Singularly Near Optimal Leader Election in Asynchronous Networks

Shay Kutten
Faculty of Industrial Engineering and Management,
Technion – Israel Institute of Technology, Haifa, Israel

William K. Moses Jr.
Department of Computer Science, University of Houston, TX, USA

Gopal Pandurangan
Department of Computer Science, University of Houston, TX, USA

David Peleg
Department of Computer Science and Applied Mathematics,
Weizmann Institute of Science, Rehovot, Israel

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 *asynchronous* networks.

Kutten et al. (JACM 2015) presented a singularly near optimal 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's diameter, number of edges and number of nodes, respectively) with high probability.[1] Both bounds are near optimal (up to a logarithmic factor), since $\Omega(D)$ and $\Omega(m)$ are the respective lower bounds for time and messages for leader election even for synchronous networks and even for (Monte-Carlo) randomized algorithms. On the other hand, for general asynchronous networks, leader election algorithms are only known that are either time or message optimal, but not both. Kutten et al. (DISC 2020) presented a randomized asynchronous leader election algorithm that is singularly near optimal for *complete networks*, but left open the problem for general networks.

This paper shows that singularly near optimal (up to polylogarithmic factors) bounds can be achieved for general *asynchronous* networks. We present a randomized singularly near optimal leader election algorithm that runs in $O(D + \log^2 n)$ time and $O(m \log^2 n)$ messages with high probability. Our result is the first known distributed leader election algorithm for asynchronous networks that is near optimal with respect to both time and message complexity and improves over a long line of results including the classical results of Gallager et al. (ACM TOPLAS, 1983), Peleg (JPDC, 1989), and Awerbuch (STOC, 89).

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

Keywords and phrases Leader election, Singular optimality, Randomized algorithms, Asynchronous networks, Arbitrary graphs

Digital Object Identifier 10.4230/LIPIcs.DISC.2021.27

Related Version *Full Version*: `https://arxiv.org/abs/2108.02197` [30]

Funding *Shay Kutten*: This work was supported in part by the Bi-national Science Foundation (BSF) grant 2016419.
William K. Moses Jr.: Part of this work was done while the author was a postdoc at the Technion – Israel Institute of Technology in Israel. This work was supported in part by a Technion fellowship and in part by NSF grants, CCF1540512, IIS-1633720, CCF-1717075, and BSF grant 2016419.

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

Gopal Pandurangan: This work was supported in part by NSF grants CCF-1717075, CCF-1540512, IIS-1633720, and BSF grant 2016419.
David Peleg: This work was supported in part by the US-Israel Binational Science Foundation grant 2016732.

1 Introduction

Background and motivation. Trade-offs between resource bounds (typically time and space) form a major subject of study in classical theory of computation. In distributed computing, it is common to focus on two fundamental measures, the time and message complexity of a distributed network algorithm, and trade-offs between time and communication have been well studied. See, e.g., [1, 4, 6, 7] for early trade-offs. A question that arose more recently regarding various distributed problems is whether the problem admits an algorithm that is optimal in time and communication simultaneously. In [21, 40], such algorithms are called *singularly optimal* (or *singularly "near optimal"* for algorithms whose complexity is polylogarithmically worse than optimal). Such algorithms have been shown in recent years for minimum spanning tree, (approximate) shortest paths, leader election, and several other problems [14, 22, 31, 40].

All the above results were shown for *synchronous* networks, and *do not* apply to asynchronous networks. In particular, singularly near optimal randomized *synchronous* leader election algorithms were presented in [31]. These algorithms require $O(D)$ time and use $O(m \log n)$ messages with high probability (where D, n and m are the network's diameter, number of nodes and number of edges, respectively). Singular near optimality follows from the fact that $\Omega(D)$ and $\Omega(m)$ are lower bounds for time and messages for leader election even for synchronous networks and even for randomized algorithms [31]. These algorithms inherently rely on the synchronous communication, so an attempt to convert them to asynchronous networks would probably incur heavy cost overheads (see the discussion of synchronizers below). In fact, the question whether similar bounds can be achieved for general *asynchronous* networks was left open, although one can obtain algorithms that are separately time optimal [41] or message optimal [17].

A singularly near optimal randomized *asynchronous* leader election algorithm was recently presented in [29], but only for *complete* networks. It requires $O(n)$ messages and $O(\log^2 n)$ time, which is singularly optimal (up to logarithmic factors) since $\Omega(n)$ and $\Omega(1)$ are the respective message and time lower bounds for leader election in complete n-node networks. The question whether leader election in general networks admits an *asynchronous* singularly near optimal algorithm or exhibits an inherent *time-messages trade-off* was again left as an open problem. The algorithm in [29] heavily utilizes the special nature of the complete graph, so designing an asynchronous algorithm for general graphs requires additional tools and insights.

Leader election is a central and intensively studied problem in distributed computing. It captures the pivotal notion of symmetry breaking in contexts involving the entire network ("global algorithms"). Given a leader, many other problems become trivial. Consequently, a singularly optimal leader election algorithm may ease the design of a singularly optimal algorithms for many other tasks. In addition to its theoretical importance, leader election is used in multiple practical contexts. The literature is too rich to cover here, but see, for example [1, 10, 11, 17, 19, 31, 33, 35, 41, 46, 49].

In our setting, an *arbitrary* subset of nodes can *wake up spontaneously at arbitrary times* and start the election algorithm by sending messages over the network. The algorithm should terminate with a *unique* node v being elected as leader (where initially, all the nodes are in

the same state, "not leader") and the leader's identity should be *known to all nodes*. Our goal in this paper is to design leader election algorithms in distributed *asynchronous* networks that are singularly near optimal.

Using synchronizers. One can convert a synchronous algorithm to work on an asynchronous network using a standard tool known as a *synchronizer* [4]; however, such a conversion typically increases substantially either the time or the message complexity or both. Moreover, there is usually a non-negligible cost associated with *constructing* such a synchronizer in the first place. For example, applying the simple α synchronizer (which does not require the a priori existence of a leader or a spanning tree) to the singularly optimal synchronous leader election algorithm of [31] yields an asynchronous algorithm with message complexity of $O(mD \log n)$ and time complexity of $O(D)$; this algorithm is not message optimal, especially for large diameter networks. Indeed, many prior works (see e.g., [9]), do construct efficient synchronizers that can achieve optimal conversion from synchronous to asynchronous algorithms with respect to both time and messages, but constructing the synchronizer itself requires a substantial preprocessing or initialization cost. For example, the message cost of the synchronizer protocol of [9] can be as high as $O(mn)$. Moreover, several synchronizer protocols, such as β and γ of [4] and that of [9], require the existence of a *leader* or *a spanning tree*; hence these synchronizers are not useful for designing leader election algorithms. For these reasons, designing singularly optimal algorithms is more challenging for asynchronous networks than for synchronous ones and requires new approaches.

