Maximum-Lifetime Routing in Wireless Networks: System Optimization & Game-Theoretic Perspectives

Liane Lewin-Eytan * Joseph (Seffi) Naor † Ariel Orda ‡

August 2006

Abstract

A major problem in wireless networks is how to route traffic so as to maximize the lifetime of the transmission. We address a two-way multicast problem, where a root wishes to transmit data to a subset of nodes (called also terminals), which can also respond to the root. In addition, we consider the anycast problem, where there is a subset of nodes that can communicate with one another. We consider both directional antennas and omnidirectional antennas.

First, we note that, for omnidirectional antennas, an optimal solution can be computed in polynomial time. Next, for directional antennas, we observe that computing an optimal solution is an NP-hard problem. Accordingly, we provide a polynomial time algorithm that finds a 2-approximate solution for the case of uniform transmission power levels. For the much harder case, where there are different transmission power levels, we provide an $O(\log^2 n)$ approximation algorithm for the general problem, and an $O(\log n)$ approximation algorithm for the special case where the set of terminals equals the set of all nodes.

Next, we consider the corresponding noncooperative game scenarios, and prove that, by following the natural game course, users converge to a Nash equilibrium. For directional antennas, we show that the price of anarchy is unbounded. On the other hand, we show that the price of stability, where the best Nash equilibrium is considered, is 1; hence, optimal (networkwide) performance can be achieved if the initial configuration can be imposed on the players. For omnidirectional antennas, we show that, if the players join the game sequentially, the Nash equilibrium is (networkwide) optimal.

1 Introduction

In recent years, stationary wireless networks were extensively studied due to their potential applications in the civil and military domains, e.g., for the implementation of sensor networks [8, 13]. Since the amount of energy that can be stored in the nodes is limited, energy efficiency is a crucial aspect in the establishment of such networks. Thus, it is essential to develop protocols that optimize the overall energy utilization of the network, in order to maximize its capability to function for the longest possible time.

Accordingly, a wide variety of energy-efficiency problems have been addressed. One approach is to find a routing scheme that minimizes the total power consumed by all nodes, e.g., [4, 16, 21, 25]. However, such an approach does not guarantee that the transmission lifetime is maximized, i.e., the time until the battery of a transmitting node drains out. Indeed, the latter mainly depends on

---

*Technion-Israel Institute of Technology, Department of Electrical Engineering, Haifa 32000, Israel. Email: liane@tx.technion.ac.il

†Microsoft Research, Redmond, WA 98052. On leave from the Technion-Israel Institute of Technology, Computer Science Department, Haifa 32000, Israel. Email: naor@cs.technion.ac.il

‡Technion-Israel Institute of Technology, Department of Electrical Engineering, Haifa 32000, Israel. Email: ariel@ee.technion.ac.il
the residual energy stored in the nodes’ batteries, while the previous approach solely refers to the instantaneous energy consumption.

In this work, we study the lifetime maximization problem in wireless networks, both for a single-recipient environment, where the nodes communicate using directional antennas, and for a multi-recipients environment, where the nodes communicate using omnidirectional antennas. Specifically, for both environments, we consider a two-way multicast problem, and an anycast problem. In the two-way multicast problem, there is a special node, the root, that wishes to transmit data to a subset of nodes (called also terminals), which can also respond to the root. The anycast problem consists of a subset of nodes that can communicate with one another.

Our work consists of two parts. In the first part, we focus on system optimization issues for the single-recipient environment. Based on theoretical analysis, we provide approximation algorithms with proven performance for problems that are found to be computationally hard. In the second part of our work, we focus on game-theoretic perspectives of the respective lifetime maximization problems. More specifically, we investigate wireless games with selfish non-cooperative players, both for the single-recipient (directional antennas) and multi-recipients (omnidirectional antennas) environments.

The problem of lifetime maximization has been studied in previous works, mainly for the standard environment of omnidirectional antennas [6, 7, 9, 15, 18]. These studies mostly focused on multicast and broadcast. In [15] and [18], two problem classes were considered, namely: single-topology routing schemes, where the transmission is routed on a single routing topology (i.e., a routing tree), and multi-topology routing schemes, where there is a set of routing topologies (routing trees) that are employed sequentially, in order to prolong the transmission duration time. The multi-topology scheme adds an additional degree of freedom to the solution, and therefore can lead to a configuration with longer lifetime. However, it is typically far more complex in terms of computational solution as well as implementation. In the present study we focus on the standard case of a single routing topology.

Specifically, the system optimization problem we consider consists of finding a routing topology that maximizes the lifetime of the transmission. That is, our goal is to find a Steiner tree spanning the terminals with maximum lifetime. For omnidirectional antennas, an optimal solution for this problem can be easily solved in polynomial time, as can readily be derived from [18]. On the other hand, for directional antennas, finding such a tree is an NP-hard problem, even in the case where all nodes have the same initial battery and have a single transmission power level.

The employment of directional antennas has been shown to possess significant advantages over the traditional omnidirectional transmission, in particular in terms of increasing network capacity with spatial reuse and mitigating the interference and contention among neighboring nodes [14]. Yet, only limited work has been done on routing protocols using directional antennas [5]. The few studies on energy-efficient routing with directional antennas focused on heuristics that were evaluated by simulations [24, 26]. The algorithmic complexity of the lifetime problem with directional antennas was first investigated in [18], where it was shown that both the single- and multi-topology multicast problems are NP-hard.

Focusing on the environment of directional antennas, we present near-optimal solutions for the maximum lifetime problem, and provide approximation algorithms with proven performance. We first consider the special case where each node has a different initial energy, but all the nodes transmit at the same transmission power level. The problem of finding the maximum lifetime Steiner tree in this case is closely related to the minimum degree Steiner tree problem, where the goal is to find a Steiner tree whose maximum degree is minimum among all feasible Steiner trees. This problem has been previously investigated [2, 10, 11, 20] and a Steiner tree whose maximum degree is within one from optimal can be found [11]. Using the technique presented in [11], we provide a polynomial time algorithm that finds a 2-approximate solution, for the case of uniform transmission power levels. For the more general, and much harder case, where there are different transmission power levels, we provide a polynomial time \( O(\log^2 n) \) approximation algorithm for the Steiner tree case, and an
$O(\log n)$ approximation algorithm for the spanning tree case, where $n$ denotes the number of nodes.

The approximate solutions we provide use a centralized approach, which is often practical in stationary (or quasi-stationary) topologies. However, a wireless network is often characterized by a distributed, dynamic, self-organizing architecture. Thus, the scenario where each node in the network is capable of independently adapting its operation based on the current environment, is intriguing. Moreover, ad hoc wireless networks are often characterized by nodes with self-optimizing (selfish) goals. In order to investigate these settings, we use game-theoretic tools to model and analyze the interaction among independent selfish nodes, and the strategic scenarios that arise in such systems. Recently, game theory was also applied to wireless communication (see [23] for a survey).

Accordingly, in the second part of our work we investigate the game-theoretic perspective of the maximum lifetime problems. We define noncooperative game scenarios with selfish non-cooperative players, both for the environments of directional and omnidirectional antennas. More specifically, we define a two-way multicast game, where there is a special source node and each player is interested in connecting to the source by making a routing decision that maximizes the lifetime of its path to the source. We also consider an anycast game, where a player connects to an existing configuration by making a routing decision that maximizes the lifetime of its paths to the other terminals. In both games, the mutual influence of the players arises from using common nodes.

In both the directional and omnidirectional environments, we prove that, by following the natural game course, users converge to a Nash equilibrium. This equilibrium raises several natural questions. We focus in this paper on the inefficiency resulting from the selfish behavior of the players, that is, we investigate how the selfish behavior of individual nodes may affect the performance of the network as a whole. We quantify this inefficiency through the ratio between the lifetime of the worst possible Nash equilibrium configuration and the lifetime of an optimal solution. In keeping with common terminology [17, 19], this ratio is called the *price of anarchy* (PoA) and it quantifies the “penalty” incurred by lack of cooperation (or coordination) between the players in a non-cooperative game.

For both the directional and omnidirectional environments, we show that the price of anarchy is unbounded. Motivated by these results, we consider the *price of stability* [3], which is the ratio between the lifetime of the best Nash equilibrium and the lifetime of an optimal solution. This notion defines the outcomes of scenarios in the “middle ground” between centrally-enforced solutions and non-cooperative games. In such scenarios, there is an underlying protocol that suggests an initial configuration to all players, which can either accept it, or try to defect from it. However, assuming the solution proposed is stable (i.e., a Nash equilibrium), no player would have an incentive to defect from it. We show that the price of stability is 1; hence, optimal (networkwide) performance can be achieved by solely requiring that the initial configuration be imposed on the players.

However, it is not clear whether such a (best) Nash equilibrium can be computed in polynomial time. Therefore, we suggest a mechanism in which a central authority starts the process by first computing the best *approximately* optimal solution computable in polynomial time, and then allows the users to follow the natural game course. We show that the lifetime of the resulting Nash equilibrium cannot be worse than the lifetime of the initial configuration. The approximation algorithms provided in the first part of our work thus get further importance in the light of this mechanism.

Nonetheless, there are several situations in which having such a central trusted authority, which can force a game to start from a specific configuration, is expensive or infeasible. Moreover, not all the players might be available at the same time; in an online setting, players might arrive one by one to join the multicast or anycast services. Motivated by these considerations, we explore in this paper the following setting. Players first join the game sequentially starting from an “empty” configuration. Upon arrival, each player picks a path selfishly. Once reaching the solution constructed by the players joining one by one, the natural game course continues until a Nash equilibrium is reached. We show that the Nash equilibrium reached in such a setting is (networkwide) optimal for the omnidirectional case, while for the directional case, the price of anarchy of an online setting is not bounded.
Table 1 summarizes our main results.

