremely constrained in terms of memory, bandwidth, and processing power. Therefore, the hierarchy maintenance protocol must be practical, that is, it must run on the real hardware.

In this paper, we attack the problem of maintaining a recursive multihop area hierarchy taking all these peculiarities of large WSNs into account. We propose a novel hierarchy maintenance protocol, dubbed PL-GOSSIP,[1] which is based on asynchronous neighborhood gossiping [17], [18]. In essence, each node, in an endless process, periodically broadcasts its local protocol state to its neighbors (i.e., the nodes within its radio range). Likewise, it periodically receives the state of every neighbor, which it merges with its own local state. The merged state is broadcast in the node's messages in subsequent periods, which allows for propagating information throughout the network. This primitive operation pattern is sufficient for nodes running PL-GOSSIP to self-organize into, and collaboratively maintain such a complex distributed data structure as a recursive area hierarchy. More importantly, however, the protocol addresses all the aforementioned issues. Its well-defined periodic operation pattern and local-only traffic offer excellent opportunities for aggressive energy saving and, in addition, facilitate provisioning an energy harvesting infrastructure. The protocol provides fast failure recovery and hierarchy bootstrap that can be configured for a desired energy consumption. In addition, it is robust to message loss and works on real hardware. We substantiate these claims with simulations and experiments with actual embedded implementations.

1. The "PL" in "PL-GOSSIP" now stands for "Polish" [16] as K. Iwanicki is Polish. Originally, it had a different meaning, though.

The rest of the paper is organized as follows: We begin by surveying existing protocols for hierarchy maintenance in WSNs in Section 2. Then, in Section 3, we discuss the basic idea behind PL-GOSSIP and explain how it addresses the issues involved in those protocols. We go on to give the details of our protocol in Section 4 and evaluate it using simulations and an implementation in Section 5. Finally, we come to conclusions in Section 6. All necessary proofs and code listings are attached as supplemental material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2009.89.

2 Background and Prior Work

2.1 Basic Terms and Definitions

Area hierarchy [2] is a recursive multilevel logical overlay; at level 0, nodes form singletons; at level 1, connected singletons are merged into groups; at level 2, the groups are organized into supergroups, and so on at higher levels (see Fig. 1a). Every node belongs to exactly one group at each level, with level-0 singleton groups that correspond to individual nodes and one top-level group that contains all nodes. The groups at subsequent levels cover exponentially growing network areas, which implies that the number of levels (i.e., the height of the hierarchy) can be polylogarithmic with respect to the number of nodes. In addition, recursiveness entails that each nontop level-i group is completely nested in exactly one level-$i{+}1$ group, that is, all members of the same level-i group are also members of the same level-$i{+}1$ group.

Each group has a special *head node* that is typically used for identifying the group and for maintaining the membership of the group in the hierarchy. Since each node has a unique identifier, a group is unambiguously identified by its level and the identifier of its head node. In the remainder of this paper, we write G^i_X to denote a level-i group with head node X. In the example from Fig. 1a, node P (marked with a double circle) is a level-2 head, as it is the head of groups G^0_P, G^1_P, and G^2_P. Node E (black circle) is a level-1 head as it is the head of groups G^0_E and G^1_E. Finally, since node D (empty circle) is only the head of group G^0_D, it is a level-0 head.

The group hierarchy is reflected in the *labels* of the nodes. A node's label is a concatenation of the group head identifiers for all the groups the node is member of, starting from level 0 (see Fig. 1b). For instance, the label of node D

from Fig. 1, which is a level-0 group head, is $L(D) = D.E.P$ as node D belongs to groups G_D^0, G_E^1, and G_P^2. The label of node E, a level-1 group head, is $L(E) = E.E.P$ because E belongs to G_E^0, G_E^1, and G_P^2. Finally, the label of node P, a level-2 group head, is $L(P) = P.P.P$ as P belongs to G_P^0, G_P^1, and G_P^2. The label of a node constitutes the *routing address* of the node.

Based on their labels, the nodes also keep *hierarchical routing tables*, which are used for maintaining the hierarchy and implementing routing for applications.[2] The entries at the ith level of a node's routing table denote the siblings of the node's level-i group in the hierarchy (see Fig. 1c). For instance, at level 0, the routing table of node D from Fig. 1, apart from the node's own group, G_D^0, contains entries for groups G_E^0 and G_R^0, which are the siblings of G_D^0 within the higher-level group G_E^1. Likewise, at level 1, D's routing table contains an entry for G_E^1, which is D's level-1 group, and for groups G_F^1, G_P^1, and G_Q^1, which are the siblings of G_E^1 in the higher-level group G_P^2. An entry for a level-i group contains the identifier of the next-hop neighbor on the shortest path to the head of this group, the number of hops to reach the head, and some other maintenance data such as counters specifying when the entry expires if not refreshed. The organization of routing tables allows the size of a node's routing table to also be polylogarithmic with respect to the number of nodes in the network [2]. This combined with the fact that a single routing entry is only a few bytes guarantees that the state maintained by a node is very small and scales gracefully with the network size.

2.2 Protocols for Hierarchy Maintenance

The problem of maintaining a recursive multihop area hierarchy has been studied for some time albeit mostly for networks with properties different from the aforementioned properties of large WSNs, such as wired networks and mobile ad hoc networks. Although PL-GOSSIP builds upon some theoretical results of those studies, because of the different properties of the target environment, it emphasizes different issues.

In his PhD dissertation on the early Internet architecture proposals [2], Hagouel proved that constructing an optimal area hierarchy that minimizes node state is NP complete. In other words, in practice, only heuristic solutions can be employed. Numerous hierarchy construction heuristics have been developed for partitioning data sets in data mining [19]. These algorithms, however, can hardly be applied to large networks of wireless autonomous devices because they require central control and would generate heavy traffic when propagating hierarchy data between nodes. Consequently, distributed hierarchy maintenance protocols have been introduced.

Depending on the assumptions on the internode connectivity, distributed hierarchy maintenance protocols for wireless networks are divided into two families: one-hop protocols and multihop protocols. One-hop protocols [20], [21], [22] have been designed for small and dense WSNs with a remote sink, in which the prime objective is minimizing the energy cost of data collection at the sink. The motivation behind such protocols is that a long-distance direct data transmission to the sink drains lots of energy. Hence, to reduce energy consumption, the nodes should avoid such transmissions. To this end, the nodes organize themselves into an area hierarchy, and each node transmits data only to its parent head node in the hierarchy (inexpensive, short-range transmission). The head node performs some compression of the received data and forwards them to its superhead, and so on such that in the end only the top-level head transmits the compressed data from all nodes to the sink. As a result, the total energy cost of data collection is significantly reduced. While such protocols for maintaining the hierarchy consider the aforementioned characteristics of WSNs, their design inherently assumes that the network is one-hop, that is, by dynamically increasing its transmission power, each node can directly communicate with any other node in the network. Although this assumption may hold in small, densely placed WSNs, for practical reasons, it does not hold in large WSNs operating in many real-world situations. One-hop protocols, however, cannot efficiently maintain an area hierarchy in multihop networks.

In contrast, the second family of hierarchy maintenance protocols is meant specifically for multihop wireless networks [4], [5], [23], [24], [25]. Those multihop protocols work by flooding beacon messages at all hierarchy levels, where the flooding radius of a node depends exponentially on the node's level as group head. More specifically, each group head periodically or after a change in the network broadcasts a beacon message that is received by the nodes within its radio range (i.e., its neighbors). The neighbors refresh their routing entries corresponding to the head's group and apply any label updates performed by the head if they belong to the head's group. Afterward, they rebroadcast the message so that their neighbors can update their state, and so on up to a certain radius depending on the level of the head in the hierarchy. Such a simple and elegant scheme allows wireless nodes to collaboratively maintain an area hierarchy. However, although the protocols assume that the nodes are wireless, they do not take all the peculiarities of wireless *sensor* networks into account, especially tight energy budgets and lossy connectivity.

It has been shown that flooding, in general, and multilevel flooding, in particular, is highly inefficient with respect to energy consumption [26], [27], [28]. First, every node forwards beacon messages for all group heads that have the node within their advertisement radius. Since a head may issue a beacon essentially at any moment, the node cannot arbitrarily sleep as it may miss a beacon message from some head. Moreover, once it receives a beacon, it has to immediately rebroadcast it in order to guarantee that an instance of the beacon that has traveled i hops is always received before an instance that has traveled j hops for all $j > i$. Otherwise, the routing tables could be invalid. Consequently, there is little room for a node to save energy. Even if an energy conserving MAC layer is employed, the fact that every node forwards myriads of short beacons results in large energy overhead on the exchanged useful protocol data. In

2. Since routing is not used by PL-GOSSIP but only by applications on top of PL-GOSSIP, we have decided to move the routing algorithm to Appendix G of the supplemental material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2009.89.

effect, the lack of support in the protocols for energy saving severely impairs their energy efficiency, as we demonstrate in Section 5.4.

The applicability of the proposed multihop hierarchy maintenance protocols to energy harvesting WSNs is also limited. As mentioned above, those protocols are reactive: a group head can issue a beacon message at any moment in reaction to a change in the network, for instance, after it has detected that some other head has died. However, the traffic pattern, and thereby, the energy consumption of a reactive protocol is not even over time but exhibits bursts corresponding to events occurring in the network. The duration and magnitude of the bursts can be large if the events affect many nodes (e.g., massive failures and network partitions) or if they cause other events leading to a domino effect. Accurately estimating energy consumption of a protocol that exhibits such bursts is virtually impossible before the actual deployment, especially considering the embedded nature of WSNs and their interactions with the surrounding environment. As a result, provisioning the energy harvesting infrastructure (e.g., selecting the size of solar cells) for the proposed hierarchy maintenance protocols may be very difficult [15]. When the energy harvesting infrastructure is underprovisioned, a burst in energy consumption may render parts of the network inoperable. Overprovisioning, in turn, increases the form factor and the cost of sensor devices, which may render them unusable for a given application.

