Almost Time-Optimal Loosely-Stabilizing Leader Election on Arbitrary Graphs Without Identifiers in Population Protocols

Haruki Kanaya
Nara Institute of Science and Technology, Japan

Ryota Eguchi
Nara Institute of Science and Technology, Japan

Taisho Sasada
Nara Institute of Science and Technology, Japan

Michiko Inoue
Nara Institute of Science and Technology, Japan

Abstract

The population protocol model is a computational model for passive mobile agents. We address the leader election problem, which determines a unique leader on arbitrary communication graphs starting from any configuration. Unfortunately, self-stabilizing leader election is impossible to be solved without knowing the exact number of agents; thus, we consider loosely-stabilizing leader election, which converges to safe configurations in a relatively short time, and holds the specification (maintains a unique leader) for a relatively long time. When agents have unique identifiers, Sudo et al. (2019) proposed a protocol that, given an upper bound N for the number of agents n, converges in $O(mN \log n)$ expected steps, where m is the number of edges. When unique identifiers are not required, they also proposed a protocol that, using random numbers and given N, converges in $O(mN^2 \log N)$ expected steps. Both protocols have a holding time of $\Omega(e^{2N})$ expected steps and use $O(\log N)$ bits of memory. They also showed that the lower bound of the convergence time is $\Omega(mN)$ expected steps for protocols with a holding time of $\Omega(e^N)$ expected steps given N.

In this paper, we propose protocols that do not require unique identifiers. These protocols achieve convergence times close to the lower bound with increasing memory usage. Specifically, given N and an upper bound Δ for the maximum degree, we propose two protocols whose convergence times are $O(mN \log n)$ and $O(mN \log N)$ both in expectation and with high probability. The former protocol uses random numbers, while the latter does not require them. Both protocols utilize $O(\Delta \log N)$ bits of memory and hold the specification for $\Omega(e^{2N})$ expected steps.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Population protocols, Leader election, Loose-stabilization, Self-stabilization

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2024.37

Related Version *Full Version*: https://arxiv.org/abs/2411.03902

Funding The second author was supported by JSPS KAKENHI Grant Number 22H03569.

1 Introduction

The population protocol model, introduced by Angluin et al. [3], is a computational model widely recognized in distributed computing and applicable to passive mobile sensor networks, chemical reaction systems, and molecular calculations, etc. This model comprises n finite state machines (called *agents*), which form a network (called a *population*). Agents' states are updated through communication (called *interaction*) among a pair of agents. A simple

28th International Conference on Principles of Distributed Systems (OPODIS 2024).
Editors: Silvia Bonomi, Letterio Galletta, Etienne Rivière, and Valerio Schiavoni; Article No. 37; pp. 37:1–37:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

connected digraph $G = (V, E)$ (called a *communication graph*) determines the possibility of interaction among the agents. In this model, only one pair of agents interacts at each step. The interactions are determined by a uniform random scheduler.

The leader election problem is one of the most studied problems in population protocols. This problem involves agents electing a unique *leader* agent from the population and maintaining this unique leader forever. Angluin et al. [3] first studied this problem for complete graphs with designated common initial state. Under this assumption, many studies have been conducted [3, 6, 11, 17], and a time and space optimal protocol [6] has already been proposed. Several studies also exist for arbitrary graphs [1, 2], and a time-optimal protocol [2] has already been proposed.

The self-stabilizing leader election problem requires that agents start from any configuration, elect and externally maintain a unique leader agent. It is known that there is no self-stabilizing leader election protocol for arbitrary graphs [4] and complete graphs [8], and researchers have explored the problem in three ways. The first approach involves assuming that all agents initially know the exact number n of agents [7, 8, 22]. The second approach introduces an oracle that informs agents about the existence of leaders [5, 9, 10]. The third approach relaxes the requirement of maintaining a unique leader forever, introducing a *loosely-stabilizing* leader election problem, where agents start from any configuration, elect a unique leader within a short time, and maintain this leader for a long time. Sudo et al. [16] first addressed this problem on complete graphs. Subsequent studies have continued to explore this problem [12, 15, 18, 19, 20, 21] as follows and summarized in Table 1.

Sudo, Ooshita, Kakugawa, and Masuzawa [18] first addressed this problem for arbitrary graphs, and it is significantly improved by Sudo, Ooshita, Kakugawa, Masuzawa, Datta, and Lawrence [20] introducing a novel concept of Same Speed Timer. They proposed two protocols. The first protocol, $\mathcal{P}_{\mathrm{ID2}}$, assumes that agents have unique identifiers and are given N as initial knowledge. $\mathcal{P}_{\mathrm{ID2}}$ converges within $O(mN \log n)$ expected steps and holds the unique leader with $\Omega(Ne^{2N})$ expected steps, using $O(\log N)$ bits of memory. The second protocol, $\mathcal{P}_{\mathrm{RD2}}$, assumes that agents can make randomized transitions and is given N as initial knowledge. $\mathcal{P}_{\mathrm{RD2}}$ converges within $O(mN^2 \log N)$ expected steps and holds the unique leader with $\Omega(Ne^{2N})$ expected steps using $O(\log N)$ bits of memory. Sudo et al. also demonstrated that the lower bound of the convergence time is $\Omega(mN)$ steps for any loosely-stabilizing protocols with holding a unique leader $\Omega(e^N)$ expected steps.

Loosely-stabilizing leader election protocols without requiring unique identifiers or random numbers were proposed [19] and then improved [22]. The protocol, $\mathcal{P}_{\mathrm{AR}}$, given N and Δ as initial knowledge, converges within $O(mnD \log n + mN\Delta^2 \log N)$ expected steps and holds with $\Omega(Ne^N)$ expected steps using $O(\log N)$ bits of memory [22].

1.1 Our Contribution

In this paper, we propose a protocol $\mathcal{P}_{\mathrm{BC}}$ whose convergence time is nearly optimal on anonymous (without unique identifiers) arbitrary graphs, as supported by the lower bound [20]. The lower bound for the complete graph [12] is $\Omega(nN)$, but this is a special case, as the lower bound [20] is known to hold even when $m = \Theta(n^2)$. Given N and Δ, $\mathcal{P}_{\mathrm{BC}}$ converges within $O(mN \log n)$ steps if the transition is randomized, and $O(mN \log N)$ steps if the transition is deterministic[1], both in expectations and with high probability. The protocol holds the unique leader with $\Omega(Ne^{2N})$ expected steps and utilizes $O(\Delta \log N)$ bits of memory. The proposed

[1] Transition is said to be *deterministic* if it does not require random numbers in the transition. Though the transition is deterministic, we allow the protocol to exploit the randomness with which initiator and responder roles are chosen.

Table 1 Convergence and Holding Times for Loosely-Stabilizing Leader Election Protocols with Exponential Holding Times. n denotes the number of agents, N denotes the upper bound of n, m denotes the number of edges of the communication graph, D denotes the diameter of the communication graph, and Δ denotes the upper bound of the maximum degree of the communication graph. All protocols are given N as initial knowledge. Protocols with $*$ are also given Δ. The symbol † represents lower bounds of convergence time or memory usage for protocols with holding time of $\Omega(e^N)$.

	Graph	Convergence	Holding	Memory	Requisite
[16]	complete	$O(nN \log n)$	$\Omega(Ne^N)$	$O(\log N)$	–
[12]	complete	$O(nN)$	$\Omega(e^N)$	$O(\log N)$	–
[12]†	complete	$\Omega(nN)$	$\Omega(e^N)$	–	–
[12]†	complete	–	$\Omega(e^N)$	$\Omega(\log N)$	–
[18]$*$	arbitrary	$O(m\Delta N \log n)$	$\Omega(Ne^N)$	$O(\log N)$	agent identifiers
[18]$*$	arbitrary	$O(m\Delta^2 N^3 \log N)$	$\Omega(Ne^N)$	$O(\log N)$	random numbers
[20]	arbitrary	$O(mN \log n)$	$\Omega(Ne^{2N})$	$O(\log N)$	agent identifiers
[20]	arbitrary	$O(mN^2 \log N)$	$\Omega(Ne^{2N})$	$O(\log N)$	random numbers
[20]†	arbitrary	$\Omega(mN)$	$\Omega(e^N)$	–	–
[19]$*$	arbitrary	$O(mnD \log n + mN\Delta^2 \log N)$	$\Omega(Ne^N)$	$O(\log N)$	–
This$*$	arbitrary	$O(mN \log n)$	$\Omega(Ne^{2N})$	$O(\Delta \log N)$	random numbers
This$*$	arbitrary	$O(mN \log N)$	$\Omega(Ne^{2N})$	$O(\Delta \log N)$	–

$\mathcal{P}_{\mathrm{BC}}$ has better convergence time than SOTA self-stabilizing leader election protocol [22] which converges with $O(mn^2 D \log n)$ steps with requiring the knowledge of n.

