Multi-hop Cluster Hierarchy Maintenance in Wireless Sensor Networks: A Case for Gossip-Based Protocols

Konrad Iwanicki[1,2] and Maarten van Steen[1]

[1] Vrije Universiteit, Amsterdam, The Netherlands
[2] Development Laboratories (DevLab), Eindhoven, The Netherlands
{iwanicki,steen}@few.vu.nl

Abstract. Multi-hop cluster hierarchy has been presented as an organization for large wireless sensor networks (WSNs) that can provide scalable routing, data aggregation, and querying. In this paper, we revisit the fundamental problem of maintenance of such a hierarchy. To this end, we observe that, due to tightly coupling their operation with the topology of the hierarchy, existing state-of-the-art cluster hierarchy maintenance protocols may not necessarily be efficient. Based on our observations, we make a case for a novel gossip-based hierarchy maintenance protocol that decouples its operation from the hierarchy topology. Through experiments with actual embedded implementations we have developed, we confirm that our protocol can outperform the existing state-of-the-art solutions by a few factors in terms of both the energy consumption and the latency of bootstrapping and recovering the hierarchy.

1 Introduction

Many of the proposed wireless sensor network (WSN) applications assume large node populations deployed over sizable areas. Due to inherent resource limitations of most of wireless sensor node platforms, large multi-hop WSNs must be organized in a way that enables scalable routing, data aggregation, and querying. Moreover, to minimize the effort involved in the deployment and upkeep of a large WSN, it is highly desirable that the organization should be maintainable with minimal human intervention.

Multi-hop cluster hierarchy is a prominent example of such an organization [1,2,3,4]. This hierarchy is a virtual multi-level overlay on the physical network topology: based on their connectivity, nodes are grouped into clusters at level 1, which in turn are grouped into superclusters at level 2, and so on at higher levels. The overlay provides addressing and routing mechanisms, which require only $O(logN)$ node state, thereby offering excellent scalability. These basic mechanisms can be further used to build more advanced services such as distributed hash tables [1,4] or multi-level in-network aggregation and querying [3,5]. Moreover, the overlay can be synthesized and maintained without human intervention, thereby minimizing deployment efforts and facilitating unattended operation. For these reasons, a number of WSN application proposals can be founded on a multi-hop cluster hierarchy, for example, object tracking [1], reactive tasking [6], energy-efficient centralized data collection [3], scalable network monitoring [7], and multi-dimensional range querying [5], to name a few. By and large, multi-hop cluster hierarchy is a compelling paradigm for organizing nodes in large WSNs.

U. Roedig and C.J. Sreenan (Eds.): EWSN 2009, LNCS 5432, pp. 102–117, 2009.

The key component in a system based on a multi-hop cluster hierarchy is the hierarchy maintenance protocol. Such a protocol not only determines the target hierarchy properties and thereby the costs of routing, data collection, and querying, but also generates additional costs due to maintaining these properties during system lifetime. The maintenance costs can easily account for the majority of energy consumed by the nodes. Therefore, since WSNs operate on tight energy budgets, it is essential that the energy costs of hierarchy maintenance are minimized. Even a few-percent improvement in energy consumption can substantially reduce the expenses involved in the upkeep of a large network or can even enable applications that require the prolonged lifetime. Likewise, the hierarchy maintenance protocol should minimize the time to bootstrap the hierarchy and recover it after changes in the network. This affects the availability of the overlay services (e.g., routing, aggregation, querying), especially since large WSNs are less stable in terms of node population and connectivity than their smaller counterparts. The challenge in cluster hierarchy maintenance is the fact that minimizing energy consumption and minimizing bootstrap and recovery time are typically conflicting goals.

In this paper, we revisit the problem of multi-hop cluster hierarchy maintenance in large WSNs with the goal of minimizing the energy consumed to maintain the hierarchy and the time to bootstrap and recover the hierarchy. In contrast to prior work, which focuses on clustering heuristics, we revisit the common general protocol scheme.

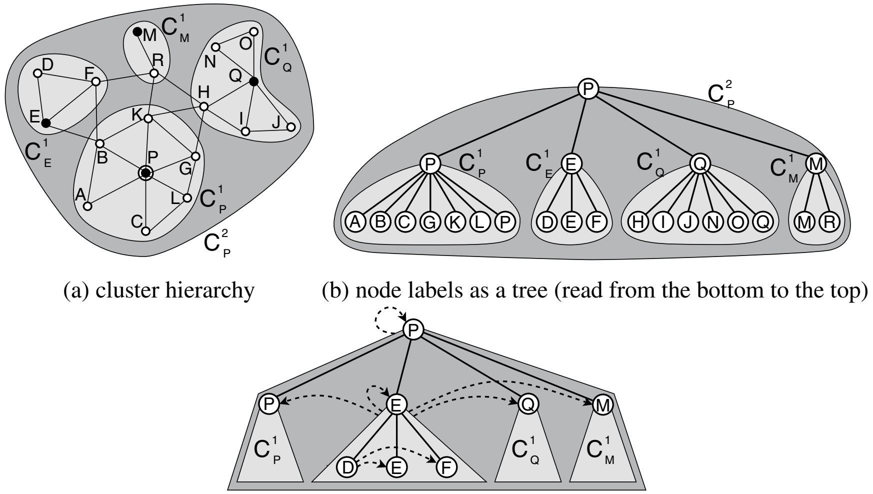
More specifically, we observe that maintaining a hierarchy with the existing state-of-the-art protocols for WSNs is expensive in terms of energy. This is because the operation of all these protocols is *tightly coupled* with the topology of the hierarchy, leading to many small messages being exchanged. In WSNs, however, small messages are typically inefficient, as they incur significant energy overhead on the actual protocol data.