Finally, for simplicity, the existing protocols for hierarchy maintenance assume a lack of message loss. In contrast, due to the low-power wireless communication they employ, WSNs exhibit considerable message loss, both anticipated, resulting from signal fading with distance, and varying, due to transmission collisions and noise. Hierarchical beaconing, however, is not resilient to message loss. A lost beacon message may disrupt the routing paths or may even lead to changes in the hierarchy, for example, when a lost beacon increases the length of a routing path beyond the limit acceptable for a group at a given level [29]. Although it is possible to alleviate the impact of message loss, for instance, by broadcasting beacons multiple times, such solutions typically further increase the already high energy consumption of hierarchical beaconing and can be difficult to implement correctly.

Due to these and other drawbacks of existing protocols, PL-GOSSIP may be an attractive alternative. As we explain in the next section, it enables self-management of an area hierarchy in a way that addresses all the peculiarities of WSNs.

3 Protocol Overview

Self-management of a recursive multihop area hierarchy corresponds to nodes autonomously bootstrapping the hierarchy and maintaining it during the whole network lifetime. Hierarchy bootstrap involves nodes autonomously synthesizing their labels and filling in their routing tables. Maintenance, in turn, encompasses detecting node failures and changes in the internode connectivity and repairing the hierarchy after such changes by updating routing tables and modifying labels. To fill in and update node routing tables, group heads advertise their groups among other nodes so that the nodes learn about any existing groups. Based on this knowledge, nodes synthesize and maintain their labels. Label synthesis is typically done in a bottom-up fashion as an alternative top-down method [30] cannot easily deal with variations in node densities [4]. More specifically, some nodes promote themselves to higher-level group heads, effectively spawning higher-level groups, and other nodes join such higher-level groups. Label repair after the death of a group head, in turn, is achieved by having another node promote itself to group head.

In PL-GOSSIP, all these activities are performed using a combination of local-only operations and asynchronous neighborhood gossiping. The nodes operate in rounds, each lasting T time units. In every round, in a single gossip message, each node broadcasts its protocol state, that is, its label and routing table. The message is received only by the node's neighbors (i.e., the nodes within the radio range of the broadcasting node). The neighbors subsequently merge the received state with their own local state. Likewise, they broadcast their state once per round. In this way, the hierarchy information can propagate throughout the network over multiple hops. In particular, nodes learn about hierarchy groups and group heads in their vicinity.

Based on this knowledge, the nodes construct and maintain the hierarchy. Hierarchy construction is performed in a bottom-up fashion. Nodes probabilistically promote themselves to higher-level group heads by locally modifying their labels, effectively spawning higher-level groups. When they broadcast gossip messages in subsequent rounds, their neighbors, the neighbors' neighbors, and so on learn about the newly created groups, and can join those groups also by modifying their labels locally. In this way, the nodes gradually bootstrap the group hierarchy. Hierarchy repair after detecting a failure of a node or a connectivity change, if necessary, is performed using the same mechanisms. Detecting a node failure is relatively easy as the failed node does not broadcast any new gossip message. The same applies for a change in the internode connectivity. Therefore, to sum up, local-only label operations in combination with asynchronous neighborhood gossiping are sufficient for nodes running PL-GOSSIP to collaboratively construct and maintain such a complex distributed structure as a multihop recursive area hierarchy.

More importantly, however, the combination of local-only operations for updating the hierarchy and asynchronous neighborhood gossiping for propagating the hierarchy information addresses all the peculiarities of WSNs, listed in Section 1. First, it offers excellent opportunities for aggressive node energy saving. In contrast to reactive protocols, which generate irregular traffic, the operation and the resulting traffic pattern of PL-GOSSIP are very regular and well defined: a node transmits only a single gossip message per round and expects to receive a similar message from every neighbor. As a result, nodes can synchronize their rounds, such that they are active only during a short period at the beginning of each round to analyze their local state and to exchange gossip messages, while being asleep for most of the round. This allows the nodes to operate on extremely tight energy budgets. For

example, for the current generation of node hardware, a round length of 60 seconds already enables months of operation with battery-based power supply. Moreover, such a regular operation pattern simplifies estimating energy consumption of a network prior to an actual deployment. This, in turn, in contrast to reactive protocols, facilitates provisioning the energy harvesting infrastructure.

Second, the self-management properties of PL-GOSSIP make the protocol robust against node failures and connectivity changes. Whenever a node fails or a new node is introduced to the network, the change in the node population is detected by other nodes, so that the nodes can collaboratively repair the hierarchy to account for the change. Likewise, when connectivity between nodes changes, for example, due to a communication obstacle emerging or disappearing, the changes are accounted for in the node routing tables and possibly also labels. There are only a few possible failures that affect many nodes, such as the failure of the top-level group head, and the great majority of node failures and connectivity changes affect very few nodes and, thus, require local-only repair activities. This makes PL-GOSSIP robust as we demonstrate in our experiments. Moreover, by varying the round length T, one can explore the trade-off between the latency of reacting to changes in the network and the energy consumption. Applications that require ultralow energy consumption would likely use large values of T, whereas applications that require fast recovery after failures would rather prefer smaller T.

Third, due to the periodic nature of gossiping, PL-GOSSIP is robust against message loss. Since, in each round, a node broadcasts its whole protocol state, there is some redundancy in the data transmitted in consecutive rounds. Therefore, even if some neighbor misses some of the node's messages, it will likely receive the data in one of the subsequent rounds. This feature enables configuring PL-GOSSIP to operate in networks with high message loss rates, as we demonstrate in our experiments, and facilitates porting the protocol from the simulation environment to the real world.

Finally, the simplicity of the concepts employed by PL-GOSSIP and its small resource requirements enable implementing the protocol for real sensor node hardware. In contrast, we are not aware of any implementations of existing hierarchy maintenance protocol. Since implementing protocols for severely constrained sensor devices is challenging and usually demonstrates a large divergence between practice and theory, the fact that PL-GOSSIP can seamlessly operate in the real world constitutes another important feature.

4 Protocol Details

We formalize the above sketch of PL-GOSSIP below. First, we give sample properties that we assume for the hierarchy in the remainder of the paper. Then, we describe how using local-only operations and asynchronous neighborhood gossiping PL-GOSSIP constructs and maintains the hierarchy with those properties. While in the paper we give a textual explanation of PL-GOSSIP, the code listings of the algorithm core can be found in Appendix F of the supplemental material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2009.89.

4.1 Sample Hierarchy Model

The nodes communicate wirelessly: each message is broadcast and the nodes able to hear the broadcast receive the message. We say that there exists a *link* between two nodes if they are able to receive each other's messages. Because wireless low-power communication employed by sensor nodes is often unreliable and asymmetric, PL-GOSSIP ensures that the links it chooses are symmetric and are of high quality (see Section 4.4.2). We say that two nodes are *neighbors* if and only if (abbreviated as *iff*) there exists a bidirectional high-quality link between them.

Nodes group themselves into sets based on their connectivity. The groups correspond to network areas and form a recursive multihop multilevel hierarchy. We say that two groups are *adjacent* iff they contain two nodes (one in each group) that are neighbors. For example, in Fig. 1a, group G_F^0 is adjacent to groups G_D^0, G_E^0, G_B^0, and G_R^0. Group G_E^1, in turn, is adjacent to G_P^1 and G_M^1, but not to G_Q^1.

For the remainder of this paper, we assume the following sample properties of the group hierarchy:

Property 1. *Level-0 groups correspond to individual nodes.*

Property 2. *There exists a single, level-$\mathcal{H}$ group that contains all nodes. We call this group the top-level group.*

Property 3. *Level-$i{+}1$ groups (where $0 \leq i < \mathcal{H}$) are composed out of level-i groups, such that each level-i group is nested in exactly one level-$i{+}1$ group. A level-$i{+}1$ group is the supergroup for its level-i groups, and likewise, these level-i groups are the subgroups of their level-$i{+}1$ group.*

Property 4. *Each level $i+1$ group (where $0 \leq i < \mathcal{H}$) contains a central subgroup that is adjacent to all other subgroups of this group.*

We define the *head* node of a level-i group recursively: 1) if $i = 0$, then the head of the group is the sole node constituting the group; 2) if $i > 0$, then the head of the group is equal to the head of the central subgroup. A node is thus a *level-i head* iff it is the head of groups at levels from 0 to i, but not the head of a level-$i{+}1$ group. For example, in Fig. 1a, group G_P^1 is the central subgroup of group G_P^2. Similarly, group G_Q^0 is the central subgroup of group G_Q^1.

To show that our group hierarchy model has the potential to provide polylogarithmic labels and routing tables, based on Properties 1-4, we have derived tight bounds on the internode distances, as formalized by the lemmas below. The proofs of these lemmas (by induction) are given in Appendices A-C of the supplemental material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2009.89. The exponentially growing bound on group diameter at subsequent hierarchy levels is a basic requirement for polylogarithmic labels and routing tables, as discussed in Section 2.1.

