Floréen, Patrik; Kaski, Petteri; Kohonen, Jukka; Orponen, Pekka

**Multicast time maximization in energy constrained wireless networks**

*Published in:*
DIALM-POMC '03 Joint Workshop on Foundations of Mobile Computing, San Diego, CA, USA, 19.9.2003

*DOI:*
10.1145/941079.941087

Published: 01/01/2003

*Peer reviewed version*

*Please cite the original version:*
Multicast Time Maximization in Energy Constrained Wireless Networks

Patrik Floréen
Helsinki Institute for Information Technology
Basic Research Unit
P.O. Box 26
FIN-00014 University of Helsinki, Finland
patrik.floreen@cs.helsinki.fi

Jukka Kohonen
Helsinki Institute for Information Technology
Basic Research Unit
P.O. Box 26
FIN-00014 University of Helsinki, Finland
jukka.kohonen@cs.helsinki.fi

Petteri Kaski
Laboratory for Theoretical Computer Science
P.O. Box 5400
FIN-02015 Helsinki University of Technology,
Finland
petteri.kaski@hut.fi

Pekka Orponen
Laboratory for Theoretical Computer Science
P.O. Box 5400
FIN-02015 Helsinki University of Technology,
Finland
pekka.orponen@hut.fi

ABSTRACT
We consider the problem of maximizing the lifetime of a given multicast connection in a wireless network of energy-constrained (e.g. battery-operated) nodes, by choosing ideal transmission power levels for the nodes relaying the connection. We distinguish between two basic operating modes: In a static assignment, the power levels of the nodes are set at the beginning and remain unchanged until the nodes are depleted of energy. In a dynamic assignment, the powers can be adjusted during operation.

We show that lifetime-maximizing static power assignments can be found in polynomial time, whereas for dynamic assignments, a quantized-time version of the problem is NP-hard. We then study the approximability of the quantized dynamic case and conclude that no polynomial time approximation scheme (PTAS) exists for the problem unless \( P = NP \). Finally, by considering two approximation heuristics for the dynamic case, we show experimentally that the lifetime of a dynamically maintained multicast connection can be made several times longer than what can be achieved by the best possible static assignment.

 Categories and Subject Descriptors
G.2.2 [Discrete Mathematics]: Graph Theory—network problems; C.2.1 [Computer-Communication Networks]: Network Architecture and Design—wireless communication

General Terms
Algorithms, Performance, Theory

Keywords
optimization, energy saving, multi-hop packet networks, wireless networks, sensor networks, multicasting, Steiner tree packing

1. MOTIVATION AND STATEMENT OF THE PROBLEM

Wireless “ad hoc” communication networks, consisting of a collection of radio transceivers with no prearranged infrastructure, have been studied intensively during the past few years [21]. The general area of topology control in ad hoc networks [23] is concerned with assigning appropriate transmission power levels to the nodes so that some desired topological property holds in the induced transmission graph. A central problem in this area is the maintenance of a multicast connection from a given source node to a group of sink nodes.

Battery power is a serious limiting constraint in many applications of ad hoc networks, and accordingly much attention has been paid to energy-efficient designs in this area [12]. In a wireless node, power is used for transmitting and receiving data, internal data processing, and simply for being “on” in an idle mode. The power required for transmission and reception, however, dominates [15, 27]. For simplicity, we consider only the power required for transmission. A similar model is considered in e.g. the papers [15, 17, 18].

We model an ad hoc network as a directed graph with the transceivers as nodes. Node \( i \) can communicate directly
to node $j$, if the transmission power from $i$ exceeds some threshold value $d_{ij}$ [17, 24]. As a consequence, when the power is high enough to reach a certain node, some other nodes may be reached simultaneously with the same transmission. In other words, the minimum “cost” of transmission from node $i$ to nodes $j_1, j_2, \ldots, j_m$ is $\max_{k} d_{ik}$ rather than $\sum_{k} d_{jk}$. With a particular setting of transmission powers for the nodes, node $j$ may transmit with power different from $i$, so a direct connection from $i$ to $j$ does not imply that direct communication from $j$ to $i$ is possible. Note that this is different from the symmetric model employed in e.g. [1].

