Deterministic Leader Election in Multi-Hop Beeping Networks

Klaus-Tycho Foerster[a], Jochen Seidel[a,], Roger Wattenhofer[a]

[a] *ETH Zurich, Zurich, Switzerland*

Abstract

We study deterministic leader election in multi-hop radio networks in the beeping model. More specifically, we address explicit leader election: One node is elected as the leader, the other nodes know its identifier, and the algorithm terminates at some point with the network being quiescent. No initial knowledge of the network is assumed, i.e., nodes know neither the size of the network nor their degree, they only have a unique identifier. Our main contribution is a deterministic explicit leader election algorithm in the synchronous beeping model with a run time of $O(D \log n)$ rounds. This is achieved by carefully combining a fast local election algorithm with two new techniques for synchronization and communication in radio networks. We also extend our results to synchronized wake-up protocols and to a Monte Carlo algorithm for anonymous networks.

1. Introduction

Distributed computing and wireless communication are prime application areas for randomization, as randomized algorithms are often both simpler and more efficient than their deterministic counterparts. However, in some cases the randomized algorithm is only of Monte Carlo nature, i.e., with some probability the algorithm fails. This is a problem if the randomized algorithm is used as a starting point for other (deterministic and Las Vegas) algorithms, as the algorithm as a whole can also not provide any guarantees anymore. A classic example for such a basic problem is leader election, which is often used to as a first step for other wireless algorithms. We would argue in this article that leader election deserves to be understood deterministically as well, and we present a new algorithm that solves leader election in the wireless beeping model – our algorithm is slower than the fastest known randomized algorithm, but the overhead is bearable.

The beeping model has emerged as an alternative to the traditional radio network model. The beeping model is binary, in a synchronous time step nodes

[*] An extended abstract appeared in the proceedings of the 28th International Symposium on Distributed Computing (DISC'14), Austin, Texas, USA, October 2014 [14].

can only choose to beep or not to beep. If a node is beeping, it does not get any feedback regarding other nodes. On the other hand, if a node is silent, it will learn whether all its neighbors are also silent, or whether at least one neighbor is beeping. The beeping model was introduced to the distributed computing community by Cornejo and Kuhn [7] shortly after it was implemented [13].

In this model, we deterministically solve leader election: All the nodes in the multi-hop network have to agree on a single leader. As leader election is impossible without nodes having unique identifiers [1], we assume that each node is equipped with a unique ID. We want our algorithm to be uniform, i.e., apart from their ID, nodes have no knowledge about any global or local network properties (e.g., the network size, or their degree).

Our main result is an algorithm that deterministically solves the leader election problem in $O(D \log n)$ time, where D is the diameter of the network and n is the number of nodes. Once a leader is elected, all nodes in the network know the leader's ID, and the network is quiescent. We achieve this task by carefully combining several methods.

1.1. Overview

First, we describe a `Campaigning` algorithm (Section 3) that can be compared to one iteration of a real word political campaign: Every node is equipped with a candidate leader and attempts to convince its neighborhood that this candidate would make a great leader. The idea is that, if enough campaigns are performed, everyone will be convinced of the same leader, since her influence spreads at least one hop per iteration. In other words, we would like to perform multiple campaigns, one after another.

As it turns out, in the beeping model ensuring that the next algorithm starts synchronized is a non-trivial task. We thus develop a technique that allows us to sequentially execute algorithms (Section 4) and apply it to the `Campaigning` algorithm (Section 4.3).

The third method establishes a "back-channel" (Section 5) that directs messages towards a specific node, in our case the current candidate leaders. This allows the last remaining candidate to detect its election and turn the network quiescent. Our main result is now obtained by executing the `Campaigning` algorithm multiple times sequentially, while at the same time using the back-channel to notify the global leader when its successful election is detected. Lastly, we briefly sketch how our algorithm can be extended to include a simple synchronized wake-up protocol (Section 6) and to anonymous networks (Section 7).

1.2. Related Work

Leader election is one of the fundamental problems in distributed computing, often used as the first step for solving a myriad of other problems in networks. As such, the problem was studied over decades in various communication and network models [25].

In radio networks, communication takes usually place in synchronous rounds, and nodes may either transmit or listen in every round. If a node transmits, it cannot hear incoming messages, but the message is sent to all its neighbors at once. If a node listens, it receives messages from all its neighbors, but the message obtained depends on the model of collision detection. Should collision detection be available, then a node can separate between no message sent, exactly one message sent, or a collision of multiple messages. With no collision detection available nodes can only distinguish between exactly one message sent to it or just noise.

Leader election in radio networks was first considered in single-hop radio networks, followed by the study of multi-hop radio networks. We start with a short coverage of the single-hop case:

For deterministic algorithms in single-hop radio networks, the run time highly depends on the availability of collision detection: With collision detection, it is $\Theta(\log n)$ [3, 20, 21, 28], while without collision detection, it is $\Theta(n \log n)$ [6]. A similar case can be made for randomized algorithms in single-hop radio networks: With collision detection, the expected run time is $\Theta(\log\log n)$ [30]. The expected run time goes to $O(\log n)$ if w.h.p. is desired. Should no collision detection be available, then the run time increases to $\Theta(\log n)$ in the expected case [2, 24], and to $\Theta(\log^2 n)$ w.h.p. [22]. In the context of single-hop beeping networks, Gilbert and Newport [18] also studied the size constraints on the leader election algorithm's state machine representation, recently also considering so-called noisy processes [19], inspired by biological distributed algorithms.

We would argue that the study of leader election in multi-hop radio networks can be divided into the following fields for related work to our results. One can consider (1) radio networks with or (2) without collision detection, and (3) the beeping model. Second, the used algorithms can be either deterministic or randomized. We refer to [4, 16, 23] for an extended overview of these areas.

For deterministic algorithms, Kowalski and Pelc [23] displayed the discrepancy between models with and without collision detection. They showed that if collision detection is available, the run time is $\Theta(n)$, while without collision detection, there is a lower bound of $\Omega(n \log n)$. Their $O(n)$ algorithm with collision detection relies on a careful combination of multiple innovative techniques, e.g., remote token elimination and distributed fuzzy-degree clustering. In contrast to the model in this article, they require messages of logarithmic size, collision detection, and the knowledge of an upper bound polynomial in the number of identifiers. Our algorithm can be simulated therein since their model is strictly stronger. Asymptotically, we achieve a better run time for graphs with a diameter $D \in o(n/\log n)$, cf. [16].

For randomized algorithms in radio networks without collision detection, Chlebus, Kowalski, and Pelc [4] broke the $\Omega(n \log n)$ barrier: They present a randomized algorithm with $O(n)$ expected time and prove a lower bound of $\Omega(n)$. Furthermore, they give a deterministic algorithm for the model without collision detection with a run time of $O(n \log^{3/2} n \sqrt{\log\log n})$. They use logarithmic size messages and also assume that an upper bound on the network size is known. Czumaj and Davies [11] improve the running time for small diameter networks and consider directed networks, see [10] for further recent advances.

Ghaffari and Haeupler [16] considered randomized leader election w.h.p. in the beeping model. Their algorithm runs in $O((D+\log n \log\log n)\cdot\min(\log\log n, \log n/D))$ time, with a lower bound of $\Omega(D + \log n)$. To choose the random starting set of candidates, they rely on knowledge of n, while we assume our algorithms to be uniform. To cope with overlapping transmissions, they present a sophisticated technique using superimposed codes. Czumaj and Davies complement their result by presenting an algorithm with an expected run time of $O(D + \log n)$ [9]. We deem our overhead of $O(\log n)$ in the worst case bearable.

The authors of [16] also consider a variant of the beeping model in which only a subset $S \subseteq V$ of the nodes wakes up in round 0 [17]. We adapt our algorithm to this setting in Section 6. The difficulty is to allow nodes that are being woken up by neighbors to synchronize their execution with that of nodes already awake. The related wake-up problem, where nodes may also activate spontaneously and no collision detection is available was studied in its own right, for single-hop [15] as well as multi-hop [5] networks, see also [12]. In [27] the goal is to activate the whole network if exactly one node is active initially.

Lastly, we note that the results from this article (in the form of the preliminary extended abstract [14]) are already employed in the work of Czumaj and Davies [8], where our deterministic leader election algorithm is used as an initial subroutine for their broadcast and gossiping algorithms.

2. Preliminaries

Network Model. The network is modeled as a connected undirected graph $G = (V, E)$ with node set V and edge set E. We denote by D the diameter of G, and by n the number of nodes in V. All nodes $u \in V$ have a unique identifier (ID), denoted by $\text{id}(u)$, from the range $\{1, 2, \ldots, O(n^\gamma)\}$, with $\gamma \geq 1$ being a constant [29, §2]. We denote by $l(v)$ the *length* of u's identifier in bits, i.e., $l(u) = \lceil \log_2(\text{id}(u))\rceil$. The *neighborhood* $\mathcal{N}(u)$ of u is the set $\{u\} \cup \{v : (u, v) \in E\}$. In a similar fashion, the *d-neighborhood* $\mathcal{N}(u, d)$ of a node u contains all nodes with a distance of at most d to u, e.g., $\mathcal{N}(u, 1) = \mathcal{N}(u)$.

Beeping Model. We consider one of the most basic communication models, the synchronous beeping model: All nodes start synchronized[1] in round 0, and communication between nodes proceeds in synchronous rounds, where messages are transmitted via the edges of the network. In every round, each node may choose to either *beep* or *listen* to incoming messages. If a node v beeps, the beep will be transmitted to all nodes in $\mathcal{N}(v)$. Otherwise, if v listens, then the message received by v in a round is defined as follows:(i) if no node in $\mathcal{N}(v)$ beeps, then v receives a 0 (*silence*), and (ii) if one or more nodes in $\mathcal{N}(v)$ beeps, then v receives a 1 (*beep*).

[1]In Section 6 we also handle the case in which only a subset of the nodes wakes up.

Uniform Algorithms. We only consider uniform algorithms. That is, unless mentioned otherwise, the input for a node v consists of only $\mathrm{id}(v)$ (but not the value of γ). Note that neither n, nor D, nor any upper bounds on those network parameters, can be inferred from the value $\mathrm{id}(v)$ (or $l(v)$) of a single node v. Nodes also do not have any knowledge about the network topology, e.g., the IDs of their neighbors, or even their own degree. Moreover, we require that in every network the algorithm reaches a quiescent state, i.e., a state in which no node transmits beeps anymore.

3. Convincing Your Neighbors

In this section, we give an algorithm called `Campaigning` that can be compared to a political campaign at a word of mouth level. Everyone is convinced that either she herself is a good candidate, or that she knows the name of a good candidate. If you know a better candidate than all of your neighbors and their neighbors, you will try to convince your direct neighbors. However, if they are aware that a better candidate is out there — they will ignore your conversion attempts. Some candidates might reach a good deal of local followers, but only a globally best candidate can guarantee to spread her sphere of influence all the time.

The algorithm `Campaigning` can be seen as one iteration of this process, where nodes exchange information only with their local neighborhood. The general idea is that after D iterations are performed, *"There can be only one!"*[2], and all nodes will be convinced of the same leader. Hence, the candidates of the different nodes do not have to be unique, e.g., the algorithm works with just one candidate for all nodes or n different candidates.

We have to reach a state where nodes can transmit information to their neighbors, without other nodes disturbing them, since beeps do not encode relevant further information. Particular challenges arise from the facts that the algorithm has to be uniform, i.e., that n is not known, and that we are confined to the restricted beeping model. E.g., one cannot just "beep the identifier" and then proceed with another part of the algorithm, since any receiving node will hear all its neighbors – and cannot distinguish if all sent a beep or just one.

The main idea is to first reach local consensus on the longest identifier, then to agree locally on the highest identifier, and finally, to let those with the locally highest identifier transmit their identifier to their neighbors. To reach a state of local consensus, we turn some nodes into buffer-nodes that no longer participate. Therefore, we divide our algorithm into three separate procedures `campaign_longest_id`, `campaign_highest_id`, and `campaign_transmit_id`.

We first give an overview of the three procedures in Subsection 3.1, followed by a detailed mode of operations for `Campaigning` in Subsection 3.2. In the appendix a pseudo code description of our algorithm is presented in Figure A.4.

[2]Connor MacLeod, 1985. In *Highlander*.

We conclude by stating correctness and run time results in Subsection 3.3, presenting the formal proof details in Subsection 3.4.