<table>
<thead>
<tr>
<th>Max. Lifetime</th>
<th>System Optimization</th>
<th>Max. Lifetime</th>
<th>System Optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem</td>
<td>(Single-Recipient)</td>
<td>Game</td>
<td>(Multi-Recipients)</td>
</tr>
<tr>
<td>(Directional Antennas)</td>
<td></td>
<td>(Omnidirectional Antennas)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>b. Non-uniform trans. powers: Simple polynomial time optimal solution</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Special Case</td>
<td>N = V</td>
<td>O(log n)-approx. algorithm</td>
<td>&amp; anycast solution</td>
</tr>
</tbody>
</table>

The rest of the paper is organized as follows. We first (Sections 2–5) consider the lifetime maximization problem in the single-recipient (directional antennas) environment. In Sections 2 and 3, we formally define the model and related problems, and describe the general technique we use for providing approximate solutions for the lifetime maximization problem. In Section 4, we consider the special case where all nodes transmit at the same transmission power level. Then, in Section 5, we consider the general case, where there are different transmission power levels. In Sections 6–8, we define and analyze the noncooperative game scenarios of the respective lifetime maximization problems, both for the environments of directional and omnidirectional antennas. Finally, conclusions are presented in Section 9.

2 The Single-Recipient Environment: Model & Problem Formulation

In this section, we formally define the network model for the single-recipient case. In the single-recipient wireless model, the nodes communicate using directional antennas, and a transmission can be received only by a single node. We assume static locations of wireless nodes. The wireless network is modeled by a graph $G = (V, E)$ (where $|V| = n$). Each node $v \in V$ has a finite battery with energy $b(v)$, and is able to transmit until it consumes all its energy. In case a sender $v$ wishes to transmit to a recipient $u$ over an edge $e = (v, u)$, the transmission power (energy per time unit) consumed by the sender is $p(e)$. As the transmission power $p(e)$ is a function of the topological distance between $v$ and $u$, it is also the transmission consumed by $u$, in case it wishes to transmit to $v$. An instance of the single-recipient wireless model is denoted by $G = (V, E, b, p)$.

We assume that, for each edge $e$, $p(e) \geq 1$. Thus, the lifetime $l(v)$ of node $v$ is the consumption time of its battery (energy), given the powers of the edges over which $v$ transmits:

$$l(v) = \frac{b(v)}{\sum_{(e=(v,u), v \text{ transmits to } u)} p(e)}.$$

The lifetime $l(p)$ of a path $p$ is the lifetime of the node with minimum lifetime in $p$. In practice, reception of data consumes energy as well; however, it does not depend on the transmission distance, and therefore does not make the related problems harder.
We first address a two-way multicast problem, where there is a special node \( r \in V \) called root (or source), that wishes to transmit data to a subset of \( k \) nodes \( N = \{t_1, t_2, \ldots, t_k\} \subseteq V \). In addition, each node \( v \in N \) can also respond, and transmit data to \( r \). The problem is to find a routing scheme that maximizes the time during which the transmission can be performed in both directions. That is, our goal is to maximize the lifetime of the routes between \( t_i, 1 \leq i \leq k \), and \( r \).

The second problem we consider is anycast, where there is a subset of nodes \( N = \{t_1, t_2, \ldots, t_k\} \subseteq V \), and each node \( v \in N \) can transmit data to the other nodes in \( N \). The problem is to find a routing scheme that maximizes the time during which the transmission between all nodes can be performed.

We define the lifetime of a solution \( T \) for both the multicast and anycast problems.

**Definition 1.** Given an instance \( G = (V, E, b, p) \), with a set \( N \) of target nodes \( N \subseteq V \) (and a root node \( r \in V \), in the multicast setting), the lifetime of a Steiner tree \( T \) spanning \( N \) (or \( N \cup r \)) is the minimum lifetime of any path in \( T \). That is, the lifetime of \( T \) is constrained by the node \( v \in T \) whose initial energy is consumed first, i.e., the node \( v \in T \) with minimum lifetime \( \sum_{e \in (v,u) \in T} P(e) \).

Note that as our setting supports a two-way communication, we consider the power of all edges adjacent to a node \( v \) in \( T \) when computing its lifetime.

Therefore, our goal is to compute a Steiner tree \( T \) spanning \( N \) (or \( N \cup r \), in the multicast setting) with maximum lifetime. In case \( N = V \), we wish to compute a maximum lifetime spanning tree of \( G \). Note that for both the multicast and anycast problems, the lifetime of the tree solution \( T \) is restricted by the node \( v \in T \) with minimum lifetime. Thus, the analysis of these problems is similar.

### 3 The Weighted Bounded Degree Steiner Tree Problem

We first note that the problem of finding a maximum lifetime spanning tree is NP-hard (and thus the hardness of the Steiner tree case follows directly). Consider the special case where all edges have the same transmission power and all nodes have the same initial energy. The lifetime of a spanning tree \( T \) is thus constrained by the node \( v \in T \) with maximum degree, which also defines the degree of \( T \). Given a graph \( G = (V, E) \), the optimal solution consists of a spanning tree whose maximum degree is the smallest among all spanning trees of \( G \). We call this solution the minimum-degree spanning tree (or Steiner tree, in case \( N \subseteq V \)). The minimum degree spanning tree problem can immediately be shown to be NP-hard by a reduction from the Hamiltonian Path problem. However, as shown in [11], there is a polynomial time algorithm that finds a spanning tree whose degree is within one from optimal. This algorithm can also be extended to the Steiner tree case (see [11]).

We present approximation algorithms for finding near-optimal solutions of several variants of the maximum-lifetime tree problem. We use the following general approach.

We first address a closely related problem, of finding a feasible tree in a graph where each node \( v \) has a degree bound \( B(v) \). The special case of this problem, where all degree bounds are equal, is another version of the minimum-degree Steiner tree problem, and is NP-complete [12]. We consider a weighted extension of this problem. Assume there is a weight function \( w : E \to R^+ \). Given a tree \( T \), the weighted degree \( \text{wdeg}_T(v) \) of a node \( v \in T \) is the sum of the weights of its adjacent edges in \( T \). The weighted bounded-degree Steiner tree problem (WBD-ST), is defined as follows.

**WBD-ST Problem:** Given a weighted graph \( G = (V, E, w) \), \( w : E \to R^+ \), a weighted degree bound \( B(v) \) for each node \( v \), and a subset of nodes \( N \subseteq V \), find a Steiner tree \( T \) spanning \( N \), such that for each node \( v \in T \), \( \text{wdeg}_T(v) \leq B(v) \).

In order to find an approximate solution for the maximum lifetime tree problem, we first give an approximate solution for the WBD-ST problem. Then, using this solution, we derive a respective approximations for the maximum lifetime tree problem, as stated by the following theorem.

**Theorem 1.** Assume there is a polynomial time algorithm \( A \), that either proves there is no feasible
solution for the WBD-ST problem, or finds a Steiner tree $T$ such that for each node $v \in T$, $\text{wdeg}_T(v) \leq \alpha \cdot B(v) + \beta$. Then, there exists a polynomial time algorithm that finds an $(\alpha + \beta)$-approximate solution for the maximum-lifetime Steiner tree problem.

**Proof.** Given an instance $G = (V, E, b, p)$, a source node $r$ and a subset of nodes $N \subseteq V$, assume there is Steiner tree $T$ spanning $N$ with a lifetime value of $\sigma$. Then, for each node $v \in T$ it holds that $rac{b(v)}{wdeg_T(v)} \geq \sigma$, where the weight of an edge $e$ is its transmission power $p(e)$. Therefore, the weighted degree $wdeg_T(v)$ of each node $v \in T$ is at most $\frac{b(v)}{\sigma}$. We perform a binary search over the possible lifetime values. For each lifetime value $\sigma$, we run algorithm $A$ on the instance $G = (V, E, p)$, where the weighted degree bound of each node $v$ is $B(v) = \frac{b(v)}{\sigma}$. In case algorithm $A$ proves there is no feasible solution for these degree bounds, we continue the binary search accordingly. We stop at the maximal value $\sigma^*$ for which algorithm $A$ finds an approximate solution. Clearly, $\sigma^*$ is an upper bound on the value of the maximum lifetime Steiner tree (as it was proved by $A$ that for each $\sigma > \sigma^*$ there is no feasible solution). Now, let us analyze the lifetime of the approximate solution $T^*$ computed by $A$. For each node $v \in T^*$ it holds that $wdeg_{T^*}(v) \leq \alpha \cdot \frac{b(v)}{\sigma} + \beta$. We thus get that

$$
\frac{b(v)}{\text{deg}(v)} \geq \frac{b(v)}{\alpha \cdot b(v)/\sigma^* + \beta} = \frac{\sigma^*}{\alpha + \beta} \geq \frac{\sigma^*}{\alpha + \beta}
$$

The last inequality follows from the fact that for each node $v \in T^*$, $B(v) = \frac{b(v)}{\sigma} \geq 1$. (As the weight of an edge is at least 1, all nodes $v$ for which $\frac{b(v)}{\sigma} < 1$ can be ignored). As (1) holds for each node $v \in T^*$, the lifetime of $T^*$ is at least $\frac{\sigma^*}{\alpha + \beta}$.

We denote by $b_{\text{min}}$ the minimum initial energy of any terminal, which is also the maximum possible lifetime, and by $RT(A)$, the running time of algorithm $A$. The next claim follows directly from the proof of Theorem 1.

**Claim 1.** The running time needed in order to find a $(\alpha + \beta)$-approximate solution for the maximum-lifetime Steiner tree problem is $RT(A) \cdot \log b_{\text{min}}$.

# 4 Uniform Power Transmissions, Non-Uniform Batteries

We first consider a simpler case of the maximum lifetime Steiner tree problem, where each node $v$ has a different initial energy $b(v)$, but all the nodes transmit at the same transmission power level $P$. Thus, the neighbors of each node $v \in V$ are the nodes to which $v$ can transmit using a transmission power of $P$. We give a 2-approximation for the maximum lifetime Steiner tree in this case.

As all edges have the same power, the weighted degree of a node $v$ with respect to a tree $T$ can be reduced to the number of its adjacent edges in $T$, that is, to the degree of $v$ in $T$, denoted by $\text{deg}_T(v)$.

We consider the unweighted version of the WBD-ST problem, called the bounded-degree Steiner tree problem (BD-ST), defined as follows.

**BD-ST Problem:** Given a graph $G = (V, E)$, a weighted degree bound $B(v)$ for each node $v$, and a subset of nodes $N \subseteq V$, find a Steiner tree $T$ spanning $N$, such that for each node $v \in T$, $\text{deg}_T(v) \leq B(v)$.