Lemma 1. *A node from a level-i group can reach a node in any adjacent level-i group in at most 3^i hops.*

Lemma 2. *The distance between the head nodes of two adjacent level-i groups is at most 3^i hops.*

Lemma 3. *The distance between any two members of a level-i group is at most $3^i - 1$ hops.*

Note that the above properties have been chosen as an example to illustrate the basic idea behind gossip-based self-management of a recursive multihop area hierarchy. For instance, there is no property that would guarantee that a group contains more than one subgroup, which as we show is not strictly necessary from the perspective of point-to-point routing that we take in this paper. Using the ideas behind PL-GOSSIP, however, one can maintain a different set of hierarchy properties, as we demonstrate in Section 5.4.

4.2 Route Information Maintenance

Self-management of the hierarchy essentially boils down to ensuring that the above properties hold. This requires maintaining the labels and routing tables appropriately. Since the label maintenance is performed depending on the state of node routing tables, we focus on the routing table maintenance first.

As described in Section 2.1, the routing table of a node contains one entry for each sibling group at every level of the hierarchy. A routing entry for a level-i group consists of the group's level i, the identifier of the group's head, a sequence number generated by the head, a bit indicating whether the group is adjacent to the node's level-i group, the identifier of the next-hop neighbor on the shortest path to the group's head, and the number of hops on this path.

Nodes maintain their routing entries with a straightforward hierarchical distance-vector algorithm. A routing entry for a group originates at the head of the group, which generates a new sequence number for the entry, zeroes the hop count of the entry, and sets the adjacency bit of the entry. A new sequence number for the entry is generated at the end of each gossiping round. Such a refreshed entry, together with all other routing entries maintained by the group head, is embedded in the next gossip message of the head. In this way, when the head broadcasts the message, its neighbors can refresh (or create) their routing entries corresponding to the head's group. When they broadcast their gossip messages, their neighbors can also refresh their entries for the group, and so on.

Merging a node's local routing table with a neighbor's routing table received in a gossip message is performed like in a standard distance-vector algorithm, but with the exception that the node considers only a subset of the received routing entries. More specifically, node A upon receiving a gossip message from node B, by comparing its label with B's label from the message, determines the minimal level of a group it shares with B. If the minimal level is i, A can update its routing table with those entries from B's routing table that are in rows no lower than $i - 1$. In contrast, if there is no common group for A and B (Property 2 is violated), A opportunistically updates its routing table by adding entries for those groups of which B is member and that are at level $l_A - 1$ and above (where l_A is the length of A's label). This latter case allows A to propagate information about the hierarchy property violation among the members of its group, which is necessary when constructing the hierarchy and recovering from failures. Like in a distance-vector algorithm, when updating its routing entries, an objective of a node is to choose the freshest entries (i.e., those with the freshest sequence numbers) that minimize the number of hops and ensure that the group adjacency information is propagated correctly. In this way, the algorithm guarantees freshness of the route information and short routes to the group heads.

If a node has not refreshed a routing entry for a certain number of rounds, derived from Lemma 3, it concludes that the group represented by the entry can no longer be reached, for instance, because the group head has died or all the links to the group have been broken. Similarly, a feedback from the application may be used to detect an unreachable routing entry. Such an entry should be removed from the node's routing table. To prevent routing cycles when node failures and connectivity changes occur, PL-GOSSIP uses route poisoning: before removing an entry a node marks it as unreachable. Such an entry is broadcast in the node's gossip messages for several rounds, which allows other nodes to detect the failure as well. To sum up, entries referring to nonexisting or unreachable groups are always evicted.

4.3 Basic Label Operations and Update Vectors: Ensuring Property 3

Based on their routing tables, the nodes maintain their labels, such that the hierarchy reflected in the labels satisfies Properties 1-4. Property 1 always holds. Properties 2-4, in turn, must be enforced using local-only operations and gossip-based information propagation. In addition, the only means of internode coordination is the round-based communication pattern. In the remainder of this section, we discuss how using these simple concepts PL-GOSSIP maintains the formal properties of the hierarchy.

We start with Property 3, which expresses the recursiveness of the hierarchy. It states that two members of the same level-i group also belong to the same level-j group, for all $j > i$. Maintaining Property 3 thus requires that, for any level-i group, any modifications to node labels at levels above i must be performed in a *consistent* way by all members of the group.

To this end, for label updates, we adopted the single-master model on a per-group basis. The dynamically designated head node of a group makes all the label updates regarding the membership of this group in the hierarchy, as formalized by the rule below.

Responsibility rule. *The $i+1$-st element of a node's label is updated only by the head of the level-i group the node is member of (denoted by the ith element of the node's label).*

Intuitively, the rule states that the head of a group is responsible for moving the whole subbranch of that group between branches corresponding to different supergroups in the label tree (cf., Fig. 1b). Other group members simply adopt the label updates by such a node.

4.3.1 Update Vectors

In asynchronous neighborhood gossiping, however, it is not trivial to ensure that all group members adopt label updates in a consistent way. For example, without additional

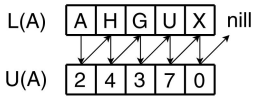

Fig. 2. A sample label and update vector.

information, a node with label $A.H.\mathrm{G}.U.X$, receiving from its neighbors gossip messages with labels $B.H.\mathrm{G}.V.Y.Z$ and $C.I.\mathrm{G}.W$, cannot determine whether its label should become $A.H.\mathrm{G}.V.Y.Z$ or $A.H.\mathrm{G}.W$, or whether it should stay unmodified.

As a solution to this problem, we introduce *update vectors*. A node's update vector corresponds to the node's label and unambiguously specifies the updates applied to the label. The ith element of the vector denotes the sequence number of the last known label update made at level $i+1$ by the node's level-i group head. For instance, in Fig. 2, node A knows that:

1. the last label update performed by A at level 1 has number 2 and wrote H at position 1 of its label;
2. the last update performed by H at level 2 has number 4 and wrote G at position 2 of its label;
3. the last update performed by G at level 3 has number 3 and wrote U at position 3 of its label;
4. the last update performed by U at level 4 has number 7 and wrote X at position 4 of its label; and
5. X acting as the top-level head has not yet made any updates at level 5 ($U(A)[4] = 0$).

A node's update vector is broadcast with the node's label in the node's gossip messages and is essential to propagating label updates, as we explain shortly.

4.3.2 Label Operations and Update Propagation

In PL-GOSSIP, nodes modify their labels with only two basic operations: label extension and label cut. Label extension (see Fig. 3a) is executed locally by a top-level group head X, when constructing or recovering the hierarchy. By extending its label, X joins its group G^i_X, to a higher-level group G^{i+1}_Y (if it extends its label with Y), or spawns a new higher-level group G^{i+1}_X (if it extends its label with X). Label cut (see Fig. 3b), in turn, is executed locally by a nontop-level head X, when X has detected that its group G^i_X, can no longer be a subgroup of a higher-level group G^{i+1}_Z. This operation removes group G^i_X

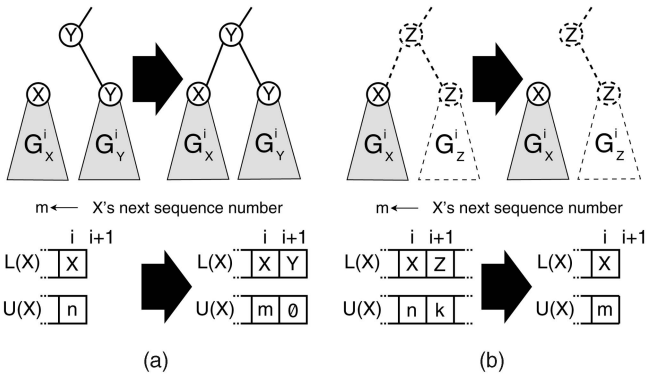

Fig. 3. Label operations. (a) Label extension. (b) Label cut.

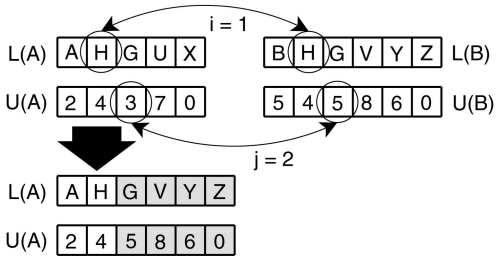

Fig. 4. An example of update propagation. Node A determined that B has a fresher update performed by S at level j, and thus, A adopts B's updates. Note that initially the lengths of the labels can differ.

from supergroup G^{i+1}_Z. Label cut can also be used to dissolve a group in order to balance group sizes or rotate group heads. However, in our experiments and subsequent research activities [29], we have noticed that from the point-to-point routing perspective we take in this paper, such functionality is not necessary. Therefore, to avoid significantly complicating the algorithm, we omit the description of how to use label cut to balance group sizes.

It is crucial to note that both label extension and label cut abide by the responsibility rule, that is, they are used by a level-i head to modify the head's label at level $i+1$. Moreover, in both operations, when modifying its label at level $i+1$, the head X also writes a new sequence number at the ith position of its update vector (in Fig. 3: $m \leftarrow X$'s next sequence number; $U(X)[i] \leftarrow m$). This is to indicate that label update performed by X is the freshest one, so that other members of G^i_X can also adopt the update using the following algorithm.

