

Logarithmic Expected-Time Leader Election in Population Protocol Model

Yuichi Sudo^{*1}, Fukuhito Ooshita², Taisuke Izumi³, Hirotsugu Kakugawa⁴, and Toshimitsu Masuzawa¹

¹Graduate School of Information Science and Technology, Osaka University, Japan

²Graduate School of Science and Technology, Nara Institute of Science and Technology, Japan

³Graduate School of Engineering, Nagoya Institute of Technology, Japan

⁴Faculty of Science and Technology, Ryukoku University, Japan

Abstract

In this paper, we present the *first* leader election protocol in the population protocol model that stabilizes $O(\log n)$ parallel time in expectation with $O(\log n)$ states per agent, where n is the number of agents. Given a rough knowledge m of the population size n such that $m \geq \log_2 n$ and $m = O(\log n)$, the proposed protocol guarantees that exactly one leader is elected and the unique leader is kept forever thereafter.

1 Introduction

We consider the *population protocol* (PP) model [Ang+06] in this paper. A network called *population* consists of a large number of finite-state automata, called *agents*. Agents make *interactions* (i.e., pairwise communication) each other by which they update their states. The interactions are opportunistic, that is, they are unpredictable. Agents are strongly anonymous: they do not have identifiers and they cannot distinguish their neighbors with the same states. As with the majority of studies on population protocols, we assume that the network of agents is a complete graph and that the scheduler selects an interacting pair of agents at each step uniformly at random.

In this paper, we focus on leader election problem, which is one of the most fundamental and well studied problems in the PP model. The leader election problem requires that starting from a specific initial configuration, a population reaches a safe configuration in which exactly one leader exists and the population keeps that unique leader thereafter.

There have been many works which study the leader election problem in the PP model (Tables 1 and 2). Angluin et al. [Ang+06] gave the first leader election protocol, which stabilizes in $O(n)$ parallel time in expectation and uses only constant space of each agent, where n is the number of agents and “parallel time” means the number of steps in an execution divided by n . If we stick to constant space, this linear parallel time is optimal; Doty and Soloveichik [DS18] showed that any constant space protocol requires linear parallel time to elect a unique leader. Alistarh and Gelashvili [AG15] made a breakthrough in 2015; they achieve poly-logarithmic stabilization time ($O(\log^3 n)$ parallel time) by increasing the number of states from $O(1)$ to only $O(\log^3 n)$. Thereafter, the stabilization time has been improved by many studies [Bil+17; AAG18; GS18; GSU18; MST18]. Gąsieniec et al. [GSU18] gave a state-of-art protocol that stabilizes in $O(\log n \cdot \log \log n)$ parallel time with only $O(\log \log n)$ states. Its space complexity is optimal; Alistarh et al. [Ali+17] shows that any leader election algorithm with $o(n/(\text{polylog } n))$ parallel time requires $\Omega(\log \log n)$ states. Michail et al. [MST18] gave a protocol with $O(\log n)$ parallel time but with a linear number of states. Those protocols with non-constant number of states [AG15; Ali+17; Bil+17; AAG18; GS18; GSU18] are not *uniform*, that

^{*}Corresponding author: y-sudou[at]ist.osaka-u.ac.jp

Table 1: Leader Election Protocols. (Stabilization time is shown in terms of parallel time and in expectation.)

	States	Stabilization Time
[Ang+06]	$O(1)$	$O(n)$
[AG15]	$O(\log^3 n)$	$O(\log^3 n)$
[Ali+17]	$O(\log^2 n)$	$O(\log^{5.3} n \cdot \log \log n)$
[AAG18]	$O(\log n)$	$O(\log^2 n)$
[GS18]	$O(\log \log n)$	$O(\log^2 n)$
[GSU18]	$O(\log \log n)$	$O(\log n \cdot \log \log n)$
[MST18]	$O(n)$	$O(\log n)$
This work	$O(\log n)$	$O(\log n)$

Table 2: Lower Bounds for Leader Election (Stabilization time is shown in terms of parallel time and in expectation.)

	States	Stabilization Time
[DS18]	$O(1)$	$\Omega(n)$
[Ali+17]	$< 1/2 \log \log n$	$\Omega(n/(\text{polylog } n))$
[SM19]	any large	$\Omega(\log n)$

is, they require some rough knowledge of n . For example, in the protocol of [GS18], an $\Theta(\log \log n)$ value must be hard-coded to set the maximum value of one variable (named l in that paper). One can find detailed information about the leader election in the PP model in two survey papers [AG18; ER18].

The stabilization time of [MST18] is optimal; any leader election algorithm requires $\Omega(\log n)$ parallel time if it uses any large number of states and assumes the exact knowledge of population size n [SM19]. At the beginning of an execution, all the agents are in the same initial state specified by a protocol. Therefore, simple analysis on Coupon Collector’s problem shows that we cannot achieve $o(\log n)$ parallel stabilization time if an agent in the initial state is a leader. The lower bound of [SM19] shows that we cannot achieve $o(\log n)$ parallel time even if we define the initial state such that all the agents are non-leaders initially.

Our Contribution In this paper, we present the *first* time-optimal leader election protocol P_{LL} with sub-polynomial number of states. Specifically, the proposed protocol P_{LL} stabilizes in $O(\log n)$ parallel time and uses only $O(\log n)$ states per agent. Compared to the state of art protocol [GSU18], P_{LL} achieves shorter (and best possible) stabilization time but uses larger space of each agent. Compared to [MST18], P_{LL} achieves drastically small space while maintaining the same (and optimal) stabilization time. The protocol P_{LL} is non-uniform as with the existing non-constant space protocols; it requires a rough knowledge m of n such that $m \geq \log_2 n$ and $m = \Theta(\log n)$.

We give P_{LL} as an asymmetric protocol in the main part of this paper *only for simplicity* of presentation and analysis of stabilization time. Actually, we can change P_{LL} to a symmetric protocol, which we discuss in Section 4. In particular, that section proposes the first implementation of totally independent and fair (*i.e.*, unbiased) coin flips in the symmetric version of the PP model. Although the implementation of coin flips in [Ali+17] is almost independent and fair, the totally independent and fair coin clips achieved in this paper can contribute a simple analysis in a variety kind of protocols in the PP model.

2 Preliminaries

A *population* is a network consisting of *agents*. We denote the set of all the agents by V and let $n = |V|$. We assume that a population is complete graph, thus every pair of agents (u, v) can interact, where u serves as the *initiator* and v serves as the *responder* of the interaction. Throughout this paper, we use the phrase “with high probability” to denote probability $1 - O(n^{-1})$.

A *protocol* $P(Q, s_{\text{init}}, T, Y, \pi_{\text{out}})$ consists of a finite set Q of states, an initial state $s_{\text{init}} \in Q$, a transition function $T : Q \times Q \rightarrow Q \times Q$, a finite set Y of output symbols, and an output function $\pi_{\text{out}} : Q \rightarrow Y$. Every agent is in state s_{init} when an execution of protocol P begins. When two agents interact, T determines their next states according to their current states. The *output* of an agent is determined by π_{out} : the output of an agent in state q is $\pi_{\text{out}}(q)$. In this paper, we assume that a rough knowledge of an upper bound of n is available. Specifically, we assume that an integer m such that $m \geq \log_2 n$ and $m = \Theta(\log n)$ are given, thus we can design $P(Q, s_{\text{init}}, T, Y, \pi_{\text{out}})$ using this input m , i.e., the parameters Q, s_{init}, T, Y , and π_{out} can depend on m .

A *configuration* is a mapping $C : V \rightarrow Q$ that specifies the states of all the agents. We define $C_{\text{init}, P}$ as the configuration of P where every agent is in state s_{init} . We say that a configuration C changes to C' by the interaction $e = (u, v)$, denoted by $C \xrightarrow{e} C'$, if $(C'(u), C'(v)) = T(C(u), C(v))$ and $C'(w) = C(w)$ for all $w \in V \setminus \{u, v\}$.

A *schedule* $\gamma = \gamma_0, \gamma_1, \dots = (u_0, v_0), (u_1, v_1), \dots$ is a sequence of interactions. A schedule determines which interaction occurs at each *step*, i.e., interaction γ_t happens at step t under schedule γ . In particular, we consider a *uniformly random scheduler* $\Gamma = \Gamma_0, \Gamma_1, \dots$ in this paper: each Γ_t of the infinite sequence of interactions is a random variable such that $\Pr(\Gamma_t = (u, v)) = \frac{1}{n(n-1)}$ for any $t \geq 0$ and any distinct $u, v \in V$. Note that we use capital letter Γ for this uniform random scheduler while we refer a deterministic schedule with a lower case such as γ . Given an initial configuration C_0 and a schedule γ , the *execution* of protocol P is uniquely defined as $\Xi_P(C_0, \gamma) = C_0, C_1, \dots$ such that $C_t \xrightarrow{\gamma_t} C_{t+1}$ for all $t \geq 0$. Note that the execution $\Xi_P(C_0, \Gamma) = C_0, C_1, \dots$ under the uniformly random scheduler Γ is a sequence of configurations where each C_i is a random variable. For a schedule $\gamma = \gamma_0, \gamma_1, \dots$ and any $t \geq 0$, we say that agent $v \in V$ *participates* in γ_t if v is either the initiator or the responder of γ_t . We say that a configuration C of protocol P is *reachable* if there exists a finite schedule $\gamma = \gamma_0, \gamma_1, \dots, \gamma_{t-1}$ such that $\Xi_P(C_{\text{init}, P}, \gamma) = C_0, C_1, \dots, C_t$ and $C = C_t$. We define $\mathcal{C}_{\text{all}}(P)$ as the set of all reachable configurations of P .

