Near Optimal Leader Election in Multi-Hop Radio Networks

Mohsen Ghaffari*, Bernhard Haeupler†

Abstract

We design leader election protocols for multi-hop radio networks that elect a leader in almost the same time T_{BC} that it takes for broadcasting one message (one ID). For the setting without collision detection our algorithm runs whp. in $O(D \log \frac{n}{D} + \log^3 n) \cdot \min\{\log\log n, \log \frac{n}{D}\}$ rounds on any n-node network with diameter D. Since $T_{BC} = \Theta(D \log \frac{n}{D} + \log^2 n)$ is a lower bound, our upper bound is optimal up to a factor of at most $\log\log n$ and the extra $\log n$ factor on the additive term. Our algorithm is furthermore the first $O(n)$ time algorithm for this setting.

Our algorithm improves over a 23 year old simulation approach of Bar-Yehuda, Goldreich and Itai with a $O(T_{BC} \log n)$ running time: In 1987 they designed a fast broadcast protocol and subsequently in 1989 they showed how it can be used to simulate one round of a single-hop network that has collision detection in T_{BC} time. The prime application of this simulation was to simulate Willards single-hop leader election protocol, which elects a leader in $O(\log n)$ rounds whp. and $O(\log\log n)$ rounds in expectation. While it was subsequently shown that Willards bounds are tight, it was unclear whether the simulation approach is optimal. Our results break this barrier and essentially remove the logarithmic slowdown over the broadcast time T_{BC}. This is achieved by going away from the simulation approach.

We also give an $O(D + \log n \log\log n) \cdot \min\{\log\log n, \log \frac{n}{D}\} = O(D + \log n) \cdot O(\log\log n)^2$ leader election algorithm for the setting with collision detection (even with single-bit messages). This is optimal up to $\log\log n$ factors and improves over a deterministic algorithm that requires $\Theta(n)$ rounds independently of D.

Our almost optimal leader election protocols are especially important because countless communication protocols in radio networks use leader election as a crucial first step to solve various, seemingly unrelated, communication primitives such as gathering, multiple unicasts or multiple broadcasts. Even though leader election seems easier than these tasks, its best-known $O(T_{BC} \log n)$ running time had become a bottleneck, preventing optimal algorithms. Breaking the simulation barrier for leader election in this paper has subsequently led to the development of near optimal protocols for these communication primitives.

1 Introduction

In this paper we present the first linear time distributed algorithm for electing a leader in a radio network without collision detection, which improves over a 23 old algorithm of Bar-Yehuda, Goldreich and Itai.

Leader election, the task of nodes agreeing on the election of a single node in a network, is one of the most fundamental problems in distributed computing. It is the ultimate way to break symmetries in an initially unknown system. As such, it is a natural primitive that is used as a first step in many more high level tasks that require or benefit from having one designated "organizer". Due to its importance, leader election has been studied in many different network settings.

The setting we are interested in is radio networks. The standard model to study these networks is the radio network model presented in [7]. Here, nodes operate in synchronous rounds in which, each node can either *transmit* a logarithmic size message to its neighbors or remain silent *listening*. Only a listening node with exactly one sending neighbor receives a message while nodes with multiple transmitting neighbors only receive a collision. Depending on the model, such a collision can be detected at the receiver or not.

This interfering behavior of transmissions makes even basic communication tasks challenging. Since the introduction of model in 1985, several hundred research papers have given more and more efficient solutions to communication problems such as single-message broadcast, leader election, aggregation, multiple unicasts or broadcasts. The two first and most influential papers in this direction are [3] and [4] published in 1987 and 1989 by Bar-Yehuda, Goldreich and Itai (BGI). In the first paper [3], BGI presented Decay protocol as an efficient single-message broadcast protocol for radio networks. Since then, Decay protocol has been one of the main methods for coping with collisions of radio networks. In the second work [4], BGI use Decay protocol

*MIT, ghaffari@mit.edu
†MIT, haeupler@mit.edu

to emulate single-hop networks with collision detection in multi-hop networks without collision detection, with a slowdown factor equal to the broadcast time T_{BC}. The prime application for this emulation was to transfer results for leader election on single hop networks with collision detection to the multi-hop networks without collision detection. In particular, this allowed for simulating a leader election algorithm of Willard [24] in multi-hop networks without collision detection. This emulation approach elects a leader in expected time $O(T_{BC} \log\log n)$ rounds and in $O(T_{BC} \log n)$ rounds with high probability.

The obvious question asked by BGI [4] was whether this time can be improved. Despite lots of works, this question remained mainly unanswered except knowing the optimal complexity of each of the pieces of the emulation approach: Novel upper and lower bounds showed that in a diameter D network T_{BC} equals $\Theta(D \log \frac{n}{D} + \log^2 n) = O(n)$. Moreover, [23] showed that $\Omega(\log n)$ rounds are needed for a high probability leader election in single hop networks with collision detection. Thus, the remaining question now was about whether the whole emulation approach is optimal.

We break this simulation barrier for leader election by presenting an algorithm which parts from simulation paradigm and achieves time complexity almost T_{BC}. More precisely, this algorithm runs with high probability in $O((D \log \frac{n}{D} + \log^3 n) \times \min\{\log\log n, \log \frac{n}{D}\})$ rounds — which is in $O(n)$ rounds — on any n node network with diameter D. This is almost optimal since $\Omega(D \log \frac{n}{D} + \log^2 n)$ is a lower bound. We also give an algorithm for radio networks with collision detection. This algorithm runs in near optimal time $O(D + \log n \log\log n) \times \min\{\log\log n, \log \frac{n}{D}\}$ — which is also in $O(n)$ — almost matching the respective $\Omega(D + \log n)$ lower bound. We note that these two are the first algorithms that solve the leader election problem in essentially the time needed to broadcast one message (each in the related settings).

Aside from the complexity of leader election, there is another side to the story: Countless communication protocols in radio networks traditionally use leader election as a crucial first step to solve various, seemingly unrelated, communication tasks like multiple unicasts or broadcasts. Even though leader election seems easier than these tasks, its $O(T_{BC} \log n)$ best-known bound had become a bottleneck and had kept time complexities of these other problems unresolved as well. Our results solve this issue and set the stage for obtaining near optimal algorithms for many other natural communication primitives that rely on leader election. In particular, we resolve the complexity of a number of these primitives in a parallel work [13].

2 Related Work

The problem of leader election has received vast amount of attention under various communication models and assumptions [22]. This problem becomes considerably more challenging in the radio networks model (see e.g. [2, 4, 6, 9, 15–17, 20, 21, 23, 24]).

Single-Hop Radio Networks: The study of leader election in radio networks started with the special case of single-hop networks, where the network is a complete graph. The story goes back to 70's and 80's, when [2, 6, 17] independently showed that in the model with collision detection, the problem can be solved in $O(\log n)$ rounds deterministically, and this was shown to be optimal for deterministic algorithms by $\Omega(\log n)$ lower bound of [16]. On the randomized side of the problem in the model with collision detection, even though the expected time was improved to $O(\log\log n)$ [23, 24], the high probability time remained $O(\log n)$ in both. These bounds were proven to be tight by $\Omega(\log\log n)$ lower bound on the expected time of uniform protocols [24], and $\Omega(\log n)$ lower bound for the high probability time of uniform protocols [23]. The assumption of uniformity in the latter result was later removed [15].

In the single-hop networks without collision detection, for deterministic algorithms [9] presented matching upper and lower bounds of $\Theta(n \log n)$. For randomized bounds, [21] showed that $\Omega(\log n)$ is a lower bound on the expected time, and [18] showed that $\Theta(\log^2 n)$ is the tight bound for the high probability time by presenting $O(\log^2 n)$ upper bound and $\Omega(\log^2 n)$ lower bound.

These bounds, altogether, in principle settle the time complexity of the single-hop case.

Multi-Hop Radio Networks: In contrast to the single-hop special case, the complexity of the general case of multi-hop networks did not see much progress, after the initial results.

The research about theoretical problems in multi-hop radio networks essentially started with the pioneering papers of Bar-Yehuda, Goldreich and Itai (BGI) [3, 4]. In the first paper, BGI devised the *Decay* protocol as a solution for single-message broadcast problem, resulting in almost optimal broadcast time of $O(D \log n + \log^2 n)$. This protocol later became the standard approach in coping with collisions of radio networks (see e.g. [4, 5, 8, 12]). Provided by this almost optimal broadcast algorithm, and given that the case of leader election in single-hop radio networks was well-studied, a natural idea was to simulate the 'single-hop' leader election algorithms over multi-hop radio networks. Along this idea, in the second paper, BGI used Decay protocol to emulate a single-hop radio network with collision detection on top a multi-hop radio net-

work without collision detection. As the prime application, they used this emulation to simulate Willard's single-hop leader election algorithm [24] in multi-hop radio networks without collision detection. This resulted in time complexity of $O((D\log n+\log^2 n)\log n) = O(n\log^2 n)$ for a with high probability result (and also $O((D\log n+\log^2 n)\log\log n) = O(n\log n\cdot\log\log n)$ for expected time).

Given this efficient algorithm, the remaining question was how to improve it to optimality. One idea would be to use a better leader election algorithm of single-hop networks, but given lower bounds $\Omega(\log\log n)$ on the expected time [24] and $\Omega(\log n)$ for high probability results [23], there was no hope in that direction.

The next idea was to improve upon the Decay broadcast algorithm. By modifying the Decay protocol, Czumaj and Rytter [11] and Kowalski and Pelc [19] reduced the time complexity of broadcast from $O(D\log n + \log^2 n)$ to optimal time of $T_{BC} = O(D\log\frac{n}{D} + \log^2 n)$, known to be optimal in light of $\Omega(D\log\frac{n}{D})$ lower bound of [21] and $\Omega(\log^2 n)$ lower bound of [1]. Albeit not being published explicitly, by providing a substitute for the old Decay in BGI's framework, this new Decay changed the time complexity of leader election (using simulation approach) to $O(T_{BC}\log n) = O(n\log n)$ for a with high probability algorithm (and $O(T_{BC}\log\log n) = O(n\log\log n)$ expected time). Given that now both elements of the emulation — single-hop leader election algorithm and broadcast algorithm — were optimal, the remaining interesting question was "*can one improve upon the leader election time bound by going away from the simulation approach?*". In this paper we answer this question in affirmative.