We present a polynomial time algorithm for the BD-ST problem, that either proves there is no feasible solution for our given instance, or computes a Steiner tree $T$ spanning $N$ such that for each node $v \in T$, $\text{deg}_T(v) \leq B(v) + 1$.

We turn to introduce the approximation algorithm for the BD-ST problem. First, we focus on the case where the algorithm proves that the given instance is infeasible. Next, we describe the other case, where the algorithm stops after computing an approximate solution. Finally, we derive the
approximation for the maximum lifetime problem, using the approximate solution provided for the BD-ST problem.

4.1 Infeasibility Proof

We first concentrate on the case where the BD-ST algorithm proves that there is no feasible solution for our given instance. The algorithm uses the notion of feasibility constraint, that holds for every instance admitting a feasible solution. In case there is no feasible solution, the algorithm finds a witness (defined later) that contradicts the feasibility constraint. This witness constitutes the infeasibility proof provided by the algorithm. The feasibility constraint of a given instance is defined by Lemma 1, as follows.

**Lemma 1.** Let \( W \subseteq V \) be a subset of the nodes in \( G \). Suppose the removal of \( W \) from \( G \) disconnects \( G \) into a set \( C \) of connected components. Let \( t \) the number of components in \( C \) that contain nodes of \( N \). If there exists a feasible solution with respect to the given degree bounds \( B(v) \), then for each such sets \( W \) and \( C \) there exists a feasible tree \( T \) such that

\[
\sum_{v \in W} B(v) \geq \sum_{v \in W} \deg_T(v) \geq \left( |W| + t - 1 \right) + |T[W]| \tag{2}
\]

where \( T[W] \) denotes the edges of \( T \) with both endpoints adjacent to nodes in \( W \).

**Proof.** Inequality 2 can also be written in the following way:

\[
B_{avg}^W \geq \frac{\sum_{v \in W} \deg_{avg}^W}{|W|} \geq \frac{|W| + t - 1}{|W|} + \delta \tag{3}
\]

Where \( B_{avg}^W = \frac{\sum_{v \in W} B(v)}{|W|} \) is the average degree bound of the nodes in \( W \), and \( \deg_{avg}^W = \frac{\sum_{v \in W} \deg_T(v)}{|W|} \) is the average degree of the nodes in \( W \) with respect to solution \( T \). Note that \( |T[W]| \leq |W| - 1 \), and thus \( \delta \in [0, 1) \).

The left-hand side of inequality 3 follows as \( T \) is a feasible solution. Now, suppose \( T \) uses only a subset \( W' \) of the nodes in \( W \) \( (W' \subseteq W) \). Thus, \( T \) contains at least \( |W'| + t - 1 \) edges connecting the nodes in \( W' \) and the \( t \) components. Each such edge is incident to at least one node in \( W' \), and therefore contributes at least one unit to the sum of the degrees of the nodes in \( W \) (the degree of a node in \( W \setminus W' \) is counted as 0). In addition, each edge in \( T \) connecting two nodes in \( W \) is counted twice when summing up the degrees of the nodes in \( W \), leading to the additive factor of \( |T[W]| \). Thus, the average degree of the nodes in \( W \) is at least \( \frac{|W'| + t - 1}{|W'|} + \delta \), which is greater or equal to \( \frac{|W| + t - 1}{|W|} + \delta \), leading to the right-hand side of inequality 3. Since the inequality is independent of \( W' \), it is true for all feasible Steiner trees spanning \( N \).

Consider the case where there is no Steiner tree \( T \) spanning \( N \) where for each node \( v \in T \), \( \deg_T(v) \leq B(v) \). In that case, the algorithm finds a witness that constitutes an infeasibility proof, as provided by the following theorem.

**Theorem 2.** Let \( G = (V, E) \) be a given graph and let \( N \subseteq V \) be an arbitrary subset of nodes. Let \( T \) be a Steiner tree spanning \( N \), with the property that every leaf of \( T \) belongs to \( N \). Let \( W \) be a set of nodes \( v \) with degree \( \deg_T(v) \geq B(v) + 1 \), such that there is at least one node \( w \in W \) with degree strictly higher than \( B(w) + 1 \). Let \( W \) be removed from the graph, breaking the tree \( T \) into a set of connected components \( C \). Suppose \( G \) satisfies the condition that there are no paths through nodes of \( V \setminus W \) between different components in \( C \). Then, the given instance is infeasible.
In this case, we also say that the set \( W \) constitutes a witness set for the infeasibility of the given instance.

**Proof.** We first modify \( T \), obtaining a Steiner tree \( T' \) in which the sum of the degrees of the nodes in \( W \) is no less. Consider each component \( C^i \in C \) that does not contain nodes of \( N \). Suppose there are \( \ell \) edges in \( T \) between \( C^i \) and \( W \). We remove these edges, and connect their endpoints in \( W \) by adding \( \ell - 1 \) pseudo-edges. Note that by doing this, we preserve connectivity and acyclicity. By removing the \( \ell \) edges, the sum of the degrees of the nodes in \( W \) decreases by \( \ell \), whereas by adding \( \ell - 1 \) pseudo-edges, the sum increases by \( 2(\ell - 1) \). In addition, \( \ell > 1 \), otherwise \( C^i \) would be a leaf component of \( T \) that does not contain any node from \( N \). Thus, \( 2(\ell - 1) \geq \ell \), and the sum of the degrees of the nodes in \( W \) after the change can only increase.

We denote by \( t \) the number of connected components in \( C \) that contain nodes of \( N \). There are no paths through nodes of \( V \setminus W \) between different components in \( C \). Thus, in order to span the nodes in \( W \) and the \( t \) connected components, there are exactly \( (|W| + t - 1) \) edges in \( T' \) that are incident to at least one node in \( W \). Note that each such edge contributes at least one unit to the sum of the degrees of the nodes in \( W \). In addition, each edge in \( T' \) connecting two nodes in \( W \) is counted twice when summing up the degrees of the nodes in \( W \), leading to the following equality.

\[
\sum_{v \in W} \text{deg}_T(v) \leq \sum_{v \in W} \text{deg}_{T'}(v) = (|W| + t - 1) + |T'[W]|. \tag{4}
\]

For each node \( v \in W \) it holds that \( \text{deg}_T(v) \geq B(v) + 1 \), and there is at least one node \( w \in W \) with \( \text{deg}_T(w) > B(v) + 1 \). Thus, we can deduce the following inequality.

\[
\sum_{v \in W} B(v) + 1 < (|W| + t - 1) + |T'[W]|. \tag{5}
\]

That is

\[
B^W_{\text{avg}} < \frac{|W| + t - 1}{|W|} + \delta' - 1, \tag{6}
\]

where \( \delta' \in [0, 1) \). Therefore, equation 3 does not hold for the set \( W \). We say that the set \( W \) is a witness to the fact that there is no feasible solution for our instance with respect to the given degree bounds.

**4.2 The BD-ST Approximation Algorithm**

Our algorithm builds on the technique described in [11], which considers a different version of the problem in which there are no specified degree bounds. Given a graph \( G = (V, E) \), and a set of nodes \( N \subseteq V \), [11] considers the problem of finding a Steiner tree spanning \( N \) in which the maximum degree (also called the degree of the tree) is minimized. The algorithm presented in [11] finds a Steiner tree spanning \( N \), whose degree is within one from optimal. We adapt this algorithm for the case of non-uniform degree bounds.

**4.2.1 Spanning Tree Case**

We first describe the algorithm for the spanning tree case, where \( N = V \). The idea of the algorithm is as follows. We begin from an arbitrary spanning tree of \( G \). The algorithm runs in phases, where in each phase, we reach a temporary solution \( T \). If for each node \( v \) it holds that \( \text{deg}_T(v) \leq B(v) + 1 \), we output \( T \) as our approximate solution. Otherwise, we try to reduce the degree of a node \( v \) for which
improvement is reached after a single iteration. The algorithm then proceeds with the next phase, that consists of adding one node iterations within phase i. Assume $T_i$ is the spanning tree built by the algorithm when reaching phase $i$. We choose $W_{i,1}$ as the set of all nodes $v$ with degree $\text{deg}_T(v) \geq B(v) + 1$. Note that there is at least one node $w \in W_{i,1}$ with degree strictly higher than $B(w) + 1$. We denote by $C_{i,1} = C^1_{i,1}, \ldots, C^m_{i,1}$ the connected components formed by deleting $W_{i,1}$ from the tree $T_i$. If there are no edges in $G$ between components of $C_{i,1}$, the algorithm stops. Otherwise, let $e = (u, v)$ be an edge connecting two different components of $C_{i,1}$. Note that the degrees of nodes $u$ and $v$ are at most $B(u)$ and $B(v)$ respectively. We consider adding $e$ to $T_i$, and observe the unique cycle $L$ that would be generated. In case there is a node $w \in L$ with degree higher than $B(w) + 1$, we improve $T_i$ by performing a swap operation that consists of adding $e$ to $T$ and deleting one of the edges in $L$ incident to $w$. In that case, the improvement is reached after a single iteration. The algorithm then proceeds with the next phase, $i+1$.

Otherwise, in case all nodes $w$ in $L$ have degree equal to $B(w) + 1$, phase $i$ consists of more than one iteration. For the time being, we do not swap edge $e$ with one of the edges in $L$ incident to a node $w$ with degree $B(w) + 1$. This possible operation is called a potential swap. Instead, we update $C_{i,1}$ by combining all components along $L$ to a single component, including the nodes in $L \cap W_{i,1}$, which are removed from $W_{i,1}$. We denote the updated sets by $C_{i,2}$ and $W_{i,2}$ respectively, and move on to the next iteration. In the following iterations, we go back to look for other edges between components of $C_{i,j}$ and consider the respective cycles that could be generated by adding these edges to $T_i$. Components along these cycles are combined as described before (updating $C_{i,j}$ and $W_{i,j}$), as long as these cycles do not contain any node $w$ with degree higher than $B(w) + 1$. Again, if there are no edges in $G$ between components of $C_{i,j}$, the algorithm stops.