Based on this observation, we propose an alternative protocol that, to improve the performance of hierarchy maintenance, *decouples* its operation from the topology of the hierarchy. The protocol employs a combination of local operations for updating the hierarchy and periodic local gossiping (i.e., asynchronous state exchanges between neighboring nodes) for propagating updates and advertising clusters. This results in more efficient traffic: irrespective of the hierarchy topology, each node periodically transmits only a single big message. Through experiments with actual embedded implementations, we show that our protocol can outperform the existing state-of-the-art protocols, while providing the same hierarchy properties.

The rest of the paper is organized as follows. We formulate the problem and survey prior work in Sect. 2. We explain our protocol in Sect. 3 and evaluate it against existing solutions in Sect. 4. Finally, in Sect. 5, we conclude. Due to space constraints, the code listings and supporting proofs have been moved to a technical report [8].

2 Background and Related Work

Since multi-hop cluster hierarchies have many potential applications in WSNs and other environments, they have received significant research attention. Our protocol builds upon prior work by allowing for using most of the existing clustering heuristics, developed and carefully tuned for particular WSN applications or deployment settings. In this way, the protocol can maintain hierarchies with the same properties as in the existing solutions, and thus, it can be used in the same applications. However, our protocol



(a) cluster hierarchy

(b) node labels as a tree (read from the bottom to the top)

(c) routing table of node D (label $D.E.P$)

Fig. 1. An example of a multi-hop cluster hierarchy

follows a novel design paradigm that results in a completely different general operation scheme as compared to the existing protocols. As we show in this paper, this new scheme can improve the efficiency of hierarchy maintenance by significantly reducing the energy consumption and the hierarchy bootstrap and recovery latencies.

In the remainder of this section, we first formulate the problem and then survey prior work. For illustration purposes, we assume a common set of target cluster hierarchy properties [1,2,3,4]. However, as mentioned above and signaled in the protocol description, one can easily obtain different hierarchy properties by simple modifications to the protocol and by employing different clustering heuristics.

2.1 Problem Formulation

A multi-hop cluster hierarchy (see Fig. 1a) consists of multiple levels, on the order of $O(logN)$ with respect to the number of nodes. A node belongs to exactly one cluster at each level, with level-0 singleton clusters that correspond to individual nodes and one or a few top-level clusters that contain all nodes. Each cluster has a *cluster head* that maintains the cluster and can have other roles in some applications, for instance, as an aggregator node for the cluster. Assuming that each node has a unique identifier, a cluster, C^i_X, is uniquely identified by its level, i, and the identifier of its head, X.

The cluster hierarchy is reflected in the *labels* of the nodes. A node's label is a concatenation of the cluster head identifiers for all the clusters the node is member of (see Fig. 1b). For instance, the label of node D from Fig. 1, which is a level-0 cluster head, is $L(D) = D.E.P$ as D belongs to clusters: C^0_D, C^1_E, and C^2_P. The label of node E, a level-1 cluster head, is $L(E) = E.E.P$ as E belongs to clusters: C^0_E, C^1_E, and C^2_P. Finally, the label of node P, a level-2 cluster head, is $L(P) = P.P.P$ as P belongs to clusters: C^0_P, C^1_P, and C^2_P.

Based on its label, each node also keeps a *$O(logN)$ hierarchical routing table* [9]. At each level, a node's routing table contains entries for the heads of sibling clusters at this level (see Fig. 1c). With node labels acting as routing addresses, the routing tables of all nodes enable hierarchical point-to-point routing [1,9]. Since routing is not used by our protocol but only by the applications on top, we omit the algorithm for brevity.

The *cluster hierarchy maintenance protocol*, therefore, is responsible for *maintaining the labels and the routing tables of all the nodes throughout the system's lifetime.* The longevity of a WSN system and its interactions with the physical environment imply changes in the node population and connectivity over time. Hence, to account for the changes, the hierarchy maintenance must be performed continuously rather than just upon system deployment. Such continuous maintenance must *consume minimal amounts of energy* and *provide fast hierarchy bootstrap and recovery after changes.*

2.2 Prior Work

In earlier work on hierarchical routing [9], Hagouel proved that constructing an optimal cluster hierarchy is an NP-complete problem. Thus, only heuristic solutions are practical. Many such heuristics have been developed in data mining for partitioning data sets with respect to a given parameter [10]. Those algorithms, however, require centralized control and fail to scale to large networks of resource-constrained devices.

For this reason, distributed protocols have been introduced. One family of such protocols provides single-level (flat) clustering [11,12,13]. While flat clustering is important in WSN applications using local sensor collaboration for event detection, the definition of cluster hierarchy implies multiple levels. Yet, flat clustering algorithms cannot be easily generalized to efficiently support a multi-hop cluster hierarchy, as their traffic pattern precludes multiple levels and long multi-hop internode distances.

Another family of clustering protocols aims at reducing the energy cost of centralized data collection in dense WSNs [14,15,16]. To this end, the nodes maintain a cluster hierarchy by dynamically adjusting their transmission power and thereby communication range, such that on average the energy consumed by data transmission is minimal. All these algorithms assume that by increasing transmission power, each node can directly communicate with any other node (i.e., one-hop communication). However, while this assumption may hold in WSNs deployed densely in small areas, for practical reasons, it does not hold hold in large-scale real-world WSNs.

These limitations led to a family of cluster hierarchy maintenance protocols aimed specifically at large multi-hop networks of low-power wireless devices [1,2,3,4]. In these protocols, nodes self-organize into a recursive hierarchy of clusters by grouping connected nodes into clusters, grouping clusters into superclusters, and so on. Such a bottom-up approach is better suited for WSNs than an alternative top-down approach [17], which has difficulties with varying deployment parameters, like node densities [1].

Although these state-of-the-art bottom-up protocols vary in clustering heuristics, they follow the same general scheme. In this scheme, nodes advertise their clusters periodically, so that other nodes join such clusters or probabilistically spawn their own higher-level clusters, thereby constructing the hierarchy. The whole process of cluster advertising and update propagation is founded on *hierarchical beaconing*, which is a multi-hop adaptation of hierarchical clustering for dense one-hop networks [14,15,16].

