Wireless Networks manuscript No.
(will be inserted by the editor)

Distributed Algorithms for Lifetime Maximization in Sensor Networks via Min-Max Spanning Subgraphs

Harri Haanpää · André Schumacher · Pekka Orponen

Received: date / Accepted: date

Abstract We consider the problem of static transmission-power assignment for lifetime maximization of a wireless sensor network with stationary nodes operating in a data-gathering scenario. Using a graph-theoretic approach, we propose two distributed algorithms, MLS and BSPAN, that construct spanning trees with minimum maximum (min-max) edge cost. MLS is based on computation of minmax-cost paths from a reference node, while BSPAN performs a binary search over the range of power levels and exploits the wireless broadcast advantage. We also present a simple distributed method for pruning a graph to its Relative Neighborhood Graph, which reduces the worst-case message complexity of MLS under natural assumptions on the path-loss. In our network simulations both MLS and BSPAN significantly outperform the recently proposed Distributed Min-Max Tree algorithm in terms of number of messages required.

Keywords Lifetime Maximization · Minimum Spanning Tree · Min-Max Spanning Tree · Relative Neighborhood Graph · Distributed Algorithm

Preliminary work has been reported in the Third International Conference on Mobile Ad-hoc and Sensor Networks (MSN), Beijing, China, December 12 – 14, 2007 and the Fifth European Conference on Wireless Sensor Networks (EWSN), Bologna, Italy, January 30 – February 1, 2008.

H. Haanpää
Department of Information and Computer Science
Helsinki University of Technology TKK
E-mail: Harri.Haanpaa@tkk.fi

A. Schumacher
E-mail: Andre.Schumacher@tkk.fi

P. Orponen
E-mail: Pekka.Orponen@tkk.fi

1 Introduction

Consider a group of sensors newly deployed in an environment. In many applications, it is desirable to have the network to self-configure, i.e., to have the nodes after wakeup contact their neighbors in order to decide where to forward the collected data, at what intervals, transmission power levels, etc. One important goal of this self-configuration process is to initialize data-gathering and transmission protocols so that the operational time of the network, for given initial battery levels, is maximized [1,5].

We address this lifetime maximization problem in the setting where it is the task for a network of stationary nodes to provide a roughly uniform, low-intensity stream of data to a designated sink node. Possible application scenarios include monitoring some environmental parameters (temperature, humidity, chemical concentrations) in a given region or, say, a forest-fire alarm network, where most of the data traffic consists of regular “status ok” messages.

More specifically, we consider the problem of determining transmission power levels for the nodes so that, under the assumption of uniform traffic load per node, all the nodes maintain connectivity to the sink for a maximum amount of time. In this paper, we only consider the case of static power assignments, i.e., we assume that once the transmission power levels have been set, they stay the same throughout the operating life of the network. We also assume that transmission costs have a dominant effect on the lifetime on the nodes, which may operate a sleep-scheduling scheme [4].

Under these assumptions of stationary nodes, uniform traffic load and static power assignments, the goal of maximizing the lifetime of a network is in fact equivalent to finding the lowest possible transmission power levels for the nodes that suffice to make all of the network connected to the sink. In graph-theoretic terms, the problem becomes to find a spanning subgraph of the *transmission graph* with the minimum possible maximum edge cost, where the transmission graph contains all nodes of the network and edges between nodes within maximum transmission range. Although the problem can in principle be solved by a distributed algorithm finding Minimum Spanning Tree (MST), the construction of an MST is more involved than the distributed search for a spanning subgraph with minimum maximum (minmax) edge cost. We present two simple and efficient distributed algorithms for the lifetime maximization problem that are able to exploit different properties of the problem and therefore also typically differ in the number of control messages and running time required.

Our Maximum Lifetime Spanner (MLS) algorithm is based on an approach similar to the distributed MST algorithm of Gupta and Srimani [12], viz., the construction of paths with minmax edge cost by breadth-first search. However, as we do not consider the construction of an MST, we obtain a significantly simpler algorithm. The solution of the problem via computation of minmax-cost paths is advantageous, as the process can be efficiently distributed and the set of candidate edges can be reduced drastically prior to the execution of the algorithm. For this purpose, we utilize an algebraic formulation of relative neighborhood graphs (RNG) [22] and present a distributed method for obtaining the RNG of a given transmission graph.

Our Binary Search for Min-Max Power Spanner (BSPAN) algorithm is a distributed algorithm based on a "binary search over transmission power levels" idea. Lloyd et al. [15] proposed a simple and efficient binary search based solution, assuming that complete information on network connectivity and edge costs is centrally available. Our proposed BSPAN algorithm is particularly suitable for implementation in wireless networks because it utilizes broadcast messages and therefore exploits the wireless broadcast advantage. As the number of available transmission power levels is expected to be rather small in practice, the search terminates in a few iterations, which generally yields a low number of control messages required.

We have implemented both algorithms down to the level of a protocol agent in the `ns2` [17] simulator, and they show quite competitive performance in comparison with the Distributed Min-Max Tree (DMMT) algorithm that was recently proposed by Guo, Yang, and Leung [11].

The rest of the paper is organized as follows. The following section overviews some of the related work on lifetime maximization. Section 3 gives a precise formulation of the version of the problem we consider. In Section 4, we propose a generic method for the initialization of nodes upon wakeup that does not assume prior neighborhood information. Section 5 describes the MLS algorithm and presents a distributed method for RNG computation. Section 6 presents the BSPAN algorithm, and in Section 7 we evaluate MLS and BSPAN in terms of the number of required control messages, and compare them with the performance of DMMT. For our experimental comparison we use the `ns2` network simulator. Section 8 concludes the paper.

2 Related work

The problem of minimizing the maximum transmission power required to establish connectivity has been considered previously in the literature. One of the earliest papers on the topic is the work of Ramanathan and Rosales-Hain [19], which addresses the problem in the setting of maximizing the lifetime of a single-session broadcast. Ramanathan and Rosales-Hain propose a centralized algorithm for finding the minmax transmission power level that maintains network connectivity, as well as two simple distributed heuristics that aim at achieving the same. Their distributed heuristics, however, are suboptimal and do not necessarily guarantee connectivity in all cases.

Kang and Poovendran [13] discuss several problems related to dynamic lifetime maximization, such as the issue of non-uniform energy levels. They also emphasize the importance of considering the minmax energy metric rather than the more often addressed minimum total energy metric for the purpose of maximizing network lifetime. For a distributed implementation, Kang and Poovendran rely on distributed methods for constructing minimum spanning trees, such as the algorithm of Gallager, Humblet and Spira [10]. These techniques are, however, rather involved, and we complement this work by suggesting an efficient and much simpler method for computing the minmax edge cost required for connectivity. For a discussion of the two different objectives, minimizing total transmission power and minimizing maximum transmission power, see, e.g., [13, 15].

Narayanaswamy et al. [18] propose a protocol for power control in wireless ad-hoc networks with discrete power levels. Their protocol also establishes a spanning subgraph of the transmission graph with minmax cost. However, the solution proposed in [18] relies on a proactive routing protocol and requires significant control overhead that renders it unsuitable for sensor networks.

The problem of minimizing the *total*, as opposed to minmax, network transmission power required for connectivity has been studied extensively (cf., e.g., [15] and the references therein). Rodoplu and Meng [20] present a distributed algorithm for this problem that is based on the concept of *relay regions*: each node is aware of its own geographic location and the location of its neighbors. Based on a path-loss model, nodes can locally determine which neighbor they should forward the message to in order to minimize the total energy consumption. The algorithm proposed in [20] is optimal but requires extensive assumptions, such as the availability of location information and a specific path-loss model.