3.1. Overview of the Procedures

Every node v gets as input an id, referred to as *campaigning-identifier*, that is stored in v's variable id_{in}. Also, all nodes start in an *active role*, but can change to be *passive* or *inactive* during the algorithm. Active nodes might convince their neighbors at the end and passive nodes might receive a new candidate, but it can be necessary to turn nodes inactive to let them act as local separators.

After the first procedure, `campaign_longest_id`, exactly those nodes v with the *longest* id_{in} in their 2-neighborhood are still active. If a node v is not active, but has an active neighbor w, then v turns passive, since it is interested in the campaigning-identifier of w. Nodes not fulfilling either of these requirements turn inactive. Furthermore, to separate clusters of active nodes with campaigning-identifiers of different length, the procedure creates buffers of inactive nodes between them. Thus, inside a cluster, all active campaigning-identifiers are of equal length, allowing each cluster to agree on a common starting time for the following procedure.

The second procedure `campaign_highest_id` mimics the first procedure, but now for the *highest* instead of the *longest* identifier. After `campaign_highest_id`, exactly those nodes v with the highest id_{in} in their 2-neighborhood remain active. The buffer of inactive nodes is extended to separate active nodes with different campaigning-identifiers. Hence, in the third procedure `campaign_transmit_id`, all still active nodes can convince their passive neighbors unhindered.

3.2. Details of the Algorithm

In this subsection, we describe the algorithm `Campaigning` and each of its three procedures for a node $v \in V$. We describe the algorithm from the perspective of a single node v. The input campaigning-identifier for v is stored in id_{in}, and the length of id_{in} in bits is stored in the variable l_{in}. Furthermore, v initializes the variables $role \leftarrow active$, $l_{out} \leftarrow l_{in}$, and $\mathrm{id}_{out} \leftarrow \mathrm{id}_{in}$. Then, the node v executes `campaign_longest_id`, `campaign_highest_id`, `campaign_transmit_id`, and the output of node v is id_{out}. Should a node become inactive at any time, i.e., if $role = inactive$, then the algorithm immediately terminates and the value currently stored in id_{out} is returned as v's output.

Each procedure consist of *phases*, which are divided into three rounds each. For ease of notation, we call the rounds in one phase *slots*, i.e., slot 0, slot 1, and slot 2. Conceptually, the first two slots 0 and 1 of each phase are used to transmit data, while slot 2 will exclusively be used for notification signals from active nodes. Recall that v *hears* a beep only if some node $u \in \mathcal{N}(v)$, $u \neq v$ transmits a beep, i.e., v does not hear beeps of itself.

campaign_longest_id. The length of `campaign_longest_id` may vary; at the end of the procedure, node v stores the number of elapsed phases in l_{out} if at the end of the procedure v is active or passive. Node v starts by beeping in slots 0 and 1 for the first $l_{in} - 1$ phases. Then, v listens in slot 0, and beeps received in slot 0 are relayed in slot 1. If v relays at least one beep, it turns passive. Should a beep be heard in the next slot 2, the node v turns inactive. Otherwise, already in phase l_{in} there was no beep to relay. In that case, if a (relayed) beep is received in slot 1, then v turns inactive. Else v beeps in slot 2 of that phase and finishes the procedure as active. Should after phase l_{in} a beep be heard in slot 1, the passive relaying node v turns inactive as well. Should there be a phase where the passive node v hears no beeps in slot 0,1, it either *i*) turns inactive if no beep is heard in slot 2, or *ii*) finishes the procedure as passive if a beep is heard in slot 2.

campaign_highest_id. This procedure consists of l_{out} phases, and we denote the current phase of node v by p.

If v is passive at the beginning of phase p, then beeps heard in slot 0 are relayed in slot 1. Should no beep be heard in slot 0, but a beep is heard in slot 1, v turns inactive. Also, if no beep is heard in slot 2 of phase $l_{out}(v)$, then v turns inactive.

We denote by the positions $1, \ldots, l_{in}$ the bits of id_{in}, starting from the most significant bit. If v is active at the beginning of phase p, then v beeps in slots 0 and 1 if position p in id_{in} is a 1 bit. Else, when a beep is heard in slot 0 or 1, v turns passive. If the current phase is l_{out} and v is still active, then v beeps in slot 2.

campaign_transmit_id. Much like `campaign_highest_id`, this procedure consists of l_{out} phases. An active node v uses the l_{out} phases to transmit the l_{out} bits of id_{in}, whereas passive nodes store the l_{out} received bits in id_{out}.

3.3. Convincing via `Campaigning`

We can now state some important properties of the algorithm `Campaigning`, which will be used in the next sections to prove our main result. All proofs are presented in Subsection 3.4. We begin with the following correctness lemma, which essentially states that nodes may only adopt identifiers from their neighborhood, i.e., identifiers spread only locally and no new identifiers are created.

Lemma 1. *Let v be a node that just finished algorithm* `Campaigning(`$id_{in}(v)$`)`*. Then $id_{out}(v) \leq \max_{w \in \mathcal{N}(v,1)} \mathrm{id}_{in}(w)$ and $\exists x \in \mathcal{N}(v,1)$ s.t. $id_{out}(v) = id_{in}(x)$.*

The proof to Lemma 1 consists of a careful case distinction based on the node's *role* in the `Campaigning` algorithm. In Theorem 1, we show that the influence of a potential leader will spread one hop per round. This is crucial for the whole leader election process, since it will be extended later on to show that D executions of the algorithm suffice to convince all nodes of the leader.

Theorem 1. *Execute algorithm* `Campaigning`$(\mathrm{id}_{in}(v))$ *for* $\forall v \in V$. *Let* $v' \in V$ *be a node with* $\mathrm{id}_{in}(v') = \max_{w \in \mathcal{N}(v',3)} \mathrm{id}_{in}(w)$. *Then for all nodes* $u \in \mathcal{N}(v',1)$ *holds:* $\mathrm{id}_{out}(u) = \mathrm{id}_{in}(v')$.

The above theorem is established by observing that a node v with a locally highest campaigning-identifier (i.e., the highest id_{in} in $\mathcal{N}(v,3)$) remains active, its neighbors do not turn inactive, and thus the campaigning-identifier is propagated one hop. Finally, Theorem 2 states that the run time of `Campaigning` depends only on the largest campaigning-identifier length in the 1-neighborhood.

Theorem 2. *Execute algorithm* `Campaigning`$(\mathrm{id}_{in}(v))$ *for* $\forall v \in V$. *The run time for each node* v *is* $O(\max_{w \in \mathcal{N}(v,1)} l_{in}(w))$ *rounds.*

This is true since the maximum run time of a node is completely determined after `campaign_longest_id` has finished. Recall that all identifiers are at most in $O(n^{\gamma})$, and hence the run time is bounded by $O(\log n)$ rounds. Since Lemma 1 ensures that no new identifiers are created in the network, we obtain the following corollary.

Corollary 1. *Let* $\max_{v \in V} l_{in}(v) \in O(\log n)$. *It holds that the run time of algorithm* `Campaigning`$(\mathrm{id}_{in}(v))$ *is* $O(\log n)$ *rounds for* $\forall v \in V$.

3.4. Proof Details

In this subsection we present the proofs for Lemma 1, Theorem 1, and Theorem 2. To this end, we first present some preliminarly statements in Subsubsection 3.4.1, which are used for the proof of Lemma 1 in Subsubsection 3.4.2 and the proof of Theorem 1 in Subsubsection 3.4.3, before we prove Theorem 2 in Subsubsection 3.4.4.

3.4.1. Supporting Statements

We begin by stating that after finishing `campaign_longest_id`, two non-inactive neighboring nodes will have reached consensus on a common l_{out}:

Lemma 2. *Let* $a \neq b$ *be two neighboring nodes, i.e.,* $a \in \mathcal{N}(b,1)$ *that just finished the first procedure* `campaign_longest_id` *of* `Campaigning`$(id_{in}(v))$. *If neither* a *nor* b *is inactive, then* $l_{out}(a) = l_{out}(b)$.

Proof (of Lemma 2). We prove the lemma by case distinction, since the role of each of the two nodes is active or passive.

Let a and b be both active, then they finished the first procedure after phase $l_{out}(a) = l_{in}(a)$ and $l_{out}(b) = l_{in}(b)$ respectively. Assume w.l.o.g. for contradiction that $l_{out}(a) < l_{out}(b)$. Then a heard a beep in slot 0 of phase $l_{out}(a)$ and thus would have turned passive.

Let w.l.o.g. a be active and b be passive. If $l_{out}(a) < l_{out}(b)$, then b would have heard a beep in slot 2 of phase $l_{out}(a)$, and would have become inactive. Thus, assume that $l_{out}(a) > l_{out}(b)$. b may only finish the first procedure as an active node if no beep is heard in a successive slot 0 and 1. However, in phase

$l_{out}(b)$, the node b would have heard a beep from node a in slot 0. Therefore, b could not have finished after phase $l_{out}(b)$.

Let a and b be both passive. Again, assume w.l.o.g. that $l_{out}(a) > l_{out}(b)$. Both a and b have an active neighbor (else they would not be passive) a' and b' with $l_{out}(a) = l_{out}(a')$ and $l_{out}(b) = l_{out}(b')$, see the item above. It holds that $a \in \mathcal{N}(b', 2)$, $a \in \mathcal{N}(a', 1)$ and $b \in \mathcal{N}(a', 2)$, $b \in \mathcal{N}(a', 2)$. Consider phase $l_{out}(b)$ of `campaign_longest_id`: b' beeped in slot 2, a' beeped in slot 0, and a relayed the beep of a' in slot 1. Thus, b heard a beep in slot 1 and 2, and would have become inactive, a contradiction. □

In the same spirit, we state that after `Campaigning` was run, two non-inactive neighboring nodes will have reached consensus on a common id_{out}:

Lemma 3. *Let $a \neq b$ be two neighboring nodes, i.e., $a \in \mathcal{N}(b, 1)$, that just finished the Algorithm* `Campaigning(`$id_{in}(v)$`)`*. If neither a nor b is inactive, then $\mathrm{id}_{out}(a) = \mathrm{id}_{out}(b)$.*

Proof (of Lemma 3). Again, we prove the Lemma by case distinction, since the role of each of the two nodes is active or passive. Conceptually, the proof is similar in some items to the proof of Lemma 2. We know from Lemma 2 that $l_{out}(a) = l_{out}(b)$.

Let a, b be both active. Being active, they only transmit their campaigning-identifier in `campaign_transmit_id` (never changing their campaigning-identifier in the algorithm). Thus, we look at `campaign_highest_id`. For each phase p' of `campaign_highest_id` holds: If a had a 0 at position p' of $\mathrm{id}_{in}(a)$, then a heard a 0 in slot 0 and a 0 in slot 1 of phase p'. Else, a would have changed its role to not active. The same holds for b. Now assume for contradiction w.l.o.g. that $l_{out}(a) > l_{out}(b)$. Consider the most significant bit p^* where a has a 1 in $\mathrm{id}_{in}(a)$ and b' has a 0 in $\mathrm{id}_{in}(b')$. Then there would have been a phase p^* in `campaign_transmit_id` s.t. a beeps a 1 in slot 0, but b listens in slot 0 and hears a 1. This would lead to b turning not active in the algorithm, a contradiction.

Let w.l.o.g. a be active and b be passive. We start with $\mathrm{id}_{out}(a) > \mathrm{id}_{out}(b)$. After the procedure `campaign_transmit_id`, node b has a 1 in its campaigning-identifier at every position where a has a 1 in its campaigning-identifier. Thus, $\mathrm{id}_{out}(a) > \mathrm{id}_{out}(b)$ is a contradiction. We now consider the remaining case: If $\mathrm{id}_{out}(b) > \mathrm{id}_{out}(a)$, then some active node $c \in \mathcal{N}(b, 1)$ transmitted a beep in slot 0 in `campaign_transmit_id` in some phase where a transmitted no beep. Note that there is a 2-hop path from a to c via b. We distinguish between the cases $\mathrm{id}_{out}(c) > \mathrm{id}_{out}(a)$ and $\mathrm{id}_{out}(c) < \mathrm{id}_{out}(a)$. If $\mathrm{id}_{out}(c) > \mathrm{id}_{out}(a)$, then b would have relayed a 1 to a in `campaign_highest_id` from c in a phase where a listened in slot 1. Then a would have changed its role to not active. In the same fashion, if $\mathrm{id}_{out}(a) > \mathrm{id}_{out}(c)$, then b would have relayed a 1 to c in `campaign_highest_id` from a in a phase where c listened in slot 1. Then c would have changed its role to not active. Both cases lead to a contradiction.