To achieve the convergence time of $\mathcal{P}_{\mathrm{BC}}$, we utilize the Same Speed Timer proposed in $\mathcal{P}_{\mathrm{RD2}}$ [20], which requires two-hop coloring. The *self-stabilizing two-hop coloring* protocol was first studied by Angluin et al. [4], and further explored by Sudo et al. [20] (see Table 2). In this paper, we propose two new self-stabilizing two-hop coloring protocols; $\mathcal{P}_{\mathrm{LRU}}$ with randomized transitions, and $\mathcal{P'}_{\mathrm{LRU}}$ with deterministic transitions. Both protocols require N and Δ as initial knowledge. $\mathcal{P}_{\mathrm{LRU}}$ converges within $O(mn)$ steps, both in expectation and with high probability, and uses $O(\Delta \log N)$ bits of memory. $\mathcal{P'}_{\mathrm{LRU}}$ converges within $O(m(n + \Delta \log N))$ steps, both in expectation and with high probability, and also uses $O(\Delta \log N)$ bits of memory. In $\mathcal{P'}_{\mathrm{LRU}}$, agents generate random numbers independently from interactions among themselves. To ensure the independence among random numbers, we employ the *self-stabilizing normal coloring* protocol $\mathcal{P}_{\mathrm{NC}}$ to assign superiority or inferiority between adjacent agents. When interacting, only the superior agent uses the interaction to generate random numbers. $\mathcal{P}_{\mathrm{NC}}$ converges within $O(mn \log n)$ steps, both in expectation and with high probability, and utilizes $O(\log N)$ bits of memory.

2 Preliminaries

In this paper, $\mathbb{N}$ denotes the set of natural numbers no less than one, and $\log x$ refers to $\log_2 x$. If we use the natural logarithm, we explicitly specify the base e by writing $\log_e x$.

A population is represented by a simple connected digraph $G = (V, E)$, where V ($|V| \geq 2$) represents a set of agents, and $E \subseteq \{(u, v) \in V \times V \mid u \neq v\}$ represents the pairs of agents indicating potential interactions. An agent u can interact with an agent v if and only if $(u, v) \in E$, where u is the initiator and v is the responder. We assume G is symmetric, that is, if for every $(u, v) \in V \times V$, the preposition $(u, v) \in E \Rightarrow (v, u) \in E$ holds. We also denote $n = |V|$ and $m = |E|$. The diameter of G is denoted by D. The degree of agent u is

Table 2 List of Convergence Times for Self-Stabilizing Two-Hop Coloring Protocols on Arbitrary Graphs. n denotes the number of agents, N denotes the upper bound of n, m denotes the number of edges of the communication graph, δ denotes the maximum degree of the communication graph, and Δ denotes the upper bound of δ.

	Convergence	Memory	Knowledge	Requisite
Angluin et al. [4]	–	$O(\Delta^2)$	Δ	random numbers
Angluin et al. [4]	–	$O(\Delta^2)$	Δ	–
Sudo et al. [20]	$O(mn\delta \log n)$	$O(\log N)$	N	random numbers
$\mathcal{P}_{\mathrm{LRU}}$ (this)	$O(mn)$	$O(\Delta \log N)$	N, Δ	random numbers
$\mathcal{P}'_{\mathrm{LRU}}$ (this)	$O(mn + m\Delta \log N)$	$O(\Delta \log N)$	N, Δ	–

denoted by $\delta_u = |\{v \in V \mid (u,v) \in E \vee (v,u) \in E\}|$, and the maximum degree is denoted by $\delta = \max_{u \in V}\{\delta_u\}$. The upper bound N of n satisfies $N \geq n$, and the upper bound Δ of δ satisfies $\delta \leq \Delta \leq 2(N-1)$ [2].

A protocol $\mathcal{P}$ is defined as a 5-tuple (Q, Y, R, T, O), where Q represents the finite set of states of agents, Y represents the finite set of output symbols, $R \subset \mathbb{N}$ represents the range of random numbers, $T : Q \times Q \times R \to Q \times Q$ is the transition function, and $O : Q \to Y$ is the output function. When an initiator u, whose state is $p \in Q$, interacts with a responder v, whose state is $q \in Q$, each agent updates their states via the transition function using their current states and a random number $r \in R$ to $p', q' \in Q$ such that $(p', q') = T(p, q, r)$. An agent whose state is $p \in Q$ outputs $O(p) \in Y$. A protocol $\mathcal{P}$ is with deterministic transitions if and only if $\forall r, \forall r' \in R, \forall p, \forall q \in Q : T(p,q,r) = T(p,q,r')$ holds. Otherwise, a protocol $\mathcal{P}$ is with randomized transitions. The memory usage of protocol $\mathcal{P}$ is defined by $\lceil \log |Q| \rceil$ bits.

A *configuration* $C : V \to Q$ represents the states of all agents. The set of all configurations by protocol $\mathcal{P}$ is denoted by $\mathcal{C}_{all}(\mathcal{P})$. A configuration C transitions to C' by an interaction $e = (u,v)$ and a random number $r \in R$ if and only if $(C'(u), C'(v)) = T(C(u), C(v), r)$ and $\forall w \in V \setminus \{u,v\} : C'(w) = C(w)$ holds. Transitioning from a configuration C to C' by an interaction e and a random number r is denoted by $C \xrightarrow{e,r} C'$. A uniform random scheduler $\Gamma = \Gamma_0, \Gamma_1, \ldots$ determines which pair of agents interact at each step, where $\Gamma_t \in E$ (for $t \geq 0$) is a random variable satisfying $\forall (u,v) \in E, \forall t : \Pr(\Gamma_t = (u,v)) = 1/m$. An infinite sequence of random numbers $\Lambda = R_0, R_1, \ldots$ represents a random number generated at each step, where R_t (for $t \geq 0$) is a random variable satisfying $\forall r \in R, \forall t : \Pr(R_t = r) = 1/|R|$. Given an initial configuration $C_0 \in \mathcal{C}_{all}(\mathcal{P})$, a uniform random scheduler Γ, and a sequence of random numbers Λ, the execution of protocol $\mathcal{P}$ is denoted by $\Xi_{\mathcal{P}}(C_0, \Gamma, \Lambda) = C_0, C_1, \ldots$ where $C_t \xrightarrow{\Gamma_t, R_t} C_{t+1}$ (for $t \geq 0$) holds. If Γ and Λ are clear from the context, we may simply write $\Xi_{\mathcal{P}}(C_0)$. A set of configurations $\mathcal{S}$ is safe if and only if there is no configuration $C_i \notin \mathcal{S}$ $(i \in \mathbb{N})$ for any configuration $C_0 \in \mathcal{S}$ and any execution $\Xi(C_0) = C_0, C_1, \ldots$. A protocol is silent if and only if there is no state changed after reached safe configurations.

For a protocol $\mathcal{P}$ that solves a population protocol problem, the expected holding time and the expected convergence time are defined as follows. The specification of the problem, which is a required condition for an execution, is denoted by $\mathcal{SC}$. For any configuration $C \in \mathcal{C}_{all}(\mathcal{P})$, any uniform random scheduler Γ, and any infinite sequence of random numbers Λ, the expected number of steps that an execution $\Xi_{\mathcal{P}}(C, \Gamma, \Lambda)$ satisfies $\mathcal{SC}$ is defined as the expected holding time, denoted $\mathrm{EHT}_{\mathcal{P}}(C, \mathcal{SC})$. For any set of configurations $\mathcal{S} \subseteq \mathcal{C}_{all}(\mathcal{P})$, any configuration $C \in \mathcal{C}_{all}(\mathcal{P})$, any uniform random scheduler Γ, and any infinite sequence

[2] Given only N, $\delta \leq 2(N-1)$ holds, thus $\Delta \leq 2(N-1)$.

of random numbers Λ, the expected number of steps from the beginning of the execution $\Xi_{\mathcal{P}}(C, \Gamma, \Lambda)$ until the configuration reaches $\mathcal{S}$ is defined as the expected convergence time, denoted $\mathrm{ECT}_{\mathcal{P}}(C, \mathcal{S})$. A computation is considered to be finished with high probability if and only if the computation finishes with probability $1 - O(n^{-c})$ for $c \geq 1$.

