Singularly Optimal Randomized Leader Election

Shay Kutten
Faculty of Industrial Engineering and Management,
Technion – Israel Institute of Technology, Haifa, Israel
kutten@technion.ac.il

William K. Moses Jr.
Faculty of Industrial Engineering and Management,
Technion – Israel Institute of Technology, Haifa, Israel
wkmjr3@gmail.com

Gopal Pandurangan
Department of Computer Science, University of Houston, TX, USA
gopal@cs.uh.edu

David Peleg
Department of Computer Science and Applied Mathematics,
Weizmann Institute of Science, Rehovot, Israel
david.peleg@weizmann.ac.il

Abstract

This paper concerns designing distributed algorithms that are *singularly optimal*, i.e., algorithms that are *simultaneously* time and message *optimal*, for the fundamental leader election problem in networks. Our main result is a randomized distributed leader election algorithm for *asynchronous complete* networks that is essentially (up to a polylogarithmic factor) singularly optimal. Our algorithm uses $O(n)$ messages with high probability[1] and runs in $O(\log^2 n)$ time (with high probability) to elect a unique leader. The $O(n)$ message complexity should be contrasted with the $\Omega(n \log n)$ lower bounds for the deterministic message complexity of leader election algorithms (regardless of time), proven by Korach, Moran, and Zaks (TCS, 1989) for asynchronous algorithms and by Afek and Gafni (SIAM J. Comput., 1991) for synchronous networks. Hence, our result also separates the message complexities of randomized and deterministic leader election. More importantly, our (randomized) time complexity of $O(\log^2 n)$ for obtaining the optimal $O(n)$ message complexity is significantly smaller than the long-standing $\tilde{\Theta}(n)$ time complexity obtained by Afek and Gafni and by Singh (SIAM J. Comput., 1997) for message optimal (deterministic) election in asynchronous networks. Afek and Gafni also conjectured that $\tilde{\Theta}(n)$ time would be optimal for message-optimal asynchronous algorithms. Our result shows that randomized algorithms are significantly faster.

Turning to *synchronous* complete networks, Afek and Gafni showed an essentially singularly optimal deterministic algorithm with $O(\log n)$ time and $O(n \log n)$ messages. Ramanathan et al. (Distrib. Comput. 2007) used randomization to improve the message complexity, and showed a randomized algorithm with $O(n)$ messages but still with $O(\log n)$ time (with failure probability $O(1/\log^{\Omega(1)} n)$). Our second result shows that synchronous complete networks admit a *tightly* singularly optimal randomized algorithm, with $O(1)$ time and $O(n)$ messages (both bounds are optimal). Moreover, our algorithm's time bound holds with certainty, and its message bound holds with high probability, i.e., $1 - 1/n^c$ for constant c.

Our results demonstrate that leader election can be solved in a simultaneously message and time-efficient manner in asynchronous complete networks using randomization. It is open whether this is possible in asynchronous general networks.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Mathematics of computing → Probabilistic algorithms; Mathematics of computing → Discrete mathematics

Keywords and phrases Leader election, Asynchronous systems, Randomized algorithms, Singularly optimal, Complete networks

[1] Throughout, "with high probability" means with probability at least $1 - 1/n^c$, for a constant $c > 0$.

34th International Symposium on Distributed Computing (DISC 2020).
Editor: Hagit Attiya; Article No. 22; pp. 22:1–22:18

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Digital Object Identifier 10.4230/LIPIcs.DISC.2020.22

Related Version A full version of the paper is available at [27], https://arxiv.org/abs/2008.02782.

Funding *Shay Kutten*: This work was supported in part by the Bi-national Science Foundation (BSF) grant 2016419.
William K. Moses Jr.: This work was supported in part by the BSF grant 2016419 and in part by a Technion fellowship.
Gopal Pandurangan: G. Pandurangan was supported, in part, by NSF grants CCF-1527867, CCF-1540512, IIS-1633720, CCF-1717075, and BSF grant 2016419.
David Peleg: Supported in part by the US-Israel Binational Science Foundation grant 2016732.

1 Introduction

Leader election is a classical and fundamental problem in distributed computing with numerous applications; see e.g., [31, 12, 1, 18, 23, 25, 26, 29, 48, 28, 32, 33, 47, 20, 34, 39, 46, 50, 24, 42, 3, 43, 16, 30, 2]. The goal is to select a unique node, called the *leader*, from a set of nodes. An *arbitrary* subset of nodes can *wake up spontaneously at arbitrary times* and start the election algorithm by sending messages over the network. When the algorithm terminates, a *unique* node v must be elected as leader and be *known to all nodes*.

Election is important both theoretically and because of the multiple applications such as implementing databases and data centers [19, 9, 49], locks [7], file servers [14, 10], broadcast and multicast [41, 8], and virtually every global task [5]. In many of these applications, the network is treated as being virtually complete. With the advent of large-scale and resource-constrained networks such as peer-to-peer systems [22, 21, 4, 44, 45, 52] and ad hoc and sensor networks (e.g., [13, 51]), it is often desirable to achieve low cost and scalable leader election. Leader election has been studied extensively over the years in various distributed computing models starting with the standard CONGEST model of networks (in particular, ring, complete networks, and arbitrary networks), radio networks, peer-to-peer networks, population protocols, and programmable matter to name a few.

Our goal in this paper is to design leader election algorithms in distributed networks such that each is efficient with respect to *both of* the two fundamental complexity measures, namely *messages* and *time*. Unfortunately, designing distributed network algorithms that are *simultaneously* time- and message-efficient has proved to be a challenging task. Consequently, research in the last three decades has focused mainly on optimizing either one of the two measures separately, typically at the cost of neglecting the other. There has been significant recent progress in obtaining algorithms that are essentially optimal in both measures (or at least work well under both measures to the extent possible) for various problems such as leader election, minimum spanning tree and shortest paths [28, 38, 11, 17]. In particular, as defined in [38] (see also [15]), the following two notions are of interest for any given problem:

- **Singular optimality:** A problem enjoys *singular optimality* if it has a distributed algorithm that is optimal (or at least optimal up to a polylogarithmic factor) with respect to both measures simultaneously.
- **Time-message trade-off:** A problem exhibits a *time-message trade-off* if it fails to admit a singularly optimal solution, namely, algorithms of better time complexity for it necessarily incur higher message complexity and vice versa.

The singularly optimal results mentioned earlier for leader election, minimum spanning tree, and shortest paths [28, 38, 11, 17], crucially apply only to *synchronous* networks. A main motivation for this work is to study whether leader election admits singularly optimal

algorithms or exhibits a time-message tradeoff in *asynchronous* networks. Unlike synchronous networks which admit a leader election algorithm which is singularly optimal (up to a logarithmic factor) [28], it is not known whether asynchronous networks admit such an algorithm (see "Background and Prior Work").

In this paper, we focus on the classical problem of leader election in *complete* networks, which itself has been studied extensively for nearly four decades. In such a network, every pair of nodes is connected by a bidirectional communication link and in the beginning, no node has any knowledge of any other nodes, including their identities (if any). Like many prior works (e.g., [1, 47, 18, 12, 2] and others), this paper focuses on the more challenging *asynchronous* model and provides the first-known singularly optimal algorithm.

It is clear that the best possible time bound for leader election in complete networks is *constant* if there is no restriction on the message complexity. The situation with respect to the message complexity is more nuanced. (The message lower bound is trivially $\Omega(n)$ since the leader has to be known to all nodes.) For *deterministic* algorithms, there is a well-known lower bound of $\Omega(n \log n)$ messages (even in the synchronous setting when an adversary decides which nodes to wake up and when) [1, 26, 25] where n is the number of network nodes. Moreover, Afek and Gafni [1] show that any deterministic algorithm that is message optimal (i.e., takes $\Theta(n \log n)$ messages) needs $\Omega(\log n)$ time; they present such a deterministic algorithm that takes $O(n \log n)$ messages and $O(\log n)$ time in *synchronous* networks. Hence for synchronous networks, there exists a deterministic algorithm that is essentially singularly optimal (up to a logarithmic factor).