An improvement is identified when first observing a cycle that contains a node $w$ with degree higher than $B(w) + 1$. We then identify and perform a sequence of potential swaps leading to the final improvement step that decreases the degree of $w$. In order to identify the potential swaps that should be performed in phase $i$, we draw the graph of potential swaps $G^{PS}_i$ as follows. The set of nodes is $V$, and there is a directed edge going from $u$ to $v$ if each of $u$ and $v$ is the endpoint of a different edge taking part in the same potential swap, or in the improvement. Specifically, $u$ should be the endpoint of an edge in $T_i$, and $v$ should be the endpoint of an edge connecting two different components of $C_{i,j}$ for some iteration $j$. Now, we look for the shortest path in $G^{PS}_i$ going from $w$ to some node $s$ with degree at most $B(v)$ (with respect to $T_i$). We perform all potential swaps along this path. Note that after performing these swaps, the degree of node $w$ is decreased, the degrees of all internal nodes $x$ along the path in $G^{PS}_i$ remain $B(x) + 1$, and the degree of $s$ increases to at most $B(s) + 1$. Thus, the degrees of all nodes $v$ taking part in the swap operations doesn’t grow above $B(v) + 1$. After performing the sequence of swaps leading to the improvement, the algorithm proceeds with the next phase, $i+1$.

We denote by $T$ the final tree computed by the algorithm. When the algorithm ends, one of the two following possibilities holds:

- For each $v \in V$, $\text{deg}_T(v) \leq B(v) + 1$. We output $T$ as our approximate solution.

- We reach a phase where no improvement can be performed. That is, we identify a set of nodes $W$ whose removal from $T$ splits it to a set $C$ of connected components that are also different connected components in $G$. Note that all nodes $v \in W$ have degree at least $B(v) + 1$, and there is at least one node $w \in W$ with degree strictly higher than $B(w) + 1$. Applying Theorem 2, the set $W$ is thus a witness to the fact that there is no feasible solution with respect to the given degree bounds.
Assume \( T \) are not used to span the set \( N \) and an arbitrary Steiner tree \( T \) connecting two different components of \( W \) set to zero. Suppose that in iteration \( W \) e turn to describe the algorithm for the Steiner tree case, where \( W \) removed from \( G \). Give an index \( \text{iter} \) of nodes that are not currently in \( T \). The problem in the Steiner tree case, is that it is not clear how a sequence of swap operations leading to an improvement can be computed without arbitrarily increasing the degree of these nodes, as described for \( T \) an edge of \( W \). A \( \text{non-tree} \) path is a path going only through nodes in \( W \), except from its endpoints. In the spanning tree case, an improvement is made by performing a sequence of swap operations, where in each swap one edge was exchanged by another. Now, the idea is to define swap operations where an edge of \( T \) can be exchanged for a \( \text{non-tree} \) path.

4.2.2 Steiner Tree Case

We turn to describe the algorithm for the Steiner tree case, where \( N \subset V \). The algorithm starts with an arbitrary Steiner tree \( T \) that spans \( N \), and drop from \( T \) any unnecessary edges (that is, edges that are not used to span the set \( N \)). The algorithm is similar to the algorithm for the spanning tree case. Assume \( T_i \) is the tree generated upon reaching phase \( i \). We denote by \( X_i \) the actual set of nodes that are not in \( T_i \). A \( \text{non-tree} \) path is a path going only through nodes in \( X_i \), except from its endpoints. In the spanning tree case, an improvement is made by performing a sequence of swap operations, where in each swap one edge was exchanged by another. Now, the idea is to define swap operations where an edge of \( T_i \) can be exchanged for a \( \text{non-tree} \) path.

Algorithm 1 Algorithm Bounded Degrees Spanning Tree

1: Find a spanning tree \( T_1 \) of \( G \).
2: Set \( i \) to 1.
3: if there is no node \( w \) such that \( \text{deg}_{T_i}(w) > B(w) + 1 \) then
4: return \( T_i \) as the approximate solution.
5: end if
6: Let \( W_{i,1} \) be the set of all nodes \( v \) with degree \( \text{deg}_{T_i}(v) \geq B(v) + 1 \).
7: Set \( j \) to 1.
8: Let \( C_{i,1} = C_{i,1}^1, \ldots, C_{i,1}^m \) be the connected components formed by deleting \( W_{i,1} \) from the tree \( T_i \).
9: while there is an edge \( e = (u, v) \) connecting two different components of \( C_{i,j} \) do
10: if there is a node \( w \in W_{i,j} \) such that \( \text{deg}_{T_i}(w) > B(w) + 1 \) then
11: Identify the improvement swaps.
12: Exit while loop.
13: else
14: Identify the potential swaps in \( L \).
15: Form \( C_{i,j+1} \) from \( C_{i,j} \) by combining all components along \( L \) to a single component.
16: Form \( W_{i,j+1} \) from \( W_{i,j} \) by removing the nodes in \( L \cap W_{i,j} \).
17: \( j = j + 1 \).
18: end if
19: end while
20: if an improvement swap was identified then
21: Construct the graph of potential swaps \( G_{PS} \).
22: Find the shortest path in \( G_{PS} \) leading to the improvement.
23: Compute \( T_{i+1} \) from \( T_i \) by performing the sequence of swaps along the shortest path.
24: \( i = i + 1 \).
25: Go back to step 3.
26: else
27: \( \text{return} \) \( W_{i,j} \) as the witness to the infeasibility of our instance.
28: end if

4.2.2 Steiner Tree Case

We turn to describe the algorithm for the Steiner tree case, where \( N \subset V \). The algorithm starts with an arbitrary Steiner tree \( T \) that spans \( N \), and drop from \( T \) any unnecessary edges (that is, edges that are not used to span the set \( N \)). The algorithm is similar to the algorithm for the spanning tree case. Assume \( T_i \) is the tree generated upon reaching phase \( i \). We denote by \( X_i \) the actual set of nodes that are not in \( T_i \). A \( \text{non-tree} \) path is a path going only through nodes in \( X_i \), except from its endpoints. In the spanning tree case, an improvement is made by performing a sequence of swap operations, where in each swap one edge was exchanged by another. Now, the idea is to define swap operations where an edge of \( T_i \) can be exchanged for a \( \text{non-tree} \) path.

Assume there is a nontree path connecting two different components of \( C_{i,j} \) in iteration \( j \) of phase \( i \). Adding this path to \( T_i \) would introduce a cycle \( L \). We consider the nodes in \( L \cap W_{i,j} \), and identify either an improvement or a potential swap, depending on the degree of these nodes, as described for the spanning tree case. The problem in the Steiner tree case, is that it is not clear how a sequence of swap operations leading to an improvement can be computed without arbitrarily increasing the degree of nodes that are not currently in \( T_i \). As described in [11], this difficulty is overcome as follows. We give an index \( \text{iter}(v) \) to each node \( v \in T_i \), in order to keep track of the iteration \( j \) at which \( v \) was removed from \( W_{i,j} \). At the beginning of phase \( i \), the index of all nodes that are not in \( W_{i,1} \) are set to zero. Suppose that in iteration \( j \), a node \( w \in W_{i,j} \) takes part in a potential swap. Then, it is removed from \( W_{i,j} \) (as specified before), and its index is set to \( j \). When identifying a nontree path connecting two different components of \( C_{i,j} \), we choose the path \( p_i = (u, x_1, x_2, \ldots, x_j, v) \) with \( \text{iter}(v) \) and \( \text{iter}(u) \) minimal for paths connecting these components. The way we choose paths connecting
different components in $C_{i,j}$ leads to the following lemma.

**Lemma 5.1** [11] No vertex of $X_i$, the set of nodes which are not in $T_i$, acquires degree more than two during the update of $T_i$ (that is, when performing the sequence of swaps leading to the improvement in phase $i$).

The proof of Lemma 5.1 can be found in [11].

Note that after performing the sequence of swaps leading to the improvement in phase $i$, we need to drop any unnecessary edges in the new tree $T_{i+1}$. The rest of the algorithm is identical to the one presented for the spanning tree case.

**Algorithm 2 Algorithm Bounded Degrees Steiner Tree**

1: Find a Steiner tree $T_1$ of $G$ spanning $N$.
2: Drop any unnecessary edges of $T_1$.
3: Set $i$ to 1.
4: if there is no node $w$ such that $\text{deg}_{T_i}(w) > B(w) + 1$ then
5: return $T_i$ as the approximate solution.
6: end if
7: Let $W_{i,1}$ be the set of all nodes $v$ with degree $\text{deg}_{T_i}(v) \geq B(v) + 1$.
8: Set $j$ to 1.
9: Let $C_{i,1} = C_{i,1}^1, \ldots, C_{i,1}^m$ be the connected components formed by deleting $W_{i,1}$ from the tree $T_i$.
10: For each node $v \in C_{i,1}^m$, $1 \leq x \leq m$, set $\text{iter}(v) = 0$.
11: while there is a non-tree path connecting two different components of $C_{i,j}$ do
12: Choose the path $p_i = (u, x_1, x_2, \ldots, x_j, v)$ with $\text{iter}(v)$ and $\text{iter}(u)$ minimal.
13: Let $W_{i,j}^L$ be the set of nodes belonging to $W_{i,j}$ in the unique cycle $L \in T_i \cup p_i$.
14: if there is a node $w \in W_{i,j}^L$ such that $\text{deg}_{T_i}(w) > B(w) + 1$ then
15: Identify the improvement swaps.
16: Exit while loop.
17: else
18: Identify the potential swaps in $L$.
19: Form $C_{i,j+1}$ from $C_{i,j}$ by combining all components along $L$ to a single component.
20: For each node $v \in L \cap W_{i,j}$, set $\text{iter}(v) = j$.
21: Form $W_{i,j+1}$ from $W_{i,j}$ by removing the nodes in $L \cap W_{i,j}$.
22: $j = j + 1$.
23: end if
24: end while
25: if an improvement swap was identified then
26: Construct the graph of potential swaps $G_i^{PS}$.
27: Find the shortest path in $G_i^{PS}$ leading to the improvement.
28: Compute $T_{i+1}$ from $T_i$ by performing the sequence of swaps along the shortest path.
29: Drop any unnecessary edges of $T_{i+1}$.
30: $i = i + 1$.
31: Go back to step 4.
32: else
33: return $W_{i,j}$ as the witness to the infeasibility of our instance.
34: end if

**Claim 2.** The running time of the BD-ST approximation algorithm is $O(n^3 \log n)$.

**Proof.** The sum of the degrees of the nodes in a tree spanning $n$ nodes is exactly $2(n - 1)$. Thus, the number of nodes $v$ of degree $B(v) + d$ is $O(n/(B(v) + d))$. In each phase of the algorithm, the degree of a node $v$ whose degree is higher than $B(v) + 1$ is decreased by one, and is not increased in the other phases. Thus, there are at most $O(n/(B(v) + d))$ phases where degrees equal to $B(v) + d$ are decreased. Summing up the harmonic series corresponding to all possible values of nodes degrees ($\in (1, n)$), we get that there are at most $O(n \log n)$ phases. As each phase can be implemented in time $O(n^2)$ (identifying a sequence of swaps leading to an improvement, and using a union-find algorithm for maintaining connected components), we get that the running time of the algorithm is $O(n^3 \log n)$.  

\[ \square \]
4.3 Maximum Lifetime Approximation

The final step consists of using the approximation algorithm for the BD-ST problem, in order to derive the respective approximate solution for the maximum lifetime problem. As described in the proof of Theorem 1, this is done by performing a binary search over the possible lifetime values. For each lifetime value \( \sigma \), the maximum lifetime problem is translated to an instance of the BD-ST problem. The binary search stops at the maximal value \( \sigma^* \) for which the BD-ST algorithm finds an approximate solution. By applying Theorem 1, we thus get the following result.