The leader election problem requires that every agent should output L or F which means “leader” or “follower” respectively. Let \mathcal{S}_P be the set of configurations such that, for any configuration $C \in \mathcal{S}_P$, exactly one agent outputs L (i.e., is a leader) in C and no agent changes its output in execution $\Xi_P(C, \gamma)$ for any schedule γ . We say that a protocol P is a leader election protocol or solves the leader election problem if execution $\Xi_P(C_{\text{init}, P}, \Gamma)$ reaches a configuration in \mathcal{S}_P with probability 1. For any leader election protocol P , we define the expected stabilization time of P as the expected number of steps during which execution $\Xi_P(C_{\text{init}, P}, \Gamma)$ reaches a configuration in \mathcal{S}_P , divided by the number of agents n . The division by n is needed because we evaluate the stabilization time in terms of parallel time.

We write the natural logarithm of x as $\ln x$ and the logarithm of x with base 2 as $\lg x$. We do not indicate the base of logarithm in an asymptotical expression such as $O(\log n)$. By an abuse of notation, we will identify an interaction (u, v) with the set $\{u, v\}$ whenever convenient.

Throughout this paper, we will use the following three variants of Chernoff bounds.

Lemma 1 ([MU05], Theorems 4.4, 4.5). *Let X_1, \dots, X_s be independent Poisson trials, and let $X = \sum_{i=1}^s X_i$. Then*

$$\forall \delta, 0 \leq \delta \leq 1 : \Pr(X \geq (1 + \delta)\mathbf{E}[X]) \leq e^{-\delta^2 \mathbf{E}[X]/3}, \quad (1)$$

$$\forall \delta, 0 < \delta < 1 : \Pr(X \leq (1 - \delta)\mathbf{E}[X]) \leq e^{-\delta^2 \mathbf{E}[X]/2}. \quad (2)$$

In the proposed protocol, we often use *one-way epidemic* [AAE08]. The notion of one-way epidemic is formalized as follows. Let $\gamma = \gamma_0, \gamma_1, \dots$ be an infinite sequence of interactions, V' be a set of agents ($V' \subseteq V$), and r be an agent in V' . The *epidemic function* $I_{V', r, \gamma} : [0, \infty) \rightarrow 2^V$ is defined as follows: $I_{V', r, \gamma}(0) = \{r\}$, and for $t = 1, 2, \dots$, $I_{V', r, \gamma}(t) = I_{V', r, \gamma}(t-1) \cup (\gamma_{t-1} \cap V')$ if $I_{V', r, \gamma}(t-1) \cap \gamma_{t-1} \neq \emptyset$; otherwise, $I_{V', r, \gamma}(t) = I_{V', r, \gamma}(t-1)$. We say that v is *infected* at step t if $v \in I_{V', r, \gamma}(t)$ in the epidemic in V' and under γ starting from agent r . At step 0, only r is infected; at later steps, an agent in V' becomes infected if it interacts with an infected agent. Once an agent becomes infected, it remains infected thereafter.

This abstract notion plays an important role in analyzing the expected stabilization time of a population protocol. For example, consider an execution $\Xi_P(C_0, \Gamma) = C_0, C_1, \dots$ where agents in V' have different values in variable *var* in configuration C_0 and the larger value is propagated from agent to agent whenever two agents in V' have an interaction. Clearly, all agents in V' have the maximum value of *var* when all agents in V' are infected in one-way epidemic in V' and under Γ starting from the agent with the maximum value *var* in configuration C_0 .

Angluin et al. [Ang+06] prove that one-way epidemic in the whole population V from any agent $r \in V$ finishes (i.e., all agents are infected) within $\Theta(n \log n)$ interactions with high probability. Furthermore, Sudo et al. [Sud+12] give a concrete lower bound on the probability that the epidemic in the whole population finishes within a given number of interactions. We generalize this lower bound for an epidemic in any set of agents (sub-population) $V' \subseteq V$ as follows while the proof is almost the same as the one in [Sud+12].

Lemma 2. *Let $V' \subseteq V$, $r \in V'$, $n' = |V'|$, and $t \in \mathbb{N}$. We have $\Pr(I_{V',r,\Gamma}(2\lceil n/n' \rceil t) \neq V') \leq ne^{-t/n}$.*

Proof. For each k ($2 \leq k \leq n'$), we define $T(k)$ as integer t such that $|I_{V',r,\Gamma}(t-1)| = k-1$ and $|I_{V',r,\Gamma}(t)| = k$, and define $T(1) = 0$. Intuitively, $T(k)$ is the number of interactions required to infect k agents in V' . Let $X_{\text{pre}} = T(\lceil \frac{n'+1}{2} \rceil)$ and $X_{\text{post}} = T(n') - T(n' - \lceil \frac{n'+1}{2} \rceil + 1)$.

Let k be any integer such that $1 \leq k \leq n'$. When k agents are infected, an agent is newly infected with probability $k(n'-k)/nC_2$ at every step. When $n'-k$ agents are infected, an agent is newly infected also with probability $k(n'-k)/nC_2$ at every step. Therefore, $T(k+1) - T(k)$ and $T(n'-k+1) - T(n'-k)$ have the same probability distribution. Thus, $X_{\text{pre}} = T(\lceil \frac{n'+1}{2} \rceil) = \sum_{j=1}^{\lceil (n'+1)/2 \rceil - 1} T(j+1) - T(j)$ and $X_{\text{post}} = T(n') - T(n' - \lceil \frac{n'+1}{2} \rceil + 1) = \sum_{j=1}^{\lceil (n'+1)/2 \rceil - 1} T(n' - j + 1) - T(n' - j)$ have the same probability distribution. Moreover, $X_{\text{pre}} + X_{\text{post}} \geq T(n')$ holds because $\lceil \frac{n'+1}{2} \rceil \geq n' - \lceil \frac{n'+1}{2} \rceil + 1$.

In what follows, we bound the probability that $X_{\text{post}} > \lceil n/n' \rceil t$. We denote $T(n' - \lceil \frac{n'+1}{2} \rceil + 1)$ by T_{half} . For any agent $v \in V$, let T_v be the minimum non-negative integer such that $v \in I_{V',r,\Gamma}(T_v)$, i.e., agent v becomes infected at the T_v -th step. We define $X_v = \max(T_{C_0,\Gamma}(v) - T_{\text{half}}, 0)$. Consider the case $v \notin I_{V',r,\Gamma}(T_{\text{half}})$. At any step $t \geq T_{\text{half}}$, at least $n' - \lceil \frac{n'+1}{2} \rceil + 1 \geq \frac{n'}{2}$ agents are infected. Therefore, each interaction Γ_t such that $(t \geq T_{\text{half}})$ infects v with the probability at least $\frac{1}{nC_2} \cdot \frac{n'}{2} > \frac{n'}{n^2}$, hence we have $\Pr(X_v > \lceil n/n' \rceil t) \leq \left(1 - \frac{n'}{n^2}\right)^{nt/n'} \leq e^{-t/n}$. Since the number of non-infected agents at step T_{half} is at most $n'/2$, $\Pr(X_{\text{post}} > \lceil n/n' \rceil t) \leq \Pr(\bigvee_{v \in V} (X_v > \lceil n/n' \rceil t)) \leq \frac{n'}{2} \cdot e^{-t/n}$ holds.

By the equivalence of the distribution of X_{pre} and X_{post} , we have

$$\Pr(I_{V',r,\Gamma}(2\lceil n/n' \rceil t) \neq V') \leq \Pr(X_{\text{pre}} > \lceil n/n' \rceil t) + \Pr(X_{\text{post}} > \lceil n/n' \rceil t) \leq ne^{-t/n}.$$

□

3 Logarithmic Leader Election

3.1 Key Ideas

In this subsection, we give key ideas of the proposed protocol P_{LL} . Each agent v keeps output variable $v.\text{leader} \in \{\text{false}, \text{true}\}$. An agent outputs L when the value of leader is true and it outputs F when it is false . An execution of P_{LL} can be regarded as a competition by agents. At the beginning of the execution, every agent has $\text{leader} = \text{true}$, that is, all agents are leaders. Throughout the execution, every leader tries to remain a leader and tries to make all other leaders followers so that it becomes the unique leader in the population. The competition consists of three modules $\text{QuickElimination}()$, $\text{Tournament}()$, and $\text{BackUp}()$, which are executed in this order. These three modules guarantees the following properties:

QuickElimination(): An execution of this module takes $O(\log n)$ parallel time in expectation. For any $i \geq 2$, exactly i leaders survive an execution of $\text{QuickElimination}()$ with probability at most 2^{1-i} . The execution never eliminates all leaders, i.e., at least one leader always survives.