Distributed Computing Model. We model a distributed network as an arbitrary undirected connected graph $G = (V, E)$, $|V| = n$, $|E| = m$, similar to the standard model of [1, 6, 17, 27], except that, in addition, processors can access *private unbiased coins*. Nodes have only knowledge of themselves and do not have any knowledge of their neighbors and their identities (if any). This is the commonly used *clean network* or KT_0 model (see e.g., [8]).

Our algorithm does not require that nodes have unique identities; however, it requires that nodes have knowledge of n, the network size, or at least a constant factor approximation of n. We note that several prior algorithms for leader election require knowledge of n [2, 6, 47]. If nodes have unique identifiers, we assume that they are of size $O(\log n)$ bits.

We assume the standard *asynchronous* $\mathcal{CONGEST}$ communication model [42], where messages (each message is of $O(\log n)$ bits) sent over an edge incur unpredictable but finite delays, in an error-free and FIFO manner (i.e., messages will arrive in sequence). However, for the sake of time analysis, it is assumed that message takes *at most one time unit* to be delivered across an edge. As is usual, we assume that local computation within a node is instantaneous and free; however, our algorithm will involve only lightweight local computations.

We follow the standard timing and wake-up assumptions used in prior asynchronous protocols (see [1, 17, 48]). Nodes are initially asleep, and a node enters the execution when it is woken up by the environment or upon receiving messages from awakened neighbors. As usual, uncertainties in the environment can be modeled by means of an *adversary* that controls some of the execution parameters. Specifically, we assume an *adversarial wake up* model, where node wake-up times are scheduled by an adversary (who may decide to keep some nodes dormant). The time complexity is measured from the moment the first node wakes up. A node can also be woken up by receiving messages from other nodes. In addition to the wake-up schedule, the adversary also decides the time delay of 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. The above adversarial wakeup model should be contrasted with the weaker *simultaneous wake up* model, where all nodes are assumed to be awake at the beginning of computation; simultaneous wake up is typically assumed in the design of synchronous protocols (see e.g., [1, 31, 32]). In the asynchronous setting, once a node enters execution, it performs all the computations required of it by the algorithm, and sends out messages to neighbors as specified by the algorithm.

Our Contribution. The main question addressed in this paper is whether singularly optimal bounds for leader election can be achieved for general *asynchronous* networks. We answer this question in the affirmative and present a randomized singularly near optimal leader election algorithm that elects a leader with high probability, runs in $O(D + \log^2 n)$ time with high probability, and has message complexity $O(m \log^2 n)$ with high probability, where high probability is probability at least $1 - 1/n^c$, for constant c. Our algorithm even works in anonymous networks. To the best of our knowledge, this is the first known distributed leader election algorithm for asynchronous general networks that is near optimal with respect to both time and message complexity and improves over a long line of results (see Table 1) including the classical results of Gallager et al. [17], Peleg [41], and Awerbuch [6]. We refer to Table 1 for a comparison of message and time complexity bounds of leader election algorithms in asynchronous networks. It should be noted that none of the prior results achieve time and message bounds that are simultaneously close to optimal bounds (even within a $O(\text{polylog}\, n)$ factor) of $\Theta(D)$ (time) and $\Theta(m)$ (messages) respectively. We note our bounds are almost as good as those obtained for the synchronous model: the work of Kutten et al. [31] presented a $O(m \log n)$ messages (with high probability) and a $O(D)$ algorithm. It is open whether one can design a (tight) singularly optimal leader election algorithm that uses $O(m)$ messages and $O(D)$ time even for the *synchronous* setting.

The importance of having a singularly optimal (or near optimal) leader election is that it can serve as a basic building block in designing singularly (or near) optimal asynchronous algorithms for other fundamental problems such as MST and shortest paths. Currently, we are not aware of such algorithms for these problems in *asynchronous* networks (unlike synchronous networks [14, 22, 40]).

Our algorithm makes use of several elementary techniques, combined in a suitable way. In particular, it exploits the idea of using groups of referees as *quorums* in order to ensure mutual exclusion, an idea utilized in several papers, e.g. [12, 20, 29, 32, 45]. Traditionally, verifying that an entire quorum has been secured is achieved by counting the number of supporting referees. It should be noted, though, that such a counting process is problematic under the asynchronous communication model. This difficulty requires us to introduce some *slack* to the quorum sizes, and rely on it in order to ensure quorum intersection with high probability. Another interesting feature of this algorithm concerns the way it manages communication. Message transmissions are performed using *flood-based broadcasts*, even when the message is targeted at a *single* recipient (unlike most previous algorithms, where such messages are sent by unicast along a specific path). This is done since in an asynchronous network, paths defined by previous broadcasts might not be shortest, so using them later might prevent us from attaining a near diameter time. The obvious down side is that using broadcasts for transmitting individual messages is expensive in communication. Hence, one delicate technical point is how to maintain a tight cap on the overall number of wasteful broadcasts, in order to save on messages and on congestion. The key idea is to ensure that the number of active "speakers" (as opposed to passive "relays" who merely forward messages) during the entire execution is kept small (logarithmic in the network size).

Table 1 Comparison of leader election algorithms for a general graph with n nodes, m edges, and D diameter in asynchronous systems along with our contributions. Note that if the algorithm was deterministic, then it was required that nodes have unique IDs. Randomized solutions may not have such a requirement.

Paper	Message Complexity	Time Complexity	Type of Solution
Gallager et al. [17]	$O(m + n \log n)$	$O(n \log n)$	Deterministic
Lavallée and Lavault [34]*	$O(m + n \log n)$	$O(n \log \log(n/\varepsilon))$	Randomized
Chin and Ting [13]	$O(m + n \log n)$	$O(n \log^* n)$	Deterministic
Gafni [16]	$O(m + n \log n)$	$O(n \log^* n)$	Non-deterministic
Awerbuch [5], Faloutsos and Molle [15]**	$O(m + n \log n)$	$O(n)$	Deterministic
Schieber and Snir [47]	$O(m + n \log n)$	$O(n)$	Randomized
Afek and Matias [2]	$O(m)$	$O(n)$	Randomized
Peleg [41]	$O(mD)$	$O(D)$	Deterministic
Awerbuch [6]***	$O(m^{1+\varepsilon})$	$O(D^{1+\varepsilon})$	Deterministic
This paper	$O(m \log^2 n)$	$O(D + \log^2 n)$	Randomized
*They claim a *virtual* running time of $O(D' \log \log(n/\varepsilon))$, corresponding to an algorithmically constructed subgraph of diameter D' of the initial network. D' may be as large as n. $0 < \varepsilon < 1$.			
**The algorithm of [15] is a corrected version of the one in [5].			
***Here, ε can be any value > 0.			