The leader election problem requires that all agents output either L or F, where L represents a leader and F represents a follower. The specification of the leader election is denoted by LE. For an execution $\Xi_{\mathcal{P}}(C_0) = C_0, C_1, \ldots, C_x, \ldots$, the configurations $C_0, \ldots, C_x$ satisfy LE if and only if there is an agent u such that $\forall i \in [0, x] : O(C_i(u)) = L$, and $\forall i \in [0, x], \forall v \in V \setminus \{u\} : O(C_i(v)) = F$ holds.

▶ **Definition 1** (Loosely-stabilizing leader election[16])**.** *A protocol $\mathcal{P}$ is an (α, β)-loosely-stabilizing leader election protocol if and only if there exists a set of configurations $\mathcal{S} \subseteq \mathcal{C}_{all}(\mathcal{P})$ such that $\max_{C \in \mathcal{C}_{all}(\mathcal{P})} \mathrm{ECT}_{\mathcal{P}}(C, \mathcal{S}) \leq \alpha$ and $\min_{C \in \mathcal{S}} \mathrm{EHT}_{\mathcal{P}}(C, LE) \geq \beta$ holds.*

A protocol $\mathcal{P}$ is a self-stabilizing protocol of a problem if and only if there exists safe configurations that any execution starting from any safe configuration satisfies the specification of the problem (called closure), and any execution starting from any configuration includes a safe configuration reaches the safe configurations (called convergence).

▶ **Definition 2.** *A protocol $\mathcal{P}$ is a self-stabilizing normal coloring protocol if and only if there exists non-negative integer x such that for any configuration $C_0 \in \mathcal{C}_{all}(\mathcal{P})$, and the execution $\Xi_{\mathcal{P}}(C_0) = C_0, C_1, \ldots, C_x, \ldots$, the following condition holds: $\forall i \in \mathbb{N}, \forall v \in V : O(C_x(v)) = O(C_{x+i}(v))$ and $\forall v, \forall u \in V : (u, v) \in E \Rightarrow O(C_x(u)) \neq O(C_x(v))$.*

▶ **Definition 3.** *A protocol $\mathcal{P}$ is a self-stabilizing two-hop coloring protocols if and only if there exists non-negative integer x such that for any configuration $C_0 \in \mathcal{C}_{all}(\mathcal{P})$, and the execution $\Xi_{\mathcal{P}}(C_0) = C_0, C_1, \ldots, C_x, \ldots$, the following condition holds: $\forall i \in \mathbb{N}, \forall v \in V : O(C_x(v)) = O(C_{x+i}(v))$ and $\forall v, \forall u, \forall w \in V : (u, v) \in E \wedge (v, w) \in E \Rightarrow O(C_x(u)) \neq O(C_x(w))$.*

3 Self-Stabilizing Two-Hop Coloring

In this section, we introduce a self-stabilizing two-hop coloring protocol with randomized transitions $\mathcal{P}_{\mathrm{LRU}}$, alongside a deterministic self-stabilizing two-hop coloring protocol with deterministic transitions $\mathcal{P}'_{\mathrm{LRU}}$.

Two distinct agents $u, v \in V$ are called two-hop located if and only if there exists $w \in V$ such that $(u, w) \in E \wedge (v, w) \in E$. A graph is considered two-hop colored if and only if, for any pair of agents u and v that are two-hop located, u and v are assigned distinct colors.

The basic strategy is similar to that described by Angluin et al. [4] and Sudo et al. [20]. The differences lie in the methods for generating colors and the length of the array used to record the colors of interacted agents. Angluin et al. memorized all generated colors using an array of length $\Delta(\Delta - 1) + 1$, whereas Sudo et al. recorded only the most recent color. In the protocols, the agents record the last Δ colors.

We present a general strategy for color collision detection. When interacting two agents, they record each other's color with a common binary random stamp. If there is no color collision, when they interact again they find that they remember each other's color with the same stamp value. Assume that agents v and w have a color collision, that is, they are two-hop located with a common neighbor u and have the same color. Consider the scenario in which interactions occur in the order of (u, v), (u, w), (u, v), and u (resp. v) records v's (resp. u's) color with a stamp 0 and then u records w's color (it is also v's color) with a stamp 1. When u and v interact again, they notice they remember each other's color with different stamp values and detect the color collision.

In both protocols, each agent has arrays whose size are Δ to record the last Δ colors and their stamps. Both protocols are the same except for the way to generate colors. In $\mathcal{P}_{\mathrm{LRU}}$, agents generate colors by using the ability of generating uniform random numbers. In $\mathcal{P'}_{\mathrm{LRU}}$, agents generate colors by using the roles of initiator and responder. To generate x-digit binary random number, each agent generates a one random bit according to its role (initiator or responder) in each interaction, and repeats it x times. To ensure the independence of random numbers, only one agent can use the interaction to generate random numbers for each interaction. To solve this issue, we use normal coloring. A graph is considered normal colored if and only if agents u and v are different colors for any pair $(u, v) \in E$. We call agents' colors which are colored by normal coloring the *normal color*. To guarantee independency among random numbers, when two agents interact, the agent with larger normal color value can use the interaction to generate random numbers. Though this mechanism does not give a chance to generate random numbers to agents with smaller normal color, random numbers are used when two agents detect a color collision. In such cases, two random numbers are provided as new colors from an agent who has larger normal color value and already generated two or more numbers.

3.1 Protocol $\mathcal{P}_{\mathrm{LRU}}$

In this subsection, we introduce the randomized self-stabilizing two-hop coloring protocol $\mathcal{P}_{\mathrm{LRU}}$. Given N and Δ, the protocol $\mathcal{P}_{\mathrm{LRU}}$ achieves convergence within $O(mn)$ steps, both in expectation and with high probability, while requiring $O(\Delta \log N)$ bits of memory per agent.

An agent a in $\mathcal{P}_{\mathrm{LRU}}$ has four variables: $a.\texttt{hopcolor} \in \{1, \ldots, 8N^3\Delta^2\}$, $a.\texttt{prev} \in \{1, \ldots, 8N^3\Delta^2\}^\Delta$, $a.\texttt{stamp} \in \{0,1\}^\Delta$, and $a.\texttt{idx} \in \{0, \ldots, \Delta\}$. The variable $a.\texttt{hopcolor}$ represents the two-hop color of the agent. The variable $a.\texttt{prev}$ is an array that stores the last Δ colors interacted by the agent. The variable $a.\texttt{stamp}$ is an array of size Δ, with each entry being either 0 or 1, used to record the stamp associated with each color memorized. Lastly, $a.\texttt{idx}$ serves as a temporary index to locate the color of the interacting agent in $a.\texttt{prev}$.

$\mathcal{P}_{\mathrm{LRU}}$ is given by algorithm 1, and consists of four parts: i) reading memory, ii) collision detection, iii) saving colors, and iv) stamping.

i) Reading memory (lines 1–6) aims to find a color of the interacting partner in an array of recorded colors. For each agent a_i (where $i \in \{0,1\}$) interacting with another agent a_{1-i}, a_i searches $a_i.\texttt{prev}$ for $a_{1-i}.\texttt{hopcolor}$ and records the minimum index in $a_i.\texttt{idx}$ if exists, otherwise, sets $a_i.\texttt{idx}$ as 0.

ii) Collision detection (lines 7, and 13–16) aims to generate new colors when a stamp collision is detected. To address this, two uniform random numbers are generated from the range $[1, 8N^3\Delta^2]$. These numbers are then used to update $a_0.\texttt{hopcolor}$ and $a_1.\texttt{hopcolor}$ respectively.

iii) Saving colors (lines 8–11) aims to maintain arrays `prev` and `stamp` in a Least Recently Used (LRU) fashion.

iv) Stamping (lines 12, and 17–18) aims to generate a common binary stamp to two interacting agents, and to move a color and a stamp of this current interacting partner to the heads of arrays `prev` and `stamp`.