**Theorem 3.** Given an instance \( G = (V,E,b) \) and a subset of nodes \( N \subseteq V \), there is a polynomial algorithm that finds a Steiner tree \( T \) spanning \( N \) with lifetime at least \( \sigma^*/2 \), where \( \sigma^* \) is the lifetime of an optimal solution.

5 Non-Uniform Power Transmissions, Non-Uniform Batteries

We now consider the general, and much harder case, where each node \( v \) has a different initial energy \( b(v) \), and can transmit at different power levels to its neighbors, depending on their topological distance from \( v \). We give an \( O(\log^2 n) \)-approximation for the maximum lifetime Steiner tree case, and an \( O(\log n) \)-approximation for the spanning tree case.

5.1 The WBD-Spanning Tree Approximation Algorithm

We first consider the WBD-ST problem for the case where \( N = V \), and describe a polynomial time algorithm that either proves that no feasible solution exists, or finds a spanning tree \( T \) such that for each node \( v \in V \), \( \frac{\text{wdeg}(v)}{B(v)} = O(\log n) \).

The technique we use is based on the method described in [10], where the minimum-degree spanning tree problem is considered. The algorithm keeps a spanning forest of the graph, and proceeds in phases. During phase \( i \), we have a spanning forest \( F_i \). Let \( G_{F_i} \) be the aggregated instance of \( G \), where each node in \( G_{F_i} \) corresponds to a connected component of \( F_i \). The algorithm starts with each node of the graph \( G \) as a separate component, thus forming \( F_1 \).

We begin each phase \( i \) by computing a maximal matching \( M_i \) of \( G_{F_i} \) (ignoring the weights of the edges). Let \( v \) be a node in \( G \), and \( wdeg(v) \) be its unweighted degree. We construct a bipartite graph \( B_{F_i} = (V_{F_i}^1, V_{F_i}^2, E_{F_i}^B) \). The node set \( V_{F_i}^1 \) contains the nodes of the original graph \( G \) that form \( V_{F_i}^M \). The node set \( V_{F_i}^2 \) equals the set \( V_{F_i}^U \) (where each unmatched component is represented by a node). The edge set \( E_{F_i}^B \) contains an edge from \( v \in V_{F_i}^1 \) to \( u \in V_{F_i}^2 \) whenever there is an edge between \( v \) and \( u \) in the original graph \( G \). The cost of the edge between \( v \) and \( u \) in \( E_{F_i}^B \) is the cost of the cheapest edge between \( v \) and \( u \) in \( G \).

Before describing the core of phase \( i \), we define a weighted bounded \( B \)-matching of \( B_{F_i} \), denoted by \( M_{F_i}^B \), as a subgraph of \( B_{F_i} \) in which the weighted degree of a node \( v \in V_{F_i}^1 \) is at most \( B(v) \), and the unweighted degree of each node in \( V_{F_i}^2 \) is at most 1. In other words, each node \( v \in V_{F_i}^1 \) is allowed to match several nodes in \( V_{F_i}^2 \), as long as the total weight of the adjacent edges of \( v \) participating in the \( B \)-matching does not exceed \( B(v) \). On the other hand, each node in \( V_{F_i}^2 \) is allowed to match at most one node in \( V_{F_i}^1 \). More generally, in a weighted bounded \( kB \)-matching of \( B_{F_i} \), the total weight of the adjacent edges of \( v \in V_{F_i}^1 \) participating in the \( kB \)-matching does not exceed \( k \cdot B(v) \). We define the size of a \( kB \)-matching of a bipartite graph \( (V_1, V_2, E^B) \) as the number of matched nodes in \( V_2 \). A maximum weighted bounded \( kB \)-matching is a \( kB \)-matching of maximum size.

After computing a maximal matching \( M_i \) in phase \( i \), we construct the bipartite graph \( B_{F_i} \) and perform one of the two possible following steps:
• We compute a 2B-matching $M^{2B}_i$ of $B_{F_i}$, with size equal to $|V^2_i|$. We form the forest of the next phase, $F_{i+1}$, by combining the components across matched edges in $M_i$ and $M^{2B}_i$.

• We prove that there is no feasible solution for our original instance, that is, there is no spanning tree $T$ of $G$ such that for each node $v \in V$, $wdeg_T(v) \leq B(v)$.

Our algorithm ends either when proving that there is no feasible solution, or when the number of connected components equals one, that is, after obtaining a spanning tree.

Algorithm 3 Algorithm Weighted Bounded Degrees Spanning Tree

1: $F_1 = V$
2: Set $i = 1$
3: if the number of connected components in $F_i = 1$ then
4: return
5: end if
6: Compute the graph $G_{F_i}$.
7: Find a maximal matching $M_i$ in $G_{F_i}$.
8: Construct the bipartite graph $B_{F_i} = (V^1_i, V^2_i, E^B_i)$, as described earlier.
9: if a 2B-matching $M^{2B}_i$ of $B_{F_i}$, with size equal to $|V^2_i|$ can be computed in polynomial time then
10: Form $F_{i+1}$ by combining the components across matched edges in $M_i$ and $M^{2B}_i$.
11: $i = i + 1$.
12: Go back to step 3.
13: else
14: Prove that there is no feasible solution for our instance.
15: return
16: end if

Lemma 2. In case there exists a feasible solution for our instance, then the size of every maximum B-matching in $B_{F_i}$ is $|V^2_i|$. Hence, in each phase $i$, there is a polynomial-time algorithm that either computes a 2B-matching of size $|V^2_i|$, or proves that no feasible solution exists for our original instance.

Proof. Let $T^*$ be a feasible spanning tree in $G$, that is, for each node $v$, $wdeg_{G^*}(v) \leq B(v)$. We consider the set of edges $E' \in (T^* \cap G_{F_i})$. These edges form a spanning tree in $G_{F_i}$. Note that as $M_i$ is a maximal matching, there are no edges between the components of $V^2_i$ in the original graph $G$. Thus, each component $u \in V^2_i$ is connected to at least one of the nodes $v \in V^1_i$ by an edge in $T^* \cap G_{F_i}$, corresponding also to an edge in $E^B_i$. For each $u \in V^2_i$, we pick one of these nodes arbitrarily and match $u$ to $v$. This matches all the components of $V^2_i$, and the weighted degree of each node $v \in V^1_i$ is bounded by $B(v)$. Thus, there exists a B-matching in $B_{F_i}$ whose size is $|V^2_i|$. Obviously, this is also a maximum B-matching.

In order to compute a 2B-matching of size $|V^2_i|$, we use a reduction to the Generalized Assignment problem (GAP), defined as follows. We are given a set of $t$ items and $m$ bins (knapsacks), such that each bin $j$ has a capacity $c(j)$, and each item $i$ has a profit $p_j(i)$ and a size $s_j(i)$ in case it is assigned to bin $j$. The goal is to find a subset of items of maximum profit such that they have a feasible packing in the bins. A 2-approximation polynomial algorithm for GAP can be found in [22]. It is shown in [22] that there is a polynomial-time algorithm that, given a value $P$, either proves that there is no feasible packing with profit $P$, or else finds a packing of profit at least $P$, where the total size of the items packed in bin $j$ is at most $2c(j)$.

We reduce our B-matching problem to GAP as follows. The bins are the nodes in $V^1_i$. The capacity of each bin is the weighted degree bound of the respective node. The items are the components of $V^2_i$. The capacity of item $u \in V^2_i$ when being assigned to bin $v \in V^1_i$ is the cost of the edge between $u$ and $v$ in $E^B_i$. If there is no such edge, the respective capacity is set to $\infty$. All items have a profit of 1, no matter what bin they are assigned to. Thus, using the polynomial-time algorithm presented
in [22], we can either prove that no feasible solution exists (that is, there is no feasible packing with profit \( |V_i^2| \)), or else compute a \( 2B \)-matching of size \( |V_i^2| \).

We turn to analyze the running time and approximation ratio of our algorithm. Observe that in each phase \( i \) of the algorithm, all components of the forest \( F_i \) are matched either by \( M_i \) or by \( M_i^{2B} \). Thus, the number of connected components in each consecutive phase goes down by at least a half, leading to at most \( O(\log n) \) phases. As each phase runs in \( O(n^2 \log n) \) time ([22]), the whole algorithm runs in \( O(n^2 \log^2 n) \) time.

In each phase \( i \) of the algorithm, the degree of each node \( v \) increases by at most \( B(v) \) due to the matching \( M_i \), and by an additional value of at most \( 2B(v) \) due to the matching \( M_i^{2B} \). Therefore, the weighted degree of any node \( v \) goes up by at most \( 3B(v) \) in each phase. As there are at most \( O(\log n) \) phases, we can either prove that there is no feasible solution, or get a spanning tree \( T \), where for each node \( v \in V \), \( \text{wdeg}_T(v) = O(\log n) \cdot B(v) \).

