Time-Optimal Leader Election in Population Protocols

Yuichi Sudo, *Member, IEEE*, Fukuhito Ooshita, *Member, IEEE*, Taisuke Izumi, Hirotsugu Kakugawa, *Member, IEEE*, and Toshimitsu Masuzawa, *Member, IEEE*

Abstract—In this article, we present the first leader election protocol in the population protocol model that stabilizes within $O(\log n)$ parallel time in expectation with $O(\log n)$ states per agent, where n is the number of agents. Given a rough knowledge m of $\lg n$ such that $m \geq \lg n$ and $m = O(\log n)$, the proposed protocol guarantees that exactly one leader is elected and the unique leader is kept forever thereafter. This protocol is time-optimal because it was recently proven that any leader election protocol requires $\Omega(\log n)$ parallel time.

Index Terms—Population protocols, leader election, stabilization time

1 Introduction

We consider the *population protocol* (PP) model [3] 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) with 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 [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], 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 the 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 configuration where all agents are in the same state, a population reaches a 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.* [3] 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 [10] showed that any constant space protocol requires linear parallel time to elect a unique leader. Alistarh and Gelashvili [4] made a breakthrough in 2015; they achieved 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 [6], [7], [8], [9], [14]. Gąsieniec *et al.* [8] gave a state-of-the-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.* [5] showed that any leader election protocol with $o(n/(\text{polylog } n))$ parallel time requires $\Omega(\log\log n)$ states. Michail *et al.* [9] gave a protocol with parameter τ $(2 \leq \tau \leq n)$, which stabilizes within $O(\log^2 n/\log \tau)$ parallel time and requires $O(\tau + \log n)$ states. When we set $\tau = \Theta(n^c)$ for some (small) constant c, this protocol stabilizes with $O(\log n)$ parallel time but requires a polynomial number of states. Those protocols with super-constant number of states [4], [5], [6], [7], [8], [9], [14] require some rough knowledge of n.[1] For example, in the protocols of [7] and [8], a $\Theta(\log\log n)$ value must be hard-coded to set the maximum value of one variable (named l and "level" in [7] and [8], respectively). One can find detailed information about the leader election in the PP model in two survey papers [15], [16].

The stabilization time of [9] is optimal; any leader election protocol requires $\Omega(\log n)$ parallel time even if it uses any large number of states and assumes the exact knowledge of population size n [11]. 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. It was shown in [11] that we cannot achieve $o(\log n)$ parallel

- *Y. Sudo and T. Masuzawa are with Osaka University, Osaka, 565-0871, Japan. E-mail: {y-sudou, masuzawa}@ist.osaka-u.ac.jp.*
- *F. Ooshita is with the Nara Institute of Science and Technology, Ikoma, Nara 630-0192, Japan. E-mail: f-oosita@is.naist.jp.*
- *T. Izumi is with the Nagoya Institute of Technology, Nagoya, Aichi 466-8555, Japan. E-mail: t-izumi@nitech.ac.jp.*
- *H. Kakugawa is with Ryukoku University, Shiga 520-2194, Japan. E-mail: kakugawa@rins.ryukoku.ac.jp.*

Manuscript received 12 Oct. 2019; revised 14 Mar. 2020; accepted 27 Apr. 2020. Date of publication 4 May 2020; date of current version 15 June 2020. (Corresponding author: Yuichi Sudo.)
Recommended for acceptance by R. Ge.
Digital Object Identifier no. 10.1109/TPDS.2020.2991771

1. Two leader election protocols are given in [9]. The first one requires knowledge of an upper bound of n. The second one does not use any knowledge of n for its transition rules (the authors of [9] call such a protocol *size-oblivious*) whereas the domain of a variable used in the protocol depends on n.

TABLE 1
Leader Election Protocols

	States	Stabilization Time	Parameter
[3]	$O(1)$	$O(n)$	
[4]	$O(\log^3 n)$	$O(\log^3 n)$	
[5]	$O(\log^2 n)$	$O(\log^{5.3} n \cdot \log\log n)$	
[6]	$O(\log n)$	$O(\log^2 n)$	
[7]	$O(\log\log n)$	$O(\log^2 n)$	
[8]	$O(\log\log n)$	$O(\log n \cdot \log\log n)$	
[9][2]	$O(\tau + \log n)$	$O(\log^2 n/\log \tau)$	$2 \le \tau \le n$
This work	$O(\log n)$	$O(\log n)$	

Stabilization time is shown in terms of parallel time and in expectation.

TABLE 2
Lower Bounds for Leader Election

	States	Stabilization Time
[10]	$O(1)$	$\Omega(n)$
[5]	$< 1/2\log\log n$	$\Omega(n/(\text{polylog } n))$
[11]	arbitrary	$\Omega(\log n)$

Stabilization time is shown in terms of parallel time and in expectation.

time even if we define the initial state such that all the agents are non-leaders initially.

1.1 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 a state-of-the-art protocol [8], P_{LL} achieves shorter (and best possible) stabilization time but uses larger space of each agent. Compared to [9] with $\tau = O(n^c)$ for some constant c, P_{LL} achieves drastically smaller space (from polynomial to logarithmic space) while maintaining the same (and optimal) stabilization time. The protocol P_{LL} requires a rough knowledge of the population size n as with the existing super-constant space protocols; it requires integer m such that $m \ge \lg n$ and $m = \Theta(\log n)$.

In the original population protocol model, interactions are *asymmetric*, that is, when two agents interact, one of them is given a role *initiator* and the other is given a role *responder*. Thus, an interaction between two agents with states p and q can show different behavior according to whether the agent with state p is an initiator or not. However, several works (e.g., [4], [5], [17]) are devoted to design a *symmetric protocol*, which does not utilize the roles, initiator and responder, to determine the next states at an interaction. (See Section 4 for the formal definition.) This property is important for some applications such as chemical reaction networks. We will give P_{LL} as an asymmetric protocol (i.e., a non-symmetric 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 [5] is almost independent and fair, the totally independent and fair coin clips achieved in this paper can contribute to a simple analysis in a variety of protocols in the PP model.

2. The protocol of [9] is not designed for the standard population protocol model, that is, this protocol requires *randomized transition* at each step. Specifically, when two agents interact, this protocol assumes that the two agents can access a random number of an arbitrary number of bits. However, fortunately, this protocol can be simulated in the standard population protocol model without increasing the stabilization time and the number of states asymptotically.

A brief announcement [1] and a preliminary extended abstract [2] of this article appeared in the proceedings of PODC 2019 and SSS 2019, respectively.

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 a 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 $T : Q \times Q \to Q \times Q$, a finite set Y of output symbols, and an output function $\pi_{\text{out}} : Q \to Y$. 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 $m \ge \lg n$ 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 $C : V \to Q$ 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* $\gamma = \gamma_0, \gamma_1, \ldots = (u_0, v_0), (u_1, v_1), \ldots$ is a sequence of interactions. A schedule determines which interaction occurs at each *step*, i.e., interaction γ_t happens at step t. In particular, we consider a *uniformly random scheduler* $\Gamma = \Gamma_0, \Gamma_1, \ldots$ 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 $t \ge 0$ and any distinct $u, v \in V$. All interactions $\Gamma_0, \Gamma_1, \ldots$ are independent of each other. Note that we use capital letter Γ for this uniformly 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 $\Xi_P(C_0, \gamma) = C_0, C_1, \ldots$ such that $C_t \xrightarrow{\gamma_t} C_{t+1}$ for all $t \ge 0$. Note that the execution $\Xi_P(C_0, \Gamma) = C_0, C_1, \ldots$ under the uniformly random scheduler Γ is a sequence of configurations where each C_i is a random variable. For a schedule $\gamma = \gamma_0, \gamma_1, \ldots$ and any $t \ge 0$, 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 $\gamma = \gamma_0, \gamma_1, \ldots, \gamma_{t-1}$ such that

$\Xi_P(C_{\mathrm{init},P},\gamma) = C_0, C_1, \ldots, C_t$ and $C = C_t$. We define $\mathcal{C}_{\mathrm{all}}(P)$ 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 $\mathcal{S}_{\mathcal{P}}$ be the set of configurations such that, for any configuration $C \in \mathcal{S}_{\mathcal{P}}$, 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 $\Xi_P(C_{\mathrm{init},P},\Gamma)$ reaches a configuration in $\mathcal{S}_{\mathcal{P}}$ 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 $\Xi_P(C_{\mathrm{init},P},\Gamma)$ reaches a configuration in $\mathcal{S}_{\mathcal{P}}$, 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 asymptotic expression such as $O(\log n)$.