Tournament(): An execution of this module takes $O(\log n)$ parallel time in expectation. By an execution of $\text{Tournament}()$, which starts with $i \geq 2$ leaders, the unique leader is elected with probability at least $1 - O(i/\log n)$. This lower bound of probability is independent of an execution of the previous module $\text{QuickElimination}()$. The execution never eliminates all leaders, i.e., at least one leader always survives.

BackUp(): An execution of this component elects a unique leader within $O(\log^2 n)$ parallel time in expectation.

From above, it holds that, after executions of *QuickElimination()* and *Tournament()* finish, the number of leaders is exactly one with probability at least $1 - \sum_{i=2}^n O\left(\frac{i}{2^{i-1} \log n}\right) = 1 - O(1/\log n)$. Therefore, combined with *BackUp()*, protocol P_{LL} elects a unique leader within $(1 - O(1/\log n)) \cdot \log n + O(1/\log n) \cdot O(\log^2 n) = O(\log n)$ parallel time in expectation.

In the remainder of this subsection, we briefly give key ideas to design the three modules satisfying the above guarantees. We will present a way to implement the following ideas with $O(\log n)$ states per agent in the next subsection (Section 3.2). In this subsection, keep in mind only that these ideas are easily implemented with poly-logarithmic number of states per agent, that is, with a constant number of variables with $O(\log \log n)$ bits. For the following description of the key ideas, we assume a kind of global synchronization, for example, we assume that each agent begins an execution of *Tournament()* after *all* agents finish necessary operations of *QuickElimination()*. We also present a way to implement such a synchronization in Section 3.2.

3.1.1 Key Idea for *QuickElimination()*

The goal of this module is to reduce the number of leaders such that, for any $i \geq 2$, the resulting number of leaders is exactly i with probability at most 2^{1-i} while guaranteeing that not all leaders are eliminated. This module is based on almost the same idea as *the lottery protocol* in [Ali+17]. The protocol P_{LL} achieves much faster stabilization time than the lottery protocol thanks to tighter analysis on the number of surviving leaders, which we will see below, and the combination with the other two modules.

First, consider the following game:

- (i) Each agent in V executes a sequence of independent fair coin flips, each of which results in head with probability $1/2$ and tail with probability $1/2$, until it observes tail for the first time,
- (ii) Let s_v be the number of heads that v observes in the above coin flips and let $s_{\max} = \max_{v \in V} s_v$
- (iii) The agents v with $s_v = s_{\max}$ are winners and the other agents are losers.

Let $i \geq 2$ and $j \geq 0$. Consider the situation that exactly i agents observe that their first j coin flips result in head and define $p_{i,j}$ as the probability that all the i agents win the game in the end starting from this situation. Starting from this situation, if all the i agents observe tail in their $j+1$ -st coin flips then exactly i agents win the game with probability 1; if all the i agents observe head in their $j+1$ -st coin flips then exactly i agents win with probability $p_{i,j+1}$; Otherwise, the number of winners of the game is less than i with probability 1. Therefore, we have $p_{i,j} = 2^{-i} + 2^{-i} \cdot p_{i,j+1}$. Since we have $p_{i,j} = p_{i,j+1}$ thanks to memoryless property of this game, solving this equality gives $p_{i,j} = 1/(2^i - 1) \leq 2^{1-i}$. Let k_i be the minimum integer j such that exactly i agents observe that all of their first j coin flips result in head. We define $k_n = 0$ for simplicity. Then, for any $i \geq 0$, we have

$$\Pr(|\{v \in V \mid s_v = s_{\max}\}| = i) = \sum_{j=0}^{\infty} \Pr(k_i = j) \cdot p_{i,j} \leq 2^{1-i} \sum_{j=0}^{\infty} \Pr(k_i = j) \leq 2^{1-i}.$$

Module *QuickElimination()* simulates this game in the population protocol model. Every time an agent v has an interaction, we regard the interaction as the coin flip by v . If v is an initiator at the interaction, we regard the result of the coin flip as head; Otherwise we regard it as tail. The correctness of this simulation for coin flips comes from the definition of the uniformly random scheduler: at each step, an interaction where v is an initiator happens with probability $1/n$ and an interaction where v is a responder also happens with probability $1/n$. Strictly speaking, this simple simulation of coin flips does not guarantee independence of coin flips by u and v for any distinct $u, v \in V$. However, the actual P_{LL} defined in Section 3.2 completely simulates independent coin flips of leaders and we will explain it in Section 3.2. Each agent v computes and stores s_v on variable $v.\text{level}_Q$ by counting the number of interactions that it participates in as an initiator until it interacts as a responder for the first time. After every agent v computes s_v on $v.\text{level}_Q$, the maximum value of level_Q , i.e., s_{\max} , is propagated from agent to agent via *one-way epidemic* [AAE08], that is,

- each agent memorizes the largest value of level_Q it has observed, and
- the larger value is propagated to the agent with smaller value at every interaction.

It is proven in [AAE08] that all agents obtain the largest value within $O(\log n)$ parallel time with high probability by this simple propagation. If agent v knows $s_v < s_{\max}$, v changes $v.\text{leader}$ from *true* to *false*, that is, v becomes a follower. Thus, when one-way epidemic of s_{\max} finishes, only the agents v satisfying $s_v = s_{\max}$ are leaders. From the above discussion, for any $i \geq 2$, the number of such surviving leaders is exactly i with probability at most 2^{1-i} . On the other hand, there are at least one agent v with $s_v = s_{\max}$, thus this module never eliminates all leaders. A logarithmic number of states is sufficient for level_Q because each agent v gets more than $c \lg n$ consecutive heads with probability at most n^{-c} for any $c \geq 1$.

3.1.2 Key Idea for *Tournament()*

Starting from a configuration where the number of leaders is i , the goal of *Tournament()* is to reduce the number of leaders from i to one with probability $1 - O(i/\log n)$ while guaranteeing that not all leaders are eliminated. The idea of this component is simple. As with the *QuickElimination()*, we use coin flips in *Tournament()*. Every leader v maintains variable $v.\text{rand}$. Initially, $v.\text{rand} = 0$. Every time it has an interaction, it updates $v.\text{rand}$ by $v.\text{rand} \leftarrow 2v.\text{rand} + j$ where j indicates whether v is a responder in the interaction or not, i.e., $j = 0$ if v is an initiator and $j = 1$ if v is a responder. This operation stops when v encounters $\lceil \log_2 m \rceil = O(\log \log n)$ interactions. Thus, when all the i leaders encounter at least $\lceil \log_2 m \rceil$ interactions, for every leader v , $v.\text{rand}$ is a random variable uniformly chosen from $\{0, 1, \dots, 2^{\lceil \log_2 m \rceil} - 1\}$. Although $u.\text{rand}$ and $v.\text{rand}$ are not independent of each other for any distinct leader u and v , we will present a way to remove any dependence between $u.\text{rand}$ and $v.\text{rand}$ in Section 3.2. As with *QuickElimination()*, the maximum value rand is propagated to the whole population via one-way epidemic within $O(\log n)$ parallel time with high probability and only leaders with the maximum value remains leaders in the end of *Tournament()*.

Let v_1, v_2, \dots, v_i be the i leaders that survive *QuickElimination()*. Let r_1, r_2, \dots, r_i be the resulting value of $v_i.\text{rand}$ and define $r_{\max}(j) = \max(r_1, r_2, \dots, r_j)$ for any $j = 1, 2, \dots, i$. Clearly, the number of leaders at the end of *Tournament()* is exactly one if $r_{j+1} \neq r_{\max}(j)$ holds for all $j = 1, 2, \dots, i-1$. By the union bound and independence between r_1, r_2, \dots, r_i , this holds with probability at least $1 - \sum_{j=1}^{i-1} 2^{-\lceil \log_2 m \rceil} \geq 1 - i/m \geq 1 - i/(\lg n)$. On the other hand, an execution of *Tournament()* never eliminates all leaders since there always at least one leader v_j that satisfies $r_j = r_{\max}(i)$.

3.1.3 Key Idea for *BackUp()*

The goal of *BackUp()* is to elect a unique leader within $O(\log^2 n)$ parallel time in expectation. We must guarantee this expected time regardless of the number of the agents that survive both *QuickElimination()* and *Tournament()* and remain leaders at the beginning of an execution of *BackUp()*. We can only assume that at least one leader exists at the beginning of the execution. We use coin flips also for *BackUp()*. Every leader v maintains $v.\text{level}_B$. Initially, $v.\text{level}_B = 0$. Every leader v repeats the following procedure until $v.\text{level}_B$ reaches $5m$ or v becomes a follower.

- Make a coin flip. If the result is head (i.e., v participates in an interaction as an initiator), v increments $v.\text{level}_B$ by one. If the result is tail, v does nothing.
- Wait for sufficiently long but logarithmic parallel time so that the maximum level_B propagates to the whole population via one-way epidemic. If it observes larger value in the epidemic, it becomes a follower, that is, it executes $v.\text{leader} \leftarrow \text{false}$. Furthermore, if v interacts with another leader with the same level during this period and v is a responder in the interaction, v becomes a follower.