The situation is less clear in the asynchronous setting and for randomized solutions. First, the $\Omega(n \log n)$ message lower bound was shown for deterministic algorithms, and it was not clear whether by using randomization one can breach this bound. Second, Afek and Gafni [1] present an *asynchronous* leader election algorithm that takes $O(n \log n)$ messages (which is message optimal), but takes $O(n)$ time. They conjecture that this is the best possible time bound for message-optimal asynchronous algorithms. Singh [47] (mildly) disproves this conjecture by presenting an algorithm that runs in $O(n/\log n)$ time and is also message optimal ($O(n \log n)$ messages). This is still a far cry from being singularly optimal, as the time bound is essentially linear. When it comes to the use of randomness, Afek and Matias [2] develop a randomized algorithm that succeeds with probability $1 - \varepsilon$ and runs in $O(\log n)$ time using $O((n/\varepsilon) \log^2(1/\varepsilon))$ messages.[2] If we allow for a constant probability of success, i.e., if we set ε to a fixed constant, then the algorithm runs in $O(\log n)$ rounds and uses $O(n)$ messages. However, for randomized algorithms, it is typically desired that algorithms succeed with high probability. In such a scenario if we set ε to $1/n^c$ for some constant $c > 0$, the message complexity significantly increases to $O(n^{1+c} \log^2 n)$.

The results of the current paper (detailed later on) *break* the long-standing barrier of $\Omega(n \log n)$ messages for deterministic algorithms and show that there exist singularly optimal randomized asynchronous algorithms that succeed with high probability.

Distributed Computing Model. We consider a system represented as an undirected complete graph $G = (V, E)$, $|V| = n$, similar to the models of [1, 18, 23, 25, 26], except that processors can access *private unbiased coins*. Our upper bounds do not require unique

[2] In the paper, it is claimed that they develop an algorithm with termination detection that runs in $O(\log n)$ time for complete graphs. However, it is unclear if indeed the running time is $O(\log n)$ for an algorithm with termination detection or rather $O(n)$ time when termination detection is required.

identities, and in particular, nodes can be anonymous. If nodes do have unique identifiers, then we assume, as in prior works on complete networks (see [1]), that nodes initially do not know the unique identifiers of other nodes.[3]

In the synchronous communication setting, the computation is divided into discrete lockstep time units called *rounds*, and every message sent over an edge arrives at the receiver after a fixed delay of one time unit, namely, at the end of the current round. In contrast, in the *asynchronous* communication setting, messages sent on edges incur unpredictable but finite delay, in an error-free and FIFO manner (i.e., messages will arrive in sequence). Nevertheless, for the sake of time analysis it is assumed that a message takes *at most one time unit* to be delivered. For both modes, we make the usual (and here, very reasonable) assumption that local computation within a node is instantaneous and free. We assume the standard $\mathcal{CONGEST}$ model [40], where a node can send at most one $O(\log n)$ bit message on each edge in each round.

Following the standard assumption in asynchronous protocols (see [1, 12, 47]), nodes are initially asleep. A node enters the execution when it is woken up by the environment (at most once), or upon receiving messages from awakened neighbors. In the asynchronous setting, once a node enters execution, it performs all the computations required of it by the algorithm, sends out messages to neighbors as specified by the algorithm. In the synchronous setting, we assume the above *adversarial* wake up assumption as well.[4]

The message complexity of an algorithm is the worst-case total number of $O(\log n)$ bit messages sent during its execution. The time complexity is the worst-case total number of time units since the first node is woken up to the last message transmission due to the algorithm. Note that a time unit in the synchronous model corresponds to one round, whereas in the asynchronous model it is an upper bound on the transmission time of a message over an edge. Hence, it is generally more difficult to design efficient algorithms in asynchronous systems than in synchronous systems.

Following the standard approach to modeling distributed networks (see [1]), we represent the environmental uncertainties by means of an *adversary* controlling some of the execution parameters. Specifically, we assume an *adversarial wake up* mode, where node wake-up is scheduled by an adversary (who may decide to keep some nodes dormant). A node can also be woken up by receiving messages from other nodes. In addition to the wake-up schedule, the adversary also decides for how long to delay each message. These decisions are done *adaptively*, i.e., when the adversary makes a decision to wake up a node or delay a message, it has access to the results of all previous coin flips. Finally, recalling that initially the nodes are unaware of which neighbor is connected to each of their outgoing edges, the adversary also controls the graph structure (i.e., the mapping of outgoing edges to neighbors). Here, we consider the adversary to be *oblivious*, i.e., if nodes use randomness to choose an outgoing edge, the adversary must choose endpoints for all such outgoing edges prior to the first use of randomness in this way.[5]

---

[3] Otherwise (in the KT_1 model [6], where each node knows the unique identifiers of other nodes), leader election is trivial in complete networks.

[4] This should be contrasted with the *simultaneous wake up* model, where all nodes are assumed to be awake at the beginning of computation; this is typically assumed in design of synchronous protocols (see e.g., [1, 29, 28]).

[5] To appreciate the implications of allowing the adversary to construct the graph *after* seeing the random choices, see the lower bound proof techniques for message complexity in [26, 1]. These techniques are for deterministic algorithms, but will work when the adversary has adaptive edge mapping abilities.

Background and additional Prior Work. The complexity of the leader election problem and (especially deterministic) algorithms for it have been very well-studied in distributed networks. Various algorithms and lower bounds are known in different models with synchronous/asynchronous communication and in networks of varying network topologies such as a cycle, a complete graph, or some arbitrary topology (e.g., see [20, 23, 28, 29, 34, 39, 46, 50] and the references therein). The problem was first studied in context of a ring network by Le Lann [31] and discussed for general graphs in the influential paper of Gallager, Humblet, and Spira [12]. Kutten et al. [28] presented a singularly optimal (up to a logarithmic factor) randomized leader election algorithm for general *synchronous* networks that ran in $O(D)$ time and used $O(m \log n)$ messages (where D, m, and n are the network diameter, number of edges, and the number of nodes respectively). We note that $\Omega(D)$ and $\Omega(m)$ are lower bounds for time and messages for leader election even for randomized algorithms [28]. It is not known whether similar bounds can be achieved for general *asynchronous* networks, although one can obtain algorithms that are separately time optimal [39] and message optimal [12].

Leader election in the class of *complete networks* – which is the focus of this paper – has come to occupy a special position of its own and has been extensively studied [1, 2, 18, 23, 25, 26, 29, 48]; see also [32, 33, 47] for leader election in complete networks where nodes have a sense of direction. While $\Omega(n)$ is an obvious lower bound on the message complexity of leader election when the leader's identity should be known for all nodes, an $\tilde{O}(\sqrt{n})$ (i.e., sublinear) message complexity can be obtained for a related but different problem in *synchronous* complete networks with *simultaneous wake up* [29] where we *do not* require that the nodes not elected know who is the leader (nor which of their ports lead to it).[6] The above result crucially uses randomization to break the linear message complexity bound that applies for deterministic algorithms.

The study of leader election algorithms is usually concerned with both message and time complexity. Korach et al. [24], Humblet [18], Peterson [42] and Afek and Gafni [1] presented $O(n \log n)$ message algorithms for *asynchronous* complete networks. Korach, Kutten, and Moran [23] presented a general method plus applications to various classes of graphs including complete networks. Afek and Matias [2] similarly presented a general method with application to a complete graph.