We have following theorems.

▶ **Theorem 4.** *Given the upper bound N and Δ, $\mathcal{P}_{\mathrm{LRU}}$ is a self-stabilizing two-hop coloring protocol with randomized transitions, and the convergence time is $O(mn)$ steps both in expectation and with high probability.*

Algorithm 1 Self-Stabilizing two-hop coloring $\mathcal{P}_{LRU}$.

```
Outout Function O: Each agent a outputs a.hopcolor.
when an initiator a_0 interacts with a responder a_1 do
    forall i ∈ {0,1} do
        a_i.idx ← 0
        for j ← 1 to Δ do
            if a_i.prev[j] = a_{1-i}.hopcolor then
                a_i.idx ← j
                break
    Generate_Color ()
    forall i ∈ {0,1} do
        if a_i.idx = 0 then  a_i.idx ← Δ
        for j ← a_i.idx − 1 downto 1 do
            (a_i.prev[j+1], a_i.stamp[j+1]) ← (a_i.prev[j], a_i.stamp[j])
    Generate_Bit ()
function Generate_Color():
    if a_0.idx > 0 ∧ a_1.idx > 0 ∧ a_0.stamp[a_0.idx] ≠ a_1.stamp[a_1.idx]) then
        generate two colors c_0, c_1 ∈ {1, ..., 8N^3Δ^2} uniformly at random
        (a_0.hopcolor, a_1.hopcolor) ← (c_0, c_1)
        a_0.idx ← a_1.idx ← 0
function Generate_Bit():
    generate bit b ∈ {0,1} uniformly at random
    (a_0.prev[1], a_1.prev[1], a_0.stamp[1], a_1.stamp[1]) ←
     (a_1.hopcolor, a_0.hopcolor, b, b)
```

▶ **Theorem 5.** *Given the upper bound N and Δ, $\mathcal{P}'_{\mathrm{LRU}}$ is a self-stabilizing two-hop coloring protocol with deterministic transitions, and converges to safe configurations within $O(m(n+\Delta\log N))$ steps both in expectation and with high probability.*

4 Loosely-Stabilizing Leader Election

In this section, we propose a loosely-stabilizing leader election protocol $\mathcal{P}_{\mathrm{BC}}$. $\mathcal{P}_{\mathrm{BC}}$ uses a self-stabilizing two-hop coloring protocol. Thus, if it uses $\mathcal{P}_{\mathrm{LRU}}$, $\mathcal{P}_{\mathrm{BC}}$ is with randomized transitions. If it uses $\mathcal{P}'_{\mathrm{LRU}}$, $\mathcal{P}_{\mathrm{BC}}$ is with deterministic transitions. In both cases, $\mathcal{P}_{\mathrm{BC}}$ holds a unique leader with $\Omega(Ne^{2N})$ expected steps and uses $O(\Delta\log N)$ bits of memory. Note that $\mathcal{P}_{\mathrm{BC}}$ (with randomized transitions) always generates random numbers deterministically like in $\mathcal{P}'_{\mathrm{LRU}}$ outside of two-hop coloring since it does not affect a whole complexity.

The basic strategy of leader election is as follows: i) All agents become followers. ii) Some candidates of a leader emerge, and the number of candidates becomes 1 with high probability. iii) If there are multiple leaders, return to i). Each agent a has 6 variables and 4 timers: $a.\mathtt{LF} \in \{\mathtt{B}, \mathtt{L}_0, \mathtt{L}_1, \mathtt{F}\}$, $a.\mathtt{type} \in \{1, ..., 2^{\lceil\log N\rceil+1}-1\}$, $a.\mathtt{id} \in \{1, ..., 2^{\lceil\log N^2\rceil+1}-1\}$, $a.\mathtt{color}$, $a.\mathtt{pcol}$, $a.\mathtt{rc} \in \{0,1\}$, $a.\mathtt{timer}_{\mathtt{LF}} \in [0, 2\mathtt{t}_{\mathrm{BC}}]$, $a.\mathtt{timer}_{\mathtt{KL}} \in [0, \mathtt{t}_{\mathrm{BC}}]$, $a.\mathtt{timer}_{\mathrm{V}} \in [0, 2\mathtt{t}_{\mathrm{BC}}]$, and $a.\mathtt{timer}_{\mathrm{E}} \in [0, 2\mathtt{t}_{\mathrm{BC}}]$. Here, $\mathtt{t}_{\mathrm{BC}}$ is a sufficiently large value for $\mathcal{P}_{\mathrm{BC}}$ to work correctly. A status of an agent a is represented by $a.\mathtt{LF}$ where $\mathtt{B}$ (leader candidate), $\mathtt{L}_0$ (leader mode), and $\mathtt{L}_1$ (duplication check mode) represent leaders and F represents a follower. The variable $a.\mathtt{type}$ is used for detecting multiple leaders. The variable $a.\mathtt{id}$ represents the identifier of leaders.

In $\mathcal{P}_{\mathrm{BC}}$, agents mainly use the broadcast (also called the epidemic and the propagation) to inform others of something. In a broadcast mechanism, information from one agent is repeatedly copied (with modification if needed) to agents when two agents interact. In order to detect the end of operations (including broadcasts) for all agents, the agents uses timers. The timers decrease by Larger Time Propagation and Same Speed Timer. Using Larger Time Propagation and Same Speed Timer, all timer values decrease gradually, almost synchronously. Larger Time Propagation means that when an agent u interacts with an agent v, $u.\mathtt{timer}$ (resp. $v.\mathtt{timer}$) is set to $\max(u.\mathtt{timer}, v.\mathtt{timer} - 1)$ (resp. $\max(v.\mathtt{timer}, u.\mathtt{timer} - 1)$). A variable $a.\mathtt{rc}$ is used to implement Same Speed Timer and represents whether a can decrease its own timers or not in the current interaction. Same Speed Timer decrease a timer value by 1 when an agent interacts the same agent continuously. For Same Speed Timer, after reaching $\mathcal{S}_{\mathrm{color}}$, the agents use the colors to determine whether the current partner is the last partner or not. A read-only variable $a.\mathtt{color}$ represents a color determined in the two-hop coloring. A variable $a.\mathtt{pcol}$ represents an agent' color that a interacted previously. The domains of $a.\mathtt{color}$ and $a.\mathtt{pcol}$ depend on a self-stabilizing two-hop coloring protocol. If we use $\mathcal{P}_{\mathrm{LRU}}$, $a.\mathtt{color}, a.\mathtt{pcol} \in \{1, \ldots, 8N^3\Delta^2\}$. If we use $\mathcal{P}'_{\mathrm{LRU}}$, $a.\mathtt{color}, a.\mathtt{pcol} \in \{0, \ldots, 2^{\lceil \log 8N^3\Delta^2 \rceil} - 1\}$.

4.1 Outline of $\mathcal{P}_{\mathrm{BC}}$

We show the outline of $\mathcal{P}_{\mathrm{BC}}$. We call a configuration satisfying certain conditions a phase. $\mathcal{P}_{\mathrm{BC}}$ mainly has 3 phases: i) Global Reset, ii) Leader Generation, iii) Leader Detection. We first explain the overview of the three phases with the roles of 4 timers using an example flow shown in Fig. 1, where the height of the timer represents the relative magnitude of its value.

i) Global Reset is the phase that resets all agents when some inconsistencies are detected. (In Fig. 1(a), multiple leaders are detected.) A configuration is in the Global Reset if there exists agents whose $\mathtt{timer}_{\mathtt{KL}} > 0$. When some inconsistency is detected, a Global Reset phase is started and the kill virus is created. The kill virus makes an agent a follower, sets the agent's identifier to 1 (a tentative value before generating id), and also erases the search virus. The presence of kill virus is represented as a positive value of $\mathtt{timer}_{\mathtt{KL}}$, which serves as timer to live (TTL). When the Global Reset phase begins, some agents' $\mathtt{timer}_{\mathtt{KL}}$ are set to the maximum value and spread to all agents with decreasing the values as shown in Fig. 1(b). Eventually, $\mathtt{timer}_{\mathtt{KL}}$ will become 0, and the Global Reset phase will be finished. While $\mathtt{timer}_{\mathtt{KL}} > 0$, $\mathtt{timer}_{\mathrm{LF}}$ takes its maximum value and $\mathtt{timer}_{\mathrm{V}}$ takes 0.