Wattenhofer et al. [23] propose a distributed algorithm for the same problem. Their algorithm, which relies on a geometric cone-based forwarding scheme, requires that nodes can measure exactly the direction of incoming radio transmissions (angle of arrival). It also makes further assumptions on geometric properties of the underlying graph model.

In a recent work, Guo, Yang, and Leung [11] proposed a distributed algorithm DMMT (Distributed Min-Max Tree) for the construction of multicast trees with minimum maximum transmission cost, following Prim's algorithm for constructing minimum spanning trees. Since their technique can easily be adapted also for the purpose of sensor network lifetime maximization, and seems to be the proposal in the literature closest to our approach, we conducted an experimental comparison of the runtime behavior of the algorithms DMMT and our proposed MLS and BSPAN algorithms.

3 The lifetime maximization problem

We consider a wireless sensor network composed of stationary nodes with distinct identifiers, operating in a data-gathering scenario. Each node is able to vary its transmission power, using a possibly large set of power levels. The energy budget that is consumed during the operation of the network is finite and initially the same for all nodes. We consider a scenario where the energy consumed by wireless transmission dominates over energy consumed by computation or sensing. We further assume that traffic is generated uniformly over the nodes and that data-aggregation techniques can be applied. When traffic is generated uniformly and aggregated on its path towards the sink, upstream nodes forward the same amount of traffic as downstream nodes, thus yielding a close to uniform load within the network.

In order to maintain connectivity to a neighbor u, a node v has to spend some energy that depends on v's transmission power level. Each node has the same maximum transmission power $p^{\max}$ that must not be exceeded. We assume that the transmission costs are symmetric, so that if v can reach u at a certain power, then u can also reach v at the same power; this is the case for example if the costs represent signal attenuation resulting from a deterministic path-loss model that only depends on the pairwise distance of nodes. We consider the notion of lifetime that regards all nodes as equally important, so that the objective is to maximize the time span after which the first node runs out of energy [6].

The transmission structure of the network is modeled as a directed graph $G = (V,E)$ with an associated edge cost function $\delta : E \mapsto [0, p^{\max}]$ that gives the minimum power necessary to use the link: v can reach u if the transmission power $\tau(v)$ satisfies $\tau(v) \geq \delta(v,u)$. As we assume transmission costs to be symmetric, $\delta(u,v) = \delta(v,u)$. The vertices in V represent the nodes of the network, and E contains an edge for each link that is usable at maximum power. A transmission power assignment $\tau : V \mapsto [0, p^{\max}]$ induces a graph $G(\tau) = (V, E(\tau))$ whose edges represent the radio links that are supported by the given assignment τ, so that $E(\tau) = \{(v,u) \mid (v,u) \in E \text{ and } \tau(v) \geq \delta(v,u)\}$. For simplicity, we assume that the transmission graph $G(\tau^{\max})$ with $\tau^{\max}(v) = p^{\max}$ for all v is a connected graph.

We consider the problem of finding a static transmission power assignment τ, such that the lifetime of the network is maximized while the network remains connected. Any power assignment τ that connects the network induces a spanning subgraph with some maximum edge cost $\alpha = \max_{(v,u)\in E(\tau)} \delta(v,u)$; we aim to find a power assignment that minimizes α. Although this condition generally does not uniquely determine τ, choosing $\tau(v) = \alpha$ for all nodes v does not decrease the lifetime. The power assignment τ is considered to be fixed after it has been once determined during the initial network setup. Note that this problem is considerably different from the case of computing a dynamic assignment of power levels, which is a computationally more complex problem [9].

Definition 1 Given a transmission graph $G(V,E)$, and an edge cost function $\delta : E \mapsto [0, p^{\max}]$, a graph $G' = (V,E')$ with $E' \subseteq E$ is an *α-spanner* if G' is connected and $\delta(v,u) \leq \alpha$ for each edge $(v,u) \in E'$.

In other words, an α-spanner is a connected spanning subgraph for the transmission graph where no edge has cost greater than α. For any network a $p^{\max}$-spanner exists exactly when the network can be connected by the nodes sending at full power.

For a given transmission graph G, an α-spanner G' is *optimal* if no α'-spanners exist for $\alpha' < \alpha$. An optimal α-spanner has a maximum edge cost α and there are no spanners with only edges of cost strictly less than α; thus, we also call such a spanner a *minmax spanner*. Network lifetime can now be maximized by determining a minmax spanner $G^* = (V,E^*)$ and choosing the power assignment $\tau(v) = \max_{(v,u)\in E^*} \delta(v,u)$.

4 Algorithm initialization and termination

In Sections 5 and 6 we describe two algorithms for computing minmax spanners. Both algorithms require prior knowledge of the network topology, such as network size and neighbor lists. Also, for both algorithms, after termination the nodes in the network should be notified so that they can set their transmission power level accordingly.

In Section 4.1 we present a method of collecting the necessary neighborhood information, and in Section 4.2 we describe a method of notifying the nodes in the network of the termination of the algorithm. Both methods employ standard distributed-algorithm techniques upon which we expand by integrating the computation of the edge-cost function δ.

Algorithm 1: Setup stage

```
node $v$ with variables beacon_count, beacon_delay,
expecting_reply, father, neighbor_list, node_count, timer;

at start
    node_count $\leftarrow 0$; expecting_reply $\leftarrow \emptyset$;
    enter state IDLE;

in state IDLE                                  // wait for incoming beacons
    if beacon$(u, f')$ is received with strength $s^{\text{recv}}$ then
        father $\leftarrow u$; $\delta \leftarrow (s^{\text{thresh}}/s^{\text{recv}})p^{\max}$;    // estimate cost
        neighbor_list $\leftarrow$ neighbor_list $\cup (u, \delta)$;
        broadcast beacon$(v$, father$)$ at power $p^{\max}$;
        timer $\leftarrow$ beacon event after rand(0,beacon_delay);
        enter state BEACON;
    end

in state BEACON                    // send beacons with random delay
    if beacon$(u, f')$ is received with strength $s^{\text{recv}}$ then
        $\delta \leftarrow (s^{\text{thresh}}/s^{\text{recv}})p^{\max}$;                    // estimate cost
        neighbor_list $\leftarrow$ neighbor_list $\cup (u, \delta)$;
        if $f' = v$ then
            expecting_reply $\leftarrow$ expecting_reply $\cup \{u\}$;
    end
    if reply$(u$, count$)$ is received with strength $s^{\text{recv}}$ then
        expecting_reply $\leftarrow$ expecting_reply $\setminus \{u\}$;
        if expecting_reply $= \emptyset$ and beacon_count $=$
        beacon_repetitions then
            unicast reply$(v$, node_count $+$ count $+ 1)$ at power
              $p^{\max}$ to father;
            enter state SETUP_FINISHED;      // end of stage
        else
            node_count $\leftarrow$ node_count $+$ count;
        end
    end
    if timer triggers beacon event then
        broadcast beacon$(v$, father$)$ at power $p^{\max}$;
        if beacon_count $<$ beacon_repetitions then
            beacon_count $\leftarrow$ beacon_count $+ 1$;
            timer $\leftarrow$ beacon event after rand(0,beacon_delay);
        else if expecting_reply $= \emptyset$ then
            unicast reply$(v$, node_count $+ 1)$ at power $p^{\max}$ to
              father;
            enter state SETUP_FINISHED;      // end of stage
        end
    end
```

4.1 Setup stage

The setup stage as described in Algorithm 1 first finds a spanning tree of the transmission graph by a process of beaconing at maximum transmission power $p^{\max}$. Each node, once it has joined the spanning tree under construction, starts sending a sequence of beacon messages using random delays between consecutive messages. These beacons enable nodes to discover their neighbors, estimate the cost of their incident edges in the transmission graph and determine whether they are leaf nodes in the spanning tree. The repeated transmission of beacons is necessary to reduce the probability of undiscovered edges due to packet collisions.