5.2 The WBD-Steiner Tree Approximation Algorithm

We turn to describe the algorithm for the Steiner tree case, where \( N \subset V \). Our algorithm either proves that there is no feasible solution for the given instance, or finds a spanning tree \( T \) such that for each node \( v \in V \), \( \frac{\text{wdeg}_T(v)}{B(v)} = O(\log^2 n) \). The algorithm is similar to the spanning tree case except that instead of computing a matching in each phase \( i \), we reduce the number of connected components by at least a half by computing paths between terminals, as follows.

The algorithm starts with each terminal in the graph \( G \) as a separate component, thus forming \( F_1 \). Focusing on phase \( i \), we denote the set of nodes in \( G_{F_i} \) as \( V_{F_i} \), and set \( k = |V_{F_i}| \). Consider a feasible Steiner tree spanning the nodes in \( G_{F_i} \). We can get an Euler tour of the tree by doubling the edges. We now can decompose the Euler tour into \( k/2 \) disjoint paths, where the endpoints of each path are terminals, and each terminal is the endpoint of at most one path. The load of the paths on each node \( v \) is at most \( 2B(v) \), since each edge appears twice in the Euler tour.

We replace the undirected graph by a directed flow network, where each undirected edge \( e = (u, v) \) is replaced by two directed edges, \( (u, v) \) and \( (v, u) \), both with weight \( w(e) \). We solve a (fractional) multicommodity flow linear program (MF-LP), where each pair of terminals \( (t_i, t_j) \) \((i < j)\) is a commodity, and the goal is to maximize the total amount of flow routed between the terminal pairs. The flow between each pair of terminals \( (t_i, t_j) \) is constrained to be at most 1 (Constraint (i)). In addition to the standard flow constraints (Constraints (ii)-(iii)), we add a load constraint (Constraint iv) according to which the maximum flow through node \( v \) is at most \( 2B(v) \). We denote by \( f_{i,j}(e) \) the flow of commodity \( (t_i, t_j) \) over edge \( e \), and by \( N(v) \) the edges adjacent to \( v \). The linear program is as follows.

\[
\text{maximize } \sum_{i\leq j \leq k} \sum_{v \in V} f_{i,j}(t_i, v) \quad \text{s.t.}
\]

For each commodity \( (t_i, t_j) \), \( i < j \):

1. \( \sum_{v \in V} f_{i,j}(t_i, v) \leq 1 \quad \text{(i)} \)
2. \( \sum_{v \in V} f_{i,j}(u, v) = 0 \quad \forall u \in V \setminus \{t_i, t_j\} \quad \text{(ii)} \)
3. \( f_{i,j}(u, v) = -f_{i,j}(v, u) \quad \forall e = (u, v) \in E \quad \text{(iii)} \)
For each node \( v \in V \):

\[
\sum_{e \in N(v)} \left( w(e) \cdot \sum_{i,j \in [1, \ldots, k]} f_{i,j}(e) \right) \leq 2B(v) \tag{iv}
\]

By the Euler tour argument, in case there exists a feasible Steiner tree spanning the nodes in \( G_{F_i} \), there is also a feasible solution to the linear program for which the value of the objective function is at least \( k/2 \). Thus, in case the value of the solution of the linear program is less than \( k/2 \), we obtain a proof for the infeasibility of our instance.

We decompose the fractional solution into a set of flow paths (at most \( n \) for each commodity) and perform randomized rounding. For each commodity \((t_i, t_j)\), we choose (at most) one of the flow paths between \( t_i \) and \( t_j \), where the probability of choosing path \( p \) is equal to the amount of flow on \( p \), denoted by \( f(p) \).

We turn to analyze the load on each node \( v \). Consider flow path \( p \) going through \( v \). We denote by \( w_v^{in}(p) \) and \( w_v^{out}(p) \) the weight of the incoming and outgoing edges of \( v \) used by \( p \), and by \( w_v(p) \) the (total) load of path \( p \) on \( v \), that is, \( w_v(p) = w_v^{in}(p) + w_v^{out}(p) \). According to the fractional solution, \( p \) contributes a value of \( f(p)w_v(p) \) to the load on \( v \). Thus, Constraint (iv) can be rewritten as:

\[
\sum_{\{p|v \in p\}} f(p) \cdot w_v(p) \leq 2B(v) \tag{7}
\]

We say that \( w_v(p) \) is the weight of flow path \( p \) with respect to \( v \). (When focusing on node \( v \), we also call it the weight of \( p \).) We denote by \( x_p \in \{0, 1\} \) the unweighted random variable associated with the possible choice of path \( p \). Recall that \( \Pr[x_p = 1] = f(p) \).

We consider only paths with weight larger than \( B(v)/n \) with respect to \( v \), and show that the probability that the load of these paths on \( v \) is beyond a logarithmic factor, i.e., \( O(\log n) \cdot B(v) \), is negligible. As there are at most \( n \) edges adjacent to \( v \) with weight \( \leq B(v)/n \), the total load on \( v \) due to paths with weight at most \( B(v)/n \) is at most \( B(v) \), and thus we may lose at most an additional additive factor of \( B(v) \).

For the sake of the analysis, we round the weight of each flow path going through \( v \) to the nearest power of two, and divide the paths going through \( v \) into groups according to their rounded weights. As the maximum weight of an edge adjacent to \( v \) is \( B(v) \), we get \( t = O(\log n) \) different weight groups. (Edges with weight larger than \( B(v) \) are ignored as they can not participate in a feasible solution). Define group \( S_i \) as containing the flow paths for which the rounded weight is \( W_0 2^i \), where \( W_0 = 2^{\lceil \log(\frac{B(v)}{n}) \rceil} \), and \( 1 \leq i \leq t \). We define \( m_i = \sum_{p \in S_i} f(p) \), i.e., the unweighted expectation of group \( S_i \). Clearly, by equation 7, it holds that

\[
W_0 \sum_{i=1}^{t} 2^i \cdot m_i = O(B_v). \tag{8}
\]

We also say that \( W_0 2^i \cdot m_i \) is the weighted expectation of group \( S_i \).

**Claim 3.** The probability that the weighted sum of the paths in group \( S_i \) is beyond a logarithmic factor from their weighted expectation, is polynomially small.

**Proof.** Within each group \( S_i \), we can aggregate flow paths \( p_j \) belonging to the same commodity by assigning a single random variable \( x_p \) where \( \Pr[x_p = 1] = \sum_j f(p_j) \). We are thus left with independent \( \{0, 1\} \) random variables \( x_p \) associated with paths belonging to different commodities. By applying Chernoff bounds, we get the following inequality, where \( c \) is a constant:
\[
\begin{align*}
\Pr \left[ W_0 2^i \sum_{p \in S_i} x_p > W_0 2^i \cdot m_i \cdot c \log n \right] < \frac{1}{n^t}. \tag{9}
\end{align*}
\]

Therefore, the probability that the bound guaranteed in equation 9 holds simultaneously for all groups \( S_i, 1 \leq i \leq t \) is at least \((1 - t/n^4)\). Now, as each group \( S_i \) contains different flow paths (i.e., for each \( i \neq j \), \( S_i \cap S_j = \emptyset \)) we can split the expectation of the sum of all the paths (specified in equation 8) between the groups and then sum up the deviation from expectation in each group. We thus get the following equation, where \( c' \) is a constant:

\[
\Pr \left[ W_0 \sum_{i=1}^{t} 2^i \sum_{p \in S_i} x_p \leq B(v) \cdot c' \log n \right] \geq 1 - \frac{t}{n^4}. \tag{10}
\]

Equation 10 implies that in phase \( i \) of the algorithm, the probability that a node \( v \) is overloaded beyond a logarithmic factor, is polynomially small \((t/n^4)\). Summing up this probability over all the nodes, we get that the probability that no node in \( G_{F_i} \) is overloaded beyond a logarithmic factor is at least \((1 - t/n^3)\).

We have thus showed how to compute an integral solution (i.e., a set of flow paths in \( G_{F_i} \)), where the expected number of paths is at least \( k/2 \), and the probability that some node is overloaded beyond a logarithmic factor \( (O(B(v) \cdot \log n)) \) is polynomially small. We note that this solution can be derandomized by using the method of conditional probabilities [1].

Finally, we form the forest of the next phase, \( F_{i+1} \), by combining the components connected by the flow paths, thus decreasing the number of connected components by half. As in each phase, the union of the flow paths decreases the number of commodities by at least a factor of 2, the number of phases is logarithmic.

**Algorithm 4** Algorithm Weighted Bounded Degrees Steiner Tree

1: \( F_1 = N \).
2: Set \( i = 1 \).
3: if the number of connected components in \( F_i = 1 \) then
   4: \hspace{1em} return
5: \hspace{1em} end if
6: Compute the graph \( G_{F_i} \), and replace the undirected graph by a directed flow network.
7: Solve the (fractional) multicommodity flow linear program MF-LP.
8: if the value of the solution of MF-LP is at least \( k/2 \) then
   9: \hspace{1em} Decompose the fractional solution into a set of flow paths.
10: \hspace{1em} Choose (at most) one path for each \((t_i, t_j)\) by randomly selecting according to the distribution \( f(p) \).
11: \hspace{1em} Form \( F_{i+1} \) by combining the components connected by the flow paths.
12: \hspace{1em} \( i = i + 1 \).
13: \hspace{1em} Go back to step 3.
14: else
15: \hspace{1em} There is no feasible solution for our instance.
16: \hspace{1em} return
17: end if

In each phase \( i \) of the algorithm, the degree of each node \( v \) increases by at most \( O(\log n)B(v) \). As there are at most \( O(\log n) \) phases, we can either prove that there is no feasible solution, or get a Steiner tree \( T \) spanning \( N \), where for each node \( v \in V, \ wdeg_T(v) = O(\log^2 n) \cdot B(v) \).
5.3 Maximum Lifetime Approximation

By applying Theorem 1, we get the following result.