Let j be an arbitrary integer such that $1 \leq i \leq 5m$. Consider the first time that level_B of some leader, say v , reaches j . Let $V' \subseteq V$ be the set of leaders at that time. By the definition of the above procedure, every $u \in V'$ other than v satisfies $u.\text{level}_B < j$, and u makes a coin flip at most once with high probability until the maximum value j is propagated from v to u . If the result of the one coin flip is tail, u becomes a follower. Therefore, with probability at least $1/2 - O(n^{-1}) > 1/3$, no less than half of leaders in $V' \setminus v$ becomes followers, that is, the number of leaders decreases

Table 3: Variables of P_{LL}

Groups	Variables	Initial values
All agents	$\text{leader} \in \{\text{false}, \text{true}\}$	true
	$\text{tick} \in \{\text{false}, \text{true}\}$	false
	$\text{status} \in \{X, A, B\}$	X
	$\text{epoch} \in \{1, 2, 3, 4\}$	1
	$\text{init} \in \{1, 2, 3, 4\}$	1
	$\text{color} \in \{0, 1, 2\}$	0
V_B	$\text{count} \in \{0, 1, \dots, c_{\max} - 1\}$	Undefined
$V_A \cap V_1$	$\text{level}_Q \in \{0, 1, \dots, l_{\max}\}$	Undefined
	$\text{done} \in \{\text{false}, \text{true}\}$	Undefined
$V_A \cap (V_2 \cup V_3)$	$\text{rand} \in \{0, 1, \dots, 2^\Phi - 1\}$	Undefined
	$\text{index} \in \{0, 1, \dots, \Phi - 1\}$	Undefined
$V_A \cap V_4$	$\text{level}_B \in \{0, 1, \dots, l_{\max}\}$	Undefined

to at most $1 + \lfloor |V'|/2 \rfloor$. Chernoff bound guarantees that the number of leaders becomes one with high probability until $v.\text{level}_B$ for every leader v reaches $5m$. Even if multiple leaders survive at that time, we have simple election mechanism to elect a unique leader; when two leaders with the same level interacts with each other, one of them becomes a follower. This simple election mechanism elects a unique leader within $O(n)$ parallel time in expectation. Therefore, the total expected parallel time to elect a unique leader is $O(m \log n) + O(n^{-1}) \cdot O(n) = O(\log^2 n)$.

3.2 Detailed Description

In this subsection, we present detailed description of the proposed protocol P_{LL} . The key ideas presented in the previous subsection achieve $O(\log n)$ stabilization time if it is implemented correctly. However, they need some kind of global synchronization. Furthermore, a naive implementation of the key ideas requires a poly-logarithmic number of states (*i.e.*, $O(\log^c n)$ states for $c > 1$) per agent while our goal is to achieve $O(\log n)$ states per agent. In this subsection, we will give how we achieve synchronization and implement the ideas shown in Section 3.1 with only $O(\log n)$ states per agent.

All variables of P_{LL} are listed in Table 3. All agents manage six variables `leader`, `tick`, `status`, `epoch`, `init`, and `color`. To implement the key ideas above with $O(\log n)$ states, we divide the population into multiple sub-populations or *groups*, as in [GSU18], where agents in different groups manage different variables in addition to the above six variables. In the remainder of this paper, we refer the above six variables by *common variables* and other variables by *additional variables*. The population is divided to six groups based on two common variables $\text{status} \in \{X, A, B\}$ and $\text{epoch} \in \{1, 2, 3, 4\}$, that is, V_X , V_B , $V_A \cap V_1$, $V_A \cap (V_2 \cup V_3)$, $V_A \cap V_4$ where we denote $V_Z = \{v \in V \mid v.\text{status} = Z\}$ for $Z \in \{X, A, B\}$ and $V_i = \{v \in V \mid v.\text{epoch} = i\}$ for $i \in \{1, 2, 3, 4\}$. We have no additional variables for agents in group V_X , one additional variable $\text{count} \in \{0, 1, \dots, c_{\max} - 1\}$ for agents in V_B where $c_{\max} = 41m$, two additional variables $\text{level}_Q \in \{0, 1, \dots, l_{\max}\}$ and $\text{done} \in \{\text{false}, \text{true}\}$ for agents in $V_A \cap V_1$ where $l_{\max} = 5m$, two additional variables $\text{rand} \in \{0, 1, \dots, 2^\Phi - 1\}$ and $\text{index} \in \{0, 1, \dots, \Phi - 1\}$ for agents in $V_A \cap (V_2 \cup V_3)$ where $\Phi = \lceil \frac{2}{3} \lg m \rceil$, and one additional variable $\text{level}_B \in \{0, 1, \dots, l_{\max}\}$ for agents in $V_A \cap V_4$. Agents in any group have only $O(\log n)$ states. This is because every common variable has constant size domain, every group other than $V_A \cap (V_2 \cup V_3)$ has at most one non-constant additional variable and any of such variables can take $O(\log n)$ values, and an agent in $V_A \cap (V_2 \cup V_3)$ has two additional variables `rand` and `index` and the combination of the two variables can take $2^\Phi \cdot \Phi = O(m^{2/3} \log m) \subset O(\log n)$ values. Therefore, the number of states per agent used by P_{LL} is $O(\log n)$.

Lemma 3. *The number of states per agent used by P_{LL} is $O(\log n)$.*

Independently of the six groups defined above, we define another groups V_L and V_F based on a common variable `leader`; V_L (resp., V_F) is the set of agents $v \in V$ such that $v.\text{leader} = \text{true}$ (resp., $v.\text{leader} = \text{false}$). We introduce these two groups only for simplicity of notation.

Algorithm 1: P_{LL}

Notations:

$l_{\max} = 5m, c_{\max} = 41m, \Phi = \lceil \frac{2}{3} \lg m \rceil$
 $V_Z = \{v \in V \mid v.\text{status} = Z\}$ for $Z \in \{X, A, B\}$
 $V_i = \{v \in V \mid v.\text{epoch} = i\}$ for $i \in \{1, \dots, 4\}$

Output function π_{out} :

if $v.\text{leader} = \text{true}$ holds, then the output of agent v is L , otherwise F .

Interaction between initiator a_0 and responder a_1 :

```
1: if  $a_0, a_1 \in V_X$  then
2:    $(a_0.\text{status}, a_0.\text{level}_Q, a_0.\text{done}, a_0.\text{leader}) \leftarrow (A, 0, \text{false}, \text{true})$ 
3:    $(a_1.\text{status}, a_1.\text{count}, a_1.\text{leader}) \leftarrow (B, 0, \text{false})$ 
4: else if  $\exists i \in \{0, 1\} : a_i \in V_X \wedge a_{1-i} \notin V_X$  then
5:    $(a_i.\text{status}, a_i.\text{level}_Q, a_i.\text{done}, a_i.\text{leader}) \leftarrow (A, 0, \text{true}, \text{false})$ 
6: end if

7:  $a_0.\text{tick} \leftarrow a_1.\text{tick} \leftarrow \text{false}$ 
8:  $\text{CountUp}()$ 
9: for all  $i \in \{0, 1\}$  such that  $a_i.\text{tick}$  do  $a_i.\text{epoch} = \max(a_i.\text{epoch} + 1, 4)$  endfor
10:  $a_0.\text{epoch} \leftarrow a_1.\text{epoch} \leftarrow \max(a_0.\text{epoch}, a_1.\text{epoch})$ 

11: for all  $i \in \{0, 1\}$  such that  $a_i.\text{epoch} > a_i.\text{init}$  do // Initialize variables for each group
12:   if  $a_i \in V_A \cap (V_2 \cup V_3)$  then  $(a_i.\text{rand}, a_i.\text{index}) \leftarrow (0, 0)$  endif
13:   if  $a_i \in V_A \cap V_4$  then  $a_i.\text{level}_B \leftarrow 0$  endif
14:    $a_i.\text{init} \leftarrow a_i.\text{epoch}$ 
15: end for

16: if  $a_0, a_1 \in V_1$  then
17:   Execute  $\text{QuickElimination}()$ 
18: else if  $a_0, a_1 \in V_2 \vee a_0, a_1 \in V_3$  then
19:   Execute  $\text{Tournament}()$ 
20: else if  $a_0, a_1 \in V_4$  then
21:   Execute  $\text{BackUp}()$ 
22: end if
```

The pseudo code of P_{LL} is given in Algorithm 1 and its modules $\text{CountUp}()$, $\text{QuickElimination}()$, $\text{Tournament}()$, and $\text{BackUp}()$ are presented in Algorithm 2, 3, 4, and 5, respectively. The main function of P_{LL} (Algorithm 1) consists of four parts. The first part (Lines 1-6) assigns status A or B to each agent. The second part (Lines 7-10) manages variable epoch using module $\text{CountUp}()$. Initially, $v.\text{epoch} = 1$ holds, that is, $v \in V_1$ holds for all $v \in V$. In an execution of P_{LL} , $v.\text{epoch}$ never decreases and increases by one every sufficiently large logarithmic parallel time in expectation until it reaches 4 as we will explain later. In the third part (Lines 11-15), we initialize additional variables when an agent increases its epoch. Each agent v has a common variable init , which is set to 1 initially. Whenever $v.\text{epoch}$ increases, $v.\text{epoch} > v.\text{init}$ must hold, then v initialize additional variables according to v 's group and executes $v.\text{init} \leftarrow v.\text{epoch}$. For example, when the epoch of agent $v \in V_A$ changes from 3 to 4 i.e., v moves from group $V_A \cap V_3$ to $V_A \cap V_4$, it initializes an additional variable $a_i.\text{level}_B$ to 0 (Line 13). Additional variables for groups V_B and $V_A \cap V_1$ are initialized not in this part but in the first part as we will explain in Section 3.2.1. In the fourth part (Lines 16-22), agents execute modules based on the values of their epoch. Specifically, agents execute $\text{QuickElimination}()$, $\text{Tournament}()$, and $\text{BackUp}()$ while they are in V_1 , $V_2 \cup V_3$, and V_4 respectively.