In reality, the actual transmission powers and threshold values $d_{ij}$ depend on several environmental and technological factors [24], as well as situations of use (e.g. QoS requirements), but these go for the most part into the computation of proper $d_{ij}$ values in the given setting, and therefore our model is independent of such technical details. A common approximation is to choose $d_{ij} \propto r_{ij}^{-\alpha}$, where $r_{ij}$ is the physical distance between nodes $i$ and $j$, and the exponent $\alpha$ in general change over time, each $p_i$ (and hence $p$) is actually a function of time $t$. A power assignment that stays constant over time is static. More precisely, a static assignment provides for each node $i$ a power value $c_i$ and time $t_i$, so that $p_i(t) = c_i$ for $0 \leq t < t_i$ and $p_i(t) = 0$ for $t \geq t_i$. More general power assignments are dynamic.

**Definition 3**. The transmission graph $G^p_{t_{\tau}}$ induced by a particular power assignment $p = (p_1, p_2, \ldots, p_n)$ at time $t_{\tau} \geq 0$ is the subgraph obtained from the power threshold graph $G$ by including only those arcs $(i, j)$ for which $p_i(t_{\tau}) \geq d_{ij}$.

Note that we use the term “transmission graph” as in [23], and differently from [1] where this term is used for a symmetric power threshold graph.

**Definition 4**. The lifetime of a node $i \in V$ with initial energy supply $c_i$, is $t_i = \sup \{ t \mid w_i(t) \leq c_i \}$, where $w_i(t)$ is the energy consumed at node $i$ up to time $t$. During the time interval $[0, t_i[$, the node is alive. A node that is not alive is dead.

**Definition 5**. The multicast connection time for a set of sink nodes $U \subseteq V$, with respect to source node 1, in a power threshold graph $G$ subject to a power assignment $p$ is the supremum over $t_{\tau} \geq 0$ such that for each $i \in U$ there is a directed path connecting 1 to $i$ in all graphs $G^p_{t_{\tau}}$ for $t_{\tau} \leq t$.

In other words, the multicast connection time for a set of sink nodes $U$ is the maximum continuous amount of time starting from instant $t = 0$ such that there are connections in $G^p_{t_{\tau}}$ from source 1 to all the nodes in $U$. As a limiting case, e.g. if $U$ is empty, the multicast connection time can also be infinite.

**Definition 6**. A power assignment $p = (p_1, p_2, \ldots, p_n)$ is feasible with respect to a given energy supply $e = (e_1, e_2, \ldots, e_n)$, if the energy expenditures determined by $p$ satisfy $w_i(t) \leq e_i$ for all $i \in V$ and $t \geq 0$.

**Definition 7**. The multicast time maximization problem for a given power threshold graph $G = (V, A, d)$, energy supply $e = (e_1, \ldots, e_n)$, and set of sink nodes $U \subseteq V$ is to determine a feasible power assignment $p$ that maximizes the multicast connection time for $U$.
Note that several of the nodes may run out of energy before the source does so. Ensuring the connectivity from the source to the sinks with the help of other nodes is analogous to finding so-called Steiner trees [11, 22] in undirected graphs.

**Definition 8.** A Steiner tree in an undirected graph \( G = (V, E) \) with a set of critical nodes \( W \subseteq V \) is a subtree \( T \) of \( G \) that includes all the nodes in \( W \).

The rest of the paper is organized as follows. In Section 2, we study multicast time maximization by static power assignments and provide a polynomial time algorithm for this problem, drawing on the results of [17]. In Section 3, we turn to general dynamic power assignments and show that a quantized-time version of the multicast time maximization now becomes NP-hard. The lemma showing that (our version of) the underlying “Steiner tree packing problem” is NP-complete is of interest in itself. We also study the approximability of the dynamic case and conclude that no polynomial time approximation scheme (PTAS) exists for the quantized-time version of the problem unless \( P = NP \). In Section 4 we suggest two randomized approximation methods for finding good dynamic assignments, and in Section 5 we report on simulation experiments. We conclude with a summary and some ideas for further research in Section 6.