When the beaconing sequence has terminated, a reply message is transmitted along the attained spanning tree edges to the reference node, starting at the leaf nodes. This reply message contains a count of the number of children of each node, so that the reference node eventually obtains a count of the total number of nodes in the network. More specifically, in a beacon message beacon(v, f) the parameter v denotes the identity of the beaconing node and f is its father in the spanning tree being constructed; in a reply message reply(v, count) the parameter v is again the identity of the sender, and count represents the number of nodes in the subtree rooted at v. When the reference node has received reply messages from all its children, it knows that the setup stage has terminated.

The setup stage is initiated as if the reference node had received a beacon message. Upon receiving a beacon message from a node u for the first time, each node v sends the message beacon(v, u) and schedules a number of retransmissions using random delays. After transmitting the beacon for the first time, v starts listening for messages from neighboring nodes and records their presence in a neighbor list together with a flag indicating whether the neighbor is a child node in the spanning tree. Note that the neighbor u is a child of v if u includes the information that it previously received the beacon from v in the message.

For each received beacon, v also estimates a lower bound on the transmission power that is required to reach the neighboring node. More specifically, we consider a message that a node v receives from node u, received with signal strength s^{recv}, arrived successfully if $s^{\text{recv}} \geq s^{\text{thresh}}$, where s^{thresh} is the threshold signal strength required for a successful transmission (disregarding interference). In our simulations, the signal strength s^{recv} is computed by the propagation model under consideration. Assuming that the received signal strength depends linearly on the sending power, $s^{\text{recv}} = X_{u,v}\, p^{\text{send}}$, and the receiver v knows the sending power used, v can estimate the attenuation coefficient $X_{u,v} = s^{\text{recv}}/p^{\text{send}}$. Assuming that $X_{v,u} = X_{u,v}$, node v can estimate the minimum transmission power $p^{\min}$ it needs to use to transmit to u by solving $s^{\text{thresh}} = X_{v,u}\, p^{\min}$. Combining these, we have

$$p^{\min} = \frac{s^{\text{thresh}}\, p^{\text{send}}}{s^{\text{recv}}},$$

where p^{send} is the power that was used by u for sending (in Algorithm 1 we use $p^{\text{send}} = p^{\max}$). If the assumption does not hold or if the measured received signal strength shows random variations, then beacon messages with varying transmission power can be used for the same purpose, similar to the techniques proposed in [14]. Recall that the edge costs $\delta(v,u)$ represent the minimum power required for v to send to u, so by estimating $p^{\min}$ node v can estimate the edge cost as $\delta(v,u) = p^{\min}$, or if the maximum power was used for sending, $p^{\text{send}} = p^{\max}$ and $\delta(v,u) = (s^{\text{thresh}}/s^{\text{recv}})p^{\max}$.

When v has sent a certain number of beacon messages, it decides that the setup stage has locally terminated. In the case that v discovers itself to be a leaf node of the constructed spanning tree, it sends a reply message to its father reporting a node count of one. If v is not a leaf node, it waits until it receives replies from all its children before it sends a reply to its father that contains the sum of its child counts incremented by one, indicating the termination within the subtree rooted at v. When the reference node has received replies from all its child nodes the setup stage has terminated.

For measuring the strength of arriving radio signals, one can utilize for example Received Signal Strength Indication (RSSI) [21] or, alternatively, methods similar to the ones proposed in [14]. To obtain the correct neighborhood information for all the nodes, in most cases the number of retransmissions for the beacon messages can be fairly small. Assuming, here and in future analysis of message complexity, that the average number of retransmissions of a packet is bounded by some constant as the network grows, the message complexity of the beaconing stage is $O(N)$, where N is the number of nodes in the network.

4.2 Notification stage

The notification stage consists of a simple network-wide broadcast by which the reference node informs all other nodes about the termination of the algorithm for computing a minmax spanner. As each node transmits at most one message during the notification stage (subject to retransmissions due to collisions), the number of required messages is at most N. The implementation of this stage depends on the algorithm for computing the minmax spanner, as described later in this paper.

From the adjacency list and by listening to notification messages all nodes can infer locally their power level assignment. More specifically, each node sets its transmission power $\tau(v)$ to the value that suffices to maintain the most expensive edge to its neighbors in the optimal α-spanner, i.e., $\tau(v) = \max_{(v,u)\in E'} \delta(v,u)$, where E' is the set of edges in the minmax spanner.

5 A distributed algorithm for minmax spanners

In the following, we describe a distributed algorithm for obtaining a minmax spanner that relies on neighborhood information gathered during the setup stage described in the previous section. Assuming $G(\tau^{\max})$ is connected, the Maximum Lifetime Spanner (MLS) algorithm computes a spanning tree with minmax edge cost by establishing paths from the reference node to any other node in the network. In Section 5.3, we present a modified beaconing method that computes a subgraph of $G(\tau^{\max})$ by pruning edges non-relevant to the search of a minmax spanner. By running the algorithm on the subgraph obtained, we are able to reduce the message complexity significantly.

Algorithm 2: Distributed algorithm for finding a minmax spanner [MLS]

```
node $v$ with local variables $\alpha, \alpha[\cdot]$, father, status$[\cdot]$

at start
    $\alpha \leftarrow \infty$; father $\leftarrow$ undefined;
    for $u \in N(v)$ do
        $\alpha[u] \leftarrow \infty$; status $[u] \leftarrow$ ready;
    end
    enter state IDLE;

in state IDLE or SEARCH
    if $(\alpha')$ with $\alpha' < \alpha$ is received from some node $u$ then
        if father is defined then send NAK$(\alpha)$ to father;
        father $\leftarrow u$;
        for $w$ in $N(v) \setminus \{u\}$ do
            if $\max(\alpha', \delta(v,w)) < \alpha[w]$ then
                send $(\max(\alpha', \delta(v,w)))$ to $w$;
                $\alpha[w] \leftarrow \max(\alpha', \delta(v,w))$;
                status $[w] \leftarrow$ wait;
            end
        end
        enter state SEARCH
    end
    if $(\alpha')$ with $\alpha' \geq \alpha$ is received from some node $u$ then
        send NAK$(\alpha')$ to $u$;
    end

in state SEARCH        // wait for incoming acknowledgements
    if status $[w]$=ready for all $w \in N(v) \setminus \{$father$\}$ then
        send ACK$(\alpha)$ to father;
        enter state IDLE
    end
    if ACK$(\alpha')$ or NAK$(\alpha')$ is received from $u$ and $\alpha[u] = \alpha'$
    then
        status $[u] \leftarrow$ ready;
    end
```

Our Algorithm 2 for finding a minmax spanner is based on distributed breadth-first search similar to the asynchronous Bellman-Ford algorithm [16, Sec. 15.4]. However, we use the properties of the minmax edge cost function to reduce the complexity of the search. First, a given reference node sends to each of its neighbors a message that contains the cost of the connecting edge. Upon first receiving the request, each node makes note of the node from which the message was received and retransmits the request to its neighbors, updating the maximum edge cost α indicated in the request accordingly. Each node also remembers the best α sent to each neighbor. If a node that has already received and forwarded a request receives a request that indicates a better route from the reference node, it retransmits the latter request to its neighbors if this leads to obtaining a route with a lower α, to those neighbors. In a typical data-gathering scenario, the natural choice for the reference node is the data sink.