In the remainder of this subsection, we explain how P_{LL} assigns status to agents, P_{LL} synchronizes the population by $\text{CountUp}()$, and the implementation of the three modules $\text{QuickElimination}()$, $\text{Tournament}()$, and $\text{BackUp}()$.

Algorithm 2: *CountUp()*

Interaction between initiator a_0 and responder a_1 :

```
23: for all  $i \in \{0, 1\}$  such that  $a_i \in V_B$  do
24:    $a_i.\text{count} \leftarrow a_i.\text{count} + 1 \pmod{c_{\max}}$ 
25:   if  $a_i.\text{count} = 0$  then
26:      $a_i.\text{color} \leftarrow a_i.\text{color} + 1 \pmod{3}$ 
27:      $a_i.\text{tick} \leftarrow \text{true}$ 
28:   end if
29: end for
30: if  $\exists i \in \{0, 1\} : a_{1-i}.\text{color} = a_i.\text{color} + 1 \pmod{3}$  then
31:    $a_i.\text{color} \leftarrow a_{1-i}.\text{color}$ 
32:    $a_i.\text{tick} \leftarrow \text{true}$ 
33:   if  $a_i \in V_B$  then  $a_i.\text{count} \leftarrow 0$  endif
34: end if
```

3.2.1 Assignment of Status

At the beginning of an execution, all agents are in V_X , that is, the statuses of all agents are the “initial” status X . Every agent is given status A or B at its first interaction where A means “leader candidate” and B means “timer agent”. As we will explain later, the unique leader is elected from V_A and agents in V_B are mainly used to synchronize the population with their count-up timers.

Agents determine their status, A or B , by the following simple way. When two agents in V_X meet, the initiator and the responder are given status A and B , respectively (Line 2-3). The initiator initializes its additional variable `levelQ` and `done` to 0 and *false* respectively and remains a leader (Line 2) while the responder initializes its additional variable `count` to 0 and becomes a follower by `leader` \leftarrow *false* (Line 3). When an agent in V_X meets an agent in V_A or V_B , it gets status A but it becomes a follower. It also initialize its additional variable `levelQ` and `done` to 0 and *true* respectively (Line 5). For agent v , assigning *true* to $v.\text{done}$ means that v never joins a game with coin flips in *QuickElimination()*.

No agent changes its status once it gets status A or B , and no follower becomes a leader in an execution of P_{LL} . Therefore, we have the following lemma.

Lemma 4. *In an execution of P_{LL} , $|V_A| \geq n/2$, $|V_F| \geq n/2$, and $|V_B| \geq 1$ always hold after every agent gets status A or B .*

Proof. Consider any configuration in $\mathcal{C}_{\text{all}}(P_{LL})$ where every agent has status A or B . Let x (resp., y and z) be the number of agents which get status A (resp., B and A) by Line 2 (resp., Line 3 and Line 5). We have $x = y \leq n/2$ by the definition of P_{LL} , which gives $|V_A| = x + z = n - y \geq n/2$. Moreover, $|V_L| \leq x \leq n/2$ holds because the number of leaders is monotonically non-increasing in an execution of P_{LL} . The first interaction of the execution assigns one agent with status V_B , hence $|V_B| \geq 1$ holds. \square

3.2.2 Synchronization and Epochs

When a unique leader exists in the population, we can synchronize the population by Phase clocks with constant space per agent [AAE08]. Recently, in [GS18] and [GSU18], it is proven that even when we cannot assume the existence of the unique leader, Phase clocks can be used for synchronization if we are allowed to use $O(\log \log n)$ states per agent. Since we use $O(\log n)$ states for another modules, we achieve synchronization in simpler way with $O(\log n)$ states per agent.

For synchronization, we use common variables `color` $\in \{0, 1, 2\}$ in all agents and an additional variable `count` $\in \{0, 1, \dots, c_{\max} - 1\}$ for agents in group V_B . Initially, all agents have the same color, namely, 0. The color of an agent is incremented by modulo 3 when the agent changes its color. We say that the agent *gets a new color* when this event happens. Roughly speaking, our goal is to guarantee that

- (i) whenever one agent gets a new color (e.g., changes its color from 0 to 1), the new color spreads to the whole population within $O(\log n)$ parallel time with high probability,
- (ii) thereafter, all agents keep the same color for sufficiently long but $\Theta(\log n)$ parallel time with high probability.

Specifically, “sufficiently long but $\Theta(\log n)$ time” in (ii) means sufficiently long period such that any $O(\log n)$ parallel time operations in *QuickElimination()*, *Tournament()*, and *BackUp()*, such as one-way epidemic of some value, finishes with high probability during the period.

At every interaction, module *CountUp()* is invoked (Line 8) and variables `color` and `count` can be changed only in this module. In *CountUp()*, every agent in V_B increments its `count` by one modulo c_{\max} (Line 24). For every $v \in V_B$, if this incrementation changes $v.\text{count}$ from $c_{\max} - 1$ to 0, v gets a new color by incrementing $v.\text{color}$ by one modulo 3 (Line 26). Once one agent gets a new color, the new color spreads to the whole population via one-way epidemic in the whole population. Specifically, if agents u and v satisfying $u.\text{color} = v.\text{color} + 1 \pmod{3}$ meets, v execute $v.\text{color} \leftarrow u.\text{color}$ and resets its count to 0 (Line 31-33).

Every time an agent v gets a new color, it raise a tick flag, i.e., assigns $v.\text{tick} \leftarrow \text{true}$ (Lines 27 and 32). This common variable $v.\text{tick}$ is used only for simplicity of the pseudo code and it does not affect the transition at v 's next interaction ($v.\text{tick}$ is reset to *false* in Line 7), unlike any other variable. When $v.\text{tick}$ is raised, $v.\text{epoch}$ increases by one unless it has already reached 4 (Line 9). After two agents u and v execute Lines 7-9 at an interaction, $u.\text{epoch} = v.\text{epoch}$ usually holds. However, this equation does not hold when synchronization fails. For this case, we substitute $\max(u.\text{epoch}, v.\text{epoch})$ into $u.\text{epoch}$ and $v.\text{epoch}$ in Line 10.

As mentioned above, every agent gets a new color in every sufficiently large $\Theta(\log n)$ parallel time with high probability. This means that, for every $v \in V$, $v.\text{tick}$ is raised and $v.\text{epoch}$ increases by one with high probability in every sufficiently large $\Theta(\log n)$ parallel time until $v.\text{epoch}$ reaches 4. If this synchronization fails, e.g., some agent gets a color 1 without keeping color 0 for $\Theta(\log n)$ parallel time, the modules *QuickElimination()* and *Tournament()* may not work correctly. However, starting from any configuration after a synchronization fails arbitrarily, module *CountUp()* and Lines 7-10 guarantees that all agents proceeds to the forth epoch within $O(\log n)$ parallel time in expectation, and thereafter *BackUp()* guarantees that exactly one leader is elected within $O(n)$ parallel time in expectation. Hence, P_{LL} guarantees that a unique leader is elected with probability 1. The above $O(n)$ parallel time never prevent us from achieving stabilization time of $O(\log n)$ parallel time in expectation because synchronization fails with probability at most $O(\log n/n)$ as we will see later.

Definition 1. For any $i = 0, 1, 2$, we define $\mathcal{C}_{\text{color}}(i)$ as the set of all configurations in $\mathcal{C}_{\text{all}}(P_{LL})$ where every agent has color i .

Definition 2. For any $i = 0, 1, 2$, we define $\mathcal{C}_{\text{start}}(i)$ as the set of all configurations in $\mathcal{C}_{\text{all}}(P_{LL})$ each of which satisfies all of the following conditions:

- some agent has color i ,
- $v.\text{count} = 0$ holds for all $v \in V_B$ such that $v.\text{color} = i$, and
- no agent has color $i + 1 \pmod{3}$.