2. STATIC POWER ASSIGNMENTS

The following definition is from [4, 17].

**Definition 9.** A graph property \( P \) is monotone, if adding edges to a graph can never make \( P \) change from true to untrue. In terms of transmission graphs, this means that increasing power at nodes preserves the property.

The property we consider in this paper is the existence of directed paths from source node 1 to all sink nodes \( i \in U \). This multicast property is clearly monotone.

The following definition is a generalization of the MaxP problem given in [17].

**Definition 10.** Given a power threshold graph \( G = (V, A, d) \), let \( f_i \) be for each node \( i \in V \) some quantity that is monotonically increasing in the power applied at node \( i \). The problem Maxf with respect to property \( P \) is to determine static \( f_i \)-values for all the nodes \( i \in V \) in such a way that the corresponding transition graph has property \( P \), and the maximum of the \( f_i \)-values at the nodes is minimized.

The following is a slight generalization of the fundamental lemma (Lemma 4.1) in [17], in which the \( f_i \)-values considered are exactly the transmission powers at the nodes.

**Lemma 1.** For any instance of the Maxf problem with respect to a monotone \( P \), there is an optimal solution in which the \( f_i \)-values at each node are equal.

**Proof.** As the \( f_i \)-values are monotonically increasing in the power values, the monotonicity of property \( P \) can equally well be considered with respect to the \( f_i \)-values as with respect to the power values.

Consider an optimal solution to the given instance where the nodes do not necessarily have the same \( f_i \)-values. Let \( Q \) denote the maximum \( f_i \)-value assigned to any node \( i \). Since the property \( P \) is monotone, for any node whose \( f_i \)-value is less than \( Q \), we can increase it to \( Q \) without destroying the property.

Now we can prove the following straightforward generalization of Theorem 4.1 in [17]. For completeness, we include the proof, even though it includes only small modifications to the proof given in [17].

**Theorem 1.** For any monotone graph property \( P \) that can be tested in polynomial time, the Maxf problem can be solved in polynomial time.

**Proof.** For any instance of the problem, there exists by the preceding lemma an optimal solution with equal \( f_i \)-values at the nodes. The basic idea is that there are only a polynomial number of possible \( f_i \)-values and we can actually search through this set of possible values efficiently using, e.g., binary search.

Consider any node \( i \in V \). The number of different power values that need to be considered for \( i \) is at most \( n \), since at most one new power value is needed for each node \( j \neq i \) to ensure the communication from \( i \) to \( j \), namely \( d_{ij} \). Therefore, for all of the \( n \) nodes, the total number of corresponding candidate \( f_i \)-values to be considered is at most \( n^2 \).

Each candidate \( f_i \)-value corresponds to a power assignment, and the induced directed transmission graph can be constructed in \( O(n^2) \) time. Let \( F_P(n) \) denote the time needed to test whether property \( P \) holds for a directed graph with \( n \) nodes. Thus, the time needed to test whether property \( P \) holds for each candidate solution value is \( O(n^2 + F_P(n)) \). An optimal solution can be obtained by sorting the \( O(n^2) \) candidate solution values and using binary search to determine the smallest \( f_i \)-value for which property \( P \) holds. Since the number of candidate solution values is \( O(n^2) \), the time taken by the sorting step is \( O(n^2 \log n) \). The binary search would try \( O(\log n) \) candidate solution values and the time spent for testing each candidate is \( O(n^2 + F_P(n)) \). Thus, the total running time of this algorithm is \( O((n^2 + F_P(n)) \log n) \), i.e., polynomial.