In hierarchical beaconing, a level-i cluster head periodically advertises its cluster by issuing a beacon message that is flooded over R_i hops. To ensure $O(logN)$ hierarchy levels, R_i depends exponentially on i (typically $R_i = 2^i$). Issuing a level-i beacon is expensive, as all nodes within R_i hops from the cluster head must forward the beacon. Therefore, the protocols usually amortize the costs of higher-level beacon forwarding over time by making the number of periods (rounds) between subsequent beacons proportional to the beacon propagation radius. For example, a level-0 cluster head issues a R_0-hop beacon every R_0 rounds, a level-1 cluster head — a R_1-hop beacon every R_1 rounds, and so on. This reduces the beacon forwarding cost at the expense of an increase in the hierarchy bootstrap and recovery latency. To mitigate this increase, a cluster head also issues a beacon whenever it modifies its label, to propagate the update fast.

3 Gossip-Based Hierarchy Maintenance

While these state-of-the-art protocols are elegant, their efficiency, crucial for most applications, can be improved considerably. To conserve energy on idle listening, sensor nodes power their radios off when idle. Such radio activity scheduling, however, incurs significant energy overhead on message transmission or reception, as the sender and receivers must coordinate to have their radios on during the transmission. Very often this overhead outweighs the cost of actual data transmission. For instance, the standard TinyOS MAC layer for WSNs [18] precedes a message with a 100- to 2000-ms preamble, which is a lot compared to less than 0.4 ms necessary to transmit a 12-byte beacon payload (in our implementation). Energy-efficient protocols should thus minimize the overhead on the exchanged data by using fewer but longer messages [18,19].

However, this is essentially not happening for hierarchical beaconing, which generates myriads of small messages. In preliminary experiments for a hierarchical cluster-based system we are building, for example, to ensure reasonable data availability in the presence of failures, beacon messages could account for more than 90% of energy spent on communication. This also illustrates that minimizing energy overhead incurred by the hierarchy maintenance protocol is crucial for any hierarchical cluster-based system.

3.1 Principal Idea

We observe that the protocols using periodic hierarchical beaconing are so expensive because hierarchical beaconing is *tightly coupled* with the cluster hierarchy. Every beacon is dedicated for one cluster and at every level, i, each node forwards beacons of all level-i heads that are within R_i hops.

Consequently, to reduce the cost of cluster hierarchy maintenance, we propose a protocol that *decouples* its operation from the topology of the hierarchy. Our protocol is based on a combination of local-only operations for updating the hierarchy and periodic local gossiping (i.e., asynchronous state exchanges between neighboring nodes) for propagating such updates and advertising clusters.

The protocol operates in rounds[1]. Once per round, each node broadcasts its protocol state in a single heartbeat message. The state consists of the node's label, update vector

[1] The rounds are local for each node, that is, the node clocks do not have to be synchronized.

(Sect. 3.2), and routing table entries. The heartbeat message is local: it is received only by the node's neighbors and is not forwarded by them. Likewise, in every round, these neighbors broadcast their own state in their heartbeats. The received neighbor state is merged with the node's own local state, so that the node can learn about any changes that have recently occurred in the hierarchy. At the end of its round, the node checks its local state to detect any such changes and to account for them by modifying its state locally. The modified state is broadcast in the node's heartbeat in the next round.

Hierarchy information is thus propagated implicitly, by repeatedly merging the node's state with the state received from its neighbors and broadcasting such a merged state in the next heartbeat message. This is in contrast to periodic hierarchical beaconing, in which the information is propagated explicitly, by forwarding multiple beacon messages dedicated for specific clusters. The result of this paradigm shift is smaller energy overhead on protocol data: instead of forwarding multiple small beacons per round, each node transmits only a single big heartbeat message.

The paradigm shift, however, requires revisiting the mechanisms for maintaining the hierarchy, which is our focus in the rest of this section. Since formally describing the protocol is outside the scope of this paper, we aim only at explaining it adequately. The pseudo-code and proofs, in turn, can be found in our technical report [8].

3.2 Update Vector

One of the key revisited mechanisms is hierarchy membership update resolution. Since in hierarchical beaconing every member of a cluster receives beacons issued by the head of this cluster, it has direct access to the label of the head. Therefore, when the head modifies its label and issues a beacon, the cluster members can consistently apply the updates to their labels. In contrast, with local gossiping, a node has access only to the labels of its immediate neighbors, and there is no way to determine which of the neighbors' labels contain fresher membership information. For example, without additional information, a node with label $A.K.$P$.R.S$, receiving from its neighbors heartbeats with labels $B.K.$P$.Q.T$ and $C.J.$P$.U.V$, cannot determine which of the three labels contains the freshest information on the membership of cluster C_P^2 in the hierarchy.

To solve 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 i-th 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 cluster head, as illustrated in Fig. 2. A node's

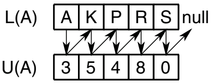
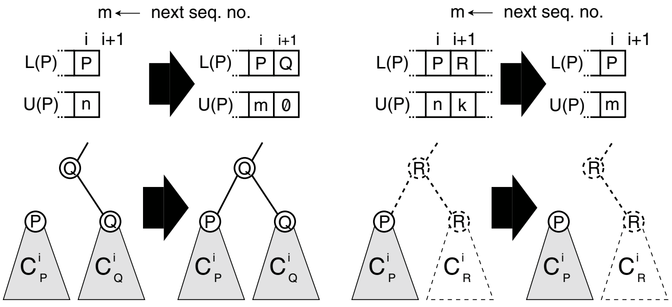


Fig. 2. A label and update vector. *A knows that: (i) the last level-*1 *label update of A has number* 3 *and wrote K at position* 1*; (ii) the last level-*2 *update of K has number* 5 *and wrote P at position* 2*; (iii) the last level-*3 *update of P has number* 4 *and wrote R at position* 3*; (iv) the last level-*4 *update of R has number* 8 *and wrote S at position* 4*; (v) S has not yet made any level-*5 *updates (*$U(A)[4] = 0$*).*



(a) label extension (b) label cut

Fig. 3. Label extension and label cut

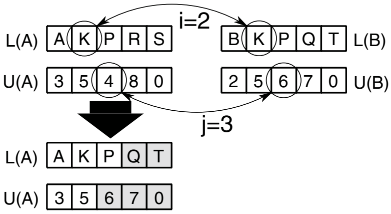


Fig. 4. Label resolution and update adoption

update vector is broadcast with the node's label in the node's heartbeat messages and is essential to synthesizing and maintaining node labels.

