



Smoothed Analysis of Leader Election in Distributed Networks

Anisur Rahaman Molla ¹  and Disha Shur ² 

¹ Computer and Communication Sciences Division, Indian Statistical Institute, Kolkata, India. molla@isical.ac.in

² Electrical and Computer Engineering, Purdue University, West Lafayette, USA. dshur@purdue.edu

† This paper is an extended version of our paper published in the Proceedings of 22nd International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2020) [1].

Abstract: We study smoothed analysis of the leader election (LE) problem in distributed networks. Smoothed analysis is a hybrid between worst-case analysis and average-case analysis. It takes a worst-case instance of the algorithm and perturbs the input by adding some random noise and analyses the algorithm on this perturbed input. We consider smoothed analysis, in which the topology of the input graph, G , is randomly perturbed by adding random edges to G . The complexity of the algorithm is parameterized by a smoothing parameter $0 \leq \epsilon(n) \leq 1$ which controls the amount of random edges to be added to the input graph G per round, where ϵ is a small function of n , e.g., n^{-4} , where n is the number of nodes in the graph G . Informally, ϵ is the probability that a random edge can be added to a node per round. We analyze the time and message complexity of leader election in the above smoothing model. We present the following three results in synchronous *CONGEST* distributed model: (1) A simple randomized algorithm that elects a leader with high probability (with high probability or w.h.p. in short means with probability $\geq 1 - 1/n$) in $O((\log n)/\epsilon)$ rounds and uses $O(\sqrt{n} \log^{2.5} n)$ messages. Note that both the time and the message bounds are optimal (up to a polylog n factor); (2) A time-improved randomized algorithm that elects a leader with high probability in $O\left(\frac{\log n}{\sqrt{\epsilon}}\right)$ rounds, but uses $O(m + n \log n)$ messages, where m is the number of edges in the input graph G ; (3) A deterministic algorithm (except the randomized smoothing part) which solves leader election in $O\left(\frac{\log^2 n}{\sqrt{\epsilon}}\right)$ rounds and incurs $O(m + n\sqrt{\epsilon} \log^2 n)$ messages. Our work extends the study of smoothed analysis of distributed problems one step further, an open direction raised by [2].

Citation: Molla, A.R.; Shur, D. Smoothed Analysis of Leader Election in Distributed Networks. *Algorithms* **2021**, *1*, 0. <https://doi.org/>

Keywords: distributed algorithms; smoothed analysis; random model; leader election

Received:

Accepted:

Published:

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2021 by the authors. Submitted to *Algorithms* for possible open access publication under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Motivated by the work of Dinitz et al. [3], the smoothed analysis of distributed algorithms is first formally modeled by Chatterjee et al. [2] and studied for the minimum spanning tree (MST) problem. In this paper, we extend the study of smoothed analysis of distributed problems, an open direction raised by [2], by considering the smoothed analysis of leader election problem. Leader election is one of the fundamental and well studied problem in the field of distributed computing. It outlines the problem of electing a particular node in a network as the leader. A version of this problem requires only the leader node to be aware of its status. All the nodes other than the leader are simply aware that they are not the leader. They need not be aware the leader's identity. This version is called the *implicit* leader election. The *explicit* version of the leader election problem requires all the nodes in the network to be aware of the identity of the leader. The widespread application of the leader election can be found in many domains, e.g., sensor networks [4], IoT networks [5], grid computing [6], peer-to-peer networks [7,8] and cloud computing [9]. In IoT networks, a leader node performs crucial tasks such as gathering

37 information, coordinate tasks among the nodes, generating encryption-decryption keys
38 etc. [5,10].

39 We consider the same smoothing model as defined in the paper [2] to analyze
40 distributed algorithms. In particular, we consider smoothed analysis, in which the
41 topology of the input graph G is randomly perturbed by adding random edges to G . The
42 perturbation is determined by a smoothing parameter $0 \leq \epsilon(n) \leq 1$ which controls the
43 amount of random edges to be added to the input graph G per round. Typically ϵ is a
44 small function of n , e.g., n^{-4} , where n is the number of nodes in the graph G . Smoothed
45 edges are used for communication only. The study of the smoothing analysis investigates
46 how these additional smoothed edges can be exploited to improve the time and message
47 complexity of a distributed algorithm.

48 Kutten et al. [11] studied the (implicit) leader election problem in both complete
49 networks and general networks. They presented an algorithm that takes in $O(1)$ time
50 and uses only $O(\sqrt{n} \log^{3/2} n)$ messages to elect a leader in the complete graph. For the
51 general graphs, they extend this algorithm which takes $O(\tau)$ time and $O(\tau \sqrt{n} \log^{3/2} n)$
52 messages, where τ is the mixing time of graph. The mixing time of a graph could be as
53 large as $O(n^3)$ [12]. This algorithm requires to know the mixing time to be known as
54 input. A major improvement was introduced in [13] where the same problem has been
55 solved without any knowledge of the mixing time of the graph. All these works build on
56 a technique of sampling smaller set of nodes (via random walks) and compute leader in
57 the sampled set. This is a standard technique that is useful for reducing the message
58 complexity. We also use the same idea in our randomized algorithms. The algorithms of
59 [11,13] analyze the worst case time and message complexities. A smoothed analysis of
60 the same problem is considered in this paper where the objective is to analyze both the
61 time and message complexities of the algorithm. Smoothed analysis can be viewed as a
62 hybrid between worst-case analysis and the average-case analysis. It takes a worst-case
63 instance for the algorithm and perturbs the input by adding some random noise and
64 analyzes the algorithm on this perturbed input.

65 The paper [2] analyzed only the time complexity of the MST problem. In this
66 paper, we analyze both the time and the message complexity of the leader election
67 problem. Similar to [2], we assume that the nodes of the input graph able to distinguish
68 between the smoothed edges and the original graph edges. We present a simple algorithm
69 that solves the problem with high probability in $O(\frac{\log n}{\epsilon})$ rounds using $O(\sqrt{n} \log^{5.2} n)$
70 messages. Then we present an improvement algorithm that takes $O(\frac{\log n}{\sqrt{\epsilon}})$ rounds and uses
71 $O(m + n \log n)$ messages. We further present a deterministic algorithm that solves leader
72 election in $O(\frac{\log^2 n}{\sqrt{\epsilon}})$ rounds and incurs $O(m + n \sqrt{\epsilon} \log^2 n)$ messages. The algorithm is
73 *deterministic* in the sense that except the randomized smoothing part, all other parts
74 are deterministic. To summarize, we present the following three results in synchronous
75 *CONGEST* distributed model:

- 76 1. A simple randomized algorithm that elects a leader with high probability¹ in
77 $O((\log n)/\epsilon)$ rounds and uses $O(\sqrt{n} \log^{2.5} n)$ messages. Note that both the time
78 and the message bounds are optimal (up to a **polylog** n factor).
- 79 2. A time-improved randomized algorithm that elects a leader with high probability
80 in $O(\frac{\log n}{\sqrt{\epsilon}})$ rounds, but uses $O(m + n \log n)$ messages, where m is the number of
81 edges in the input graph G .
- 82 3. A deterministic algorithm (except the randomized smoothing part) which solves
83 leader election in $O(\frac{\log^2 n}{\sqrt{\epsilon}})$ rounds and incurs $O(m + n \sqrt{\epsilon} \log^2 n)$ messages.