From this proof we learn that we need consider only a discrete set of possible power values, i.e., those that coincide with some \( d_{ij} \). The same idea will appear in the algorithms for the dynamic case in Section 4.

**Corollary 1.** The multicast time maximization problem constrained to static power assignments can be solved in polynomial time.

**Proof.** For a constant power \( p_i \), the lifetime of node \( i \) is \( t_i = c_i/p_i \), a decreasing function of \( p_i \). Take as the \( f_i \)-value at node \( i \) the negative of its lifetime. This is clearly a monotonically increasing function of the power \( p_i \). Moreover, maximizing the lifetime \( t_i \) is equal to minimizing \( f_i \). The situation of Lemma 1 corresponds to all nodes being alive for exactly the same time and then running out of energy simultaneously.

3. DYNAMIC POWER ASSIGNMENTS

We now turn our attention to the more difficult task of multicast time maximization using dynamic power assignments. For simplicity, we assume that the possible control times (discontinuities) for the power assignment are quantized to occur at intervals of one time unit.\(^1\)

\(^1\)While this is a natural assumption, it remains a possibility that one can achieve more efficient power assignments by applying arbitrarily fine-grained controls.
We first give a direct combinatorial proof showing that the multicast time maximization problem under quantized dynamic power assignments is NP-hard. For this we need the following lemma, which is of interest in itself. Note that the term “Steiner tree packing” as used in the literature, e.g. in [3], refers to a more general problem that could more appropriately be called Steiner forest packing: the set of critical nodes to be included in the Steiner tree are predetermined, but may be different for each tree.

**Definition 11.** [11, p. 221] The 3-dimensional matching problem (3DM) is the following. Given a set \( T \subseteq X \times Y \times Z \), where \( |X| = |Y| = |Z| = q \), find a matching \( M \) for \( T \), i.e., a subset \( M \subseteq T \) such that \(|M| = q \) and no two distinct elements in \( M \) agree in any coordinate.

**Definition 12.** The node-disjoint (edge-disjoint) Steiner tree packing problem is the following. Given an undirected graph \( G = (V, E) \), a set of critical nodes \( W \subseteq V \), and a positive integer \( N \), decide whether there exist at least \( N \) pairwise node-disjoint (edge-disjoint) Steiner trees. A trivial upper bound for the number of node-disjoint subgraphs in \( G \) is \(|V| + 1 \). Hence, the problem is in NP because a non-deterministic algorithm can guess \( N \) subgraphs of \( G \) and check in deterministic polynomial time that these subgraphs form a collection of node-disjoint Steiner trees for \( W \).

We show completeness by transformation from 3DM. Let \( T \subseteq X \times Y \times Z \) be an instance of 3DM with \(|X| = |Y| = |Z| = q \) and \(|T| = m \). Without loss of generality we may assume that \( m \geq q \) and that the sets \( X, Y, Z, T \) are pairwise disjoint. Denote the elements of \( X \) by \( x_1, \ldots, x_q \) and similarly for the other sets \( Y, Z, T \).

To make the proof more readable, we shall consider an example alongside the formal treatment. In the example, let \( X = \{x_1, x_2\}, Y = \{y_1, y_2\}, Z = \{z_1, z_2\} \) and \( T = \{t_1, t_2, t_3, t_4\} \), where

\[
\begin{align*}
t_1 &= (x_1, y_1, z_1), & t_2 &= (x_1, y_2, z_2), \\
t_3 &= (x_2, y_1, z_2), & t_4 &= (x_2, y_2, z_1).
\end{align*}
\]

We construct the input graph \( G = (V, E) \) and the set of critical nodes for the Steiner tree packing instance from the 3DM instance in three steps. In the first step, let the node set of the input graph be \( V = X \cup Y \cup Z \cup T \). All of these nodes are noncritical. Now, for every \( \ell = 1, \ldots, m \), we add the three edges \( \{x_\ell, t_\ell\}, \{y_\ell, t_\ell\}, \{z_\ell, t_\ell\} \) to the graph, where \( x_\ell, y_\ell, z_\ell \) are determined from \( t_\ell = (x_\ell, y_\ell, z_\ell) \). The graph resulting from the example instance after the first step is depicted below.