We first compare the technical contribution of this work to the known singularly near optimal *synchronous* algorithm of Kutten et al. [31] for *general graphs*. The task there is significantly easier since nodes know when to terminate: once a node stops receiving an echo, exactly one node (the one with the highest random rank) will know it is the leader. Moreover, in the synchronous setting, this happens after $O(D)$ rounds. In the asynchronous setting, this is not possible (unless some heavy message overhead is added, e.g., by using a synchronizer). This leads to technical challenges that we overcome utilizing randomization, broadcasts, and quorums. Randomization is used not only to reduce the number of messages (similar to the synchronous case), but more importantly to implement the quorums.

We also compare the technical contribution of this work to the known singularly near optimal *asynchronous* algorithm of Kutten et al. [29] for *complete graphs*, which uses similar techniques. One key change concerns the way candidates communicate with referees. In [29], this is done by sending messages directly, which is doable in complete graphs. In contrast, in the current algorithm the candidates must rely on broadcasts to send messages to referees. This change raises additional challenges and necessitates a major modification to the algorithm of [29]. Specifically, in that algorithm, each candidate selects, and hence knows, its referees in each phase (this is doable because the graph is complete). In contrast, in our algorithm for an arbitrary graph, a candidate does not know the referees (because referees are chosen independently of the candidate). As a result, it is necessary to keep an accurate estimate of the number of referees. We rely on this estimate to ensure the existence of exactly one leader, with high probability (if the estimate is too low, multiple nodes might become leaders; if it is too high, no one will become a leader).

Moreover, it should be stressed that the technique used in [29] for saving on messages cannot be used here. Therein, candidates compete with each other in *phases*, and get eliminated gradually, until a single candidate remains. Here, there does not appear to be a way for the candidate to save by sending information to only specific referees in a message optimal manner. Specifically, applying the algorithm of [29] on a general graph by replacing direct communication with broadcasts (and making no other changes) would result in a high communication cost of $\Omega(mn)$ messages and possibly a higher run time (due to congestion), compared to the performance of the current algorithm.

Related Work. Leader election has been very well-studied in distributed networks for many decades. Le Lann [33] first studied the problem in a ring network and the seminal paper of Gallager, Humblet, and Spira [17] studied it in general graphs. Since then, various algorithms and lower bounds are known in different models with synchronous/asynchronous communication and in networks of varying topologies, such as cycles, complete graphs, or arbitrary graphs. See, e.g., [25, 27, 31, 32, 35, 41, 46, 49] and the references therein.

Prior to this paper, there were no known singularly near optimal algorithms for *general asynchronous* networks, i.e., algorithms that take $\tilde{O}(m)$ messages and $\tilde{O}(D)$ time.[2][3] Gallager, Humblet, and Spira [17] presented a minimum weight spanning tree (MST) algorithm (also applicable for leader election) with message complexity $O(m + n \log n)$ and time complexity $O(n \log n)$; this is (essentially) message optimal [31] but not time optimal. Hence, further research concentrated on improving the time complexity of MST algorithms. The time complexity was first improved to $O(n \log \log(n/\varepsilon))$, $0 < \varepsilon < 1$, by Lavallée and Lavault [34], then to $O(n \log^* n)$ independently by Chin and Ting [13] and Gafni [16], and finally to $O(n)$ by Awerbuch [5] (see also [15]). Thus using MST algorithms for leader election in asynchronous networks does not yield the best possible time complexity of $O(D)$. Peleg's leader election algorithm [41], on the other hand, takes $O(D)$ time and uses $O(mD)$ messages; this is time optimal, but not message optimal.

Korach et al. [28], Humblet [23], Peterson [44] and Afek and Gafni [1] presented $O(n \log n)$ message algorithms for *asynchronous complete* networks. Korach, Kutten, and Moran [27] presented a general method plus applications to various classes of graphs.

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. As a starting step, Schieber and Snir [47] developed a randomized algorithm for leader election in anonymous asynchronous networks that took $O(m + n \log n)$ messages and $O(n)$ time. Afek and Matias [2] presented a randomized algorithm that took the same time but improved the message complexity to $O(m)$, which is message optimal.[4]

For synchronous networks, Pandurangan et al. [40] and Elkin [14] have presented singularly near optimal distributed algorithms for MST. Note that optimal time for MST means $\Omega(D + \sqrt{n})$ [43], while for leader election in asynchronous networks, optimal time is $O(D)$, as shown in [41] but using worse message complexity.

We note that the singularly optimal algorithm of this paper (for asynchronous networks) as well as those of [40] and [14] (for synchronous networks) assume the so-called *clean network model*, a.k.a. KT_0 [42] (see Section 1), where nodes do not have initial knowledge of the identity of their neighbors. But the above optimal results do not in general apply to the KT_1 model, where nodes have initial knowledge of the identities of their neighbors. Clearly, the distinction between KT_0 and KT_1 has no bearing on the asymptotic bounds for the time complexity, but it is significant when considering message complexity. Awerbuch et al. [8] show that $\Omega(m)$ is a message lower bound for broadcast (and hence for leader

[2] $\tilde{O}$ notation hides a polylog(n) factor.

[3] There was, however, work done on a ring where Itai and Rodeh [24] presented an algorithm with $O(n \log n)$ bit complexity on expectation. Further analysis shows that the message complexity is $O(n \log n)$ on expectation and the time complexity is $O(n \log n)$ on expectation.

[4] Note that [2]'s message complexity is $O(m)$ when the solution succeeds with constant probability. For a solution that succeeds with high probability, the message complexity grows to $O(mn \log^2 n)$.

election and MST) in the KT_1 model, if one allows only (possibly randomized Monte Carlo) comparison-based algorithms, i.e., algorithms that can operate on IDs only by comparing them. (We note that all algorithms mentioned earlier are comparison-based, including ours.) Hence, the result of [8] implies that our leader election algorithm (which is comparison-based and randomized) is *time* and *message near optimal* in the KT_1 model if one considers comparison-based algorithms only.