84 Note that one can directly solve leader election in general graphs in $O(D \log n)$
85 rounds and $O(m \log n)$ messages deterministically using algorithm in [14] or using a

¹ With high probability (or w.h.p. in short) means with probability $\geq 1 - 1/n$.

86 randomized algorithm in time $\tilde{O}(\tau)$ and $\tilde{O}(\tau\sqrt{n})$ messages [11,13], but the diameter D
 87 or mixing time τ of a graph could be large (\tilde{O} hides a polylog n factor).

88 1.1. Model

89 *Distributed Network Model*

90 We model the communication network as an undirected, unweighted, connected
 91 graph $G = (V, E)$, where $|V| = n$, and $|E| = m$. Every node has limited initial knowledge.
 92 We assume anonymous network, i.e., nodes do not know their neighbors. We assume
 93 that nodes are associated with a distinct identity number (e.g., its IP address). If
 94 not, then each node can randomly pick a number in the range $[1, n^4]$ such that the
 95 numbers are distinct for all the nodes. The random number can be used as the ID of the
 96 nodes. The node may also accept some additional inputs as specified by the problem at
 97 hand. The nodes are allowed to communicate through the edges of the graph G . The
 98 communication occurs in synchronous rounds. In one round, nodes can send messages,
 99 receive messages (from neighbors) and perform some local computation. Our algorithms
 100 use only small-sized messages. In particular, in each round, each node sends a message of
 101 size $O(\log n)$ through its adjacent edges. This is a widely used standard model known
 102 as the *CONGEST* model of distributed computing [15], and captures the bandwidth
 103 constraints inherent in real-world computer networks.

104 *Smoothing Model*

105 There are many smoothing models exists in sequential settings, see [2,3,16]. We
 106 consider the smoothing model defined by Chatterjee et al. [2]. The smoothing model of
 107 [2] is appropriate to analyze distributed network algorithms.

108 Given an arbitrary graph G , smoothing allows to introduce some “perturbance” to
 109 the input graph. In particular, the smoothing model allows adding some random edges,
 110 determined by a smoothing parameter ϵ , to the input graph G . The smoothing parameter
 111 $0 \leq \epsilon = \epsilon(n) \leq 1$ is a function of n , which controls the amount of random edges to be
 112 added in the graph per round (typically ϵ is a small function of n , e.g., n^{-4}). Thereby the
 113 graph structure is altered. This model is called the ϵ -smoothing model [2]. More precisely,
 114 in every (smoothing) round, every node add an edge with probability ϵ to a randomly
 115 chosen node from V in the given graph G . The added edges are called smoothed edges.
 116 In case of multiple edges being present between two nodes, unless specified, only one
 117 smoothed edge is used for communication. The added edges persists in the network and
 118 can be used for communication in the later rounds. Let the induced graph formed by the
 119 random edges be $R(G) = R(V, S)$, where S is the set of only the smoothed edges. $R(G)$
 120 is called smoothed graph.

121 As noted in [2], one can view the smoothing model as a generalization of the congested
 122 clique model. Suppose the graph G is embedded in a congested clique. A node, besides
 123 using its incident edges in E , can also choose to use a random edge in the clique (not in
 124 G) with probability ϵ in a round to communicate (once chosen, a random edge can be
 125 used subsequently till the end of computation). Thus, the smoothed edges are the clique
 126 edges.

127 Smoothed edges are used for faster communication. The study of the smoothing
 128 analysis investigates how these additional smoothed edges can be exploited to improve
 129 the time and message complexity of a distributed algorithm. In this paper, we consider
 130 only addition of edges to the input graph; while a smoothing model also allows deletion
 131 of edges from the graph.

132 A formal definition of the leader election problem.

133 **Definition 1.** Every node u maintains a variable $status_u$ that it can set to a value in
 134 $\{\perp, NON-ELECTED, ELECTED\}$; initially $status_u = \perp$ for all u . An algorithm A solves
 135 leader election in T rounds if, from round T on, exactly one node has its status set to

136 *ELECTED* while all other nodes are in state *NON-ELECTED*. This is the requirement
 137 for standard (implicit) leader election.

138 Note that the implicit leader election algorithm can be converted to an *explicit* leader
 139 election (where every node knows the ID of the leader) by simply the leader sends its ID
 140 to all the nodes.

141 *Paper Organization*

142 Section 2 discusses related works. Section 3 contains smoothed analysis of leader
 143 election algorithm. We first present a simple optimal time and message complexity leader
 144 election algorithm in Section 3.1. Then an improved time algorithm in Section 3.2 and a
 145 deterministic smoothed analysis in Section 3.3. Finally, we conclude in Section 4.

146 **2. Related Work**

147 Leader election is one of the fundamental problem in distributed networks. It has
 148 been extensively studied in various models and settings, starting from its introduction by
 149 Le Lann [17] in ring network and then by a seminal work of Gallager-Humblet-Spira [18]
 150 in general graphs. The problem is well studied in complete network itself [19–24], and
 151 reference there in, and also in general networks [13,14,25].

152 Some closely related leader election works in the classical congest model are [11,13,
 153 14,26]. Kutten et al. [11] studies the (implicit) leader election problem in both complete
 154 networks and general networks and showed efficient time and message bounds of leader
 155 election algorithms. In particular, they presented an algorithm that executes in $O(1)$ time
 156 and uses only $O(\sqrt{n} \log^{3/2} n)$ messages to elect a leader in a complete graph. They also
 157 showed an almost matching lower bound for randomized leader election. The standard
 158 technique of sampling smaller set of nodes and compute leader among themselves is
 159 further used in general graphs and achieve $\tilde{O}(\tau)$ -time and $\tilde{O}(\tau\sqrt{n})$ -message complexity
 160 leader election algorithm [11,13], where τ is the mixing time of a graph (\tilde{O} hides a
 161 polylog n factor). We also use this sampling techniques in the randomized algorithms.
 162 Several tight results are shown in general graphs in [14], including a notable deterministic
 163 algorithm which solves leader election in $O(D \log n)$ rounds and $O(m \log n)$ messages.
 164 We adapt this deterministic algorithm in our deterministic smoothing analysis of leader
 165 election. The our paper is inspired mainly by the works of [11,13,14].

166 Smoothed algorithms have been explored in [27] as the next step in bridging the gap
 167 between the “theoretical predictions and empirical observation” in their performances.
 168 Since its introduction in [16], analyses have shown the practical running time of algorithms
 169 to be closer to their smoothed complexities than the worst-case ones. Smoothed analysis
 170 of some popular graph problems have been explored in [28], [29], [30] and [31]. The
 171 first smoothed analysis of distributed algorithms, to the best of our knowledge, has
 172 been conducted in [3] where in they study the robustness of their algorithm for dynamic
 173 networks. They analyse three problems, namely random walks, flooding and aggregation
 174 and their upper and lower bounds on dynamic networks. Their smoothing procedure
 175 consists of addition as well as deletion of edges from the evolving graph. Robustness
 176 is measured by monitoring the change in bounds with the magnitude of smoothing
 177 introduced in the model.