Lemma 5. In an execution of $\Xi(C_{\text{init}, P_{LL}}, \Gamma)$, each agent in V_B always gets a new color within $O(\log n)$ parallel time with high probability.

Proof. Simple Chernoff bound gives the lemma because any agent has an interaction with probability $2/n$ at each step and each agent in V_B gets a new color before it has c_{\max} interactions. \square

The goal of our synchronization, (i) and (ii), are formalized as follows.

Lemma 6. Let $i \in \{0, 1, 2\}$, $C_0 \in \mathcal{C}_{\text{start}}(i)$, and $\Xi_{P_{LL}}(C_0, \Gamma) = C_0, C_1, \dots$. Then, all of the following propositions hold.

- P_1 : No agent gets color $i + 1 \pmod{3}$ by the first $\lfloor 21n \ln n \rfloor$ steps in $\Xi_{P_{LL}}(C, \Gamma)$ with high probability.

Algorithm 3: *QuickElimination()*

Interaction between initiator a_0 and responder a_1 :

```
35: if  $\exists i \in \{0, 1\}$  such that  $a_i \in V_L \wedge a_{1-i} \in V_F \wedge \neg a_i.\text{done}$  then  
36:   if  $i = 0$  then  $a_0.\text{level}_Q \leftarrow \max(a_0.\text{level}_Q + 1, l_{\max})$  endif  
37:   if  $i = 1$  then  $a_1.\text{done} \leftarrow \text{true}$  endif  
38: end if  
39: if  $a_0, a_1 \in V_A \wedge a_0.\text{done} \wedge a_1.\text{done} \wedge \exists i \in \{0, 1\} : a_i.\text{level}_Q < a_{1-i}.\text{level}_Q$  then  
40:    $a_i.\text{leader} \leftarrow \text{false}$   
41:    $a_i.\text{level}_Q \leftarrow a_{1-i}.\text{level}_Q$   
42: end if
```

- P_2 : Execution $\Xi_{P_{LL}}(C_0, \Gamma)$ reaches a configuration in $\mathcal{C}_{\text{color}}(i)$ by the first $\lfloor 4n \ln n \rfloor$ steps with high probability.
- P_3 : Execution $\Xi_{P_{LL}}(C_0, \Gamma)$ reaches a configuration in $\mathcal{C}_{\text{start}}(i + 1 \pmod{3})$ within $O(\log n)$ parallel time with high probability.

Proof. Propositions P_2 and P_3 immediately follows from Lemma 2 with $n' = n$ and Lemma 5, respectively. In the following, we prove proposition P_1 . Starting from a configuration $C_0 \in \mathcal{C}_{\text{start}}(i)$, no agent gets color $i + 1 \pmod{3}$ until some agent in V_B participates in no less than c_{\max} interactions. For any agent v , v participates in an interaction with probability $2/n$ at every step. Therefore, letting X be a binomial random variable such that $X \sim B(\lfloor 21n \ln n \rfloor, 2/n)$, v participates in no less than c_{\max} interactions with probability $\Pr(X \geq c_{\max})$, which is bounded as follows.

$$\begin{aligned} \Pr(X \geq c_{\max}) &= \Pr\left(X \geq \frac{c_{\max}}{42 \ln n} \mathbf{E}[X]\right) \\ &\leq \Pr\left(X \geq \frac{58}{42} \mathbf{E}[X]\right) \\ &\leq \exp\left(-\frac{(58 - 42)^2}{42^2 \cdot 3} \mathbf{E}[X]\right) \\ &\leq \exp(-2 \ln n + 0.05) \\ &= O(n^{-2}). \end{aligned}$$

where we use $c_{\max} \geq 41 \lg n \geq 58 \ln n$ for the second inequality and Chernoff Bound in the form of (1) in Lemma 1 for the third inequality. Thus, the union bound gives that no agent gets color $i + 1 \pmod{3}$ by the first $\lfloor 21n \ln n \rfloor$ interactions in $\Xi_{P_{LL}}(C, \Gamma)$ with probability $1 - O(n^{-1})$. \square

3.2.3 *QuickElimination()*

The module *QuickElimination()* uses additional variables $\text{level}_Q \in \{0, 1, \dots, l_{\max} - 1\}$ and $\text{done} \in \{\text{false}, \text{true}\}$ of group $V_A \cap V_1$. Each agent v executes this module only when $v.\text{epoch} = 1$ holds. As mentioned in section 3.2.1, when an agent v is assigned with status V_A , it holds that v is a leader and $v.\text{done} = \text{false}$ or v is a follower and $v.\text{done} = \text{true}$.

In an execution of module *QuickElimination()*, each leader $v \in V_A$ makes fair coin flips repeatedly until it sees “tail” for the first time and stores on $v.\text{level}_Q$ the number of times it observes “heads”. Specifically, a leader with $v.\text{done} = \text{false}$ makes a fair coin flip every time it interacts with a follower (*i.e.*, an agent in V_F). If the result is head (*i.e.*, v is an initiator at the interaction), it increments level_Q by one (Line 36). Otherwise, it stops coin flipping by assigning $v.\text{done} \leftarrow \text{true}$ (Line 37). The largest level_Q among all agents in V_L spreads to the whole sub-population V_A via one-way epidemic. Specifically, when two stopped agents $u, v \in V_A$ meet, they update their level_Q to $\max(u.\text{level}_Q, v.\text{level}_Q)$ (Line 41). When an agent $v \in V_A$ meets an agent with larger level_Q than $v.\text{level}_Q$, it becomes a follower (Line 40). The correctness of *QuickElimination()* is formalized as the following lemma.

Algorithm 4: *Tournament()*

Interaction between initiator a_0 and responder a_1 :

```
43: if  $i \in \{0, 1\} : a_i \in V_L \wedge a_{1-i} \in V_F \wedge a_i.\text{index} < \Phi$  then  
44:    $a_i.\text{rand} \leftarrow 2a_i.\text{rand} + i$   
45:    $a_i.\text{index} \leftarrow \max(a_i.\text{index} + 1, \Phi)$   
46: end if  
47: if  $a_0, a_1 \in V_A \wedge a_0.\text{index} = a_1.\text{index} = \Phi \wedge \exists i \in \{0, 1\} : a_i.\text{rand} < a_{1-i}.\text{rand}$  then  
48:    $a_i.\text{leader} \leftarrow \text{false}$   
49:    $a_i.\text{rand} \leftarrow a_{1-i}.\text{rand}$   
50: end if
```

Lemma 7. Let $\Xi = \Xi_{P_{LL}}(C_{\text{init}, P_{LL}}, \Gamma) = C_0, C_1, \dots$. In a configuration $C_{\lfloor 21n \ln n \rfloor}$, $\Pr(|V_L| = i) < 2^{1-i} + \epsilon_i$ holds for any $i = 2, 3, \dots, n$ where ϵ_i is a non-negative number such that $\sum_{i=2}^n \epsilon_i = O(n^{-1})$.

Proof. Coin flips in *QuickElimination()* are not only fair but also independent of each other. This is because we assume the uniformly random scheduler Γ and at most one agent makes a coin flip at each step (*i.e.*, at each interaction) since a coin flip is made only when a leader and a follower meet. Therefore, an execution of this module correctly simulates the competition game introduced in Section 3.1.1 and the simulation of the game finishes within the first $\lfloor 21n \ln n \rfloor$ interactions if all of the following conditions hold in $C_{\lfloor 21n \ln n \rfloor}$:

- every agent v is still in the first epoch, *i.e.*, $v.\text{epoch} = 1$ holds,
- $v.\text{level}_Q < l_{\max}$ holds for all $v \in V_A$
- all agents in V_A has the same level_Q and $v.\text{done} = \text{true}$ holds for all $v \in V_A$.

Intuitively, the second condition guarantees that no agent increases level_Q to the upper limit l_{\max} within the first $\lfloor 21n \ln n \rfloor$ interactions and the third condition means that every leader finishes coin flips and the maximum value of level_Q propagates to the whole sub-population V_A within the first $\lfloor 21n \ln n \rfloor$ interactions. The second condition $\forall v \in V_A : v.\text{level}_Q < l_{\max}$ is necessary because if some agent in V_A increases level_Q to l_{\max} , then it may fail to simulate the competition game successfully.

Note that the competition game guarantees that exactly i agents survives the game with probability at most 2^{1-i} . Therefore, it suffices to prove that all the three conditions hold with high probability, *i.e.*, with probability $1 - O(n^{-1})$. Since $C_{\text{init}, P_{LL}} \in \mathcal{C}_{\text{start}}(0)$ holds, it directly follows from Lemma 6 that the first condition holds with high probability. The second condition holds with high probability because the second condition does not hold only when some leader gets head l_{\max} times in a row and the probability that such an event happens is at most $n(1/2)^{l_{\max}} \leq n \cdot 2^{-5 \lg n} = O(n^{-1})$.