Afek and Gafni (as a part of presenting a tradeoff between time and message complexity) showed that the time complexity of a message optimal *synchronous* algorithm was $\Theta(\log n)$, while for a message optimal *asynchronous* algorithm, they only demonstrated an $O(n)$ time upper bound (improving previous time bounds for message optimal algorithms). They conjectured that *in the asynchronous case, time complexity of any message optimal algorithm is* $\Omega(n)$. Singh [48] (as a part of presenting a different tradeoff), presented a somewhat better ($O(n/\log n)$) time for message optimal asynchronous algorithms, but still posed as an important open problem the question whether $\Omega(n/\log n)$ time was optimal for such algorithms. The current paper demonstrates that this is not the case, at least for randomized algorithms that succeed w.h.p., by giving a $O(\log^2 n)$ time bound for message optimal $O(n)$ algorithm.

The above mentioned papers on "*sense of direction*" demonstrated that when the nodes possessed additional knowledge on the topology, it was possible to reduce the number of messages to $O(n)$. In the same vein, note that for *deterministic* algorithms in *synchronous* networks and under the strong assumption of *simultaneous wake up*, there exists an $O(n)$

[6] This variant of leader election is sometimes called "implicit", as opposed to the version studied here, where all nodes need to know the identity of the leader.

messages algorithm [28]. In contrast, an $\Omega(n \log n)$ message lower bound in synchronous networks was shown by Afek and Gafni [1] under the more common assumption of adversarial wake up. The message complexity in the current paper is $O(n)$ without using sense of direction or simultaneous wakeup assumptions (but using randomized algorithms).

For anonymous networks under some reasonable assumptions, deterministic leader election was shown to be impossible, using symmetry arguments [3]. Randomization comes to the rescue in this case; random rank assignment is often used to assign unique identifiers, as done herein. Randomization also allows us to beat the lower bounds for deterministic algorithms, albeit at the risk of a small chance of error. For example, Afek and Matias [2] developed a randomized algorithm that succeeded with probability $1 - \varepsilon$ and ran in $O(\log n)$ time using $O((n/\varepsilon) \log^2(1/\varepsilon))$ messages. It should be noted that Singh's [47] deterministic algorithm allowed a trade-off between time and memory such that it was possible to achieve a running time as low as $O(\log n)$ in exchange for more messages ($O(n^2/\log n)$). By setting ε to $1/n^c$, for a constant $c > 0$, we see that Afek and Matias's algorithm succeeds with high probability and takes less messages ($O(n^{1+c} \log^2 n)$) for the same running time.

Turning to *synchronous* complete networks, Afek and Gafni showed an essentially singularly optimal deterministic algorithm with $O(\log n)$ time and $O(n \log n)$ messages. Ramanathan et al. used randomization to improve the message complexity, and showed a randomized leader election algorithm for synchronous networks that could err with probability $O(1/\log^{\Omega(1)} n)$ with time $O(\log n)$ and $O(n)$ messages[7] [43]. That paper also extends the synchronous algorithm to work for *partially* synchronous networks, where message delays are bounded. It also surveys some related papers about randomized algorithms in other models that use more messages for performing leader election [16] or related tasks (e.g., quorum systems, Malkhi et al. [35]). In the context of self-stabilization, a randomized algorithm with $O(n \log n)$ messages and $O(\log n)$ time until stabilization was presented in [30].

Table 1 Comparison of previous upper bound results for leader election in complete networks of n nodes along with our contributions. The variables $2 \leqslant c \leqslant n$, $\log n \leqslant k \leqslant n$, and $0 < \varepsilon < 1$ are parameters provided to the respective algorithms.

Paper	Message Complexity	Time Complexity	Communication Mode	Type of Solution
[24]	$O(n \log n)$	$O(n \log n)$	Asynchronous	Deterministic
[1]	$O(n \log n)$	$O(n)$	Asynchronous	Deterministic
[47]	$O(nk)$	$O(n/k)$	Asynchronous	Deterministic
[2]	$O(n \log^2(1/\varepsilon)/\varepsilon)$	$O(\log n)$	Asynchronous	Randomized, success prob. $1 - \varepsilon$
This paper	$O(n)$ w.h.p.	$O(\log^2 n)$ w.h.p.	Asynchronous	Randomized, success w.h.p.*
[1]	$O(cn \log_c n)$	$O(\log_c n)$	Synchronous	Deterministic
[43]	$O(n)$	$O(\log n)$	Synchronous	Randomized, success prob. $1 - O(1/\log^{\Omega(1)} n)$
This paper	$O(n)$ w.h.p.	$O(1)$	Synchronous	Randomized, success w.h.p.*

*The algorithm always succeeds when nodes have unique identifiers (as opposed to anonymous networks).

Our Main Results. The main focus of this paper is on studying how randomization can help in designing singularly optimal algorithms for leader election in asynchronous as well as synchronous networks. Our results are summarized in Table 1. Our main result is a randomized asynchronous leader election algorithm for complete networks that runs in

[7] In contrast, the synchronous algorithm presented in the current paper succeeds with high probability, its time complexity is constant, and its complexity bounds hold with probability $1 - O(1/n^{\Omega(1)})$.

$O(\log^2 n)$ time and uses only $O(n)$ messages to elect a unique leader with high probability (Section 2). This is a significant improvement over the $\Omega(n \log n)$ messages needed for any deterministic algorithm, and an even larger improvement over the time complexity of previous message optimal ($O(n \log n)$) deterministic algorithms for asynchronous networks, which required $O(n)$ or $O(n/\log n)$ time [1, 47]. In addition, we show that for the synchronous setting too, we can obtain a singularly optimal algorithm that tightly matches the best possible asymptotic time and message lower bounds. We present a randomized algorithm in synchronous networks that takes $O(1)$ time and $O(n)$ messages with high probability (see Section 3). Note that our algorithms succeed w.h.p. in anonymous networks and always succeed when each node has a unique identifier.

Our results, while providing near-optimal message and time bounds for complete asynchronous networks, are a step towards designing singularly optimal algorithms for *general* asynchronous networks.

2 A Randomized Algorithm for Asynchronous Networks & Analysis

In this section, we first give some high level intuition of how we use randomization to overcome the barriers of time and message complexity posed to deterministic asynchronous algorithms. Subsequently, we present a randomized algorithm that solves leader election in $O(\log^2 n)$ time with high probability using $O(n)$ messages with high probability.

High level intuition behind the use of randomness. In [1], an $\Omega(n \log n)$ message lower bound is presented. A key idea in the proof is that the adversary is able to control the destination of messages sent by a node. It can do this for a deterministic algorithm because the adversary may predict which edges will be used in the course of the algorithm and construct the initial graph accordingly. Intuitively, if multiple clusters of nodes are active, a leader cannot be determined until they interact. If the adversary can consistently control the destination of messages over unknown links, the adversary may blow up the number of messages until the clusters interact. We bypass this issue by having nodes choose the edges they use to communicate at random, similar to the idea used in [29]. Since the adversary cannot predict which edge will be used, it cannot initially construct an undesirable graph. Our second use of randomness is to have each node, once awake, choose its $RANK$ uniformly at random from $[1, n^4]$. This helps us achieve a significantly smaller running time compared to the deterministic case. If $RANK$s are not randomly chosen but deterministically assigned, then the adversary may wake up nodes such that it takes $O(n)$ time from the time the first node is woken up until the algorithm terminates. This is because, in our algorithm, a node's $RANK$ plays an important role in deciding if it will go on to become the leader when interacting with another node or if it will stop trying to be the leader.

The Randomized algorithm. We now describe the algorithm. Each awake node may play up to two roles in the algorithm, namely (i) a candidate and (ii) a referee. During the process, candidates attempt to progress towards becoming the leader. The position of a candidate u in this process is represented by the pair $\langle RANK_u, PH_u \rangle$, where $RANK_u$ is u's *rank* and PH_u is its current *phase*. Intuitively, we say that candidate v is *ahead of* candidate u, and u is *behind* v, if v has a higher phase number, or, in the case of a tie, v has a higher RANK. Formally, we say that $v \gg u$ if either (i) $PH_v > PH_u$, or (ii) $PH_v = PH_u$ and $RANK_v > RANK_u$.[8]

[8] If nodes have unique identifiers, then in the case of two nodes in the same phase with the same rank, their unique identifiers can be used to break ties. Furthermore, a node will append its unique identifier

During the process, candidates gradually either *progress* or *retire*, eventually resulting in a single un-retired candidate who then becomes the leader. Essentially, a candidate u retires upon encountering another candidate v that is ahead of it. Such an encounter occurs when a referee learns of both candidates, and it is the referee's task to make one of the candidates retire. (The referee is typically a third party, but may also be u or v.) Hence, candidates fundamentally drive the algorithm forward, while referees serve a complementary role of guardians, assisting candidates in deciding whether to retire or to proceed. Each node is in one of three candidate states Candidate, Non-elected, Elected, stored in the variable CAND-STATE, corresponding to whether the node is still a candidate in the running to become a leader, is retired, or is elected, respectively.