Let a and b be both passive. Again, assume w.l.o.g. that $\mathrm{id}_{out}(a) > \mathrm{id}_{out}(b)$. Both a and b have an active neighbor after `campaign_highest_id` (else they would not be passive, since only an active neighbor would have beeped in slot

2 in the last phase of `campaign_highest_id`). After `campaign_highest_id`, for every active neighbor $a' \in \mathcal{N}(a,1)$ and every active neighbor $b' \in \mathcal{N}(b,1)$ holds: $\text{id}_{out}(a) = \text{id}_{out}(a')$ and $\text{id}_{out}(b) = \text{id}_{out}(b')$. This holds due to the item above and the fact that, if a node is active after `campaign_highest_id`, it stays active and never changes its campaigning-identifier. Thus, by assumption, $\text{id}_{out}(a') > \text{id}_{out}(b')$. Also, $a \in \mathcal{N}(b',2)$, $a \in \mathcal{N}(a',1)$ and $b \in \mathcal{N}(a',2)$, $b \in \mathcal{N}(a',2)$. Consider the most significant bit p' where a' has a 1 in $\text{id}_{in}(a')$ and b' has a 0 in $\text{id}_{in}(b')$. I.e., in all phases of `campaign_highest_id` before p', both a' and b' sent a 1 in each slot 0 and 1. In phase p' of `campaign_highest_id`, the node a received a 1 in slot 0 and the node b received a 0 in slot 0, since each active neighbor of b has the same campaigning-identifier Thus, a beeps in slot 1 and b hears a 1, relayed by a. Therefore, b turns inactive, a contradiction to b being passive.

From Lemma 3 it follows that if a node is adjacent to a node with another identifier, at least one of the two nodes has to be inactive:

Corollary 2. *Let $a \neq b$ be two neighboring nodes, i.e., $a \in \mathcal{N}(b,1)$, that just finished the Algorithm* `Campaigning(`$id_{in}(v)$`)`*. If $\text{id}_{out}(a) \neq \text{id}_{out}(b)$, then at least one of the nodes a, b is not active.*

We continue with lemmas that show that the algorithm does assign new identifiers in a correct fashion. We start by stating that nodes can never get a smaller identifier:

Lemma 4. *Let v be a node that just finished Algorithm* `Campaigning(`$id_{in}(v)$`)`*. Then v did not get a smaller campaigning-identifier as its output, i.e., $id_{in}(v) \leq id_{out}(v)$.*

Proof (of Lemma 4). If v is inactive or active after `Campaigning`, then it holds that $\text{id}_{out}(v) = \text{id}_{in}(v)$, since only passive nodes receive a new identifier in `Campaigning`. The assignment of an $\text{id}_{out}(v)$ with $\text{id}_{out}(v) \neq \text{id}_{in}(v)$ can only happen in `campaign_transmit_id` to a passive node. Assume for contradiction, that $\text{id}_{out}(v) < \text{id}_{in}(v)$ for a passive node after `campaign_transmit_id`. Then there would have been an active node $w \in \mathcal{N}(v,1)$ with $\text{id}_{in}(w) < \text{id}_{in}(v)$ that transmitted its $\text{id}_{in}(w)$ to v in `campaign_transmit_id`.

We first consider the special case $l_{in}(w) < l_{in}(v)$: In that case, v would have beeped in slot 0 of phase $l_{in}(v) - 1$ of `campaign_longest_id`, while w would have listened in slot 0 of the same phase. Thus, w would have heard a 1 in slot 1 and turned inactive, a contradiction.

We can therefore assume $l_{in}(w) \geq l_{in}(v)$, but $l_{in}(w) > l_{in}(v)$ is impossible due to the assumption $\text{id}_{in}(w) < \text{id}_{in}(v)$. Hence, we only need to consider $l_{in}(w) = l_{in}(v)$ (meaning that v is still active at the beginning of `campaign_highest_id`) and $\text{id}_{in}(w) < \text{id}_{in}(v)$. With Lemma 3, there can be no active node $u \in \mathcal{N}(v,1)$ with $\text{id}_{out}(w) = \text{id}_{in}(w) \neq \text{id}_{in}(u) = \text{id}_{out}(u)$ after `Campaigning`, since we assumed $\text{id}_{out}(v) = \text{id}_{out}(w)$. Thus, the only active nodes $x \in \mathcal{N}(v,1)$ after `campaign_highest_id` will have $\text{id}_{in}(x) = \text{id}_{in}(w)$. However, v was active at the beginning of `campaign_highest_id` and is passive at the end of `campaign_highest_id`. v therefore heard a beep in a slot 0 of `campaign_highest_id`, which caused v to change its role to passive from active. Those beeps in slot 0 must have come from an active node, i.e., a node x with $\mathrm{id}_{in}(x) = \mathrm{id}_{in}(w)$. Since $\mathrm{id}_{in}(w) < \mathrm{id}_{in}(v)$, consider the most significant bit p' where $\mathrm{id}_{in}(v)$ has a 1, but $\mathrm{id}_{in}(w)$ has a 0. Then, the node v will beep in slot 0 of phase p' of `campaign_highest_id` and node x will listen in slot 0 of phase p' – leading to node x turning passive. Actually, every active node $y \in \mathcal{N}(v, 1)$ with $\mathrm{id}_{in}(y) = \mathrm{id}_{in}(x)$ will turn passive, and thus w with $\mathrm{id}_{in}(w) = \mathrm{id}_{in}(w)$ too. This is a contradiction to w being active in `campaign_transmit_id`. □

Furthermore, nodes that were convinced will receive a higher identifier:

Lemma 5. *Let v be a node that just finished Algorithm* `Campaigning(`$id_{in}(v)$`)` *with the role of being passive. Then v got a larger campaigning-identifier as its output, i.e., $id_{in}(v) < id_{out}(v)$.*

Proof (of Lemma 5). Again, v can only receive a new id_{out} in `campaign_transmit_id()` – and only from one of its active neighbors $x \in \mathcal{N}(v, 1)$. From the proof of Lemma 4 it follows that all active neighbors in $\mathcal{N}(v, 1)$ have the same campaigning-identifier. It also follows from the same proof that these active neighbors cannot have a smaller campaigning-identifier than v. I.e., $\mathrm{id}_{in}(x) < \mathrm{id}_{in}(v)$ is false.

This only leaves to show that the case of $\mathrm{id}_{in}(x) = \mathrm{id}_{in}(v)$ leads to a contradiction: But if $\mathrm{id}_{in}(x) = \mathrm{id}_{in}(v)$, then v listened in slot 0 and 1 in the phase $l_{out}(v)$ of `campaign_highest_id`, and did not hear a beep in the two slots. Thus, v beeped in the following slot 2 and stayed active, a contradiction.

Hence, $\mathrm{id}_{in}(x) > \mathrm{id}_{in}(v)$, and v received a larger campaigning-identifier $\mathrm{id}_{out}(v) = \mathrm{id}_{out}(x) > \mathrm{id}_{in}(v)$. □

Also, active nodes do not change their identifier:

Lemma 6. *Let v be a node that just finished Algorithm* `Campaigning(`$id_{in}(v)$`)` *with the role of not being passive. Then v kept its campaigning-identifier as its output, i.e., $id_{in}(v) = id_{out}(v)$.*

Proof (of Lemma 6). v only receives a new id_{out} in `campaign_transmit_id`, but only if v is passive. Since v is not passive, $\mathrm{id}_{in}(v) = \mathrm{id}_{out}(v)$ holds. □

3.4.2. Construction for the Proof of Lemma 1

We can now prove Lemma 1, making use of the proofs of Lemmas 3, 4, 5.

Proof (of Lemma 1). If v is inactive or active, then $\mathrm{id}_{in}(v) = \mathrm{id}_{out}(v)$ holds due to Lemma 6. The passive node v may only have received its new and larger campaigning-identifier in `campaign_transmit_id` from an active neighbor $x \in \mathcal{N}(v, 1)$, see the proofs of Lemma 4 and 5. With Lemma 3 it holds that $\mathrm{id}_{out}(v) = \mathrm{id}_{out}(x) = \mathrm{id}_{in}(x)$. □

3.4.3. Construction for the Proof of Theorem 1

We begin by stating Lemmas 7 and 8, which in turn leads to Corollary 3, which in turn implies Theorem 1. To this end, the proof of Lemma 7 will make use of the Lemmas 2 and 3.

If an active node had the highest campaigning-identifier in its 3-hop neighborhood, it will convince all of its neighbors:

Lemma 7. *Let v be a node that just finished Algorithm* `Campaigning(`$id_{in}(v)$`)`. *If v is active and $\mathrm{id}_{in}(v) = \max_{z\in\mathcal{N}(v,3)} \mathrm{id}_{in}(z)$, then for all nodes $x \in \mathcal{N}(v,1)$ holds: $\mathrm{id}_{out}(x) = \mathrm{id}_{in}(v)$.*

Proof (of Lemma 7). Let us assume that there is a node $y \in N(v,1)$ with $\mathrm{id}_{out}(y) < \mathrm{id}_{out}(v) = \mathrm{id}_{in}(v)$. Then y must be inactive due to Lemma 3. The node y could have only gone inactive in the procedure `campaign_longest_id` or `campaign_highest_id`.

We start with `campaign_longest_id`: If $l_{in}(v) \leq l_{in}(y)$, then v could have never heard a beep in slot 1 until phase $l_{in}(v)$, since v would have beeped in every phase before in slot 0 – and y either beeps itself or relays the beep in these slots 1. Furthermore, no node in $\mathcal{N}(y,2)$ can beep in slot 1 in phase $l_{in}(v)$, since this would violate the condition that v has the highest campaigning-identifier in its 3-hop neighborhood. However, y could turn inactive if it hears a beep in slot 2 in any phase before phase $l_{in}(v)$. This is prevented by y beeping in every slot 1 of these phases before phase $l_{in}(v)$, since a node is prevented from beeping in slot 2 of a phase if it hears a beep in slot 1 or beeps in slot 1. Also, v beeps in every slot 0 before phase $l_{in}(v)$ and beeps in slot 2 of phase $l_{in}(v)$, thus preventing the last possible option for y going inactive.

This leaves only `campaign_highest_id`: We know that y is either active or passive after `campaign_longest_id`. Thus, $l_{out}(v) = l_{out}(y)$ with Lemma 2. While it is active, it can only change its role to passive. y can only change to being inactive if its role is passive in `campaign_longest_id`. There are two options for a passive node to become inactive in `campaign_longest_id`: First, y could have heard a beep in slot 1 in some phase p'. This can only happen if v has a 0 at the most significant position p' of its campaigning-identifier $\mathrm{id}_{in}(v)$, because else y would relay the beep in slot 1. Consider such a phase p' where y would hear a beep in slot 1. There must be an active node $u \in \mathcal{N}(y,1)$ that beeps in phase p', but we know that $\mathrm{id}_{in}(u) < \mathrm{id}_{in}(v)$. Hence, y would have relayed a beep to u in some previous phase before p' where u listens, turning u passive, which is a contradiction. Second, the last option for y to turn inactive would be to not hear a beep in slot 2 in phase $l_{out}(y) = l_{out}(v)$. This is not possible, since v will beep in slot 2 of phase $l_{out}(v)$ of `campaign_highest_id`.

This concludes the proof, since y cannot be inactive. □

Also, if a node had the highest campaigning-identifier in its 3-hop neighborhood, it will end the algorithm being in an active role:

Lemma 8. *Let v be a node that just finished Algorithm* `Campaigning(`$id_{in}(v)$`)`. *If $\mathrm{id}_{in}(v) = \max_{z\in\mathcal{N}(v,3)} \mathrm{id}_{in}(z)$, then v is active.*

Proof (of Lemma 8). The node v can become inactive or passive either during the procedure `campaign_longest_id` or during the procedure `campaign_highest_id`.

We begin with analyzing `campaign_longest_id`: v cannot become inactive or passive until phase $l_{in}(v)$. In phase $l_{in}(v)$, v can only become inactive or passive if v hears a beep in slot 0 or 1. But a beep in that phase in slot 0 or 1 can only happen if there is a node $z \in \mathcal{N}(v, 2)$ with $l_{in}(z) > l_{in}(v)$, which is a contradiction.