In the second step, we insert four critical nodes \( x', y', z', t' \) into the graph, and join these by an edge to all of the nodes in the respective sets \( X, Y, Z, T \); that is, \( x' \) is adjacent to all the nodes in \( X \), \( y' \) is adjacent to all the nodes in \( Y \), and so on.

In the third step, we insert \( m - q \) noncritical nodes \( P = \{p_1, \ldots, p_{m-q}\} \) into the graph, and join each of these by an edge to the critical nodes \( x', y', z' \) and to all of the nodes in \( T \).

This completes the description of \( G \). The transformation is clearly computable in polynomial time. We now show that \( G \) contains \( m \) pairwise node-disjoint Steiner trees if and only if there exists a matching \( M \) for \( T \).

We start with the "if" direction. Suppose \( M = \{t_{i_1}, \ldots, t_{i_q}\} \subseteq T \) is a matching for \( T \). We can construct \( q \) pairwise node-disjoint Steiner trees in \( G \) by taking for each \( s = 1, \ldots, q \) the subgraph induced by the nodes \( x', y', z', t', t_s \), \( x_s, y_s, z_s \), where \( x_s, y_s, z_s \) are determined from \( t_s = (x_s, y_s, z_s) \). The remaining \( m - q \) trees can be constructed by taking the subgraph induced by \( x', y', z', t' \), and any two nodes \( p_i \) and \( t_s \) not used so far.

It remains to prove the "only if" direction. Suppose we are given \( m \) node-disjoint Steiner trees in \( G \). Because each of the critical nodes \( x', y', z', t' \) is adjacent to exactly \( m \) nodes, which are noncritical nodes, we must have that each of the Steiner trees contains exactly one node from \( T \) and either (i) exactly one node from \( P \) or (ii) exactly one node from each of the sets \( X, Y, Z \). We can construct a matching for \( T \) from the nodes \( t \in T \) in the \( q \) trees in which case (ii) occurs.

**Theorem 2.** The multicast time maximization problem under quantized dynamic power assignments is NP-hard.

**Proof.** We transform an instance \( G, W, N \) of the node-disjoint Steiner tree packing problem into an instance of the multicast time maximization problem. Without loss of generality we may assume that the critical nodes \( W \) are pairwise nonadjacent in \( G \). First, replace every undirected edge \( \{i, j\} \) in \( G \) with two directed edges \( (i, j), (j, i) \) to obtain a directed graph. Then, put \( d_{ij} = 1 \) if \( (i, j) \) is an edge; otherwise \( d_{ij} = \infty \). Select (arbitrarily) one of the critical
node $v$ for every $k$ with information later, we color all the edges $e_k$ they form a 
instance into a Steiner tree only if the transformed instance contains $v$
information for each noncritical node
Clearly, the above proof of Theorem 2 does not apply to the
general case where the discontinuities in the power assignments are not quantized to occur at intervals of one time unit.
We now turn to study the approximability of the multicast time maximization problem. We show that the multicast time maximization problem is hard for the complexity class APX, which consists of all NP optimization problems that admit a polynomial time approximation algorithm that achieves a constant performance ratio (see [2]). In particular, this implies that the multicast time maximization problem under quantized dynamic power assignments does not admit a polynomial time approximation scheme (PTAS) unless $P = NP$.
We base our APX-hardness proof on the following result. 2

Theorem 3. [13, Corollary 4.3] The edge-disjoint Steiner tree packing problem is APX-hard.

Lemma 3. There exists an approximation preserving polynomial time reduction from the edge-disjoint Steiner tree packing problem to the node-disjoint Steiner tree packing problem.

Proof. Let $G = (V, E)$ be an instance of the edge-disjoint Steiner tree packing problem, where $W \subseteq V$ is the set of critical nodes. We transform this instance into an instance of the node-disjoint Steiner tree packing problem in two steps.