For networks with collision detection, Kowalski and Pelc [20] presented an $O(n)$ deterministic algorithm. This highlighted the difference between models with and without collision detection as a lower bound of $\Omega(n\log n)$ was known for deterministic leader election without collision detection even for single-hop networks [9]. We remark here that the time complexity of this algorithm remains $\Theta(n)$ even when diameter D of the network is small.

3 Preliminaries

3.1 Network Models We consider the standard radio multi-hop network model [3, 4, 7]. In this model the network is represented by a connected undirected graph $G = (V, E)$ with $n = |V|$ nodes and diameter D. Communication in such a network takes place in synchronous rounds; in each round, each node is either listening or transmitting a $\Theta(\log n)$-bit packet. In each round, each node $v \in V$ can receive a packet only from its neighbors and only if v itself is not transmitting in that round. If two or more neighbors of v transmit in a round, then these transmissions *collide* at v and v does not receive any packet. In this case, we consider two models variants: (1) the model with no *collision detection* (CD) where the node v can not distinguish this collision from silence, and (2) the model with CD where v gets to know that a collision happened.

Instead of studying the radio network model with collision detection directly, we choose a strictly weaker model, namely the *beep model* as introduced in [10]. A beep network works in synchronous rounds. In each round, each node can either *beep* (transmit) or remain silent. At the end of a round, each silent node gets to know whether at least one of its neighbors was beeping or not. We note that the beep model can be seen as a radio network model with collision detection and 1-bit packets but with the additional weakening limitations that nodes can not distinguish between one neighbor sending a 1 or 0 or between the cases where exactly one or more than one neighbor is beeping. This extremely basic communication model is an interesting weakening of the standard model with collision detection, from both theoretical and practical viewpoints. Any algorithm designed for beeping model can be directly used for the standard model with collision detection. However, designing such algorithms is typically more challenging. On the practical side, it has been argued that the beeping model can be implemented easily in most environment, e.g., using extremely simple radios or carrier sensing [10].

All algorithms we study in these networks are randomized and distributed. All stated running times hold with high probability (in contrast to merely in expectation). As is standard for distributed algorithms, we assume that nodes have no knowledge about the topology except for knowing n and D (up to constant factors). We remark that the assumption of knowing D can be removed easily without any asymptotic loss in time bounds, using standard double-and-test parameter estimation techniques. We also assume, without loss of generality, that nodes have unique logarithmic size IDs.

3.2 The Leader Election Problem The problem studied in this paper is the Leader Election problem. The goal of this task is to elect a single node in the network. More formally, we say an algorithm solves the leader election problem in time $T(n, D)$ if, when run on every node in any network with n nodes and diameter D, within $T(n, D)$ rounds each node outputs exactly one ID of a node in the network and all nodes output the same ID, with high probability.

The lower bounds on the leader election problem stated next easily follow from the lower bounds on the single-message broadcast problem given in [1, 21] or those of single-hop model [15, 23]:

LEMMA 3.1. *Any algorithm requires at least* $\Omega(D \log \frac{n}{D} + \log^2 n) = \Omega(T_{BC})$ *rounds to solve the leader election problem in radio networks without collision detection.*

LEMMA 3.2. *Any algorithm requires at least* $\Omega(D + \log n)$ *rounds to solve the leader election problem in radio networks with collision detection or beep networks.*

We remark that one can alternatively define a slightly weaker version of the leader election problem that only requires exactly one node to output a 1 while all other nodes output a 0. The same lower and upper bounds hold for this problem as well.

3.3 Message Dissemination in Radio Networks: The Decay-Protocols In this section we present a recap on the Decay broadcast algorithm from [3] and [11]. This algorithm is one of the standard techniques for resolving collisions in radio networks, and as many other papers in this area, our algorithms also build on it extensively. The Decay algorithm is used to spread information present in some nodes to neighboring nodes or all nodes in a radio network without collision detection. The specific protocol we use in this paper is a tweaked version of the protocol in [11] which itself slightly speeds up the classical Decay-protocol of [3]. Our main modifications is the introduction of the delay parameter δ which allows us to control the speed of the information dissemination process. In the following, we present our variant of the Decay, and its properties. These properties are straightforward to prove or follow from [11].

DEFINITION 3.1. (DECAY(δ) ALGORITHM) *For any n and D, we define the sequence $I_{[30 \log n]}$ such that for every $j \in [10 \log n]$ we have: $I_{3j} = \log \frac{n}{D}$, $I_{3j+1} = j \bmod \log \frac{n}{D}$ and $I_{3j+2} = j \bmod \log n$. Given this sequence, we say a group of nodes, some of which have messages to send, perform r rounds of Decay(δ) during rounds t to $t + 30 \log n$ if these nodes do as follows: For every $j = 1$ to $30 \log n$, every* sender *(node with a message) transmits its message in round $t + j$ independently with probability 2^{-I_j} and any* non-sender *node, upon receiving its first message switches to becoming a sender of that message after δ rounds.*

LEMMA 3.3. *After $T \geq 30 \log n$ rounds of Decay(T), each listening node with at least one sender neighbor, receives at least one message from its neighbors, with probability at least $1/2$.*

LEMMA 3.4. *After $T \geq 30 \log^2 n$ rounds of Decay(T), each listening node with at least one sender neighbor receives at least one message from its neighbors, with high probability.*

LEMMA 3.5. ([11]) *For $\delta = \log \frac{n}{D}$, if Decay($\delta$) is run for $T = 30(D\delta + \log^2 n)$ rounds, then any node v with distance d to the closest node that is initially a sender will with high probability receive a message for the first time between round $(d-1)\delta$ and round $\min\{D\delta, d \log n\} + \log^2 n$ rounds.*

LEMMA 3.6. ([11]) *For $\delta \geq \log n$, if Decay(δ) is run for $T = 30(D\delta + \log^2 n)$ rounds, then any node v with distance d to the closest node that is initially a sender will with high probability receive a message for the first time between round $d\delta$ and round $\Theta(d\delta + \log^2 n)$.*

4 Our Results

We show the following two results:

THEOREM 4.1. *In radio network without collision detection, there is a distributed randomized algorithm that in any network with n nodes and diameter D solves the leader election problem in time*

$$T_{noCD} = O\left(D \log \frac{n}{D} + \log^3 n\right) \cdot \min\left\{\log \log n, \log \frac{n}{D}\right\}$$

THEOREM 4.2. *In beeping networks (or radio networks with collision detection), there is a distributed randomized algorithm that in any network with n nodes and diameter D solves the leader election problem in time*

$$T_{beep} = O\left(D + \log n \log \log n\right) \cdot \min\left\{\log \log n, \log \frac{n}{D}\right\}$$

5 Overview

In this section we present an overview of our leader election algorithms. All our algorithms try to implement an ideal leader election template which we describe next. The methods for implementing this template differ depending on whether one is in a setting without or with collision detection. We explain the key ideas of these implementations in Section 5.2 and Section 5.3, respectively.

5.1 Leader Election Template The main outline of our algorithms and the topmost level of their ideas are as follows.

Main Outline and the Debates: Given the number of nodes n, we first use sampling to reduce the number of possible candidates for leadership. Each node decides to be a candidate, independently, with probability $\frac{10 \log n}{n}$. A Chernoff bound then shows that with

high probability, this leads to at least one and at most $20 \log n$ candidates. To elect a leader among these candidates, we then run in phases, called *"debates"*. In each *debate*, we eliminate at least a constant fraction of candidates, while keeping the guarantee that always at least one remains. After $O(\log \log n)$ debates, only exactly one candidate remains. At the end, this candidates declares itself as the leader by broadcasting its ID. This outline is presented in Algorithm 1.

Algorithm 1 Leader Election Algorithm @ node u

1: **with probability** $\frac{10 \log n}{n}$ **do** $candidate_u \leftarrow true$ **otherwise** $candidate_u \leftarrow false$
2: **for** $i = 1$ to $\Theta(\log \log n)$ **do**
3: Debate
▷ some $candidate_u$ variables become $false$
4: **if** $candidate_u$ **then**
5: Broadcast ID_u
6: output the received ID as the leader

Clusters, Overlay Graph and Communication Actions: To achieve the above goal for debates, we need to provide some way of communication between the candidates. For this, in each debate, we grow *clusters* around each candidate, *e.g.*, by assigning each non-candidate node to the candidate closest to it. Any such clustering induces an overlay graph H on the candidates by declaring two candidates to be *adjacent* iff their clusters have borders that are close. This graph H also captures which candidates can communicate with each other using specially designed cluster communication actions. In particular, we design three *communication actions*: an *Uplink* protocol that allows a candidate to send a message to the nodes in its cluster, an *Intercommunication* protocol that allows adjacent clusters to exchange information, and a *Downlink* protocol that allows nodes in a cluster to send a message to their candidate.

We show that both creating clusters and communicating within each cluster can be done in broadcast time T_{BC} while intercommunication over borders, which is a local problem, can be solved in (poly-)logarithmically many rounds. With this one communication in H takes $O(T_{BC} + \text{poly} \log)$ rounds to implement which is already (almost) our desired final running time. We thus want an algorithm that makes sufficient progress in each debate such that only a (near) constant number of debates are needed while each debate requires only constant rounds of communication in H. We achieve this using the following *Elimination* algorithm.

Elimination Algorithm: The Elimination algorithm is a simple, deterministic algorithm which makes at least half of the candidates drop out while at least one candidate remains. This algorithm is run by candidates and as a $\mathcal{LOCAL}$ model algorithm on the overlay graph H.

Exchange 1:
Each candidate sends its ID and determines its degree by counting number of different IDs received.

Exchange 2:
Each candidate sends its ID and its degree.

Elimination:
Each candidate that is *dominated* by a neighboring candidate with a larger degree or by a neighbor with equal degree but larger ID gets *marked* for *elimination* from candidacy.

Given the guarantee of the Elimination algorithm, we can infer that a total of $\log(20 \log n)$ debates suffice to reduce the number of candidates from $20 \log n$ to one remaining leader. Since the formal statement and its proof are both simple and instructive, we present them here.