178 **3. Leader Election in the Smoothing Model**

179 Given an arbitrary graph $G = (V, E)$ and CONGEST model of communication,
 180 the goal is to compute a leader in the ϵ -smoothing model of G . We first present a
 181 simple algorithm that solves the leader election problem in $O(\log n/\epsilon)$ rounds using
 182 $O(\sqrt{n} \log^{2.5} n)$ messages in the ϵ -smoothing model. The message complexity is optimal
 183 (up to a polylog n factor) and the time complexity is also optimal (up to a polylog n
 184 factor) for any $\epsilon \geq 1/\text{polylog } n$. Then we present a time improved algorithm which
 185 solves leader election in $O(\log n/\sqrt{\epsilon})$ rounds, but uses $O(m + n \log n)$ messages. While

186 these two algorithms are randomized, we present a deterministic algorithm that takes
 187 $O(\log^2 n / \sqrt{\epsilon})$ rounds and $O(m + n\sqrt{\epsilon} \log^2 n)$ messages to elect a leader. The determin-
 188 istic algorithm builds on the similar idea of the improved algorithm. In all the algorithms,
 189 we assume that the nodes can distinguish between original edges in the given graph and
 190 the smoothed edges.

191 3.1. A Simple Algorithm

192 We present a simple, yet efficient algorithm for leader election in the ϵ -smoothing
 193 model. At a high-level, the algorithm consists of two parts: (I) Smoothing, i.e., adding
 194 random edges to the given graph in such a way that the smoothed graph becomes an
 195 expander, (II) Compute a leader in the smoothed graph. Note that the nodes are fixed;
 196 so the elected leader is a valid leader in the given graph. The smoothed edges are used
 197 for the communication only.

198 **(I) Constructing smoothed graph (a random expander graph).** In the beginning,
 199 the algorithm adds $O(\log n)$ random edges per node according to the smoothing model.
 200 In particular, the algorithm runs the smoothing procedure for $\Theta(\log n / \epsilon)$ rounds. In
 201 every round, each node adds a smoothed edge with probability ϵ to one of the nodes
 202 selected uniformly at random. Thus, it is easy to show that with high probability a node
 203 will add $\Theta(\log n)$ random edges (smoothed edges). Let us call this graph as the smoothed
 204 graph $R(G) = (V, F)$ which is induced only by the smoothed edges F after $\Theta(\log n / \epsilon)$
 205 rounds. It is intuitive that $R(G)$ is an Erdős-Rényi random graph (expander), which is
 206 formally shown in [2] and gives the following Lemma.

207 **Lemma 1** (Lemma 3.1, [2]). *The smoothed graph $R(G)$ has a constant conductance and*
 208 *$O(\log n)$ mixing time.*

209 **(II) Computing a leader.** The second part of the algorithm computes leader among
 210 the n nodes. For this, the algorithm uses $R(G)$ as the communication graph. Our goal
 211 is to compute a leader using minimum number of messages and time. For this part, we
 212 adapt the leader election approach of [11] for general graphs. If the mixing time τ of a
 213 graph is known, then an algorithm in [11] computes a leader with high probability in
 214 $O(\tau)$ rounds and uses $O(\tau\sqrt{n} \log^{1.5} n)$ messages. In essence, we adapt this algorithm in
 215 the smoothed graph $R(G)$ since the mixing time $O(\log n)$ of $R(G)$ is known.

216 Let us discuss an outline of the algorithm. Recall that we consider anonymous
 217 network, i.e., nodes do not know each other's ID. We assume that nodes have unique
 218 IDs; otherwise each node can randomly pick a number (or rank) in the range $[1, n^4]$
 219 such that the numbers are distinct for all the nodes. The rank can be used as the ID
 220 of the nodes. The idea of the algorithm is to select a smaller committee nodes (called
 221 *candidate nodes*) which are responsible for electing a leader node among themselves. In
 222 fact, the maximum ID node among the candidate nodes will be elected as the leader and
 223 all other nodes put themselves in the NON-ELECTED state. This is a standard technique
 224 to reduce the message complexity. A random set of candidate nodes of size $\Theta(\log n)$
 225 is selected. For this, each node selects itself with probability $O(\log n / n)$ to become a
 226 candidate node. This selection is done locally at each node and hence the candidate
 227 nodes do not know each other initially. All the non-candidate nodes put themselves in
 228 the NON-ELECTED state. The candidate nodes communicate among themselves via
 229 some other nodes, called *referee nodes*. For this, each candidate node samples $2\sqrt{n \log n}$
 230 random referee nodes in the network. This referee sampling is done via performing
 231 random walks on $R(G)$ by token forwarding. Each candidate node creates $2\sqrt{n \log n}$
 232 random walk tokens and each token performs random walk of length $O(\log n)$ on the
 233 smoothed graph $R(G)$. Since the mixing time of $R(G)$ is $O(\log n)$, the tokens stop at
 234 random nodes after $O(\log n)$ steps. The ending nodes of the tokens after $O(\log n)$ steps
 235 act as the referee nodes. In this way each candidate node samples $2\sqrt{n \log n}$ random
 236 referee nodes. The reason behind sampling so many referee nodes is to make sure at least

237 one common referee node between any pair of candidate nodes' referees. Each candidate
 238 node sends its ID with the random walk tokens. An intermediate node or referee node
 239 may receive IDs (or tokens) from multiple candidate nodes. Since our goal is to elect
 240 the maximum ID node to be the leader, the referee nodes send back a *winner* message
 241 $\langle "WIN", node_id, count \rangle$ to the maximum ID candidate node only, via back tracking
 242 the random walk paths. In the winner message, *node_id* carries the maximum ID of the
 243 candidate node and *count* variable stores the number of tokens having the maximum ID.
 244 During the token forwarding process, an intermediate node forwards only the tokens with
 245 maximum ID among all the tokens received in a round (and discards all other tokens with
 246 smaller ID). The intermediate node also stores this ID and the port numbers, through
 247 which it has received the maximum ID token, for back tracking. In subsequent rounds, if
 248 an intermediate node receives any token with higher ID than the previous one, then it
 249 updates its stored ID and the port number accordingly.

250 During back tracking, an intermediate node on receiving multiple winner messages
 251 $\langle "WIN", node_id, count \rangle$, adds up the *count* for the maximum *node_id* and forwards
 252 to the back track node (it discards the winner messages with lower *node_id*). When
 253 back track finishes, each candidate node sums up the *count* in the winner messages it
 254 has received. The candidate node which receives $2\sqrt{n \log n}$ winner messages enters the
 255 ELECTED state. All the other candidate nodes enter the NON-ELECTED state. It
 256 is easy to see that only the maximum ID candidate node receives $2\sqrt{n \log n}$ winner
 257 messages and elected as the leader.

258 One difficult part in the algorithm is the congestion over the edges when performing
 259 many random walks in parallel as we consider CONGEST model. The congestion is
 260 handled by sending only the count of random walk tokens that need to be sent by a
 261 particular candidate node, and not the tokens themselves. A pseudocode is given in
 262 Algorithm 1 in the Appendix.

263 To show the correctness of the algorithm, we first discuss the following two results.
 264 The first one says that there is at least one candidate node with high probability. Also
 265 the size of the candidate nodes is not too large— bounded by $O(\log n)$. The second one
 266 says that there is at least one common referee node between any pair of candidate nodes
 267 with high probability.