In the proposed protocol, we often use *one-way epidemic* [18]. The notion of one-way epidemic is formalized as follows. Let $\gamma = \gamma_0, \gamma_1, \ldots = (u_0, v_0), (u_1, v_1), \ldots$ 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* $I_{V',r,\gamma} : [0,\infty) \to 2^V$ is defined as follows: $I_{V',r,\gamma}(0) = \{r\}$, and for $t = 1, 2, \ldots$,

$$I_{V',r,\gamma}(t) = \begin{cases} I_{V',r,\gamma}(t-1) & \text{if } u_{t-1} \notin V' \vee v_{t-1} \notin V' \\ I_{V',r,\gamma}(t-1) \cup \{u_{t-1}\} & \text{else if } v_{t-1} \in I_{V',r,\gamma}(t-1) \\ I_{V',r,\gamma}(t-1) \cup \{v_{t-1}\} & \text{else if } u_{t-1} \in I_{V',r,\gamma}(t-1) \\ I_{V',r,\gamma}(t-1) & \text{otherwise.} \end{cases}$$

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 from V'. Once an agent becomes infected, it remains infected thereafter.

The one-way epidemic plays an important role in analyzing the expected stabilization time of a population protocol. For example, consider an execution $\Xi_P(C_0,\Gamma) = C_0, C_1, \ldots$ 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.* [18] 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.* [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 [12].

Lemma 1. *Let $V' \subseteq V$, $r \in V'$, $n' = |V'|$, and $t \in \mathtt{N}$. We have* $\Pr(I_{V',r,\Gamma}(2\lceil n/n' \rceil t) \neq V') \leq n' e^{-t/n}$.

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 $X_{\mathrm{pre}} = T(\lceil \frac{n'+1}{2} \rceil)$ and $X_{\mathrm{post}} = T(n') - T(n' - \lceil \frac{n'+1}{2} \rceil + 1)$.

Let k be any integer such that $1 \leq k \leq n'$. When k agents are infected, an agent is newly infected with probability $k(n'-k)/\binom{n}{2}$ at every step. When $n'-k$ agents are infected, an agent is newly infected also with probability $k(n'-k)/\binom{n}{2}$ at every step. Therefore, $T(k+1) - T(k)$ and $T(n'-k+1) - T(n'-k)$ have the same probability distribution. Thus, $X_{\mathrm{pre}} = T(\lceil \frac{n'+1}{2} \rceil) = \sum_{j=1}^{\lceil (n'+1)/2 \rceil - 1} T(j+1) - T(j)$ and $X_{\mathrm{post}} = T(n') - T(n' - \lceil \frac{n'+1}{2} \rceil + 1) = \sum_{j=1}^{\lceil (n'+1)/2 \rceil - 1} T(n'-j+1) - T(n'-j)$ have the same probability distribution. Moreover, $X_{\mathrm{pre}} + X_{\mathrm{post}} \geq T(n')$ 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 $X_{\mathrm{post}} > \lceil n/n' \rceil t$. 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_vth step. We define $X_v = \max(T_{C_0,\Gamma}(v) - T_{\mathrm{half}}, 0)$. Consider the case $v \notin I_{V',r,\Gamma}(T_{\mathrm{half}})$. At any step $t \geq T_{\mathrm{half}}$, at least $n' - \lceil \frac{n'+1}{2} \rceil + 1$ ($\geq n'/2$) agents are infected. Therefore, each interaction Γ_t such that ($t \geq T_{\mathrm{half}}$) infects v with the probability at least $(1/\binom{n}{2}) \cdot (n'/2) > n'/n^2$, 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 $n'/2$, $\Pr(X_{\mathrm{post}} > \lceil n/n' \rceil t) \leq \Pr(\bigvee_{v \in V}(X_v > \lceil n/n' \rceil t)) \leq (n'/2) \cdot e^{-t/n}$ holds.

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

$$\begin{aligned} &\Pr\big(I_{V',r,\Gamma}(2\lceil n/n' \rceil t) \neq V'\big) \\ &\quad \leq \Pr\big(X_{\mathrm{pre}} > \lceil n/n' \rceil t\big) + \Pr\big(X_{\mathrm{post}} > \lceil n/n' \rceil t\big) \leq n' e^{-t/n}. \end{aligned}$$

□

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

Lemma 2 ([19], Theorems 4.4, 4.5). *Let $X_1, \ldots, X_s$ be independent Poisson trials, and let $X = \sum_{i=1}^{s} X_i$. Then*

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

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

3 Logarithmic Leader Election

3.1 Key Ideas

In this subsection, we give the key ideas of the proposed protocol P_{LL}. Each agent v keeps an output variable $v.\mathtt{leader} \in \{false,\ true\}$. 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 $\mathtt{leader} = true$, 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 *Quick Elimination*(), *Tournament*(), and *BackUp*(), which are executed in this order. These three modules guarantee 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 $QuickElimination()$ 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 $Tournament()$, 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 $QuickElimination()$. 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 $O(\log n) + O(1/\log n) \cdot O(\log^2 n) = O(\log n)$ parallel time in expectation.

In the remainder of this subsection, we briefly give the key ideas to design the three modules satisfying the above guarantees. 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. 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 the rest of this subsection, 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 [5]. 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:

- 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,
- Let s_v be the number of heads that v observes in the above coin flips and let $s_{\max} = \max_{v \in V} s_v$,
- 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. For simplicity, we define $k_i = -1$ if no such j exists and we define $k_n = 0$. Then, for any $i \geq 2$, 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 $v.\texttt{level}_Q$ 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 $v.\texttt{level}_Q$, the maximum value of $\texttt{level}_Q$, i.e., $s_{\max}$, is propagated from agent to agent via *one-way epidemic* [18], that is,

- each agent memorizes the largest value of $\texttt{level}_Q$ it has observed, and
- the larger value is propagated to the agent with smaller value at every interaction.

From Lemma 1, 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 $v.\texttt{leader}$ 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 is 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 $\texttt{level}_Q$ 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 $v.\texttt{rand}$. Initially, $v.\texttt{rand} = 1$. Every time it has an interaction, it updates $v.\texttt{rand}$ by $v.\texttt{rand} \leftarrow 2v.\texttt{rand} + j$ 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 $\lceil \lg m \rceil = O(\log \log n)$ interactions. Thus, when all the i leaders encounter at least $\lceil \lg m \rceil$ interactions, for every leader v, $v.\texttt{rand}$ is a random variable uniformly chosen from the set $\{2^{\lceil \lg m \rceil}, 2^{\lceil \lg m \rceil} + 1, \ldots, 2^{\lceil \lg m \rceil + 1} - 1\}$, whose size is $2^{\lceil \lg m \rceil}$. Although $u.\texttt{rand}$ and $v.\texttt{rand}$ are not independent of each other for any distinct leader u and v, we will present a way to remove any dependence between $u.\texttt{rand}$ and $v.\texttt{rand}$ in Section 3.2. As with $QuickElimination()$, the maximum value of 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 $v_1, v_2, \ldots, v_i$ be the i leaders that survive $QuickElimination()$. Let $r_1, r_2, \ldots, r_i$ be the resulting values of $v_0.\texttt{rand}, v_1.\texttt{rand}, \ldots, v_i.\texttt{rand}$ and define $r_{\max}(j) = \max(r_1, r_2, \ldots, r_j)$ for any $j = 1, 2, \ldots, i$. Clearly, the number of leaders at the end of $Tournament()$ is exactly one if $r_{j+1} \neq r_{\max}(j)$ holds for all $j = 1, 2, \ldots, i-1$. By the union bound and independence between $r_1, r_2, \ldots, r_i$, this holds with probability at least $1 - \sum_{j=1}^{i-1} 2^{-\lceil \lg m \rceil} \geq 1 - i/m \geq 1 - i/(\lg n)$. On the other hand, an execution of $Tournament()$ never eliminates all leaders since there is 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 $v.\texttt{level}_B$. Initially, $v.\texttt{level}_B = 0$. Every leader v repeats the following procedure until $v.\texttt{level}_B$ reaches s or v becomes a follower, where s is a sufficiently large integer in $O(\log n)$. (As we will see later, we set $s = 41m$.)