If a node is woken up by the adversary, then it plays the role of a candidate and may eventually become the leader. If a node is woken up by a message from a neighbor, then it is immediately considered to be a retired candidate. When a node receives a message, depending on the type of message received, the node may either act in the role of a candidate or in the role of a referee, with an appropriate procedure called to handle the message. Thus, during the execution, a node either acts as just a referee or as both a candidate and a referee.

Each node u, once woken up, checks if it was woken by the adversary (i.e., spontaneously wakes up itself) or by a message from a neighbor. If u was woken by a neighbor, u sets its candidate state to Non-elected. If u was woken by the adversary, u sets its candidate state to Candidate, chooses an integer in $[1, n^4]$ uniformly at random as its ID_u and attempts to progress. The initialization is described in Algorithm 1 in the full version.

A candidate u progresses through at most $K = \lceil \log \sqrt{4n \log n} \rceil + 1$ phases until it either changes its candidate state to Non-elected or completes phase K and stays in candidate state Candidate, in which case it declares itself leader. At the beginning of each phase $i = 1$ to $K - 1$, a candidate node u chooses a set $\mathcal{S}$ of $\min\{10 \cdot 2^i, \lceil \sqrt{4n \log n} \rceil\}$ nodes uniformly at random designated as its *referees* and seeks their approval to progress.[9] In phase $i = K$, u sets $\mathcal{S} \leftarrow V$, i.e. all nodes of the network form set $\mathcal{S}$. Node u sends a *request* message $\langle RANK_u, PH_u, \text{REQUEST} \rangle$ to each referee in $\mathcal{S}$.

When node u receives replies from all nodes in $\mathcal{S}$, it checks if any of those replies is a decline of the form $\langle RANK_u, PH_u, \text{DECLINED} \rangle$. If so, u changes its candidate state to Non-elected.[10] In case u received no declines, it may proceed. In particular, if $i < K$, then u increases PH_u to $i + 1$ and starts the next phase, and if $i = K$ and u still retains candidate state Candidate, then u declares itself as leader, namely, it changes its candidate state to Elected, broadcasts the final announcement $\langle RANK_u, 0, \text{LEADER} \rangle$ and terminates. The above process is described in Procedure 2 in the full version.

Each referee r helps candidates retire (until only one is left) by comparing candidate pairs, keeping the more advanced one, and instructing the other to retire. Referee r can be in one of four referee states stored in the variable REF-STATE.

- REF-STATE $= C0$ holds when r has not been approached by any candidate yet.
- REF-STATE $= C1$ holds when r has one approved candidate, referred to as its *chosen* candidate, and has declined every other candidate that approached it so far. A record containing the position of the chosen candidate v, namely, $\mathcal{P}(v) = \langle RANK_v, PH_v \rangle$, is kept in the variable Chosen.

to its $RANK$ to avoid ambiguity.

[9] We assume that node u treats itself as a referee as well in addition to the nodes of $\mathcal{S}$.

[10] The candidate state of u can also be changed to Non-elected as a result of processing a DECIDE message, described later.

- REF-STATE $= C2$ holds when r currently keeps track of two candidates: the chosen v (in the variable Chosen) and a contender w (in the variable Contender), such that $w \gg v$, and all other candidates that approached r so far were declined. Moreover, a *dispute* is currently in progress between v and w. This state is typically reached when r has a chosen candidate v that was approved by it, and later it gets a request from another candidate w such that $w \gg v$. In this situation, r cannot decline w (since it is ahead of its current chosen), but at the same time it cannot approve w, since it may be that v has progressed in the meantime, and it is now ahead of w. To resolve this uncertainty, r declares a dispute, and sends a DECIDE message containing w's position $\langle RANK_w, PH_w \rangle$ to v, asking it to make a comparison between w and itself (based on its current phase PH_v). While waiting for v's response, r keeps a record containing the details of w, namely, $\mathcal{P}(w) = \langle RANK_w, PH_w \rangle$, in the variable Contender.
- REF-STATE $= C3$ is similar to $C2$, i.e., it holds when r currently keeps track of a chosen v and a contender w, all other candidates that approached r so far were declined, and a dispute is currently in progress. The difference, however, is that the on-going dispute does not involve w. Rather, it is between v and some *previous* contender z such that $w \gg z \gg v$. This state is typically reached when a new candidate w approaches r while r is in referee state $C2$ with a dispute in progress between a chosen candidate v and a contender $z \gg v$, and r discovers that $w \gg z$. This allows r to decline the current contender z immediately (since even if the outcome of the dispute favors z over v, the new candidate w is ahead of z, so z must retire). Now w takes z's place as the contender.

If r receives a message $\langle RANK_u, PH_u, \text{REQUEST} \rangle$ from node u over some edge e, it responds as follows.

- If r is in referee state $C0$, then it registers u as its chosen candidate, sends back an approval message, and switches its referee state to $C1$.
- If r is in referee state $C1$, then the following sub-cases may apply: If the current chosen is also u (from an earlier phase), then r updates the record stored in Chosen. If it is another node v such that $v \gg u$, then r sends u a decline message. Otherwise (i.e., if $u \gg v$), r registers u as the contender and initiates a dispute by sending a DECIDE message to v, requesting it to compare its current position with that of u. It also switches its referee state to $C2$.
- If r is in referee state $C2$, signifying that a dispute is in progress, then r compares the new candidate u with the current contender w. If $u \ll w$ then r declines u's request (and thus retires u). Otherwise (i.e., if $u \gg w$), r retires w, registers u as the new contender, and switches its referee state to $C3$.
- If r is in referee state $C3$, then it does the same as in referee state $C2$.

The pseudocode for the referee's actions on receiving a request is given in Procedure 3 in the full version.

When a node v which is currently the chosen candidate of the referee r (but may have possibly retired since the time it was approved by r) receives from r an $\langle RANK_u, PH_u, \text{DECIDE} \rangle$ message, it must decide whether to end its candidacy (if it is still a candidate) or that of the other candidate. To do so, it compares its own current position with that of the contender u.

- If v is in candidate state Non-elected or $v \ll u$, then v returns the message $\langle RANK_u, PH_u, \text{WINS} \rangle \circ \langle RANK_v, PH_v, \text{LOSES} \rangle$.
- Otherwise (v is still in candidate state Candidate and is ahead of u), it returns the message $\langle RANK_u, PH_u, \text{LOSES} \rangle \circ \langle RANK_v, PH_v, \text{WINS} \rangle$.

Pseudocode for the chosen's actions on receiving a DECIDE message from a referee is given in Procedure 4 in the full version.

Once the chosen's reply message is received by r, it is processed as follows.

- If v replied that the contender u has won the dispute, r sends an approval message to the current contender (which is u if the referee state is $C2$, and another candidate if the referee state is $C3$), makes it the chosen, and switches to referee state $C1$.
- If v replied that it has progressed beyond the contender u, so u has lost the dispute and must retire, then there are two sub-cases to consider. If the referee state is $C2$, r sends a decline message to the contender u, and switches to state $C1$. Now suppose the referee state is $C3$ and the current contender is some candidate w. If v's current position is such that $v \gg w$, then r sends a decline message to w and switches to referee state $C1$. Otherwise ($w \gg v$), a new dispute is required, this time between v and w.