268 **Lemma 2.** *The size of the candidate nodes is $\Theta(\log n)$ with high probability.*

269 **Proof.** Each node selects itself with probability $O(\log n/n)$ to become a candidate node.
 270 Thus, in expectation, $O(\log n)$ candidate nodes are selected. Then one can show using a
 271 standard Chernoff bound that the number of the selected candidate nodes is $\Theta(\log n)$
 272 with high probability. \square

273 **Lemma 3** (Theorem 1, [11]). *With high probability, there is a common referee node
 274 between any pair of candidate nodes.*

275 This implies that the maximum ID candidate node has a common referee node with
 276 all other candidate nodes. Thus, those common referees generate the winner message
 277 for the maximum ID candidate node only and discards all other random walk tokens.
 278 Further, during the token forwarding procedure, the maximum ID candidate node's tokens
 279 dominate all other tokens. This means in subsequent rounds when the winner messages
 280 reach their respective candidate nodes, no candidate node, other than the one having
 281 maximum ID, would have received all $2\sqrt{n \log n}$ winner messages with high probability.
 282 Therefore, only the candidate node with the maximum ID enters the ELECTED state
 283 with a high probability.

284 Thus we get the following result.

285 **Theorem 1.** *Given an anonymous graph $G = (V, E)$ in the ϵ -smoothing model, there
 286 exists a randomized distributed algorithm that computes a leader in G with high probability*

287 in $O(\frac{\log n}{\epsilon})$ rounds and incurs $O(\sqrt{n} \log^{2.5} n)$ messages (assuming that the smoothed edges
288 are added without sending any messages).

289 **Proof.** We already discussed that the algorithm correctly elects a leader with high
290 probability.

291 The time complexity of the algorithm is determined by two procedures: (I) Con-
292 structing smoothed graph– which takes $O(\log n/\epsilon)$ rounds. (II) Computing leader in the
293 smoothed graph, which requires to perform random walks of length $O(\log n)$ and back
294 tracking (in parallel). Further, there is no congestion due to performing multiple random
295 walks in parallel as we are sending the token counts and not the tokens themselves. All
296 other computations are done locally. This procedure takes $O(\log n)$ rounds. Thus, the
297 time complexity of the algorithm is $O(\log n/\epsilon + \log n) = O(\log n/\epsilon)$ rounds.

298 The message complexity of the algorithm is determined by the leader election
299 procedure in the smoothed graph $R(G)$. The number of the candidate nodes is $\Theta(\log n)$
300 with high probability. Each candidate node performs $2\sqrt{n \log n}$ random walks for
301 $O(\log n)$ steps. Thus a total of $O(\sqrt{n} \log^{5/2} n)$ messages are required for this step. The
302 back tracking of the winner messages may take at most the same number of messages.
303 Therefore message complexity of the algorithm is $O(\sqrt{n} \log^{2.5} n)$. \square

304 **Remark 1.** *In the above theorem, we assume that the smoothed edges are added by*
305 *the system without sending any messages. If we consider one message is used per edge*
306 *addition, then the message complexity of the smoothing process would be $O(n \log n)$, as*
307 *each node adds $O(\log n)$ random edges. Then the message complexity of the algorithm*
308 *would be $O(n \log n)$.*

309 3.2. An Improved Algorithm

310 Now we present an improved algorithm which is a variant of the previous algorithm
311 and has a lower time complexity. This algorithm solves the leader election in $O(\log n/\sqrt{\epsilon})$
312 rounds. It crucially applies the previous algorithm over a super-graph induced by
313 minimum-spanning-tree (MST) fragments as the super nodes. Broadly, this algorithm
314 consists of three parts: (I) Compute MST fragments. (II) Apply smoothing to add an
315 expander over the super-graph induced by the MST fragments. (III) Compute a leader
316 using the previous random walk based sampling algorithm on the smoothed super-graph.

317 Let us describe the algorithm and simultaneously its analysis.

318 **(I) Computing MST fragments.** We use controlled Gallagher-Humblet-Spira (GHS)
319 algorithm to construct MST fragments, see Section 7.4 in [32]. The main difference
320 compared to the standard GHS algorithm is that the growth (size, diameter) of fragments
321 are controlled during merging of fragments.

322 The controlled GHS algorithm runs in phases [32]. The algorithm starts with each
323 individual node as a fragment and merges fragments in each phase. In each phase, the
324 algorithm maintains the following invariant: Each MST fragment has a leader (which is
325 the root of the tree) and all nodes know their respective parents and children. Initially,
326 each node (a singleton fragment) is a leader node; subsequently each fragment will have
327 one leader (root) node. Each fragment is identified by the identifier of its root (called
328 the fragment ID) and each node in the fragment knows its fragment ID. Each fragment's
329 operation is coordinated by the respective fragment's leader. Each phase consists of two
330 major operations: (1) Finding minimum-weight-outgoing-edge (MOE) of all fragments
331 and (2) Merging fragments via their MOEs. Note that, while the controlled GHS finds
332 an MST in a weighted graph (where edges have weight), it also works on an unweighted
333 graph where one can consider the edge-weights are 1. The details on finding MOE and
334 merging fragments can be found in Section 7.3.1 of [32].

335 The algorithm starts by running a controlled GHS algorithm for $\log(\frac{1}{\sqrt{\epsilon}})$ phases (ϵ
336 is the smoothing parameter). The size and diameter of each MST fragment are $\Omega(1/\sqrt{\epsilon})$
337 and $O(1/\sqrt{\epsilon})$ respectively, and there will be $O(n\sqrt{\epsilon})$ such fragments [[32], Section 7.4].

Each fragment will be treated as a super-node. Each node, that is not a root node, maintains two IDs: 1. Its fragment ID, denoted by FID, and 2. Its own ID in the given graph which we denote by GID in this section. For a root node, its FID would be same as its GID.

The following Lemma shows that this MST fragments construction takes $O(\frac{\log^* n}{\sqrt{\epsilon}})$ rounds and uses $O(m)$ messages.

Lemma 4. *The number of MSTs formed after $\log(\frac{1}{\sqrt{\epsilon}})$ phases is $O(n\sqrt{\epsilon})$. The MST fragments construction takes $O(\frac{\log^* n}{\sqrt{\epsilon}})$ rounds and $O(m)$ messages, where m is the number of edges in the given graph.*

Proof. It is shown in Corollary 7.1 of [32] that the number of MST fragments at the beginning of phase i is at most $\frac{n}{2^i}$. So after $\log(\frac{1}{\sqrt{\epsilon}})$ phases, the number of MSTs is at most $n/2^{\log(\frac{1}{\sqrt{\epsilon}})} = O(n\sqrt{\epsilon})$. It also follows from Lemma 7.3 of [32] that the diameter of each MST at the beginning of phase i is bounded by 2^i . Hence the diameter of the MST fragments after $\log(\frac{1}{\sqrt{\epsilon}})$ phases is $O(2^{\log(\frac{1}{\sqrt{\epsilon}})}) = O(1/\sqrt{\epsilon})$.

The time and message complexities follows from the 3 sub-procedures.