Moreover, the nodes collect acknowledgements from their neighbors. When a node receives the request, it forwards it to its neighbors, and waits for each neighbor to either accept (ACK) or reject (NAK) it. When acknowledgements have been received from each neighbor, the node sends an ACK to the node from which it received the request. A NAK is sent if the node receiving the request already knows of a better path, or if a node learns of a better path while waiting for the acknowledgements from its neighbors. In this way, an ACK response means that the responding node has accepted the other node as its father in the tree being constructed, while a NAK signifies refusal. It can happen that a node will first respond with an ACK but later send a NAK; however, when the reference node has received acknowledgements from its neighbors, the algorithm has finished. A sample run of Algorithm 2 is given in Fig. 1.

In Algorithm 2, α is the current estimate of the minmax cost of a path from the reference node to each node v; and father is the node from which v has received the last accepted message. Initially, father is undefined and $\alpha = \infty$ for each v. The minmax spanner is defined by the father variables of each node after the algorithm has terminated.

To justify the algorithm, we firstly observe that it always terminates. Let $\Delta = \{\delta(u,v) \mid (u,v) \in E\}$ be the set of distinct edge costs in the graph. Obviously, no node can learn of a new route with better α more than $|\Delta|$ times. Secondly, at the end each node has a correct α_v: if from some node v there would exist a path with maximum edge cost $\alpha_0 < \alpha_v$ to the reference node, then on the path there is some edge of cost at most α_0 where exactly one endpoint would have a maximum edge cost estimate higher than α_0. This cannot happen, since the endpoint with cost at most α_0 should send a message along that edge. Thirdly, it cannot happen that a node would remain in the search state, since its neighbors will respond to the queries either by an immediate NAK if the cost was too large, a delayed ACK once the neighbor has received responses from its children, or a delayed NAK in case the neighbor later learns of a path with a lower maximum edge cost.

To consider the communication complexity of the algorithm, observe that the number of distinct edge costs is bounded by $|\Delta| \leq |E|$. In this regard the minmax edge cost spanner problem is different from finding minimum cost routes, where the number of routes with different total cost between two nodes can be exponential in the number of nodes [16, Sec. 15.4]. When a node learns of a better α, it will send a message to its neighbors, who will eventually answer with an ACK or a NAK. Since the requests sent by a node to its neighbor are in order of decreasing α, at most $|\Delta|$ requests are sent along each of the $|E|$ directed edges, and there are a total of $O(|\Delta|\,|E|)$ messages of constant size (we consider node ids, node counts and power levels to be of constant size).

5.1 Notification stage

To notify the remaining nodes about the termination of the algorithm, the reference node initiates a network-wide broadcast using the edges of the computed spanning tree. Each node v receiving this broadcast message can then decrease its transmission power $\tau(v)$ to the minimum power required to reach its father and the neighboring nodes that have chosen v to be their father.

5.2 Relative neighborhood graphs

Algorithm 2 requires nodes to exchange messages with all neighbors. In a dense sensor network where the number of nodes within transmission range may be large, it is beneficial to limit the number of nodes that need to be contacted, while maintaining network connectivity at the same minmax transmission cost. For this purpose, we use *relative neighborhood graphs* [22]. Relative neighborhood graphs and related structures have been used for topology control [2, 3], mostly in a geometric context, where nodes are placed in a plane and $\delta(v,u)$ depends only on the Euclidean distance between u and v. However, we only assume that path loss is symmetric, i.e., $\delta(v,u) = \delta(u,v)$. We will find, though, that when the nodes are placed in the Euclidean plane, our algorithm runs much faster.

Definition 2 Given a graph $G = (V,E)$ and an edge cost function δ, the relative neighborhood graph of G is the graph with vertex set V and edge set $\{(v,u) \mid (v,u) \in E, \nexists w \text{ s.t. } (v,w),(w,u) \in E, \delta(v,w) < \delta(v,u), \delta(w,u) < \delta(v,u)\}$.

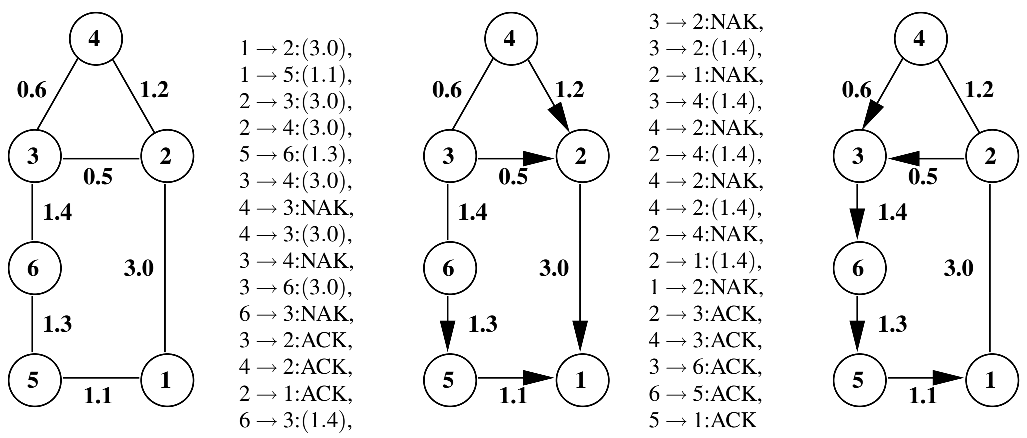
Thus, the relative neighborhood graph is obtained from the original graph by deleting each edge (v,u) if there is a path v–w–u of two cheaper edges. Such a generalization of the concept of RNG has been already successfully applied to other problems, such as searching and broadcasting in peer-to-peer networks [8].

Claim For any α, the RNG of G contains an α-spanner if G does.

Proof Consider an α-spanner in the original graph. Order the k edges removed from the original graph in constructing the relative neighborhood graph in decreasing order of cost as $e_1, e_2, \ldots, e_k$. Let E_0 denote the edge set of the original graph, and let $E_i = E_{i-1} \setminus \{e_i\}$ for $0 < i \leq k$. Now for $0 < i \leq k$, since E_{i-1} admits an α-spanner and the endpoints of e_i are connected by a path of two cheaper edges, E_i also admits an α-spanner, so the RNG (V, E_k) admits an α-spanner.

5.3 Distributed algorithms for RNGs

In this section, we describe a modification to the beaconing method proposed in Section 4.1 to obtain a distributed



$1 \rightarrow 2$:(3.0),
$1 \rightarrow 5$:(1.1),
$2 \rightarrow 3$:(3.0),
$2 \rightarrow 4$:(3.0),
$5 \rightarrow 6$:(1.3),
$3 \rightarrow 4$:(3.0),
$4 \rightarrow 3$:NAK,
$4 \rightarrow 3$:(3.0),
$3 \rightarrow 4$:NAK,
$3 \rightarrow 6$:(3.0),
$6 \rightarrow 3$:NAK,
$3 \rightarrow 2$:ACK,
$4 \rightarrow 2$:ACK,
$2 \rightarrow 1$:ACK,
$6 \rightarrow 3$:(1.4),

$3 \rightarrow 2$:NAK,
$3 \rightarrow 2$:(1.4),
$2 \rightarrow 1$:NAK,
$3 \rightarrow 4$:(1.4),
$4 \rightarrow 2$:NAK,
$2 \rightarrow 4$:(1.4),
$4 \rightarrow 2$:NAK,
$4 \rightarrow 2$:(1.4),
$2 \rightarrow 4$:NAK,
$2 \rightarrow 1$:(1.4),
$1 \rightarrow 2$:NAK,
$2 \rightarrow 3$:ACK,
$4 \rightarrow 3$:ACK,
$3 \rightarrow 6$:ACK,
$6 \rightarrow 5$:ACK,
$5 \rightarrow 1$:ACK

Fig. 1 Sample execution of Algorithm 2 from reference node 1; messages listed as *source→destination:message*. The three graphs show the initial state, intermediate state, and final state of the algorithm with messages listed between states in the order of their transmission.

method for constructing RNGs. We again do not assume that a node initially knows about the cost of the edges to its neighbors, but we assume that a node can estimate the strength of arriving radio signals. To construct the RNG, each node again starts beaconing at maximum power after receiving an initial wakeup message originating from the reference node. However, in addition to its distinct node identifier, a node also includes the identifiers of neighboring nodes and the associated δ the node has learned so far in the beacon message.