**Theorem 4.** For an instance $G = (V, E, b, p)$, a Steiner tree $T$ with lifetime at least $\sigma^*/O(\log^2 n)$ (or, $\sigma^*/O(\log n)$ for the spanning tree case), where $\sigma^*$ is the lifetime of an optimal solution, can be computed in polynomial time.

Improving the above approximation bounds remains an open problem.

6 The Maximum-Lifetime Multicast Game in the Single-Recipient Environment

6.1 The Multicast Game: Model & Definitions

We define a 2-way multicast game (referenced shortly as the multicast game) for the single-recipient environment, with a finite number of selfish non-cooperative users (called also players). There is a special node $r \in V$ called root (or source) and a subset of $k$ nodes $N = \{t_1, t_2, \ldots, t_k\}$ representing the multicast players $\{1, 2, \ldots, k\}$. Each player is interested in connecting to the source by choosing a route with maximum lifetime (assuming a 2-way transmission).

A course of action chosen by player $i$ at any time is called its strategy and is denoted by $s_i$. In our game, a strategy of player $i$ is a path connecting $t_i$ to the root. The strategy space of player $i$ (i.e., the set of all its possible strategies) is denoted by $S_i$, and it is the set of all the possible paths between $t_i$ and the root. At any given moment, a strategy profile (or a configuration) of the game is the vector of all the strategies of the players, $s = (s_1, \ldots, s_k)$. Given a strategy profile $s$, the lifetime of player $i$ is constrained by the node with minimum lifetime on its path to $r$. Thus, the mutual influence of the players arises from using common nodes.

A strategy profile $s \in S$ is at **Nash equilibrium** if no player has an incentive to change its routing strategy, assuming the strategies of the other players are fixed. We assume that a player changes its routing choice if and only if it increases its lifetime, given the strategies of the other players. We therefore say that at each step the strategy of the current player is a **best response** to the other players’ strategies. A change of strategy by any player is called a **Nash defection** and the player is called **Nash defector**.

We consider the following best-response dynamics of the game. We start from some initial configuration, and at each step one of the players makes a routing decision that maximizes its lifetime. The routing decisions can be changed throughout the game.

Consider the following scenario: two players connect to the source $r$ via a node $v$. Each player chooses a different sub-path from $v$ to $r$, creating a cycle. In case the lifetime of their path is constrained by a node in $v$’s subtree, or in case both sub-paths from $v$ to $r$ have the same lifetime, none of the players has an incentive to join the other in its path from $v$ to $r$. In order to avoid such cases, we refine the game model as follows. Each player $i$ making a routing decision can only decide how to connect to the existing tree (formed by the routing decisions of the other players). As all other users connect through maximum lifetime paths, this restriction cannot decrease the maximum possible lifetime of player $i$ given the routing decisions of the other players.

In addition, a player $i$ wishing to change its sub-path from a node $v$ (currently in its path) to the root, will cause a similar change in the routes of all terminals in $v$’s subtree. As $i$’s new routing decision improves its lifetime, it can only improve the lifetime of the terminals in $v$’s subtree. Clearly, each move performed by a player changing its strategy guarantees that the resulting configuration induces a tree rooted at $r$. 


6.1.1 Nash Equilibrium Existence & Convergence

**Theorem 5.** A Nash equilibrium exists for every instance of the single-recipient wireless multicast game. Moreover, best-response dynamics always converge to a Nash equilibrium.

*Proof.* Given an arbitrary initial configuration, we arrange the players lexicographically according to a non-decreasing order \( \Theta \) of the lifetime of their routes, that is, the user with minimum lifetime is at the most significant place of the ordering. In order to prove Theorem 5, we first prove the following claim.

**Claim 4.** Assume that player \( i \) moves from path \( p_1 \) to path \( p_2 \). Then, the lexicographic ordering \( \Theta \) can only increase.

*Proof.* We denote the endpoint of \( p_1 \) and \( p_2 \) by \( v \). Note that in case \( p_1 \) and \( p_2 \) are sub-paths of player \( i \)'s route to \( r \), then \( v \neq t_i \). Given a node \( v \), the *first common ancestor* of \( v \) is the first ancestor of \( v \) that is used by another player. We denote the first ancestors of \( v \) in \( p_1 \) and \( p_2 \) by \( w_1 \) and \( w_2 \) respectively.

Player \( i \) defects from its current strategy only if it increases its path lifetime. Similarly, the path lifetime of users in the subtree of \( v \) can only increase. Thus, after changing its path, player \( i \) moves to a less significant place of \( \Theta \). Similarly, users in \( v \)'s subtree may also move to less significant places of \( \Theta \). As a result, \( \Theta \) increases.

In addition, there are two kinds of players whose paths lifetimes may be influenced by the move of player \( i \).

1. Players who use \( w_1 \). As player \( i \) does no longer use \( w_1 \), the lifetime of the paths of these players can only increase. Thus, after the move of player \( i \), these players may move to less significant places of \( \Theta \), resulting in an increase of \( \Theta \).

2. Players who use \( w_2 \). As player \( i \) joins the users of \( w_2 \), the lifetime of the players that are currently constrained by \( w_2 \) decreases. We denote these players by \( N_{w_2} \). Assuming \( N_{w_2} \neq \emptyset \), the lifetime of player \( i \) is also constrained by \( w_2 \) (otherwise the lifetime of players in \( N_{w_2} \) would have been constrained by another node in \( p_2 \)). Note that \( w_2 \) constraint consists of an increase in player \( i \)'s lifetime, but a decrease in the lifetime of players in \( N_{w_2} \). Thus, before player \( i \)'s move, the lifetime of these players was bigger than player \( i \)'s original lifetime. Accordingly, the original place of these players in \( \Theta \) was less significant than player \( i \)'s original place. After player \( i \)'s move, players in \( N_{w_2} \) and player \( i \) have an equal lifetime. Therefore, as a result of the change in the lifetime of players in \( N_{w_2} \), \( \Theta \) decreases by a smaller value than the increase caused by the change in player \( i \)'s lifetime.

\[\Box\]

**Corollary 1.** In the game course induced by best-response dynamics, the minimum lifetime of all nodes can only increase (otherwise, the ordering \( \Theta \) could also decrease, in contradiction to the proof of Claim 4). Thus, the lifetime of the solution can only increase.

As each player has a finite strategy space (number of possible routes to \( r \)), the lexicographic ordering \( \Theta \) is bounded. Each Nash defection causes an increase in the lexicographic ordering. Thus, best-response dynamics of the single-recipient wireless multicast game always converges to a Nash equilibrium.

\[\Box\]
6.1.2 Price of Anarchy

The price of anarchy denotes the ratio of the cost of the worst Nash equilibrium and the (social) optimal solution. This value quantifies the “penalty” incurred by lack of cooperation (or coordination) between the players in a non-cooperative game. In our case, the price of anarchy is the ratio between the lifetime of the optimal solution and the lifetime of the worst Nash solution.

We observe that, when considering the worst case Nash equilibrium, the price of anarchy is not bounded. To see that, consider Figure 1. Assume all nodes have the same initial energy. The power of edges $(1, x)$ and $(2, y)$ is a large number $M$, and the power of all other edges is 1 ($M >> 1$). A solution where player 1 connects to the source through node $x$ and player 2 connects to the source through node $y$ is a Nash equilibrium with lifetime $1/M$. On the other hand, in the optimal solution, player 1 connects through $y$ and player 2 connects through $x$. The lifetime of the resulting tree is 1.

Notice however that the bad solution in the example in Figure 1 cannot be reached if the players join an initially empty game one-by-one, each of them choosing the best path to connect to the source.

Next, we consider an online setting, where players first join the game sequentially starting from an “empty” configuration. Upon arrival, each player picks a path (selfishly) that maximizes its lifetime. Once all players have joined the game, the natural game course induced by best-response dynamics continues until a Nash equilibrium is reached.

We show that even for this setting, the price of anarchy is not bounded. Consider the instance described in Figure 2. Assume all nodes have the same initial energy, and all edges have the same power, all equal to 1. Consider the tree formed by the solid lines. Let $v_1, \ldots, v_d$ be the first level
users (or nodes), i.e., the children of $r$. The second level users (or nodes) are the children of $v_1, \ldots, v_d$. In the optimal solution, all nodes at the same level are connected through the dotted lines, and the leftmost child is connected to its parent through the solid line. In this solution, node $r$ has a single outgoing edge (to $v_1$), each node $v_i$, $1 \leq i < d$, has 2 outgoing edges (one to its right sibling, and one to its leftmost child), and each of the second level nodes has at most one outgoing edge (to its right sibling). The lifetime of this solution is therefore $1/2$.

Now, consider the online setting where players join the game sequentially beginning from an empty configuration. The arrival schedule of the players is represented by the numbers next to the first and second level nodes in Figure 2. The number of each user denotes the order of its arrival (the user at $v_1$, numbered as 1, arrives first, the user at $a_1$, numbered as 2, arrives second, etc.). Each time a first level user arrives it connects directly to $r$, as the number of outgoing edges of $r$ and of its left sibling are equal. Each time a second level user arrives, the lifetime of its path is constrained by $r$, and thus it connects directly to the respective first level node. The resulting solution is a Nash equilibrium. The degree of $r$ and of all first level nodes is equal to $d$, and thus no user has an incentive to change its strategy. The lifetime of this solution is $1/d$, where $d$ can be arbitrarily large.

6.1.3 Price of Stability

Motivated by the negative results on the price of anarchy, we turn our attention to the price of stability, defined to be the ratio between the cost of the best Nash solution and the cost of an optimal solution. In our case, the price of stability is the ratio between the lifetime of the maximum lifetime Steiner tree and the lifetime of the best Nash solution. This notion defines the outcomes of scenarios in the “middle ground” between centrally-enforced solutions and non-cooperative games. In such scenarios, there is an underlying protocol that suggests an initial configuration to all players, who can either accept it, or try to defect from it. Assuming the solution proposed is stable (i.e., a Nash equilibrium), no player would have an incentive to defect from it assuming the other players accept it. Therefore, the goal of the underlying protocol is to find the best possible such solution, leading to the notion of the price of stability.

Theorem 6. The price of stability of the single-recipient wireless multicast game is 1.