3.3 Basic Label Operations

To maintain their labels, the nodes use three basic operations: label extension, label cut, and label resolution. Label extension (see Fig. 3a) is executed locally by a top-level cluster head, P, during hierarchy construction or recovery. It corresponds to joining cluster C^i_P to a higher-level cluster, C^{i+1}_Q (if P's label is extended with Q), or spawning a new higher-level cluster, C^{i+1}_P (if P's label is extended with P). Label cut (see Fig. 3b), in turn, is executed locally by a non-top-level head, P, during hierarchy recovery when P detects that its supercluster, C^{i+1}_R, no longer exists as, for example, its head, R, may have died. This operation corresponds to removing cluster C^i_P from the no longer existing higher-level cluster, C^{i+1}_R. In both operations, when modifying its label at level i+1, node P also writes a new sequence number at the i-th position of its update vector. This is to indicate that this label update by P is the freshest one, so that other members of C^i_P can also adopt the update through label resolution.

Label resolution is thus the way to propagate label updates in our protocol. It is done every time a node, A, receives a heartbeat from a neighbor, B, and works as follows (see Fig. 4). A checks if it shares a cluster with B, that is, A looks for the minimal i such that $L(A)[i] = L(B)[i]$. If such i exists, A checks which of the labels is fresher by comparing its update vector, $U(A)$, with B's update vector, $U(B)$, from position i. If for some $j \geq i$, $U(A)[j] \neq U(B)[j]$ then one of the labels is fresher than the other (they can differ 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 from position $j{+}1$ and j, respectively: $L(A)[j{+}1 \ldots \mathcal{H}] \leftarrow L(B)[j{+}1 \ldots \mathcal{H}]$ and $U(A)[j \ldots \mathcal{H}] \leftarrow U(B)[j \ldots \mathcal{H}]$. In this way, A's hierarchy membership information becomes consistent with the fresher information from B. Moreover, when A broadcasts its next heartbeat, its other neighbors also adopt the fresh information, and so on. In our technical report [8], we formally prove that this algorithm guarantees consistent label update adoption by all cluster members.

3.4 Maintaining Routing Tables

While label resolution is performed for every received heartbeat, the usage of label extension and cut is dependent on the state of a node's routing table. Node routing tables are maintained with a custom distance-vector algorithm.

A routing entry corresponds to a cluster (cf. Sect. 2.1). It consists of the level and the identifier of the cluster head, a sequence number, the link-layer address of a next-hop neighbor on the path to the head, the number of hops on this path, and an additional bit indicating whether the cluster is adjacent to the present node's cluster at the same level.

An entry for a cluster originates at the cluster head, which sets the hop count of the entry to zero and generates a new sequence number for the entry. Generating the sequence numbers follows the same pattern as issuing beacon messages in hierarchical beaconing: a level-i head generates a new sequence number for its cluster entry every R_i rounds. Alternatively, a new sequence number can be generated in each round.

Clusters are advertised via heartbeat messages. When the head broadcasts a heartbeat, its neighbors can refresh the entries for the head's cluster with a new sequence number. When they broadcast their heartbeats, their neighbors can also refresh their routing entries, and so on up to R_i hops. More specifically, a heartbeat broadcast by a node contains those entries from the node's routing table that were refreshed in the past round (or k rounds if we want to tolerate message loss). A node receiving a heartbeat refreshes those entries in its routing table that are present in the heartbeat and have fresher sequence numbers than the node's own entries. In addition, since the same routing entry can be present in heartbeats of different neighbors, the node must choose one of the neighbors as the next hop for the routing entry. As in the distance-vector algorithm, it chooses the neighbor whose routing entry has had the smallest hop count.

When a node does not refresh a routing entry for a certain number of rounds, depending on R_i, it concludes that the cluster corresponding to the entry is no longer reachable, for instance, because the head died. Consequently, the entry can be removed from the node's routing table. To prevent routing cycles in the presence of node failures, we use route poisoning: before removing an entry a node marks it as unreachable and broadcasts such an entry in its heartbeat messages for several rounds, thereby allowing other nodes to learn about the failure as well.

For a given cluster hierarchy, the routes maintained by the above algorithm are the same as the routes maintained by periodic hierarchical beaconing. This is because the *rules* for constructing the routes are the same. However, the protocols employ completely different *paradigms* for propagating route information. In periodic hierarchical beaconing, every routing entry of a node is refreshed by a separate beacon message received by the node. In our protocol, in contrast, a single heartbeat message can refresh multiple routing entries of the receiving node. As a result, our protocol maintains the same routing tables more efficiently as we demonstrate empirically in the next section.

3.5 Synthesizing and Maintaining Labels

Based on its routing table, each node synthesizes and maintains its label with the three basic label operations. During a round, a node receives heartbeat messages from its neighbors. The heartbeats are used for refreshing the node's routing table and for resolving and adopting any label updates made by the heads of the node's clusters. At the end of the round, the node, being itself a cluster head at some level, analyzes its routing table and, as a result, possibly extends or cuts its label locally to account for any network changes that have occurred. The possibly updated node label is broadcast in the node's heartbeat in the next round, allowing the members of the node's cluster to gradually adopt the label updates. By repeating this scheme in every round, the nodes synthesize and continuously maintain their labels.