After having learned about the neighbors of their neighbors, the nodes prune unnecessary edges from the transmission graph. If a node v learns that for some third node w it holds that $\delta(v,w) < \delta(v,u)$ and $\delta(w,u) < \delta(v,u)$, then v can determine that the edge (v,u) is not in the RNG, as per Definition 2. Thus, the nodes can prune their neighborhood so that only the RNG remains in $O(|V|)$ messages, the size of each of which is proportional to the number of neighbors the beaconing node has. Thus, the total amount of data transmitted during the RNG construction is $O(|E|)$. Note that a natural point for including the pruning of non-RNG edges into Algorithm 1 is the state when the node sends the reply message to its father node during the setup stage.

Pruning the transmission graph down to the RNG before running Algorithm 2 can give very considerable savings in complexity. Beaconing and determining the RNG requires sending $O(N)$ messages with $O(E)$ bits in total, and then determining the minmax spanner requires $O(|\Delta||E'|)$ messages of constant size, where E' is the set of edges in the RNG. With an arbitrary path loss function, the RNG can still contain $O(|V|^2)$ edges. However, when the nodes are in a plane and path loss is an increasing function of distance, the RNG is a subgraph of the Delaunay triangulation of the original graph and contains only $O(|V|)$ edges [22]. In this special case, then, the algorithm would require sending a total of $O(|\Delta||V|)$ messages of constant size.

6 Distributed binary search for minmax spanners

Previously, we described a distributed algorithm for computing a minmax spanner that did not require any assumptions on the edge-cost function δ besides being symmetric. In practice, however, the range of δ most likely corresponds to a small set of possible transmission power levels. Furthermore, it is beneficial to use broadcast messages to further reduce the total number of control messages required. In this section, we propose a distributed algorithm for determining a minmax spanner, given a graph with edge costs and the set of available transmission power levels $P = \{p_1, p_2, \ldots, p_{|P|}\}$, where $p_1 < p_2 < \ldots < p_{|P|} = p^{\max}$, so now the transmission power level assignment is of the type $\tau : V \mapsto P$. The algorithm utilizes broadcast messages to reduce the total number of messages sent during the execution of the algorithm.

As also the MLS algorithm presented in Section 5, the Binary Search for Min-Max Power Spanner (BSPAN) algorithm relies on the availability of a reference node for coordination, which initiates Algorithm 1 and thus obtains a count of the number of nodes in the network. Thereafter, it performs a binary search over the range of transmission power levels to find the minmax power required for connectivity and uses this value to establish a minmax spanner.

At each iteration of the algorithm, the reference node initiates the computation of a rooted tree spanning the nodes that can be reached from the reference node using paths with maximum edge cost at most α. The construction of the tree is achieved by flooding request messages over edges with cost at most α. In the second phase of the iteration, the reference node then checks whether this tree spans all nodes in the network by comparing the number of nodes reached with the total number of nodes in the network. The counting of nodes reached using edges with cost at most α is performed by a convergecast of reply messages back to the reference node. In Algorithm 3, $N(v)$ denotes the set of neighbors of node v in the given transmission graph $G(\tau^{\max}) = G(V,E)$, i.e., $N(v) = \{u \in V | (v,u) \in E\}$.

Algorithm 3: Distributed binary search for a minmax spanner [BSpan]

```
node $v$ with variables $\alpha$, lower, curr, upper, child_candidates,
father, is_reference_node, $N(v)$, node_count, status;

at start
    if is_reference_node then
        lower ← 0; upper ← $|P|$;
        curr ← $\lfloor(\text{lower} + \text{upper})/2\rfloor$; $\alpha \leftarrow p_{\text{curr}}$;
    for $u \in N(v)$ do status[$u$] ← processed;
    enter state RESET;

in state RESET
    father ← none; node_count ← 0;
    if is_reference_node then
        if lower + 1 < upper then
            enter state SEND_REQUEST;
        else enter state SEARCH_FINISHED;
    else enter state IDLE;

in state IDLE                              // wait for incoming requests
    if request($u, \alpha', f'$) with $\alpha' \geq \delta(u,v)$ is received then
        $\alpha \leftarrow \alpha'$; father ← $u$;
        enter state SEND_REQUEST;
    end

in state SEND_REQUEST          // broadcast request to neighbors
    child_candidates ← $\{w \in N(v) \setminus \{\text{father}\} \mid \delta(v,w) \leq \alpha\}$;
    for $w \in$ child_candidates do status[$w$] ← wait;
    if child_candidates $\neq \emptyset$ then
        broadcast request($v, \alpha$, father);
    enter state PROCESSING;

in state PROCESSING         // process requests, wait for replies
    if request($u, \alpha', f'$) is received then
        if $f' = v$ then      // $u$ has acknowledged $v$ as its father
            status[$u$] ← child;
        else                             // u has father $f'$ different from v
            status[$u$] ← processed;
    end
    if reply($u$, nodes) is received then
        status[$u$] ← processed;
        node_count ← node_count + nodes;
    if status[$w$] = processed for all $w \in N(v) \setminus \{\text{father}\}$ then
        if is_reference_node then
            if total_nodes = node_count then upper ← curr;
            else lower ← curr;
            curr ← $\lfloor(\text{lower} + \text{upper})/2\rfloor$; $\alpha \leftarrow p_{\text{curr}}$;
        else                                        // report node count
            unicast reply($v$, node_count + 1) to father;
        enter state RESET;
    end
```

6.1 Request phase

The request messages are of the form (v, α, f) where v is the identity of the sending node, α is the maximum allowable edge cost in this iteration, and f is the father of node v. In the first step, each node v, upon receiving a request from a neighbor u, broadcasts a request message at most once by broadcasting it to all neighboring nodes. We assume that all messages are sent at maximum power, although that assumption is not critical to the algorithm: choosing a power corresponding to α would be possible as well.

Node v decides that sending the message is required under the following conditions. Firstly, v must not have broadcast a request earlier in this iteration. If so, and the cost of the edge (u,v) is less or equal to α, the current edge cost under consideration, u becomes the *father* of v. Note that the edge costs are assumed to be symmetric. Secondly, there must still be adjacent nodes w different from u such that the edge (v,w) has cost less than or equal to α.

6.2 Reply phase

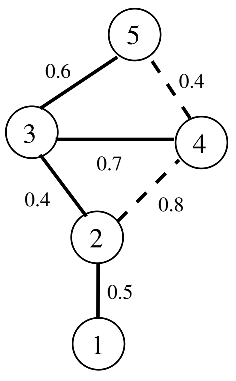
After sending the request (v, α, u), v waits for a request from any w that meets the condition above. In the case that v receives a request (w, α, v') from w, it will mark w as *child* if $v' = v$, and as *processed* otherwise. A neighbor marked *child* corresponds to w being v's child in the tree of the current iteration, and the label *processed* corresponds at this step to w being in the tree already with a different father node v'.

In the case that v has no child nodes, either because there are no adjacent nodes with low enough edge costs or if they all have different father nodes, it can determine that it is a leaf node in the current tree. Subsequently, it originates a reply message that contains its id and a node count of one, which it sends to its father node u. If v has at least one child w, v waits for replies from all its child nodes before sending a reply. After receiving a reply from w, node v marks w as *processed*.