We now analyze `campaign_highest_id`: In order to change the role from active, v has to become passive first. v this can only happen in phases where v listens in slot 0 and 1. Let p' be the first phase of `campaign_highest_id` where v hears a beep in slot 0 or 1. If a beep is heard in slot 0, then there must be a node $y \in \mathcal{N}(v, 1)$ with $\mathrm{id}_{in}(y) > \mathrm{id}_{in}(v)$, because else v would have turned y passive before. The same argument can be made for slot 1: If v hears a beep in slot 1 of phase p', then it must have been relayed by a passive node $u \in \mathcal{N}(v, 1)$, originating from a node $x \in \mathcal{N}(v, 2)$ with $\mathrm{id}_{in}(x) > \mathrm{id}_{in}(v)$, because else v would have turned x passive before via u. □

Combining Lemma 7 and Lemma 8 yields:

Corollary 3. *Let v be a node that just finished Algorithm* `Campaigning(`$id_{in}(v)$`)`. *If $\mathrm{id}_{in}(v) = \max_{z \in \mathcal{N}(v,3)} \mathrm{id}_{in}(z)$, then v is active and for all nodes $x \in \mathcal{N}(v, 1)$ holds: $\mathrm{id}_{out}(x) = \mathrm{id}_{in}(v)$.*

Proof (of Theorem 1). The theorem holds by applying Corollary 3 to every node $v \in V$ for one run of the Algorithm `Campaigning`. □

3.4.4. Construction for the Proof of Theorem 2

We now prove the last open statement from Section 3:

Proof (of Theorem 2). The value $l_{out}(v)$ after the first procedure `campaign_longest_id` is exactly the number of phases it took to finish the first procedure. The following procedures `campaign_highest_id` and `campaign_transmit_id` also take $l_{out}(v)$ phases. Thus, the total run time is in $O(l_{out}(v))$ rounds.

Since the value of $l_{out}(v)$ is determined in `campaign_longest_id`, we now consider now the behavior of v in only this procedure:

If v was active after `campaign_longest_id`, then $l_{out}(v) = l_{in}(v)$, and $l_{out}(v) \le \max_{w \in \mathcal{N}(v,1)} l_{in}(w)$.

If v was passive after `campaign_longest_id`, then v terminated `campaign_longest_id` by hearing a beep in slot 2 from one of its active neighbors $x \in \mathcal{N}(v, 1)$. Then, $l_{out}(v) = l_{in}(x)$ and $l_{out} v \le \max_{w \in \mathcal{N}(v,1)} l_{in}(w)$.

The remaining case is that v turned inactive in `campaign_longest_id`. If v turned inactive due to a beep heard in a slot 2, then this beep came from one of its active neighbors $x \in \mathcal{N}(v, 1)$. Then, $l_{out}(v) = l_{in}(x)$ and $l_{out}(v) \le \max_{w \in \mathcal{N}(v,1)} l_{in}(w)$.

If v turned inactive due to no beeps heard in consecutive slots 0,1,2, then v must have been passive before – else v would have beeped in slot 2 after not hearing/sending a beep in slot 0 and 1. Let the phase where this happens be $p^* > 1$. In phase $p^* - 1$, v heard a beep in slot 0 (thus remaining/becoming passive) and relayed the beep in slot 1, and did not hear a beep in slot 2. Thus, v had an active neighbor $u \in \mathcal{N}(v, 1)$ that beeped in slot 0 of phase $p^* - 1$. Hence, $l_{out}(v) < l_{in}(u)$. If v now hears no beeps in slots 0,1,2 of phase p^*, it turns inactive, and for its l_{out} now holds $l_{out}(v) \leq l_{in}(u)$. Thereby, $l_{out}(v) \leq \max_{w \in \mathcal{N}(v,1)} l_{in}(w)$

The only other option for v to turn inactive would be to hear a beep in a slot 1. If a beep was heard in slot 1 in phase l_{in}, then $l_{out} = l_{in}$. Thus, let $p' > l_{in}$ be the phase where v turned inactive by hearing a beep in a slot 1. If v hears a beep in slot 1, no beep was sent in slot 0 and no beep was heard in slot 0 by v in phase p'. Assume that a node $u \in \mathcal{N}(v, 1)$ sent a beep in slot 1 of phase p'. Then u was passive in phase p', but for all phases before p', the node u was active. Thus $p' - 1 \leq l_{in}(u)$. Since v turns inactive in phase p', we can deduce that $p' \leq l_{in}(u) + 1$. Hence, $l_{out}(v) \leq \max_{w \in \mathcal{N}(v,1)} l_{in}(w) + 1$ □

More precisely, the proof yields the following corollary.

Corollary 4. *Let v be a node. If $\mathrm{id}_{in}(v) = max_{u \in V}\, \mathrm{id}(u)$, then the run time of* `Campaigning` *is exactly $9 \cdot l_{in}(v) = 9 \cdot l_{out}(v)$. In any case, the run time of* `Campaigning` *is at most $9 \cdot max_{u \in V} l(u)$.*

4. Convincing Your Network

We would like to apply the campaigning method presented in the previous section to propagate the highest ID further. In other words, we need to execute `Campaigning` multiple times in succession. This task would be easy if there was some kind of global synchronization in order to guarantee that all nodes can start the next invocation of the campaigning algorithm at the same time. However, since the node labels have different lengths, so does each campaign. To overcome this obstacle, we design a generic approach to sequentially execute arbitrarily many algorithms in the beeping model. The key ingredient in our approach is the following *balanced counter* technique.

4.1. Balanced Counters

We present a method that enables the network to manage a balanced counter for every node u. At every node u, our balanced counter technique stores an integer value denoted by *counter*. To manipulate *counter* the two methods `increment` and `reset`, which instruct the counter to increment its value by one or reset it to zero, respectively, are provided. Our goal is to satisfy the following *balancing property*: For any two neighboring nodes u, v *participating*, i.e., not currently resetting their counters, the *counter* values of u and v shall differ by at most 1.

Note that transmitting the whole counter value in every round is not feasible due to the limited nature of the communication means the nodes have at their disposal. However, it turns out that transmitting the *counter* value modulo 3 suffices to ensure the balancing property. The transmission technique we use requires three reserved rounds, and allows a node to determine whether their neighbors have a lower counter value than themselves. The idea is now that nodes refrain from incrementing the counter as long as there are neighbors that are still behind.

We describe the balanced counter technique from the perspective of some node u using a state machine. Each node may be in one of the following states: Count, Reset-Notify, or Reset-Wait, and we denote u's current state by *state*. If *state* = Count, then u is considered to be a *participating* node, and either `increment` or `reset` may be invoked at u. In the other two states those operations are not available to u. The only allowed state transitions for node u are

1. Count $\rightarrow$ Reset-Notify if no node $v \in \mathcal{N}(u)$ is in Reset-Wait,
2. Reset-Notify $\rightarrow$ Reset-Wait if no node $v \in N(u)$ is in Count, and
3. Reset-Wait $\rightarrow$ Count if no node $v \in N(u)$ is in Reset-Notify.

Communication of the balanced counter technique is subdivided into phases indexed by the positive integers. Each individual phase consists of 6 rounds; to avoid confusion we use the term slot to refer to the individual rounds within a phase. The role of the first three slots $(0, 1, 2)$ is to transmit the counter increments, whereas the last three slots $(3, 4, 5)$ are used to transmit the node's current state. We now give a detailed description of the balanced counter technique; for convenience, in the appendix we also include a pseudo-code description (Figure A.1).

Initially, the state of u is Count, and *counter* $= 0$. In each phase, the operation at node u is as follows:

1. If *state* = Count, then u beeps in slot *counter* $\pmod 3$ and in slot 3;
2. If *state* = Reset-Notify, then u beeps in slot 4; and
3. If *state* = Reset-Wait, then u beeps in slot *counter* $\pmod 3$ and in slot 5.

Node u listens in all slots in which it does not beep.

`Increment`. The purpose of this operation is to increment *counter* by one without violating the balancing property. When `increment` is invoked at node u, then u waits for the first phase in which no beep is received in slot *counter* $- 1 \pmod 3$ (note that u never transmits in slot *counter* $- 1 \pmod 3$). Node v increments *counter* by 1 at the end of that phase and returns from the `increment` operation.

`Reset`. The purpose of this operation is to reset node u's value of *counter* to zero in accord with neighboring nodes $v \in N(u)$, while allowing nodes v to proceed participating before invoking `reset` themselves. Specifically, when `reset` is invoked at node u, then u successively transitions (1) from Count to Reset-Notify, thereby setting *counter* $\leftarrow 0$, (2) from Reset-Notify to Reset-Wait, and eventually (3) from Reset-Wait back to Count. In this process u respects the aforementioned restrictions for state transitions, utilizing the transmissions in slots 3 to 5. In particular, the aforementioned transition

(i), $1 \leq i \leq 3$, is consummated in the first phase in which no beep is received in slot $2 + i$.

We establish the following lemma, whose proof is deferred to Subsection 4.4.

Lemma 9. *The balanced counter technique satisfies the balancing property.*

4.2. Balanced Executions

Consider two algorithms $\mathcal{A}$ and $\mathcal{B}$ that shall be simulated sequentially. To achieve our goal, we intend to simulate the execution of $\mathcal{A}$ and $\mathcal{B}$ in the network. In $\mathcal{A}$'s simulation, the balanced counter is used as a round counter. Since the round counter satisfies the balancing property, it is ensured that the simulations performed by neighboring nodes progress at the same rate. When at some node u the simulation of $\mathcal{A}$ terminates, the round counter is reset by u. Node u then waits until its round counter returns to the COUNT state and thereupon starts the simulation of $\mathcal{B}$.

One needs to ensure that when round r of $\mathcal{A}$ (or $\mathcal{B}$) is simulated at node u, then u can determine whether one of its neighbors transmitted a beep in round $r-1$ of the simulation. To that end, we extend each phase of the counter technique by three additional slots and reserve the first 6 slots (0–5) for the balanced counter technique. Consider a phase p and a node u currently simulating algorithm $\mathcal{A}$, and denote by r the *counter* value for node u at the beginning of phase p. The three new slots (6–8) are used to transmit and receive the beeps emitted during the simulation as follows.

Assuming that $\mathcal{A}$ did not terminate in round $r-1$, the goal in phase p is to simulate $\mathcal{A}$'s round r. Node u simulates round r of algorithm $\mathcal{A}$ utilizing slot $r \pmod 3 + 6$ to replace $\mathcal{A}$'s access to the communication channel, where beeps received in slot $(r-1 \pmod 3 + 6)$ replace the beeps received by v in the simulation if node u listened in round $r-1$ of $\mathcal{A}$. Moreover, in slot $r-1 \pmod 3$, node u re-transmits a beep if u beeped in the last simulated round $r-1$ under $\mathcal{A}$. If u incremented the counter to the value r in the current phase, i.e., the counter progressed from $r-1$ to r, then v invokes `increment` again. Note that `increment` may delay incrementing r for several phases; in that case, the same round r of $\mathcal{A}$ is simulated in phase p multiple times, and if the beeps received in slot $r-1$ change, then so does the simulated execution of $\mathcal{A}$'s round r.

Otherwise, if $\mathcal{A}$ terminated its execution in the previous round $r-1$, the goal is to safely start the simulation of the next algorithm $\mathcal{B}$ at node u. To that end, node u invokes the `reset` operation. However, the simulated execution of the next algorithm $\mathcal{B}$ (possibly using u's output of $\mathcal{A}$ as input) only starts once u continues participating in the balanced counter, i.e., when *state* = COUNT. We establish the following lemma, the proof of which is deferred to Subsection 4.4.

Lemma 10. *Let $A = (\mathcal{A}_1, \ldots, \mathcal{A}_k)$ be a finite sequence of algorithms. Denote, for every $v \in V$, by $\hat{o}_1(v)$ the output produced at v by $\mathcal{A}_1$ when executed on G. For $i > 1$ and for every $v \in V$, denote by $\hat{o}_i(v)$ the output produced at v by $\mathcal{A}_i$ when executed on G, where the input to every $u \in V$ for $\mathcal{A}_i$ is specified as $\hat{o}_{i-1}(u)$.*

It holds that for every node v, the output $o(v)$ produced at v when using the balanced execution technique for A is $o(v) = \hat{o}_k(v)$.