On the other hand, for *randomized non-comparison-based* algorithms, the message lower bound of $\Omega(m)$ does not apply in the KT_1 model. King et al. [26] presented a randomized, non-comparison-based Monte Carlo algorithm in the KT_1 model for spanning tree, MST (and hence leader election) in $\tilde{O}(n)$ messages ($\Omega(n)$ is a message lower bound) and in $\tilde{O}(n)$ time (see also [36]). While this algorithm shows that one can achieve $o(m)$ message complexity (when $m = \omega(n \operatorname{polylog} n)$), it is *not* time-optimal; the time complexity for spanning tree and leader election is $\tilde{O}(n)$. The works of [18, 21] showed bounds with with improved round complexity, but with worse bounds on the message complexity and more generally, trade-offs between time and messages [21]. We note that all these results are for *synchronous* networks. For *asynchronous* networks, Mashreghi and King [37, 38] presented a spanning tree algorithm (also applies for leader election and MST) that takes $\tilde{O}(n^{1.5})$ messages and $\tilde{O}(n)$ time. It is an open question whether one can design a randomized (non-comparison based) algorithm that takes $\tilde{O}(n)$ messages and $\tilde{O}(D)$ rounds in the KT_1 model; these are the optimal bounds possible in the KT_1 model.

2 A Singularly Optimal Asynchronous Leader Election Algorithm

In this section, we present a leader election algorithm in asynchronous networks that is essentially optimal with respect to both message and time complexity. We assume that all nodes have knowledge of n, the number of nodes in the network.

2.1 Algorithm

Brief Overview. The process is initiated by one or more nodes, which are woken up by the adversary, possibly at different times (see earlier discussion of the adversarial wakeup model in Section 1). These nodes wake up the rest of the nodes. Subsequently, each awake node decides whether it will participate in the algorithm in the role of (i) a candidate and/or (ii) a referee. To do that, the node chooses randomly (for each role separately), with probability $O(\log n/n)$, whether to take on the role or not.[5] Each *candidate* attempts to become the leader by winning over a sufficient number of referees. *Referees* are used to decide which candidate will go on to become the leader, essentially preferring a stronger candidate (one who randomly chose a higher rank) over a weaker (lower rank) one, provided the stronger candidate does not "show up too late" (namely, after the weaker candidate has already accumulated sufficiently many referees to declare itself leader). Once a candidate becomes a leader, it broadcasts the message that it is leader and that all nodes should terminate.

Detailed Description. Each awake node u maintains the following information: (i) a rank $RANK_u$, chosen uniformly at random from $[1, n^4]$ (we assume that all nodes have knowledge of n, the network size), (ii) two indicator variables CAND-STATE and REF-STATE reflecting the status of its candidacy for leadership and its role as a referee, respectively, and (iii)

[5] It is possible for a node to participate in both roles or not participate in either.

the set M-List of all messages u has heard over the course of the algorithm.[6] The variable CAND-STATE can take one of three values, Candidate, Non-elected, and Elected, denoting whether the node is a candidate for leadership, it irrevocably committed itself to not be a leader, or it irrevocably committed itself to be a leader, respectively. The variable REF-STATE can take one of four values Non-selected, Ready, Chosen-Selected, and In-Dispute, with each of the states described in detail later on.

Initially, a node u is asleep and may be awoken by either the adversary or a $\langle$WAKEUP$\rangle$ message that originated at another node. (For the sake of uniformity, it is convenient to think of the the adversary waking up a node as a WAKEUP message arriving from outside the system, and treat both events in the same way.) Once awoken, u calls Procedure **Initialize**, which first wakes up adjacent nodes by sending the message $\langle$WAKEUP$\rangle$ to u's neighbors.[7] Subsequently, u chooses randomly whether or not to be a candidate by flipping a biased coin with probability $1000 \log n/n$: if successful, u initializes CAND-STATE to Candidate, chooses a rank at random - an integer in $[1, n^4]$, and broadcasts a request carrying its rank.[8] If not successful, u sets CAND-STATE to Non-elected. Node u also chooses randomly whether or not to be a referee, with probability $1000 \log n/n$: if successful, u initializes REF-STATE to Ready and sets the variables CONTENDER and CHOSEN to -1, else it sets REF-STATE to Non-selected. The pseudocode for the initialization procedure, as well as for others, is present in the full version of the paper.

Consider some node v. Any message that v wants to send, whether to continue a previous broadcast or to pass on a newly generated message, is added to its list Send-List(e) for each of its edges e. Whenever one of v's outgoing edges e is free for a new message to be sent across it, v invokes Procedure **Send_Message** for that edge, which picks an arbitrary unsent message from the list Send-List(e), transmits it over the edge, and erases it from the list. Any messages that were generated by the node are added also to M-List, a list of messages the node has already heard.

If node u becomes a candidate (respectively, a referee), then its subsequent actions in this role are governed by Procedure **Candidate** (resp., Procedure **Referee**).[9] The appropriate procedure between these two procedures is called when the node receives a message. Another node's leader announcement message is processed outside of these two procedures. Additionally, when a relevant message is received, the Procedure **Candidate_Dispute_Response** may also be invoked to resolve a dispute (to be described later). In addition to whichever procedure is called, if any, every awake node stores all the messages it receives in its list M-List and participates in every broadcast that reaches it by also adding the message to Send-List(e) for all its edges e except those over which the message was received.[10] A node u, once awake, will eventually either broadcast a LEADER message (in its role as a candidate) and terminate, or receive such a message about another candidate, at which point u stores the rank of the leader and then terminates (having also forwarded the transmitted message to its neighbors). The process governing the responses of the nodes to different incoming messages is called **On_Receive_Message**.

---

[6] Actually, our algorithm can be extended if nodes have a knowledge of n that is within some (known) constant factor. We discuss this at the end of Section 2.2.

[7] Here, and in other places, we say that a node u sends messages only to its neighbors. This is the local description of the algorithm. Globally, this results in the message being broadcast throughout the graph.

[8] The theorems in this paper hold for all sufficiently large n, say $n \geqslant n_0$, where the value of n_0 depends on the probability used for the coin flip. The particular value of $1000 \log n/n$ was chosen for ease of exposition, but it can be modified in order to yield a smaller n_0. No attempt was made to optimize this value.

[9] Recall that these two procedures may run concurrently in the same node, in case it assumed both roles.

[10] Note that if a message currently in v's Send-List(e) is received over edge e, then v removes that message from Send-List(e).