Whenever a node A receives a gossip message from a neighbor B, it checks if it shares a group with B. More specifically, A looks for the minimal i such that $L(A)[i] = L(B)[i]$ (see Fig. 4). If such i does not exist, then A has just discovered a violation of Property 2 of the group hierarchy, which will be propagated through routing tables and handled by the hierarchy construction algorithm of A's top-level head node, as we explain further in the paper. Otherwise, A determines which of the two labels is fresher by comparing its update vector $U(A)$ with B's update vector $U(B)$ starting from position i. If for some $j \geq i, U(A)[j] \neq U(B)[j]$ (see Fig. 4), then one of the labels is fresher than the other [they can differ starting from the $(j+1)$st element]. If B's label is fresher ($U(A)[j] < U(B)[j]$), then A copies B's label and update vector starting from position j: $L(A)[j\ldots] \leftarrow L(B)[j\ldots]$ and $U(A)[j\ldots] \leftarrow U(B)[j\ldots]$. In this way, A's information on the hierarchy membership becomes consistent with the fresher information from B, and moreover, when A broadcasts the next gossip message, its neighbors can also adopt the fresh information, and so forth. As a result, any label update (extension or cut) made by a group head is eventually adopted by all members of the group, as formalized by the lemma below. This is even true for a node that has rejoined the network after a long disconnection period.

Lemma 4. *Update propagation based on the responsibility rule and update vectors guarantees eventual consistency of node labels.*

The proof is given in Appendix D of the supplemental material, which can be found on the Computer Society

Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2009.89. Essentially, the responsibility rule designates a single group head to update each element of a node's label. Based on this rule, the update vectors, in turn, guarantee that because the nodes gossip continuously, any members of the same level-i group will eventually have equal labels at all levels above i. This ensures that Property 3 eventually holds. We can even give stronger consistency guarantees by using Lemma 3 to bound the maximal number of rounds to propagate an update. It is also possible to further optimize the update adoption algorithm, but we do not present the optimization here for simplicity.

4.4 Hierarchy Construction and Recovery: Ensuring Properties 2 and 4

Ensuring Properties 2 and 4 is directly related to the way nodes construct the hierarchy and recover it after failures and changes in the internode connectivity. In fact, hierarchy construction and recovery is performed by detecting violations of Properties 2 and 4 in each round and by reacting to such violations. Nodes learn about the violations from their routing tables. They react to the detected violations autonomously by extending or cutting their labels. When performing such local-only label updates, the nodes abide by the responsibility rule, which, combined with the above update propagation algorithm, guarantees that all hierarchy properties eventually hold.

4.4.1 Hierarchy Construction

Initially, each node is a top-level head (of its level-0 group—Property 1), that is, its label length is equal to 1. Hierarchy construction is performed by top-level heads detecting that they are not the sole top-level heads (Property 2 is violated) and reacting to such violations by extending their labels, which corresponds to spawning a new higher-level group or joining a group to an existing higher-level group (see Fig. 3a). Label extensions gradually eliminate all violations of Property 2, leading to the convergence of the hierarchy. Eventually, only a single top-level group exists.

The head X of a top-level group G^i_X discovers a violation of Property 2 iff its routing table contains entries for an adjacent G^k_Y, where $k \geq i$. There are two possible scenarios: 1) if $k = i+1$, X can try to make G^i_X a subgroup of G^{i+1}_Y or 2) X can spawn a new supergroup G^{i+1}_X hoping that other adjacent level-i groups will join this group, or that it will be possible to make G^{i+1}_X a subgroup of some level-$i+2$ group. Making G^i_X a subgroup of G^{i+1}_Y corresponds to X extending its label with Y at level-$i+1$ (see Fig. 3a). Other members of G^i_X will gradually learn about the membership update and extend their labels, as guaranteed by our update propagation algorithm. This algorithm also guarantees that if G^{i+1}_Y is itself a member of some G^{i+2}_Z, all members of G^{i+1}_Y (in particular, the members of G^i_X) will also gradually extend their labels at level $i+2$ with Z, and so forth. Likewise, spawning a new supergroup G^{i+1}_X corresponds to X extending its label with X.

Making G^i_X a subgroup of an existing G^{i+1}_Y is always preferred, as it reduces the number of groups at level $i+1$ compared to level i. Yet, due to Property 4, it is only possible if G^i_X is adjacent to the central subgroup of G^{i+1}_Y, that is, G^i_Y. Formally, X can extend its label at level $i+1$ with Y iff its routing table contains entries for adjacent G^i_Y and G^{i+1}_Y.

Otherwise, X cannot immediately make G^i_X a subgroup of any level-$i+1$ group, and thus, it must potentially spawn a new level-$i+1$ group G^{i+1}_X. Convergence requires preventing all groups from becoming supergroups in the same round. In particular, in the beginning, each node forms a single level-0 group, so allowing all nodes to create singleton level-1 groups would not guarantee convergence. To this end, X probabilistically defers spawning a supergroup for a number of rounds. Although different probabilistic heuristics are possible, for the purpose of this paper, we adopted the following simple one. Upon discovering that it must potentially spawn a supergroup, X first clusters rounds into S virtual slots, each lasting r rounds, then randomly selects a slot $s \in \{0 \ldots S-1\}$, and subsequently, defers spawning a supergroup for $r \cdot s + 1$ rounds, hoping that in that time some adjacent group spawns a supergroup, so that it will be possible to make G^i_X a subgroup of this supergroup.

$S = 2$ already ensures that the number of groups on consecutive levels drops exponentially fast, provided that the slot size r is long enough. Such a decrement is a direct consequence of the following lemma, with a proof in Appendix E of the supplemental material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2009.89:

Lemma 5. *Assume that the slot size is longer than the number of rounds it takes to propagate information between the heads X and Y of two adjacent "top-level" groups G^i_X and G^i_Y. In this case, with probability $\geq \frac{1}{4}$, G^i_X will be able to join G^{i+1}_Y or vice versa.*

Oversimplifying things, assuming $S = 2$ and r meeting the above assumption, we could expect that half of the groups (the ones that chose slot 1) will be able to join the supergroups formed by the other half (the ones that chose slot 0), that is, the number of groups drops exponentially with the level. This not only guarantees convergence, but also results in a polylogarithmic height of the membership tree.

We can choose the slot size r guaranteeing the above requirements based on the entries in the routing table. More formally, assuming no message loss, a level-i head deferring supergroup creation chooses r equal to the number of hops to the furthest adjacent level-i head. Note that although this value is bounded by Lemma 2, it is smaller, on average.

4.4.2 Handling Failures and Message Loss

A *benign failure* of a node or a link does not require any repair apart from removing some routing table entries. In contrast, a *disruptive failure*, like a group head crash, violates Property 4, and thus, requires repairing the group hierarchy. The repair boils down to having a head node that detected the failure cut its label down to the level the failure occurred, which corresponds to removing a subgroup from a no longer existing group (see Fig. 3b). Later, if necessary, the above hierarchy construction algorithm will join such a removed subgroup to a different group, restoring all the hierarchy properties.

The head X of a group G_X^i, which is a subgroup of G_Z^{i+1}, discovers a violation of Property 4 iff its routing table does not contain an entry for the central subgroup G_Z^i, of group G_Z^{i+1}, or such an entry exists, but its adjacency flag is not set. This implies that G_X^i should no longer be a subgroup of G_Z^{i+1}. To this end, X cuts its label down to position i, which corresponds to removing G_X^i from G_Z^{i+1} (see Fig. 3b). Such an operation of restoring Property 4 may, in turn, generate a violation of Property 2. However, this violation will be subsequently handled by the hierarchy construction algorithm. Our update propagation mechanism guarantees that all members of G_X^i will adopt the decision of X to leave G_Z^{i+1} and later possibly join some other level-$i+1$ group, which guarantees restoring all the hierarchy properties.

Message loss may also be viewed as failure. PL-GOSSIP tries to tolerate a certain rate of message loss in three ways. First, following a standard practice [31] and making use of its well-defined, periodic traffic pattern, PL-GOSSIP measures the bidirectional link quality that reflects the *expected message loss*. Two nodes are allowed to be neighbors iff the bidirectional quality of their link is above a certain threshold θ (e.g., $\theta = 80\%$). This way PL-GOSSIP ensures the global expected message loss for the neighbor links to be at most $1-\theta$. Second, by introducing the local age field of each entry in a node's routing table, PL-GOSSIP allows several consecutive gossip messages that refresh this entry to be lost, which deals with *transient variations in message loss*. Finally, while constructing the hierarchy, PL-GOSSIP adds a custom-tolerable deviation of $k \cdot (1-\theta)$ to the the slot size (e.g., for $k = 2, r^* \leftarrow \lceil r \cdot (1 + 2 \cdot (1-\theta)) \rceil$), which forces a head to defer spawning a supergroup a bit longer to compensate for the expected message loss that otherwise might prevent timely delivery of information from another head. Heavy repeated message loss above the tolerated values is simply treated as a link failure, handled in a standard way by PL-GOSSIP.

5 Evaluation

We evaluated PL-GOSSIP on three platforms: our own event-driven high-level simulator, TOSSIM—a bit-level node simulator for the TinyOS sensor node operating system, and a subset of our testbed consisting of 55 TelosB nodes [32].

Our simulator is based on other high-level simulators for WSNs (i.e., [27], [33]), and thus, it makes several standard assumptions which allow for simulating very large networks and for repeating experiments multiple times. First, it models nodes as having a fixed circular radio range: a node has links to all and only those nodes that fall within its range. Second, it ignores the capacity of and congestion in the network. Finally, it pessimistically fixes message loss to $1-\theta$ for all links (i.e., the message loss matches the bidirectional link quality threshold). Later experiments with the actual embedded implementation confirmed that these assumptions do not impair real-world operation of PL-GOSSIP because 1) PL-GOSSIP creates the logical network structure based solely on physical links and the measured link quality, and thus, it makes no implicit simplifying assumptions regarding connectivity or message loss and 2) the state exchanged between nodes is small whereas the round length is large, and therefore, the MAC layer can efficiently schedule packet transmissions without causing congestion.