LEMMA 5.1. *The deterministic Elimination algorithm uses just two rounds of full-message exchanges in H (between candidates) and eliminates at least half of the non-isolated nodes, while keeping at least one.*

Proof. Clearly, the node with maximum $(degree(.), id(.))$ pair remains. To see that half of the non-isolated nodes are eliminated, we use a potential argument. We give a charge of one to each non-isolated unmarked node. Then we redistribute these charges by each node splitting its charge evenly between its neighbors. Since only non-isolated unmarked nodes initially get charged, and as no two unmarked nodes are neighbors, all charges gets redistributed to marked nodes. Furthermore, each marked node u gets charge of at most one, because each of its unmarked neighbors gives it a charge of at most $\frac{1}{d(u)}$, where $d(u)$ is the degree of node u. The total charge is therefore at most as large as the number of marked nodes. Since the total charge was initially equal to the number of unmarked nodes, and since the total charge did not change in the redistribution step, we get that the number of unmarked nodes is at most as large as the number of marked nodes. Thus, the number of marked nodes is at least half the total number of nodes, which completes the proof.

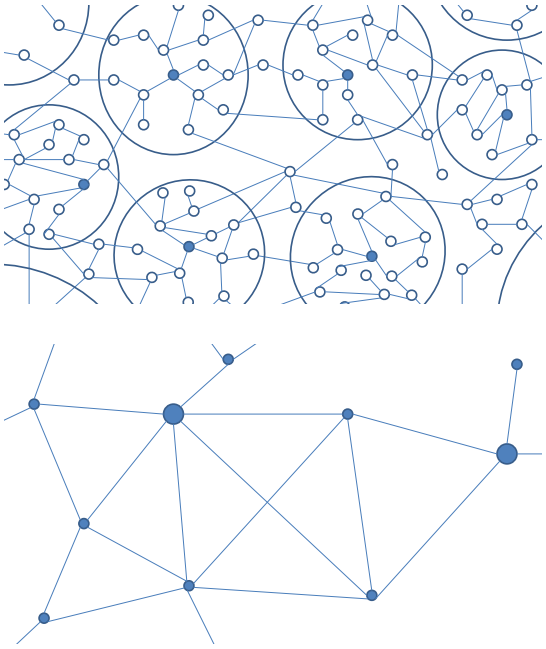

Figure 1: The first figure shows a clustered graph with solid candidate nodes. The resulting overlay graph H on the candidate nodes is depicted in the second figure. The big candidates are the ones that remain after the elimination algorithm is run on H.

Debate Template: Each debate is an implementation of the elimination algorithm on top of the overlay graph H. Given the communication primitives available atop the overlay graph, this implementation follows roughly from the outline presented in Algorithm 2. In the following sections, we describe how this debate template can be implemented in each model.

Algorithm 2 Template of a Debate

1: Cluster ▷ Overlay Design
2: Uplink candidate ID ▷ Exchange 1
3: Intercommunicate IDs
4: Downlink IDs
5: Candidate determine their degree in H by counting the number of distinct received IDs
6: Uplink pairs of $(degree, ID)$ from candidates ▷ Exchange 2
7: Intercommunicate the pairs
8: Downlink the pairs
9: Candidates remains iff their $(degree, ID)$ pair is greater than all received pairs ▷ Elimination

5.2 Implementation of a Debate Without Collision Detection Here we present the main ideas for how to implement the aforementioned debate templates in the radio network model without collision detection. The goal is to run one debate in $O(T_{BC} + \log^3 n)$ rounds and thus obtain the leader election algorithm claimed in Theorem 4.1.

Clustering: There are two different ways to use the Decay-Broadcast for building the clusters. One is to simply run a global broadcast in time T_{BC} with the candidate IDs as messages. If every node simply keeps and forwards the first ID it receives, in the end, every node belongs to a cluster, and also the clusters are connected. Unfortunately, these clusters do not have nice shapes and do not allow for efficient intercommunication between clusters. This problem can be avoided using the second way which uses a slower variant of Decay: we repeatedly use $T = \Theta(\log^2 n)$ rounds of Decay(T) to grow the clusters by one step, every time. Each time all unclustered nodes with a clustered neighbor get included in the cluster with high probability. This leads to nicely shaped clusters in which each node joins the closest candidate. But, the running time of this method is unfortunately $\Theta(T_{BC} \log^2 n)$ rounds, which we cannot afford.

Our solution is to combine these two methods to get the best of both worlds. For this we start with a fast-growth phase in which we use the first method to advance the clusters in iterations of $\Theta(\log^2 n)$ rounds. After each iteration we ensure that clusters do not interfere with each other, by cutting them back (trimming them) if they do. After the fast-growth phase which takes $O(T_{BC})$ rounds, clusters are at most $\Theta(\log n)$ far away from each other. Now we use the slower second method to grow the clusters carefully spending $\Theta(\log^2 n)$ rounds for each of the remaining $\Theta(\log n)$ steps. This gives us a nice clustering for a total of $\Theta(T_{BC} + \log^3 n)$ rounds.

Overlay Communication: Due to the nice clustering, the overlay communication routines for intercommunication and uplink can be easily implemented in $\Theta(\log^3 n)$ and $\Theta(T_{BC})$ rounds, respectively. Unfortunately, implementing a downlink is more troublesome. The subtle reason is that while there are at most $20 \log n$ distinct IDs of neighboring clusters that need to be collected in each candidate, there are copies of each of these IDs registered at up to $O(n)$ different cluster nodes. This prevents classical gathering protocols (e.g., [5]) to work for this task.

To remedy this, we use a broadcast algorithm, with time complexity $O(T_{BC})$, within each cluster to inform the candidate about just one of its neighbors. After this, the candidate can use the uplink to give feedback to all cluster nodes that it has received this particular

ID, again in broadcast time of $O(T_{BC})$ rounds. This guarantees that after this, only nodes with a new piece of information will participate. Repeating this r times results in at least r distinct IDs being learned by the candidate in $O(T_{BC}\ r)$ rounds. While this is an improvement over the naive gathering (which would take $\Theta(n)$ rounds), it still takes a prohibitively large $\Theta(T_{BC} \log n)$ rounds to learn about all neighboring clusters. Next, we show how to work around this issue by modifying the elimination algorithm.

The Modified Elimination Algorithm: Our modifications to the elimination algorithm are based on the following two ideas: First, we run the elimination algorithm not directly on the overlay graph H but instead on a sparse subgraph H' of H. Secondly, we modify the elimination algorithm such that a node needs to be aware of at most 6 of its neighbors, instead of all $O(\log n)$ of them which was required in the original elimination algorithm.

To carve out the sparse sub-graph H', each cluster selects one edge to a neighbor and we define H' to be the sub-graph consisting of the union of these edges. Note that while the average degree of a node is at most 2 in this graph, this does not hold true for the maximum degree. Nonetheless, using the inward-communication scheme explained above with $r = 5$, each candidate can learn about all of its neighbors if it has at most 5 of them, and at least detect that it has five or more neighbors otherwise. With this knowledge, we run the same elimination algorithm as before except for the modification that, now nodes with degree of at least 5 remain unmarked, This is because they cannot safely determine whether their degree is dominated by a neighbor. Note that there is at most a $\frac{2}{5}$-fraction of the nodes with degree of at least five since more would lead to an average degree of more than 2. The Modified Elimination Algorithm therefore still eliminates at least a $\frac{1}{2} - \frac{2}{5} = \frac{1}{10}$ fraction of the candidates, while remaining safe.

5.3 Implementation of a Debate via Beeps In this section we describe the main ideas for implementing a debate in the beep model (or radio network model with collision detection). Our algorithm works along the lines of the debate template presented in Section 5.1: it first clusters the nodes and then uses overlay communication protocols to run the elimination algorithm.

We first introduce our main tools, *beep waves* and *superimposed codes*, and explain how to use them to cluster the graph and implement the overlay communication protocols mentioned in Section 5.1. We then put everything together and present a simple debate implementation that runs in $O(D+\log^3 n)$ rounds. Lastly, we show how to achieve the running time claimed in Theorem 4.2 by modifying the simple debate implementation to run in $O(D + \log n \log\log n)$ rounds.

Beep Waves: The main difference between radio models without collision detection and those with collision detection (or beeping) is the ability to create what we call *beep waves*. Beep waves start at one or more nodes by sending a beep, and after this initiation, each node that hears a beep forwards it by beeping in the following round. This way, the beep propagates in the form of a wave throughout the network, moving exactly one step per round.

Beep waves have several applications. For one, they can be used to determine the distance of a node u from a (beep) source, by measuring the number of rounds for the wave to reach u. Secondly, pipelining multiple beep waves from a source can be used to transmit bit strings, by coding 1 into a beep and 0 into absence of a beep. Pipelined beep waves will be our main tool in implementing the communication protocols used in our leader election algorithms of the beep model.

Superimposed Codes: Another interesting feature of using beep waves to transmit information is that, when two different sources s_1 and s_2 simultaneously send different bit strings to one node v with equal distance from s_1 and s_2, then v receives the superimposition or bitwise OR of the two strings. Typically, such bit string is considered useless. Thus, protocols designed for radio networks so far have mostly focused on using collision detection and randomization to detect and avoid collisions. In this paper, we take the exact opposite stance: instead of avoiding collisions, we propose to embrace them and leverage their superimposition nature. The key element is superimposed codes. These codes consist of codewords that allow any superimposition of a bounded number of codewords to be decomposed and decoded. Good superimposed codes can be easily shown to exist by a simple probabilistic analysis of certain random codes. Using these superimposed codes for communication in beep networks (or radio networks with collision detection) is to our knowledge a novel[1] approach which we think will be useful in the future research as well. Interestingly, the construction, existence and use of superimposed codes itself turns out to be a very old concept from the 40's, e.g., where they were used as an efficient way to use punch-card archival systems.

[1] A connection between superimposed codes and radio networks has also been established in [8], where superimposed codes were seen as selective families, to schedule successful transmissions by avoiding collisions. As such, their use is completely different from our approach of using intentional collisions for coded communication.