1. Finding MOE from each node in parallel takes $O(2^i)$ rounds for the i^{th} phase. So for the $\log(\frac{1}{\sqrt{\epsilon}})$ phases, it takes: $O\left(\sum_{i=1}^{\log(\frac{1}{\sqrt{\epsilon}})} 2^i\right) = O\left(\frac{2}{\sqrt{\epsilon}}(1 - \sqrt{\epsilon})\right) = O\left(\frac{1}{\sqrt{\epsilon}}\right)$ rounds.

The number of messages required is: $O\left(\sum_{v \in V} 2 \cdot d(v) + \sum_{i=1}^{\log(\frac{1}{\sqrt{\epsilon}})} \sum_{v \in V} O(1)\right) = O\left(m + n \log\left(\frac{1}{\sqrt{\epsilon}}\right)\right)$.

2. Selecting MOE for merging fragments takes $O(2^i \log^* n)$ rounds and $O(n \log^* n)$ messages per phase. Thus in total it takes, $O\left(\sum_{i=1}^{\log(\frac{1}{\sqrt{\epsilon}})} (2^i \log^* n)\right) = O\left(\frac{\log^* n}{\sqrt{\epsilon}}\right)$ rounds and $O\left(n \log^* n \log\left(\frac{1}{\sqrt{\epsilon}}\right)\right)$ messages.

3. Merging fragments takes $O(2^i)$ rounds and $O(n)$ messages per phase. So in total it takes, $O\left(\frac{1}{\sqrt{\epsilon}}\right)$ rounds and $O\left(n \log\left(\frac{1}{\sqrt{\epsilon}}\right)\right)$ messages.

Thus, MST fragments construction takes $O(\frac{\log^* n}{\sqrt{\epsilon}})$ rounds and $O(m)$ messages. \square

(II) Constructing a smoothed graph. After constructing the MST forest, perform $O(\frac{\log n}{\sqrt{\epsilon}})$ rounds of smoothing. Let S denotes the set of smoothed edges added in the graph. The probability that a smoothed edge added between two nodes in G is $\Theta\left(\frac{\sqrt{\epsilon} \log n}{n}\right)$. Consider each MST fragment as a super-node. Let the set of super-node be V' ; then $|V'| = O(n\sqrt{\epsilon})$ as there are so many MST fragments. Let $S' \subseteq S$ be the set of inter-super-node smoothed edges. Consider the super-graph $R'(V', S')$; we call it as smoothed super-graph. Then it is easy to show that $R'(V', S')$ is an Erdős-Rényi random graph [2]. Thus, the mixing time of $R'(V', S')$ is $O(\log(n\sqrt{\epsilon})) = O(\log n)$.

(III) Computing a leader. Now we apply the similar random walk based approach on the smoothed super-graph $R'(V', S')$ to elect a leader in G . In particular, we run the ‘‘Computing a leader’’ procedure of the previous algorithm over this super-graph $R'(V', S')$, where the root node in each super-node simulates and coordinates the tasks of a node there (recall that each MST fragment has a specified root node). First, $O(\log n)$ random set of candidate super-nodes are selected. For this, each super-node selects itself with probability $\Theta\left(\frac{\log n}{n\sqrt{\epsilon}}\right)$. Note that when we say a ‘‘super-node does something’’, it means the root node inside the super-node does this and communicate to all the nodes in the fragment. To implement this in a super-node, the root node coordinates the tasks with the fragment nodes. Thus, an extra term of $O(1/\sqrt{\epsilon})$ rounds may be incurred due

381 to the communication within a super-node. This is because, the diameter of each MST
 382 fragment is $O(1/\sqrt{\epsilon})$, so the communication within a fragment takes $O(1/\sqrt{\epsilon})$ rounds.
 383 Further recall that each super-node has an ID, the fragment ID which is essentially the
 384 ID of the root node in the fragment. Then in the similar way, each candidate super-node
 385 samples $O(\sqrt{n \log n})$ referee nodes (super-nodes)² by performing $O(\sqrt{n \log n})$ random
 386 walks of length $O(\log n)$ (since the mixing time of R' is $O(\log n)$). Then the referee
 387 nodes send back winner message to the maximum ID super-node. Finally, the super-node
 388 with maximum ID becomes the leader. Since the super-node carries the ID of its root
 389 node, the root node in the maximum ID super-node becomes the leader in G .

390 One crucial part remains to discuss is— how the super-nodes perform multiple random
 391 walks on $R'(V', S')$ in parallel. Let r be the root node in a super-node T . In the beginning
 392 of this procedure – “computing a leader” – every node in T sends the number of its
 393 inter-super-node smoothed edges (i.e., outgoing smoothed edges) to the root r . The root
 394 node computes the total number of outgoing smoothed edges from T and also stores
 395 the ID of the fragment nodes and their outgoing edge number. Suppose a super-node
 396 T has k random walk tokens to forward in the next round. For each token, r (locally)
 397 selects one outgoing smoothed edge randomly among all the outgoing edges of T . Then r
 398 sends a count of the number of tokens to the corresponding fragment nodes. A count
 399 ℓ of a fragment node v indicates that ℓ number of random walk tokens are selected to
 400 move over the outgoing edges of the node v . The root sends this random walk count
 401 information to all the selected nodes in T in parallel. Then the fragments nodes forward
 402 their tokens to the outgoing neighbors selected uniformly at random (among the outgoing
 403 edges). The node forwards the tokens together with the count through its corresponding
 404 outgoing smoothed edges to avoid any congestion. When a node receives random walk
 405 tokens from a different super-node, it sends the count (of the tokens) to the root node of
 406 its super-node. The root node sums up the count to know the total number of received
 407 tokens from the same candidate node. Then it forwards the tokens, in the same way as
 408 described above, for the next step. A pseudocode is given in Algorithm 2 in the Appendix.

409 **Theorem 2.** *Given an anonymous graph $G = (V, E)$ in the ϵ -smoothing model, there*
 410 *exists a randomized distributed algorithm that computes a leader in G with high probability*
 411 *in $O(\frac{\log n}{\sqrt{\epsilon}})$ rounds and incurs $O(m + n \log n)$ messages.*

412 **Proof.** The algorithm correctly elects a leader as the previous algorithm does.

413 **Time complexity.** The first procedure (‘computing MST fragments’) takes $O(\frac{\log^* n}{\sqrt{\epsilon}})$
 414 rounds (follows from Lemma 4). The second procedure (‘constructing smoothed graph’)
 415 takes $O(\frac{\log n}{\sqrt{\epsilon}})$ rounds as we apply smoothing for so many rounds. The third procedure
 416 (‘computing a leader’) takes $O(\frac{\log n}{\sqrt{\epsilon}})$ rounds. This is because, the random walks are
 417 performed over the super-nodes in parallel for $O(\log n)$ rounds and one step of the
 418 random walk may take extra $\frac{1}{\sqrt{\epsilon}}$ rounds for communication inside a super-node. In
 419 the calculation, we implicitly assume $O(\log(n\sqrt{\epsilon})) = O(\log n)$. Therefore, the time
 420 complexity the algorithm is: $O(\frac{\log^* n}{\sqrt{\epsilon}} + \frac{\log n}{\sqrt{\epsilon}} + \frac{\log n}{\sqrt{\epsilon}}) = O(\frac{\log n}{\sqrt{\epsilon}})$ rounds.