ii) Leader Generation is the phase where agents generate leaders. In $\mathcal{P}_{\mathrm{BC}}$, a leader keeps a values of $\mathtt{timer}_{\mathtt{LF}}$ to its maximum value, while followers propagate the value with Larger Time Propagation and Same Speed Timer. If there is no leader (Fig. 1(c)), values of $\mathtt{timer}_{\mathtt{LF}}$ for some followers eventually become 0. Then these followers become candidates (B) (Fig. 1(d)). Each candidate for leaders generates a random number as an identifier using interactions. The way to generate random numbers is described in Section 3. The candidates broadcast their ids to all agents, and become a follower (F) if it encounters a larger id value. While generating an identifier (id), $\mathtt{timer}_{\mathrm{LF}}$ takes its maximum value, and it gradually decreases after generating id. When $\mathtt{timer}_{\mathrm{LF}}$ of a candidate becomes 0, it becomes a leader ($\mathtt{L}_0$). When there are no candidates for leaders, Leader Generation is finished.

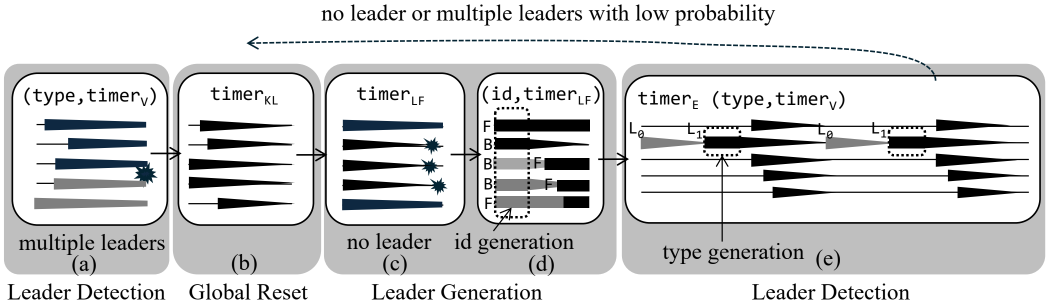

Figure 1 An example flow of $\mathcal{P}_{\mathrm{BC}}$.

iii) Leader Detection is the phase where leaders determine whether there are multiple leaders or not. The leaders generate search viruses periodically using $\mathtt{timer}_{\mathrm{E}}$. When $\mathtt{timer}_{\mathrm{E}}$ becomes 0, the agent start generating a `type` of search virus. If there are multiple leaders, leaders generate search viruses almost simultaneously thanks to Larger Time Propagation. When leaders generate search viruses, they generate random numbers using interactions to determine the type of search virus. The type of search virus with its TTL $\mathtt{timer}_{\mathrm{V}}$ spreads to all agents. When two different types of search viruses meet, agents create the kill virus and move to the Global Reset phase (Fig. 1(a)). While, if there is a unique leader, one search virus is periodically generated and expired (Fig. 1(e)).

Phases circulates Global Reset, Leader Generation, and Leader Detection in this order. If there exists a unique leade , the configuration stays in Leader Detection phase with high probability. Otherwise, the phase moves to Global Reset phase. Timers $\mathtt{timer}_{\mathtt{KL}}$ and $\mathtt{timer}_{\mathtt{LF}}$ are used for Global Reset and Leader Generation phases to have enough steps.

4.2 Details of $\mathcal{P}_{\mathrm{BC}}$

$\mathcal{P}_{\mathrm{BC}}$ is given by Algorithm 2, Algorithm 3, and Algorithm 4. We explain the details of $\mathcal{P}_{\mathrm{BC}}$. $\mathcal{P}_{\mathrm{BC}}$ has five parts: i) Two-hop coloring, ii) Timer Count Down, iii) Reset, iv) Leader Generation, v) Leader Detection. The relationship between the three phases and five parts is as follows. The Global Reset phase corresponds to the Reset part, the Leader Generation phase to the Generate part, and the Leader Detection phase to the Detect part. The Timer Count Down part operates throughout all phases, while the Two-Hop Coloring part is completed before the phases begin. Throughout this explanation, we consider when an initiator a_0 interacts with a responder a_1.

i) In line 1, the agents execute the two-hop coloring protocol $\mathcal{P}_{\mathrm{LRU}}$ or $\mathcal{P}'_{\mathrm{LRU}}$.

ii) Timer Count Down (lines 2, 4–12, and 15–22) aims to increase or decrease timers. First, the interacting agents determine whether the current partner is the same as the last partner in REPEAT_CHECK (lines 2, and 17–20) to implement Same Speed Timer. After that, each agent saves the current partner's color to its own `pcol`. Then, agents decrease timers if $a_i.\mathtt{rc} = 1$ holds. $\mathtt{timer}_{\mathtt{KL}}$ (lines 4–5) and $\mathtt{timer}_{\mathrm{E}}$ (lines 6–7) are handled by Larger Time Propagation and Count Down. That is, for $i \in \{0, 1\}$, $a_i.\mathtt{timer}_{\mathtt{KL}}$ is set to $\max(a_i.\mathtt{timer}_{\mathtt{KL}}, a_{1-i}.\mathtt{timer}_{\mathtt{KL}} - 1)$, and if $a_i.\mathtt{rc} = 1$ holds, $a_i.\mathtt{timer}_{\mathtt{KL}}$ is decreased by 1 (resp. $\mathtt{timer}_{\mathrm{E}}$). If both interacting agents are not candidates ($\mathtt{B}$), they increase or decrease $\mathtt{timer}_{\mathtt{LF}}$ like $\mathtt{timer}_{\mathtt{KL}}$ (lines 8–10). For $i \in \{0, 1\}$, if a_i is a leader ($\mathtt{L}_0$ or $\mathtt{L}_1$), $a_i.\mathtt{timer}_{\mathtt{LF}}$ is set to $\mathtt{t}_{\mathrm{BC}}$ (lines 11–12). If a_0 or a_1 is a candidate ($\mathtt{B}$), $\mathtt{timer}_{\mathtt{LF}}$ increases or decreases in Leader Generation. $\mathtt{timer}_{\mathrm{V}}$ increases or decreases in Leader Detection.

Algorithm 2 Loosely-Stabilizing Leader Election Protocol $\mathcal{P}_{\mathrm{BC}}$ (1/3).

```
   Output function O: An agent a outputs L if a.LF ∈ {B, L_0, L_1}, otherwise, F.
   when an initiator a_0 interacts with a responder a_1 do
1      Execute self stabilizing two-hop coloring protocol
2      REPEAT_CHECK()
3      if ∃i ∈ {0,1} : a_i.timer_KL > 0 then Reset()
4      LARGER_TIME_PROPAGATE(KL)
5      COUNT_DOWN(KL)
6      LARGER_TIME_PROPAGATE(E)
7      COUNT_DOWN(E)
8      if a_0.LF, a_1.LF ∈ {L_0, L_1, F} then
9          LARGER_TIME_PROPAGATE(LF)
10         COUNT_DOWN(LF)
11     forall i ∈ {0,1} do
12         if a_i.LF ∈ {L_0, L_1} then a_i.timer_LF ← t_BC
13     GenerateLeader()
14     Detect()
   function LARGER_TIME_PROPAGATE(x):
15     if ∃i ∈ {0,1} : a_i.timer_x < a_{1-i}.timer_x then
16         a_i.timer_x ← a_{1-i}.timer_x − 1
   function REPEAT_CHECK():
17     forall i ∈ {0,1} do
18         if a_i.pcol = a_{1-i}.color then a_i.rc ← 1
19         else a_i.rc ← 0
20         a_i.pcol ← a_{1-i}.color
   function COUNT_DOWN(x):
21     forall i ∈ {0,1} do
22         if a_i.rc = 1 then a_i.timer_x ← max(0, a_i.timer_x − 1)
   function Reset():
23     forall i ∈ {0,1} do
24         (a_i.LF, a_i.id, a_i.timer_V) ← (F, 1, 0)
```