In what follows, we prove that the third condition holds with high probability. In the similar way to analyze the probability of the second condition, we can easily prove that no leader gets head $2 \lg n$ times in a row with high probability. Furthermore, at each step, any leader meets a follower with probability at least $|V_F|/n C_2 \geq 1/n$ by Lemma 4, hence it holds with probability $1 - n \cdot n^{-2} = 1 - O(n^{-1})$ by Chernoff bound in the form of (2) in Lemma 1 that every leader meets a follower no less than $2 \lg n$ times during the first $\lfloor 9n \ln n \rfloor$ ($\geq \lceil 6n \lg n \rceil$) interactions. Therefore, with high probability, all agents in V_A finish making coin flips within the first $\lfloor 9n \ln n \rfloor$ interactions. Thereafter, the maximum value of level_Q is propagated to the whole sub-population V_A by one-way epidemic in V_A . The epidemic finishes within the next $\lfloor 8n \ln n \rfloor$ interactions with high probability by Lemma 2. Since $\lfloor 9n \ln n \rfloor + \lfloor 8n \ln n \rfloor < \lfloor 21n \ln n \rfloor$, the third condition also holds with high probability. \square

Note that an execution of *QuickElimination()* never eliminates all leaders from population because a leader v with $v.\text{level}_Q = \max_{u \in V_A} u.\text{level}_Q$ never becomes a follower.

Algorithm 5: *BackUp()*

Interaction between initiator a_0 and responder a_1 :

```
51: if  $a_0.\text{tick} \wedge a_0 \in V_L \wedge a_1 \in V_F$  then  
52:    $a_0.\text{level}_B \leftarrow \max(a_0.\text{level}_B + 1, l_{\max})$   
53: end if  
54: if  $a_0, a_1 \in V_A \wedge \exists i \in \{0, 1\} : a_i.\text{level}_B < a_{1-i}.\text{level}_B$  then  
55:    $a_i.\text{level}_B \leftarrow a_{1-i}.\text{level}_B$   
56:    $a_i.\text{leader} \leftarrow \text{false}$   
57: end if  
58: if  $\forall i \in \{0, 1\} : a_i \in V_L$  then  $a_1.\text{leader} \leftarrow \text{false}$  endif
```

3.2.4 *Tournament()*

In the key idea depicted in Section 3.1.2, each leader v makes fair coin flips exactly $\lceil \lg m \rceil = \Theta(\log \log n)$ times. However, this requires $\Omega(\log n \cdot \log \log n)$ states per agent because this procedure requires not only variable $v.\text{rand}$ that stores the results of those flips but also variable $v.\text{index}$ to memorize how many times v already made coin flips. Therefore, in an execution of *Tournament()*, each agent makes fair coin flips only $\Phi = \lceil \frac{2}{3} \lg m \rceil$ times, and we execute this module *Tournament()* twice. That is why we assign two epochs (*i.e.*, the second and the third epochs) to *Tournament()*.

In an execution of *Tournament()*, each leader v gets a random number, say *nonce*, uniformly at random from $\{0, 1, \dots, 2^\Phi - 1\}$ by making coin flips Φ times, and stores it in $v.\text{rand}$ (Line 43-46). The uniform randomness of this nonce is guaranteed because these coin flips are not only fair but also independent of each other, as mentioned in Section 3.2.3. Leaders who finishes generating a nonce begins one-way epidemics of the largest value of these nonces (Lines 47-50). By Chernoff bound, it holds with high probability that all leaders finishes generating its nonce within $O(\log n)$ parallel time and the largest value of these nonces propagates to the whole sub-populations V_A within $O(\log n)$ parallel time. Note that an execution of *Tournament()* never eliminates all leaders from population because a leader with the largest nonce never becomes a follower.

Lemma 8. *In an execution $\Xi = \Xi_{PLL}(C_{\text{init}, PLL}, \Gamma) = C_0, C_1, \dots$, the number of leaders become exactly one before some agent enters the fourth epoch (*i.e.*, epoch = 4) with probability $1 - O(1/\log n)$.*

Proof. By Lemmas 6 and 7, there exist at most $\lceil \lg \lg n \rceil$ leaders and all agents are still in the first epoch in configuration $C_{\lfloor 21n \ln n \rfloor}$ with probability $1 - \left(\sum_{i=\lceil \lg \lg n \rceil+1}^n 2^{1-i} \right) - O(n^{-1}) = 1 - O(1/\log n)$. Thereafter, execution Ξ reaches a configuration in $C_{\text{start}}(1)$ within the next $O(n \log n)$ interactions with high probability by Lemmas 4 and 5.

Therefore, in order to prove the lemma, we can assume that there exists an integer $t' = O(n \log n)$ such that there exists at most $\lceil \lg \lg n \rceil$ leaders in $C_{t'}$, every agent is in the first or the second epoch in $C_{t'}$, and $C_{t'} \in C_{\text{start}}(1)$ holds. We say that an execution of module *Tournament()* finishes completely if every leader finishes generating a nonce and the maximum value of nonces is propagated to the whole sub-population V_A . Since a leader generates a nonce uniformly at random among $\{0, 1, \dots, 2^\Phi - 1\}$ in each of the two executions of *Tournament()*, the same arguments in section 3.1.2 yields that exactly one leader exists with probability at least $1 - (\lceil \lg \lg n \rceil - 1) \cdot 2^{-\Phi} \geq 1 - O(\log \log n / \log^{2/3} n)$ after one execution of module *Tournament()* finishes completely.

Each leader generates a nonce by meeting a follower $\Phi = O(\log m) = O(\log \log n)$ times while any leader meets a follower with probability at least $\frac{1}{n}$ at each step by Lemma 4. Therefore, by Chernoff bound and Lemma 2, an execution of *Tournament()* finishes completely within $\lfloor 21n \ln n \rfloor - \lfloor 4n \ln n \rfloor \geq \lfloor 17n \ln n \rfloor$ interactions with high probability for sufficiently large n . Hence, by Lemma 6, both the first and the second executions of *Tournament()* finish completely in the next $O(n \log n)$ steps with high probability. Therefore, by Lemma 6, the two executions of *Tournament* decreases the number of leaders from at most $\lceil \lg \lg n \rceil$ to exactly one before some agent enters the fourth epoch with probability $1 - O((\log \log n / \log^{2/3} n)^2) = 1 - O(1/\log n)$. \square

3.2.5 BackUp()

This module *BackUp()* uses only one additional variable level_B to elect the unique leader. For any $v \in V_A$, variables $v.\text{level}_B$ is initialized by $v.\text{level}_B \leftarrow 0$ at the first time $v.\text{epoch} = 4$ holds (Line 9).

As long as synchronization succeeds, each $v.\text{tick}$ is raised every $\Theta(\log n)$ (but sufficiently long) parallel time. In an execution of *BackUp()*, each leader has a chance of making a coin flip every time its tick is raised. Specifically, a leader v makes a coin flip when v has an interaction with a follower and $v.\text{tick}$ is raised at that interaction. If v sees “head” (i.e., it is an initiator at that interaction), it increments $v.\text{level}_B$ by one unless $v.\text{level}_B$ already reaches l_{\max} (Lines 51-53). The largest value of level_B is propagated via one-way epidemic in sub-population V_A (Lines 54-57). If a leader v observes the larger value of level_B than $v.\text{level}_B$, it becomes a follower (Line 56). Furthermore, this module includes simple leader election [Ang+06]; when two leaders interact and they observe that they have the same value of level_B at Line 58, then the responder becomes a follower. Note that an execution of *BackUp()* never eliminates all leaders from population because a leader with the largest value of level_B never becomes a follower.

Lemma 9. *Let C be any configuration in $\mathcal{C}_{\text{all}}(P_{LL})$ and let $\Xi = \Xi_{P_{LL}}(C, \Gamma) = C_0, C_1, \dots$. Execution Ξ reaches a configuration where all agents are in the fourth epoch within $O(\log n)$ parallel time with high probability and in expectation.*

Proof. After the first step of Ξ , at least one agent v in V_B always exists. By Lemma 5, v enters the forth epoch within $O(\log n)$ parallel time with high probability and in expectation. Since the largest value of epoch is propagated to the whole population via one-way epidemic, all agents enters the fourth epoch within $O(\log n)$ parallel time with high probability and in expectation by Lemma 2. \square

Lemma 10. *Let C be any configuration where all agents are in the fourth epoch and let $\Xi = \Xi_{P_{LL}}(C, \Gamma) = C_0, C_1, \dots$. Then Ξ reaches a configuration where there exists exactly one leader within $O(n)$ parallel time in expectation.*

Proof. Execution Ξ elects the unique leader within $O(n)$ parallel time in expectation because module *BackUp()* includes the simple leader election mechanism [Ang+06], i.e., one leader becomes a follower when two leaders meet. \square

Definition 3. We define $\mathcal{B}_{\text{start}}$ as the set of all configurations in $\mathcal{C}_{\text{color}}(0)$ where every agent is in the fourth epoch (i.e., epoch = 4), and $v.\text{level}_B \leq 1$ holds for all agents $v \in V_A$.