Label Synthesis. Initially, each node is a top-level head of its level-0 cluster, as its label contains only one element. If the node discovers in its routing table an entry for another cluster head at the same or a higher level, it has to either spawn a new higher-level cluster itself or join the higher-level cluster of the other node. In the first case, it would extend its label with its own identifier, promoting itself to a higher-level head. In the second case, it would extend its label with the identifier of the other node (see Fig. 3a).

Joining an existing cluster is preferred, as it decreases the number of clusters at subsequent levels. However, depending on clustering heuristics, joining may not always be possible. In that case, our protocol utilizes the same heuristics as the existing state-of-the-art protocols [1,2,3,4]. More specifically, the node probabilistically defers spawning the higher-level cluster by drawing a random promotion slot and waiting for this slot. If during this time other nodes in a similar situation spawn new higher-level clusters, the node may be able to join one of such clusters. Otherwise, the node spawns its own higher-level cluster. In any case, when the node broadcasts its heartbeat after extending its label, the node's neighbors that belong to the node's cluster can also extend their labels through the label resolution operation. When they broadcast their heartbeats, their neighbors that belong to the node's cluster can extend their labels as well and so on. In our technical report, we prove that this ensures probabilistic label convergence [8].

Label Recovery. When a node has died, it no longer advertises its cluster. Thus, other nodes do not refresh the routing entries for that cluster. If a node has not refreshed a routing entry for a certain number of rounds, the entry is evicted from the node's routing table. If a node, being a level-i cluster head, discovers that the entry for its parent level-i+1 head has been evicted, it concludes that its cluster must not be a subcluster of the no longer existing level-i+1 cluster. Hence, it cuts its label down to level i (see Fig. 3b). Later, by virtue of the above label synthesizing mechanisms, the dangling cluster of this node will join to some other higher-level cluster, completing the recovery.

By employing the same probabilistic clustering heuristics as the existing state-of-the-art protocols, our solution constructs the same cluster hierarchies as those protocols. One of the consequences of building upon these well-established heuristics is that our protocol can be utilized in the same applications, for instance, to improve performance.

Table 1. Memory breakdown of the test application for a TelosB node. *Both protocols require little memory, with their RAM footprints being dominated by the pools for the routing table entries. Due to the more sophisticated operation scheme, however, our protocol (Decoupled) requires slightly more code and data space. The total data space, in turn, is higher for the application with the other protocol (Coupled), as it requires more message buffers (for beacon messages).*

Variant /	**Coupled**		**Decoupled**	
Component	RAM	ROM	RAM	ROM
Protocol Only	1,382 B	3,600 B	1,469 B	4,269 B
Total	4,408 B	22,452 B	4,094 B	23,774 B

4 Experimental Evaluation

We evaluate the existing state-of-the-art solutions against our approach using actual embedded implementations. To the best of our knowledge, this is the first reported implementation-based evaluation of multi-hop cluster hierarchy maintenance protocols for WSNs. While prototyping our protocol, we also conducted extensive experiments with a custom simulator. Those results are presented in our technical report [8].

4.1 Protocol Implementations

The discussed existing multi-hop cluster hierarchy maintenance protocols for WSNs [1,2,3,4] operate according to the same common scheme, founded on hierarchical beaconing, but differ slightly in clustering heuristics. Therefore, we have implemented the most recent protocol with arguably the most efficient heuristics [3]. To enable fair comparison, we have used the same heuristics in the implementation of our solution. In general, where possible, both implementations use precisely the same components. As a result, both the implementations build hierarchies with the same target properties, and differ only in mechanisms for maintaining these properties. We believe that this is a sound approach to show the performance gains that can be obtained with our protocol.

For the purpose of the evaluation, we have written a simple test application that contains only statistic reporting functionality and either of the protocol implementations. The test application enables testing hierarchy maintenance in isolation from other services present in a complete real system. We leave the evaluation of a complete representative system for future work. In the remainder of this section, we use *Coupled* to refer to the variant of the application with the state-of-the-art protocol based on periodic hierarchical beaconing, and *Decoupled* to refer to the variant with our protocol.

As the implementation platform for the protocols and the test application, we have chosen TinyOS 2.0. To estimate link quality, which is necessary for discriminating node neighbors from poorly connected nodes, we use the standard link estimator based on exponentially weighted moving average of packet reception rate [20]. This estimator is fast and portable, and offers acceptably accurate link estimates. As the MAC layer, we use the standard TinyOS 2.0 MAC, that is, CSMA/CA with low-power listening [18]. This MAC layer is well suited for hierarchical cluster-based systems as it provides low energy consumption, scales to large networks, and efficiently handles varying workloads and network dynamics. The memory breakdown of the test application for each of the hierarchy maintenance protocols is presented in Table 1.

4.2 Experimental Setup

We have conducted our experiments on a small indoor testbed consisting of 55 TelosB nodes [21] and in TOSSIM, a low-level TinyOS simulator. In both environments, we used precisely the same test application, described above. Since the evaluated protocols display most of their scaling properties only in large networks, starting from some hundred nodes [1,2,3,4,8], due to space constraints, in this paper we focus mostly on the TOSSIM experiments and only briefly summarize the small-scale testbed results.

TOSSIM provides a realistic and accurate simulation environment for TinyOS applications. It captures the TinyOS behavior at a low level and offers signal propagation and noise models derived from real deployments. This enables accurate evaluation of the protocols in realistic settings that give good predictions on the real-world protocol behavior, as we have seen in the testbed experiments.

We have conducted TOSSIM experiments in a number of network configurations. To evaluate protocol scalability, we exponentially varied the node population from 64 to 1024 nodes. The nodes were placed on a square grid. By also varying the grid spacing, we obtained different node densities: from 11.27 (sparse) to 46.42 (dense) good-quality neighbors per node on average. Using traces from real-world deployments and TOSSIM tools, for each configuration, we generated a realistic signal propagation model. The resulting connectivity exhibited many irregularities that are common in real-world WSNs. For example, nearby nodes were often not connected and there were many asymmetric links. As a result, the grid node placement was not mirrored by the neighbor relation. This alleviates the problem that due to page constraints on this paper, we omit other tested topologies (i.e., uniform and random).