4.3. Leader Election through Campaigning

We now have the tools available to design a non-quiescent leader election algorithm. Utilizing the balanced execution technique, every node executes the `Campaigning` algorithm sequentially, again and again. For every node u, the input to the first invocation of `Campaigning` is $id(u)$, and the input to every following invocation of `Campaigning` is the output of the previous one. In the following we refer to this basic protocol as the `Restless-LE` (for leader election) algorithm. It is immediate from the design of `Restless-LE`, that the network will never reach a quiescent state — for instance, the balanced counter technique never ceases to transmit. The following lemma states that `Restless-LE` obtains the desired result after at most D invocations of `Campaigning`.

Lemma 11. *If network G executes* `Restless-LE`*, then for every node $u \in V$, the output produced at u by the D-th invocation of* `Campaigning` *is* $\max_{v \in V} \mathrm{id}(v)$.

Utilizing the balanced execution technique, Lemma 11 can be obtained by inductively applying Lemma 1 and Theorem 1 for D times. Simulating D invocations of `Campaigning` takes $O(D \log n)$ rounds, as is stated in Theorem 3. The proofs to both Lemma 11 and Theorem 3 appear in Subsection 4.4.

Theorem 3. *If network G executes* `Restless-LE`*, then for every node $u \in V$, the D-th invocation of* `Campaigning` *terminates after $O(D \log n)$ rounds.*

Note that the network never reaches quiescence since the balanced counter technique continues to beep even after the D-th invocation of `Campaigning` has terminated. Moreover, without knowledge of D, node u has no means to decide when sufficiently many campaigns have been run.

4.4. Proof Details

in this subsection, we give the promised proofs of Lemmas 9, 10, 11 and Theorem 3. We start with Lemma 9 in Subsubsection 4.4.1.

4.4.1. Construction for the Proof of Lemma 9

For the analysis, we use the identifiers of variables in the balanced counter technique, parametrized by node u and phase p, to refer to the variable's value for u at the end of phase p. Specifically, $\mathit{state}(u, p)$ and $\mathit{counter}(u, p)$ denote the value of the corresponding variables for node u at the end of phase p.

Our analysis begins with the observation that for every node u in every phase p, if $\mathit{state}(u, p) = \textsc{Reset-Notify}$ or $\mathit{state}(u, p) = \textsc{Reset-Wait}$, then $\mathit{counter}(u, p) = 0$. Moreover, the balanced counter technique respects the aforementioned state transition restrictions due to the beeps emitted in slots 3–5 in combination with the implementation of `reset`. It follows that if in phase p there is a pair $(u, v) \in E$ with $\mathit{state}(u, p) = \textsc{Reset-Wait}$ and $\mathit{state}(v, p) = \textsc{Count}$, then $state(v, x) = \textsc{Reset-Notify}$ in some phase $x < p$. Let q be the largest such x. The state transitions for node v after round q were $\textsc{Reset-Notify} \to \textsc{Reset-Wait} \to \textsc{Count}$, and when v arrived in $\textsc{Count}$ in some phase r, it is guaranteed that $counter(v, r) = 0$. In all phases y, $r \le y \le p$, node v received a beep in slot 0 (sent by u), thus ensuring that $counter(v, y) \le 1$. We summarize our findings in the following lemma.

Lemma 12. *Let p be a phase. If $state(u, p) = \textsc{Reset-Wait}$ for some node $u \in V$, then for all $v \in \mathcal{N}(u)$ either $counter(v, p) = 0$ or $counter(v) = 1$.*

We can now establish that the two counters of neighboring participating nodes (i.e., nodes in state $\textsc{Count}$) differ by at most 1.

Proof (of Lemma 9). One needs to show that at the end of every phase p for every participating node $u \in V$, $|counter(u, p) - counter(v, p)| \le 1$ for all participating $v \in N(u)$. (Recall that a node is considered to be participating in phase p if $state(u, p) = \textsc{Count}$.) Our proof is by induction on the phases p, and the induction is based on phase $p = 1$ as follows. At the beginning of phase 1 the variable counter of all nodes is initialized to be 0, and the state of all nodes is $\textsc{Count}$. Thus, at the end of phase 1, for all $v \in V$ $counter(v, 0) \le 1$, and the induction hypothesis holds for $p = 1$.

For the induction step, consider some $p > 1$, and an arbitrary edge $(u, v) \in E$. Our goal is to show that the balancing property holds for (u, v) at the end of phase p. Since we install the aforementioned for arbitrary (u, v), this is sufficient to establish the induction.

If $state(u, p) \ne \textsc{Count}$ or $state(v, p) \ne \textsc{Count}$, the induction hypothesis holds. Otherwise, if $state(u, p) = state(v, p) = \textsc{Count}$, then there are two cases. The first case is that $state(u, p-1) = state(v, p-1) = \textsc{Count}$. Then, by the induction hypothesis, $|counter(u, p-1) - counter(v, p-1)| \le 1$ holds. Assume without loss of generality that $counter(u, p-1) \ge counter(v, p-1)$. The `increment` method increases the *counter* variable of node u in phase p only if no beep is received in slot $(counter(u, p-1) - 1 \pmod 3)$. If $counter(v, p) = counter(u, p-1) - 1$, then v transmits a beep in slot $(counter(v, p) - 1 \pmod 3)$. Otherwise, if $counter(v, p) = counter(u, p-1)$, then $counter(u, p) \le counter(v, p) + 1$. It follows that the assertion holds for phase p.

In the second case, at least one of the two nodes was not in $\textsc{Count}$ in the previous phase. Assume without loss of generality that $state(u, p-1) \ne \textsc{Count}$. Due to the allowed state transitions we conclude that $state(u, p-1) = \textsc{Reset-Wait}$, and thus, due to Lemma 12, $count(v, p-1) \le 1$. Since $count(u, p-1) = 0$, the same line of arguments as above shows that the logic of `increment` guarantees that $counter(u, p) = counter(u, p-1) + 1$ only if $counter(v, p) \le counter(u, p) + 1$, and the assertion holds for phase p. □

Combining Lemmas 9 and 12 yields that nodes in state $\textsc{Reset-Wait}$ stall the counters of neighbors that are in the $\textsc{Count}$ state. Note that a node v in the $\textsc{Count}$ state may only transition to $\textsc{Reset-Notify}$ if v has no more neighbors in state $\textsc{Reset-Wait}$. In conclusion, two nodes $(u, v) \in E$ that satisfy

$state(u,p) = state(v,p) =$ COUNT in some phase p have reset their counter the same number of times. We cast this insight in the following Lemma 13.

Lemma 13. *Let p be a phase and let u and v be nodes with $(u,v) \in E$. If $state(u,p) = state(v,p) =$ COUNT, then u and v invoked `reset` the same number of times.*

Achieving our initial goal, namely, to execute multiple algorithms in sequence, will rely heavily on the properties guaranteed by Lemmas 9 and 13.

4.4.2. Construction for the Proof of Lemma 10

Armed with Lemmas 9, 12, 13 we can now prove Lemma 10.

PROOF (OF LEMMA 10). For $1 \leq i \leq k$ and for every node $v \in V$, denote by $\hat{\eta}_i(v)$ the execution of $\mathcal{A}_i$ at node v obtained by executing $\mathcal{A}_i$ on G when using $\hat{o}_{i-1}(v)$ as input for node v. For the special case $i = 1$ we set $o_0(v)$ to the reserved symbol ε, indicating that no input other than the node's ID is available to v in $\mathcal{A}_1$. Note that the first round in $\hat{\eta}_i(v)$ is fully specified by v's input. Every following round r in $\hat{\eta}_i(v)$ is fully specified by the previous rounds $r' < r$ of every $\hat{\eta}_i(u)$, $u \in V$, and the choice whether v beeps or not.

We establish the statement by induction on k. The induction hypothesis is as follows. There is some k so that for every node $v \in V$ the following two assumptions hold:(1) $\hat{o}_k(v) = o_k(v)$, where $o_k(v)$ is the output observed by node v after $\mathcal{A}_k$ terminated at v; and (2) after $\mathcal{A}_k$ terminated at node v the `reset` operation was invoked exactly k times by v.

If $k = 0$, we are free to set $\hat{o}_0(v) = \varepsilon$ and the induction hypothesis holds vacuously. For $k > 0$, denote by $\eta_k(v)$ the execution of $\mathcal{A}_k$ obtained from v as follows. We claim that $\eta_k(v) = \hat{\eta}_k(v)$ for every node v. Note that v transmits a beep in round r of $\eta_k(v)$ if a beep is transmitted in slot $r \pmod 3$ of the first phase p in which $counter(v,p) = r+1$. Using the induction hypothesis for $k-1$, we conclude that for every node $v \in V$:(1) the input to $\mathcal{A}_k$ in $\eta_k(v)$ is the same as in $\hat{\eta}_k(v)$; and (2) when v starts $\mathcal{A}_k$, the `reset` operation was invoked $k-1$ times by v.

For the first part of the induction hypothesis it remains to show that v beeps in round r under $\eta_k(v)$ if and only if v beeps in round r under $\hat{\eta}_k(v)$. To that end, it is sufficient to argue that if for all nodes v, the first $r-1$ rounds of $\eta_k(v)$ are equal to the first $r-1$ rounds of $\hat{\eta}_k(v)$, then the same statement is true for round r. Observe that $\mathcal{A}$ proceeds from round r to $r+1$ only in a phase p if none of its participating neighbors are still in round $r-1$ due to the balancing property (Lemma 9). Therefore, a beep transmitted by a participating node $u \in \mathcal{N}(v)$ in round $r-1$ of $\eta_k(u)$ arrives at node v in phase p. If v has a non-participating neighbor u, then there are two cases.

In the first case, node u is in the RESET-WAIT state. From Lemma 13 we conclude that u's execution of A_k has not yet terminated, since `reset` is only invoked if that is the case. It follows from Lemma 12 that v may only proceed to the round $r = 1$ of $\mathcal{A}_k$. We already concluded that round 0 of $\eta_k(v)$ corresponds to round 0 of $\hat{\eta}_k(v)$.

In the second case, node u is in the Reset-Notify state. This means that $\eta_k(u)$ has terminated and hence u invoked `reset` k times, whereas u invoked `reset` only $k-1$ times. Since u does not proceed from Reset-Notify to Count while v remains in Count, u does not beep in slots 3–5, and hence no interfering beeps are received from u. This establishes the first part of the induction hypothesis.

For the second part of the induction hypothesis, consider a node v that invokes `reset`. Since $\eta_k(v) = \hat{\eta}_k(v)$ is already established, we conclude that the output $o_k(v)$ obtained at node v satisfies $o_k(v) = \hat{o}_k(v)$. We have established the lemma. □

Lemma 10 states that the final result of a finite sequence of sequentially executed algorithms is correct. We will, however, also require a slightly different statement that involves the *intermediate* results of an arbitrarily long sequence of algorithms. From the proof of the above lemma we obtain the following corollary.

Corollary 5. *Let $A = (\mathcal{A}_1, \mathcal{A}_2, \dots)$ be a sequence of algorithms. Denote, for every $v \in V$, by $\hat{o}_1(v)$ the output produced at v by $\mathcal{A}_1$ when executed on G. For $i > 1$ and for every $v \in V$, denote by $\hat{o}_i(v)$ the output produced at v by $\mathcal{A}_i$ when executed on G, where the input to every $u \in V$ for $\mathcal{A}_i$ is specified as $\hat{o}_{i-1}(u)$.*

It holds that for every node v, and for every $i \in \mathbb{N}, i \geq 1$, the output $o_i(v)$ produced at v by algorithm $\mathcal{A}_l$ when using the balanced execution technique for A is $o_i(v) = \hat{o}_i(v)$.

4.4.3. Construction for the Proof of Lemma 11

Using the previously stated Corollary 5, we can now prove Lemma 11.

Proof (of Lemma 11). For $i \geq 1$ and for every $v \in V$, denote by $o_i(v)$ the output produced at v by the i-th invocation of `Campaigning`. Set further $m = \max_{v \in V} \mathrm{id}(v)$. Due to Corollary 5 we may reason about the output of `Restless-LE` as if the next invocation of `Campaigning` starts synchronized after the previous algorithm reached a quiescent state. The statement can now be obtained by applying Lemma 1 and Theorem 1 for D times: Lemma 1 states that every output that is generated by `Campaigning` is identical to the identifier of some node in the network. It follows that $m = \max_{v \in V} o_i(v)$ for every integer $i \geq 1$. This is sufficient to guarantee that after invocation i of `Campaigning` it holds that $o_i(v) = m$ for node $v \in V$ if $o_{i-1}(u) = m$ for some $u \in \mathcal{N}(v)$ due to Theorem 1. By repeating this process D times, we obtain the assertion. □

4.4.4. Construction for the Proof of Theorem 3

We now prove Theorem 3, using Corollary 4 from Subsubsection 3.4.4.