When v receives the last outstanding reply (all neighbors except its father are marked *processed* in v's neighbor table), v updates the last reply to contain the sum of all node counts received from its child nodes incremented by one and then forwards the reply to its father. Thus, the reference node can determine the number of nodes in the network reachable by edges with cost at most the current candidate edge cost α. By comparing this count with the count obtained during the setup stage, the reference node is able to determine whether α is an upper or lower bound of the minmax transmission cost and update α correspondingly. See Algorithm 3 for details and Fig. 2 for a toy example of a single iteration of the algorithm.

6.3 Notification stage

After the search has terminated, the reference node initiates the notification stage and informs all other nodes about the termination and the minimum edge cost necessary to connect all nodes. The notification stage again uses broadcast



(a) Spanning Tree

1: transmit request$(1, 0.75, -)$,
2: mark 1 as father,
2: transmit request$(2, 0.75, 1)$,
4: drop request from 2,
1: mark 2 as child,
3: mark 2 as father,
3: transmit request$(3, 0.75, 2)$,
2: mark 3 as child,
4, 5: mark 3 as father,
5: transmit request$(5, 0.75, 3)$,
4: transmit request$(4, 0.75, 3)$,
4: drop request from 5,
5: drop request from 4,

(b) Request phase

$4 \rightarrow 3$: reply$(4, 1)$,
$5 \rightarrow 3$: reply$(5, 1)$,
$3 \rightarrow 2$: reply$(3, 3)$,
$2 \rightarrow 1$: reply$(2, 4)$

(c) Reply phase

Fig. 2 Simple example of a single iteration of BSPAN as described in Algorithm 3, initiated by reference node with id 1. (a) shows the spanning tree that results from the father record at each node at the end of the iteration; edges that are not contained in the tree are shown dashed. (b) shows the request messages and the resulting actions of the nodes during the construction of the tree. (c) shows the replies that are sent along the attained spanning tree edges and the node counting operation. Note that requests reach all neighboring nodes (broadcast), while replies are sent from a child to its father (unicast). The set of transmission power levels is $P = \{0.05, 0.1, \ldots, 0.95, 1.0\}$.

messages over edges of cost at most α and constructs a spanning tree of the transmission graph. Nodes observe which of the incident edges are part of the spanning tree and then set their transmission power level to the minimum level required to reach the father and all child nodes in the tree.

6.4 Message complexity

In each iteration each node that has been reached by a request, except the reference node, sends at most two messages, one request and one reply. The reference node sends a request but no reply. Therefore, the total number of messages sent in a single iteration is at most $2|V| - 1$. The binary search over a set of power levels P requires $\lceil \log(|P|) \rceil$ iterations, where the logarithm is taken in base 2. Thus, the algorithm has message complexity $O(\log(|P|)\ |V|)$.

7 Experimental evaluation

We experimentally evaluated MLS, BSPAN, the beaconing method of Section 4.1, and the distributed method for RNG construction of Section 5.3 using the ns2 [17] network simulator. We also compared MLS and BSPAN with the previously proposed DMMT algorithm [11].

To measure the performance of the algorithms, we considered both the number of control messages and the time it takes for the algorithms to finish. In our simulations, we use the *disk graph model* to represent a wireless network topology: the networks are created by randomly scattering nodes onto a square area with given dimensions, and connectivity is defined by the ns2 default maximum transmission range. We used the TwoRayGround model as the propagation model, for it perfectly meets the conditions as outlined in Section 4, and discarded disconnected graphs.

The parameter values used for the simulations are given in Table 1. We chose a rather large number of 10 beacons per node during the setup stage to reduce the probability of repeated collisions of beaconing messages; it would also be possible to increase the time interval between beaconing messages. As our main focus is on analyzing MLS and BSPAN, we discarded trials where some node had an incomplete list of neighbors or the initializing node had the wrong node count. RNG pruning requires information about the neighbors to be broadcast, which makes the beacon packets longer and susceptible to collisions. After initially trying a smaller number of 4 beacons per node and observing a failure rate of 1 to 2% of the trials for the setup stage, we settled on a value of 10 for which the setup terminated successfully in all trials. To safeguard MLS and BSPAN against deadlocks arising from permanent node failures, one should consider implementing a timeout scheme.

7.1 Distributed minmax tree algorithm

The Distributed Min-Max Tree (DMMT) algorithm proposed in [11] determines for a given set M of nodes (the *multicast group*) a spanning tree with minmax edge cost. Choosing $M = V$, DMMT can be readily applied to solve the lifetime maximization problem as formulated in Section 3. In this paper, we focus on the version of the algorithm that was proposed for omnidirectional antennas.

The DMMT algorithm borrows ideas from the well-known Prim's algorithm for constructing minimum spanning trees (see for example [7, pp. 570-573]). Prim's algorithm grows a subtree of the original graph starting from an initial node, such that in each step the minimum cost edge is added that connects one node belonging to the tree and another node not yet in the tree. After all nodes have been added, the al-

Table 1 Simulation parameters; the input graphs were generated by random placement of nodes within the area, while disconnected graphs were discarded.

ns2 version	2.31	Node density	1 node per 130 m×130 m
Transmission range	250 m	Number of nodes	50-500
Max jitter (MLS, BSPAN)	0.5 s	Propagation model	TwoRayGround
Message timeout	2.1 s	$\lvert P\rvert$	128
Beacon delay (max)	1.5 s	Beacon repetitions	10

gorithm terminates and the resulting tree forms a minimum spanning tree.

The DMMT algorithm finds a minmax spanner by adding an additional step to each iteration, the so-called *growth phase*: after the attempt of finding the minimum outgoing-edge-cost has terminated, this cost is propagated to all tree nodes in a *join request* message. Each tree node v then forwards this message to each neighbor u that v believes is not yet in the tree if the cost of the edge (v,u) is less or equal to the minimum outgoing edge-cost. This operation corresponds to growing the tree along edges with cost less or equal to the current threshold cost. After a non-tree node has been added via an edge adjacent to the tree node, the tree node becomes the father of the non-tree node – which itself can become father of one or more non-tree nodes added during the current growth phase – in the minmax spanner being constructed.

However, DMMT does not necessarily always find an outgoing edge in the *search phase* of the algorithm, as is the case for an iteration of Prim's algorithm. This is due to the fact that nodes only learn about their neighbors being in the tree when these forward request messages to them and can thus result in costly non-progress iterations of the algorithm.

The formulation in [11] employs timers at each node in order to let the nodes distributively estimate the termination of the growth phase. In our evaluation we considered a more synchronized method initiated by the reference node to notify the nodes to switch from the growth to the search phase. This modification was considered necessary in order to make DMMT more resilient against network failures, such as packet drops at the MAC level. The additional control messages were not taken into account for the comparisons described below.

7.2 Network simulations

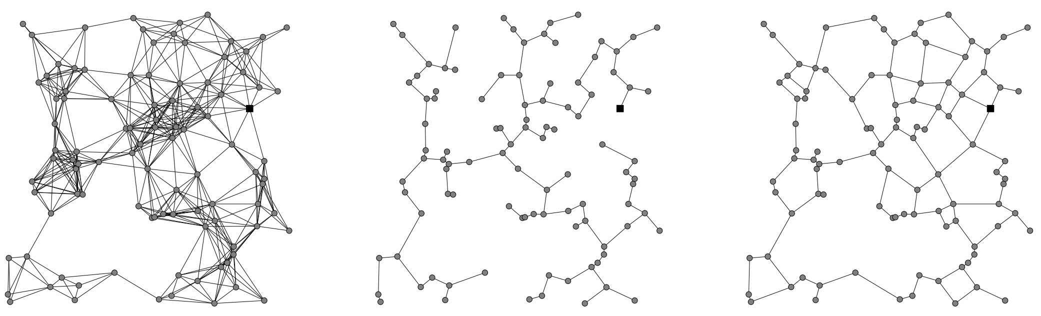
We implemented the aforementioned algorithms, DMMT, MLS, and BSPAN, as protocol agents in ns2. For MLS and BSPAN, the reference node starts the protocol by initiating Algorithm 1 to obtain the weighted neighbor lists and a count of the nodes in the network. The topology information required by DMMT is loaded onto the nodes prior to the execution of the protocol, but could also be obtained by the methods described in Section 4. For the following observations we fixed P at 128 distinct equally spaced power levels.