In all configurations, a node was identified with 10 bits. The top hierarchy level was 5, as this was enough for a top-level cluster to cover the whole network. Consequently, the label size in a beacon or heartbeat message was at most $\lceil (10 \cdot (5+1))/8 \rceil = 8$ bytes. This was also the size of the update vector in a heartbeat message. A single routing table entry in a heartbeat message, in turn, was 4 bytes long.

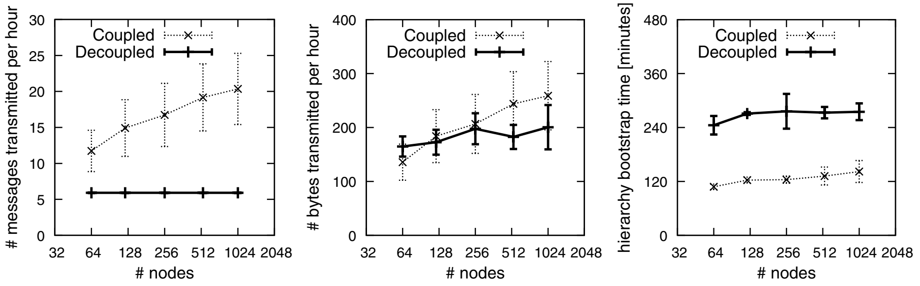
4.3 Experimental Results

When maintaining the desired properties of a multi-hop cluster hierarchy, the goal is to minimize both the energy consumption and the bootstrap/recovery latency. Since these two are often in conflict, our target performance metric is the energy consumption per given time period versus the latency of bootstrapping and recovering the hierarchy.

We start with standard experiments in which all nodes are booted simultaneously and have to construct the hierarchy from scratch. For both protocols, the round length, T, is equal to 10 minutes. This allows us to illustrate the strengths and weaknesses of each of the protocols. Selected results of the experiments are presented in Fig. 5.

As we argued in the previous section, the Coupled protocol uses substantially more messages than our Decoupled protocol (Fig. 5a). In a sparse 1024-node network, for example, it generates more than 20 messages per node per hour (> 3 messages per round), and this value varies significantly between nodes. Our Decoupled protocol, in turn, produces small flat traffic of 6 messages per node per hour (1 message per round).

Likewise, because it advertises clusters less efficiently, the Coupled protocol requires more bandwidth in larger networks (Fig. 5b). Every beacon message has two coupled



(a) messages per node per hour (b) bytes per node per hour (c) bootstrap latency

Fig. 5. Protocol scaling properties. *Each point represents the average and the standard deviation over 10 runs. The round length, T, is* 10 *minutes for both protocols. The networks are sparse. The results for denser networks do not differ significantly. Note also a logarithmic scale of the x-axes.*

objectives: consistently propagating hierarchy membership updates for a cluster and advertising the cluster to refresh node routing tables. To this end, apart from a hop count and a sequence number, a beacon has to store the full label of the cluster head (up to 8 bytes in our implementation). In our Decoupled protocol, in contrast, propagating cluster advertisements is independent of update propagation, which is performed through label resolution. Therefore, instead of a full label, an advertisement of a cluster corresponds only to the routing entry for the cluster (4 bytes in our implementation). Note also that due to maintaining the same target hierarchy properties, both protocols generate roughly the same number of cluster advertisements. Consequently, since in larger networks cluster advertisements dominate the payload of heartbeat messages, our protocol requires less bandwidth, even though it incurs some additional overhead due to transmitting neighbor labels and update vectors. Moreover, due to transmitting less messages, our approach saves bandwidth on TinyOS message headers and footers. These results, however, are not included in the plot.

For the hierarchy bootstrap latency, the relationship is opposite (Fig. 5c). Since a beacon is forwarded by a node shortly after it is received, hierarchy updates in the Coupled protocol propagate fast. In contrast, because heartbeats are broadcast by a node only once per round, update propagation through label resolution is slower. In the most pessimistic scenario, it can take up to d rounds to propagate an update over d hops. For this reason, given the same round length, the Decoupled protocol bootstraps the hierarchy slower than the Coupled one. In the networks plotted, this difference is roughly a factor of two. The bootstrap latencies in the plot seem flat because for implementation purposes we limited the top level to 5. If this is not the case and the bootstrap criterion is a single top-level cluster, the latency grows with the network size [8]. Moreover, the latency can change depending on the number of rounds the link estimator requires to identify good-quality links. In our implementation, this number ranged from 5 to 6.

Since the reduction in energy consumption is typically sublinear with respect to the reduction in the traffic volume, to avoid exaggerating the performance improvements of our protocol, we directly compare the energy consumed by both the protocols. We obtain the energy consumption by following a standard methodology for the underlying MAC layer [18,19], which produces relatively accurate results. More specifically,

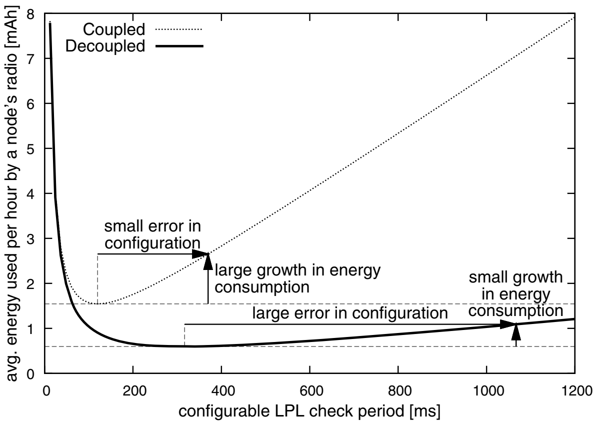


Fig. 6. Energy consumption as a function of the LPL check period. *The round length, T, is* 10 *minutes for both protocols. The network consists of* 1024 *densely deployed nodes. Again, for sparse networks the results are consistent. The current draw measurements correspond to* 250-*kbit/s IEEE 802.15.4 radios (as in TelosBs) and come from Klues et al. [19]. The results for slower radios, such as in Mica2s, are consistent, albeit the difference is slightly smaller.*

we combine radio activity traces from the above experiments with publicly available measurements of the current drawn by a node's radio in various sensor node platforms. Sample results for the TelosB platform are depicted in Fig. 6.