The referee's actions on receiving a reply from the chosen about an ongoing dispute are given in Procedure 5 in the full version.

Analysis of the algorithm. We establish the following theorem.

▶ **Theorem 1.** *Consider a complete anonymous network G of n nodes in the $\mathcal{CONGEST}$ model. Assume communication is asynchronous with adversarial wakeup. Then there is a randomized algorithm to solve leader election with high probability in $O(\log^2 n)$ time with high probability using $O(n)$ messages with high probability. If each node has a unique identifier, then the algorithm always succeeds. All nodes terminate at the end of the algorithm.*

We show three properties: exactly one leader is chosen w.h.p. for anonymous networks and always when nodes have unique identifiers and all nodes subsequently terminate, the time complexity is $O(\log^2 n)$ w.h.p., and the message complexity is $O(n)$ w.h.p.

Before we continue with the proof, we make an important observation.

▶ **Observation 1.** *For each node u that participates in the algorithm, if u begins phase i, then u will finish phase i.*

Observation 1 is used implicitly in the remaining proof whenever a node is mentioned to finish some phase and possibly perform some calculation as a result of completing that phase.

▶ **Lemma 2.** *By the end of the algorithm, exactly one node is in candidate state* Elected *w.h.p. for anonymous networks and always when nodes have unique identifiers and the remaining nodes are in candidate state* Non-elected*. Furthermore, all nodes eventually terminate.*

Proof. Let us consider a network where each node has a unique identifier. We show that exactly one leader is always chosen by arguing that exactly one $RANK$ is chosen as the leader. It is clear to see that if we then consider an anonymous network where each node chooses its $RANK$ from $[1, n^4]$ and nodes do not have access to unique identifiers to break ties, the $RANK$ chosen as the leader will belong to not more than one node w.h.p.

We prove by induction that at least one node is a candidate at the end of phase i for $1 \leqslant i \leqslant K$. For the induction basis, consider, for the sake of the proof, that when a node wakes up, it is in phase 0, but it immediately moves to phase 1. Clearly, this does not change the outcome of the algorithm. Now the lemma holds for phase zero since at least one node is woken up by the adversary initially, and the induction is proven with phase zero as base case.

Assume that the claim holds true up to the end of some phase $k < K$. Let u be the candidate with largest RANK in phase $k+1$. If u is not retired by any other node in phase $k+1$, then it will be in candidate state Candidate at the end of phase $k+1$. Hence the claim holds for $k+1$. Otherwise, if u is retired in phase $k+1$, then necessarily either u came into contact with (i) some candidate u' in a higher phase $PH_{u'} > k+1$ or a referee to such

a candidate, (ii) a candidate u'' with higher rank $RANK_{u''} > RANK_u$ in the same phase $k+1$ or a referee to such a candidate, or (iii) a candidate v that completed all phases and is still in candidate state Candidate or a message from such a candidate v. In all cases, the induction claim still holds true for $k+1$.

We subsequently show that exactly one node will change its candidate state to Elected. By the previous claim, at least one candidate completes phase $K-1$. Let u_{max} be the node with the largest RANK that is still a candidate at the beginning of phase K. In phase K, u_{max} will approach all other nodes, receive an approval from each of them, and subsequently change its candidate state to Elected. Any other candidate v will approach u_{max}, receive a DECLINED message from it, and thus change its candidate state to Non-elected. Thus exactly one node will change its candidate state to Elected and broadcast a $\langle \cdot, 0, \text{LEADER} \rangle$ message.

Finally, we argue that all nodes terminate. It is clear that if exactly one node changes its candidate state to Elected and broadcasts a $\langle \cdot, 0, \text{LEADER} \}$ message, then that node terminates the algorithm. All other nodes change their candidate state to Non-elected and terminate the algorithm upon receiving this message. Thus, the proof is complete. ◀

We now prove the time complexity bound. First, we prove the following useful lemma about the amount of time one phase for a candidate takes.

▶ **Lemma 3.** *If a node u starts some phase i as a candidate, then it exits the phase (either by increasing its phase to $i+1$ or by retiring) within $O(1)$ time.*

Proof. For any candidate u, for each phase i, the largest delay until that phase ends occurs when candidate u sends a message to a referee r, which in turn sends a message to its current chosen v. Subsequently, v sends a reply to r, which in turn sends a reply to u. If the congestion on both these edges is $O(1)$ messages in the time interval from start to end of phase i, then the total time duration of the phase is $O(1)$. Let $e_{u,r}$ and $e_{r,v}$ denote edges between u and r and between r and v resp. We show at most $O(1)$ messages need to be transmitted on both $e_{u,r}$ and $e_{r,v}$ during phase i, thus allowing each phase to take $O(1)$ time.

Consider edge $e_{u,r}$. In phase i, u can send an invite to r and receive a reply to the invite from r. Furthermore, u can also be the chosen for r and receive a DECIDE message from r and subsequently send back a reply.[11] Additionally, r may also be a candidate with u as its referee or r may be the chosen for u, resulting in at most a doubling of messages over the edge. Thus at most $O(1)$ messages are sent across that edge for phase i of u. Now consider edge $e_{r,v}$. A similar analysis as above renders $O(1)$ an upper bound on the messages on the edge. Thus, we see that each phase takes $O(1)$ time units. ◀

In order to bound the running time, we also make use of the following useful property from [37], denoted in the reference as Problem C.2 in Appendix C.6. We slightly modify the statement to suit our needs.

▶ **Lemma 4.** *(Problem C.2 in [37]) Let $a_1, a_2, \ldots, a_n$ be a sequence of n values. Each value a_i is independently and randomly chosen from a fixed distribution $\mathcal{D}$. Let $m_i = \max\{a_1, a_2, \ldots, a_i\}$, i.e., the maximum of the first i values. Let random variable Y denote the number of times the maximum value is updated, i.e., the number of times $m_i \neq m_{i+1}$. Then $E[Y] = O(\log n)$.*

[11] Recall that once some node u, in phase i, is the chosen of some node r and receives a DECIDE message, either u is retired or r learns that u is in phase i and will not send any further messages from other candidates in phase $< i$. If another DECIDE message is subsequently sent to u from r, then either u wins (implying it is in a higher phase), or else u is retired and no further messages are sent along the edge.

▶ **Lemma 5.** *The running time of the algorithm is $O(\log^2 n)$ with high probability.*

Proof. For any given execution of the algorithm, eventually one candidate, C_ℓ, woken up by the adversary, becomes the leader. We show two properties for any such C_ℓ. First, once C_ℓ is woken up by the adversary, the algorithm takes an additional at most $O(K)$ time until C_ℓ broadcasts that it is the leader and the algorithm terminates for every node. Second, we bound the time from when the first node is woken up by the adversary to when candidate C_ℓ is woken up as $O(K \cdot \log n)$ with high probability.

Once C_ℓ is woken up, it must complete K phases before it broadcasts that it is the leader. By Lemma 3, each of its phases takes $O(1)$ time to complete. Thus, once C_ℓ is woken up by the adversary, it takes at most $O(K)$ time until all nodes terminate.

Now, we bound the time until C_ℓ is woken up. For this proof, we say that two candidates u and v *encounter* each other at time t when a referee (which could be either u or v itself) is made aware of both u and v for the first time. If, as a result of the encounter, v is retired, then we call u the *winner* of the encounter and v the *loser*. Note the following observation.

▶ **Observation 2.** *Consider some execution of the algorithm where a candidate u while in phase k retires candidate w. Subsequently, u reaches phase i and is retired by another candidate v in phase j. When u and v encounter each other, then either $j > i$ or v has a larger $RANK$ than u and is also in phase i. Furthermore, from the time u retired w, at most $O(i-k)$ time units have passed until the encounter between u and v (by Lemma 3) and an additional $O(1)$ time units pass until u retires.*

Observation 2 has the following implications. Either the winner of an encounter has a larger $RANK$ than the loser or else the phase number of the winner is larger than that of the loser when they encounter one another. Furthermore, the amount of time that passes between two encounters involving a common candidate can be bounded by the difference in phases of that candidate plus $O(1)$ time units.