In the first step, we perform the following local transformation for each noncritical node $v \in V \setminus W$. Let $v$ be incident with $k$ edges $e_1, \ldots, e_k$. We remove $v$ from the graph and replace it with $k$ new noncritical nodes $v_1, \ldots, v_k$ so that, for every $i = 1, \ldots, k$, the edge $e_i$ becomes incident with the node $v_i$. We then connect the nodes $v_i$ pairwise so that they form a $k$-clique. For purposes of analyzing the reduction later, we color all the edges $e_i$ blue and the edges in the $k$-clique green.

In the second step, we subdivide each edge joining two critical nodes into two edges by inserting a new noncritical node in the middle; these two edges are colored red. This completes the description of the transformation, which is clearly computable in polynomial time. We claim that the input instance contains $N$ edge-disjoint Steiner trees if and only if the transformed instance contains $N$ node-disjoint Steiner trees.

We can transform a Steiner tree $T$ in the edge-disjoint instance into a Steiner tree $T'$ in the node-disjoint instance as follows. First, for each edge in $T$, we include the corresponding blue edge (or both of the corresponding red edges) into $T'$. We then connect the blue edges into each other by a minimal number of green edges. This transformation maps edge-disjoint trees to node-disjoint trees because distinct blue edges cannot have a common noncritical endpoint.

Conversely, we can transform a Steiner tree $T'$ in the node-disjoint instance into a Steiner tree $T$ in the edge-disjoint instance by taking all the blue and red edges in $T'$ and inserting the corresponding edges into $T$. There is one exception: if only one red edge from a pair of red edges occurs in $T'$, then we do not insert the corresponding edge into $T$ because this would create a cycle. This transformation maps node-disjoint trees map to edge-disjoint trees because each blue edge occurs in at most one tree in a collection of node-disjoint trees. Furthermore, the red edges from a pair of red edges cannot occur in different trees of a node-disjoint collection because the edges share a noncritical endpoint.

Combining Theorem 3 and the reduction in Lemma 3, we obtain:

Corollary 2. The node-disjoint Steiner tree packing problem is APX-hard.

Corollary 3. The multicast time maximization problem under quantized dynamic power assignments is APX-hard.

Proof. The transformation from node-disjoint Steiner tree packing into multicast time maximization in the proof of Theorem 2 is approximation preserving.

It remains an open problem whether the node-disjoint Steiner tree packing problem or the multicast time maximization problem belong to the class APX, i.e. possess polynomial time approximation algorithms with a constant performance ratio.

4. ALGORITHMS FOR THE DYNAMIC POWER ASSIGNMENT PROBLEM

We present two randomized algorithms for the multicast time maximization problem under dynamic power assignments, and a method for bounding the multicast connection time from above.

The first algorithm is based on the static algorithm presented in Section 2. In the static solution, all nodes will run out of energy simultaneously. Our algorithm RND-GREEDY (see box) reduces node powers in a random order to save energy for further multicast trees. As a randomized algorithm, it produces varying results, so it is useful to iterate it several times and choose the best solution found.

Although this relatively simple algorithm routinely gives dynamic solutions with multicast times that are 2 to 4 times the static one, even better performance can be obtained. For this, the dynamic power assignment problem is split into two essentially separate subproblems: sampling a good collection from the set of all viable static power assignments, and scheduling, i.e. deciding for how long each assignment in the collection should be used. By a viable static power assignment we mean one that fulfills the multicast property. We shall tackle the scheduling problem first.