421 **Message complexity.** The first procedure uses $O(m)$ messages (follows from Lemma
 422 4). The second procedure uses no messages if we assume that the smoothing process
 423 (addition of random edges) is done by the system without incurring any messages. It
 424 may use $O(n \log n)$ messages if we assume one message cost per one edge addition. The
 425 third procedure uses $O(\sqrt{n} \log^{2.5} n + n \log n)$ messages. The term $O(\sqrt{n} \log^{2.5} n)$ comes
 426 from performing $O(\sqrt{n} \log n)$ random walks for $O(\log n)$ steps from $O(\log n)$ candidate
 427 nodes. The term $O(n \log n)$ comes from the communication inside super-nodes or MST

² Since $|V'| = O(n\sqrt{\epsilon})$, it would suffice to sample $O(\sqrt{n\sqrt{\epsilon} \log n})$ referee nodes to ensure a common referee node between any pair of candidate super-nodes.

428 fragments via the spanning tree edges for $O(\log n)$ rounds. Thus, the message complexity
 429 of the algorithm is $O(m + n \log n)$. \square

430 3.3. A Deterministic Algorithm

431 In this section, we describe a deterministic algorithm for leader election in the
 432 smoothing model. Note that the smoothing part uses random bits, which we cannot
 433 avoid in this smoothing model. The rest of the algorithm is deterministic. That's why we
 434 are calling the algorithm *deterministic*. The algorithm builds on the similar ideas of the
 435 previous improved algorithm (cf. Section 3.2). Analogously, it has three procedures: (I)
 436 Compute MST fragments, (II) Construct smoothed graph, and (III) Compute a leader.
 437 The first two procedures (compute MST fragments, construct smoothed graph) follow the
 438 same procedures as in the previous algorithm. Note that the controlled GHS algorithm
 439 (which is used to compute MST fragments) is deterministic. Now for the third procedure,
 440 we use a deterministic approach instead of applying random walks. In fact, we use
 441 a deterministic algorithm from [14] which had solved the leader election problem in
 442 $O(D \log n)$ rounds and $O(m \log n)$ messages.

443 After running the first two procedures, we obtain the smoothed graph $R'(V', S')$
 444 where V' is the set of super-nodes (MST fragments), $|V'| = O(n\sqrt{\epsilon})$ and S' is the set
 445 of inter-super-node smoothed edges. Since $R'(V', S')$ is a random expander graph, its
 446 diameter is $O(\log(n\sqrt{\epsilon})) = O(\log n)$, e.g., see Theorem 8.13 of [33].

447 Now for the third procedure (computing a leader), we use the deterministic algorithm
 448 from [14] and describe how to run it on R' to elect a leader. This algorithm runs in
 449 phases, and each phase consists of 4 stages. First all the nodes become candidate nodes.
 450 In each phase i , every candidate node is required to develop a BFS tree of depth 2^{i-1}
 451 (for $i = 1, 2, \dots, \log(n\sqrt{\epsilon})$) and then carry out operations in 4 stages. In the first stage,
 452 every candidate node v (which translates as the root node of the BFS tree) sends a token
 453 **ELECT(phase, ID, counter)** on its BFS tree, where phase refers to the phase i that
 454 this candidate node is in, ID is candidate node's ID and counter refers to the depth of
 455 BFS tree in any phase i that this candidate node needs to develop. As the tree develops,
 456 this counter is decremented by 1. At the beginning, that is, at the candidate node, it is
 457 set to 2^{i-1} . In the 2nd stage, it receives an ACK token from all its BFS children of the
 458 form: **ACK(ID, max, phase_status)**, where ID is the ID of the candidate node that
 459 generated this token, max is the maximum ID encountered by that candidate node, and
 460 phase_status is about whether its phase status is same as or lower than this candidate
 461 node's phase. Next, the candidate node v updates its field **v.max** with the highest ID
 462 it has received, that was contained in the max field among the ACK tokens and sends
 463 a **CONFIRM(v.max)** token along its BFS tree in the 3rd stage. In the 4th stage,
 464 v receives a **VICTOR(phase, ID)** token from all its neighbors which contains the
 465 **v.max** – the highest ID node – encountered by them. If this **v.max** is the same as
 466 the ID of the candidate node, v , then v continues to the next phase; otherwise assumes
 467 a NON-ELECTED state. In every phase, all the above 4 stages are repeated. After
 468 $\log(n\sqrt{\epsilon})$ phases, only the maximum ID node will be ELECTED as a leader and all the
 469 other nodes have NON-ELECTED status.

470 We now explain how to adapt this procedure on the smoothed graph $R'(V', S')$.
 471 The super-nodes are MST fragments which contain a root node. Since the root node
 472 coordinates the tasks inside a super-node, all the root nodes mark themselves as the
 473 candidate nodes in the beginning of the algorithm (essentially, all the super-nodes become
 474 candidate nodes). Each super-node acts as a single node and implement all the phases
 475 on the smoothed graph $R'(V', S')$ only, e.g., the BFS trees are constructed on $R'(V', S')$.
 476 Inside a super-node, the root controls and coordinates the tasks and all the communication
 477 done through MST edges inside the supernode. Thus, an extra factor of $O(1/\sqrt{\epsilon})$ rounds
 478 may be incurred for each step of the above algorithm due to the communication within a
 479 super-node. This is because, the diameter of each MST fragment is $O(1/\sqrt{\epsilon})$, so the
 480 communication within a fragment takes $O(1/\sqrt{\epsilon})$ rounds. In the end, one super-node

481 will be elected as the leader. The root node of this super-node will be the leader in the
 482 graph.

483 We now discuss the time and message complexity of the entire deterministic algo-
 484 rithm.

485 **Theorem 3.** *Given an anonymous graph $G = (V, E)$ in the ϵ -smoothing model, there*
 486 *exists a deterministic distributed algorithm that solves the leader election problem in*
 487 $O\left(\frac{\log^2 n}{\sqrt{\epsilon}}\right)$ rounds using $O(m + n\sqrt{\epsilon} \log^2 n)$ messages.

488 **Proof.** It follows from Theorem 2 that the first two procedures takes $O\left(\frac{\log n}{\sqrt{\epsilon}}\right)$ rounds
 489 and $O\left(m + n \log^* n \log\left(\frac{1}{\sqrt{\epsilon}}\right)\right)$ messages.