- Make a coin flip. If the result is head (i.e., v participates in an interaction as an initiator), v increments $v.\texttt{level}_B$ by one. If the result is tail, v does nothing.
- Wait for sufficiently long but logarithmic parallel time so that the maximum $\texttt{level}_B$ 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 $v.\texttt{leader} \leftarrow false$. 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 $1 \leq j \leq s$. Consider the first time that $\texttt{level}_B$ 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 $u.\texttt{level}_B < j$, 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 $1/2 - O(n^{-1})$, no less than half of leaders in $V' \setminus v$ becomes followers, that is, the number of leaders decreases to at most $1 + \lfloor |V'|/2 \rfloor$. Chernoff bound guarantees that the number of leaders becomes one with high probability until $v.\texttt{level}_B$ for some leader v reaches s. 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 $O(m \log n) + O(n^{-1}) \cdot O(n) = O(\log^2 n)$.

TABLE 3
Variables of P_{LL}

Groups	Variables	Initial values
All agents	$\texttt{leader} \in \{false, true\}$	$true$
	$\texttt{tick} \in \{false, true\}$	$false$
	$\texttt{status} \in \{X, A, B\}$	X
	$\texttt{epoch} \in \{1, 2, 3\}$	1
	$\texttt{init} \in \{1, 2, 3, 4\}$	1
	$\texttt{color} \in \{0, 1, 2\}$	0
V_B	$\texttt{count} \in \{0, 1, \ldots, c_{\max} - 1\}$	Undefined
$V_A \cap V_1$	$\texttt{level}_Q \in \{0, 1, \ldots, c_{\max}\}$	Undefined
	$\texttt{done} \in \{false, true\}$	Undefined
$V_A \cap V_2$	$\texttt{rand} \in \{1, 2, \ldots, 2^{\lceil \lg m \rceil + 1} - 1\}$	Undefined
$V_A \cap V_3$	$\texttt{level}_B \in \{0, 1, \ldots, c_{\max}\}$	Undefined

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 [8], where agents in different groups manage different variables in addition to the above six variables. In the remainder of this paper, we refer to the above six variables as *common variables* and other variables by *additional variables*. The population is divided to six groups based on two common variables $\texttt{status} \in \{X, A, B\}$ and $\texttt{epoch} \in \{1, 2, 3\}$. That is, we divide the population to V_X, V_B, $V_A \cap V_1$, $V_A \cap V_2$, $V_A \cap V_3$, where we denote $V_Z = \{v \in V \mid v.\texttt{status} = Z\}$ for

Algorithm 1. P_{LL}

Notations:
$c_{\max} = 41m$
$V_Z = \{v \in V \mid v.\texttt{status} = Z\}$ for $Z \in \{X, A, B\}$
$V_i = \{v \in V \mid v.\texttt{epoch} = i\}$ for $i \in \{1, 2, 3\}$
Output function π_{out}:
if $v.\texttt{leader} = \mathit{true}$ 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: $(a_0.\texttt{status}, a_0.\texttt{level}_Q, a_0.\texttt{done}, a_0.\texttt{leader}) \leftarrow (A, 0, \mathit{false}, \mathit{true})$
3: $(a_1.\texttt{status}, a_1.\texttt{count}, a_1.\texttt{leader}) \leftarrow (B, 0, \mathit{false})$
4: **else if** $\exists i \in \{0, 1\} : a_i \in V_X \wedge a_{1-i} \notin V_X$ **then**
5: $(a_i.\texttt{status}, a_i.\texttt{level}_Q, a_i.\texttt{done}, a_i.\texttt{leader}) \leftarrow (A, 0, \mathit{true}, \mathit{false})$
6: **end if**
7: $a_0.\texttt{tick} \leftarrow a_1.\texttt{tick} \leftarrow \mathit{false}$
8: $\mathit{CountUp}()$
9: **for all** $i \in \{0, 1\}$ such that $a_i.\texttt{tick}$ **do** $a_i.\texttt{epoch} \leftarrow \max(a_i.\texttt{epoch} + 1, 3)$ **endfor**
10: $a_0.\texttt{epoch} \leftarrow a_1.\texttt{epoch} \leftarrow \max(a_0.\texttt{epoch}, a_1.\texttt{epoch})$
11: **for all** $i \in \{0, 1\}$ such that $a_i.\texttt{epoch} > a_i.\texttt{init}$ **do**
12: **if** $a_i \in V_A \cap V_2$ **then** $a_i.\texttt{rand} \leftarrow 1$ **endif**
13: **if** $a_i \in V_A \cap V_3$ **then** $a_i.\texttt{level}_B \leftarrow 0$ **endif**
14: $a_i.\texttt{init} \leftarrow a_i.\texttt{epoch}$
15: **end for**
16: **if** $a_0, a_1 \in V_1$ **then**
17: Execute $\mathit{QuickElimination}()$
18: **else if** $a_0, a_1 \in V_2$ **then**
19: Execute $\mathit{Tournament}()$
20: **else if** $a_0, a_1 \in V_3$ **then**
21: Execute $\mathit{BackUp}()$
22: **end if**

$Z \in \{X, A, B\}$ and $V_i = \{v \in V \mid v.\texttt{epoch} = i\}$ for $i \in \{1, 2, 3\}$. We have no additional variables for agents in group V_X, one additional variable $\texttt{count} \in \{0, 1, \ldots, c_{\max} - 1\}$ for agents in V_B where $c_{\max} = 41m$, two additional variables $\texttt{level}_Q \in \{0, 1, \ldots, c_{\max}\}$ and $\texttt{done} \in \{\mathit{false}, \mathit{true}\}$ for agents in $V_A \cap V_1$, one additional variable $\texttt{rand} \in \{1, 2, \ldots, 2^{\lceil \lg m \rceil + 1} - 1\}$ for agents in $V_A \cap V_2$, and one additional variable $\texttt{level}_B \in \{0, 1, \ldots, c_{\max}\}$ for agents in $V_A \cap V_3$. Agents in any group have only $O(\log n)$ states. This is because every common variable has constant size domain, every group has at most one non-constant additional variable and any of such variables can take $O(\log n)$ 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 groups V_L and V_F based on a common variable $\texttt{leader}$; V_L (resp., V_F) is the set of agents $v \in V$ such that $v.\texttt{leader} = \mathit{true}$ (resp., $v.\texttt{leader} = \mathit{false}$). We introduce these two groups only for simplicity of notation.

The pseudocode of P_{LL} is given in Algorithm 1 and its modules $\mathit{CountUp}()$, $\mathit{QuickElimination}()$, $\mathit{Tournament}()$, and $\mathit{BackUp}()$ are presented in Algorithms 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 $\texttt{epoch}$ using module $\mathit{CountUp}()$. Initially, $v.\texttt{epoch} = 1$ holds for all $v \in V_1$, that is, $V_1 = V$. In an execution of P_{LL}, $v.\texttt{epoch}$ never decreases and increases by one every sufficiently large logarithmic parallel time in expectation until it reaches 3, as we will explain later. Note that $v.\texttt{epoch}$ may increase by more than one at an interaction, but this happens with a negligibly small probability, as we will see 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 $\texttt{init}$, which is set to 1 initially. Whenever $v.\texttt{epoch}$ increases, $v.\texttt{epoch} > v.\texttt{init}$ must hold, then v initializes additional variables according to v's group and executes $v.\texttt{init} \leftarrow v.\texttt{epoch}$. For example, when the $\texttt{epoch}$ of agent $v \in V_A$ changes from 2 to 3 i.e., v moves from group $V_A \cap V_2$ to $V_A \cap V_3$, it initializes an additional variable $a_i.\texttt{level}_B$ 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 $\texttt{epoch}$. Specifically, agents execute $\mathit{QuickElimination}()$, $\mathit{Tournament}()$, and $\mathit{BackUp}()$ while they are in V_1, V_2, and V_3, respectively.