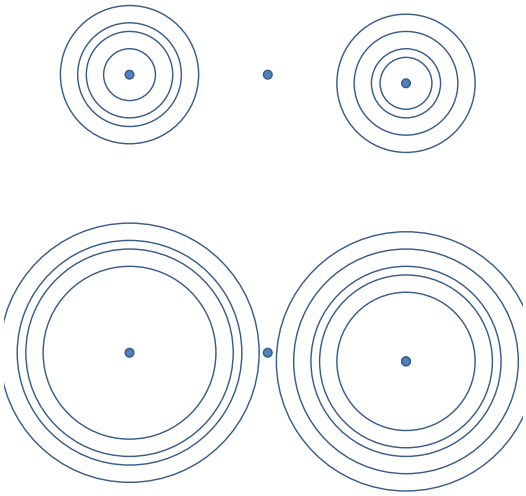

Figure 2: Pictorial example of two nodes sending a bit string encoded in beep waves. The left node sends the bit pattern 101101 while the node on the right sends 101011. The second figure shows the propagated waves at a later time. The node in the middle will receive the superimposition of the waves and thus receive the bit pattern 101111. If a superimposed code is used and the aforementioned bit patterns are the results, the middle node would be able to decode the received bit pattern 101111 into two separate initial bit patterns 101101 and 101011

Clustering and Overlay Communication: To *cluster* nodes around candidates, we assign each node to the closest candidate if there is a unique such candidate, and leave all other nodes unclustered. To this end, we first use beep waves from each candidate to determine for each node its distance to the closest candidate. Using these distance numberings, we then send out the IDs of the candidate nodes as pipelined beep waves. To prevent nodes confusing superimposition of IDs with clean IDs, we use a superimposed code that allows to distinguish whether a received bit string is a coded ID or the superimposition of multiple coded IDs. This allows nodes with more than one closest candidate to stay unclustered while nodes with a unique closest candidate join the cluster of that candidate. This clustering also induces the overlay graph H.

Next, we implement our uplink, intercommunication, and downlink *communication protocols* that allow communication atop the overlay graph H. We implement these by using the distance numbering to synchronize pipelined beep waves for communication. This is relatively straightforward but leads to both downlink and intercommunication not actually delivering all messages from all clustered nodes or neighboring clusters respectively, but instead delivering the superimposition of all these messages. This is where superimposition codes show their full power. Instead of sending messages directly in the intercommunication phase, we use messages coded with a superimposition code. When a candidate then receives the messages from its adjacent candidates, in superimposition after they got combined in the intercommunication and downlink phase, it can still fully reconstruct all original messages. This way superimposition codes allow us to implement a full local message exchanges in H even though the actual intercommunication and downlink protocols merely deliver superimposition.

Implementing the Elimination Algorithm Optimally: Given the full message exchange overlay communication, one can directly implement the debate template of Section 5.1. The running time of such a debate would be $O(D+\log^3 n)$. Here the $\log^3 n$ comes from the length of superimposed codes that allow to decode messages from the superimposition of up to $\log n$ codewords which we use to implement the full message exchange over H. In the remainder of this section we show how we can improve over this to achieve an $O(D+\log n \log\log n)$ debate. This completes the proof for Theorem 4.2 and provides a leader election algorithm that is optimal up to a $\log^2 \log n$ factor.

The main observation that leads to the speedup is as follows: The $O(D + \log^3 n)$ debate implementation does not actually use the full power of message exchange that is given by the use of the $\log^3 n$ long superimposed codes. Instead, we eventually use this communication only for two tasks: (1) determining the number of different messages received; used for determining the degree on the overlay graph, and (2) checking whether there is a neighbor with a larger $(degree, id)$ string; used to decide whether a node marks itself in the elimination algorithm. We show that both tasks can be achieved with a smaller overhead.

For (1), i.e., determining the degree, we design a new set of codes that are just strong enough to enable us to estimate the degree of each candidate up to a multiplicative factor of 2. These new codings encode each ID of length $\log n$ into a codeword that is only $\Theta(\log n \log\log n)$ bits large. We then show that eliminating candidates with the Elimination algorithm based on these approximate degrees still works, that is, still removes a constant factor of candidates per debate while keeping at least one. This can be easily checked by following the same potential argument as proof of Lemma 5.1, noting that now, each remaining node gets

charge of at most 4. With this the number of remaining candidates is at most 4 times larger than the number of marked candidates. In other words, still at least $\frac{1}{5}$ of the candidates get removed.

For usage (2), i.e., detecting whether a neighboring candidate has a message numerically larger or not, we use a slightly different intercommunication algorithm. In this new intercommunications, nodes go through the bits of messages of their candidates one by one and compare them. They *mark* themselves if they detect a larger message in the neighborhood. A node that gets marked does not continue the process anymore. It is easy to see that a node gets marked in this procedure if there is an adjacent cluster with a larger message. Finally, to deliver this information to the candidates, we simply use a Downlink with single-bit messages (marked or not) and each candidate gets to know whether any of nodes in its cluster is marked or not.

5.4 Obtaining Linear Time Leader Election Algorithms In this section, we explain a simple optimization which reduces the multiplicative $\log\log n$ factor in our bounds for networks with (near) linear diameter. In particular this optimization makes all our running times $O(n)$.

While we do not know how to reduce the number of debates below $\Theta(\log\log n)$, we show that less time can be spent on initial debates. That is, if there are many candidates, we can work with clusters that have a smaller diameter. In particular, we note that in the both leader election algorithms presented above, in each debate, the time selected for growing clusters is chosen large enough such that the radius of each cluster can potentially grow up to D. This is done to avoid isolated clusters which never get eliminated in the Elimination Algorithm (see Lemma 5.1). Furthermore, the time needed for a debate depends directly on this *radius of growth*. In particular, in the model without collision detection, growing clusters up to radius of d takes $O(d\log\frac{n}{d}+\log^3 n)$ and in the beeping model, it takes $O(d+\log n)$.

The key observation is as follows: if the number of remaining candidates is k, then at most half of the candidates can be such that they have no other candidate within their $2n/k$ distance. Because of this, the idea is that in the i^{th} debate, instead of building clusters for radius up to D, we grow the clusters for radius only up to $\min\{D, \frac{4n\ 1.05^i}{\log n}\}$. This still ensures that at least half of the candidates are non-isolated, which allows the Elimination Algorithm to remove at least a constant fraction of all (non-isolated) nodes. It is an easy calculation to see that this change in radius of growth reduces the $\log\log n$ factor in our time bounds to a $\min\{\log\log n, \log\frac{n}{D}\}$ factor, as claimed in Theorem 4.1 and Theorem 4.2. We defer the details to the full version.

6 Leader Election without Collision Detection

This section is devoted to providing the technical details and proofs for Theorem 4.1. As described in Section 5, this algorithm follows the template given in Section 5.1 and uses the ideas explained in Section 5.2 to implement this template[2]. In particular this section gives the clustering algorithm in Section 6.1, shows how to obtain the sparsified overlay graph H' and perform overlay communication protocols on top of it in Section 6.2 and lastly explains in Section 6.3 how to implement a debate in $O(T_{BC}+\log^3 n)$ time. Together with the optimization described in Section 5.4 this proves Theorem 4.1.

6.1 Clustering In the clustering phase, we partition the network into disjoint *clusters*, one around each candidate, such that these clusters provide a platform for easy communications between candidate nodes.

Formally, a *clustering* is a partial assignment which for each node v, it either *clusters* v by assigning it to (the *cluster* of) a candidate, or it leaves v *unclustered*. Given a clustering, we say a clustered node v is a *boundary* node if v has a neighbor u that is unclustered or belongs to a different cluster. Otherwise, we say the clustered node v is an *internal* node. However, as a small exception, candidates themselves are considered as internal always. We will at all times preserve the invariant that each clustered node is connected to the candidate it is assigned to via a path of internal nodes of the same cluster. For a distance d, we say two clusters $\mathcal{C}_1$ or $\mathcal{C}_2$ or their respective candidates are *d-adjacent* if there are internal nodes $v_1 \in \mathcal{C}_1$ and $v_2 \in \mathcal{C}_2$ such that v_1 and v_2 are within distance at most d of each other. This notion of d-adjacency defines the *candidate* graph H_d. We say a clustering has *connectivity gap* at most d if graph H_d has no isolated nodes (or exactly one isolated node if that's the only node of graph), i.e., if each cluster is d-adjacent to at least one other cluster.

We remark that the time complexity of the intercommunication task on a clustering is monotonically increasing with (and almost directly proportionally to) the connectivity gap of that clustering. Thus, we desire our clustering routine to produce a clustering with small connectivity gap. In particular, we achieve a clustering with constant connectivity gap. Such a clustering is then used for building the desired *overlay* graph H' and

[2]The reader is advised to read both Section 5.1 and Section 5.2 first.

for communication over H'. We see that this constant connectivity gap translates to a $O(\log^3 n)$ intercommunication time complexity.

LEMMA 6.1. *Given a set of candidates, there exists a distributed algorithm in the model without collision detection that with high probability achieves a clustering around candidates with connectivity gap at most* 10*, in* $O(D \log \frac{n}{D} + \log^3 n)$ *rounds.*

To achieve a clustering with constant connectivity gap, we start from a trivial clustering with large connectivity gap and then reduce the gap by refining the clustering using the following algorithm:

LEMMA 6.2. *Consider* $d' \in [10, \frac{\log^2 n}{\log \frac{n}{D}}]$ *and* $d > d'$*. Given a clustering with connectivity gap at most d, the algorithm Cluster(d,d′) runs in* $O(\frac{d+d'}{d'} \log^2 n)$ *rounds and produces a clustering with connectivity gap at most d′, with high probability.*

Before going into the details of the Cluster algorithm, let us assume that we have algorithm Cluster(d,d') as a black box and conclude the proof of Lemma 6.1 by showing how to use it to achieve a clustering with constant connectivity gap.

Proof. [Proof of Lemma 6.1] We start with the trivial clustering in which clusters only include the candidates, i.e., all candidates are clustered (and internal) while all other nodes are unclustered. Clearly, this initial clustering has connectivity gap at most D. We then choose $d' = \frac{\log^2 n}{\log \frac{n}{D}}$ and run Cluster(d, d') to achieve a clustering with connectivity gap at most $d' = \frac{\log^2 n}{\log \frac{n}{D}}$. This process takes $O(D \log \frac{n}{D} + \log^2 n) = O(T_{BC})$ rounds. To obtain a clustering with connectivity gap at most $d'' = 10$, we further improve upon this, using Cluster(d', 10) algorithm. This part takes $O(\frac{\log^4 n}{\log \frac{n}{D}})$ rounds. Finally, note that the total complexity of the whole procedure used above is $O(D \log \frac{n}{D} + \log^2 n) + O(\frac{\log^4 n}{\log \frac{n}{D}})$ which is equal to $O(D \log \frac{n}{D} + \log^3 n)$.