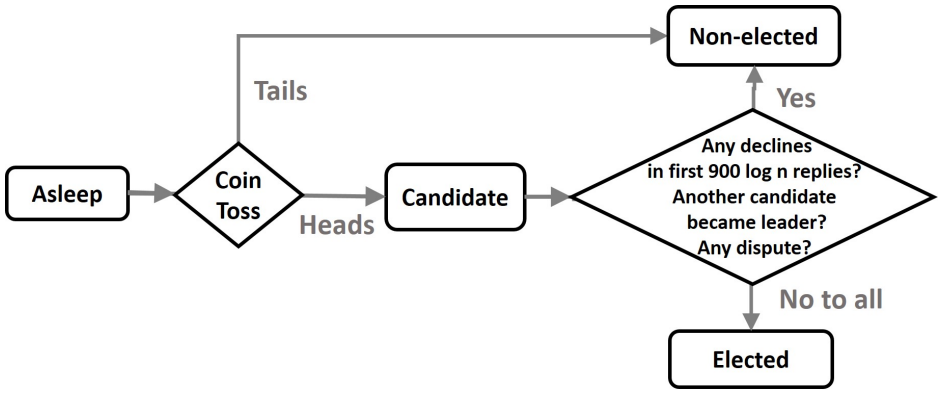

Figure 1 The state progression of a candidate.

Procedure **Candidate** is run by a candidate to help it determine whether it will become the leader or not. Each candidate u broadcasts a request message during Procedure **Initialize**. Candidate u then waits for $900 \log n$ replies from different referees.[11] If one of these replies is a decline message, i.e., a referee says that u is declined from being the leader, then u updates its CAND-STATE to Non-elected and sends the message $\langle RANK_u, \text{LOSES} \rangle$ to all neighbors. Otherwise if all the replies are APPROVED messages (and u has not terminated yet), u becomes the leader, i.e., it changes its CAND-STATE to Elected, then announces this to all the nodes and terminates.[12] The state progression of a candidate is pictorially represented in Figure 1.

Procedure **Referee** is run by a referee r to decide which candidate becomes the leader. A referee's REF-STATE may be set to one of the following three values.

- REF-STATE = Ready holds when r has not heard yet of any candidate.
- REF-STATE = Chosen-Selected holds when r has approved one candidate, referred to as its *chosen* candidate, and has declined every other candidate that approached it so far. The rank of the chosen candidate is stored in the variable CHOSEN.
- REF-STATE = In-Dispute holds when r currently keeps track of two candidates: the *chosen* v, whose rank $RANK_v$ is stored in the variable CHOSEN, and a *contender* w, whose rank $RANK_w$ is stored in the variable CONTENDER, such that $RANK_w > RANK_v$, and all other candidates that approached r so far were declined. In such a case, a *dispute* is currently in progress between v and w.

The state In-Dispute 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 $RANK_w > RANK_v$. In this situation, r cannot decline w (since it has a higher $RANK$ than its current chosen), but at the same time, it cannot approve w, since it may be that v had already collected enough approvals and became the leader in the meantime. To resolve this uncertainty, r declares a dispute, and broadcasts a message containing the dispute information, $\langle RANK_v, RANK_w, \text{DISPUTE} \rangle$, so that when v eventually receives the message, it can make the comparison between itself and w and resolve the dispute (v wins iff it already became the leader prior to receiving the dispute message). While waiting for v's response, r stores $RANK_w$ of w in the variable CONTENDER.

[11] A candidate may receive more than $900 \log n$ replies as there may be more referees. In such a case, the candidate only considers the *first* $900 \log n$ replies it receives.

[12] Note that messages about another candidate becoming the leader (and thus possibly affecting u's candidacy) are not handled in this procedure but are instead handled in Procedure **On_Receive_Message**.

As can be seen, the referee changes state only as a consequence of receiving a request from a candidate or receiving the result of a dispute from its chosen candidate.

We now observe how a referee r responds upon receiving a message $\langle RANK_u, \text{REQUEST}\rangle$ from a candidate u. We only consider what happens if r's REF-STATE $\neq$ Non-selected as otherwise r is not a referee. Three states are possible.

(A) If r's REF-STATE = Ready, meaning that u is the first contender approaching r with a request, then r registers u as its chosen candidate by storing $RANK_u$ in the variable CHOSEN, broadcasts an approval message for u, $\langle RANK_u, RANK_r, \text{APPROVED}\rangle$, and switches its REF-STATE to Chosen-Selected.

(B) If r's REF-STATE = Chosen-Selected, and the current chosen candidate is v, then there are two possibilities. The simpler situation is when v is stronger than u, i.e., $RANK_v > RANK_u$, in which case r may immediately broadcast a decline message for u, $\langle RANK_u, RANK_r, \text{DECLINED}\rangle$.

The other case is that u is the stronger of the two candidates, i.e., $RANK_v < RANK_u$. In this case, u should normally replace v as the chosen candidate, except if v has already declared itself leader in the meantime. The way to resolve this question is a dispute between u and v. First, r checks its list M-List of previously received messages to see if such a dispute between u and v is already in progress, i.e., if M-List contains a previously received message of the form $\langle RANK_v, RANK_u, \text{DISPUTE}\rangle$ announcing the initiation of a dispute between u and v, or even a message of the form $\langle RANK_v, \text{LOSES}\rangle$ announcing the outcome of such a dispute (such a message necessarily indicates that v has *lost* the dispute, since a "win" by v can only occur if v has already declared itself leader, in which case v has already broadcast this fact and hence need not reply to the dispute).[13] There are three possible situations.

- Referee r has already received a message with the outcome of the dispute (namely, v lost). Then r only updates CHOSEN to $RANK_u$ and broadcasts an approval message for u, $\langle RANK_u, RANK_r, \text{APPROVED}\rangle$.
- Referee r has received a message announcing a dispute, but has not yet heard about the outcome. Then r only updates CONTENDER to $RANK_u$ and updates REF-STATE to In-Dispute and awaits news on the outcome (but does not broadcast any new messages).
- Referee r did not hear of an existing dispute between u and v. Then it is up to r to initiate a dispute, so r registers u as the contender by storing $RANK_u$ in the variable CONTENDER, sets its REF-STATE to In-Dispute, and broadcasts a dispute message for v of the form $\langle RANK_v, RANK_u, \text{DISPUTE}\rangle$.