TOSSIM and our testbed, in turn, both ran an actual, embedded TinyOS 2.0 implementation of PL-GOSSIP, which, to the best of our knowledge, is also the first actual implementation of an area hierarchy maintenance algorithm ever reported for WSNs in the literature. TOSSIM incorporates a realistic low-level wireless signal propagation and noise model, derived from a number of real-world experiments. Therefore, it allowed us to accurately validate the protocol communication aspects in realistic settings. The testbed evaluation, in turn, was aimed at validating the claims about the systems aspects of PL-GOSSIP. It was performed on TelosB nodes, which are good representatives of resource-constrained sensors: a TelosB node has an 8-Mhz 16-bit MCU, 10 KBs of RAM, 48 KBs of flash memory for the code, and a 250-kbit/s radio with a 50-m indoor range. The overall goal of the implementation-based experiments was thus validating simulation results and proving that PL-GOSSIP can seamlessly operate in the real world on real hardware. Nevertheless, since this paper focuses on algorithmic aspects of PL-GOSSIP, the majority of the presented experimental results were obtained via simulations.

Finally, in addition to PL-GOSSIP, we have implemented a sample state-of-the-art protocol based on hierarchical beaconing [4], [5]. We use the performance results obtained for the implementation of the protocol to illustrate some of the benefits offered by PL-GOSSIP.

5.1 Basic Protocol Properties

We simulated PL-GOSSIP with various network sizes (growing exponentially from 1 to 4,096 nodes), densities (from sparse $\sim$12 neighbors per node to very dense $\sim$80 neighbors per node), and topologies (grid, uniform, random). Because the results were consistent in all cases, for the sake of brevity, in this paper, we present only a subset of the experiments. In these experiments, we arranged nodes into a square grid with unit spacing between nodes. The radio range of a node was 2 units, resulting in sparse neighborhoods of at least 5 (corner nodes) and at most 12 (most of the nodes) neighbors per node. Since we wanted to get insight into general properties of PL-GOSSIP, for the experiments presented in this section, we assumed no failure or message loss.

All nodes were booted simultaneously in round 0 and the experiment was stopped when the hierarchy had converged, that is, all the nodes had equal-length labels with the same last element. Simultaneous boot is a pessimistic scenario for a hierarchy maintenance protocol, as there are no higher-level groups formed, and consequently, all nodes must potentially spawn such groups. In contrast, normally the deployments are incremental, that is, nodes are added to the network one after another. Highly unrealistic simultaneous boot, however, allows us to study the worst-case performance of PL-GOSSIP.

When deferring spawning a supergroup, the number of slots S used by a "top-level" head (see Section 4.4.1) varied based on the level: at level 0, $S = 10$; at higher levels, $S = 2$. The rationale behind such a choice is minimizing the hierarchy height for dense networks. Oversimplifying things, by having 10 slots instead of 2 at level 0, we reduce

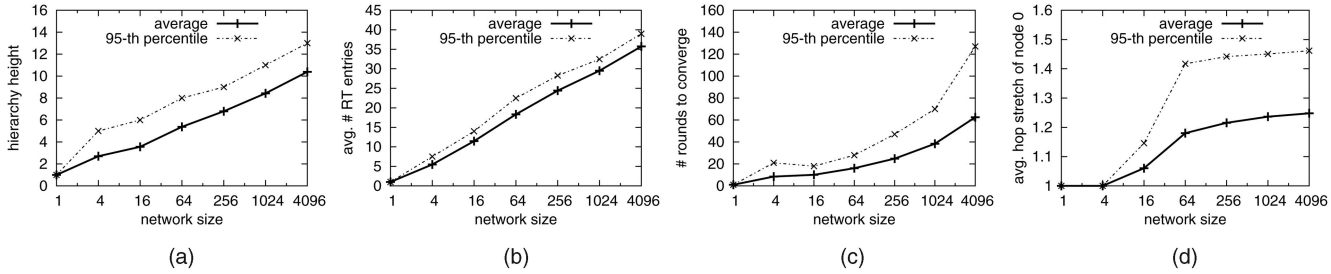

Fig. 5. Basic properties of PL-GOSSIP with respect to the network size. All values were obtained over 100 runs. (a) Hierarchy height. (b) Average routing table size. (c) Rounds to converge. (d) Average hop stretch.

the number of level-1 heads with respect to the number of level-0 heads (all nodes) roughly 10 times instead of 2 times. This generates a more shallow hierarchy. Moreover, the convergence time does not grow drastically, as from Lemma 2, the slot length at level 0 is equal to only one round, that is, after at most 10 rounds each node is guaranteed to be a member of some level-1 group.

The state maintained by a node is analyzed based on the height of the hierarchy (i.e., the label length) and the average size of a node's routing table. The hierarchy bootstrapping time is measured in the number of rounds necessary to construct the complete hierarchy. Finally, the quality of routes for the applications on top is measured using a standard metric, the average hop stretch: the average ratio of the number of hops on the route between two nodes to the number of hops in the shortest path in the neighborhood graph.

We conducted experiments for exponentially growing network sizes, with 100 runs for each size. Fig. 5 presents the results. It can be seen that both the hierarchy height (see Fig. 5a) and the average routing table size (see Fig. 5b) grow polylogarithmically with the network size. In particular, for a 1,024-node network, in 95 percent of the cases, these values are below 11 and 33, respectively. This is a direct consequence of our hierarchy properties (and their corollaries, Lemmas 1-3) and, especially, the construction algorithm with probabilistic group head election, supported by Lemma 5. Short labels and small routing tables require little memory, which is crucial as sensor devices typically have only a few kilobytes of RAM. More importantly, however, small local state also minimizes the bandwidth required by the protocol. These features, combined with the simplicity of the code (the listing of the core has only 130 lines including comments; cf., Appendix F in the supplemental material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2009.89), facilitate implementation on hardware-constrained sensor nodes.

The convergence time depends on the diameter of the network, and thus, it grows exponentially with the exponentially growing network size (see Fig. 5c). However, the absolute values indicate that the convergence is relatively fast. For instance, for a 1,024-node network (diameter 32), the hierarchy is formed within 38.4 rounds, on average, and at most 70 rounds in 95 percent of the cases. Assuming a long gossiping period, $T = 5$ minutes, with which a current-generation sensor node can likely last for years on a pair of batteries, we need 3.2 hours, on average, and at most 5.8 hours in 95 percent of the cases. We believe that this is insignificant compared to the expected theoretical network lifetime achievable with such a long gossiping period.

Again, it is crucial to note that simultaneous boot is the worst-case scenario. Normally, the network is built incrementally by adding one node after another. In such scenarios, the network converges typically in two to three rounds after the last node has been added. This is because adding a node to a network typically does not require spawning any new groups as the node can typically join some already existing level-1 group. The probability that this is not the case and that a new level-i group has to be spawned drops exponentially with i. Consequently, the amortized cost of adding a new node to the network is constant, which is also important for scalability.

Finally, from the application perspective, the average hop stretch is relatively stable for increasing network sizes (see Fig. 5d). Although the results reported for other route maintenance algorithms are not directly comparable, they indicate that the route overhead of PL-GOSSIP is small. For instance, alternative routing techniques, such as graph embedding [34], geographic routing [33], and compact routing [35] report similar hop-stretch values. We thus conjecture that PL-GOSSIP can provide acceptable routes for the applications.

As mentioned earlier, the experiments performed with different node densities and network topologies produced consistent results, and thus, we do not present the plots here. Denser networks result in shorter labels, but larger routing tables. The hierarchy convergence time in such networks is also shorter due to smaller network diameters which enable faster information propagation. The growth of the routing tables, in turn, decreases the hop stretch, thereby improving route quality for the applications. A shift to more irregular topologies, such as uniform or random, has minimal impact on the average values of the considered metrics. However, it slightly increases the variance and the 95th percentile of the metric values. In general, the behavior of PL-GOSSIP is very predictable for different configurations.

5.2 Robustness to Message Loss and Node Failures

Message loss, an inherent feature of WSNs, can have two effects on PL-GOSSIP: 1) it may prevent a head node deferring supergroup creation from learning in time about a newly created supergroup it could join, which can increase the hierarchy height and routing table sizes and 2) it may cause a

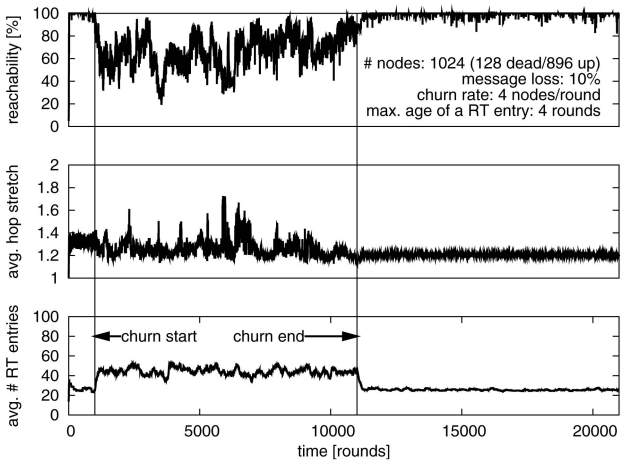

Fig. 6. An example of the system behavior with message loss and network dynamics.

node to falsely determine that a link is dead, which may possibly lead to unnecessary changes in the group hierarchy.