To conserve radio energy, the standard MAC layer for TinyOS 2.0 employs a technique called low-power listening (LPL) [18]. LPL involves one configurable parameter, the LPL check period, which determines how often the radio is turned on to check for a carrier and how long the message preamble is. This parameter reflects the trade-off between energy consumption when the radios are idle and the energy overhead on data transmission. Figure 6 shows that, for the same round length, our Decoupled protocol outperforms the Coupled one for all reasonable settings of the LPL check period. In particular, with the LPL check period configured optimally for each of the protocols, our protocol requires 2.5 times less energy than the Coupled protocol. Moreover, in the Coupled protocol even a small deviation from the optimal setting results in large growth of the energy consumption. This, however, is undesirable as such deviations are expected in the real world because experience shows that WSN applications often exhibit traffic patterns that are difficult to predict prior to the actual deployments.

In the experiments so far, both protocols operated with the same round length, $T =$ 10 minutes. In this configuration, the Coupled protocol bootstrapped the hierarchy faster (cf. Fig. 5c), but consumed more energy than our Decoupled protocol (cf. Fig. 6). Therefore, a systematic comparison requires varying the round length, T, as in Fig. 7.

Our Decoupled protocol consistently outperforms the Coupled one, thereby getting closer to the ideal protocol. For example, when the round length for each of the protocols is configured such that both protocols consume the same amount of energy per hour, our protocol bootstraps the hierarchy $\sim$2.6-3.1 times faster. When the two protocols are configured to bootstrap the hierarchy with the same speed, in turn, our Decoupled protocol consumes only $\sim$0.49-0.67 of the energy consumed by the Coupled

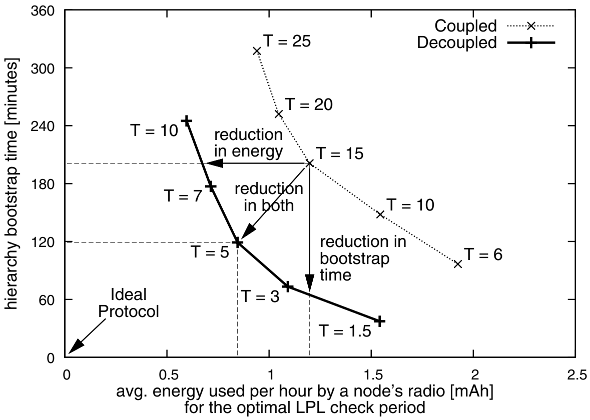


Fig. 7. Energy consumption versus hierarchy bootstrap latency. *Each point represents the average over 10 runs. The network consists of* 1024 *densely deployed IEEE 802.15.4 nodes. Again, the results for different node densities and radios are consistent. The round lengths, T, are in minutes.*

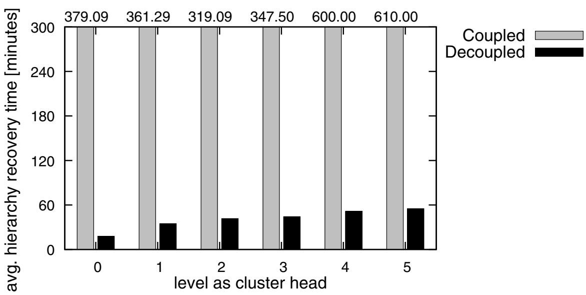


Fig. 8. Latency of recovery after a failure of a cluster head at a given level. *The network consists of* 1024 *sparsely deployed nodes. The round length, T, is* 3 *minutes for our Decoupled protocol and* 15 *minutes for the Coupled one as this results in a similar average energy consumption.*

protocol. Moreover, it is possible to configure our protocol to outperform the Coupled one in both the metrics. These results constitute a major performance improvement as 50% reduction in energy consumption typically doubles the network lifetime.

After the above experiments, to compare the performance of failure recovery, we conducted the following micro-benchmarks. We set the round length for both protocols such that they consume roughly the same amount of energy per hour in the stable state. After the hierarchy has converged, we killed a single node and measured the time to recover the hierarchy depending on the node's level as cluster head. By recovery we mean a state in which neither the label nor the routing table of any alive node contains the identifier of the failed node. Reaching this state guarantees that all higher-level services (e.g., routing, aggregation, querying) will operate correctly. Afterward, we reincarnated the dead node and let it fully rejoin the system. We then repeated the above steps for all other nodes in the network. The results of the experiment are presented in Fig. 8.

Table 2. Comparison of the TOSSIM experiments from Fig. 5 with the testbed experiments. *In general, the results match. The small differences in protocol performance stem mainly from the differences in the deployment parameters, like the network size, density, and topology.*

Experimental Setting /	**TOSSIM**		**Testbed**	
Metric Name	**Coupled**	**Decoupled**	**Coupled**	**Decoupled**
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 Node per Hour	135.75	164.73	143.78	149.30
Hierarchy Bootstrap Time [minutes]	108.02	245.02	135 ± 5	235 ± 5

These results again demonstrate higher efficiency of our protocol. Failure recovery involves two mechanisms: detection and repair. In both protocols, cluster head failure detection requires the same number of rounds, depending on the level of the cluster head. Because our Decoupled protocol performs better than the Coupled protocol when operating with shorter rounds (cf. Fig. 7), it detects cluster head failures faster. Moreover, since the repair is done by the same mechanisms as the hierarchy construction (Sect. 3.5) and since our protocol constructs the hierarchy more efficiently, it is also more efficient when repairing the hierarchy. Consequently, our protocol outperforms the Coupled one also in failure recovery. In the conducted experiment, for instance, our protocol recovered from a level-5 cluster head failure ~ 11.12 times faster (not visible in the plot), while using the same amounts of energy as the Coupled protocol.