We now construct a new graph where the nodes are the subset of the nodes that are woken up by the adversary (and which become candidates). There is an edge from candidate C_i to C_j if candidate C_i encounters and subsequently is retired by C_j. Notice that this graph is a directed acyclic graph since once a candidate C_i is retired by some other node C_j, C_i cannot go on to retire C_j or any candidate that retires C_j. Notice also that a candidate may encounter multiple other candidates, resulting in nodes with an in degree or out degree (or both) that is > 1. Finally, note that candidates that do not retire any other candidates are sources and C_ℓ is a sink. Consider a path in the graph from a source to C_ℓ such that the node first woken up by the adversary is one of the nodes in this path. If there are multiple such paths, choose one arbitrarily. Label the nodes from the source to the node just before C_ℓ as $C_1, C_2, \ldots, C_w$. Let C_1 be in some phase p at the time of its encounter with C_2. Let R be the time that pass from the first encounter until the final encounter in the chain. Then, the total time from when the first node was woken up until C_ℓ was woken up is upper bounded by $O(p + R) = O(K + R)$. We now bound the value of R.

With each of the candidates C_i, $1 \leqslant i \leqslant w$, associate a bit b_i which indicates how C_i was retired. Specifically, b_i is set to 0 if C_{i+1} was in a higher phase than C_i at the time of the encounter.[12] Bit b_i is set to 1 if C_{i+1} has a higher $RANK$ than C_i and is in the same phase as it at the time of the encounter. By Observation 2 and its implications, we see that every bit b_i, $1 \leqslant i \leqslant w$ is set to 0 or 1.

[12] Note that we are referring to the actual phases C_i and C_{i+1} are in at this time, not necessarily the phase numbers stored in the referee when it was first aware of both of them, as one of these values may be outdated. Also note that if $i = w$, then C_{i+1} refers to C_ℓ.

Consider the bit string $B = b_1 b_2 \dots b_w$. We will show that R is upper bounded $O(K+|B|)$. In order to bound the size of B, we first bound the number of 0's that can be present in B, and then we bound the number of 1's that can be present between any two 0's in B.

Denote by P_i the phase of C_i when it was retired. Observe that, regardless of whether $b_i = 0$ or 1, the phase P_{i+1} of the subsequent candidate C_{i+1} can never be less than that of the candidate C_i it just retired. Furthermore, when $b_i = 0$, $P_{i+1} > P_i$. Thus, there can be at most K bits set to 0 in B. We now bound the number of bits set to 1 between any two bits that are set to 0.

Between any two bits set to 0, at most $n-1$ nodes can be woken up by the adversary to trigger an encounter leading to a bit being set to 1. An encounter where a bit is set to 1 involves the $RANK$ of the awakened node being higher than that of the currently considered candidate, resulting in the higher $RANK$ node becoming the currently considered candidate. By Lemma 4, we see that on expectation, $O(\log n)$ such encounters can thus be triggered. Applying a simple Chernoff bound, we see that $O(\log n)$ is in fact a high probability upper bound on the number of such encounters, and by extension the number of bits set to 1 between any two bits set to 0. Since there are at most K bits set to 0, $|B| = O(K \cdot \log n)$.

Each encounter of two candidates contributes $O(1)$ time towards the running time. Furthermore, we must account for the time between encounters as well. Between any two encounters, the phase of a candidate may increase. We have already seen that if candidate C_i is in phase P_i, then for subsequent candidates C_j where $j > i$, $P_j \geqslant P_i$. From Lemma 3, we see that for any candidate a phase takes $O(1)$ time. Therefore, the total number of time units between all encounters is at most $O(K)$ time units. Thus, R can be upper bounded by $O(K + |B|) = O(K \cdot \log n)$ and the total running time is $O(K + R) = O(\log^2 n)$ time with high probability. ◀

We now prove that the message complexity of the randomized algorithm is $O(n)$ with high probability. We first show that the number of candidates that can participate in every phase is reduced by a factor of four from one phase to the next up to phase $\rho = K - \lceil \log \log n \rceil - 5$ with high probability.

▶ **Lemma 6.** *The total number of candidates that participate in phase i of the algorithm is at most $\lceil n/4^{i-1} \rceil$ with high probability for $1 \leqslant i \leqslant \rho$.*

Proof. We prove the claim by induction. Initially, even if the adversary wakes up all nodes, at most n candidates participate in phase 1. Assume that the claim holds true until some phase $k < \rho$. We prove that the claim holds in phase $k+1$ when the number of candidates participating in phase k is upper bounded by $\lceil n/4^{k-1} \rceil$.

Notice that if $\leqslant \lceil n/4^k \rceil$ candidates participate in phase k, then the claim holds immediately for phase $k+1$. Now, let us assume that the number of candidates participating in phase k lies in $(\lceil n/4^k \rceil, \lceil n/4^{k-1} \rceil]$.

Organize the candidates in increasing order of their $RANK$s. Group the top one-sixteenth of these candidates into the set *top* and group the remaining candidates into the set *bottom*. We ignore the retirement of candidates from *top* in phase k by assuming that none of them are retired in this phase and show that a sufficient number of candidates from *bottom* are retired for the claim to hold in phase $k+1$.[13] In this phase, there are at least $1/16 \cdot \lceil n/4^k \rceil$

[13] Notice that the claim is an upper bound on the number of candidates in each phase. By showing that the claim holds when none of the candidates from *top* is retired, it is easy to see that the claim also holds when at least one candidate from *top* is retired.

candidates in *top*, each of which makes $10 \cdot 2^k$ requests.[14] Thus, altogether, candidates from *top* make at least $m_0 = (\lceil n/4^k \rceil / 16) \cdot 10 \cdot 2^k \geqslant 5n/2^{k+3}$ requests uniformly at random.[15]

Consider a candidate u from *bottom*. Let r be some specific referee approached by u.

Call r *top-free* if none of the candidates from *top* has approached r and r itself is not in *top*. Then the probability that r is top-free is at most $p_0 = (1-1/n)^{m_0} \cdot 15/16 \leqslant (1-1/n)^{m_0}$. Note that u will be retired if even one of its requests is to a non-top-free referee. Therefore, the probability that u is not retired in this phase is at most $p_0^{10 \cdot 2^k} \leqslant (1-1/n)^{(5n/2^{k+3}) \cdot 10 \cdot 2^k} \leqslant e^{-25/4}$. Thus, denoting the number of candidates from *bottom* that are not retired by NR_B, we have $\mathbb{E}[NR_B] \leqslant (15/16) \cdot \lceil n/4^{k-1} \rceil \cdot e^{-25/4} \leqslant (15/2) e^{-25/4} \cdot n/4^k \leqslant (1/16) \cdot n/4^k$. As NR_B is the sum of independent Bernoulli trials, we can apply a Chernoff bound (second bound of Theorem 4.4 in [36]). When $k < \rho$, the probability that NR_B exceeds twice the expected value is upper bounded by $1/n^3$. Thus the total number of candidates that can participate in phase $k+1$ with high probability is at most $(1/16) \cdot \lceil n/4^{k-1} \rceil + (2/16) \cdot (n/4^k) \leqslant \lceil n/4^k \rceil$.

Note that for each phase, the upper bound on the number of candidates who are not retired holds with probability $1 - 1/n^3$, assuming that the bound held in the previous phase. Applying a union bound over all phases of the induction, we see that this upper bound holds for phase ρ with probability $1 - O(1/n^2)$, i.e. w.h.p., and for each previous phase with a larger probability. Thus the claim is true for all $1 \leqslant i \leqslant \rho$ with high probability. ◀

We next show that in the final phase of the algorithm, only a single candidate will not be retired with high probability.