To study the first effect of message loss, we repeated the experiments from Section 5.1 with message loss rates of 1 percent, 5 percent, and high 10 and 20 percent. In these experiments, we isolated the first message loss effect from the second one by blocking the eviction of unrefreshed routing table entries. The results (not plotted) practically do not differ from Fig. 5, which confirms the efficacy of the countermeasures PL-GOSSIP employs against message loss (Section 4.4.2).

Since the second effect of message loss is correlated with failure detection, we combined the experiments on this effect with the experiments on network dynamics. In both cases, a (seeming) failure of a node or a link may turn out disruptive and trigger hierarchy changes that can temporarily lengthen or break routes between nodes, enlarge routing tables, or change the hierarchy height. In the experiments, we used various models of network dynamics (e.g., uniform failures, correlated failures, massive failures, and network partitions). PL-GOSSIP proved to be robust in all the experiments. Thus for brevity, here we present only the experiments with uniform failures.

In each such experiment, a 1,024-node network operated for 21,000 rounds. In any round, some 128 nodes out of 1,024 (12.5 percent) were dead. Moreover, 32 randomly selected nodes were always alive and were used for measuring the routing quality by letting them send messages to each other. In the initial 1,000 rounds, there were no changes in the node population. During the next 10,000 rounds, we generated node churn of a given rate. For instance, with a churn rate of 2, in every round, one random live node was killed and one random dead node was rebooted. Finally, during the last 10,000 rounds, there was again no churn. We ran the experiments for different rates of churn and message loss. We also varied the maximal age of a routing table entry, which determines how fast the unrefreshed entries are cleaned. The huge number of configurations and the long duration of a single experiment allowed us to conduct only one 21,000-round run per configuration.

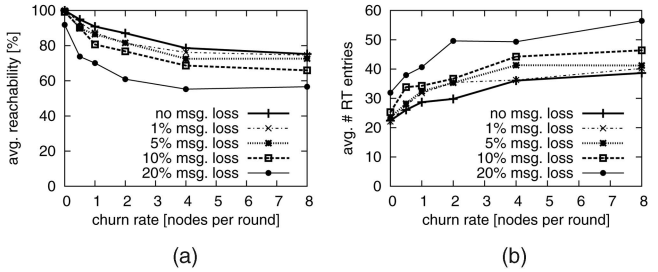

Fig. 7. The hierarchy and route behavior in the presence of churn and message loss. (a) Reachability. (b) Average routing table size.

Fig. 6 presents the results of a sample run, for presentation purposes, with high churn and message loss. Due to repeated message loss triggering unnecessary hierarchy changes, the reachability (top plot), that is, the existence of routes between nodes, occasionally falls during the initial 1,000 and the last 10,000 rounds. This is because with such high message loss it is very likely that some node falsely determines that a link failed. If such a "failure" triggers a membership change for a group, the communication to and from the group is temporarily disrupted (the communication within the group is preserved). This reduces reachability of a number of nodes, depending on the level of the group in the hierarchy. Node churn, which introduces real failures in the system, amplifies this effect causing greater oscillations in reachability.

Similarly, network dynamics generate peaks in the hop stretch (center plot). This is because propagating a new short route via a just-booted node requires some time.

Node churn also leads to larger routing tables (bottom plot). It takes a few rounds, depending on the maximal age of a routing table entry, to determine that a node is dead or a group ceased to exist. Therefore, the routing tables are polluted with entries for no longer existing groups. In addition, new nodes are constantly added to the system, further increasing the node routing tables. Nevertheless, even under high churn, the average routing table size is relatively small and stable, and it decreases fast when the churn stops.

Finally, message loss and network dynamics may result in the increments or decrements of the hierarchy height (not plotted). Such events, however, are very rare.

Fig. 7 shows the reachability deterioration and the routing table growth for different rates of churn and message loss. These results illustrate trends rather than absolute values because the churn and message loss rates we chose for the experiments were very high. For the churn rate of 8, the mean interfailure interval of a node is $(1{,}024 - 128) \cdot (\frac{2}{8}) = 224$ rounds, which even for a long duty cycle, $T = 5$ minutes, translates to 18.67 hours. In practice, once successfully deployed, a sensor node usually works for weeks or months. Consequently, even a churn rate of 1 results in a high mean interfailure interval of 6 days 5 hours and 20 minutes.

We finish the discussion on fault tolerance by augmenting the above experimental results with two important observations. First, a single node failure is rarely disruptive. Unless a failed node is a group head or the sole node connecting two groups, no changes in the hierarchy are necessary. Since the number of such nodes decreases

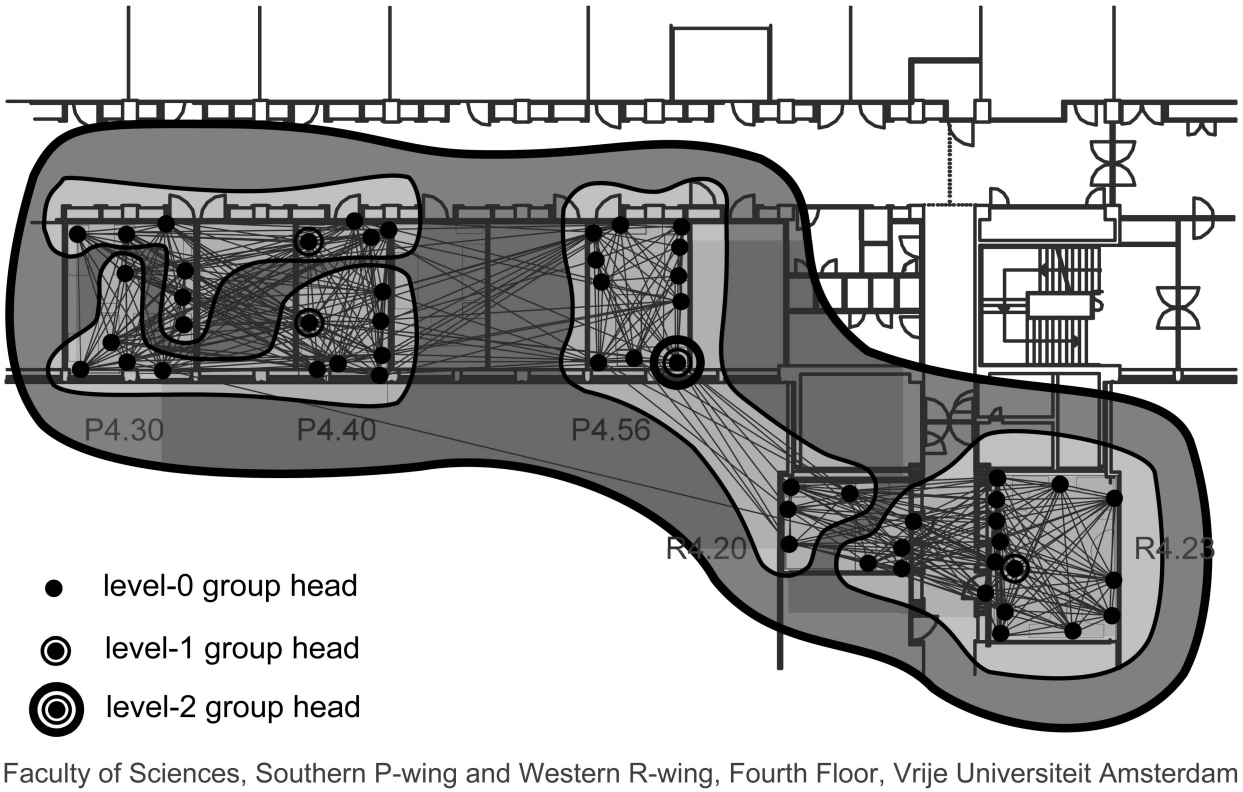

Fig. 8. An example hierarchy built by PL-GOSSIP on our 55-node testbed.

exponentially with the level, the probability that a failure is disruptive also decreases exponentially with the level. As a result, on average, the work involved in repairing a node failure is small. Second, the reachability depends on node proximity. If a disruptive failure occurs within a group, the reachability between the members of this group deteriorates. However, if a failure occurs outside the group, all members of the group are able to reach each other anyway. Such behavior is crucial in many applications, for instance, systems for disaster containment.

5.3 Implementation-Based Experiments

Our TinyOS 2.0 implementation of PL-GOSSIP was tested using a simple application that periodically broadcasts gossip messages and reported node statistics to a PC over a serial port. In addition, the application incorporated standard link estimation functionality [36]. As mentioned earlier, this functionality was used by PL-GOSSIP to discriminate neighbors with high-quality links from poorly connected nodes.

We conducted experiments with the implementation in TOSSIM and on our indoor testbed consisting of 55 TelosB sensor nodes [32]. The testbed was used to confirm that the implementation can run on real hardware. We tested how PL-GOSSIP maintains the group hierarchy and routes on a 4-hop network topology, scattered across several rooms and changing when nodes were turned off or rebooted. A sample hierarchy built and maintained by PL-GOSSIP in one of such testbed experiments is depicted in Fig. 8. The hierarchy simply meets all the properties defined in Section 4.1. Moreover, whenever the internode connectivity or the node population changes such that those properties become invalid, PL-GOSSIP detects and accounts for such changes, in exactly the same way as in the above high-level simulations. In other words, the testbed experiments evidence that PL-GOSSIP seamlessly operates on the real hardware.