Cluster(d,d') Algorithm: This algorithm consists of $330 \log^2 n \, \frac{d+d'}{d'}$ rounds, which are divided into epochs of $330 \log^2 n$ rounds each. At the beginning and end of each epoch, we will have a valid clustering in which each clustered node knows the ID of the candidate it is assigned to. Also, in these valid clusterings, each node is either "internal", "boundary" or "unclustered". During each epoch, instead of the status "boundary", we have a different status called *"undecided"*. At the beginning of each epoch, we first change the status of every "boundary" node to "undecided". Then, only candidates and the unclustered and "undecided" nodes participate in the transmissions of that epoch. Note that candidates always remain internal, regardless of what happens.

Each epoch consists of four steps. In a nutshell, the first step is for growing the clusters whereas the other three steps are for refining the shape of the newly grown parts of the clusters. In the first step, we grow the clusters creating more "undecided" nodes. Then, in the second and third steps, we *mark* some "undecided" nodes using a particular rule and we use these marks in the fourth step to determine the new statuses. The details of these steps are as follows.

Step 1: This step consists of $4 \cdot (30 \log^2 n)$ rounds. Let $\delta = \frac{300 \log^2 n}{d'}$. In these rounds, we grow the clusters using the Decay(δ) protocol. For this, each candidate and each "undecided" node starts with the message of the related candidate. With these messages, we run $4 \cdot (30 \log^2 n)$ rounds of the Decay(δ) protocol. In these rounds, some unclustered nodes receive the message of one or possibly more candidates. Each node ignores all the received messages but the first one. Then, each unclustered node that received some message temporarily joins the cluster of the candidate whose ID is mentioned in that message and changes its status to "undecided".

Step 2: Here, we *mark* any "undecided" node that is adjacent to either an unclustered node or a node from a different cluster. We do this in $2 \cdot (30 \log^2 n)$ rounds In these rounds, each unclustered node simply runs a Decay($30 \log^2 n$) protocol, sending message declaring that they are unclustered. On the other hand, for clustered nodes, these rounds are divided into $2 \log n$ parts and in each part, a subset of clustered nodes are active. More precisely, in each part, nodes of each cluster unanimously decides on either being active or listening, each with probability $\frac{1}{2}$. Note that this unanimous cluster decision can be achieved by each candidate sharing $2 \log n$ bits of randomness with nodes of its cluster by attaching these randomness bits to its ID (so the size of the messages remain asymptotically the same). In each part, all active nodes then perform $T = 30 \log n$ rounds of Decay($30 \log n$), sending their cluster ID. All clustered nodes that receive a message different than their own cluster ID, and all unclus-

tered nodes that receives a cluster id become marked.

Step 3: In this step we mark "undecided" nodes that were (indirectly) recruited in Step 1 by node which got marked in Step 2. To make this more precise, we say a node v was *directly recruited* by node u if in Step 1, the first message that node v received (which is also the message that resulted in the status of v to become "undecided") was received directly from node u. Similarly, we say node v was *directly recruited* by node u in Step 1 if the first message that node v received came from a node that was directly or indirectly recruited from u. To achieve the marking of nodes that got (indirectly) recruited by marked nodes we repeat the exact same transmissions of Step 1 but now, each node transmits a bit indicating whether it is marked instead of an ID. Any node that receives a set bit from the node that recruited it in Step 1 becomes marked.

Step 4: In this step we determine the final statuses. For this, all non-marked "undecided" nodes become "internal". Then, we run $T' = 30\log^2 n$ rounds of Decay(T') and all non-internal nodes that receive a message become "boundary". Lastly, we set the status of any remaining "undecided" node to be unclustered.

Proof. [Proof of Lemma 6.2] Consider candidate node u and its cluster $\mathcal{C}_u$ after running the algorithm. To prove the lemma, we show that: (1) there is another cluster $\mathcal{C}_{u'}$ within distance d' of $\mathcal{C}_u$, and (2) each clustered node v of $\mathcal{C}_u$ is connected to u via a path made of only internal nodes of $\mathcal{C}_u$.

Let $\delta = \frac{300\log^2 n}{d'}$. For the first property, note that if node v was labeled "undecided" during the first $2\cdot(30\log^2 n)$ rounds of Decay(δ) in Step 1, then v becomes "internal" unless the cluster growth interferes with the growth of another cluster. The reason for this is as follows. Consider a node w that was labeled "undecided" in these rounds. Then, Lemma 3.4 shows that in the absence of other clusters, during the last $2\cdot(30\log^2 n)$ rounds of Decay(δ), any node that is within distance 2 of w will receive a message, and thus become clustered. After that, only nodes that are at distance exactly one or two from w will get marked in Step 2, while w does not get marked. Thus, w remains clustered and becomes internal.

Now the above fact shows that, unless a cluster gets close to another cluster, in every epoch, the growth of the cluster dominates the growth of a regular Decay(δ) broadcast that is run for $2\cdot(30\log^2 n)$ rounds. Given this, we get that running $(d\delta + \log^2 n)$ epochs of clustering corresponds to at least $60(d\delta + \log^2 n)$ rounds of Decay(δ) with $\delta > \log\frac{n}{d}$, unless the cluster get close to another cluster. From Lemma 3.5 and Lemma 3.6, we then get that unless there is any candidate other than u, with high probability all nodes belonging to the cluster $\mathcal{C}_u$.

Now let us consider the more interesting case where there are at least two candidates remaining. From above discussions, we conclude that during a complete run of Cluster(d,d') there is a round in which one "undecided" gets marked for neighboring a node from another cluster in Step 2. Suppose that i is the first epoch such that there exists a node $w \in \mathcal{C}_u$ that is marked during step 2 of epoch i. Then, in that step, there exists a cluster $\mathcal{C}_{u'} \neq \mathcal{C}_u$ and a node $w' \in \mathcal{C}_{u'}$ such that w and w' are neighbors. Since by design of the delay parameter δ, in each epoch each cluster can grow at most $\frac{120\log^2 n}{\delta}$ hops, in trimming part of epoch i (steps 2 to 4), each cluster only backtracks by at most $\frac{120\log^2 n}{\delta}$ hops. But then, at the end of trimming, the internal statuses are permanent and thus, are not altered later. Hence, at the end of epoch i, there exist two nodes v and v' that are within distance $\frac{240\log^2 n}{\delta}$ of each other and are permanently assigned to $\mathcal{C}_u$ and $\mathcal{C}_{u'}$ as internal nodes, respectively. This completes the proof of the property (1).

For property (2), we first show that every internal node has a path of internal nodes connecting it to the related candidate. This is true initially and remains true because the only way a node v becomes internal is if it receives a message in Step 1 over a sequence of transmissions starting from a previously boundary node. Furthermore, for the node v to become internal, none of the nodes that recruited v directly or indirectly can be marked in Step 2. This is because otherwise v would also get marked in Step 3. Once node v becomes internal, all nodes connecting it to a node that was internal before (all those that recruited v directly or indirectly) are getting an internal status too, and this preserves the invariant. Now, it is easy to see that property (2) also holds for boundary nodes created in Step 4 since these boundary nodes are recruited only by internal nodes.

6.2 Constructing of the Sparsified Overlay Graph H' and Overlay Communication In this component, we use the clusters obtained in the clustering component to design a directed *overlay* graph between the candidates. We get an *overlay* with following properties: (1) each candidate has exactly one incoming edge. If (w, v) is the edge going from w to v, we say that w is the *parent* of v, and v is a *child*

of w, (2) each candidate can send a message to all its children, in one round of communication atop the overlay (3) each candidate knows all of its children, if there are at most 5 of them, and it knows at least 5 of them, if there are more, and can receive from these children in a constant number of rounds of communication atop the overlay, (4) each round of communication atop this overlay takes $O(D \log \frac{n}{D} + \log^3 n)$ rounds, and also (5) this overlay is built in time $O(D \log \frac{n}{D} + \log^3 n)$.

In this overlay, we have three types of *communication actions* in or amongst the clusters. These communication actions are as follows. (**a**) Uplink: a candidate delivers its message to all internal nodes of its cluster. (**b**) Intercommunication: internal nodes exchange messages with other internal nodes of other clusters which are at distance at most 10. (**c**) Downlink: internal nodes send some messages towards the candidate; we guarantee that candidate gets at least one of them. We explain the implementation details of these communication actions later.

Implementation of Overlay atop communication actions: Above these abstractions of communication actions, the algorithm for designing the overlay to get the aforementioned properties (1) to (5) is as follows. First, candidates send their id to all internal nodes of their cluster using the Uplink. Then, we use intercommunication so each internal node knows the id of clusters that are within distance 10 of it. Then, internal nodes use the down-link to send the id of these adjacent clusters to their respective candidates. For each candidate u, the first adjacent candidate that u hears about becomes parent of u. Then, using an up-link communication, the candidates send the id of their parent to the internal nodes of their clusters. After that, we use another intercommunication where internal nodes inform close-by internal nodes of who their parents are. Hence, after this, internal nodes of each cluster, altogether, know which clusters are their children. Now, the goal is to get this information, about all up to 5 of those children, to the candidate. For this, we simply use 5 turns of the down-link and up-link where in each turn, candidate asks "*So far I know about children listed as follows =[...]. Tell me something new*". This way, a candidate gets to know all up to 5 of its children. Using Uplinks and Intercommunications and then, with the exact same transmissions as done in the above down-links, we can communicate from each candidate to its children, and also back from all up to 5 of its children to the candidate itself.

Implementation of communication actions: In the following, we explain how we implement the communications actions Uplink, Intercommunication, and Downlink and what are the time complexities of these actions.

1. In uplink, candidates start with messages for transmission. Every internal node that receives a message runs a Decay($\log \frac{n}{D}$) algorithm. However, the boundary or unclustered nodes do not participate in transmissions. It is easy to see that after broadcast time T_{BC}, each internal node receives the message of the respective candidate. Thus, the complexity of uplink is simply $T_{BC} = O(D \log \frac{n}{D} + \log^2 n)$.

2. In downlink, we do exactly the opposite of uplink. This time, some internal nodes start with messages and we want to deliver at least one of these messages to the related candidate. Again, every internal node (other than candidates) that has a message (or receives a message) runs a Decay($\log \frac{n}{D}$) algorithm, and the boundary or unclustered nodes do not participate in transmissions. Again, by properties of Decay($\log \frac{n}{D}$) algorithm, after T_{BC} time, the candidate receives the message of at least one internal node of the related cluster. For the sake of cleanness, the candidate can ignore all but the first received message. Clearly, the complexity of downlink is also $T_{BC} = O(D \log \frac{n}{D} + \log^2 n)$.