▶ **Lemma 7.** *At the end of phase $K-1$, exactly one node will be in candidate state* Candidate *with high probability.*

Proof. Consider the candidate with the largest RANK, u_{max}, in phase $K-1$. We show that for each candidate $v \neq u_{max}$ in this phase, the intersection of referees approached by v and u_{max} is non-zero with high probability. Thus u_{max} will retire each such candidate v with high probability. Notice that $10 \cdot 2^K \geqslant \lceil \sqrt{4n \log n} \rceil$ and so each candidate in this phase makes exactly $\lceil \sqrt{4n \log n} \rceil$ requests. Then

$$\mathbb{P}[v \text{ is not retired by } u_{max}] \;=\; (1-1/n)^{\lceil \sqrt{4n \log n} \rceil \cdot \lceil \sqrt{4n \log n} \rceil} \;\leqslant\; 1/n^4.$$

By Lemma 6, there are at most $\lceil 2^{13} \log n \rceil$ candidates participating in phase ρ. As the number of candidates participating in subsequent phases cannot increase beyond this value, it acts as an upper bound for the number of candidates participating in phase $K-1$.

We see that the probability that any of these candidates is not retired is at most $1/n^3$ through the use of a union bound. Thus, w.h.p. u_{max} retires every other candidate in this phase. Thus only u_{max} ends the phase in candidate state Candidate w.h.p. ◀

▶ **Lemma 8.** *The message complexity of the algorithm is $O(n)$ messages with high probability.*

Proof. Let $\psi = \lfloor \log(1/10 \cdot \lceil \sqrt{4n \log n} \rceil) \rfloor$. We start by analysing the message complexity that can be "charged" to candidates in phases 1 to ψ. In each phase $i \leqslant \psi$, each candidate u generates and receives up to $2 \cdot 10 \cdot 2^i$ messages from its referees. Furthermore, for each of its

[14] Notice that for all phases i that the claim applies to, $10 \cdot 2^i \leqslant \lceil \sqrt{4n \log n} \rceil$. Thus, any candidate in one of these phases will make $10 \cdot 2^i$ requests.

[15] Note that the requests by a candidate in a single phase are made without repetition, i.e. a candidate does not send more than one request along the same edge in a single phase. During the subsequent analysis, we consider them to be made with repetition so as to simplify the analysis. This assumption only decreases the probability of a candidate from *bottom* inviting a referee that was approached by one of the *top* candidates and thus this simplifying assumption is acceptable.

referees in a given phase i and from all previous phases, a candidate may receive a message to decide candidacy, resulting in at most $\sum_{j=1}^{i}\sum_{k=1}^{j} 2\cdot 10\cdot 2^k \leqslant \sum_{j=1}^{i} 2\cdot 10\cdot 2^{j+1} = 5\cdot 2^{i+4}$ additional messages received and generated by the candidate in this phase. (For a candidate u and a referee r chosen by u in phase i or a previous one, only one DECIDE message will be sent from r to to u.)

In each phase $1 \leqslant i \leqslant \rho$, there are at most $\lceil n/4^{i-1}\rceil$ candidates w.h.p. by Lemma 6. As calculated earlier, we have $5\cdot 2^{i+2} + 5\cdot 2^{i+4} = 25\cdot 2^{i+2}$ messages per candidate in each phase i, resulting in a total of $O(n)$ messages w.h.p. generated across all candidates in all phases.

In phases $\rho+1 \leqslant i \leqslant \psi$, there are at most $\lceil 2^{13}\log n\rceil$ candidates in each phase with high probability by Lemma 6, each making $10\cdot 2^i$ requests in that phase. Thus, the total number of messages over these phases is $\sum_{i=\rho+1}^{\psi}\lceil 2^{13}\log n\rceil 25\cdot 2^{i+2} = O(\sqrt{n}\log^{3/2} n)$ w.h.p.

Now we analyze the message complexity attributed to candidates in phases ψ and above. In each phase $i > \psi$, each candidate u generates and receives $2\cdot\lceil\sqrt{4n\log n}\rceil$ messages from its referees. Furthermore, for each of its referees in a given phase i and from all previous phases, u may receive a message to decide candidacy, resulting in at most $2\lceil\sqrt{4n\log n}\rceil(i-\psi)$ additional messages due to DECIDE messages from referees of phases $j > \psi$, plus $5\cdot 2^{\psi+4}$ additional messages due to DECIDE messages from referees of phases $j \leqslant \psi$. Thus, at most $2\cdot(i+5-\psi)\cdot\lceil\sqrt{4n\log n}\rceil$ messages are received and generated by u in this phase.

In phases $i > \psi$, there are most $\lceil 2^{13}\log n\rceil$ candidates in each phase w.h.p. by Lemma 6. Thus, the total number of messages across all these phases is at most $\sum_{i=\psi+1}^{K}\lceil 2^{13}\log n\rceil\cdot 2\cdot (i+5-\psi)\cdot\lceil\sqrt{4n\log n}\rceil = O(\sqrt{n}\log^{5/2} n)$ w.h.p.

By Lemma 7, exactly one candidate will complete all the first $K-1$ phases and remain in candidate state Candidate w.h.p. This candidate will generate $O(n)$ messages in phase K.

Thus, over all phases of the algorithm, there are totally $O(n)$ messages w.h.p. ◀

3 A (Tightly) Singularly Optimal Synchronous Algorithm

We also develop a message and time optimal algorithm for the synchronous setting, i.e., an algorithm that takes $O(1)$ time and $O(n)$ messages with high probability. Note that these upper bounds are tight (hence the term "tightly" in the section title).

The following theorem captures the properties of the algorithm. Due to space constraints, we refer the reader to the full version of the paper for the description of the algorithm and the proof of the theorem.

▶ **Theorem 9.** *Consider a complete anonymous network G of n nodes in the CONGEST model, with synchronous communication and adversarial wakeup. There is a randomized algorithm to solve leader election w.h.p. using $O(n)$ messages w.h.p. in 9 rounds (deterministically). (If nodes have unique identifiers, then the algorithm always elects a leader.)*

4 Conclusion

In the asynchronous setting, no singularly optimal *deterministic* leader election algorithm is known to exist. In contrast, we have shown that using randomization, a singularly optimal algorithm (whose message complexity is asymptotically optimal and whose time complexity is optimal up to a polylogarithmic factor) can be obtained.

One open question is whether we can improve the time complexity of our message-optimal ($O(n)$) randomized asynchronous algorithm from the current $O(\log^2 n)$ time, to say, $O(\log n)$ time. Also, can we obtain such a (almost) singularly optimal *deterministic* algorithm?

Another important question is to find out whether (and when) one can construct time and message-efficient asynchronous algorithms also for *general* graphs, in particular algorithms that are (essentially) singularly optimal. In general graphs, that would mean algorithms with $O(m)$ (or, at least, $\tilde{O}(m)$) messages and $\tilde{O}(D)$ (or, at least, $\tilde{O}(D)$) time. This was shown for the case of synchronous networks in [28].

References

1 Yehuda Afek and Eli Gafni. Time and message bounds for election in synchronous and asynchronous complete networks. *SICOMP*, 20(2):376–394, 1991.

2 Yehuda Afek and Yossi Matias. Elections in anonymous networks. *Information and Computation*, 113(2):312–330, 1994.

3 Dana Angluin. Local and global properties in networks of processors (extended abstract). In *STOC*, pages 82–93, 1980. doi:10.1145/800141.804655.

4 John Augustine, Gopal Pandurangan, Peter Robinson, and Eli Upfal. Towards robust and efficient distributed computation in dynamic peer-to-peer networks. In *SODA*, pages 551–569, 2012.

5 Baruch Awerbuch, Israel Cidon, and Shay Kutten. Communication-optimal maintenance of replicated information. In *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*, pages 492–502. IEEE, 1990.

6 Baruch Awerbuch, Oded Goldreich, Ronen Vainish, and David Peleg. A trade-off between information and communication in broadcast protocols. *J. ACM*, 37:238–256, 1990.