Proof. Following is a constructive proof. Assume the initial configuration consists of a maximum lifetime Steiner tree. This configuration is not necessarily a Nash equilibrium, however, by following the natural game course induced by best-response dynamics a Nash equilibrium is reached (Theorem 5). As the lifetime of the respective solution can only increase during best-response dynamics (Corollary 1), it follows that the lifetime of the Nash equilibrium reached is equal to the lifetime of the optimal solution.

Thus, if a central authority could enforce the initial configuration, the best Nash equilibrium would be its choice. In this case, no user would defect from this configuration, and its lifetime would be optimal.

However, it is not clear whether the best Nash equilibrium can be computed in polynomial time. Nevertheless, in case the initial configuration is not stable, by following the natural game course induced by best-response dynamics, our game would converge to a Nash equilibrium. Therefore, we suggest a mechanism in which a central authority starts the process by first computing the best approximately optimal solution computable in polynomial time, and then allows the users to follow their best-response dynamics.
7 The Maximum-Lifetime Multicast Game in the Multi-Recipients Environment

7.1 The Multi-Recipients Environment

We consider a multi-recipients wireless network, where the nodes communicate using omnidirectional antennas. We assume static locations of wireless nodes. In the multi-recipients model, when a node uses enough power in order to transmit over a distance $d$, the transmission can be received by any other node whose distance from the sender is at most $d$. The transmission power consumed by a sender is increasing in the distance over which the transmission is performed.

The wireless network is modeled by a complete graph $G = (V,E)$. Each node $v \in V$ has a finite battery with energy $b(v)$, and is able to transmit until it consumes all its energy. For each edge $e = (v,u)$, $p(e)$ is the power (energy per time unit) required for node $v$ to transmit to node $u$.

The lifetime $l(v)$ of node $v$ is the consumption time of its battery. The lifetime of $v$ is constrained only by the edge with maximum power over which $v$ transmits (as by transmitting over this edge, all closer nodes receive the transmission anyway), and is thus equal to

$$l(v) = \frac{b(v)}{\max_{e=(v,u) \in T:v \text{ transmits to } u\{p(e)\}}}. $$

The lifetime $l(p)$ of path $p$ is the lifetime of the node with minimum lifetime in $p$.

We first address a two-way multicast problem, defined as in the single-recipient environment. The problem is thus to find a routing scheme that maximizes the time during which the transmission can be performed both from the source and from the terminals. We define the lifetime of a Steiner tree $T$ spanning $N$ as the lifetime of the path with minimum lifetime in $T$. That is, the lifetime of $T$ is defined by the node $v \in T$ whose initial energy (battery) is consumed first, i.e, the node with minimum lifetime $\max_{e=(v,u) \in T:v \text{ transmits to } u\{p(e)\}}$.

7.2 Nash Equilibrium Existence & Convergence

The two-way multicast game (referenced shortly as the multicast game) for the multi-recipients case is defined as for the single-recipient case.

**Theorem 7.** A Nash equilibrium exists for every instance of the multi-recipients wireless multicast game. Moreover, best-response dynamics always converges to a Nash equilibrium.

**Proof.** The proof of this theorem is identical to the Nash equilibrium existence proof of the single-recipient wireless multicast game. The only difference is that players whose lifetime is constrained by $w_2$ are not necessarily affected by the change in $i$’s path (consider the case where the power of the new outgoing edge from $w_2$, resulting from player $i$’s move to $p_2$, is lower than the power of existing outgoing edges from $w_2$). In this case, the increase in the value of the ordering $\Theta$ is even larger. □

**Corollary 2.** In the game course induced by best-response dynamics, the lifetime of the solution can only increase.

7.3 Price of Anarchy

We first observe that the price of anarchy is not bounded in the case of the worst case Nash equilibrium. To see that, consider Figure 1. The initial energy of the nodes, and the power of the edges, are as specified for the single-recipient model. The multi-recipients wireless network should be modeled by a complete graph, thus, we assume that all the edges that do not appear in Figure 1 have powers much
higher than \( M \), and are therefore ignored. The worst Nash equilibrium and optimal solution in this example are similar to those described for the multicast game in the single-recipient model.

Notice however that the bad solution in the example in Figure 1 cannot be reached if the players join an initially empty game one-by-one, each of them choosing the best path to connect to the source. Next, we consider an online setting, where players first join the game sequentially starting from an empty configuration, and then follow the natural game course induced by best-response dynamics.

**Lemma 3.** In the multi-recipients wireless multicast game, the ratio of the lifetime of a Nash equilibrium reached in an online setting to the lifetime of an optimal solution, is 1.

**Proof.** We denote by \( p^*_i \) the path with maximum lifetime from \( r \) to \( t_i \), assuming there are no other users, and by \( l^*_i \) the lifetime of \( p^*_i \). Thus, \( l^*_i \) is the lifetime of the node with minimum lifetime in \( p^*_i \). We denote by \( l^* \) the lifetime of the path \( p^* \) with minimum lifetime over all paths \( p^*_i \) (that is, \( l^* = \min_i \{ l^*_i \} \)) and by \( v^* \) the node with minimum lifetime in \( p^* \). Recall that the lifetime of \( v^* \) is constrained only by the edge with maximum power over which \( v^* \) transmits. Clearly, \( l^* \) is the lifetime of an optimal solution, consisting, for example, of the paths \( p_i \). In an online setting, the users join the game sequentially by making their best possible routing decisions. Thus, \( l^* \) can neither be increased nor decreased by the routing choices of the players along the game, and it is therefore also the lifetime of a Nash equilibrium reached in such a setting. \( \square \)

### 7.4 Price of Stability

**Theorem 8.** The price of stability of the multi-recipients wireless multicast game is 1.

The proof is identical to the proof of the price of stability for the single-recipient wireless multicast game (Theorem 6).

We thus suggest a mechanism in which a central authority starts the process by first computing the optimal solution, and then allows the users to follow their best-response dynamics. Note that in the multi-recipients case, the optimal solution can be computed in polynomial time.

### 8 The Maximum-Lifetime Anycast Game

We define an anycast game both for the single-recipient and multi-recipients environments, with a finite number of selfish non-cooperative players. There is a subset of \( k \) nodes \( N = \{ t_1, t_2, \ldots, t_k \} \) representing the multicast players \{1,2,\ldots,k\}, that wish to communicate with one another. Given a strategy profile \( s \), the lifetime \( l_i \) of player \( i \) is defined as the lifetime of its minimum lifetime path to one of the other players, i.e., \( l_i = \min_{t_j \in N} \{ l(t_j) \} \). Each player is interested in connecting to the existing configuration by choosing a route that maximizes its lifetime. Given a strategy profile \( s \), the strategy space of player \( i \) is the set of all the possible paths between \( t_i \) and the existing configuration, consisting of the routing choices of the other users. The rest of the model and definitions are similar to the multicast game.

We note that all the theorems, results and corollaries given for the multicast game in each environment remain valid for the anycast game in the respective environment. The following slight changes are only due to the difference between the model definitions.

In both the directional and omnidirectional environments, we prove the existence and convergence to a Nash equilibrium by arranging the players lexicographically according to a non-decreasing order \( \Theta \) of their lifetime (that is, their minimum lifetime to one of the other players). The rest of the proof is similar to the proof of Theorem 5. Proving that the price of stability is 1 (for both environments) follows from the proof of the Nash equilibrium convergence, as for the multicast game.

In order to prove that the price of anarchy of an online setting in the multi-recipients environment is 1, we denote by \( P^*_i \) the set of paths with maximum lifetime from \( t_i \) to all other players \( t_j \in N \).
We denote by $p^*_i$ the path with minimum lifetime in $P^*_i$, and by $l^*_i$ the lifetime of $p^*_i$. Note that by choosing the paths $P^*_i$ for each player $i$, we get an optimal solution. The rest of the proof is similar to the proof of Lemma 3.

In all the examples given for the multicast game, we assume here that $r \in N$. Also, in the example described in Figure 2, assume that $r$ is the first player joining the game. The arrival schedule of the other users is as specified for the multicast game.

9 Conclusions

We have considered routing algorithms for maximizing the lifetime of the 2-way multicast and anycast problems in wireless networks, under the standard and practical single routing topology setting.

We first assumed a centralized approach and provided a theoretical analysis for the lifetime system optimization problem. Focusing on the case of directional antennas, we observed that computing an optimal solution is an NP-hard problem. Accordingly, we established polynomial time approximation algorithms with proven performance. To our knowledge, these are the first near-optimal solutions given for the lifetime maximization problem with directional antennas.

In the second part of our work, we investigated a distributed, dynamic setting, where each node makes its own routing decisions based on the current environment. We considered the corresponding noncooperative game scenarios of the maximum lifetime problems, both for the environments of directional and omnidirectional antennas. In both settings, we proved that by following the natural game course, users converge to a Nash equilibrium. We considered several Nash equilibrium points of particular interest, and quantified the inefficiency resulting from these games through the ratio between the lifetime of these equilibrium points, and the lifetime of an optimal solution. Among other results, we showed that for the directional case, the price of stability, where the best Nash equilibrium is considered, is 1; hence, optimal (networkwide) performance can be achieved if the initial configuration can be imposed on the players. These results give further significance to the approximate solutions computed in the first part of the work, as they can be used as initial configurations of the game, assuring the performance of the Nash equilibrium reached by following the natural game course.

Several issues merit further research. One direction is to improve the approximation bounds for the maximum lifetime problem, mainly for the general case where there are different transmission power levels. Another issue is the investigation of the standard single-way multicast setting, for the single-recipient environment. In this setting, the battery of a node is consumed only by the transmission power over its adjacent edges to the downstream receivers. Note that our approximation algorithms are not valid for this case, as it should be modeled by a directed graph. However, our results for the 2-way multicast game apply directly to the single-way multicast game. In addition, a few questions remain open for the maximum lifetime games. First, for the online setting of the single-recipient wireless multicast game we provide a worst case analysis. An interesting issue would be to perform an average case analysis, and evaluate it by way of simulations. Another issue is the time convergence to a Nash equilibrium, which remains to be explored.

Finally, this study has focused on stationary wireless networks, in which nodes do not change locations. While our results, in particular for the dynamic setting of noncooperative games, provide important insight for handling the mobile case, future research should explore the precise impact of node mobility and establish efficient solutions in terms of lifetime maximization.
References