(C) If r's REF-STATE = In-Dispute, signifying that *another* dispute (between v and some other candidate) is in progress, then r compares the new candidate u with the current contender w. If u is weaker than w ($RANK_u < RANK_w$), then r immediately broadcasts a decline message for u, $\langle RANK_u, RANK_r, \text{DECLINED}\rangle$. Otherwise ($RANK_u > RANK_w$), r broadcasts a decline message for w, $\langle RANK_w, RANK_r, \text{DECLINED}\rangle$, updates CONTENDER to $RANK_u$, and initiates a new dispute by broadcasting a dispute message for CHOSEN of the form $\langle \text{CHOSEN}, RANK_u, \text{DISPUTE}\rangle$.

The procedure to handle the referee's actions on receiving a request is called Procedure **Referee_Request_Response**.

[13] Note that a $\langle RANK_v, \text{LOSES}\rangle$ message may be the result of a dispute between a candidate v and some other candidate, not necessarily u. It is sufficient that v lost its candidacy.

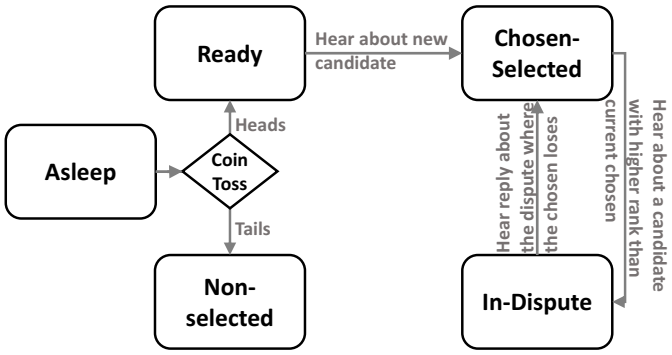

Figure 2 The state progression of a referee.

Finally, let us describe how a node v which is currently the chosen candidate of some referee r (but may have possibly changed its CAND-STATE since the time it was approved by r) handles a dispute request. Notice that a $\langle RANK_v, RANK_u, \text{DISPUTE}\rangle$ message is only sent to v when it is weaker than u ($RANK_v < RANK_u$), so when v receives such a message, it must abdicate its candidacy (if it is still a candidate), unless it has already elected itself as leader (which is an irreversible decision) and terminated. Hence, if v's CAND-STATE = Candidate, then v relinquishes its candidacy by setting CAND-STATE to Non-elected and broadcasts the result of the dispute as $\langle RANK_v, \text{LOSES}\rangle$.[14] If, however, v's CAND-STATE is set to Elected, then v has already broadcast a leader announcement message and terminated, so no additional response to the dispute message is required. The procedure to handle the actions of the chosen candidate on receiving a dispute message from a referee is called Procedure **Candidate_Dispute_Response**.

As there are multiple referees generating messages for various disputes, a referee may receive the results of a dispute it does not need to immediately process (but the result is stored for future processing, if any). If r's REF-STATE = In-Dispute, r's chosen is v, r's contender is u, and r receives a reply to a dispute of the form $\langle RANK_v, \text{LOSES}\rangle$, then r immediately processes the message as follows.[15] First, r updates CHOSEN to $RANK_u$, sets CONTENDER to -1, and updates REF-STATE to Chosen-Selected. Subsequently, r initiates the broadcast of an approval message for u, $\langle RANK_u, RANK_r, \text{APPROVED}\rangle$. The procedure to handle the referee's actions on receiving a reply from the chosen about an ongoing dispute is called Procedure **Referee_Dispute_Reply_Response**. The state progression of a referee is seen in Figure 2.

2.2 Analysis

We now prove the correctness of the algorithm and analyze its complexity. We prove a weaker time complexity bound of $O(D \log^2 n)$ here, with the stronger result of $O(D + \log^2 n)$ in Section 2.3.2. Before getting into the meat of the proof, let us make an important observation and subsequently state a useful lemma.

[14] Note that this broadcast operation is unnecessary when v's CAND-STATE = Non-elected, as v would have previously broadcast a message announcing its loss. This broadcast would have been the result of either another dispute involving v or v receiving a decline from one of the referees.

[15] Note that the message $\langle RANK_v, \text{LOSES}\rangle$ may also be generated by a candidate v upon receiving a decline message from a referee. For ease of writing, when we say "reply to a dispute", we mean any message of the form $\langle RANK_v, \text{LOSES}\rangle$, regardless of how it was generated.

▶ **Observation 1.** *From the time the first node is awake, all nodes are awakened in $O(D)$ time using $O(m)$ messages.*

Set $\mathcal{R}_\ell = 900 \log n$ and $\mathcal{R}_h = 1100 \log n$. By Observation 1, we see that all nodes awaken and thus participate in candidate selection and referee selection. Denote by N_C and N_R the number of candidates and referees selected in the algorithm, respectively. We now bound N_C and N_R with high probability.

▶ **Lemma 2.** *With probability $1 - 1/n^3$, both the number of candidates N_C and the number of referees N_R are in $[\mathcal{R}_\ell, \mathcal{R}_h]$.*

Proof. The random choice of a single candidate can be viewed as a Bernoulli trial with probability $1000 \log n/n$. Thus, the total number of candidates chosen N_C is the sum of n independent Bernoulli trials. Using known Chernoff bounds such as Theorem 4.4 and 4.5 in [39] where $\delta = 1/10$ and $\mu = 1000 \log n$, we see that the bounds in the lemma hold with the required probability. A similar argument holds for N_R. ◀

We are now ready to argue the correctness of the algorithm, i.e., show that exactly one candidate becomes a leader with high probability. This is done in two stages, showing first that the number of candidates that become leaders is *at least* one and then that this number is *at most* one. Throughout the following lemmas, we require that each node has a unique $RANK$. It is easy to see that this is true with high probability since each node selects its rank uniformly at random from $[1, n^4]$.

▶ **Lemma 3.** *At least one candidate becomes a leader with high probability.*

Proof. By Lemma 2, at least one node becomes a candidate with high probability and chooses a RANK. Each candidate waits for replies from $\mathcal{R}_\ell$ referees before deciding to become a leader. By Lemma 2, at least that many nodes become referees with high probability. Thus, there exist enough referees with high probability that generate replies for a candidate so that it can become a leader.

