Lifetime Maximization in Wireless Sensor Networks by Distributed Binary Search

André Schumacher, Pekka Orponen, Thorn Thaler, and Harri Haanpää

Lab. for Theoretical Computer Science, TKK – Helsinki University of Technology, P.O. Box 5400, FI-02015 TKK, Finland
Andre.Schumacher@tkk.fi, Pekka.Orponen@tkk.fi, Tthaler@tcs.hut.fi, Harri.Haanpaa@tkk.fi

Abstract. We consider the problem of determining the transmission power assignment that maximizes the lifetime of a data-gathering wireless sensor network with stationary nodes and static transmission power levels. We present a simple and efficient distributed algorithm for this task that works by establishing the minimum power level at which the network stays connected. The algorithm is based on a binary search over the range of feasible transmission power levels and does not require prior knowledge of network topology. We study the performance of the resulting BSpan protocol by network simulations and compare the number of control messages required by BSpan to two other recently proposed methods, the Distributed Min-Max Tree (DMMT) and Maximum Lifetime Spanner (MLS) algorithms. We find that BSpan outperforms both DMMT and MLS significantly.

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 determine data gathering and transmission protocols so that the operational time of the network, for given initial battery levels, is maximized [1, 2].

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 [3].

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. This version of the problem was considered by Lloyd et al. [4] who presented a simple and efficient binary search based solution to it, assuming that the full internode transmission power threshold matrix of the network is centrally available.

Our Binary Search for Minmax Power Spanner (BSpan) algorithm presented below is basically a distributed implementation of the "binary search over transmission power levels" idea of Lloyd et al. [4]. However, getting this natural approach to work in a fully distributed environment, starting in an initial state where the nodes upon wakeup know nothing about their neighbors, let alone the global topology of the network, is a somewhat nontrivial task. Nevertheless, we have implemented this approach down to the level of a protocol agent in the `ns2` [5] simulator, and it shows quite competitive performance in comparison with other recently proposed approaches to the same task. In graph-theoretic terms the algorithm finds a spanning tree with maximum edge cost at most ϵ greater than the minimum maximum edge cost possible, where ϵ is a parameter of the algorithm. Thus we obtain a power assignment that, to arbitrary accuracy, maximizes the time for which we can keep the network connected.

The rest of the paper is organized as follows. The following section overviews some of the related work on lifetime maximization, and Section 3 gives a precise formulation of the version of the problem we consider. Section 4 describes our distributed method for finding a spanning tree of a given network with minimum maximum transmission cost. In Section 5 we evaluate our proposed BSpan algorithm in terms of the number of required control messages, and compare it to the performance of the Distributed Min-Max Tree algorithm proposed in [6] and the Maximum Lifetime Spanner (MLS) algorithm proposed in [7]. For our experimental comparison we use the `ns2` network simulator. Section 6 summarizes the paper.

2 Related Work

The problem of minimizing the maximum transmission power required to establish connectivity has been considered previously in the literature several times. One of the earliest papers on the topic is the work of Ramanathan and Rosales-Hain [8], 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 minimum maximum (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 [9] 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. [4,9].

The problem of minimizing the *total*, as opposed to minmax, network transmission power required for connectivity has been studied extensively (cf. e.g. [4] and the references therein). Rodoplu and Meng [11] 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 [11] is optimal but requires extensive assumptions, such as the availability of location information and a specific path-loss model.

In a recent work, Guo, Yang, and Leung [6] 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 BSpan approach, we conducted an experimental comparison of the runtime behavior of the algorithms DMMT, the recently proposed Maximum Lifetime Spanner (MLS) algorithm [7] and our proposed BSpan algorithm.

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, either using a possibly large set of discrete power levels, or by choosing the power from a continuous range of possible values. We further assume that each node has a finite energy budget that is consumed during the operation of the network and whose value is 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 possibly be applied, thereby yielding a close to uniform load within the network.

In order to maintain connectivity to a neighbor u, each 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 link 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 [12].