7 Mike Burrows. The chubby lock service for loosely-coupled distributed systems. In *7th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2006.

8 Miguel Castro, Peter Druschel, A-M Kermarrec, and Antony IT Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in communications*, 20(8):1489–1499, 2002.

9 Tushar D Chandra, Robert Griesemer, and Joshua Redstone. Paxos made live: an engineering perspective. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, pages 398–407, 2007.

10 Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):1–26, 2008.

11 Michael Elkin. A simple deterministic distributed mst algorithm, with near-optimal time and message complexities. In *PODC*, pages 157–163, 2017.

12 Robert G. Gallager, Pierre A. Humblet, and Philip M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Trans. Programming Languages & systems (TOPLAS)*, 5(1):66–77, January 1983. doi:10.1145/357195.357200.

13 Saurabh Ganeriwal, Ram Kumar, and Mani B. Srivastava. Timing-sync protocol for sensor networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems, SenSys 2003, Los Angeles, California, USA, November 5-7, 2003*, pages 138–149, 2003.

14 Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 29–43, 2003.

15 Robert Gmyr and Gopal Pandurangan. Time-message trade-offs in distributed algorithms. In *32nd International Symposium on Distributed Computing, DISC 2018, New Orleans, LA, USA, October 15-19, 2018*, pages 32:1–32:18, 2018.

16 Indranil Gupta, Robbert van Renesse, and Kenneth P. Birman. A probabilistically correct leader election protocol for large groups. In *Proceedings of the 14th International Conference on Distributed Computing*, DISC '00, pages 89–103, 2000. URL: http://dl.acm.org/citation.cfm?id=645957.675964.

17 Bernhard Haeupler, D Ellis Hershkowitz, and David Wajc. Round-and message-optimal distributed graph algorithms. In *PODC*, pages 119–128, 2018.

18 Pierre A. Humblet. Selecting a leader in a clique in $O(n \log n)$ messages. Memo, Lab. for Information & Decision Systems, MIT, 1984.

19 Michael Isard. Autopilot: automatic data center management. *ACM SIGOPS Operating Systems Review*, 41(2):60–67, 2007.

20 Maleq Khan, Fabian Kuhn, Dahlia Malkhi, Gopal Pandurangan, and Kunal Talwar. Efficient distributed approximation algorithms via probabilistic tree embeddings. In *Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing*, PODC '08, pages 263–272, New York, NY, USA, 2008. ACM. doi:10.1145/1400751.1400787.

21 Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. Scalable leader election. In *SODA*, pages 990–999, 2006.

22 Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. Towards secure and scalable computation in peer-to-peer networks. In *FOCS*, pages 87–98, 2006. doi:10.1109/FOCS.2006.77.

23 Ephraim Korach, Shay Kutten, and Shlomo Moran. A modular technique for the design of efficient distributed leader finding algorithms. *ACM Trans. Programming Languages & Systems (TOPLAS)*, 12(1):84–101, January 1990. doi:10.1145/77606.77610.

24 Ephraim Korach, Shlomo Moran, and Shmuel Zaks. Tight lower and upper bounds for some distributed algorithms for a complete network of processors. In *Proceedings of the third annual ACM symposium on Principles of distributed computing*, pages 199–207, New York, NY, USA, 1984. ACM. doi:10.1145/800222.806747.

25 Ephraim Korach, Shlomo Moran, and Shmuel Zaks. The optimality of distributive constructions of minimum weight and degree restricted spanning trees in a complete network of processors. *SIAM J. Computing*, 16(2):231–236, 1987. doi:10.1137/0216019.

26 Ephraim Korach, Shlomo Moran, and Shmuel Zaks. Optimal lower bounds for some distributed algorithms for a complete network of processors. *Theoretical Computer Science*, 64(1):125–132, 1989. doi:10.1016/0304-3975(89)90103-5.

27 Shay Kutten, William K. Moses Jr., Gopal Pandurangan, and David Peleg. Singularly optimal randomized leader election. *CoRR*, abs/2008.02782, 2020. arXiv:2008.02782.

28 Shay Kutten, Gopal Pandurangan, David Peleg, Peter Robinson, and Amitabh Trehan. On the complexity of universal leader election. *J. ACM*, 62:7, 2015.

29 Shay Kutten, Gopal Pandurangan, David Peleg, Peter Robinson, and Amitabh Trehan. Sublinear bounds for randomized leader election. *Theoretical Computer Science*, 561:134–143, 2015.

30 Shay Kutten and Dmitry Zinenko. Low communication self-stabilization through randomization. In *DISC*, pages 465–479. Springer, 2010.

31 Gérard Le Lann. Distributed systems - towards a formal approach. In *IFIP Congress*, pages 155–160, 1977.

32 Michael C. Loui, Teresa A. Matsushita, and Douglas B. West. Election in a complete network with a sense of direction. *IPL*, 22(4):185–187, 1986.

33 Michael C. Loui, Teresa A. Matsushita, and Douglas B. West. Election in a complete network with a sense of direction. *IPL*, 28(6):327, 1988. doi:10.1016/0020-0190(88)90181-0.

34 Nancy Lynch. *Distributed Algorithms*. Morgan Kaufman Publishers, Inc., San Francisco, USA, 1996.

35 Dahlia Malkhi, Michael Reiter, and Rebecca Wright. Probabilistic quorum systems. In *PODC 1997*, pages 267–273, New York, NY, USA, 1997. ACM. doi:10.1145/259380.259458.

36 Michael Mitzenmacher and Eli Upfal. *Probability and computing: randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press, 2017.

37 Gopal Pandurangan. Distributed network algorithms. https://sites.google.com/site/gopalpandurangan/dna. Accessed: 2020-02-14.

38 Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. A time-and message-optimal distributed algorithm for minimum spanning trees. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 743–756. ACM, 2017.

39 David Peleg. Time-optimal leader election in general networks. *J. Parallel & Distributed Computing*, 8(1):96–99, 1990. doi:10.1016/0743-7315(90)90074-Y.

40 David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM, 2000.

41 Radia Perlman. *Interconnections: bridges, routers, switches, and internetworking protocols*. Addison-Wesley Professional, 2000.

42 Gary Peterson. Efficient algorithms for elections in meshes and complete graphs. Technical report, TR 140, Dept. of CS, Univ. Rochester, 1985.

43 Murali Krishna Ramanathan, Ronaldo A. Ferreira, Suresh Jagannathan, Ananth Grama, and Wojciech Szpankowski. Randomized leader election. *Distributed Computing*, pages 403–418, 2007.

44 Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *SIGCOMM 2001*, pages 161–172, New York, NY, USA, 2001. ACM. doi:10.1145/383059.383072.

45 Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, Middleware '01, pages 329–350. Springer-Verlag, 2001. URL: http://dl.acm.org/citation.cfm?id=646591.697650.

46 Nicola Santoro. *Design and Analysis of Distributed Algorithms (Wiley Series on Parallel and Distributed Computing)*. Wiley-Interscience, 2006.

47 Gurdip Singh. Efficient leader election using sense of direction. *Distributed Computing*, 10(3):159–165, 1997. doi:10.1007/s004460050033.

48 Gurdip Singh. Leader election in complete networks. *SIAM J. Comput.*, 26(3):772–785, 1997.

49 Andrew S. Tanenbaum and Maarten Van Steen. *Distributed systems: principles and paradigms*. Prentice-Hall, 2007.

50 Gerard Tel. *Introduction to distributed algorithms*. Cambridge University Press, New York, NY, USA, 1994.

51 Yong Yao and Johannes Gehrke. The cougar approach to in-network query processing in sensor networks. *SIGMOD Record*, 31(3):9–18, 2002.

52 Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John D. Kubiatowicz. Tapestry: a resilient global-scale overlay for service deployment. *IEEE J. Selected Areas in Communications*, 22(1):41–53, January 2004. doi:10.1109/JSAC.2003.818784.