Finally, to verify the TOSSIM results we ran the protocols on our 55-node testbed [21]. Table 2 confirms that the testbed results match the TOSSIM results. Moreover, it indicates that our solution can smoothly operate in the real world.

Altogether, the results demonstrate that by decoupling its operation from the hierarchy topology, our protocol can propagate hierarchy information more efficiently, through less frequent albeit much bigger messages. This allows for shortening the protocol round, and thus, for reducing the bootstrap and recovery latency, but without increasing the energy consumption. While the absolute performance gains may differ in different environments (i.e., hardware platforms, MAC layers, deployment settings), we believe that our protocol has potential to improve the efficiency of large-scale hierarchical cluster-based systems built using wireless low-power devices.

5 Conclusions

We argued that, by proverbially "doing more with less," the efficiency of state-of-the-art multi-hop cluster hierarchy maintenance protocols for large WSNs can be significantly improved. To illustrate our claim, we proposed a novel protocol that maintains the same target hierarchy properties with fewer albeit bigger messages, thereby minimizing energy overhead on the exchanged hierarchy information. These reductions are achieved by having redesigned hierarchy maintenance to use a combination of local updates and periodic local gossiping, which decouples protocol operation from the topology of the hierarchy. As we showed, the reductions in energy consumption and hierarchy bootstrap and recovery latency due to such decoupling can be substantial.

References

1. Kumar, S., Alaettinoglu, C., Estrin, D.: SCalable Object-tracking through Unattended Techniques (SCOUT). In: Proc. IEEE ICNP 2000, Osaka, Japan, pp. 253–262 (2000)
2. Subramanian, L., Katz, R.H.: An architecture for building self-configurable systems. In: Proc. ACM MobiHoc 2000, Boston, MA, USA, pp. 63–73 (2000)
3. Bandyopadhyay, S., Coyle, E.J.: An energy efficient hierarchical clustering algorithm for wireless sensor networks. In: Proc. IEEE INFOCOM 2003, San Francisco, CA, USA (2003)
4. Du, S., Khan, A., PalChaudhuri, S., Post, A., Saha, A.K., Druschel, P., Johnson, D.B., Riedi, R.: Self-organizing hierarchical routing for scalable ad hoc networking. Technical Report TR04-433, Rice University, Houston, TX, USA (2004)
5. Li, X., Kim, Y.J., Govindan, R., Hong, W.: Multi-dimensional range queries in sensor networks. In: Proc. ACM SenSys 2003, Los Angeles, CA, USA, pp. 63–75 (2003)
6. Akyildiz, I.F., Kasimoglu, I.H.: Wireless sensor and actor networks: Research challenges. Ad Hoc Networks 2(4), 351–367 (2004)
7. Iwanicki, K., van Steen, M.: Towards a versatile problem diagnosis infrastructure for large wireless sensor networks. In: Proc. OTM PerSys 2007, Vilamoura, Portugal, pp. 845–855 (2007)
8. Iwanicki, K., van Steen, M.: The PL-Gossip algorithm. Technical Report IR-CS-034, Vrije Universiteit, Amsterdam, the Netherlands (2007)
9. Hagouel, J.: Issues in Routing for Large and Dynamic Networks. PhD thesis, Columbia University (1983)
10. Han, J., Kamber, M.: Data Mining: Concepts and Techniques. Morgan Kaufmann, San Francisco (2001)
11. Younis, O., Fahmy, S.: Distributed clustering in ad-hoc sensor networks: A hybrid, energy-efficient approach. In: Proc. IEEE INFOCOM 2004, Hong Kong, China, pp. 640–651 (2004)
12. Jia, L., Rajaraman, R., Suel, T.: An efficient distributed algorithm for constructing small dominating sets. Distributed Computing 15(4), 193–205 (2002)
13. Amis, A.D., Prakash, R., Vuong, T.H.P., Huynh, D.T.: Max-min d-cluster formation in wireless ad hoc networks. In: Proc. IEEE INFOCOM 2000, Tel-Aviv, Israel, pp. 32–41 (2000)
14. Heinzelman, W., Chandrakasan, A., Balakrishnan, H.: Energy-efficient communication protocols for wireless microsensor networks. In: Proc. 33rd HICSS, Maui, HI, USA (2000)
15. Manjeshwar, A., Agrawal, D.P.: TEEN: A routing protocol for enhanced efficiency in wireless sensor networks. In: Proc. IEEE IPDPS 2001 Workshops, San Francisco, CA (2001)
16. Ye, M., Li, C., Chen, G., Wu, J.: EECS: An energy efficient clustering scheme in wireless sensor networks. In: Proc. IEEE IPCCC 2005, Phoenix, AZ, USA, pp. 535–540 (2005)
17. Thaler, D., Ravishankar, C.V.: Distributed top-down hierarchy construction. In: Proc. IEEE INFOCOM 1998, San Francisco, CA, USA, pp. 693–701 (1998)
18. Polastre, J., Hill, J., Culler, D.: Versatile low power media access for wireless sensor networks. In: Proc. ACM SenSys 2004, Baltimore, MD, USA, pp. 95–107 (2004)
19. Klues, K., Handziski, V., Lu, C., Wolisz, A., Culler, D., Gay, D., Levis, P.: Integrating concurrency control and energy management in device drivers. In: Proc. ACM SOSP 2007, Stevenson, WA, USA, pp. 251–264 (2007)
20. Woo, A., Tong, T., Culler, D.: Taming the underlying challenges of reliable multihop routing in sensor networks. In: Proc. ACM SenSys 2003, Los Angeles, CA, USA, pp. 14–27 (2003)
21. Iwanicki, K., Gaba, A., van Steen, M.: KonTest: A wireless sensor network testbed at Vrije Universiteit Amsterdam. Technical Report IR-CS-045, Vrije Universiteit, Amsterdam, the Netherlands (2008)