490 It follows Lemma 4.8 in [14] that third procedure can have $O(\log(n\sqrt{\epsilon}))$ phases.
 491 Further, each of the 4 stages can take at most the diameter time of the smoothed graph
 492 R' . Since the diameter of R' is $O(\log(n\sqrt{\epsilon}))$, and communication inside a super-node
 493 takes $O\left(\frac{1}{\sqrt{\epsilon}}\right)$ rounds, the total time for the third procedure is:

$$494 O\left(\frac{1}{\sqrt{\epsilon}}\right) \cdot \log(n\sqrt{\epsilon}) \cdot \log(n\sqrt{\epsilon}) = O\left(\frac{\log^2(n\sqrt{\epsilon})}{\sqrt{\epsilon}}\right) = O\left(\frac{\log^2 n}{\sqrt{\epsilon}}\right) \text{ rounds.}$$

495 Thus, the time complexity of the algorithm is $O\left(\frac{\log^2 n}{\sqrt{\epsilon}}\right)$ rounds.

496 Let us calculate the message complexity of the third procedure. Consider a phase
 497 i of the algorithm. The message exchanges within a fragment happen over the MST
 498 edges. Thus, it uses $O(n\sqrt{\epsilon}) \times O\left(\frac{1}{\sqrt{\epsilon}}\right) = O(n)$ messages inside all the MST fragments
 499 (super-nodes). Message exchanges between the MST fragments happen over the inter-
 500 super-node smoothed edges, which is $O(n\sqrt{\epsilon} \log n)$. Therefore, total number of messages
 501 uses in $O(\log(n\sqrt{\epsilon}))$ phases is: $O(\log(n\sqrt{\epsilon})) \times O(n + n\sqrt{\epsilon} \log n) = O(n\sqrt{\epsilon} \log^2 n)$.

502 Thus, the message complexity of the algorithm is: $O\left(m + n \log^* n \log\left(\frac{1}{\sqrt{\epsilon}}\right)\right) +$
 503 $O(n\sqrt{\epsilon} \log^2 n) = O(m + n\sqrt{\epsilon} \log^2 n)$. \square

504 4. Conclusion

505 We studied smoothed analysis of leader election, one of the fundamental problem
 506 in distributed networks. We consider the same smoothing model as introduced by
 507 Chatterjee et al. [2] in distributed networks. We present two randomized algorithms and
 508 a deterministic algorithm and discuss their smoothed complexity of time and messages.
 509 The time and message complexity of our first algorithm are optimal, up to a **polylog n**
 510 factor. For the second algorithm there is a trade off as it solves the problem in less
 511 number of rounds, but incurs more messages. We present a third algorithm which is
 512 deterministic but takes slightly more time to solve the leader election problem.

513 We believe this work extends the study of smoothed analysis of distributed problems.
 514 An obvious next step is to investigate how tight these complexities are by analyzing the
 515 lower time and message bound of these algorithms. Another line of work could probe the
 516 behavior of these algorithms in a different smoothing model and when the nodes can not
 517 differentiate between the input edges and the smoothed edges.

518 **Funding:** This work was supported, in part, by NTRO project fund, R.C.Bose Centre for
 519 Cryptology and Security, ISI, Kolkata, India.

520 **Appendix A. Pseudocode of the First Algorithm in Section 3.1**

Algorithm 1 SIMPLE-LEADER ELECTION-ALGORITHM

- 1: Run $\Theta(\frac{\log n}{\epsilon})$ rounds of smoothing.
 - 2: Each node decides to become a candidate with probability $c_1 \frac{\log n}{n}$. If it does not become a candidate, it assumes the NON-ELECTED state.
 - 3: The following procedure is carried out by all candidate nodes in parallel.
 - 4: **procedure** RANDOM WALK
 - 5: Each candidate node u starts $2\sqrt{n \log n}$ random walks of length $O(\log n)$ in parallel with other candidate nodes and sends tokens as $\langle node_id, count = 2\sqrt{n \log n} \rangle$ each of $O(\log n)$ bits. Node, v , that receives a token of the form $\langle node_id, count \rangle$, samples $count$ times randomly among its $O(\log n)$ neighbors. It then prepares a token, for each of its sampled neighbors, of the form: $\langle node_id, \alpha \rangle$, where $\alpha (\leq count)$ is the number of times that neighbor was sampled.
 - 6: At any intermediate node, v , in a random walk, of all the received candidate tokens, v is only concerned about the one containing the highest ID. The ID in that token is stored in a variable called **highest** and the ID of the port on which it was received in **port**. Every intermediate node, v , only stores the particulars of, and forwards, the token containing highest ID, and discards all the other tokens.
 - 7: After round $O(\log n)$, when all the random walks are completed, referees (i.e. a node at the end of a random walk), owing to the last step in the **Random Walk** procedure, will only receive the token containing highest ID encountered in their random walk. Thus, each referee generates a winner message of the form $\langle "WIN", node_id, count \rangle$ and sends it on the port id stored in their field **port**. Here $node_id$ refers to the highest ID it has received, "WIN" denotes the winner token and $count$ refers to accumulated count of winner tokens for each $node_id$, which is initially 1. The intermediate nodes will not generate this message, only the referee nodes will. The intermediate nodes will add the $count$ they receive for one particular candidate's $node_id$ and update that field with the sum value. The intermediate nodes then send the token $\langle "WIN", node_id, count \rangle$ with updated fields, on the port id stored in their field **port**, leading to the previous intermediate node in that random walk.
 - 8: When a candidate node receives a message of this form $\langle "WIN", node_id, count \rangle$ from all its random walks (which will be equal to its degree $O(\log n)$), and achieves a total $count = 2\sqrt{n \log n}$, it enters the ELECTED state. The candidate nodes, when they do not receive $2\sqrt{n \log n}$ "WIN" tokens, assume the NON-ELECTED state.
-

521 **Appendix B. Pseudocode of the Second Algorithm Section 3.2****Algorithm 2** IMPROVED-LEADER-ELECTION-ALGORITHM

-
- 1: Run $\log(\frac{1}{\sqrt{\epsilon}})$ phases of controlled GHS.
 - 2: Run $\frac{\log(n\sqrt{\epsilon})}{\sqrt{\epsilon}}$ rounds of smoothing.
 - 3: The root node in every MST decides to become a candidate node independently with probability $c_1 \frac{\log n}{n\sqrt{\epsilon}}$. All nodes that do not become a candidate, assume the NON-ELECTED state.
 - 4: All candidate nodes generate a token of the form $\langle node_id, count = O(\sqrt{n\sqrt{\epsilon} \log n}) \rangle$ and communicate to the other nodes in their fragment so as to begin the Random walk procedure as described below. All candidate nodes carry out the following procedure in parallel.
 - 5: **procedure** RANDOM WALK
 - 6: Node, v , belonging to an MST, T , checks if it has a smoothed edge with any node from the neighboring fragment. Nodes ask their neighbors for their FID. If it is different, the node in this fragment decides to pass on the token to it.
 - 7: That node sends this message, $\langle \text{number of outgoing smoothed edges, } GID \rangle$ to its root node. After receiving the total number of outgoing edges, the root node samples $count$ times among them. It then sends each of those nodes a token of the form $\langle GID, node_id, \text{no. of times each of the outgoing edge was sampled} \rangle$, where $node_id$ is the candidate node's ID. Further, on receiving this token with the updated count, the nodes with the GID forward these tokens on their outgoing edges.
 - 8: As the node belonging to the next intermediate fragment receives multiple tokens, it discards all but the token with the highest ID. This node stores the ID from this token in **highest** and the port it was received on in **port**. It then forwards the updated token, along with its GID, to the root node of its fragment which stores the value of the highest ID received in its field **highest**, the port on which it was received in the field **port**. The fields **highest** and **port** are not to be updated unless a candidate token is encountered.
 - 9: After round $O(\frac{\log(n\sqrt{\epsilon})}{\sqrt{\epsilon}})$, when all the random walks are completed, all the referee supernodes' roots generate a winner token $\langle "WIN", node_id, count \rangle$ for their respective stored candidate nodes' token and send it on the port stored in **port**, on which they received the last candidate token.
 - 10: When an intermediate node receives winner tokens, it sends it to the root of its MST. The root node sends the winner token after adding up the $count$ value from all the received tokens of same $node_id$, to the node whose GID is stored at the root node.
 - 11: That node with ID as GID also has the port, on which it received the last token with the highest ID, stored in its field **port**. It sends out the token on that port. This way one and only one random walk is completely traced back.
 - 12: The root node that receives the token with $count = O(\sqrt{(n\sqrt{\epsilon}) \log n})$ enters the ELECTED state. After $O(\frac{\log(n\sqrt{\epsilon})}{\sqrt{\epsilon}})$ rounds, candidate nodes that do not receive that value for $count$ enter the NON ELECTED state.
-