Let u be the candidate with the highest RANK. Now u broadcasts its candidacy to all referees. Either u wins at every referee, receives the responses, and becomes a leader. Or else, u loses at some referee, which implies that some other candidate is a leader. Thus at least one candidate becomes a leader. ◀

▶ **Lemma 4.** *At most one candidate becomes a leader with high probability.*

Proof. Each candidate waits for positive replies from $\mathcal{R}_\ell$ referees in order to become a leader. By Lemma 2, with high probability, the number of referees that exist in the system satisfies

$$N_R \geqslant \mathcal{R}_\ell > 0.8 N_R.$$

Put another way, given that $N_R \in [\mathcal{R}_\ell, \mathcal{R}_h]$, for any two candidates u and v that receive replies from the sets of referees R_u and R_v in order to decide on becoming a leader,

$$\begin{aligned} |R_u \cap R_v| &= |R_u \cup R_v| - |R_u \setminus R_v| - |R_v \setminus R_u| \;\geqslant\; \mathcal{R}_\ell - (\mathcal{R}_h - \mathcal{R}_\ell) - (\mathcal{R}_h - \mathcal{R}_\ell) \\ &= 900 \log n - (1100 \log n - 900 \log n) - (1100 \log n - 900 \log n) \;>\; 1. \end{aligned}$$

We show that when two candidates u and v share a referee r whose replies help them determine if they may become a leader, it is impossible for both candidates to become leaders. Thus, no more than one candidate becomes a leader with high probability. Without loss of generality, let $RANK_u < RANK_v$. Consider the sequence of arrival of candidacy messages at r and replies. The following three cases cover all possible scenarios.

Case 1: r knows of a candidate w where either $RANK_w > RANK_u$ or w has become a leader before a dispute message generated by r reaches it.

If r knows of a candidate w where $RANK_w > RANK_u$, then it is clear that at least u will be rejected and not become the leader. Otherwise, if r receives a candidacy message from either u or v, r will generate a dispute message and send it to w. If w has become the leader before this dispute message reaches it, then w would have already generated a leader announcement message and terminated. Node r meanwhile will not confirm u or v as a leader until it hears back from w. So whichever of u's or v's candidacy message was at r will not be approved once w's leader announcement message reaches node r and both candidates will not become the leader.

In this situation, it is guaranteed that at least one of u or v not become the leader.

Case 2: v's candidacy message reaches r first and r subsequently replies that v may become the leader, all before u's candidacy message reaches r.

In this case, since $RANK_u < RANK_v$, r declines u once its candidacy message reaches r. It is also possible that v may have become the leader and generated a leader announcement message. In this situation, that message may propagate to r and u, also resulting in u not becoming the leader. In either situation, u will not become the leader.

Case 3: u's candidacy message reaches r first and r subsequently replies that u may become the leader, all before v's candidacy message reaches r.

In this case, if u received enough approvals and became the leader, then it may generate a leader announcement message. Now, this leader announcement message will either reach r before or after v's candidacy message reaches it. In either case, v will not become the leader because r will not approve v before receiving the result of the dispute from u. If however, u did not receive enough approvals before v's candidacy message reaches r and r's subsequently generated dispute reaches u, then u will give up its candidacy.

In either case, at most one of u or v will become the leader (perhaps neither of them). ◀

Thus, the algorithm is correct. We now give a useful lemma that is subsequently used to bound both the message and time complexity of the algorithm.

▶ **Lemma 5.** *Any node generates at most $O(\log n)$ unique messages with high probability to be broadcast over the course of the algorithm.*

Proof. There is one instance of a WAKEUP message sent through the system. In addition, each candidate generates one candidacy request message and possibly one leader announcement message or candidacy loss message (as a reply to a dispute). Thus each candidate generates $O(1)$ messages. Each referee generates a reply to each candidate it hears from and possibly a dispute message with the current chosen as well. By Lemma 2, there are $O(\log n)$ candidates a referee may have to reply to and generate disputes for, resulting in each referee generating $O(\log n)$ messages. ◀

▶ **Lemma 6.** *There are $O(\log^2 n)$ unique messages with high probability broadcast in the system over the course of the algorithm.*

Proof. Aside from the initial WAKEUP message, only candidates and referees generate unique messages to be broadcast. There are $O(\log n)$ such candidates and referees with high probability by Lemma 2. Thus, there are totally $O(\log^2 n)$ unique messages with high probability broadcast in the system over the course of the algorithm. ◀

Combining the lemma with the fact that each broadcast of a unique message results in $O(m)$ messages, we get the following.

▶ **Corollary 7.** *The total message complexity is $O(m \log^2 n)$ with high probability.*

▶ **Lemma 8.** *The run time of the algorithm is $O(D \log^2 n)$ with high probability.*

Proof. By Observation 1, all nodes wake up in $O(D)$ time. One of the woken nodes, say u, will go on to become the leader. Let us bound the number of "logical phases" of broadcasts needed until all the nodes are made aware that u is the leader. (Note that we only use the word "phase" in this analysis, but we do not use this terminology in the algorithm itself.) Each phase is responsible for certain information being broadcast to nodes, and the phases, as described, occur sequentially. Furthermore, different phases may take different amounts of time. In the first phase, u broadcasts its candidacy. In the next phase, each of the referees may need to broadcast a dispute. In the subsequent phase, each of the candidates that is a target of a dispute needs to broadcast its reply. In the next phase, each of these referees broadcasts its reply to u's candidacy request. In the final phase, u broadcasts that it is the leader. Thus, there are $O(1)$ such phases.

In each of these phases, a broadcast originating at some node u is complete when the message m reaches all other nodes. The shortest path between u and any other node is of length at most $O(D)$. By Lemma 6, there are at most $O(\log^2 n)$ unique messages generated in the system with high probability. Thus m may be delayed at each node in the shortest path by at most $O(\log^2 n)$ other messages with high probability, resulting in a total time of $O(D \log^2 n)$ with high probability for the phase to complete. Since there are $O(1)$ phases, the total time until the algorithm completes is $O(D \log^2 n)$ with high probability. ◀

▶ **Theorem 9.** *There exists an algorithm that solves leader election with high probability in any arbitrary graph with n nodes, m edges, and diameter D in $O(D \log^2 n)$ time with high probability using $O(m \log^2 n)$ messages with high probability in an asynchronous system with adversarial node wakeup.*

2.3 Improvements

2.3.1 Knowledge of n

In the above analysis, it may be noted that nodes do not need to know the exact value of n. In fact, it is easy to extend the algorithm and analysis if nodes know the value of n up to a constant factor. More precisely, it is sufficient if all nodes know either (i) the value of a constant c_1, where $0 < c_1 \leqslant 1$, and a lower bound on n, n', such that $c_1 n \leqslant n' \leqslant n$ or (ii) the value of a constant c_2, where $1 \leqslant c_2$, and an upper bound on n, n^*, such that $n \leqslant n^* \leqslant c_2 n$. By adjusting the coin toss probability to some $c \log n'/n'$ (or $c \log n^*/n^*$) for a carefully chosen value of c, we can show that the analysis goes through for a sufficiently large n.