Figure 3 depicts one transmission graph of 100 nodes, its MST, RNG, and the trees constructed by DMMT, MLS, and BSPAN. One can see that the shortest-hop distances between pairs of nodes in the minmax spanner resulting from MLS initialized by the RNG are generally slightly longer than in the spanner resulting from the original graph. Running MLS on the RNG instead of the original graph generally reduces the number of messages required, but it also removes paths with low minmax cost and a small hopcount. The algorithms DMMT and BSPAN are insensitive to which input graph is used in terms of the number of control messages required, as DMMT considers only the single least-cost outgoing edge in each iteration and BSPAN relies on broadcast messages to all neighbors.

The total message counts of DMMT, MLS, and BSPAN averaged over a set of graph instances are depicted in Fig. 4. Note that the number of messages for MLS and BSPAN also include the messages transmitted in the setup-stage of the protocols. However, as opposed to MLS and BSPAN, the number of messages required for obtaining this information are not included in the total message counts of DMMT. Despite this handicap, both outperform DMMT significantly when comparing the number of control messages required by the protocols.

More specifically, one can see from Fig. 4(a) that DMMT requires between 2 and 6 times more messages than MLS run on the transmission graph and between 6 and more than 30 times more messages than BSPAN. Therefore, DMMT does not scale well with the size of the network. Comparing MLS with BSPAN, one can see from Fig. 4(b) that BSPAN outperforms MLS by a factor of 2.5 for 50 nodes and 4 for 200 nodes when MLS is run on the transmission graph. When using the distributed algorithm for constructing the RNG, however, MLS outperforms BSPAN by a factor of 1.5 for 50 nodes and 1.2 for 200 nodes. One should also note that BSPAN significantly benefits from using requests, which are transmitted as broadcast messages, as implicit acknowledgements.

For a fixed P the number of messages required by BSPAN is linear in the number of nodes, whereas for a fixed number of nodes the message count for BSPAN is linear in $\log\lvert P\rvert$. Figure 5 illustrates the effect of different numbers of power levels. As opposed to DMMT and MLS when run on the transmission graph, MLS run on the RNG and BSPAN scale well with the number of nodes.


(a) Transmission graph (b) MST of transmission graph (c) RNG of transmission graph

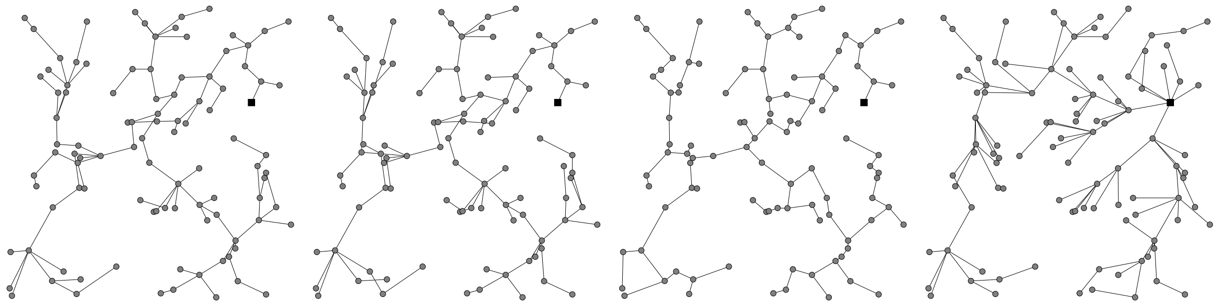
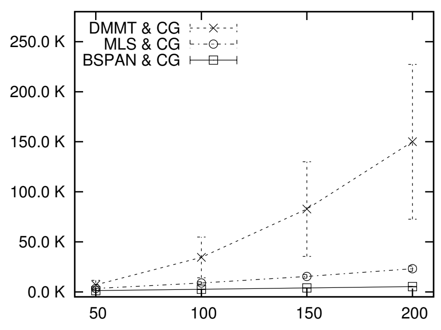
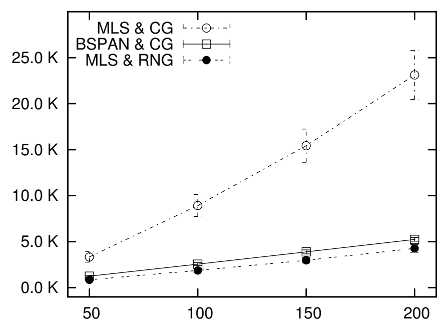

(d) Tree constructed by DMMT on transmission graph (e) Tree constructed by MLS on transmission graph (f) Tree constructed by MLS on RNG (g) Tree constructed by BSPAN on transmission graph

Fig. 3 Resulting minmax spanner for a graph with 100 nodes; the reference node is indicated by a black square and the critical edge is the long edge in the bottom left part of the graph.



(a) DMMT, MLS, and BSPAN run on transmission graph



(b) MLS and BSPAN run on transmission graph and MLS run on RNG

Fig. 4 Number of messages required by DMMT, MLS, and BSPAN versus number of nodes in the network. Error bars represent standard deviations over 200 repetitions. The number of messages for MLS and BSPAN also includes the messages of the setup stage; the notification stage was excluded from the results, as it is not part of the DMMT algorithm, although required for global termination. For BSPAN the value of $|P|$ in both plots is 128. The plots show data for MLS run on the transmission graph TG and its RNG. Note the different scale in (a) and (b).

When evaluating running time, one has to consider the effect of timers on the performance of the different protocols. Assuming a collision free network, MLS and BSPAN would only require a timer in the setup stage of the protocol, which uses beacon messages to establish local network topology information. A node v discovers that it is a leaf node if no other node u has forwarded a beacon message indicating that v is the father of u in the spanning tree constructed in the setup stage. Hence, a timer is required in order to wait a certain time to discover child nodes in the tree. In order to avoid deadlocks caused by packet collisions due to interference at a later stage in algorithms MLS and BSPAN, it was necessary to introduce retransmission timers, whose timeout values, however, only depend on the propagation delay between neighboring nodes.

The DMMT protocol makes extensive use of timers, whose values naturally have a strong impact on the running time. Figure 6 shows that BSPAN is slightly slower than MLS, and that both significantly outperform DMMT.

As already mentioned above, running MLS on the RNG instead of the original graph reduces the number of messages required, but it also removes paths with low minmax

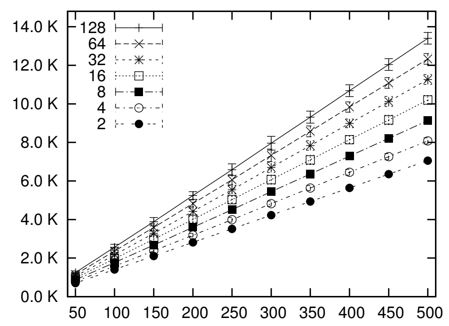


Fig. 5 Number of messages required by BSPAN versus number of nodes in the network over a range of values for $|P|$. Error bars represent standard deviations over 200 repetitions. The number of messages includes the messages of the setup stage but not the notification stage.