3. The intercommunication consists of $\Theta(\log^3 n)$ rounds which are divided into $\Theta(\log^2 n)$ epochs. In each epoch, all internal nodes of each cluster randomly decide to be active or to be listening unanimously, with probability $\frac{1}{\log n}$ for being active[3]. Then, in each epoch, internal nodes of the active clusters and all the unclustered or boundary nodes that receive a message perform $300 \log n$ rounds of Decay($\log n$) algorithm, based on their own local coins. Let us say a cluster is globally isolated in an epoch if this cluster is the only cluster that is active in that epoch. Note that, in expectation, each cluster $\mathcal{C}_u$ becomes globally isolated in $\Theta(\log n)$ epochs. Thus, using Chernoff bound, w.h.p., $\mathcal{C}_u$ becomes globally isolated in at least $\Theta(\log n)$epochs. Then, from properties of Decay protocol, we know that in each such epoch, each (internal) node of any other cluster that is within distance 10 of internal nodes of $\mathcal{C}_u$ receive message of $\mathcal{C}_u$ with constant probability. Thus, $\Theta(\log n)$ epochs, each such node receives

[3]This requires $\log^2 n$ bits of randomness shared in the cluster. We can get this shared randomness in time $O(D \log \frac{n}{D} + \log^3 n)$ by sending $k = \log n$ packets of $\log n$ random bits each via a very simple single source k-message broadcast algorithm from [14] with time complexity $O(D \log \frac{n}{D} + k \log^2 n)$.

this message with high probability. Then, using union bounds, we get that this holds for any such node, and also in another level, for every cluster $\mathcal{C}_u$. This proves the correctness of the intercommunication algorithm.

6.3 Implementing the Modified Elimination Algorithm As the last component of a debate (after clustering and overlay design), we implement the Modified Elimination Algorithm (MEA) on our overlay graph. Each candidate knows whether its degree is less than 5 or not, and in the former case, the node also knows the degree exactly. Thus, each node knows its degree rounded down to 5. Using the overlay graph, each candidate sends this *flattened* degree and its id, first, to its children, and then, to its parent. Given property (2) of overlay, sending this message to children is straightforward. On the other hand, using property (3) of overlay, we know that each candidate receives the message of all up to 5 of its children. After these message exchanges, each candidate uses the MEA algorithm to decide whether it remains alive or becomes removed from candidacy. If a candidate v has a degree higher than 5, it remains alive. Otherwise, noting the aforementioned properties (2) and (3), v receives all the messages it needs. Thus, v can compare its $(degree(.), id(.))$ pair with the pairs in the received messages and decide about remaining alive or being removed, accordingly.

Proof. [Proof of Theorem 4.1] What remains to show is that the above implementation of the Modified Elimination Algorithm on top of the overlay graph satisfies the desired properties of a debate, as mentioned at the start of the section. Note that if the number of remaining candidates is greater than one, the overlay graph might be disconnected. For the purpose of analysis, we look at each connected component of the overlay graph separately. If there is only one candidate remaining in the network, we are done with the proof. Otherwise, we know that each connected component has more than one candidate (assuming more than one candidates are remaining). Hence, Modified Elimination Algorithm removes at least $\frac{1}{10}$ fraction of the nodes of each connected component which means that it removes at least $\frac{1}{10}$ fraction of all the candidates. Also, it is clear that always at least one candidate remains alive. This is because one candidate remains alive in each connected component of the overlay graph. These two show that the two desired properties of the debates are indeed satisfied.

7 Leader Election via Beeps

This section provides the technical details and proofs for Theorem 4.1. As described in Section 5, this algorithm follows the template given in Section 5.1 and uses the ideas explained in Section 5.2 to implement this template. The reader is advised to read Sections 5.1 and 5.2 before reading this section.

In this section, we present a leader election algorithm for the beep model, which has time complexity $O(D + \log n \log\log n) \cdot \log\log n$ rounds. The outline of this algorithms is the same as the one presented in Section 5 as Algorithm 1. Starting with $\Theta(\log n)$ candidates the algorithm runs in $\Theta(\log\log n)$ debates. In each debate, we reduce the number of remaining candidates by a constant factor, while keeping the guarantee that at least one candidate remains. Each debate consists of a clustering phase and then an implementation of a debate using the induced overlay graph and overlay communication protocols. In this section we describe these implementations. For this we first introduce superimposed codes in Section 7.1. Then in Section 7.2 we give a simple implementation of a debate in $O(D + \log^3 n)$ rounds and then finally in Section 7.3 we show how to improve this to $O(D + \log n \log\log n)$ rounds. This running time for a debate leads to the (near) optimal leader election algorithm promised in Theorem 4.1.

7.1 Superimposed Codes In this part we define the two types of superimposed codes used in our algorithms and show their existence.

Definition 7.1. *A k-superimposed code or $SI(k)$-code of length l for a finite set N assigns each element in N a binary codeword of length l such that (1) every superposition of k or less codewords is unique and (2) every superposition of more than k codewords is different from any superposition of k or less codewords.*

It is easy to see that good superimposed codes of short length exist:

Lemma 7.1. *For every N and any k there exists a $SI(k)$-code for N of length $l = 4(k+1)^2 \log N$.*

Proof. We show that a random code C in which each codeword position is set to one with probability $p = 1/(k+1)$ has the desired properties with good probability. To see this we take any $k+1$ codewords $c_0, c_1, \ldots, c_k$ and note that the probability that there is a position for which c_0 is one and all codewords $c_1, \ldots, c_k$ are zero is exactly $(1 - p(1-p)^k)^l$ which is at most

$$(1 - 1/e(k+1))^l < (1/e)^{(k+1)\log N} < N^{-(k+1)}.$$

Taking a union bound over all $\binom{N}{k+1} < N^{k+1}$ choices of codewords we get that with probability at least one we get the property that the superposition of any k codewords differs from any different codeword c_0 by

having a zero where c_0 has a one. In particular this implies that given two sets S and S' with $S' \geq |S|$, $|S| \leq k$ and $S \neq S'$ the superposition of all codewords in S' has a one on a position in which the superposition of all codewords in S does not. This is true because S' contains at least one codeword c_0 that is not in a superset of S of size k and making S smaller and S' larger does not change this fact. It is easy to see that both property (1) and (2) now follow directly.

We also use the following approximate counting superimposed code:

DEFINITION 7.2. *For any N and any $k < N$ an approximate k-counting superimposed code of length l consists of a distribution D over binary codewords of length l and a decoding function decode : $\{0,1\}^l \to [k]$ such that for every $j \in [k]$ and $c_1, \ldots, c_j$ independently sampled from D we get that:*

$$P\left[\; j \;\leq\; decode\left(\bigoplus_{i=1}^{j} c_i\right) \;\leq\; 4j \;\right] \geq 1 - 1/N.$$

As the next lemma shows their length is only logarithmically dependent on k and N.

LEMMA 7.2. *For any N and any $k < N$ there exists an approximate k-counting superimposed code of length $l = \Theta(\log N \log k)$.*

Proof. Each codeword in the distribution D of the code we are constructing consists of $\log k$ blocks of $\log N$ bits where each bit in the ith block b_i for $i \in [\log k]$ is independently set to one with probability 2^{-i}. The decoding function d takes a binary word of length l dissects it into its blocks and outputs 2^{i-1} where b_i is the first block in which less than a $0.9 \cdot (1-(1-2^{-i})^{2^{i-1}}) \approx 0.9/\sqrt{e}$ fraction of the block bits are one. To show that this works we note that in expectation exactly a $1-(1-2^{-i})^j$ fraction of block b_i is ones and for a large enough block length $\Theta(\log n)$ a Chernoff bound shows that the probability that this fraction deviates by 0.9 or more is at most $1 - 2/N^3$. A union bound over all $k < N$ values for j and all $\log k < N$ values for i then shows that the probability of a too small estimation is at most $2/N$. The analog argument also shows that an overestimation happens at most with probability $2/N$ which completes the proof.

7.2 $O(D+\log^3 n)$-length debate The outline of this debate algorithm is exactly that of the simple debate algorithm sketched in Section 5.1. We first grow clusters around candidates, then each candidate find its degree in the overlay graph, then candidates exchange their (*degree*, *id*) pairs, and at the end, each candidate remain a candidate only if its pair is greater than all the pairs that it received. Next, we zoom in on how we implement each of these steps with beeps.

Algorithm 3 Debate-1 Algorithm, run @ node u

```
1:  Cluster                                                    ▷ Step 1

2:  Uplink coded ids                                           ▷ Step 2
3:  Intercommunicate
4:  Downlink

5:  if candidate_u then
6:      S_u ← decoding of received message as set of ids
7:      s_u ← |S_u|
                                                               ▷ Step 3
8:  Uplink C(s_u, ID_u)
9:  Intercommunicate
10: Downlink

11: if candidate then                                          ▷ Step 4
12:     T ← decoding of received messages as set of ordered pairs of degree and id
13:     if received a pair greater than that of u then
14:         candidate ← false
```

Clustering: For clustering nodes via beeps, we assign each node to the cluster of the candidate which is closest to it. In the case of a tie — where there are more than one closest candidates — we leave the node as unclustered.

To achieve this clustering goal, we use an Uplink. Trying to adapt to the superimposition nature of the beeping model, we re-define Uplink action as follows. For each node u, we denote by $dist(u)$ the distance of u to the closets candidate. In Uplink, each candidate has a message of length L for transmission, and we want each node u to receive the superimposition of the messages of the candidates at distance $dist(u)$ from u. We later see why this Uplink procedure is a natural fit to the beeping model and how we can implement this Uplink easily. Before going to these implementation related details, let us finish the discussion about clustering. Suppose that there exists a Black-box algorithm $\mathcal{A}_{Up}$ for the above Uplink description. Now we explain how to use $\mathcal{A}_{Up}$ to cluster nodes in the desired manner. For this, we use a SI(1) code. That is, each candidate encodes its id using this code, and candidates Uplink these coded ids. On the receiving end, each node u receives the superimposition of the coded ids of the candidates at distance $dist(u)$ from u. Noting the properties of SI(1) codes, if there is only one such candidate, u can decode

Algorithm 4 Clustering, run @ node u

```
Numbering
C ← SI(1)-code
m_u ← C(0, ID_u)
Uplink m_u, receive bit-sequence m'_u

if m' is a valid id then
    Cluster-ID ← decoding of m' into an ID
    clustered ← true
else
    Cluster-ID ← ∅
    clustered ← false

boundary ← false
for t=0 to L − 1 do
    if m'[t] = 1 then
        BEEP
        LISTEN
    else
        LISTEN
        BEEP
    if heard a beep while listening then
        boundary ← true
```

Algorithm 5 Numbering Algorithm, run @ node u

Output: distance $dist(u)$ to the closest candidate

```
active ← false
if candidate then
    dist(u) ← 0
for t = 1 to D do
    if active or candidate then
        BEEP
    else
        LISTEN
        if heard a beep then
            active ← true
            dist(u) ← t
```

the id of that candidate. On the other hand, if there are two or more of those candidates, u can distinguish this case and declare itself as *unclustered*. This concludes the clustering task.