The transmission structure of the network is modelled as a graph $G = (V, E)$ with an associated edge cost function $\delta : E \mapsto \mathbb{R}^+$. Here δ is scaled so that v can reach u as long as $\tau(v) \geq \delta(v, u)\, p^{\max}$. A transmission power assignment $\tau : V \mapsto \mathbb{R}^+$ induces a graph $G(\tau) = (V, E(\tau))$ whose edges represent the radio links that are supported by the given assignment τ: an edge (v, u) is an element of $E(\tau)$ if and only if $\tau(v) \geq \delta(v, u)\, p^{\max}$, i.e. if and only if node v transmits at a power that is sufficient for u to correctly receive messages from v. For simplicity, we assume that $G(\tau^{\max})$ with $\tau^{\max}(v) = p^{\max}$ for all v is a connected graph.

We consider the problem of finding a static transmission power assignment $\tau : V \mapsto [0, p^{\max}]$, such that the lifetime of the network is maximized while the network remains connected. Within the context of this problem, 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 $\tau(v) = \alpha\, p^{\max}$ 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 [13].

Definition 1. *Given a set of nodes V, and an edge cost function $\delta : V \times V \mapsto \mathbb{R}^+$, a graph $G = (V, E)$ 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 graph for the nodes in V where the no edge has cost greater than α. Note that since we normalize the edge costs for any network a 1-spanner exists exactly when the network can be connected by the nodes sending at full power.

Definition 2. *For given V and δ, an α-spanner is ϵ-optimal, if $\alpha' \geq \alpha - \epsilon$ for all α'-spanners.*

A 0-optimal α-spanner has a maximum edge cost α, but there are no spanners with only edges of cost less than α; thus, we also call such spanner a *minmax cost spanner*. Network lifetime can now be maximized by determining the minmax cost spanner (V, E) and choosing the power assignment $\tau(v) = p^{\max} \max_{(v,u)\in E} \delta(v, u)$.

4 Binary Search for a Minmax Power Spanner

We propose a distributed algorithm for determining an ϵ-optimal α-spanner, given a graph with edge costs and the accuracy parameter ϵ. The algorithm determines transmission power levels that maximize the time until the first node runs out of energy.

The resulting protocol BSpan consists of three stages. All stages are initiated by a designated reference node which also detects their termination (except for the final stage that only requires local termination), but otherwise the computations proceed without central coordination. The protocol thus admits an efficient implementation in a distributed setting such as a wireless sensor network.

In the first stage, nodes collect neighborhood information and estimate link costs by transmitting and receiving beacon messages. The reference node also obtains a count of the number of nodes in the network, assuming the transmission graph $G(\tau_{\max})$ is connected. The second stage of the protocol performs a binary search over the range of possible transmission power levels. The final stage consists of a network broadcast in which the reference node notifies all other nodes of the global termination of the algorithm and the resulting minmax power level. The three stages of the BSpan protocol are executed in the order indicated in Fig. 1. We now discuss the three stages of the protocol in more detail.

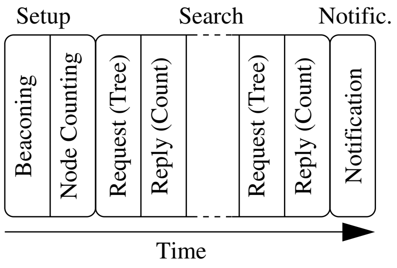


Fig. 1. Overview of the different stages of the BSpan protocol

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. 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 child nodes 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 parent 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.

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 power p^{recv}, arrived successfully if $p^{\text{recv}} \geq p^{\text{thresh}}$, where p^{thresh} is the threshold power required for a successful transmission (disregarding interference). In our simulations, the power p^{recv} is computed by the propagation model under consideration. Assuming that the received power depends linearly on the sending power, $p^{\text{recv}} = X_{u,v}\, p^{\text{send}}$, and the receiver knows the sending power used, the receiver v can estimate the attenuation coefficient $X_{u,v} = p^{\text{recv}}/p^{\text{send}}$. 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 $p^{\text{thresh}} = X_{v,u}\, p^{\text{min}}$. Combining these, we have

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

where p^{send} is the power that was used by u for sending. If the assumption does not hold or if the measured transmission power 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 link costs $\delta(v, u)$ are normalized so that the minimum power required for v to send to u is $p^{\text{min}} = \delta(v, u)\, p^{\text{max}}$; due to the normalization the link costs lie in the interval $[0, 1]$, and v can estimate the link cost as $\delta(v, u) = p^{\text{min}}/p^{\text{max}}$, or if the maximum power was used for sending, $p^{\text{send}} = p^{\text{max}}$ and $\delta(v, u) = p^{\text{thresh}}/p^{\text{recv}}$.