- **iii)** Reset (lines 3, and 23–24) aims to reset the population when some inconsistency is detected. For $i \in \{0,1\}$, if $a_i.\mathtt{timer}_{\mathtt{KL}} > 0$ holds, a_i sets $(a_i.\mathtt{LF}, a_i.\mathtt{id}, a_i.\mathtt{timer}_{\mathrm{V}})$ to $(\mathtt{F}, 1, 0)$. In other words, a_i becomes a follower, its identifier becomes 1, and a_i erases the search virus. This Reset is happened when and only when there are multiple leaders and candidates' id have not been reset to 1. Specifically, when different search virus meets (lines 60–61, 65–66, 69–70), and when candidates' id have not been reset (lines 27–28).
- **iv)** Generate Leader (lines 13, and 25–40) aims to generate a new leader when there are no leaders. For $i \in \{0,1\}$, if $a_i.\mathtt{timer}_{\mathtt{KL}} > 0$ holds, each agent sets $a_i.\mathtt{LF}$ to $\mathtt{t}_{\mathrm{BC}}$ to prevent starting Leader Generation during the Global Reset phase (lines 25–26). Firstly, for $i \in \{0,1\}$, if $a_i.\mathtt{LF}$ becomes 0, a_i determines that there are no leaders, and becomes a candidate for leaders (B) and sets $a_i.\mathtt{timer}_{\mathtt{LF}}$ to $2\mathtt{t}_{\mathrm{BC}}$ (lines 27–29). At this time, if $a_i.\mathtt{id}$ is not 1 (i.e., , it has not been reset), a_i sets $a_i.\mathtt{timer}_{\mathtt{KL}}$ to $\mathtt{t}_{\mathrm{BC}}$ and moves to the Global

Algorithm 3 Loosely-Stabilizing Leader Election Protocol $\mathcal{P}_{BC}$ (2/3).

function GenerateLeader():

25 **if** $\exists i \in \{0,1\} : a_i.\mathtt{timer}_{\mathtt{KL}} > 0$ **then**

26 $a_0.\mathtt{timer}_{\mathtt{LF}} \leftarrow a_1.\mathtt{timer}_{\mathtt{LF}} \leftarrow \mathtt{t}_{\mathrm{BC}}$ // prevent starting GenerateLeader

27 **if** $\exists i \in \{0,1\} : a_i.\mathtt{LF} = \mathtt{F} \land a_i.\mathtt{timer}_{\mathtt{LF}} = 0$ **then**

28 **if** $a_i.\mathtt{id} \neq 1$ **then** $a_0.\mathtt{timer}_{\mathtt{KL}} \leftarrow a_1.\mathtt{timer}_{\mathtt{KL}} \leftarrow \mathtt{t}_{\mathrm{BC}}$ // id hasn't been reset

29 **else** $(a_i.\mathtt{LF}, a_i.\mathtt{timer}_{\mathtt{LF}}) \leftarrow (\mathtt{B}, 2\mathtt{t}_{\mathrm{BC}})$ // a new candidate is created

30 **if** $\forall i \in \{0,1\} : a_i.\mathtt{LF} = \mathtt{B} \land a_i.\mathtt{id} < 2^{\lceil \log N^2 \rceil}$ **then**

31 $(a_1.\mathtt{LF}, a_1.\mathtt{id}, a_1.\mathtt{timer}_{\mathtt{LF}}) \leftarrow (\mathtt{F}, 1, \mathtt{t}_{\mathrm{BC}})$

32 **if** $\exists i \in \{0,1\} : a_i.\mathtt{LF} = \mathtt{B} \land a_i.\mathtt{id} < 2^{\lceil \log N^2 \rceil}$ **then**

33 $(a_i.\mathtt{id}, a_i.\mathtt{timer}_{\mathtt{LF}}) \leftarrow (2a_i.\mathtt{id} + i, 2\mathtt{t}_{\mathrm{BC}})$

34 **if** $a_0.\mathtt{LF} \in \{\mathtt{F},\mathtt{B}\} \land a_1.\mathtt{LF} \in \{\mathtt{F},\mathtt{B}\} \land (\exists i \in \{0,1\} : ((a_i.\mathtt{LF} = \mathtt{B} \land a_i.\mathtt{id} > 2^{\lceil \log N^2 \rceil}) \lor a_i.\mathtt{LF} = \mathtt{F}) \land a_i.\mathtt{id} > a_{1-i}.\mathtt{id})$ **then**

35 $(a_{1-i}.\mathtt{LF}, a_{1-i}, \mathtt{id}) \leftarrow (a_i.\mathtt{LF}, a_i.\mathtt{id})$

36 **if** $\exists i \in \{0,1\} : a_i.\mathtt{LF} = \mathtt{F} \land a_{1-i}.\mathtt{LF} = \mathtt{B}$ **then**

37 $a_i.\mathtt{timer}_{\mathtt{LF}} \leftarrow \mathtt{t}_{\mathrm{BC}} - 1$ // consider a_{1-i} as a leader

38 **forall** $i \in \{0,1\}$ **do**

39 **if** $a_i.\mathtt{LF} = \mathtt{B} \land a_i.\mathtt{rc} = 1$ **then** $a_i.\mathtt{timer}_{\mathtt{LF}} \leftarrow \max(0, a_i.\mathtt{timer}_{\mathtt{LF}} - 1)$

40 **if** $a_i.\mathtt{LF} = \mathtt{B} \land a_i.\mathtt{timer}_{\mathtt{LF}} = 0$ **then** $(a_i.\mathtt{LF}, a_i.\mathtt{timer}_{\mathtt{LF}}) \leftarrow (\mathtt{L}_0, \mathtt{t}_{\mathrm{BC}})$

Reset phase (line 29). Secondly, candidates ($\mathtt{B}$) generate random numbers as their own identifiers ($\mathtt{id}$). For each interaction, if the candidate u is an initiator, $u.\mathtt{id}$ is updated to $2u.\mathtt{id}$; otherwise, $u.\mathtt{id}$ is updated to $2u.\mathtt{id} + 1$ until $u.\mathtt{id}$ becomes no less than $2^{\lceil \log N^2 \rceil}$ (lines 32–33). For the independence of random numbers, if both agents are candidates and generating random numbers, the responder becomes a follower ($\mathtt{F}$) by resetting $\mathtt{id}$ to 1 (lines 30–31). Thirdly, if a candidate's $\mathtt{id}$ becomes no less than $2^{\lceil \log N^2 \rceil}$, the candidate starts to broadcast its own $\mathtt{id}$ to other agents (lines 34–35). This broadcast allows all agents to know the maximum $\mathtt{id}$ of candidates. For $i \in \{0,1\}$, if a_i is a candidate and $a_i.\mathtt{id} < a_{1-i}.\mathtt{id}$ holds, a_i becomes a follower ($\mathtt{F}$) and sets $a_i.\mathtt{id}$ to $a_{1-i}.\mathtt{id}$. For $i \in \{0,1\}$, if a_i is a follower and $a_i.\mathtt{id} > a_{1-i}.\mathtt{id}$ holds, a_{1-i} sets $a_{1-i}.\mathtt{id}$ to $a_i.\mathtt{id}$. To avoid generating new candidates when there are candidates in the population, for $i \in \{0,1\}$, if a_i is a follower and a_{1-i} is a candidate, a_i sets $a_i.\mathtt{timer}_{\mathtt{LF}}$ to $\mathtt{t}_{\mathrm{BC}} - 1$ (lines 36–37). That is, we consider a_{1-i} has $\mathtt{timer}_{\mathtt{LF}} = \mathtt{t}_{\mathrm{BC}}$ virtually. Eventually, all agents' $\mathtt{id}$s become the same, and most candidates become followers (and some candidates remain). The candidates measure until all candidates finish generating the $\mathtt{id}$ and $\mathtt{id}$s are broadcast for a sufficiently long time using $\mathtt{timer}_{\mathtt{LF}}$. A candidate decreases $\mathtt{timer}_{\mathtt{LF}}$ by 1 if the agent's $\mathtt{rc}$ is 1 (lines 39). Finally, when a candidate's $\mathtt{timer}_{\mathtt{LF}}$ becomes 0, the candidate becomes a new leader ($\mathtt{L}_0$) and sets $\mathtt{timer}_{\mathtt{LF}}$ to $\mathtt{t}_{\mathrm{BC}}$ (lines 40). The range of generated identifiers ($\mathtt{id}$) is $[2^{\lceil \log N^2 \rceil}, 2^{\lceil \log N^2 \rceil + 1})$, so there exists a unique leader with high probability.