A collection of $m$ viable static power assignments can be concisely represented as columns of an $n \times m$ matrix $P$, where $P_{ij}$ indicates the power of node $i$ in the $j$th assignment. Now scheduling is a linear programming (LP) problem: Given $P$ and an energy supply vector $e = (e_1, \ldots, e_m)$, find a schedule vector $x = (x_1, \ldots, x_m)$ that maximizes $\sum x_j$ (total multicast time), subject to $Px \leq e$ (energy constraints at all nodes), and $x_j \geq 0$ for all $j$. The dynamic power assignment is then as follows: for each $j = 1, \ldots, m$, run the $j$th power assignment for $x_j$ time units. Note that

\begin{align*}
\text{Proof.}
\end{align*}
Algorithm RNDGREEDY

1. Choose powers \( p = (p_1, \ldots, p_n) \) with the polynomial time algorithm given by Corollary 1.
2. Repeat until the source has zero energy:
   2.1. For all nodes \( i \) in random order: Turn the power \( p_i \) as low as possible without breaking the multicast connectivity from the source to all sinks. (Perform a binary search over the values \( d_{ij} \) that lie between 0 and the \( p_i \) given by Step 1; for each value, check connectivity with depth-first search.)
   2.2. Run the network with this power assignment until some node runs out of energy. Update the energy supplies of all nodes according to the consumption.

Algorithm LPSCHEDULE

1. Initialize: \( P = [], e_{red} \leftarrow e \) (the true energy supply of each node).
2. Repeat for a number of iterations:
   2.1. Sampling: Generate a set of power assignments with RNDGREEDY, using reduced energy supply \( e_{red} \). Append all of them as new columns to \( P \).
   2.2. Scheduling: \( x \leftarrow \) optimum schedule for \( P \) with full energy supply \( e \) (solve as LP).
   2.3. Consumption: \( w \leftarrow P x \).
   2.4. Energy for next iteration: \( r \leftarrow \) uniform random number in \([0, 1]\), \( e_{red} \leftarrow e - r \cdot w \).
3. Return the dynamic assignment given by \( P \) and \( x \).

here we treat time as a continuous quantity and not quantized to unit-length intervals as in Section 3.

This is a natural formulation of the dynamic power assignment problem in the following sense: If \( P \) contains all the viable static power assignments for a given network, scheduling will give the optimal dynamic assignment. However, for networks of nontrivial size, this is out of question, as the number of viable assignments is exponential in \( n \). But for relatively large collections (e.g. \( m \approx 1000 \)), the LP scheduling problem can be efficiently solved.

For sampling, simply generating a collection of individually good static power assignments is not enough. As pointed out in [18], such a collection is not necessarily very good for the dynamic power assignment. Instead, we would like to find static power assignments that exploit the energy supplies of different subsets of all nodes. This is the underlying idea in our second algorithm.

The algorithm LPSCHEDULE (see box) generates a collection \( P \) iteratively. New power assignments are obtained using RNDGREEDY and accumulated. Power assignments are never discarded from the collection; we rely on the scheduler to allocate zero time for inferior assignments. Scheduling is computationally cheap, as long as \( P \) remains reasonable in size. In our experiments, most of the running time of LPSCHEDULE is spent in generating new viable assignments using RNDGREEDY.

The trick is that the new assignments are encouraged to exploit other nodes than those heavily used by the current \( P \), by telling RNDGREEDY that only a part of the true energy supply is available. The strength of this encouragement varies according to a random factor \( r \in [0, 1] \). Note that for \( r = 0 \), Step 2.1 of LPSCHEDULE reduces into another independent iteration of RNDGREEDY for the original problem. On the other hand, for \( r = 1 \), the new assignments are restricted to “leftover” nodes. Intermediate values of \( r \) provide a “soft” penalty for competing over energy with the current solution.\(^3\)

Finally, we attack the problem from the opposite direc-

\(^3\)In fact, the randomization of \( r \) is not crucial, and a fixed value of e.g. 0.25 or 0.5 works quite well.

and formulate upper bounds for the dynamic multicast time in a given network. Such an upper bound can be used e.g. for assessing the quality of the solutions found by the algorithms.

Let the source node be 1, and its nearest neighbor 2. A trivial upper bound follows from the fact that the source has to reach at least its nearest neighbour, so \( p_1 \geq d_{12} \). No matter how we choose the powers of other nodes, the multicast time cannot exceed \( \frac{d_{12}}{p_1} \).