When v has sent a certain number of beacon messages, it decides that the setup phase 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 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) for a system with IEEE 802.11 network interfaces or, alternatively, methods similar to the ones proposed in [14].

Algorithm 1. Setup stage

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

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

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

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

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. In the absence of interference and resulting collisions, it would be even sufficient that each node beacons exactly once. Thus, the message complexity of the beaconing stage is $O(N)$, where N is the number of nodes in the network.

4.2 Search Stage

After the global termination of the setup stage, each node has information of all other nodes in its maximum transmission range and the costs of the incident edges. The reference node also knows the total number of nodes within the network. In the second stage Algorithm 2 performs a binary search over the range of possible transmission power levels, coordinated at the reference node. 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 reference node then checks whether this tree spans all nodes in the network. After the search has terminated, the reference node informs all other nodes about the termination and the minimum edge cost necessary to connect all nodes. This cost can then be used to locally determine the transmission power level required at each node.

Each iteration of the binary search algorithm consists of two steps: The first step is initiated by the reference node and consists of a flooding of request messages over edges with cost at most α. The second step consists of a convergecast of reply messages back to the reference node in order to count the nodes in the tree computed in the first step. This count is then compared by the reference node to the total number of nodes in the network.

The request messages are of the form (v, α, f) where v is the identity of the sending node, α is the maximum allowable link cost in this iteration, and f is the parent 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 link (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 link (v, w) has cost less than or equal to α.

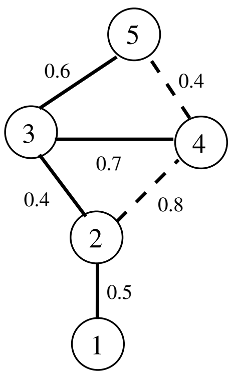
After sending the request $(v,\ \alpha,\ u)$, v waits for a request from any w that meets the condition above. In the case that v receives a request $(w,\ \alpha,\ v')$ from w, it will mark w as *child* if $v' = v$, and as *processed* otherwise. A neighbor marked *child* corresponds to w being v's child 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.*

Algorithm 2. Search stage

```
node $v$ with variables $\alpha$, $\alpha_{\max} = 1$, $\epsilon$, expecting_msg_from, father, lower,
is_reference_node, $N(v)$, node_count, status, upper ;

at start
    if is_reference_node then
        $\alpha \leftarrow \alpha_{\max}/2$; lower $\leftarrow 0$; upper $\leftarrow \alpha_{\max}$;
    for $u \in N(v)$ do  status[$u$] $\leftarrow$ processed;
    enter state RESET;

in state RESET
    father $\leftarrow$ none; node_count $\leftarrow 0$;
    if is_reference_node then
        if $\epsilon <$ upper $-$ lower 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 $\alpha' \leq \delta(u, v)$ is received then
        $\alpha \leftarrow \alpha'$; father $\leftarrow u$;
        enter state SEND_REQUEST;
    end

in state SEND_REQUEST               // broadcast a request to neighboring nodes
    expecting_msg_from $\leftarrow \{w \in N(v) \setminus \{$father$\} \mid \delta(v, w) \leq \alpha\}$;
    for $w \in$ expecting_msg_from do  status[$w$] $\leftarrow$ wait;
    if expecting_msg_from $\neq \emptyset$ 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[$u$] $\leftarrow$ child;
        else                                    // u has father $f'$ different from v
            status[$u$] $\leftarrow$ processed;
    end
    if reply($u$, nodes) is received then
        status[$u$] $\leftarrow$ processed; node_count $\leftarrow$ node_count + nodes;
    if status[$w$] = processed for all $w \in N(v) \setminus \{$father$\}$ then   // end of iteration
        if is_reference_node then
            if total_nodes = node_count then  upper $\leftarrow \alpha$;
            else  lower $\leftarrow \alpha$;
            $\alpha \leftarrow$ (upper + lower)$/2$;
        else
            unicast reply($v$, node_count + 1) to father;          // report node count
        enter state RESET;
    end
```

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



(a) Spanning Tree

1: transmit request$(1, 0.75, -)$,
2: mark 1 as parent,
2: transmit request$(2, 0.75, 1)$,
4: drop request from 2,
1: mark 2 as child,
3: mark 2 as parent,
3: transmit request$(3, 0.75, 2)$,
2: mark 3 as child,
4, 5: mark 3 as parent,
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) Spanning Tree Construction

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

(c) Node Counting

Fig. 2. Simple example of a single iteration of the search stage as described in Algorithm 2, initiated by reference node with id 1. (a) shows the spanning tree that results from the parent 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 parent (unicast).