Lemma 11. *Execution $\Xi = \Xi_{P_{LL}}(C_{\text{init}, P_{LL}}, \Gamma)$ reaches a configuration in $\mathcal{B}_{\text{start}}$ within $O(\log n)$ parallel time with high probability.*

Proof. This lemma directly follows from Lemma 6 and the definition of *BackUp()*. \square

Lemma 12. *Let C be any configuration in $\mathcal{B}_{\text{start}}$ and let $\Xi = \Xi_{P_{LL}}(C, \Gamma) = C_0, C_1, \dots$. Then Ξ reaches a configuration where there exists exactly one leader within $O(\log^2 n)$ parallel time in expectation*

Proof. By applying Lemma 6 repeatedly, it holds for sufficiently large $O(\log^2 n)$ parallel time with probability $1 - O(\log n/n)$ that synchronization does not fail and no agent raises tick twice within any $[21n \ln n] - [4n \ln n] \geq [17 \ln n]$ steps. Thus, in the following, we assume that no agent raises tick twice within any $[17 \ln n]$ steps.

Define $B = \max_{v \in V_A \cap V_4} v.\text{level}_B$. By Lemma 6, each leader raises its tick in every $O(\log n)$ parallel time with high probability and makes a coin flip if it meets a follower at that interaction. Thus, each leader increments its level_B with probability $1/4$ in every $O(\log n)$ parallel time because $|V_F| \geq n/2$ by Lemma 4. Therefore, the maximum value B is increased by one within $O(\log n)$ parallel time in expectation, thus B reaches l_{\max} within $O(\log^2 n)$ parallel time in expectation.

Consider that now B is increased from k to $k + 1$. At this time, only one leader has the largest value $B = k + 1$ in level_B . Thereafter, this value $k + 1$ is propagated to the whole sub-population V_A within $[8n \ln n]$ steps with high probability by Lemma 2, during which no leader makes coin flips twice by the above assumption. Therefore, the number of leaders decreases almost by half, specifically decreases from $1 + i$ to at most $1 + \lfloor i/2 \rfloor$, with probability $1/2 - O(1/n)$. Clearly, in execution Ξ , B is eventually increased from 0 or 1 to l_{\max} . Therefore, ignoring the

probability that synchronization fails or the one-way epidemic does not finish within $\lfloor 8n \ln n \rfloor$ steps, we can observe by Chernoff bound that the number of leaders becomes one before B reaches l_{\max} with probability $1 - O(\log n/n)$.

Even if B reaches l_{\max} before one leader is elected, Ξ elects the unique leader within $O(n)$ parallel time in expectation thereafter by Lemma 10. Therefore, Ξ reaches a configuration where exactly one leader exists within $O(\log^2 n) + O(\log n/n) \cdot O(n) = O(\log^2 n)$ in expectation. \square

Theorem 1. *Let $\Xi = \Xi_{P_{LL}}(C_{\text{init}, P_{LL}}, \Gamma) = C_0, C_1, \dots$. Execution Ξ reaches a configuration where exactly one leader exists within $O(\log n)$ parallel time in expectation.*

Proof. First, Lemmas 8 and 9, execution Ξ reaches a configuration where exactly one leader exists within $O(\log n)$ parallel time with probability $1 - O(1/\log n)$. Second, by Lemma 11, execution Ξ reaches a configuration in $\mathcal{B}_{\text{start}}$ within $O(\log n)$ parallel time with high probability. Thereafter, execution Ξ reaches a configuration where exactly one leader exists within $O(\log^2 n)$ parallel time in expectation by Lemma 12. Finally, Lemmas 9 and 10 shows that starting from any configuration in $\mathcal{C}_{\text{all}}(P_{LL})$, Ξ reaches a configuration where exactly one leader is elected within $O(n)$ parallel time in expectation. To conclude, starting from initial configuration $C_{\text{init}, P_{LL}}$, execution Ξ reaches a configuration where the unique leader is elected within $O(\log n) + O(1/\log n) \cdot O(\log^2 n) + O(1/n) \cdot O(n) = O(\log n)$ parallel time in expectation. \square

4 Discussion towards Symmetric Transitions

In the field of PP model, several works are devoted to design a *symmetric protocol*. Suppose that two agents have an interaction and their states changes from p, q to p', q' , respectively. A protocol is symmetric if $p = q \Rightarrow p' = q'$ always hold. In other words, a symmetric protocol is a protocol that does not utilize the roles of the two agents at an interaction, initiator and responder. This property is important for some applications such as chemical reaction networks.

The proposed protocol P_{LL} described above is not symmetric, however, we can make it symmetric by the following strategy. Protocol P_{LL} performs asymmetric actions only for assignment of status (Section 3.2.1) and flipping fair and independent coins (Sections 3.2.3, 3.2.4, and 3.2.5). To assign the agents their statuses by symmetric transitions, we only have to add additional status Y and make the following three rules: $X \times X \rightarrow Y \times Y$, $Y \times Y \rightarrow X \times X$, $X \times Y \rightarrow A \times B$. Furthermore, similarly to the original rules of P_{LL} , when an agent v with status X or Y meets an agent with status A or B , v gets status A but it becomes a follower. This modification does not make any harmful influence on the analysis of stabilization time, at least asymptotically. Coin flips are dealt with in the same way. We assign a *coin status* J, K, F_0 , or F_1 to each follower. Every time a leader v becomes a follower, initial status J is assigned to v . Thereafter, when two followers meet, they change their coin statuses according to the following rules: $J \times J \rightarrow K \times K$, $K \times K \rightarrow J \times J$, $J \times K \rightarrow F_0 \times F_1$. These rules guarantees that the numbers of the followers with state F_0 and F_1 are always equal. Therefore, a leader can make a fair and independent coin flip every time it meets a follower whose coin state is F_0 or F_1 . If it meets a follower with coin state F_0 (resp. F_1), it recognizes that the result of the flip is head (resp. tail).

5 Conclusion

In this paper, we gave a leader election protocol with logarithmic stabilization time and with logarithmic number of agent stats in the population protocol model. Given a rough knowledge m of the population size n such that $m \geq \log_2 n$ and $m = O(\log n)$, the proposed protocol guarantees that exactly one leader is elected from n agents within $O(\log n)$ parallel time in expectation.

References

- [AAE08] Dana Angluin, James Aspnes, and David Eisenstat. “Fast computation by population protocols with a leader”. In: *Distributed Computing* 21.3 (2008), pp. 183–199.

- [AAG18] Dan Alistarh, James Aspnes, and Rati Gelashvili. “Space-optimal majority in population protocols”. In: *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM. 2018, pp. 2221–2239.
- [AG15] Dan Alistarh and Rati Gelashvili. “Polylogarithmic-time leader election in population protocols”. In: *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming*. Springer. 2015, pp. 479–491.
- [AG18] Dan Alistarh and Rati Gelashvili. “Recent algorithmic advances in population protocols”. In: *ACM SIGACT News* 49.3 (2018), pp. 63–73.
- [Ali+17] Dan Alistarh, James Aspnes, David Eisenstat, Rati Gelashvili, and Ronald L Rivest. “Time-space trade-offs in population protocols”. In: *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM. 2017, pp. 2560–2579.
- [Ang+06] D. Angluin, J Aspnes, Z. Diamadi, M.J. Fischer, and R. Peralta. “Computation in networks of passively mobile finite-state sensors”. In: *Distributed Computing* 18.4 (2006), pp. 235–253. ISSN: 0178-2770.
- [Bil+17] Andreas Bilke, Colin Cooper, Robert Elsässer, and Tomasz Radzik. “Brief Announcement: Population Protocols for Leader Election and Exact Majority with $O(\log^2 n)$ States and $O(\log^2 n)$ Convergence Time”. In: *Proceedings of the 38th ACM Symposium on Principles of Distributed Computing*. Springer. 2017, pp. 451–453.
- [DS18] David Doty and David Soloveichik. “Stable leader election in population protocols requires linear time”. In: *Distributed Computing* 31.4 (2018), pp. 257–271.
- [ER18] Robert Elsässer and Tomasz Radzik. “Recent results in population protocols for exact majority and leader election”. In: *Bulletin of EATCS* 3.126 (2018).
- [GS18] Leszek Gąsieniec and Grzegorz Stachowiak. “Fast space optimal leader election in population protocols”. In: *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM. 2018, pp. 2653–2667.
- [GSU18] Leszek Gąsieniec, Grzegorz Stachowiak, and Przemysław Uznański. “Almost logarithmic-time space optimal leader election in population protocols”. In: *arXiv preprint arXiv: 1802.06867* (2018).
- [MST18] Othon Michail, Paul G Spirakis, and Michail Theofilatos. “Simple and Fast Approximate Counting and Leader Election in Populations”. In: *International Symposium on Stabilizing, Safety, and Security of Distributed Systems*. Springer. 2018, pp. 154–169.
- [MU05] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.
- [SM19] Yuichi Sudo and Toshimitsu Masuzawa. *Leader Election Requires Logarithmic Time in Population Protocols*. 2019. arXiv: 1906.11121 [cs.DC].
- [Sud+12] Y. Sudo, J. Nakamura, Y. Yamauchi, F. Ooshita, H. Kakugawa, and T. Masuzawa. “Loosely-stabilizing leader election in a population protocol model”. In: *Theoretical Computer Science* 444 (2012), pp. 100–112.