References

1. Molla, A.R.; Shur, D. Smoothed Analysis of Leader Election in Distributed Networks. Proceedings of 22nd International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2020); Devismes, S.; Mittal, N., Eds. Springer, 2020, Vol. 12514, *Lecture Notes in Computer Science*, pp. 183–198. doi:10.1007/978-3-030-64348-5_14.
2. Chatterjee, S.; Pandurangan, G.; Pham, N.D. Distributed MST: A Smoothed Analysis. Proc. of the 21st International Conference on Distributed Computing and Networking (ICDCN), 2020, pp. 15:1–15:10.
3. Dinitz, M.; Fineman, J.; Gilbert, S.; Newport, C. Smoothed Analysis of Dynamic Networks. Distributed Computing; Moses, Y., Ed.; Springer Berlin Heidelberg: Berlin, Heidelberg, 2015; pp. 513–527.
4. Shi, E.; Perrig, A. Designing secure sensor networks. *IEEE Wirel. Commun.* **2004**, *11*, 38–43.

5. Rahman, M.U. Leader Election in the Internet of Things: Challenges and Opportunities. *CoRR* **2019**, *abs/1911.00759*, [1911.00759].
6. Anderson, D.P.; Kubiawicz, J. Introduction to Distributed Algorithms **2002**.
7. Kutten, S.; Zinenko, D. Low Communication Self-stabilization through Randomization. DISC 2010. Springer, 2010, Vol. 6343, pp. 465–479.
8. Ratnasamy, S.; Francis, P.; Handley, M.; Karp, R.M.; Shenker, S. A scalable content-addressable network. SIGCOMM. ACM, 2001, pp. 161–172.
9. Wright, A. Contemporary approaches to fault tolerance. *Commun. ACM* **2009**, *52*, 13–15.
10. Kadjouh, N.; Bounceur, A.; Bezoui, M.; Khanouche, M.E.; Euler, R.; Hammoudeh, M.; Lagadec, L.; Jabbar, S.; Al-Turjman, F. A Dominating Tree Based Leader Election Algorithm for Smart Cities IoT Infrastructure. *Mobile Networks and Applications* **2020**, *65*. doi:10.1007/s11036-020-01599-z.
11. Kutten, S.; Pandurangan, G.; Peleg, D.; Robinson, P.; Trehan, A. Sublinear bounds for randomized leader election. *Theor. Comput. Sci.* **2015**, *561*, 134–143.
12. Levin, D.A.; Peres, Y.; Wilmer, E.L. *Markov chains and mixing times*; American Mathematical Society, 2006.
13. Gilbert, S.; Robinson, P.; Sourav, S. Leader Election in Well-Connected Graphs. Proc. of the ACM Symposium on Principles of Distributed Computing (PODC), 2018, pp. 227–236.
14. Kutten, S.; Pandurangan, G.; Peleg, D.; Robinson, P.; Trehan, A. On the Complexity of Universal Leader Election. *J. ACM* **2015**, *62*, 7:1–7:27.
15. Peleg, D. Distributed Computing: A Locality-Sensitive Approach **2000**.
16. Spielman, D.A.; Teng, S.H. Smoothed Analysis of Algorithms: Why the Simplex Algorithm Usually Takes Polynomial Time. *J. ACM* **2004**, *51*, 385–463. doi:10.1145/990308.990310.
17. Lann, G.L. Distributed Systems - Towards a Formal Approach. Information Processing, Proceedings of the 7th IFIP Congress 1977. North-Holland, 1977, pp. 155–160.
18. Gallager, R.G.; Humblet, P.A.; Spira, P.M. A Distributed Algorithm for Minimum-Weight Spanning Trees. *ACM Trans. Program. Lang. Syst.* **1983**, *5*, 66–77.
19. Afek, Y.; Gafni, E. Time and Message Bounds for Election in Synchronous and Asynchronous Complete Networks. *SIAM J. Comput.* **1991**, *20*, 376–394.
20. Humblet, P. Electing a leader in a clique in $O(n \log n)$ messages **1984**.
21. Korach, E.; Kutten, S.; Moran, S. A Modular Technique for the Design of Efficient Distributed Leader Finding Algorithms. *ACM Trans. Program. Lang. Syst.* **1990**, *12*, 84–101.
22. Korach, E.; Moran, S.; Zaks, S. The Optimality of Distributive Constructions of Minimum Weight and Degree Restricted Spanning Trees in a Complete Network of Processors. *SIAM J. Comput.* **1987**, *16*, 231–236.
23. Korach, E.; Moran, S.; Zaks, S. Optimal Lower Bounds for Some Distributed Algorithms for a Complete Network of Processors. *Theor. Comput. Sci.* **1989**, *64*, 125–132.
24. Singh, G. Efficient distributed algorithms for leader election in complete networks. ICDCS. IEEE Computer Society, 1991, pp. 472–479.
25. Khan, M.; Kuhn, F.; Malkhi, D.; Pandurangan, G.; Talwar, K. Efficient distributed approximation algorithms via probabilistic tree embeddings. *Distributed Comput.* **2012**, *25*, 189–205.
26. Augustine, J.; Molla, A.R.; Pandurangan, G. Sublinear Message Bounds for Randomized Agreement. PODC. ACM, 2018, pp. 315–324.
27. Roughgarden, T. Beyond Worst-Case Analysis. *Commun. ACM* **2019**, *62*, 88–96. doi:10.1145/3232535.
28. Etscheid, M.; Röglin, H. Smoothed Analysis of Local Search for the Maximum-Cut Problem. *ACM Trans. Algorithms* **2017**, *13*. doi:10.1145/3011870.
29. Elsässer, R.; Tscheuschner, T. Settling the Complexity of Local Max-Cut (Almost) Completely. ICALP, 2011.
30. Angel, O.; Bubeck, S.; Peres, Y.; Wei, F. Local max-cut in smoothed polynomial time. *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing* **2017**.
31. Arthur, D.; Manthey, B.; Röglin, H. Smoothed Analysis of the K-Means Method. *J. ACM* **2011**, *58*. doi:10.1145/2027216.2027217.
32. Pandurangan, G. Distributed network algorithms **2019**.
33. Blum, A.; Hopcroft, J.; Kannan, R. *Foundations of Data Science*; Cambridge University Press, 2020. doi:10.1017/9781108755528.