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 2 for details and Fig. 2 for a toy example of a single iteration of the search stage.

In each iteration each node that has been reached by a request, except the reference node and nodes that have no adjacent neighbors that they could reach with cost at most α, sends exactly 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 $2(N-1)+1$, where N is the number of nodes in the network. The binary search over intervals of size ϵ of the range $[0, 1]$ requires $\lceil \log(1/\epsilon) \rceil$ iterations, where the logarithm is taken in base 2. Thus, the search stage has message complexity $O(N \ \log(1/\epsilon))$.

Note that in a realistic setting the different costs resulting from the available transmission power levels are not necessarily equidistant. Instead, the possible power levels would be represented by an ordered set PL of real numbers. In this case, instead of using the range $[0, 1]$ to represent the edge costs, one would perform a search on the set of available power levels using their rank, rather than their cost value. The number of iterations required by the algorithm would then be $\lceil \log(|PL|) \rceil$, where $|PL|$ is the number of distinct power levels.

4.3 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 and the final edge cost value α (the upper bound in Algorithm 2). Notification messages are sent at power α and nodes keep track from which node they first receive a notification message. In this way, similar to the previous stages, the notification stage constructs a spanning tree of the transmission graph that is an ϵ-optimal α-spanner. From the adjacency list and by listening to notification messages all nodes can infer locally their power level assignment.

The number of messages that are required for the notification stage is at most N. However, the search stage clearly dominates the message complexity of the protocol. We therefore conclude that the complete BSpan protocol has message complexity $O(N\ \log(1/\epsilon))$.

5 Experimental Evaluation

We experimentally evaluated the algorithm described in the previous section using the `ns2` [5] network simulator and compared it to two previously proposed algorithms, Distributed Min-Max Tree (DMMT) [6] and Maximum Lifetime Spanner (MLS) [7]. 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. The network topologies were generated by scattering nodes randomly in a square, whose dimensions were chosen such that the expected node density was constant for all number of nodes. We used the TwoRayGround model as the propagation model, for it perfectly meets the conditions as outlined in Section 4.1. Instances for which the placement does not yield a connected network were discarded from the simulations. We also disregard simulation runs that result in an incorrect node count due to beacon collisions during the setup stage of BSpan, as this event possibly invalidates results obtained during the later stages. During the experiments this event occurred in at most 0.5% of runs for any network size. Refer to Table 1 for the list of simulation parameters.

Table 1. Simulation parameters; the input graphs were generated by random placement of nodes within the area, while disconnected graphs and graphs for which the beaconing did not yield the correct node count were discarded

`ns2` version	2.31	Node density	1 node per 130 m×130 m
Transmission range	250 m	Number of nodes	50-500
Number of nodes	50-500	Propagation model	TwoRayGround
Max jitter (BSpan)	0.5 s	BSpan iterations	7
Message timeout	2.1 s	ϵ	2^{-7}
Beacon delay (max)	1.5 s	Beacon repetitions	3

5.1 Distributed Min-Max Tree Algorithm

The DMMT algorithm proposed in [6] 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 2. 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. 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 algorithm terminates and the resulting tree forms a minimum spanning tree.

The DMMT algorithm finds a minimum power 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 u then forwards this message to each neighbor v that u believes is not yet in the tree if the cost of the edge (u, v) 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 *parent node* of the newly added node which becomes a *child node* of its parent.

However, the DMMT algorithm 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 result in costly non-progress iterations of the algorithm.

The formulation in [6] 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. Additional control messages were not taken into account for the comparisons described below.

5.2 Maximum Lifetime Spanner Algorithm

The Maximum Lifetime Spanner (MLS) algorithm proposed in [7] uses a breadth-first search approach to construct paths with minmax edge cost, which are combined to form a minmax power spanner of the transmission graph. Starting from the reference node, messages containing the lowest edge cost known so far are propagated in the network. Upon receiving a request for the first time from a neighbor v containing the maximum edge cost α on the path the request has taken from the reference node to v, each node u keeps track of its father and forwards the message to its neighbors. When u forwards the message received from v, u updates α to be the maximum of α and the cost of the edge (v, u).