Proof (of Theorem 3). Since a phase of the balanced execution technique has constant length, it suffices to show that the D-th invocation of `Campaigning` terminates after $O(D \log n)$ phases. Let $m = \max_v \mathrm{id}(v)$ be the highest ID in G. For $i \in \mathbb{N}$, $i \geq 1$, denote by S_i the set of nodes v starting the i-th `Campaigning` invocation with $id_{\text{in}}(v) = m$, by t_i the phase in which the first node in S_i begins the i-th `Campaigning` invocation, in particular, $t_1 = 0$. We claim that, for $i \geq 1$,

1. $t_{i+1} = t_i + 9l(m) + 3$;
2. for all $v \in S_i$, invocation i of `Campaigning` starts in phase t_i; and
3. for all $v \in V \setminus S_i$, in phase t_i, node v is either simulating invocation i, or v invoked `reset` j times with $j \geq i$.

The claim is established by induction on i. The induction is based at $i = 1$. Due to Corollary 4 the first `Campaigning` invocation takes at most $9l(m)$ phases for any node. Moreover, for z, the invocation takes exactly $9l(m)$ phases. At every node v, in phase $9l(m) + 3$, the first `reset` operation has terminated and either invocation $i + 1$ or a later one is simulated; At node z, the latter is not the case.

For the induction step, consider some $i \geq 2$ and assume that the assertion holds for $i - 1$. Let $\mathcal{T}$ be the set of nodes v simulating invocation $i - 1$, and consider phase t_{i-1}. For $k \in \mathbb{N}$, let $T_k \subseteq \mathcal{T}$ be the nodes $v \in \mathcal{T}$ for which $counter(v, t_{i-1}) = k$. Since nodes in T_0 are free to increase their *counter*, in phase $t_{i-1} + 1$ the set T_0 is empty. Similarly, in phase $t_{i-1} + x$, the sets $T_0, \ldots, T_{x-1}$ are empty, and hence nodes in T_x are free to increase their *counter*. Thus, in phase $t_{i-1} + 9l(m)$, invocation $(i - 1)$ has terminated for all nodes in $\mathcal{T}$. Following the same line of arguments used in the induction base, after three additional rounds, nodes in S_i start invocation i. Moreover, in phase $t_{i-1} + 9l(m) + 3$, nodes in $\mathcal{T} \setminus S_i$ are either simulating invocation i, or invoked `reset` j times with $j \geq i$. We have established the induction hypothesis for i, and since $l(m) \in O(\log n)$ conclude that $t_D \in O(D \log n)$, thus establishing the theorem.

5. Terminating & Achieving Quiescence

It seems that in the previous section we robbed Peter to pay Paul: We obtained `Restless-LE` which finds a leader in time $O(D \log n)$, but now our algorithm does not achieve quiescence, nor does a node know when to terminate. These two flaws could be considered a major drawback if one wishes to use the leader election algorithm as a foundation for another algorithm, since it is unclear when the latter can be started. To overcome this obstacle we implement an overlay network protocol that executes concurrently to the `Campaigning` invocations. The overlay network we establish on top of the original communication graph resembles the layers of an onion with the elected leader at its core. Utilizing the overlay network, we then describe how candidates detect whether the leader election process has terminated. Causing all non-elected nodes to terminate is now achieved by sending a broadcast message.

In order to form the overlay network, each node u keeps track of one additional variable *depth* taking values from the set $\{0, 1, 2\}$, initially set to 0. We say that a path $p = (u_1, \ldots, u_k)$, $(u_{i-1}, u_i) \in E$ for $2 \leq i \leq k$, is a *downward overlay path* if for all $i \geq 2$ it holds that $depth(u_i) = depth(u_{i-1}) + 1 \pmod 3$, and we denote the length k of a path p by $\text{length}(p)$. Conversely, we say that p is an *upward overlay path* if p reversed is a downward overlay path. One can think of downward overlay paths as leading away from the network's core, whereas upward overlay paths lead towards it. Note that initially, all overlay paths consist of only a single node. The general idea is to relay beeps along upward and downward overlay paths. Before extending the `Restless-LE` algorithm to utilize the overlay network, thus obtaining the quiescent terminating leader election algorithm `Quiescent-LE` in Section 5.1, we describe the operation of our overlay network technique in more detail. Note that in the appendix, we include a pseudo-code representation of the overlay network technique in Figure A.2.

Every round r of the leader election algorithm is replaced by phases consisting of 10 slots, one single slot and three triplets of slots. The single slot is reserved to execute the non-terminating leader election algorithm we obtained in Section 4.3. For clarity, we refer to the first slot triplet as *control* slots, to the second triplet as *up channel* slots, and to the last triplet as *down channel* slots. The control slots, up channel slots, and down channel slots are numbered from 0 to 2 (e.g., up channel slot 2 is the last slot in the second triplet of slots in a phase). While the role of the control slots is to establish the overlay network, the up and down channel slots are used to transmit beeps to nodes with smaller and higher depth, respectively.

More specifically, in every phase p, node u listens in the up channel $\mathit{depth}-1 \pmod 3$ and in the down channel $\mathit{depth}+1 \pmod 3$. If a beep is received in one of those slots, then in the following phase $p+1$, u beeps in the up channel *depth* or in the down channel *depth*, correspondingly. The overlay network further provides the two operations `beep_depth` and `join`, which are implemented as follows. When `beep_depth` is invoked by node u, then u transmits a beep in control slot *depth*. The corresponding `join` operation causes u to listen in the three control slots; node u then sets $\mathit{depth} \leftarrow i+1 \pmod 3$, where i denotes the index of the first control slot in which a beep was received, thereby joining the overlay network of one of its neighbors that invoked `beep_depth`.

5.1. Quiescent Leader Election

In this section, we describe the `Quiescent-LE` algorithm that utilizes the overlay network technique in conjunction with the `Restless-LE` algorithm. Formation of the overlay network is tightly coupled with `Restless-LE` and the invocations of `Campaigning` therein. Namely, whenever an invocation of `Campaigning` at node u returns a new ID x, node u joins the overlay network of a neighbor that transmitted x to u. Nodes that are currently being convinced of a new leader emit a signal into the upward channel of neighboring nodes, thus ensuring that no candidate terminates unless a consensus on the leader's ID has been reached.

In particular, for a node u in phase p, denote by σ the state in which the last `Campaigning` invocation that terminated for node u was upon termination. Denote further by *last_role*, *last_in*, and *last_out* the values of the corresponding variables *role*, $\mathrm{id}_{\mathrm{in}}$ and $\mathrm{id}_{\mathrm{out}}$ in σ. In phase p a node u is called a *candidate* if $\mathit{last\_in} = \mathrm{id}(u)$, and we say that node c is the candidate of u if $\mathit{last\_in} = \mathrm{id}(c)$. The idea is now to utilize the overlay network so that nodes may join the overlay network of their corresponding candidate. This is accomplished by setting the *depth* variable accordingly whenever the value of *last_out* changes.

In `Quiescent-LE`, the operation of node u is as follows (for convenience a pseudo-code description appears in Figure A.3 in the appendix). If $\mathit{last\_role} = \mathit{active}$, then u invokes `beep_depth`, thus allowing nodes $v \in \mathcal{N}(u)$ to set their depth. Correspondingly, u invokes `join` if its candidate has changed (i.e., if $\mathit{last\_role} = \mathit{passive}$), in order to assign a new value to its depth variable. In any case, if $\mathit{last\_role} \neq \mathit{active}$, then node u beeps in all three up channel slots in contrast to the normal up channel operation. A candidate that has not received a beep through the up channel for 18 consecutive rounds emits a signal in the down channel, thus instructing nodes to terminate.

5.2. Analysis of `Quiescent-LE`

For the analysis, we use the same identifiers as in the `Quiescent-LE` algorithm, parametrized by node and phase, to refer to a variable's value at the end of phase p. Namely, we denote by $\mathit{last\_role}(u,p)$, $\mathit{last\_in}(u,p)$, and $\mathit{last\_out}(u,p)$ the three corresponding values for node u at the end of phase p. For a node u, we further denote by $\sigma_i(u)$ the state of `Campaigning` stored in variable σ at u after the i-th invocation of `Campaigning` has terminated for u. If *var* is a variable in $\sigma_i(u)$, then we write $\mathit{var}^{\sigma_i(u)}$ to denote the value of that variable in $\sigma_i(u)$. For example, $\mathit{last\_role}(u,p) = \mathit{role}^{\sigma_i(u)}$ if u is executing invocation $i+1$ of `Campaigning` in phase p.

For a candidate c, we call the nodes $v \in V$ with $\mathit{last\_out}(v,p) = \mathrm{id}(c)$ the *followers* of c in phase p. The *coalition* of c in phase p is the set $C(c,p) = \{v \in V | v \text{ is a follower of } c \text{ and there is a downward overlay path from } c \text{ to } v\}$. Observe that every node is a member of at most one coalition, and that at the beginning of the leader election algorithm the coalitions are exactly all singleton sets $\{u\}$ where $u \in V$. It follows from Lemma 11 that after D invocations of `Campaigning`, there is only the grand coalition V.

Consider some coalition C of a candidate c. We say that a node $b \in C$ is a *border node* (of C) in phase p if there is a node $v \in \mathcal{N}(b)$ so that $\mathit{last\_in}(b,p) \neq \mathit{last\_in}(v,p)$. Our analysis starts with the following lemma, which essentially implies that up channel beeps are generated in the neighborhood of a border node.

Lemma 14. *Let p be a phase. If b is a border node in phase p and $\mathit{last\_role}(b,p) \neq \bot$, then there is a node $v \in \mathcal{N}(b)$ and a phase p' with $p \leq p' \leq p+9$ for which $\mathit{last\_role}(v,p')$ is either inactive or passive.*

Proof (of Lemma 14). Let b be a border node in phase p, i.e., there is a node $v \in \mathcal{N}(b)$ with $\mathit{last\_out}(b,p) \neq \mathit{last\_out}(v,p)$. If b and v both terminate the invocation of `Campaigning` in phase p, then by Corollary 2 either b or v satisfy that the claim for phase p.

Otherwise, the simulation of `Campaigning` at node b is delayed from node u by at most 1 round due to Lemma 9. Without loss of generality assume that b is ahead. In that case, b updated $\mathit{last\_role}(b,p)$, and if $\mathit{last\_role}(b,p) \neq \mathit{active}$ the statement follows. If otherwise $\mathit{last\_role}(b,p) = \mathit{active}$, then denote by t

the round of `Campaigning` in which b terminated, and by q the phase in which Quiescent-LE finished simulating round t of `Campaigning` at node u. We argue that then v is either inactive or passive in phase $q+9$. Observe that v's invocation of `Campaigning` terminates in the simulated round t, and that by Corollary 2 node v finishes the `Campaigning` algorithm in an inactive role in round t. Since simulating one round of `Campaigning` takes 9 rounds for node v, we conclude that $last\_role(v, q+9) = inactive$, thus establishing the claim. □

Consider a node v in phase p, and let v be executing the $(i+1)$-st invocation of the campaigning algorithm. We say that v is *unhappy* in phase p if v is a border node, $last\_role(v,p) \in \{active, inactive\}$, and there is a $w \in \mathcal{N}(v)$ such that $id_{\text{out}}^{\sigma_i(w)}$. In other words, v is unhappy if it tried to, but could not convince all its neighbors of its candidate in the last invocation of `Campaigning`. In any other case, v is called *happy*. Correspondingly, a candidate c is said to be *progressing* in phase p, if in all phases $p' \leq p$ all nodes in $C(c,p')$ were happy, otherwise we refer to c as being *stalled*.

Lemma 15. *Let c be a candidate in phase p. If c is progressing in phase p, then there is a downward overlay path from c to all its followers.*

Proof (of Lemma 15). The proof to this lemma is based on induction over p. Initially, in phase 0, the coalition of c is $C = \{c\}$. Unless $V = \{c\}$, the candidate c itself is a border node, and since c's identifier is unique no other node has c as its candidate. (If $V = \{c\}$, then path from c to c consists of only one node and is a downward overlay path.)