TOSSIM, in turn, was employed to validate the claims about large-scale operation with realistic communication, and thus, we used a similar grid-based 1,024-node configurations as for the experiments with our simulator. With the standard tools and empirical real-world data provided by TOSSIM, we set up two realistic environments, *outdoor* and *indoor*, differing in signal propagation characteristics. Both environments exhibited irregularities in the neighborhood graph. In the outdoor environment, the irregularities were smaller, and thus, the link quality was mainly a function of distance. In contrast, in the indoor one, due to phenomena such as multipath reflections, nearby nodes were not necessarily connected and there were many asymmetric links. In fact, this environment exhibited much worse

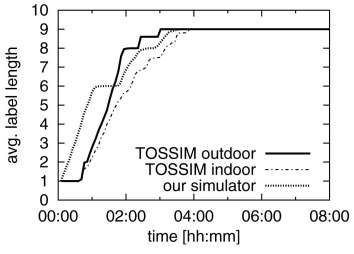

Fig. 9. Hierarchy bootstrap in TOSSIM and our simulator.

Metric	Indoor TOSSIM	Outdoor TOSSIM
# nodes	1024	1024
network diameter	17	12
# neighbors (avg./max.)	12.39 / 42	34.91 / 80
max. hierarchy height	9	9
avg. routing table size	66.07	89.03
avg. hop stretch	1.29	1.17
rounds to converge	47	36

Fig. 10. Common statistics for the implementation-based experiments of PL-GOSSIP.

communication quality than our testbed. A sample of the experimental results from TOSSIM are presented in Figs. 9 and 10. Again, the results match with the high-level simulations, which confirms that PL-GOSSIP can operate in realistic large deployment scenarios.

5.4 Comparison with Existing Protocols

Encouraged by the above results, we also implemented an existing state-of-the-art hierarchy maintenance protocol [4], [5]. The protocol is based on hierarchical beaconing (cf., Section 2.2). Periodically, every T time units, or after a change in the hierarchy, each group head broadcasts a beacon message. The beacon message is flooded over multiple hops depending on the level of the issuing node as a group head, as explained in Section 2.2. Typically, a level-i group head issues a beacon that is forwarded over 2^i hops which offers a polylogarithmic hierarchy height. In this way, in a single period, each node receives and rebroadcasts multiple beacon messages from the group heads that have the node in their advertisement radii. Using the received beacons, a node fills in and refreshes its routing table. Based on the contents of their routing tables, some nodes probabilistically promote themselves to higher-level group heads using heuristics similar to the ones of PL-GOSSIP (cf., Section 4.4). Since a cost of forwarding higher-level beacon in every period is prohibitive, the protocol amortizes the cost over longer periods by increasing the interbeacon interval proportionally to the beacon propagation radius: a level-i beacon, which is forwarded over 2^i hops, is issued every 2^i protocol periods. Even with this modification, however, in every period, a node forwards multiple beacon messages. Since, to the best of our knowledge, this was again the first implementation of the protocol on real sensor nodes, it took us considerable time to make it work in the real world.

To systematically compare the performance of PL-GOSSIP with the performance of the above protocol, we have adapted the implementation of PL-GOSSIP such that it built the same hierarchy as the implementation of the other protocol. To this end, we have replaced in PL-GOSSIP the hierarchy properties assumed in Section 4.1 with the hierarchy properties employed by the existing protocol (e.g., the radius of a group could not exceed 2^i). This, in particular, required implementing in PL-GOSSIP the same group head promotion heuristics as in the other protocol. In practice, we tried to make the implementations share as much code as possible. In effect, PL-GOSSIP maintained a hierarchy with the same properties as the other protocol, but with completely different (gossip-based) mechanisms. Consequently, we obtained a means of systematically comparing the existing hierarchy maintenance mechanisms—notably hierarchical beaconing—with the gossip-based mechanisms introduced by PL-GOSSIP. In addition, by having modified PL-GOSSIP, we confirmed our claims from Section 4.1 that the ideas behind our protocol can indeed be employed to maintain group hierarchies with various sets of properties.

We have compared the two different hierarchy maintenance protocol implementations both in TOSSIM and on our testbed. We were interested to see how the protocols differ in terms of the energy consumption and the latency of bootstrapping and recovering the hierarchy. In contrast to PL-GOSSIP, the previously proposed protocol does not offer any opportunities for aggressive energy conservation and simply assumes that the MAC layer is solely responsible for powering a node's radio down when inactive. Therefore, to compare in a fair manner the energy consumption of that protocol with that of PL-GOSSIP, we used the standard TinyOS 2.0 MAC layer [37], which opportunistically shuts node radios down when inactive while ensuring that a sleeping node always wakes up to hear a message transmission. Employing that MAC layer, however, implies that the resulting energy consumption of PL-GOSSIP can be orders of magnitude higher than when employing a customized MAC layer that makes use of the well-defined periodic traffic pattern offered by the protocol [38], [39]. Since presenting all the details of the experiments is beyond the scope of this paper, we give only a sample of the experimental results in Fig. 11. An interested reader should refer to our subsequent paper [28].

In short, even without the customized MAC layer, PL-GOSSIP maintains the hierarchy more efficiently. In Fig. 11, we can see that when the protocols operate with the same period/round, $T = 10$ minutes, PL-GOSSIP generates fewer messages, but bootstraps the hierarchy more slowly. We can vary T for the protocols to make them either consume the same amount of energy or bootstrap the hierarchy with the same speed. When configured to bootstrap the hierarchy with a similar speed as the other protocol, PL-GOSSIP consumes 33-51 percent less energy. When the two protocols are configured to consume a similar amount of energy, in turn, PL-GOSSIP bootstraps the hierarchy 2.6-3.1 times faster and recovers it after failures even 11.12 times faster [28]. This is because the hierarchy information in PL-GOSSIP is disseminated more efficiently. Instead of using a single message to refresh a single routing entry at a node, as in hierarchical beaconing, in PL-GOSSIP, a single

Experimental Setting / Metric Name	TOSSIM		Testbed	
	Existing Prot.	PL-GOSSIP	Existing Prot.	PL-GOSSIP
Number of Nodes	64		55	
Avg. Number of Neighbors per Node	7.72		19.51	
Avg. Messages per Node per Hour	11.73	6.0	12.44	6.00
Avg. Bytes per Message	12.00	27.46	12.00	24.88
Hierarchy Bootstrap Time [minutes]	108.02	245.02	135 ± 5	235 ± 5

Fig. 11. Comparison of PL-GOSSIP and an existing protocol based on hierarchical beaconing.

received gossip message refreshes multiple routing entries at a node. Consequently, nodes generate fewer messages to maintain the hierarchy, and thus, the MAC layer generates fewer radio on/off events, which are the main cause of overhead on energy consumption. As a result, even without a customized MAC layer that makes use of the well-defined traffic offered by PL-GOSSIP, PL-GOSSIP maintains the group hierarchy in a more efficient way than existing state-of-the-art protocols.

6 Conclusion

In this paper, we considered the problem of maintaining a recursive multihop area hierarchy in large WSNs. We presented a gossip-based protocol that maintains such a hierarchy in a manner that addresses all the peculiarities of WSNs. More specifically, our protocol offers excellent opportunities for aggressive energy saving and facilitates provisioning energy harvesting infrastructure. In addition, it bootstraps and recovers the hierarchy after failures relatively fast while also being robust to message loss. Finally, it seamlessly operates on real sensor node hardware in realistic deployment scenarios. We have confirmed these claims through simulations and experiments with an actual embedded implementation of the protocol. In addition, we have shown that our solution can outperform existing state-of-the-art hierarchy maintenance protocols.

Acknowledgments

The authors would like to express their gratitude to A. Bakker, A. Gaba, M. Szymaniak, and G. Urdaneta for providing the computing power necessary to conduct some of the simulations. Moreover, the authors would like to thank their associate editor and the anonymous reviewers for their feedback, which helped to improve the final version of this paper.

References

[1] F. Kamoun, "Design Considerations for Large Computer Communication Networks," PhD dissertation, Univ. of California, Apr. 1976.

[2] J. Hagouel, "Issues in Routing for Large and Dynamic Networks," PhD dissertation, Columbia Univ., May 1983.

[3] P.F. Tsuchiya, "The Landmark Hierarchy: A New Hierarchy for Routing in Very Large Networks," *ACM SIGCOMM Computer Comm. Rev.*, vol. 18, no. 4, pp. 35-42, Aug. 1988.

[4] S. Kumar, C. Alaettinoglu, and D. Estrin, "Scalable Object-Tracking through Unattended Techniques (SCOUT)," *Proc. Eighth IEEE Int'l Conf. Network Protocols (ICNP '00)*, pp. 253-262, Nov. 2000.

[5] S. Bandyopadhyay and E.J. Coyle, "An Energy Efficient Hierarchical Clustering Algorithm for Wireless Sensor Networks," *Proc. IEEE INFOCOM '03*, pp. 1713-1723, Mar./Apr. 2003.

[6] S. Du, A. Khan, S. PalChaudhuri, A. Post, A.K. Saha, P. Druschel, D.B. Johnson, and R. Riedi, "Self-Organizing Hierarchical Routing for Scalable Ad Hoc Networking," Technical Report TR04-433, Rice Univ., Mar. 2004.

[7] K. Iwanicki and M. van Steen, "Using Area Hierarchy for Multi-Resolution Storage and Search in Large Wireless Sensor Networks," *Proc. IEEE Int'l Conf. Comm. (ICC '09)*, June 2009.