If a node learns about a better route during the execution of the algorithm, it informs the old father by sending a NAK message (negative acknowledgement) to its father. Then the node changes the father to the node it learns the better route from. As soon as it has received replies form each of its neighbors, the node sends an ACK message (positive acknowledgement) to its father.

These ACK and NAK messages serve two purposes. Firstly, they allow each node to be aware about both, its father and its children, and secondly, they guarantee the termination of the algorithm. As soon as the reference node has received acknowledgements, either positive or negative, from all its neighbors, the algorithm terminates. At termination each node u knows about which of its neighbors it is responsible for. Thus, it can decide which power level it has to choose to assure connectivity.

5.3 Network Simulations

We implemented the aforementioned algorithms, BSpan, DMMT and MLS, as protocol agents in `ns2`. All three protocols are initiated by the designated reference node. DMMT and MLS require topology information in the form of neighbor lists and edge costs that are loaded into the nodes prior to the execution of the algorithms. For BSpan, the reference node starts the protocol by initiating the setup stage to obtain the weighted neighbor lists and a count of the nodes in the network. The total message counts of BSpan, DMMT and MLS are depicted in Fig. 3(a). Both DMMT and MLS require prior topology information in the form of neighbor lists. However, as opposed to BSpan the number of messages required for obtaining this information are not included in the total message counts of DMMT and MLS. Despite this handicap, BSpan outperforms MLS by a factor of 3 for 50 nodes and 5 for 200 nodes. DMMT even requires between 8 and more than 40 times more messages than BSpan and therefore does not scale well with the size of the network. One should note that BSpan also benefits from the broadcast advantage of wireless networks.

Recall that BSpan is guaranteed to find an ϵ-optimal spanner. For a fixed ϵ 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 $\log 1/\epsilon$. Figure 3(b) illustrates the effect of different values for ϵ. As opposed to DMMT and MLS, BSpan scales well with the number of nodes. For our further observations we fixed ϵ at 2^{-7} which corresponds to a difference of less than one percent compared to the optimal value.

When evaluating running time, one has to consider the effect of timers on the performance of the different protocols. Assuming a collision free network, BSpan would only require a timer in the setup stage of the protocol. However, as this assumption is not necessarily realistic, one has to introduce a retransmission timer into the search stage of the protocol in order to avoid the following erroneous state: When a request that was sent by a node v is not received correctly at node u due to a collision at node u, or the later request sent by u is not received due to a collision at v, the node v will end up in a deadlock. This

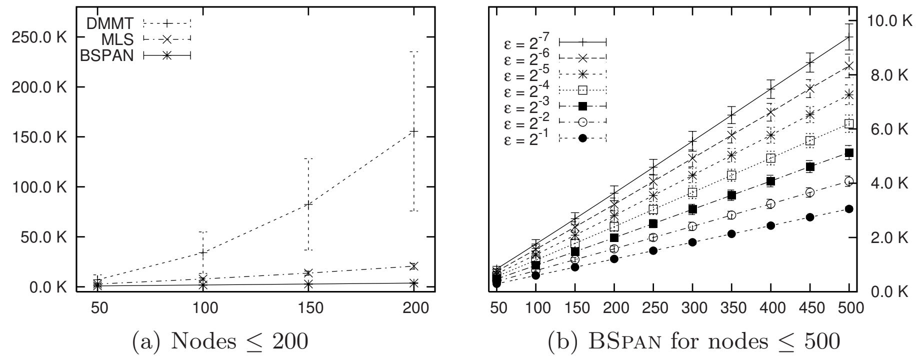


Fig. 3. Number of messages required by BSpan, DMMT and MLS for networks of increasing size. Errorbars represent standard deviations over 1000 repetitions. The number of messages for BSpan includes the messages in the setup and search stages; the notification stage was excluded from the results, as it is not part of the DMMT and MLS algorithms, although required for global termination of the algorithms. The choice for ϵ in (a) is 2^{-7}. Note the different scale in (a) and (b).

is due to the fact that v waits for u to broadcast a request, possibly indicating v to be its parent, or unicast a reply message if u is in fact a leaf node.