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

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 $\texttt{level}_Q$ and $\texttt{done}$ to 0 and false respectively and remains a leader (Line 2) while the responder initializes its additional variable $\texttt{count}$ to 0 and becomes a follower by $\texttt{leader} \leftarrow \mathit{false}$ (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 initializes its additional variable $\texttt{level}_Q$ and $\texttt{done}$ to 0 and true respectively (Line 5). For agent v, assigning true to $v.\texttt{done}$ means that v never joins a game with coin flips in $\mathit{QuickElimination}()$.

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 lemmas.

Lemma 4. *In an execution of P_{LL}, $|V_B| \geq 1$ always holds after the first interaction finishes.*

Proof. The first interaction of the execution assigns status B to one agent, and it never changes its status thereafter. □

Lemma 5. *In an execution of P_{LL}, $|V_A| \geq n/2$ and $|V_F| \geq n/2$ 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 the

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

number of agents which get status A (resp., B and A) by Line 2 (resp., Line 3 and Line 5). We have $x = y \le n/2$ by the definition of P_{LL}, which gives $|V_A| = x + z = n - y \ge n/2$. Moreover, $|V_L| \le x \le n/2$ holds because the number of leaders is monotonically non-increasing in an execution of P_{LL}. □

Lemma 6. *In an execution of P_{LL}, with high probability, every agent gets status A or B within $\lceil n \ln n \rceil$ steps.*

Proof. By definition of P_{LL}, once an agent has an interaction, it gets status A or B and never changes its status thereafter. Thus, it suffices to show that, with high probability, every agent has at least one interaction during the first $\lceil n \ln n \rceil$ steps. Each agent v has an interaction with probability $(n-1)/\binom{n}{2} = 2/n$ at each step. Therefore, v has no interaction during the first $n \ln n$ steps with probability at most $(1-2/n)^{n \ln n} \le e^{-2 \ln n} = n^{-2}$. Thus, the union bound gives the lemma. □

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 [18]. Recently, in [7] and [8], 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 other modules, we achieve synchronization in much simpler way with $O(\log n)$ states per agent.

For synchronization, we use common variables $\texttt{color} \in \{0,1,2\}$ in all agents and an additional variable $\texttt{count} \in \{0, 1, \ldots, c_{\max} - 1\}$ 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

1) 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,
2) 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 the second item 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 $v.\texttt{count}$ from $c_{\max} - 1$ to 0, v gets a new color by incrementing $v.\texttt{color}$ 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 $u.\texttt{color} = v.\texttt{color} + 1 \pmod 3$ meet, v executes $v.\texttt{color} \leftarrow u.\texttt{color}$ and resets its count to 0 (Line 31-33).

Every time an agent v gets a new color, it raises a tick flag, i.e., assigns $v.\texttt{tick} \leftarrow$ *true* (Lines 27 and 32). This common variable $v.\texttt{tick}$ is used only for simplicity of the pseudocode and it does not affect the transition at v's next interaction ($v.\texttt{tick}$ is reset to *false* in Line 7), unlike any other variable. When $v.\texttt{tick}$ is raised, $v.\texttt{epoch}$ increases by one unless it has already reached 3 (Line 9). After two agents u and v execute Lines 7-9 at an interaction, $u.\texttt{epoch} = v.\texttt{epoch}$ usually holds. However, this equation does not hold when synchronization fails. For this case, we substitute $\max(u.\texttt{epoch}, v.\texttt{epoch})$ into $u.\texttt{epoch}$ and $v.\texttt{epoch}$ 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 $v \in V$, $v.\texttt{tick}$ is raised and $v.\texttt{epoch}$ increases by one with high probability in every sufficiently large $\Theta(\log n)$ parallel time until $v.\texttt{epoch}$ reaches 3. 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 guarantee that all agents proceed to the third 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,*
- *$v.\texttt{count} = 0$ holds for all $v \in V_B$ such that $v.\texttt{color} = i$, and*
- *no agent has color $i + 1 \pmod 3$.*

Lemma 7. *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.*

Algorithm 3. $QuickElimination()$

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

Proof. Simple Chernoff bound gives the lemma because each 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. □

The goal of our synchronization, 1) and 2), are formalized as follows.

Lemma 8. *Let $i \in \{0,1,2\}$, $C_0 \in \mathcal{C}_{\text{start}}(i)$, and $\Xi_{P_{LL}}(C_0, \Gamma) = C_0, C_1, \ldots$. 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 $\Xi_{P_{LL}}(C_0, \Gamma)$ with high probability.*
- *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. Proposition P_2 immediately follows from Lemma 1 with $n' = n$. Proposition P_3 also immediately follows from Lemmas 4 and 7. 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.

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

where we use $c_{\max} \geq 41 \lg n \geq 58 \ln n$ for the first inequality and Chernoff Bound in the form of (1) in Lemma 2 for the second 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 $1 - O(n^{-1})$. □

3.2.3 $QuickElimination()$

The module $QuickElimination()$ uses additional variables $\texttt{level}_Q \in \{0, 1, \ldots, c_{\max}\}$ and $\texttt{done} \in \{\mathit{false}, \mathit{true}\}$ of group $V_A \cap V_1$. Each agent v executes this module only when $v.\texttt{epoch} = 1$ 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 $v.\texttt{done} = \mathit{false}$ or v is a follower and $v.\texttt{done} = \mathit{true}$.

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 $v.\texttt{level}_Q$ the number of times it observes "heads". Specifically, a leader with $v.\texttt{done} = \mathit{false}$ 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 $\texttt{level}_Q$ by one (Line 36); otherwise, it stops coin flipping by assigning $v.\texttt{done} \leftarrow \mathit{true}$ (Line 37). The largest $\texttt{level}_Q$ 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 $\texttt{level}_Q$ to $\max(u.\texttt{level}_Q, v.\texttt{level}_Q)$ (Line 41). When an agent $v \in V_A$ meets an agent with larger $\texttt{level}_Q$ than $v.\texttt{level}_Q$, it becomes a follower (Line 40). The correctness of $QuickElimination()$ is formalized as the following lemma.

Lemma 9. *Let $\Xi = \Xi_{P_{LL}}(C_{\text{init},P_{LL}}, \Gamma) = C_0, C_1, \ldots$. In a configuration $C_{\lfloor 21n \ln n \rfloor}$, $\Pr(|V_L| = i) < 2^{1-i} + \epsilon_i$ holds for any $i = 2, 3, \ldots, n$ 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., $v.\texttt{epoch} = 1$ holds,
- $v.\texttt{level}_Q < c_{\max}$ holds for all $v \in V_A$
- all agents in V_A have the same $\texttt{level}_Q$ and $v.\texttt{done} = \mathit{true}$ holds for all $v \in V_A$.

Intuitively, the second condition guarantees that no agent increases $\texttt{level}_Q$ to the upper limit $c_{\max}$ 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 $\texttt{level}_Q$ propagates to the whole sub-population V_A within the first $\lfloor 21n \ln n \rfloor$ interactions. The second condition $\forall v \in V_A : v.\texttt{level}_Q < c_{\max}$ is necessary because if some agent in V_A increases $\texttt{level}_Q$ to $c_{\max}$, then it may fail to simulate the competition game successfully.

Note that the competition game guarantees that exactly i agents survive 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 8 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 $c_{\max}$ times in a row and the probability that such an event happens is at most $n(1/2)^{c_{\max}} \leq n \cdot 2^{-41 \lg n} = O(n^{-1})$.