2.3.2 Reducing Time Complexity to $O(D + \log^2 n)$

The analysis of the runtime in Section 2.2 can be tightened further. The below lemma comes from Theorem 1 in Topkis [50], adapted to the current setting and terminology. Notice that we can use their Theorem because the process of flooding they study is being implemented here via **Send_Message**(e).

▶ **Lemma 10.** *It takes $D + k - 1$ time to broadcast k messages in a graph with diameter D in the asynchronous setting.*

Combining Lemma 10 with Lemma 6, which states that there are at most $O(\log^2 n)$ unique messages with high probability, and the argument (from the proof in Section 2.2) that there are $O(1)$ "logical phases" of broadcasts, we see that the run time of the algorithm is $O(D + \log^2 n)$ with high probability. When coupled with our previous analysis of correctness and message complexity, we arrive at the following theorem.

▶ **Theorem 11.** *There exists an algorithm that solves leader election with high probability in any arbitrary graph with n nodes, m edges, and diameter D in $O(D + \log^2 n)$ time with high probability using $O(m \log^2 n)$ messages with high probability in an asynchronous system with adversarial node wakeup.*

3 Conclusion

We have presented a randomized algorithm for asynchronous leader election in general networks. Our algorithm has message and time bounds that are both within a polylogarithmic factor of the lower bound, and is the first such singularly optimal algorithm presented for general asynchronous networks.

Two important open questions remain. First, is it possible to obtain near singularly optimal bounds using a *deterministic* algorithm? Our algorithm needs an accurate knowledge of the network size (at least up to a constant factor). Second, can we get a (near) singularly optimal algorithm (even randomized) without the restriction that nodes need an accurate knowledge of n or even no knowledge of n? Third, can we design (near) singularly optimal algorithms for other fundamental problems such as minimum spanning tree and shortest paths in the asynchronous model.

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 Baruch Awerbuch. Complexity of network synchronization. *Journal of the ACM (JACM)*, 32(4):804–823, 1985.

5 Baruch Awerbuch. Optimal distributed algorithms for minimum weight spanning tree, counting, leader election, and related problems. In *Proceedings of the 19th ACM Symposium on Theory of Computing (STOC)*, pages 230–240, 1987.

6 Baruch Awerbuch. Distributed shortest paths algorithms (extended abstract). In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 490–500, 1989.

7 Baruch Awerbuch and Robert G Gallager. Distributed bfs algorithms. In *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*, pages 250–256. IEEE, 1985.

8 Baruch Awerbuch, Oded Goldreich, David Peleg, and Ronen Vainish. A trade-off between information and communication in broadcast protocols. *J. ACM*, 37(2):238–256, 1990.

9 Baruch Awerbuch and David Peleg. Network synchronization with polylogarithmic overhead. In *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*, pages 514–522. IEEE, 1990.

10 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.

11 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.

12 Soumyottam Chatterjee, Gopal Pandurangan, and Peter Robinson. The complexity of leader election in diameter-two networks. *Distributed Computing*, pages 1–17, 2019.

13 F. Chin and H. F. Ting. Improving the time complexity of message-optimal distributed algorithms for minimum-weight spanning trees. *SIAM Journal on Computing*, 19(4):612–626, 1990.

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

15 Michalis Faloutsos and Mart Molle. A linear-time optimal-message distributed algorithm for minimum spanning trees. *Distributed Computing*, 17(2):151–170, 2004.

16 Eli Gafni. Improvements in the time complexity of two message-optimal election algorithms. In *Proceedings of the 4th Symposium on Principles of Distributed Computing (PODC)*, pages 175–185, 1985.

17 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.

18 Mohsen Ghaffari and Fabian Kuhn. Distributed MST and broadcast with fewer messages, and faster gossiping. In *Proceedings of the 32nd International Symposium on Distributed Computing (DISC)*, pages 30:1–30:12, 2018.

19 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.

20 Seth Gilbert, Peter Robinson, and Suman Sourav. Leader election in well-connected graphs. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, pages 227–236, 2018.

21 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.

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

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

24 Alon Itai and Michael Rodeh. Symmetry breaking in distributed networks. *Inf. Comput.*, 88(1):60–87, 1990.

25 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.

26 Valerie King, Shay Kutten, and Mikkel Thorup. Construction and impromptu repair of an MST in a distributed network with $o(m)$ communication. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 71–80, 2015.

27 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, 1990. doi:10.1145/77606.77610.

28 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.

29 Shay Kutten, William K. Moses Jr., Gopal Pandurangan, and David Peleg. Singularly optimal randomized leader election. In Hagit Attiya, editor, *34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference*, volume 179 of *LIPIcs*, pages 22:1–22:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.DISC.2020.22.

30 Shay Kutten, William K. Moses Jr., Gopal Pandurangan, and David Peleg. Singularly near optimal leader election in asynchronous networks. *CoRR*, abs/2108.02197, 2021. arXiv:2108.02197.

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

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

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

34 Ivan Lavallée and Christian Lavault. Spanning tree construction for nameless networks. In *International Workshop on Distributed Algorithms*, pages 41–56. Springer, 1990.

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

36 Ali Mashreghi and Valerie King. Time-communication trade-offs for minimum spanning tree construction. In *Proceedings of the 18th International Conference on Distributed Computing and Networking (ICDCN)*, 2017.

37 Ali Mashreghi and Valerie King. Brief announcement: Faster asynchronous MST and low diameter tree construction with sublinear communication. In Jukka Suomela, editor, *33rd International Symposium on Distributed Computing, DISC 2019, October 14-18, 2019, Budapest, Hungary*, volume 146 of *LIPIcs*, pages 49:1–49:3, 2019.

38 Ali Mashreghi and Valerie King. Broadcast and minimum spanning tree with o(m) messages in the asynchronous CONGEST model. *Distributed Computing*, pages 1–17, 2021.

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

40 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, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 743–756, 2017.

41 David Peleg. Time-optimal leader election in general networks. *J. Parallel & Distributed Computing*, 8(1):96–99, 1990.

42 David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. Society for Industrial and Applied Mathematics, 2000.

43 David Peleg and Vitaly Rubinovich. A near-tight lower bound on the time complexity of distributed mst construction. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pages 253–261. IEEE, 1999.

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

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

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

47 Baruch Schieber and Marc Snir. Calling names on nameless networks. *Information and Computation*, 113(1):80–101, 1994.

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

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

50 Donald M. Topkis. Performance analysis of information dissemination by flooding. *IEEE journal on selected areas in communications*, 7(3):335–340, 1989.