To avoid the situation above, a retransmission timer ensures that v sends a copy of the last request by unicast every 2.1 s until the node receives any message from the particular neighbor, whose reply is outstanding. Unicast communication is sufficient as possibly not all neighbors of v are required to react. A node u that receives a retransmitted request from node v will either unicast its last transmitted request back to v in order to signal v that v is not u's parent in the

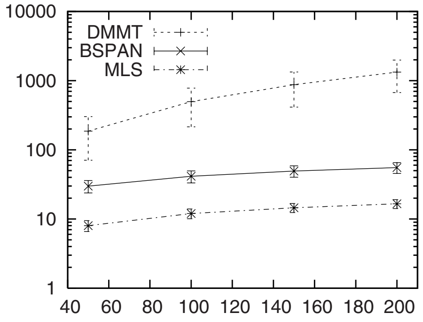


Fig. 4. Total simulated running time (in seconds). Errorbars represent standard deviations over 1000 repetitions; note that the total running times are plotted on a logarithmic scale. The duration of the notification stage of BSpan was excluded from the results, as it is not part of the DMMT and MLS algorithms, although required for global termination of the algorithms.

current iteration, or process it as usual, if it has not processed any request in the iteration previously.

The second timer involved is used in the setup stage to allow a leaf node to wait some time before it starts to reply, in order to discover neighbors. This timer triggers after $\Delta =$ (Number of Beacons) $\times$ (Maximum Jitter) $= 1.5\,\mathrm{s}$ after the node has broadcasted the request for the last time, since, unless all beacon messages were dropped, a potential neighbor sends its last broadcast message after $\Delta\,\mathrm{s}$ at the latest. In a collision free network this timer could be set to $\Delta = 2 \times$ (Maximum Propagation Delay), for in the ideal case no jitter would be needed, and the only delay is due to radio propagation.

The DMMT protocol makes extensive use of timers, whose values naturally have a strong impact on the running time. Figure 4 shows that BSpan is slightly slower than MLS, which - one could argue - is partly due to the absence of timers in MLS and that BSpan significantly outperforms DMMT.

6 Conclusions

We have presented an efficient distributed algorithm for the problem of lifetime maximization in a wireless sensor network with stationary nodes and static transmission power assignments. Unlike many previously proposed algorithms for related problems, our algorithm does not rely on prior knowledge of the network, such as network size or neighbor lists. The algorithm is based on a binary search for the minimum maximum edge cost that is required to connect the network, where connectivity is determined in each iteration of the algorithm by counting the nodes reachable from the reference node.

The algorithm has been formulated as a network protocol BSpan and implemented using the `ns2` network simulator. In our experiments comparing the runtime behavior of BSpan to the DMMT algorithm for constructing minmax trees and the previously proposed MLS protocol, BSpan systematically outperforms both other algorithms in terms of number of control messages generated, and it also performs clearly better than DMMT in terms of execution times.

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 [13]. One possible (suboptimal) heuristic would be to build a dynamic schedule iteratively from solutions to appropriately scaled static problems, using the BSpan protocol as an auxiliary routine.

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. Cerpa, A., Estrin, D.: ASCENT: adaptive self-configuring sensor networks topologies. IEEE Transactions on Mobile Computing 3(3), 272–285 (2004)

3. 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)
4. 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)
5. McCanne, S., Floyd, S., Fall, K., Varadhan, K.: The network simulator `ns2` (1995) The VINT project (1995), `http://www.isi.edu/nsnam/ns/`
6. 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. Article ID 98938, p. 10 (2007)
7. Haanpää, H., Schumacher, A., Thaler, T., Orponen, P.: Distributed computation of maximum lifetime spanning subgraphs in sensor networks. In: Zhang, H., Olariu, S., Cao, J., Johnson, D. (eds.) MSN 2007. LNCS, vol. 4864, Springer, Heidelberg (2007)
8. Ramanathan, R., Hain, R.: Topology control of multihop wireless networks using transmit power adjustment. In: INFOCOM. Proceedings of the Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies, pp. 404–413 (2000)
9. Kang, I., Poovendran, R.: Maximizing network lifetime of broadcasting over wireless stationary ad hoc networks. Mobile Networks and Applications 10(6), 879–896 (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. Rodoplu, V., Meng, T.H.: Minimum energy mobile wireless networks. IEEE Journal on Selected Areas in Communications 17(8), 1333–1344 (1999)
12. Chang, J.H., Tassiulas, L.: Energy conserving routing in wireless ad-hoc networks. In: INFOCOM. Proceedings of the Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies, pp. 22–31 (2000)
13. 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)
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)