One remark about the shape of the clusters achieved by this algorithm is as follows: each node w can be unclustered only if it has two neighbors v_1 and v_2 that belong to different clusters. Thus, each cluster grows from every side till it either reaches the margins of the network or it is within distance 2 hops from another cluster. Let us call a node u *boundary* if u is clustered but it is adjacent to a node u' such that u' is either unclustered or it belongs to a cluster other than that of u. We say two clusters $\mathcal{C}_1$ and $\mathcal{C}_2$ are *adjacent* if there exist two nodes $v_1 \in \mathcal{C}_1$ and $v_2 \in \mathcal{C}_2$ such that v_1 and v_2 are within distance 2 of each other. It is clear that in that case, v_1 and v_2 are boundary nodes. If the distance between v_1 and v_2 is exactly 1, then clusters are directly touching each other, whereas if distance is two, with an unclustered node w in the middle, then w serves as a bridge connecting the two clusters.

Communications on the Overlay Graph: For implementing communications, we want to devise protocols such that using these protocols, each candidate can exchange messages with neighboring candidates in the overlay graph. Trying to adapt to the superimposition nature of beeping networks, we do this in two layers: we first implement communications between candidates such that each candidate receives the superimposition of the messages of the neighboring candidates, in the overlay graph. Then, we use a $SI(\log n)$ code on top of these superimposition channels to get to full message exchange. Note that $SI(\log n)$ codes are robust enough because the number of candidates is at most $\log n$. On the negative side, these codings come with a cost, the encoding of the $\Theta(\log n)$ bit messages is $\Theta(\log^3 n)$ bits, which leads to the $\log^3 n$ term in the time bound of the debates. later we explain how to modify the debate algorithms to get over this cost.

Thus, what remains is to implement communications between candidates such that each candidate receives the superimposition of the messages of the neighboring candidates, in the overlay graph. For this, we first *number* each node u with its distance from the closest candidate $dist(u)$. This *numbering* is essentially the backbone of the clusters and serves as the spine of our intra-cluster communications.

The algorithm for numbering is simple and as explained in Algorithm 5. In each round, each node is active or inactive; at the start, only candidates are active; and each node simply record the time in which it becomes active. In each round, active nodes beep and each inactive node becomes active if hears a beep. This way, the wave of the activation (the wave of beeps) proceeds exactly one hop in every round. Thus, every node u gets activated after exactly $dist(u)$ rounds where $dist(u)$ is the distance of u to the closest candidate.

Having this numbering, we implement the communications between candidates via three communication actions in or amongst the clusters: Uplink, Intercommunication, and Downlink. However, we change the definitions of these three task to adapt them to the superimposition nature of beeping model.

(a) In the default definition of Uplink, candidates start with messages and the message of its candidate to nodes in its cluster. In the adapted definition, we deliver to each node u, the superimposition of messages of candidates that are at distance $dist(u)$ from u. Thus, in particular, each clustered node receives the message of its related candidate. Moreover, unclustered nodes receive superimposition of more than one messages. This later helps us to distinguish clustered versus unclustered nodes. After the clustering, we essentially use the Uplink only for delivering the message of each candidate to the boundary nodes of the related cluster.

(b) By default, Intercommunication is the action where boundary nodes of different clusters exchange messages with each other. Adapting to the superimposition nature of beeping model, in intercommunication, the goal is for each boundary node to receive the superimposition of the messages of adjacent boundary nodes.

(c) At the end, in the usual definition, Downlink is where the message is brought down from the boundary nodes to the candidates. Adapting to the beep model, the new goal is for every candidate to receive the superimposition of the messages that boundaries of its cluster send.

Having these new definitions, now it is the time to zoom into the implementation details of these task, and see why these new definitions are easy to implement, in the beeping model and thanks to the numbering that we created.

Uplink: For Uplink, the algorithm is as presented in Algorithm 6. The main technique in here is the usual idea of pipelining the beeps. First, consider what happens inside one cluster ignoring the effect from the other clusters. A node at distance $dist(u)$ does its transmission about ℓ^{th} bit of the message at round $dist(u) + 3l$. In particular, the candidate starts the transmission about the first bit in round 0 and it finishes its transmissions in round $3L$. In each round t, a node u is allowed to transmit a bit only if $t - dist(u) \equiv 0 \pmod 3$. In that round, nodes that are one hop away are listening to this bit. That is, node w is listening to this bit (and recording it) if $t - dist(u) \equiv 2 \pmod 3$. This way, let us consider what happens to the first bit of the message. In the first round, candidate transmits or remains silent depending on what is the first bit of message. Then, inductively we see that for each $i \in [D]$, in the i^{th} round nodes that are at distance i from source transmit or remain silent depending on the first bit of the message. This way, the first bit reaches D-hops away after D rounds. Now note that when first bit has traveled only three hops from the candidate, the candidate starts transmitting the second bit, and thus, the wave of transmissions of $(j+1)^{th}$ bit follows the wave of transmissions of j^{th} bit with a three hop lag. Hence, by $D + 3L$ rounds, all the bits have reached every node.

Now let us see what happens what is the effect beep waves of different clusters on each other. Consider two neighboring clusters $\mathcal{C}_1$ and $\mathcal{C}_2$ respectively related to candidates u_1 and u_2. First suppose that $\mathcal{C}_1$ and $\mathcal{C}_2$ are connected via a bridging unclustered node w, where w is connected to $v_1 \in \mathcal{C}_1$ and $v_2 \in \mathcal{C}_2$. Then $dist(v_1) = dist(v_2) = dist(w)-1$. Thus, using the above beep waves, w always listens to the transmissions of v_1 and v_2 (and gets superimposition of them) while v_1 and v_2 ignore transmissions of w. Hence, the unclustered nodes receive the superimposition of the messages of their respective closest candidates. More importantly, the beep waves *clash* at the bridging node and don't go inside the other clusters. Hence, the progresses of the beep waves inside clusters remain intact. A similar thing happens when $\mathcal{C}_1$ and $\mathcal{C}_2$ are directly touching each other. In that case, for related boundary nodes v_1 and v_2, we have $dist(v_1) = dist(v_2)$ and thus, v_1 and v_2 do not listen to transmissions of each other.

Intercommunication: With the new definition, Intercommunication task is now easy to implement. An ideal algorithm would be like this: boundary nodes go through the bits of the messages that they have, bit by bit, and for each bit, they beep if the bit is a one, and listen otherwise. Each node record a 1 if it beeps itself or if hears a beep. This way, if two clusters are touching, then on the related boundary nodes, the beep of one would be immediately observable by the other. However, if two clusters are connected via an unclustered bridging node w, then the beeps of two clusters don't reach each other. To remedy this, we do a slight modification to the above simple ideal algorithm: now for each bit, we use two rounds instead of one round. Each boundary beeps twice or listens twice depending on the bit that it has. Also, unclustered nodes listen in the first round and propagate whatever they received in the first round (beep iff they received a beep). Then, for each bit, each boundary records a one if it beeps itself or it senses a beep in any of the related two rounds. This protocol is presented in Algorithm 7. It is easy to see that, this protocol achieves the desired superimposed-type intercommunication goal.

Downlink: As presented in Algorithm 8 The implementation of Downlink is simply reversing the direction of beep waves of the Uplink. Now, the transmissions start

Algorithm 6 Uplink Algorithm, run @ node u

Given: $dist(u)$, and message bit sequence m_u (for any candidate u)
Output: bit sequence m'_u at each node

```
1: active ← false
2: for t = 0 to D + 3L − 3 do
3:     if candidate then
4:         active ← (m_u[⌊t/3⌋] == 1)
5:     switch t − dist(u) (mod 3) do
6:         case 0:
7:             if active then
8:                 BEEP
9:             else
10:                LISTEN
11:        case 1:
12:            LISTEN
13:        case 2:
14:            LISTEN
15:            if heard a beep then
16:                m'_u[⌊(t − dist(u) + 1)/3⌋] ← 1
17:                active ← true
18:            else
19:                m'_u[⌊(t − dist(u) + 1)/3⌋] ← 0
20:                active ← false
```

Algorithm 7 Intercommunication, run @ node u

Given: clustering, and message bit sequence m''_u if u is boundary
Output: superposition bit sequence μ_u if u is candidate

```
1: for t=0 to L − 1 do
2:     if clustered & boundary then
3:         if m_u[t] == 1 then
4:             BEEP
5:             BEEP
6:             m'''_u[t] ← 1
7:         else
8:             LISTEN
9:             LISTEN
10:            if heard a beep in above two rounds then
11:                m'''_u[t] ← 1
12:            else
13:                m'''_u[t] ← 0
14:    else
15:        LISTEN
16:        if heard a beep then
17:            BEEP
18:        else
19:            LISTEN
```

at the nodes furtherest away from the candidate, move towards the candidate. Nodes go through the bits with a lag of three hops between the waves related to two consequent bits. Using the transmission schedules based on the numbering, each node v only listens to transmissions of nodes that are at distance $dist(v) + 1$ from the candidate. In this case, v receives the superimposition of messages of those nodes. Since superimposition of superimpositions is simply a superimposition, what at the end the candidate receives is the superimposition of the messages sent out from the boundary nodes.

7.3 $O(D + \log n \log\log n)$**-length debates** Now we show how to modify the debate algorithm presented above to get its time complexity to $O(D + \log n \log\log n)$, which leads to optimal $O(D + \log n \log\log n) \cdot \log\log n$ leader election (optimal up to $\log\log n$ factors).