Algorithm 4. $Tournament()$

Interaction between initiator a_0 and responder a_1:
43: **if** $i \in \{0,1\} : a_i \in V_L \wedge a_{1-i} \in V_F \wedge a_i.\texttt{rand} < 2^{\lceil \lg m \rceil}$ **then**
44: $\quad a_i.\texttt{rand} \leftarrow 2a_i.\texttt{rand} + i$
45: **end if**
46: **if** $a_0, a_1 \in V_A \wedge a_0.\texttt{rand} \geq 2^{\lceil \lg m \rceil} \wedge a_1.\texttt{rand} \geq 2^{\lceil \lg m \rceil} \wedge \exists i \in \{0,1\}$: $a_i.\texttt{rand} < a_{1-i}.\texttt{rand}$ **then**
47: $\quad a_i.\texttt{leader} \leftarrow \mathit{false}$
48: $\quad a_i.\texttt{rand} \leftarrow a_{1-i}.\texttt{rand}$
49: **end if**

In what follows, we prove that the third condition holds with high probability. By Lemma 6, with high probability, every agent gets status A or B during the first $\lceil n \ln n \rceil$ interactions. After that, each leader meets a follower with probability at least $|V_F|/\binom{n}{2} \geq 1/n$ at each step by Lemma 5. Hence, Chernoff bound in the form of (2) in Lemma 2, it holds with high probability that every leader meets followers no less than $2\lg n$ times during the first $\lceil 10n \ln n \rceil (\geq \lceil n \ln n \rceil + \lceil 6n \lg n \rceil)$ interactions. As with the analysis of the probability of the second condition, we can easily prove that, with high probability, no leader gets "heads" $2\lg n$ times in a row. Therefore, with high probability, all agents in V_A finish making coin flips within the first $\lceil 10n \ln n \rceil$ interactions. Thereafter, the maximum value of $\texttt{level}_Q$ is propagated to the whole sub-population V_A by one-way epidemic in V_A. The epidemic finishes within the next $\lceil 8n \ln n \rceil$ interactions with high probability by Lemma 1. Since $\lceil 10n \ln n \rceil + \lceil 8n \ln n \rceil < \lfloor 21n \ln n \rfloor$, the third condition also holds with high probability. □

Note that an execution of $QuickElimination()$ never eliminates all leaders from the population because a leader v with $v.\texttt{level}_Q = \max_{u \in V_A \cap V_1} u.\texttt{level}_Q$ never becomes a follower.

3.2.4 *Tournament*()

In an execution of $Tournament()$, each leader v gets a random number, say *nonce*, uniformly at random from $\{2^{\lceil \lg m \rceil}, 2^{\lceil \lg m \rceil} + 1, \ldots, 2^{\lceil \lg m \rceil + 1} - 1\}$ by making coin flips $\lceil \lg m \rceil$ times, and stores it in $v.\texttt{rand}$ (Line 43-45). 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 begin one-way epidemics of the largest value of these nonces (Lines 46-49). By Chernoff bound, it holds with high probability that all leaders finish generating their nonces 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 the population because a leader with the largest nonce never becomes a follower.

Lemma 10. *In an execution $\Xi = \Xi_{P_{LL}}(C_{\mathrm{init},P_{LL}}, \Gamma) = C_0, C_1, \ldots$, the number of leaders becomes exactly one before some agent enters the third epoch (i.e., $\texttt{epoch} = 3$) with probability $1 - O(1/\log n)$.*

Algorithm 5. $BackUp()$

Interaction between initiator a_0 and responder a_1:
50: **if** $a_0.\texttt{tick} \wedge a_0 \in V_L$ **then**
51: $\quad a_0.\texttt{level}_B \leftarrow \min(a_0.\texttt{level}_B + 1, c_{\max})$
52: **end if**
53: **if** $a_0, a_1 \in V_A \wedge \exists i \in \{0,1\} : a_i.\texttt{level}_B < a_{1-i}.\texttt{level}_B$ **then**
54: $\quad a_i.\texttt{level}_B \leftarrow a_{1-i}.\texttt{level}_B$
55: $\quad a_i.\texttt{leader} \leftarrow \mathit{false}$
56: **end if**
57: **if** $\forall i \in \{0,1\} : a_i \in V_L$ **then** $a_1.\texttt{leader} \leftarrow \mathit{false}$ **end if**

Proof. By Lemma 8, all agents are still in the first epoch in configuration $C_{\lfloor 21n \ln n \rfloor}$ with high probability. Thereafter, execution Ξ reaches a configuration in $\mathcal{C}_{\mathrm{start}}(1)$ within the next $O(n \log n)$ interactions with high probability by Lemma 8. After that, we execute $Tournament()$ in the second epoch. In what follows, we assume that $|V_F| \geq n/2$ and $|V_A| \geq n/2$ always hold in the second epoch, which does not ruin the correctness of this proof because we have Lemmas 5 and 6 and allow an error probability of $O(1/n) \subseteq O(1/\log n)$.

We say that an execution of $Tournament()$ finishes completely if every leader finishes generating a nonce and the maximum value of nonces is propagated to the whole sub-population V_A. In the second epoch, each leader generates a nonce by meeting followers $\lceil \lg m \rceil = O(\log m) = O(\log \log n)$ times while each leader meets a follower with probability $|V_F|/\binom{n}{2} \geq 1/n$ at each step. Therefore, by Chernoff bound and Lemma 1 (we assume $|V_A| \geq n/2$ above), 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 8, an execution of $Tournament()$ finishes completely before some agent enters the third epoch, with high probability.

Let i be the number of leaders in $C_{\lfloor 21n \ln n \rfloor}$. Since a leader generates a nonce uniformly at random among $\{2^{\lceil \lg m \rceil}, 2^{\lceil \lg m \rceil} + 1, \ldots, 2^{\lceil \lg m \rceil + 1} - 1\}$, the arguments in Section 3.1.2 yields that exactly one leader is elected with probability at least $1 - i \cdot 2^{-\lceil \lg m \rceil} \geq 1 - O(i/\log n)$ after an execution of module $Tournament()$ finishes completely. Therefore, by Lemma 9, the execution of $Tournament$ decreases the number of leaders to exactly one before some agent enters the third epoch with probability

$$1 - O\left(n^{-1} + \sum_{i=2}^{n} 2^{1-i} \cdot \left(\frac{i}{\log n}\right)\right) = 1 - O\left(n^{-1} + \frac{1}{\log n} \sum_{i=2}^{n} \frac{i}{2^{i-1}}\right) = 1 - O\left(\frac{1}{\log n}\right).$$

□

3.2.5 *BackUp*()

This module $BackUp()$ uses only one additional variable $\texttt{level}_B$ to elect the unique leader. For any $v \in V_A$, variables $v.\texttt{level}_B$ is initialized by $v.\texttt{level}_B \leftarrow 0$ at the first time $v.\texttt{epoch} = 3$ holds (Line 9).

As long as synchronization succeeds, each $v.\texttt{tick}$ 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 at an interaction. If v sees "head" (i.e., it is an initiator at that interaction), it increments $v.\texttt{level}_B$ by one unless $v.\texttt{level}_B$ already reaches $c_{\max}$ (Lines 50-52). The largest value of $\texttt{level}_B$ is propagated via one-way epidemic in sub-population V_A (Lines 53-56). If a leader v observes a larger value of $\texttt{level}_B$ than $v.\texttt{level}_B$, it becomes a follower (Line 55). Furthermore, this module includes simple leader election [3]; when two leaders interact and they observe that they have the same value of $\texttt{level}_B$ at Line 57, then the responder becomes a follower. Note that, unlike coin flips in $QuickElimination()$ and $Tournament()$, coin flips made by leaders in $BackUp()$ are not necessarily independent of each other because two coin flips may be made simultaneously at an interaction when two leaders interact directly and then their results are no longer independent of each other. However, this dependence does not matter since an interaction between two leaders makes one of them a follower, which will be carefully analyzed in the proof of Lemma 13.