This idea can be extended to an arbitrary cut \( C \subset V \) that contains the source, but not all the sinks. At all times, the power levels inside \( C \) must be large enough so that the transmission graph contains at least one path from the source to some node outside \( C \). For small \( C \), we can enumerate all such paths and find a time-maximizing schedule between them by solving an LP similar to that described above.

Experiments suggest that quite small cuts can give tight bounds on the dynamic multicast time. Indeed, for multicast trees in a large network, the bottleneck is in escaping a small neighbourhood of the source, constrained by its limited energy supply. After that, the large number of nodes in the rest of the network provides an abundance of alternative routes to the sinks.

5. EXPERIMENTS

The algorithms were implemented in MATLAB 6.5, using the Optimization Toolbox 2.2 for LP solving. Experiments were run on a workstation with a 1333 MHz AMD Athlon processor.

In the experiments, we place 100 nodes (1 source, 4 sinks) at random, uniformly distributed in the unit square. Power thresholds are computed as \( d_{ij} \approx r_{ij}^2 \). All nodes are given 1 unit of energy. An upper bound for the multicast time is computed using a cut consisting of the source and its six nearest neighbours.

An example network is shown in Fig. 1. The maximum static multicast time for this network is 59.7 units. Both dynamic algorithms were run for 100 iterations. Performance of the algorithms is compared in Fig. 2. The multicast time given by RNDGREEDY quickly reached about 3.3 times
the static solution, but remained constant after the first 39 iterations. Algorithm LPSCHEDULE generated 695 different power assignments, two of which are shown in detail in Fig. 1. The solution gives an improvement ratio of 3.96 over the static multicast time and reaches 92.6% of the upper bound.

Figure 2: Progress of RNDGREEDY and LPSCHEDULE for the Fig. 1 network. (Total computation time 126 and 199 CPU seconds, respectively.)

To gain some idea of the performance of the algorithms in general, we next generated 50 random networks. For each network, we ran the static algorithm and the two dynamic ones, for 50 iterations each. The results are shown in Fig. 3. It can be seen that the dynamic power assignments gave multicast times which in general were about three times the static ones. Also, LPSCHEDULE often finds near-optimal solutions; in fact, in more than half of the networks (28 out of 50) the solution is within 1% of the upper bound. Note that in one case the network was extreme in the sense that even the static solution reaches the upper bound, i.e., gives an optimal solution.

The results show that dynamic power assignments can be clearly superior to static ones and that our algorithm LPSCHEDULE can achieve very good solutions.

6. CONCLUSION

A large amount of current work has been directed towards energy minimization in wireless ad hoc networks, with the underlying goal of maximizing the lifetime. Our approach has been to optimize the multicast time directly. We have provided an optimal polynomial time algorithm for determining the maximum multicast time under the constraint that the transmission powers of all nodes are set to fixed values at the outset, and two approximation algorithms for finding good power assignment schedules when the powers at the nodes can be dynamically adjusted during operation. In fact, for small networks, the optimal dynamic multicast time can be determined using linear programming as long as the total number of all viable static power assignments is small enough, i.e., less than a few thousand.

We have also proved that finding optimal power assignment schedules in the dynamic case is NP-hard and thus not likely to be exactly solvable by a polynomial-time algorithm, when the time is quantized instead of real-valued. Furthermore, this problem is not likely to have a polynomial time approximation scheme. Whether the problem admits a constant performance ratio approximation algorithm remains an open problem. Also the computational complexity under real-valued time remains open.

We have assumed that the nodes are immobile, as in e.g.
sensor networks. In addition, our power assignment algorithms require some degree of centralized control of the network. Either all the nodes need to be aware of the network’s complete initial energy state, or they need to communicate with some central coordinating node. The problems of node mobility and distributed approximate optimization of the power assignment schedules remain to be studied.

7. REFERENCES