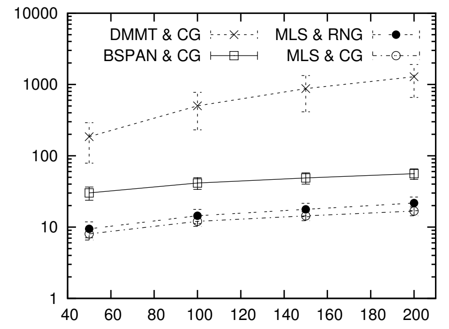


Fig. 6 Total simulated running time (in seconds). Error bars represent standard deviations over 200 repetitions; note that the total running times are plotted on a logarithmic scale. The duration of the notification stage of MLS and BSPAN was excluded from the results, as it is not part of the DMMT algorithms although required for global termination. For BSPAN the value of $|P|$ is 128. The plot shows data for MLS run on the transmission graph TG and its RNG.

cost and a small hopcount. Indeed, the experiments show a slightly higher running time, as propagating ACKs and NAKs along the tree takes longer.

8 Conclusions

We have presented two efficient distributed algorithms for the problem of lifetime maximization in a wireless sensor network with stationary nodes and static transmission power assignments. The first algorithm is based on a distributed computation of paths from a reference node to all other nodes which have minimum maximum cost, while the second algorithm performs a binary search over the range of transmission power levels. Both algorithms have been formulated as network protocols, which, unlike many previously proposed solutions to related problems, do not rely on prior knowledge of the network, such as network size or neighbor lists.

In our network simulations, using the `ns2` network simulator, both the proposed algorithms, MLS and BSPAN, significantly outperform the recently proposed Distributed Min-Max Tree algorithm in terms of the number of messages required. When run on the transmission graph, BSPAN typically shows better performance than MLS. When MLS is run on the RNG of the transmission graph, however, one observes a drastic reduction in the number of control messages required. We also present a distributed method for pruning a transmission graph to its RNG. This method enables us to reduce the number of messages required by MLS if nodes are placed in a plane and transmission costs depend only on the Euclidean distance.

Both algorithms proposed in this paper solve the lifetime maximization problem optimally. However, there are different cases in which one would prefer one algorithm over the other. Depending on the radio conditions, transmission of broadcast messages might not be feasible. Furthermore, the RNG of the transmission graph could be easily computed if the network topology is predetermined, nodes know the locations of their neighbors, and the edge costs are increasing in distance. In these cases one might prefer MLS over BSPAN. In other cases, e.g., if the number of power levels is quite small and broadcast messages are feasible, BSPAN would be the better choice. BSPAN also provides a tighter bound on the message complexity of the algorithm, as is evident from our simulation results.

A natural extension of the present work would be to consider the task of lifetime maximization under dynamic transmission power assignments. This is, however, a computationally much more challenging problem than the static one considered here [9], so obtaining an optimal solution by an efficient distributed algorithm may be impossible. Naturally, heuristic methods could be used to obtain reasonable practical solutions.

Acknowledgements A. Schumacher has been supported by the Helsinki Graduate School of Computer Science and Engineering. The MLS and BSPAN simulations were conducted in collaboration with Thorn Thaler.

References

1. Akyildiz, I.F., Su, W., Sankarasubramaniam, Y., Cayirci, E.: A survey on sensor networks. IEEE Communication Magazine **40**(8), 102–114 (2002)
2. Bhardwaj, M., Misra, S., Xue, G.: Distributed topology control in wireless ad hoc networks using β-skeletons. In: Workshop on High Performance Switching and Routing, pp. 371– 375 (2005)
3. Borbash, S., Jennings, E.: Distributed topology control algorithm for multihop wireless networks. In: Proceedings of the 2002 International Joint Conference on Neural Networks (2002)
4. Cao, Q., Abdelzaher, T., He, T., Stankovic, J.: Towards optimal sleep scheduling in sensor networks for rare-event detection. In: Proceedings of the 4th International Symposium on Information Processing in Sensor Networks, pp. 20–27. IEEE Press, Piscataway, NJ, USA (2005)
5. Cerpa, A., Estrin, D.: ASCENT: adaptive self-configuring sensor networks topologies. IEEE Transactions on Mobile Computing **3**(3), 272–285 (2004)

6. Chang, J.H., Tassiulas, L.: Energy conserving routing in wireless ad-hoc networks. In: Proceedings of the Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), pp. 22–31 (2000)
7. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. The MIT Press, Cambridge, MA, USA (2001)
8. Escalante, O., Pérez, T., Solano, J., Stojmenovic, I.: RNG-based searching and broadcasting algorithms over internet graphs and peer-to-peer computing systems. In: The 3rd ACS/IEEE International Conference on Computer Systems and Applications, pp. 47–54 (2005)
9. Floréen, P., Kaski, P., Kohonen, J., Orponen, P.: Lifetime maximization for multicasting in energy-constrained wireless networks. IEEE Journal on Selected Areas in Communications **23**(1), 117–126 (2005)
10. Gallager, R.G., Humblet, P.A., Spira, P.M.: A distributed algorithm for minimum-weight spanning trees. ACM Transactions on Programming Languages and Systems **5**(1), 66–77 (1983)
11. Guo, S., Yang, O.W.W., Leung, V.C.M.: Tree-based distributed multicast algorithms for directional communications and lifetime optimization in wireless ad hoc networks. EURASIP Journal on Wireless Communications and Networking **2007** (2007)
12. Gupta, S.K.S., Srimani, P.K.: Self-stabilizing multicast protocols for ad hoc networks. Journal of Parallel and Distributed Computing **63**(1), 87–96 (2003)
13. Kang, I., Poovendran, R.: Maximizing network lifetime of broadcasting over wireless stationary ad hoc networks. Mobile Networks and Applications **10**(6), 879–896 (2005)
14. Kohvakka, M., Suhonen, J., Hannikainen, M., Hamalainen, T.D.: Transmission power based path loss metering for wireless sensor networks. In: 17th Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, pp. 1–5 (2006)
15. Lloyd, E.L., Liu, R., Marathe, M.V., Ramanathan, R., Ravi, S.: Algorithmic aspects of topology control problems for ad hoc networks. Mobile Networks and Applications **10**(1-2), 19–34 (2005)
16. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann, San Francisco, CA, USA (1996)
17. McCanne, S., Floyd, S., Fall, K., Varadhan, K.: The network simulator `ns2` (1995). The VINT project, available for download at `http://www.isi.edu/nsnam/ns/`.
18. Narayanaswamy, S., Kawadia, V., Sreenivas, R.S., Kumar, P.R.: Power control in ad-hoc networks: Theory, architecture, algorithm and implementation of the COMPOW protocol. In: Proceedings of the European Wireless Conference (2002)
19. Ramanathan, R., Rosales-Hain, R.: Topology control of multihop wireless networks using transmit power adjustment. In: Proceedings of the Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), pp. 404–413 (2000)
20. Rodoplu, V., Meng, T.H.: Minimum energy mobile wireless networks. IEEE Journal on Selected Areas in Communications **17**(8), 1333–1344 (1999)
21. Srinivasan, K., Levis, P.: RSSI is under-appreciated. In: Proceedings of the Third Workshop on Embedded Networked Sensors (EmNets), (2006)
22. Toussaint, G.T.: The relative neighbourhood graph of a finite planar set. Pattern Recognition **12**, 261–268 (1980)
23. Wattenhofer, R., Li, L., Bahl, P., Wang, Y.M.: Distributed topology control for power efficient operation in multihop wireless ad hoc networks. In: Proceedings of the Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), pp. 1388–1397 (2001)