Definition 3. *We define $b_{\max} = \max_{v \in V_A \cap V_3} v.\texttt{level}_B$. We do not care about $b_{\max}$ when $V_A \cap V_3$ is empty.*

Lemma 11. *There is always at least one leader in an execution of P_{LL}.*

Proof. As mentioned above, $QuickElimination()$ and $Tournament()$ never eliminate all leaders from the population. Therefore, it suffices to show that $BackUp()$ never eliminates all leaders from the population. Define $V_{\max} = \{u \in V_L \cap V_A \cap V_3 \mid u.\texttt{level}_B = b_{\max}\}$. In the third epoch, only leaders can increase $b_{\max}$ (Line 51). Therefore, $V_{\max}$ is not empty when all agents enter the third epoch, and a leader in $V_{\max}$ becomes a follower only if it interacts with another leader in $V_{\max}$. This means that $|V_{\max}|$ decreases if and only if an interaction happens such that two leaders in $V_{\max}$ meet or a leader in $V_{\max}$ increases its $\texttt{level}_B$. After such an interaction, at least one leader exists in $V_{\max}$. Thus, $BackUp()$ also never eliminates all leaders. □

Definition 4. *We define $\mathcal{B}$ as the set of all configurations in $\mathcal{C}_{\text{all}}(P_{LL})$ where every agent is in the third epoch (i.e., $\texttt{epoch} = 3$) and has status A or B, and all agents in V_A have the same value for $\texttt{level}_B$ (i.e., $\forall v \in V_A : v.\texttt{level}_B = b_{\max}$). For $j = 0, 1, \ldots, c_{\max}$, we define $\mathcal{B}_j$ as the set of all configurations in $\mathcal{B}$ where $b_{\max} = j$.*

Lemma 12. *Execution $\Xi = \Xi_{P_{LL}}(C_{\text{init},P_{LL}}, \Gamma)$ reaches a configuration in $(\mathcal{B}_0 \cup \mathcal{B}_1) \cap \mathcal{C}_{\text{color}}(2)$ within $O(\log n)$ parallel time with high probability.*

Proof. By Lemmas 6 and 8, execution Ξ reaches a configuration $C \in \mathcal{C}_{\text{start}}(2)$ where every agent has status A or B within $O(n \log n)$ steps with high probability. After that, by Lemma 8, it holds with high probability that Ξ reaches a configuration in $\mathcal{C}_{\text{color}}(2)$ within the next $\lfloor 4n \ln n \rfloor$ steps and $b_{\max} \in \{0, 1\}$ holds at this time. Thereafter, the maximum value $b_{\max}$, which is either 0 or 1, is propagated to the whole population within the next $\lceil 8n \ln n \rceil$ steps with high probability by Lemma 1 with $t = 2 \ln n$. At this time, a configuration is still in $\mathcal{C}_{\text{color}}(2)$ with high probability by Lemma 8 again and the fact that $\lfloor 4n \ln n \rfloor + \lceil 8n \ln n \rceil \leq \lfloor 21n \ln n \rfloor$. Overall, Ξ reaches a configuration in $(\mathcal{B}_0 \cup \mathcal{B}_1) \cap \mathcal{C}_{\text{color}}(2)$ within $O(\log n)$ parallel time with high probability. □

Lemma 13. *Let $i \in \{0, 1, 2\}$, $j \in \{0, 1, \ldots, c_{\max} - 1\}$, and $k \in \{1, 2, \ldots, n-1\}$. Let C_0 be a configuration in $\mathcal{B}_j \cap \mathcal{C}_{\text{color}}(i)$ where exactly $1 + k$ leaders exist. Let $\Xi_{P_{LL}}(C_0, \Gamma) = C_0, C_1, \ldots$. Then, the following propositions P_4 and P_5 hold.*

- *P_4: With probability $1/4 - O(1/n)$, execution Ξ reaches a configuration in $\mathcal{B}_{j+1} \cap \mathcal{C}_{\text{color}}(i + 1 \pmod 3)$ where at most $1 + \lfloor k/2 \rfloor$ leaders exist within $O(\log n)$ parallel time.*
- *P_5: With high probability, execution Ξ reaches a configuration in $(\mathcal{B}_j \cup \mathcal{B}_{j+1}) \cap \mathcal{C}_{\text{color}}(i + 1 \pmod 3)$ within $O(\log n)$ parallel time.*

Proof. First, we prove P_5. A leader increases its $\texttt{level}_B$ only when it raises a tick. Therefore, by Lemma 1 with $n' = |V_A|$ and Lemmas 5 and 8, Ξ reaches a configuration $C' \in (\mathcal{B}_j \cup \mathcal{B}_{j+1}) \cap \mathcal{C}_{\text{color}}(i + 1 \pmod 3)$ within $O(\log n)$ parallel time, with high probability. Thus, P_5 holds.

Then, we prove P_4. As mentioned above, the population reaches a configuration $C' \in (\mathcal{B}_j \cup \mathcal{B}_{j+1}) \cap \mathcal{C}_{\text{color}}(i + 1 \pmod 3)$ within $O(\log n)$ parallel time, with high probability. Before the population reaches a configuration C', every leader raises a tick exactly once. When a leader raises a tick, it increases its $\texttt{level}_B$ with probability $1/2$. (Note that this probability is independent of the probability that synchronization succeeds.) Since there is always at least one leader, $\texttt{level}_B$ increases from j to $j + 1$ before the population reaches C' with probability at least $1/2$. Consider this case and let v_l be the leader that increases its $\texttt{level}_B$ to $j + 1$ for the first time. Then, the new level $j + 1$ is propagated to the whole population, by which all leaders with $\texttt{level}_B = j$ become followers. Divide the set of the k leaders except for v_l in C_0 to three subsets L_1, L_2, and L_3 based on how each leader makes a coin flip before the population reaches C'. Set L_1 is the set of the leaders each of which makes a coin flip when it interacts with a follower. Set L_2 is the set of the leaders each of which makes a coin flip when it interacts with a leader. Set L_3 is the set of the leaders each of which becomes a follower before it raises a tick in Ξ. Let $k_1 = |L_1|$, $k_2 = |L_2|$, and $k_3 = |L_3|$. Each of the k_1 leaders in L_1 makes a coin flip independently and it sees "tail" with probability $1/2$, after which it becomes a follower before the population reaches C'. Since an interaction between two leaders makes one of them a follower, exactly half of the leaders in L_2 becomes a follower. By definition, every leader in L_3 becomes a follower before the population reaches C'. Therefore, with probability at least $1/2$, at most $\lfloor k_1/2 \rfloor + k_2/2 < \lfloor k/2 \rfloor$ leaders survive (i.e., are still leaders) in $C' \in \mathcal{B}_{j+1} \cap \mathcal{C}_{\text{color}}(i + 1 \pmod 3)$. To conclude, with probability $1/4 - O(1/n)$, execution Ξ reaches a configuration in $\mathcal{B}_{j+1} \cap \mathcal{C}_{\text{color}}(i + 1 \pmod 3)$ where at most $1 + \lfloor k/2 \rfloor$ leaders exist within $O(\log n)$ parallel time. Thus, P_4 holds. □

Lemma 14. *Let C_0 be a configuration in $(\mathcal{B}_0 \cup \mathcal{B}_1) \cap \mathcal{C}_{\text{color}}(2)$. Then execution $\Xi = \Xi_{P_{LL}}(C_0, \Gamma)$ reaches a configuration where there exists exactly one leader within $O(\log^2 n)$ parallel time, with probability $1 - O((\log n)/n)$.*

Proof. Lemma 13 yields that, in every $O(\log n)$ parallel time, the number of leaders decreases almost by half, specifically decreases from $1 + k$ to at most $1 + \lfloor k/2 \rfloor$, with

probability $1/4 - O(1/n)$. During this period, no matter whether the number of leaders decreases almost by half or not, $b_{\max}$ increases by at most one with high probability. Therefore, by applying Lemma 13 $c_{\max} - 1$ times, Chernoff bound in the form of (2) and the union bound, the number of leaders decreases to one within $O(\log^2 n)$ parallel time with probability $1 - O((\log n)/n)$. □