[8] X. Li, Y.J. Kim, R. Govindan, and W. Hong, "Multi-Dimensional Range Queries in Sensor Networks," *Proc. First ACM Int'l Conf. Embedded Networked Sensor Systems (SenSys '03)*, pp. 63-75, Nov. 2003.

[9] C. Sharp, S. Schaffert, A. Woo, N. Sastry, C. Karlof, S. Sastry, and D. Culler, "Design and Implementation of a Sensor Network System for Vehicle Tracking and Autonomous Interception," *Proc. Second European Workshop Wireless Sensor Networks (EWSN '05)*, pp. 93-107, Jan. 2005.

[10] I.F. Akyildiz and I.H. Kasimoglu, "Wireless Sensor and Actor Networks: Research Challenges," *Ad Hoc Networks*, vol. 2, no. 4, pp. 351-367, Oct. 2004.

[11] K. Whitehouse, G. Tolle, J. Taneja, C. Sharp, S. Kim, J. Jeong, J. Hui, P. Dutta, and D. Culler, "Marionette: Using RPC for Interactive Development and Debugging of Wireless Embedded Networks," *Proc. Fifth Int'l Conf. Information Processing in Sensor Networks (IPSN '06)*, pp. 416-423, Apr. 2006.

[12] K. Iwanicki and M. van Steen, "Towards a Versatile Problem Diagnosis Infrastructure for Large Wireless Sensor Networks," *Proc. Second Int'l Workshop Pervasive Systems (PerSys '07)*, pp. 845-855, Nov. 2007.

[13] D. Ganesan, B. Greenstein, D. Perelyubskiy, D. Estrin, and J. Heidemann, "An Evaluation of Multi-Resolution Storage for Sensor Networks," *Proc. First ACM Int'l Conf. Embedded Networked Sensor Systems (SenSys '03)*, pp. 89-102, Nov. 2003.

[14] D. Ganesan, B. Greenstein, D. Estrin, J. Heidemann, and R. Govindan, "Multiresolution Storage and Search in Sensor Networks," *ACM Trans. Storage*, vol. 1, no. 3, pp. 277-315, Aug. 2005.

[15] A. Kansal, J. Hsu, S. Zahedi, and M.B. Srivastava, "Power Management in Energy Harvesting Sensor Networks," *ACM Trans. Embedded Computing Systems*, vol. 6, no. 4, p. 32, Sept. 2007.

[16] K. Iwanicki and M. van Steen, "The PL-Gossip Algorithm," Technical Report IR-CS-034, Vrije Univ., http://www.few.vu.nl/~iwanicki/, Mar. 2007.

[17] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry, "Epidemic Algorithms for Replicated Database Maintenance," *Proc. Sixth Ann. ACM Symp. Principles of Distributed Computing (PODC '87)*, pp. 1-12, Aug. 1987.

[18] R. van Renesse, "Power-Aware Epidemics," *Proc. 21st IEEE Int'l Symp. Reliable Distributed Systems (SRDS '02)*, pp. 358-361, Oct. 2002.

[19] J. Han and M. Kamber, *Data Mining: Concepts and Techniques.* Morgan Kaufmann, 2001.

[20] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-Efficient Communication Protocols for Wireless Microsensor Networks," *Proc. 33rd Hawaii Int'l Conf. System Sciences*, vol. 8, Aug. 2000.

[21] A. Manjeshwar and D.P. Agrawal, "TEEN: A Routing Protocol for Enhanced Efficiency in Wireless Sensor Networks," *Proc. 15th IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS '01 Workshops)*, Apr. 2001.

[22] M. Ye, C. Li, G. Chen, and J. Wu, "EECS: An Energy Efficient Clustering Scheme in Wireless Sensor Networks," *Proc. 24th IEEE Int'l Performance, Computing, and Comm. Conf. (IPCCC '05)*, pp. 535-540, Apr. 2005.

[23] N. Shacham and J. Westcott, "Future Directions in Packet Radio Architectures and Protocols," *Proc. IEEE*, vol. 75, no. 1, pp. 83-99, Jan. 1987.

[24] B. Chen and R. Morris, "L^+: Scalable Landmark Routing and Address Lookup for Multi-Hop Wireless Networks," Technical Report MIT-LCS-TR-837, Mass. Inst. of Technology, Mar. 2002.

[25] L. Subramanian and R.H. Katz, "An Architecture for Building Self-Configurable Systems," *Proc. ACM MobiHoc '00*, pp. 63-73, Aug. 2000.

[26] S.-Y. Ni, Y.-C. Tseng, Y.-S. Chen, and J.-P. Sheu, "The Broadcast Storm Problem in a Mobile Ad Hoc Network," *Proc. ACM MobiCom '99*, pp. 151-162, Aug. 1999.

[27] P. Levis, N. Patel, D. Culler, and S. Shenker, "Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks," *Proc. First USENIX Symp. Networked Systems Design and Implementation (NSDI '04)*, pp. 15-28, Mar. 2004.

[28] K. Iwanicki and M. van Steen, "Multi-Hop Cluster Hierarchy Maintenance in Wireless Sensor Networks: A Case for Gossip-Based Protocols," *Proc. Sixth European Conf. Wireless Sensor Networks (EWSN '09)*, pp. 102-117, Feb. 2009.

[29] K. Iwanicki and M. van Steen, "On Hierarchical Routing in Wireless Sensor Networks," *Proc. Eighth ACM/IEEE Int'l Conf. Information Processing in Sensor Networks (IPSN '09)*, pp. 133-144, Apr. 2009.

[30] D. Thaler and C.V. Ravishankar, "Distributed Top-Down Hierarchy Construction," *Proc. IEEE INFOCOM '98,* pp. 693-701, Mar./Apr. 1998.
[31] J. Polastre, J. Hui, P. Levis, J. Zhao, D. Culler, S. Shenker, and I. Stoica, "A Unifying Link Abstraction for Wireless Sensor Networks," *Proc. Third ACM Int'l Conf. Embedded Networked Sensor Systems (SenSys '05),* pp. 76-89, Nov. 2005.
[32] K. Iwanicki, A. Gaba, and M. van Steen, "KonTest: A Wireless Sensor Network Testbed at Vrije Universiteit Amsterdam," Technical Report IR-CS-045, Vrije Univ., http://www.few.vu.nl/~iwanicki/, Aug. 2008.
[33] B. Leong, B. Liskov, and R. Morris, "Geographic Routing without Planarization," *Proc. Third USENIX Symp. Networked Systems Design and Implementation (NSDI '06),* pp. 339-352, May 2006.
[34] J. Newsome and D. Song, "GEM: Graph EMbedding for Routing and Data-Centric Storage in Sensor Networks without Geographic Information," *Proc. First ACM Int'l Conf. Embedded Networked Sensor Systems (SenSys '03),* pp. 76-88, Nov. 2003.
[35] Y. Mao, F. Wang, L. Qiu, S.S. Lam, and J.M. Smith, "S4: Small State and Small Stretch Routing Protocol for Large Wireless Sensor Networks," *Proc. Fourth USENIX Symp. Networked Systems Design and Implementation (NSDI '07),* pp. 101-114, Apr. 2007.
[36] A. Woo, T. Tong, and D. Culler, "Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks," *Proc. First ACM Int'l Conf. Embedded Networked Sensor Systems (SenSys '03),* pp. 14-27, Nov. 2003.
[37] J. Polastre, J. Hill, and D. Culler, "Versatile Low Power Media Access for Wireless Sensor Networks," *Proc. Second ACM Int'l Conf. Embedded Networked Sensor Systems (SenSys '04),* pp. 95-107, Nov. 2004.
[38] T. Melodia, M.C. Vuran, and D. Pompili, "The State of the Art in Cross-Layer Design for Wireless Sensor Networks," *Proc. Second EuroNGI Workshop Wireless and Mobility,* pp. 78-92, July 2005.
[39] P. Dutta, D. Culler, and S. Shenker, "Procrastination Might Lead to a Longer and More Useful Life," *Proc. Sixth ACM Workshop Hot Topics in Networks (HotNets-VI),* Nov. 2007.

Konrad Iwanicki is currently working toward the PhD degree in the Computer Systems Group, Vrije Universiteit Amsterdam. He is a student member of the ACM, the IEEE, and the IEEE Computer Society. His research interests include large-scale distributed systems and network protocols, especially for low-power wireless networks. His recent research activities have been focused on scalable point-to-point routing for low-power wireless networks. In particular, he has been investigating the applicability of hierarchical routing and other small-state routing protocols to wireless sensor networks. Prior to his current position, he was working on a scalable archival storage system in the Robust and Secure Systems Group of NEC Laboratories America, Inc. The system he helped create has recently become a successful commercial product.

Maarten van Steen is full professor in the Computer Systems Group, Vrije Universiteit Amsterdam. He teaches modules and courses covering distributed systems, computer networks, operating systems, and complex networks to academics and professionals. He has coauthored two textbooks on networked computer systems. His research is focused on large-scale distributed systems with a strong emphasis on adaptive techniques that support automatic replication, management, and organization of wired and wireless systems. Recently, he has been exploring gossip-based solutions to achieve decentralized autonomous systems, partly focusing on very large wireless sensor networks and pervasive computing. Furthermore, he is a consultant for Philips Research, and closely participates with a collaboration of high-tech SMEs for developing and deploying real-world pervasive computing systems. He is a senior member of the IEEE and the IEEE Computer Society.