v) Detect (lines 41–66) aims to determine whether there are multiple leaders or not. Leaders generate search viruses every time their $\mathtt{timer}_{\mathrm{E}}$ becomes 0. If a_0 or a_1 is a candidate ($\mathtt{B}$), agents set their $\mathtt{timer}_{\mathrm{E}}$ to $2\mathtt{t}_{\mathrm{BC}}$ to prevent generating a search virus (lines 41–42). Firstly, for $i \in \{0,1\}$, if a_i is a leader ($\mathtt{L}_0$ or $\mathtt{L}_1$) whose $\mathtt{timer}_{\mathrm{E}}$ becomes 0, a_i becomes

Algorithm 4 Loosely-Stabilizing Leader Election Protocol $\mathcal{P}_{\mathrm{BC}}$ (3/3).

function Detect():
41 **if** $\exists i \in \{0,1\} : a_i.\mathrm{LF} = \mathtt{B}$ **then**
42 $a_0.\mathtt{timer}_{\mathrm{E}} \leftarrow a_1.\mathtt{timer}_{\mathrm{E}} \leftarrow 2\mathtt{t}_{\mathrm{BC}}$ // prevent starting Detect
43 **forall** $i \in \{0,1\}$ **do**
44 **if** $a_i.\mathrm{LF} \in \{\mathtt{L}_0, \mathtt{L}_1\} \wedge a_i.\mathtt{timer}_{\mathrm{E}} = 0$ **then**
45 $(a_i.\mathrm{LF}, a_i.\mathtt{type}, a_i.\mathtt{timer}_{\mathrm{E}}) \leftarrow (\mathtt{L}_1, 1, 2\mathtt{t}_{\mathrm{BC}})$ // start the type generation
46 **if** $a_i.\mathrm{LF} = \mathtt{L}_1 \wedge a_i.\mathtt{type} < 2^{\lceil \log N \rceil}$ **then**
47 $(a_i.\mathtt{type}, a_i.\mathtt{timer}_{\mathrm{E}}) \leftarrow (2a_i.\mathtt{type} + i, 2\mathtt{t}_{\mathrm{BC}})$
48 **if** $a_i.\mathtt{type} \geq 2^{\lceil \log N \rceil}$ **then**
49 $a_i.\mathtt{timer}_{\mathrm{V}} \leftarrow 2\mathtt{t}_{\mathrm{BC}}$
50 **if** $\exists i \in \{0,1\} : a_i.\mathrm{LF} = \mathtt{F} \wedge a_{1-i}.\mathrm{LF} \in \{\mathtt{L}_0, \mathtt{L}_1\}$ **then**
51 **if** $a_i.\mathtt{timer}_{\mathrm{V}} > 0 \wedge (a_{1-i}.\mathrm{LF} = \mathtt{L}_0 \vee a_{1-i}.\mathtt{type} < 2^{\lceil \log N \rceil} \vee$ $a_i.\mathtt{type} \neq a_{1-i}.\mathtt{type})$ **then**
52 $a_0.\mathtt{timer}_{\mathtt{KL}} \leftarrow a_1.\mathtt{timer}_{\mathtt{KL}} \leftarrow \mathtt{t}_{\mathrm{BC}}$ // different types are detected
53 **else if** $a_i.\mathtt{timer}_{\mathrm{V}} = 0 \wedge a_{1-i}.\mathrm{LF} = \mathtt{L}_1 \wedge a_{1-i}.\mathtt{timer}_{\mathrm{V}} > 0$ **then**
54 $(a_i.\mathtt{type}, a_i.\mathtt{timer}_{\mathrm{V}}) \leftarrow (a_{1-i}.\mathtt{type}, a_{1-i}.\mathtt{timer}_{\mathrm{V}} - 1)$
55 **else if** $a_0.\mathrm{LF} = \mathtt{F} \wedge a_1.\mathrm{LF} = \mathtt{F}$ **then**
56 **if** $a_0.\mathtt{timer}_{\mathrm{V}} > 0 \wedge a_1.\mathtt{timer}_{\mathrm{V}} > 0 \wedge a_0.\mathtt{type} \neq a_1.\mathtt{type}$ **then**
57 $a_0.\mathtt{timer}_{\mathtt{KL}} \leftarrow a_1.\mathtt{timer}_{\mathtt{KL}} \leftarrow \mathtt{t}_{\mathrm{BC}}$ // different types are detected
58 **else if** $\exists i \in \{0,1\} : a_i.\mathtt{timer}_{\mathrm{V}} = 0 \wedge a_{1-i}.\mathtt{timer}_{\mathrm{V}} > 0$ **then**
59 $(a_i.\mathtt{type}, a_i.\mathtt{timer}_{\mathrm{V}}) \leftarrow (a_{1-i}.\mathtt{type}, a_{1-i}.\mathtt{timer}_{\mathrm{V}} - 1)$
60 **else if** $a_0.\mathrm{LF} \in \{\mathtt{L}_0, \mathtt{L}_1\} \wedge a_1.\mathrm{LF} \in \{\mathtt{L}_0, \mathtt{L}_1\}$ **then**
61 $a_0.\mathtt{timer}_{\mathtt{KL}} \leftarrow a_1.\mathtt{timer}_{\mathtt{KL}} \leftarrow \mathtt{t}_{\mathrm{BC}}$ // multiple leaders are detected
62 LARGER_TIME_PROPAGATE(V)
63 COUNT_DOWN(V)
64 **forall** $i \in \{0,1\}$ **do**
65 **if** $a_i.\mathtt{timer}_{\mathrm{V}} > 0$ **then** $a_i.\mathtt{timer}_{\mathrm{E}} \leftarrow 2\mathtt{t}_{\mathrm{BC}}$ // prevent restarting Detect
66 **if** $a_i.\mathrm{LF} = \mathtt{L}_1 \wedge a_i.\mathtt{timer}_{\mathrm{E}} < \mathtt{t}_{\mathrm{BC}}/2$ **then** $a_i.\mathrm{LF} \leftarrow \mathtt{L}_0$

$\mathtt{L}_1$ and starts generating random numbers to get the type of search virus (lines 44–45). At the beginning of generating random numbers, a_i sets $a_i.\mathtt{type}$ to 1. The way of generating random numbers is the same as id generation. While generating random numbers, a leader sets own $\mathtt{timer}_{\mathrm{E}}$ to $2\mathtt{t}_{\mathrm{BC}}$ to inform that there exist agents generating random numbers (lines 46–47). When a leader finished generating random numbers, the leader sets own $\mathtt{timer}_{\mathrm{V}}$ to $2\mathtt{t}_{\mathrm{BC}}$ (lines 48–49). Secondly, agents detect multiple leaders if there are multiple leaders. $\mathtt{L}_1$ broadcasts the generated search virus to all agents via some agents until $\mathtt{timer}_{\mathrm{V}}$ becomes 0 (lines 50–59). If a follower having search virus and a leader not having search virus interact except the cases their types are same, they set $\mathtt{timer}_{\mathtt{KL}}$ to $\mathtt{t}_{\mathrm{BC}}$ and the phase moves to Global Reset (lines 51–52). If a follower not having search virus and a leader having search virus interact, the follower set own $\mathtt{timer}_{\mathrm{V}}$ to the leader's $\mathtt{timer}_{\mathrm{V}} - 1$ and set own type to the leader's type (lines 53–54). If a_0 and a_1 are followers and they have different types of search viruses, they set $\mathtt{timer}_{\mathtt{KL}}$ to $\mathtt{t}_{\mathrm{BC}}$ and move to Global Reset phase (lines 56–57). If a_0 and a_1 are followers and there exists a_i, a_{1-i} agents satisfying $a_i.\texttt{timer}_\mathrm{V} = 0$ and $a_{1-i}.\texttt{timer}_\mathrm{V} > 0$ for $i \in \{0, 1\}$, $a_i.\texttt{timer}_\mathrm{V}$ is set to $a_{1-i}.\texttt{timer}_\mathrm{V} - 1$ and $a_i.\texttt{type}$ is set to $a_{1-i}.\texttt{type}$ (lines 58–59). If a_0 and a_1 are leaders, they set $\texttt{timer}_\texttt{KL}$ to $\texttt{t}_\mathrm{BC}$ and move to Global Reset phase (lines 60–61). Finally, both agents' $\texttt{timer}_\mathrm{V}$ run Larger Time Propagation and decrease by 1 if $a_i.\texttt{rc} = 1$ holds (lines 62–63). For $i \in \{0, 1\}$, if $a_i.\texttt{timer}_\mathrm{V} > 0$ holds, $a_i.\texttt{timer}_\mathrm{E}$ is set to $2\texttt{t}_\mathrm{BC}$ to prevent generating a new search virus when there is a search virus in the population (line 65). For $i \in \{0, 1\}$, if a_i is a leader and $a_i.\texttt{timer}_\mathrm{E}$ becomes less than $\texttt{t}_\mathrm{BC}/2$, a_i becomes $\texttt{L}_0$ (line 66). The range of generating random numbers of types is $[2^{\lceil \log N \rceil}, 2^{\lceil \log N \rceil + 1})$, so when there are multiple leaders in the population, the types generated by leaders are not the same with high probability.