Lemma 15. *Let C be any configuration in $\mathcal{C}_{\mathrm{all}}(P_{LL})$ where all agents are in the third epoch and let $\Xi = \Xi_{P_{LL}}(C, \Gamma) = C_0, C_1, \ldots$. 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 [3], i.e., one leader becomes a follower when two leaders meet. □

Lemma 16. *Let C_0 be a configuration in $(\mathcal{B}_0 \cup \mathcal{B}_1) \cap \mathcal{C}_{\mathrm{color}}(2)$. Then, execution $\Xi = \Xi_{P_{LL}}(C_0, \Gamma)$ reaches a configuration where there is exactly one leader within $O(\log^2 n)$ parallel time in expectation.*

Proof. By Lemmas 14 and 15, Ξ reaches such a configuration within $O(\log^2 n) + O((\log n)/n) \cdot O(n) = O(\log^2 n)$ parallel time in expectation. □

Lemma 17. *Let C be any configuration in $\mathcal{C}_{\mathrm{all}}(P_{LL})$ and let $\Xi = \Xi_{P_{LL}}(C, \Gamma) = C_0, C_1, \ldots$. Execution Ξ reaches a configuration where all agents are in the third epoch within $O(\log n)$ parallel time with high probability and in expectation.*

Proof. Each agent increments its `epoch` every time it gets a new color until it reaches the third epoch (Lines 9, 26–27, and 31–32). Therefore, by Lemmas 4 and 7, at least one agent in V_B enters the third 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 enter the third epoch within $O(\log n)$ parallel time with high probability and in expectation by Lemma 1. □

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

Proof. First, by Lemmas 10 and 17, 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 12, execution Ξ reaches a configuration in $(\mathcal{B}_0 \cup \mathcal{B}_1) \cap \mathcal{C}_{\mathrm{color}}(2)$ 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 16. Finally, Lemmas 15 and 17 shows that starting from any configuration in $\mathcal{C}_{\mathrm{all}}(P_{LL})$, Ξ reaches a configuration where exactly one leader exists within $O(n)$ parallel time in expectation. To conclude, starting from initial configuration $C_{\mathrm{init},P_{LL}}$, execution Ξ reaches a configuration where the unique leader exists 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. That configuration must belong to $\mathcal{S}_{\mathcal{P}_{\mathcal{LL}}}$ because the number of leaders are monotonically non-increasing and no interaction brings a configuration with no leader in an execution of P_{LL} (Lemma 11). □

Algorithm 6. Symmetric $BackUp()$

Interaction between initiator a_0 and responder a_1:
1: **for all** $i \in \{0,1\}$ **do** $a_i.\mathtt{status}_{\mathrm{L}} = 1 - a_i.\mathtt{status}_{\mathrm{L}}$ **endfor**
2: **for all** $i \in \{0,1\}$ such that $a_i \in V_L$ **do**
3: **if** $a_i.\mathtt{tick}$ **then**
4: $a_i.\mathtt{ready} \leftarrow true$
5: **else if** $a_i.\mathtt{ready} \wedge a_{1-i} \in V_{F0}$ **then**
6: $a_i.\mathtt{level}_B \leftarrow \min(a_i.\mathtt{level}_B + 1, c_{\max})$
7: $a_i.\mathtt{ready} \leftarrow false$
8: **else if** $a_i.\mathtt{ready} \wedge a_{1-i} \in V_{F1}$ **then**
9: $a_i.\mathtt{ready} \leftarrow false$
10: **end if**
11: **end for**
12: **if** $a_0, a_1 \in V_A \wedge \exists i \in \{0,1\} : a_i.\mathtt{level}_B < a_{1-i}.\mathtt{level}_B$ **then**
13: $a_i.\mathtt{level}_B \leftarrow a_{1-i}.\mathtt{level}_B$
14: $a_i.\mathtt{leader} \leftarrow false$
15: **end if**
16: **if** $\exists i \in \{0,1\} : a_i \in V_{L0} \wedge a_{1-i} \in V_{L1}$ **then**
17: $a_{1-i}.\mathtt{leader} \leftarrow false$
18: **end if**

4 Discussion

A protocol is *symmetric* if its transition function T satisfies $\delta(p,q) = (p',q') \Rightarrow \delta(q,p) = (q',p')$ for any $p, q \in Q$. In other words, a symmetric protocol is a protocol that does not utilize the roles of the two agents at an interaction. In particular, when two agents with the same state p meet, they cannot updates their states to different states, that is, $\delta(p,p) = (q,r) \Rightarrow q = r$ for any $p \in Q$, which immediately follows from the definition of a symmetric protocol.

The proposed protocol P_{LL} described above is not symmetric, however, we can make it symmetric with keeping $O(\log n)$ stabilization time in expectation. Protocol P_{LL} performs asymmetric actions only for assignment of status (Lines 2-3), the simple leader election (Line 57 in $BackUp()$), and flipping fair and independent coins in $Quick\ Elimination()$, $Tournament()$, and $BackUp()$.

To assign the agents their statuses by symmetric transitions, we only have to add additional status Y and make the following three rules: $X \times X \to Y \times Y$, $Y \times Y \to X \times X$, $X \times Y \to A \times B$. 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: $J \times J \to K \times K$, $K \times K \to J \times J$, $J \times K \to F_0 \times F_1$. These rules guarantee 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).

Making $BackUp()$ symmetric is slightly complicated, thus we give a pseudocode of a symmetric version of $BackUp()$ in Algorithm 6. In the pseudocode, we denote the set of the

followers with coin status F_0 (resp., F_1) by V_{F0} (resp., V_{F1}). Similarly, we denote the set of the leaders whose $\texttt{status}_\texttt{L}$, a new variable introduced later, is 0 (resp., 1) by V_{L0} (resp.,V_{L1}). In $BackUp()$ described in Section 3.2.5, a leader makes a coin flip every time its `tick` is raised. However, a leader may not be able to make a coin flip that uses the coin status of a follower because the leader may interact not with a follower but with a leader when its `tick` is raised. We introduce a variable $\texttt{ready} \in \{false,\ true\}$ to overcome this issue. Initially, $\texttt{ready} = false$. It becomes $true$ when its `tick` is raised, and it remains $true$ until it interacts with a follower and makes a coin flip (Lines 2-11). To implement the simple leader election in $BackUp()$ by symmetric transitions, we introduce variable $\texttt{status}_\texttt{L} \in \{0, 1\}$. An agent flips its $\texttt{status}_\texttt{L}$, from 0 to 1 or from 1 to 0, every time it has an interaction (Line 1). When a leader with $\texttt{status}_\texttt{L} = 0$ and a leader with $\texttt{status}_\texttt{L} = 1$ meet, the latter becomes a follower (Lines 16-18). The $\texttt{status}_\texttt{L}$ of a follower is meaningless, but a follower also executes Line 1 for simplicity of the following analysis.

It is enough to prove the stabilization time to be $O(n \log n)$ parallel time in expectation because then, the stabilization time of the overall protocol is $O(\log n) + O(1/\log n) \cdot O(\log^2 n) + O(1/n) \cdot O(n \log n) = O(\log n)$ parallel time in expectation. We say that two agents have a *match* when they meet and their $\texttt{status}_\texttt{L}$s have different values. Note that this definition of having a match includes the case that one or two agents in an interaction are followers, for simplicity of the analysis. Clearly, exactly one leader exists in the population when all agents have a match at least once with each other after all agents enter the third epoch. Let u, v be any distinct agents. Consider a Markov chain with two states, where one state s_1 represents that the $\texttt{status}_\texttt{L}$s of u and v have the same value and the other state s_2 represents that they have the opposite values. At each step, regardless whether the state of the chain is s_1 or s_2, the chain changes its state if and only if either u or v but not both of them has an interaction at the step. This happens with probability $2(n-2)/\binom{n}{2}$ while the chain keeps its state with probability $1 - 2(n-2)/\binom{n}{2}$. When the state of the chain is s_2, u and v have an interaction with probability $1/\binom{n}{2} = O(1/n^2)$, at which they have a match. By a simple analysis of the chain, u and v have a match within $O(n^2)$ steps in expectation. Hence, from Markov's inequality, there exists a constant c such that u and v have a match within cn^2 steps with probability at least $1/2$. By repeating this observation, u and v have a match within $\lceil 3cn^2 \lg n \rceil$ steps with probability at least $1 - 2^{-3 \lg n} = 1 - O(1/n^3)$. By the union bound, all agents have a match with each other within $\lceil 3cn^2 \lg n \rceil$ steps with high probability. Hence, with every $O(n \log n)$ parallel time, exactly one leader is elected with high probability. Thus, the stabilization time is $O(n \log n)$ also in expectation.