Now consider $p > 0$ and assume that the induction hypothesis, namely, that the assertion of the lemma is true in phase $p-1$, holds. If c is progressing in phase p, then c was progressing in phase $p-1$. First, consider a node $v \in C(c,p)$ that is not a border node. In that case v was in $C(c,p-1)$, and the downward path p from c to v that existed in $C(c,p-1)$ is also present in $C(c,p)$. If on the other hand $v \in C(c,p)$ is a border node, then $last\_role(v,p)$ is *passive* and v invokes `join` at the beginning of phase p. All active nodes $w \in \mathcal{N}(b)$ are in the overlay network of $C(c,p)$ and invoke `beep_depth` at the beginning of phase p. Node v sets its *depth* value accordingly, thus establishing the assertion at the end of phase p. □

From the last lemma we understand that the overlay network of a progressing candidate remains intact in the sense that all followers of c are connected to c via an overlay path. The *reach* of candidate c in phase p, denoted by $\text{reach}(c,p)$ is the minimum length of a shortest downward overlay path from c to one of $C(c,p)$'s border nodes. The following Lemma 16 states a basic property of progressing candidates, namely, that they extend their reach in steps of at most 1. Lemma 15 together with the fact that IDs can propagate only 1 hop at a time (Lemma 1) yields the following lemma.

Lemma 16. *Let c be a candidate in phase p. If c is progressing, then* $\text{reach}(c,p) \leq \text{reach}(c,p-1) + 1$.

After a candidate becomes stalled, Lemma 15 ceases to hold. However, we can show that a statement in the spirit of Lemma 16 holds also for stalled candidates. To that end, recall that a border node b of some coalition C turns unhappy if it cannot convince one of its neighbors. We can deduce from the next stated Lemmas 17 and 18 that this is the case only if the candidate of some node $v \in \mathcal{N}(b, 3)$ is larger than b's own candidate c: essentially, the Lemmas 17 and 18 state how the spread of influence for a candidate will be halted by a node with an higher identifier in the 3-neighborhood. Such a node v will never accept c as a candidate (due to Lemma 1), and therefore c's reach cannot extend beyond some node $w \in \mathcal{N}(b, 2)$.

Lemma 17. *Let v be a node that just finished Algorithm* `Campaigning(`$id_{in}(v)$`)`. *If v is active and there is a node $w \in \mathcal{N}(v, 1)$ s.t. $\mathrm{id}_{out}(w) \neq \mathrm{id}_{out}(v)$, then there is a node $u \in \mathcal{N}(v, 3)$ with $\mathrm{id}_{out}(u) \geq \mathrm{id}_{in}(u) > \mathrm{id}_{out}(v)$.*

Proof (of Lemma 17). Note that $\mathrm{id}_{out}(u) \geq \mathrm{id}_{in}(u)$ always holds due to Lemma 4. Also, active nodes do not change their campaigning-identifier, i.e., $\mathrm{id}_{out}(v) = \mathrm{id}_{in}(v)$ We know from 3 that if $w \in \mathcal{N}(v, 1)$ is not active or passive, the node w may only be inactive. Let us assume for contradiction that there is no node $u \in \mathcal{N}(v, 3)$ with $\mathrm{id}_{in}(u) > \mathrm{id}_{out}(v) = \mathrm{id}_{in}(v)$. Thus, we know that for the active node v holds: $\mathrm{id}_{out}(v) = \max_{z \in \mathcal{N}(v,3)} \mathrm{id}_{in}(z)$. But this is a contradiction due to Lemma 7. □

Lemma 18. *Let v be a node that just finished Algorithm* `Campaigning(`$id_{in}(v)$`)`. *If v is inactive and there is a node $w \in \mathcal{N}(v, 1)$ s.t. $\mathrm{id}_{out}(w) \neq \mathrm{id}_{out}(v)$, then there is a node $u \in \mathcal{N}(v, 3)$ with $\mathrm{id}_{out}(u) \geq \mathrm{id}_{in}(u) > \mathrm{id}_{out}(v)$.*

Proof (of Lemma 18). Again, note that $\mathrm{id}_{out}(u) \geq \mathrm{id}_{in}(u)$ always holds with Lemma 4. Also, inactive nodes do not change their campaigning-identifier, i.e., $\mathrm{id}_{out}(v) = \mathrm{id}_{in}(v)$ We assume for contradiction that there is no node $u \in \mathcal{N}(v, 3)$ with $\mathrm{id}_{in}(u) > \mathrm{id}_{out}(v) = \mathrm{id}_{in}(v)$. Then for v holds $\mathrm{id}_{out}(v) = \max_{z \in \mathcal{N}(v,3)} \mathrm{id}_{in}(z)$ But this implies that v is active due to Lemma 8, a contradiction. □

We therefore obtain from Lemmas 17 and 18 the following Lemma 19.

Lemma 19. *Let c be a candidate in phase p. Denote by U the set of all unhappy nodes in $C(p, v)$, and for a node u by $P(c, u)$ the set containing all downward overlay paths from c to u. If c is stalled in phase p, then the following two statements hold:*
1. $\mathrm{reach}(c, p + 1) = \min_{b \in U}\{\mathrm{length}(p)|p \in P(c, b)\} + 1$, *and*
2. for all $j \geq p + 2$ $reach(c, j) = \min_{b \in U}\{\mathrm{length}(p)|p \in P(c, b)\} + 2$.

It remains to finish our study of the border nodes.

Lemma 20. *Let c be a candidate in phase p. If the coalition $C(v, p)$ is unhappy, then it contains a border node.*

Proof. Let p be the first round in which $C(c,p)$ is unhappy. Then there is a border node b that caused $C(c,p)$ to be unhappy in phase p. In $C(c,p-1)$ there was downward overlay path from c to b, and this path still exists in phase p. We establish the assertion by induction, showing that in every phase $p+i$, $i>0$, there is a border node in $C(c,p+i)$.

Assume that there is a border node b in $C(c,p+i)$ for some $i \geq 0$, and let P denote the shortest downward overlay path from c to b in $C(c,p+i)$. Consider $C(c,p+i+1)$. If $b \in C(c,p+i+1)$ then we are finished. Otherwise, there must be a node x on P that has changed $depth(x,p)$ or $last\_out(x,p)$, thus interrupting the path; let x be the first such node on P. For every node v it holds that the value of $depth$ changes only if also the value of $last\_out$ does. Thus the predecessor of x on P is now a border node of $C(c,p)$, thus establishing the statement for $i+1$. The assertion follows. $\square$

By combining Lemmas 15 and 20 we obtain the following:

Corollary 6. *Let c be a candidate in phase p. If $C(c,p)$ contains border nodes, then there is at least one downward overlay path from c to a border node.*

Lemma 14 guarantees that at every border node b and in every phase p, a beep is transmitted through the up channel at least every 9 rounds. With help of Lemmas 16 and 19 we conclude that this beep reaches can only be delayed for 9 more rounds. Due to Corollary 6, a candidate whose coalition contains borders never receives 18 consecutive rounds of silence in its up channel, and thus never emits the termination signal through the down channel. Let $z \in V$ be the eventually elected leader node, i.e., $\mathrm{id}(z) > \mathrm{id}(u)$ for all $u \in V \setminus \{z\}$. The next Lemma 21 establishes that the coalition of z is always happy, whereas every other coalition turns unhappy at some point.

Lemma 21. *Let c be a candidate in phase p. It holds that $\mathrm{id}(c) = max_{v \in V}\, \mathrm{id}(v)$ if and only if $C(c,p')$ is happy in every phase $p' \geq p$ such that c is a candidate in phase p'.*

Proof. We show both directions of the if and only if separately, starting with the if direction. Let q denote the phase in which c became unhappy, and denote by b an unhappy border node in $C(c,q)$. Since b is unhappy, it follows from Lemmas 17 and 18 that there exists a $u \in V$ with $\mathrm{id}(u) > \mathrm{id}(c)$. Thus, c cannot become the leader chosen by our leader election algorithm.

For the opposite direction, recall Theorem 1 which implies that border nodes of $C(c,p')$ convince their neighbors within a single execution of `Campaigning`. Therefore no border node of $C(c,p')$ becomes unhappy, for any phase p'. This establishes the statement. $\square$

It follows from Lemma 11 that after at most D invocations of `Campaigning` have terminated for all nodes, the only coalition C of the leader node z has no border nodes. Moreover, since z was never unhappy $C(z,D)$ contains all nodes, and $reach(c) \leq D$. Therefore, the algorithm terminates and achieves quiescence due to the signal that z broadcasts through the down channel. By applying the run time Theorem 3, we obtain our main Theorem 4.

Theorem 4. *The uniform algorithm* `Quiescent-LE` *terminates after* $O(D \log n)$ *rounds at every node. Every node returns the same output* $\max_{v \in V} \mathrm{id}(v)$.

6. Synchronized Wake-Up Protocol

Note that one may also study a variant of the beeping model (see, e.g., [17]) in which only a subset $S \subseteq V$ of the nodes wakes up in round 0. Nodes in $V \setminus S$ are initially *asleep*, and wake up only if they receive a beep from one of their neighbors. In particular, such a node is no longer considered asleep. We briefly discuss how our algorithm can be extended to include a wake-up protocol.

Every original slot in a phase of `Quiescent-LE` is replaced by two slots, where the first slot takes the role of the corresponding original slot, and in the second slot a node is always silent. Additionally, the phase is preceded by another two slots, referred to as wake-up slots. Consider an asleep node u. As soon as u receives a beep, it enters an intermediate *snooze* state, and if u receives a beep in the next round as well, then it turns *awake*. Otherwise, snoozing nodes turn awake after receiving two beeps consecutively. A node that just turned awake enters the protocol after the wake-up slots, thus aligning its execution with awake neighbors. That is, the first round in which u participates corresponds to the first original slot of `Quiescent-LE`. Note that in particular, due to the balanced execution technique, node u postpones the progress of awake neighboring nodes. Lastly, a node u that is awake beeps in both wake-up slots whenever u starts a phase of `Quiescent-LE` that coincides with the beginning of a balanced execution phase, and remains silent in the wake-up slots otherwise.

7. Anonymous Networks

Throughout the article, we have so far assumed that all nodes are equipped with an unique identifier, where the node with the largest identifier will succeed in our leader election protocol. We now show how to perform leader election in anonymous networks, where nodes do not have any identifier assigned.

Observe that our methods do not rely on all nodes having pairwise disjoint identifiers, it suffices that the largest identifier is unique. As such, we can make use of results from Métivier, Robson, and Zemmari [26]: the authors provide methods, where without any communication, with very high probability (*w.v.h.p*)[3], some node $v \in V$, $|V| = n$, generates a unique and largest identifier of size $O((\log n)(\log^* n)^2)$ [26, Proposition 6].

We thus provide a Monte Carlo leader election algorithm for anonymous multi-hop beeping networks, by first running the algorithm of Métivier, Robson, and Zemmari [26] to generate identifiers, followed by an execution of our deterministic algorithm: its run time is $O(D(\log n)(\log^* n)^2)$, succeeding w.v.h.p.

[3] *w.v.h.p*: $1 - o(n^{-c})$ for any $c \geq 1$

8. Conclusion

We described a deterministic uniform leader election algorithm in the beeping model that achieves quiescence after $O(D \log n)$ rounds. There are three main ingredients to our algorithm:

1. A `Campaigning` algorithm that propagates the locally highest identifier one hop per invocation.
2. A technique to sequentially execute arbitrarily many algorithms in the beeping model, based on a simple balanced counter approach.
3. An overlay network, based on the onion layer principle.

Our algorithm is obtained by using the sequential execution technique (2.) to execute the `Campaigning` method (1.) multiple times, one after the other. In its first invocation, the algorithm essentially creates a 2-hop independent set containing at least one node. The independent nodes are potential leaders and transmit their identifier to their neighbors. In subsequent invocations, potential leaders correspond to clusters of nodes with the same campaigning-identifier. When clusters touch, the cluster C having the larger campaigning-identifier wins, and the neighboring clusters shrink as bordering nodes join C. This yields a non-quiescent uniform algorithm `Restless-LE` for leader election, where the leader is not informed about her successful election.