As explained in the overview section, the main change is based on the simple observation, which is, at the end, in the debate algorithm, we use something significantly weaker than full message communication. We only use two things: (1) the number of different messages received; used for determining the degree on the overlay graph, and (2) whether a neighbor has a message numerically larger or not; used for detecting whether a neighboring candidate has a greater(*degree*, *id*) pair or not. In the following, we explain how to achieve these two goals without going through the high cost of full message communications. Having the implementation of these two, the change in the analysis of the main elimination algorithm is as presented in Section 5.3, where we proved that each new debate reduces the number of remaining candidates by a constant factor, while keeping at least one.

For determining the degree, instead of $SI(\log n)$ codes, we use a new set of codes that are just strong enough to enable us to find a 2-estimate of the degree of each candidate in the overlay graph. These new codes encode each message of length $\log n$ bits into a codeword of $\Theta(\log n \log\log n)$ bits (see Lemma 7.2).

In second use, for detecting whether a neighboring candidate has a numerically larger message, we need a slight modification in the intercommunication algorithm. Let us say that boundary node u should be *marked* if u has a node w (from a different cluster) within its two hops such that the message of w is numerically larger than that of u. In the new intercommunications, the goal is for each boundary node to detect whether it should be marked. Once the marking procedure is done, we simply use a Downlink with single-bit

Algorithm 8 Downlink, run @ node u

Given: clustering, and bit sequence μ_u if u is boundary
Output: a bit sequence μ'_u in each candidate

```
1: active ← 0
2: for t = D + 3L − 3 downto 0 do
3:     if clustered & boundary then
4:         if t − dist(u) ∈ [0, 3(L − 1)] then
5:             active ← μ_u[⌊t − dist(u))/3⌋]
6:     switch t − dist(u) (mod 3) do
7:         case 0:
8:             if active == 1 then
9:                 BEEP
10:            else
11:                LISTEN
12:        case 1:
13:            LISTEN
14:            if heard a beep then
15:                active ← 1
16:            else
17:                active ← 0
18:            if candidate then
19:                if t ∈ [1, 3(L − 1) + 1] then
20:                    μ'_u[(t − dist(u) − 1)/3] ← active
21:        case 2:
22:            LISTEN
```

messages (marked or not) and each candidate gets to know whether any of nodes in its cluster is marked. This means that, each candidate knows if it has a neighboring candidate in the overlay graph with a numerically larger message or not.

For marking the boundary nodes according to above rule, the ideal algorithm is for boundary nodes to go through the bits of their messages and compare them. In each round, each unmarked boundary node beeps if the related bit of its message is one, and listens otherwise. Then, each unmarked boundary gets *marked* if it was listening but heard a beep. A boundary node that gets marked does not continue the intercommunication procedure. Similar to intercommunication in previous debate algorithm, to remedy the issue that neighboring clusters might be not directly touching, we use an extra beeping round. For each bit we spend two rounds, each unmarked boundary with bit 1 in respective place of its message beeps twice, each other unmarked boundary listens twice and each unclustered node listens first and then repeats what it hears in the next round. An unmarked boundary gets marked if it was not beeping but heard a beep in any other rounds. The related pseudocode is presented in Algorithm 9.

Algorithm 9 Max-Detection-Intercommunication, run @ node u

Given: clustering, and message m_u if u is a boundary
Output: boolean *marked*, for any boundary u

```
1: marked ← false
2: for t=0 to L − 1 do
3:     if clustered & boundary & ¬marked then
4:         if bit(m_u, t) = 1 then
5:             BEEP
6:             BEEP
7:             rec_u[t] ← 1
8:         else
9:             LISTEN
10:            LISTEN
11:            if heard a beep in above two rounds then
12:                marked ← true
13:    else
14:        LISTEN
15:        if heard a beep then
16:            BEEP
17:        else
18:            LISTEN
```

8 Conclusion

In this paper we presented the first linear time distributed algorithm for electing a leader in a radio network without collision detection. More importantly our algorithm runs with high probability in

$$O\left(D\log\frac{n}{D}+\log^3 n\right)\cdot\min\left\{\log\log n,\log\frac{n}{D}\right\}$$

rounds which is almost optimal given the $T_{BC} = \Omega(D\log\frac{n}{D})$ and $T_{BC} = \Omega(\log^2 n)$ lower bounds from [21] and [1] for the broadcast problem. Presenting a leader election algorithm that works in essentially T_{BC} rounds improves over the 23 year old simulation approach of Bar-Yehuda, Goldreich and Itai.

We believe that it should be possible to reduce the additive $\log^3 n$ to the optimal $\log^2 n$. A more interesting question is whether it is possible to remove the multiplicative factor of $\log\log n$. Possibly the ideas that reduce the running time from $O(n\log\log n)$ to $O(n)$ described in Section 5.4 can be useful here. We also give an almost optimal $O\left(D+\log n\log\log n\right)\cdot\min\left\{\log\log n,\log\frac{n}{D}\right\}$ leader election algorithm for radio networks with collision detection and the more restricted beep networks. This improves over the deterministic algorithm from [20] which takes $\Theta(n)$ rounds independently of the network diameter D.

Leader election is a crucial first step in communication primitives such as multiple broadcasts, multiple unicasts or message aggregation. Thus, the $\Theta(T_{BC} \log n)$ running time of leader election had became a bottleneck for getting better algorithms for these tasks. In this paper, we showed that leader election is no more a barrier for getting algorithms for these tasks that (almost) run in broadcast time T_{BC}. This opened the road for the results in [13], where the authors develop near optimal algorithms for these communication primitive.

References

[1] Alon, N., Bar-Noy, A., Linial, N., and Peleg, D. A lower bound for radio broadcast. *Journal of Computer and System Sciences 43*, 2 (1991), 290–298.

[2] B. S. Tsybakov, V. A. M. Free synchronous packet access in a broadcast channel with feedback. *Problems Inform. Transmission 14*, 4 (1978), 259–280.

[3] Bar-Yehuda, R., Goldreich, O., and Itai, A. On the time-complexity of broadcast in radio networks: an exponential gap between determinism randomization. In *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing* (New York, NY, USA, 1987), PODC '87, ACM, pp. 98–108.

[4] Bar-Yehuda, R., Goldreich, O., and Itai, A. Efficient emulation of single-hop radio network with collision detection on multi-hop radio network with no collision detection. In *Proceedings of the 3rd International Workshop on Distributed Algorithms* (London, UK, UK, 1989), Springer-Verlag, pp. 24–32.

[5] Bar-Yehuda, R., Israeli, A., and Itai, A. Multiple communication in multi-hop radio networks. *SIAM Journal on Computing 22*, 4 (1993), 875–887.

[6] Capetanakis, J. Tree algorithms for packet broadcast channels. *Information Theory, IEEE Transactions on 25*, 5 (sep 1979), 505 – 515.

[7] Chlamtac, I., and Kutten., S. On broadcasting in radio networks: Problem analysis and protocol design. *IEEE Transactions on Communications 33*, 12 (1985), 1240–1246.

[8] Clementi, A., Monti, A., and Silvestri, R. Selective families, superimposed codes, and broadcasting on unknown radio networks. In *The Proceedings of ACM-SIAM Symposium on Discrete Algorithms* (2001), pp. 709–718.

[9] Clementi, A. E. F., Monti, A., and Silvestri, R. Distributed broadcast in radio networks of unknown topology. *Theor. Comput. Sci. 302*, 1-3 (June 2003), 337–364.

[10] Cornejo, A., and Kuhn, F. Deploying wireless networks with beeps. In *Proceedings of the 24th international conference on Distributed computing* (Berlin, Heidelberg, 2010), DISC'10, Springer-Verlag, pp. 148–162.

[11] Czumaj, A., and Rytter, W. Broadcasting algorithms in radio networks with unknown topology. In *The Proceedings of the Symposium on Foundations of Computer Science* (2003), pp. 492–501.

[12] Gasieniec, L., Peleg, D., and Xin, Q. Faster communication in known topology radio networks. In *Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing* (New York, NY, USA, 2005), PODC '05, ACM, pp. 129–137.

[13] Ghaffari, M., and Haeupler, B. Optimal communication in radio networks: Aggregation, unicasts, broadcasts and gossip. *manuscript* (2012).

[14] Ghaffari, M., Haeupler, B., and Khabbazian, M. The Complexity of Mutli-Message Broadcast in Radio Networks with Known Topology. *arXiv* (2012).

[15] Ghaffari, M., Lynch, N., and Sastry, S. Leader election using loneliness detection. In *Proceedings of the 25th international conference on Distributed computing* (Berlin, Heidelberg, 2011), DISC'11, Springer-Verlag, pp. 268–282.

[16] Greenberg, A. G., and Winograd, S. A lower bound on the time needed in the worst case to resolve conflicts deterministically in multiple access channels. *J. ACM 32*, 3 (July 1985), 589–596.

[17] Hayes, J. An adaptive technique for local distribution. *Communications, IEEE Transactions on 26*, 8 (aug 1978), 1178 – 1186.

[18] Jurdzinski, T., and Stachowiak, G. Probabilistic algorithms for the wake-up problem in single-hop radio networks. *Theor. Comp. Sys. 38*, 3 (May 2005), 347–367.

[19] Kowalski, D., and Pelc, A. Broadcasting in undirected ad hoc radio networks. In *The Proceedings of the International Symposium on Principles of Distributed Computing* (2003), pp. 73–82.

[20] Kowalski, D. R., and Pelc, A. Leader election in ad hoc radio networks: A keen ear helps. In *Proceedings of the 36th Internatilonal Collogquium on Automata, Languages and Programming: Part II* (Berlin, Heidelberg, 2009), ICALP '09, Springer-Verlag, pp. 521–533.

[21] Kushilevitz, E., and Mansour, Y. An $\Omega(D \log(N/D))$ lower bound for broadcast in radio networks. *SIAM Journal on Computing 27*, 3 (1998), 702–712.

[22] Lynch, N. A. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

[23] Nakano, K., and Olariu, S. Uniform leader election protocols for radio networks. *IEEE Trans. Parallel Distrib. Syst. 13*, 5 (May 2002), 516–526.

[24] Willard, D. E. Log-logarithmic selection resolution protocols in a multiple access channel. *SIAM J. Comput. 15*, 2 (May 1986), 468–477.