5 Conclusion

In this paper, we gave a leader election protocol with logarithmic stabilization time and with logarithmic number of agent states in the population protocol model. Given a rough knowledge m of $\lg n$ such that $m \geq \lg n$ 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, where n is the number of agents in a population.

Acknowledgments

This work was supported by JSPS KAKENHI Grant No. 17K19977, 18K11167, 18K18000, 19H04085, 19K11826, and 20H04140 and JST SICORP Grant No. JPMJSC1606.

References

[1] Y. Sudo, F. Ooshita, T. Izumi, H. Kakugawa, and T. Masuzawa, "Brief anouncement: Logarithmic expected-time leader election in population protocol model," in *Proc. 38th ACM Symp. Princ. Distrib. Comput.*, 2019, pp. 60–62.

[2] Y. Sudo, F. Ooshita, T. Izumi, H. Kakugawa, and T. Masuzawa, "Logarithmic expected-time leader election in population protocol model," in *Proc. 21st Int. Symp. Stabilizing Saf. Secur. Distrib. Syst.*, 2019, pp. 323–337.

[3] D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta, "Computation in networks of passively mobile finite-state sensors," *Distrib. Comput.*, vol. 18, no. 4, pp. 235–253, 2006.

[4] D. Alistarh and R. Gelashvili, "Polylogarithmic-time leader election in population protocols," in *Proc. 42nd Int. Colloquium Automata Lang. Program.*, 2015, pp. 479–491.

[5] D. Alistarh, J. Aspnes, D. Eisenstat, R. Gelashvili, and R. L. Rivest, "Time-space trade-offs in population protocols," in *Proc. 28th Annu. ACM-SIAM Symp. Discrete Algorithms*, 2017, pp. 2560–2579.

[6] D. Alistarh, J. Aspnes, and R. Gelashvili, "Space-optimal majority in population protocols," in *Proc. 29th Annu. ACM-SIAM Symp. Discrete Algorithms*, 2018, pp. 2221–2239.

[7] L. Gąsieniec and G. Stachowiak, "Fast space optimal leader election in population protocols," in *Proc. 29th Annu. ACM-SIAM Symp. Discrete Algorithms*, 2018, pp. 2653–2667.

[8] L. Gąsieniec, G. Stachowiak, and P. Uznanski, "Almost logarithmic-time space optimal leader election in population protocols," in *Proc. 31st ACM Symp. Parallelism Algorithms Architectures*, 2019, pp. 93–102.

[9] O. Michail, P. G. Spirakis, and M. Theofilatos, "Simple and fast approximate counting and leader election in populations," in *Proc. 20th Int. Symp. Stabilizing Saf. Secur. Distrib. Syst.*, 2018, pp. 154–169.

[10] D. Doty and D. Soloveichik, "Stable leader election in population protocols requires linear time," *Distrib. Comput.*, vol. 31, no. 4, pp. 257–271, 2018.

[11] Y. Sudo and T. Masuzawa, "Leader election requires logarithmic time in population protocols," Parallel Processing Letters, vol. 30, no. 1, 2050005:pp. 1–13, 2020.

[12] Y. Sudo, J. Nakamura, Y. Yamauchi, F. Ooshita, H. Kakugawa, and T. Masuzawa, "Loosely-stabilizing leader election in a population protocol model," *Theor. Comput. Sci.*, vol. 444, pp. 100–112, 2012.

[13] Y. Sudo, F. Ooshita, H. Kakugawa, T. Masuzawa, A. K. Datta, and L. L. Larmore, "Loosely-stabilizing leader election with polylogarithmic convergence time," in *Proc. 22nd Int. Conf. Princ. Distrib. Syst.*, 2018, pp. 30:1–30:16.

[14] A. Bilke, C. Cooper, R. Elsässer, and T. 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 *Proc. 38th ACM Symp. Princ. Distrib. Comput.*, 2017, pp. 451–453.

[15] D. Alistarh and R. Gelashvili, "Recent algorithmic advances in population protocols," *ACM SIGACT News*, vol. 49, no. 3, pp. 63–73, 2018.

[16] R. Elsässer and T. Radzik, "Recent results in population protocols for exact majority and leader election," *Bull. EATCS*, vol. 3, 2018, Art. no. 126.

[17] P. Berenbrink, R. Elsässer, T. Friedetzky, D. Kaaser, P. Kling, and T. Radzik, "A population protocol for exact majority with $O(\log^{5/3} n)$ stabilization time and $\Theta(logn)$ states," in *Proc. 32nd Int. Symp. Distrib. Comput.*, pp. 10:1–10:18, 2018.

[18] D. Angluin, J. Aspnes, and D. Eisenstat, "Fast computation by population protocols with a leader," *Distrib. Comput.*, vol. 21, no. 3, pp. 183–199, 2008.

[19] M. Mitzenmacher and E. Upfal, *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge, U.K.: Cambridge Univ. Press, 2005.

Yuichi Sudo (Member, IEEE) received the BE, ME, and PhD degrees in computer science from Osaka University, Japan, in 2009, 2011, and 2015, respectively. He had worked at NTT Corporation and had been engaged in research on network security during 2011–2017. He is currently an assistant professor at the Graduate School of Information Science and Technology, Osaka University, Japan. His research interests include distributed algorithms, graph theory, and network security.

Fukuhito Ooshita (Member, IEEE) received the ME and DI degrees in computer science from Osaka University, Japan, in 2002 and 2006, respectively. He had been an assistant professor at the Graduate School of Information Science and Technology, Osaka University, Japan during 2003–2015. He is currently an associate professor of Graduate School of Science and Technology, Nara Institute of Science and Technology (NAIST), Japan. His research interests include parallel algorithms and distributed algorithms. He is a member of ACM, IEICE, and IPSJ.

Taisuke Izumi received the ME and DI degrees in computer science from Osaka University, Japan, in 2003 and 2006, respectively. During 2006–2008, he worked at Nagoya Institute of Technology, Japan as an assistant professor. He is currently an associate professor at the Graduate School of Engineering, Nagoya Institute of Technology, Japan. His research interests include algorithms and distributed systems. He is a member of IEICE, IPSJ, and ACM.

Hirotsugu Kakugawa (Member, IEEE) received the BE degree in engineering from Yamaguchi University, Japan, in 1990, and the ME and DE degrees in information engineering from Hiroshima University, Japan, in 1992, 1995, respectively. He is currently a professor of Ryukoku University, Japan. He is a member of the IEEE Computer Society and the Information Processing Society of Japan.

Toshimitsu Masuzawa (Member, IEEE) received the BE, ME, and DE degrees in computer science from Osaka University, Japan, in 1982, 1984, and 1987, respectively. He had worked at Osaka University, Japan during 1987–1994, and was an associate professor of Graduate School of Information Science, Nara Institute of Science and Technology (NAIST), Japan during 1994–2000. He is currently a professor at the Graduate School of Information Science and Technology, Osaka University, Japan. He was also a visiting associate professor with Department of Computer Science, Cornell University, Ithaca, New York between 1993–1994. His research interests include distributed algorithms, parallel algorithms and graph theory. He is a member of ACM, IEICE, and IPSJ.