If the diameter D was known to all nodes, then termination could be achieved by stopping after the D^{th} invocation of `Campaigning`. However, we want our algorithm to be uniform. We create an onion layer overlay network (3.) in order to achieve uniformity and quiescence. Potential leaders form the core of an onion, and nodes in a cluster are layered according to their distance to the core. Since the cluster of the eventual leader grows in each step, eventually all nodes will be part of a single cluster. The onion layer principle can now be used to establish a communication channel from outer layers towards the core and vice versa. When the cluster stops growing, the leader is informed about her successful election, in turn allowing her to issue a termination signal to all nodes. Lastly, we explain how the algorithm can be extended to handle the synchronous wake-up situation described in [17] and anonymous networks.

Acknowledgements We would like to thank the anonymous reviewers of the 28th International Symposium on Distributed Computing (DISC'14) for their helpful comments on our preliminary extended abstract [14].

References

[1] Dana Angluin. Local and global properties in networks of processors (extended abstract). In *STOC*, pages 82–93. 1980.

[2] Reuven Bar-Yehuda, Oded Goldreich, and Alon Itai. On the time-complexity of broadcast in multi-hop radio networks: An exponential gap between determinism and randomization. *J. Comput. Syst. Sci.*, 45 (1): pages 104–126, 1992.

[3] John Capetanakis. Tree algorithms for packet broadcast channels. *IEEE Transactions on Information Theory*, 25 (5): pages 505–515, 1979.

[4] Bogdan S. Chlebus, Dariusz R. Kowalski, and Andrzej Pelc. Electing a leader in multi-hop radio networks. In *OPODIS*, pages 106–120. 2012.

[5] Marek Chrobak, Leszek Gasieniec, and Dariusz R. Kowalski. The wake-up problem in multihop radio networks. *SIAM J. Comput.*, 36 (5): pages 1453–1471, 2007.

[6] Andrea E. F. Clementi, Angelo Monti, and Riccardo Silvestri. Distributed broadcast in radio networks of unknown topology. *Theor. Comput. Sci.*, 302 (1-3): pages 337–364, 2003.

[7] Alejandro Cornejo and Fabian Kuhn. Deploying wireless networks with beeps. In *DISC*, pages 148–162. 2010.

[8] Artur Czumaj and Peter Davies. Communicating with beeps. In *OPODIS*, pages 30:1–30:16. 2015.

[9] Artur Czumaj and Peter Davies. Brief announcement: Optimal leader election in multi-hop radio networks. In *PODC*, pages 47–49. 2016.

[10] Artur Czumaj and Peter Davies. Exploiting spontaneous transmissions for broadcasting and leader election in radio networks. In *PODC*, pages 3–12. ACM, 2017.

[11] Artur Czumaj and Peter Davies. Deterministic communication in radio networks. *SIAM J. Comput.*, 47 (1): pages 218–240, 2018.

[12] Jurek Czyzowicz, Leszek Gasieniec, Dariusz R. Kowalski, and Andrzej Pelc. Consensus and mutual exclusion in a multiple access channel. *IEEE Trans. Parallel Distrib. Syst.*, 22 (7): pages 1092–1104, 2011.

[13] Roland Flury and Roger Wattenhofer. Slotted programming for sensor networks. In *IPSN*, pages 24–34. 2010.

[14] Klaus-Tycho Foerster, Jochen Seidel, and Roger Wattenhofer. Deterministic leader election in multi-hop beeping networks - (extended abstract). In *DISC*, pages 212–226. 2014.

[15] Leszek Gasieniec, Andrzej Pelc, and David Peleg. The wakeup problem in synchronous broadcast systems. *SIAM J. Discrete Math.*, 14 (2): pages 207–222, 2001.

[16] Mohsen Ghaffari and Bernhard Haeupler. Near optimal leader election in multi-hop radio networks. In *SODA*, pages 748–766. 2013.

[17] Mohsen Ghaffari and Bernhard Haeupler. Near optimal leader election in multi-hop radio networks. *CoRR*, abs/1210.8439v2, April 2014.

[18] Seth Gilbert and Calvin C. Newport. The computational power of beeps. In *DISC*, pages 31–46. 2015.

[19] Seth Gilbert and Calvin C. Newport. Symmetry breaking with noisy processes. In *PODC*, pages 273–282. 2017.

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

[21] Jeremiah F. Hayes. An adaptive technique for local distribution. *Communications, IEEE Transactions on*, 26 (8): pages 1178–1186, 1978.

[22] Tomasz Jurdzinski and Grzegorz Stachowiak. Probabilistic algorithms for the wake-up problem in single-hop radio networks. *Theory Comput. Syst.*, 38 (3): pages 347–367, 2005.

[23] Dariusz R. Kowalski and Andrzej Pelc. Leader election in ad hoc radio networks: A keen ear helps. *Journal of Computer and System Sciences*, 79 (7): pages 1164 – 1180, 2013.

[24] Eyal Kushilevitz and Yishay Mansour. An $\Omega(D \log(N/D))$ lower bound for broadcast in radio networks. *SIAM J. Comput.*, 27 (3): pages 702–712, 1998.

[25] Nancy A. Lynch. *Distributed Algorithms.* Morgan Kaufmann, 1996.

[26] Yves Métivier, John Michael Robson, and Akka Zemmari. Analysis of fully distributed splitting and naming probabilistic procedures and applications. *Theor. Comput. Sci.*, 584: pages 115–130, 2015.

[27] Andrzej Pelc. Activating anonymous ad hoc radio networks. *Distributed Computing*, 19 (5-6): pages 361–371, 2007.

[28] Boris S. Tsybakov and V.A. Mikhailov. Free synchronous packet access in a broadcast channel with feedback. *Probl Inf Transm*, 14 (4): pages 259–280, 1978.

[29] Shailesh Vaya. Information dissemination in unknown radio networks with large labels. *Theor. Comput. Sci.*, 520: pages 11–26, 2014.

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

Appendix A. Pseudo Code of Algorithms

```
Technique: Balanced-Counter from the perspective of node $u$
    initialize counter ← 0 and state ← Count
    for ever do
        if state = Count then
            beep in slot counter (mod 3)
            beep in slot 3
        else if state = Reset-Notify then
            beep in slot 4
        else if state = Reset-Wait then
            beep in slot counter (mod 3)
            beep in slot 5
        listen in all other slots
    Procedure increment:
        wait for a phase $p$ in which no beep is received in slot
         counter − 1 (mod 3)
        counter ← counter + 1 at the end of phase $p$
    Procedure reset:
        wait for a phase $p$ in which no beep is received in slot 5
        state ← Reset-Notify at the end of phase $p$
        counter ← 0 at the end of phase $p$
        wait for a phase $q > p$ in which no beep is received in slot 3
        state ← Reset-Wait at the end of phase $q$
        wait for a phase $r > q$ in which no beep is received in slot 4
        state ← Count at the end of phase $r$
```

Figure A.1: The balanced counter technique.

```
**Technique:** *Overlay Network* from the perspective of node $u$
    **initialize** $depth \leftarrow 0$
    **for** *every phase* $p$ **do**
        **listen** in up channel $depth + 1 \pmod 3$
        **if** *beep received in up channel* $depth + 1 \pmod 3$ *in phase* $p - 1$ **then**
            **beep** in up channel $depth \pmod 3$
        **listen** in down channel $depth - 1 \pmod 3$
        **if** *beep received in down channel* $depth - 1 \pmod 3$ *in phase* $p - 1$ **then**
            **beep** in down channel $depth \pmod 3$
        **listen** in all control slots
    **Procedure** join**:**
        $i \leftarrow$ the index $\in \{0, 1, 2\}$ of the first control slot in which a beep is received
        $depth \leftarrow i + 1 \pmod 3$
    **Procedure** beep_depth**:**
        **beep** in control slot *depth*
```

Figure A.2: The overlay network technique.

```
Algorithm: Quiescent-LE, from the perspective of node $u$
    initialize the overlay network technique
    initialize $\mathit{last\_role} \leftarrow \bot, \mathit{last\_in} = \mathrm{id}(u)$
    for every phase $p$ of the overlay network technique do
        ▷ In the first slot:
        execute one round of the non-quiescent leader election algorithm
          from Section 4.3
        ▷ In the remaining slots use the overlay network as follows:
        if the invocation of Campaigning executed by $u$ has terminated
          then
            $\sigma \leftarrow$ the state of $u$ at the end of the last Campaigning
              invocation
            $\mathit{last\_role} \leftarrow$ the value of $\mathit{role}$ in $\sigma$
            $\mathit{last\_in} \leftarrow$ the value of $\mathrm{id}_{\mathrm{in}}$ in $\sigma$
            $\mathit{last\_out} \leftarrow$ the value of $\mathrm{id}_{\mathrm{out}}$ in $\sigma$
        if $\mathit{last\_role} = \mathit{active}$ then
            beep_depth
        else if $\mathit{last\_role} = \mathit{passive}$ then
            join
        if $\mathit{last\_role} \neq \mathit{active}$ then
            beep in up channels 0, 1, and 2
        if $u$ is a candidate then
            if no beep received in up slot 1 for 18 consecutive phases
              then
                beep in down channel 1
                terminate after phase $p + 1$
        else if beep received in down channel then
            terminate after phase $p + 1$
```

Figure A.3: Usage of the overlay network for the leader election algorithm.

```
Algorithm: Campaigning($\mathrm{id}_{in}$)
    initialize $role \leftarrow active$ and $l_{out} \leftarrow l_{in}$ and $\mathrm{id}_{out} \leftarrow \mathrm{id}_{in}$
    campaign_longest_id
    campaign_highest_id
    campaign_transmit_id
    return $\mathrm{id}_{out}$
Procedure campaign_longest_id:
    for phase $p \leftarrow 1, 2, \ldots, l_{in} - 1$ do
        beep in slot 0 and slot 1
        listen in slot 2
    listen in slot 0
    if a beep was heard in slot 0 then beep in slot 1 and $role \leftarrow passive$
    else listen in slot 1
    if a beep was heard in slot 1 then $role \leftarrow inactive$ and terminate and return $\mathrm{id}_{out}$
    if $role = active$ then beep in slot 2
    else listen in slot 2
    if a beep was heard in slot 2 and slot 0 then $role \leftarrow inactive$ and terminate and
      return $\mathrm{id}_{out}$
    if  $role = passive$ then
        repeat
            listen in slot 0 and $l_{out} \leftarrow l_{out} + 1$
            if a beep was heard in slot 0 then beep in slot 1
            else listen in slot 1
            if a beep was heard in slot 1 then $role \leftarrow inactive$ and terminate and
              return $\mathrm{id}_{out}$
            listen in slot 2
            if no beep was heard in slot 2, and no beep in slot 0 and 1 then
              $role \leftarrow inactive$ and terminate and return $\mathrm{id}_{out}$
        until no beep is heard in a successive slot 0 and slot 1
Procedure campaign_highest_id:
    for phase $p \leftarrow 1, 2, \ldots, l_{out}$ do
        if $role = passive$ then
            listen in slot 0
            if a beep was heard in slot 0 then beep in slot 1
            else listen in slot 1
            if a beep was heard in slot 1 then $role \leftarrow inactive$, terminate and return
              $\mathrm{id}_{out}$
            listen in slot 2
            if no beep was heard in slot 2 and $p = l_{out}$ then $role \leftarrow inactive$ and
              terminate and return $\mathrm{id}_{out}$
        if $role = active$ then
            if at position $p$ of $\mathrm{id}_{in}$ is a 1 then beep in slot 0
            else listen in slot 0
            if at position $p$ of $\mathrm{id}_{in}$ is a 1 then beep in slot 1
            if a beep was heard in slot 0 then beep in slot 1 and $role \leftarrow passive$
            else listen in slot 1
            if a beep was heard in slot 1 then $role \leftarrow passive$
            if $p = l_{out}$ and $role = active$ then beep in slot 2
            else listen in slot 2
Procedure campaign_transmit_id:
    if $role = passive$ then $\mathrm{id}_{out} \leftarrow$ a bitstring of $l_{out}$ 0s
    for phase $p \leftarrow 1, 2, \ldots, l_{out}$ do
        if $role = active$ and at position $p$ of $\mathrm{id}_{in}$ is a 1 then beep in slot 0
        else listen in slot 0
        if $role = passive$ and a beep was heard in slot 0 then position $p$ of $\mathrm{id}_{out} \leftarrow 1$
        listen in slot 1 and slot 2
```

Figure A.4: The Algorithm `Campaigning`.