▶ **Lemma 6.** *For any execution, all candidates'* $\texttt{id}$ *no less than* $2^{\lceil \log N^2 \rceil}$ *are independent and uniform if they are started to be generated during the execution. All leaders'* $\texttt{type}$ *no less than* $2^{\lceil \log N \rceil}$ *are independent and uniform if they are generated from the beginning of this execution.*

4.3 Analysis

In this subsection, we analyze the expected convergence time and the expected holding time of $\mathcal{P}_\mathrm{BC}$. We assume $\tau \geq \max(2d, \lceil \log N \rceil/2, 15 + 3\log n)$, and $\texttt{t}_\mathrm{BC} = 16\tau$. We will prove the following equations under these assumptions:

- $\max_{C \in \mathcal{S}_\mathrm{color}} \mathrm{ECT}_{\mathcal{P}_\mathrm{BC}}(C, \mathcal{S}_\mathrm{LE}) = O(m\tau \log n)$.
- $\min_{C \in \mathcal{S}_\mathrm{LE}} \mathrm{EHT}_{\mathcal{P}_\mathrm{BC}}(C, LE) = \Omega(\tau e^\tau)$.

Here, $\mathcal{S}_\mathrm{color}$ and $\mathcal{S}_\mathrm{LE}$ are the sets of configurations described later.

We define the sets of configurations to prove the above equations:

- $\mathcal{S}_\mathrm{color}$ is the safe configurations of the self-stabilizing two-hop coloring.
- $\mathcal{S'}_\mathrm{color} \subset \mathcal{S}_\mathrm{color}$ is the set of configurations where each agent's $\texttt{pcol}$ is the same as the last interacted agent's color.
- $\mathcal{KL}_\mathrm{zero} = \{C \in \mathcal{S'}_\mathrm{color} \mid \forall v \in V : C(v).\texttt{timer}_\texttt{KL} = 0\}$.
- $\mathcal{B}_\mathrm{no} = \{C \in \mathcal{S'}_\mathrm{color} \mid \forall v \in V : C(v).\texttt{LF} \neq \texttt{B}\}$.
- $\mathcal{L}_\mathrm{one} = \{C \in \mathcal{S'}_\mathrm{color} \mid |\{v \in V \mid C(v).\texttt{LF} \in \{\texttt{L}_0, \texttt{L}_1\}\}| = 1\}$.
- $\mathcal{LF}_\mathrm{qua} = \{C \in \mathcal{S'}_\mathrm{color} \mid \forall v \in V : C(v).\texttt{LF} \neq \texttt{B} \Rightarrow C(v).\texttt{timer}_\texttt{LF} \geq \texttt{t}_\mathrm{BC}/2\}$.
- $\mathcal{L}_\mathrm{v1} = \{C \in \mathcal{S'}_\mathrm{color} \mid \exists v \in V : C(v).\texttt{LF} = \texttt{L}_1\}$.
- $\mathcal{V}_\mathrm{clean} = \{C \in \mathcal{S'}_\mathrm{color} \mid \forall v \in V : C(v).\texttt{timer}_\mathrm{V} = 0\}$.
- $\mathcal{V}_\mathrm{make} = \{C \in \mathcal{S'}_\mathrm{color} \mid \forall v \in V : (C(v).\texttt{LF} = \texttt{L}_1 \Rightarrow C(v).\texttt{type} < 2^{\lceil \log N \rceil}) \wedge (C(v).\texttt{LF} \neq \texttt{L}_1 \Rightarrow C(v).\texttt{timer}_\mathrm{V} = 0)\}$.
- $\mathcal{V}_\mathrm{only} = \{C \in \mathcal{S'}_\mathrm{color} \mid \forall v, \forall u \in V : C(v).\texttt{timer}_\mathrm{V} > 0 \wedge C(u).\texttt{timer}_\mathrm{V} > 0 \Rightarrow C(v).\texttt{type} = C(u).\texttt{type}\} \cap \{C \in \mathcal{S'}_\mathrm{color} \mid \forall v \in V : C(v).\texttt{LF} = \texttt{L}_1 \Rightarrow C(v).\texttt{type} \geq 2^{\lceil \log N \rceil}$.
- $\mathcal{E}_\mathrm{half} = \{C \in \mathcal{S'}_\mathrm{color} \mid \forall v \in V : C(v).\texttt{LF} \in \{\texttt{L}_0, \texttt{L}_1\} \Rightarrow C(v).\texttt{timer}_\mathrm{E} \geq \texttt{t}_\mathrm{BC}\}$.
- $\mathcal{S}_\mathrm{LE} = \mathcal{B}_\mathrm{no} \cap \mathcal{L}_\mathrm{one} \cap \mathcal{LF}_\mathrm{qua} \cap \mathcal{KL}_\mathrm{zero} \cap (\mathcal{V}_\mathrm{clean} \cup (\mathcal{L}_\mathrm{v1} \cap (\mathcal{V}_\mathrm{make} \cup \mathcal{V}_\mathrm{only}) \cap \mathcal{E}_\mathrm{half}))$.

4.3.1 Expected Holding Time

▶ **Lemma 7.** *Let* $C_0 \in \mathcal{S}_\mathrm{LE}$ *and* $\Xi_{\mathcal{P}_\mathrm{BC}}(C_0) = C_0, C_1, \ldots$. *If* $\Pr(\forall i \in [0, 2m\tau] : C_i \in LE \wedge C_{2m\tau} \in \mathcal{S}_\mathrm{LE}) = 1 - O(ne^{-\tau})$ *holds, then* $\min_{C \in \mathcal{S}_\mathrm{LE}} \mathrm{EHT}_{\mathcal{P}_\mathrm{BC}}(C, LE) = \Omega(\tau e^\tau)$ *holds.*

Proof. Let $A = \min_{C_0 \in \mathcal{S}_\mathrm{LE}} \mathrm{EHT}_{\mathcal{P}_\mathrm{BC}}(C_0, LE)$. We assume that $C_0, \ldots, C_{2m\tau} \in LE \wedge C_{2m\tau} \in \mathcal{S}_\mathrm{LE}$ holds with probability at least $p = 1 - O(ne^{-\tau})$. Then, We have $A \geq p(2m\tau + A)$. Solving this inequality gives $A \geq 2m\tau/(1-p) = \Omega(\tau e^\tau)$. ◀

We say that an agent u encounters a counting interaction when u interacts with an agent v such that $u.\mathtt{color} = v.\mathtt{pcol}$ holds.

▶ **Lemma 8.** *Let $C_0 \in \mathcal{S}'_{\mathrm{color}}$ and $\Xi_{\mathcal{P}_{\mathrm{BC}}}(C_0) = C_0, C_1, \dots$. The probability that every agent encounters less than $\mathtt{t}_{\mathrm{BC}}/2$ counting interactions while $\Gamma_0, \dots, \Gamma_{2m\tau-1}$ is at least $1 - ne^{-\tau}$.*

▶ **Lemma 9.** *Let $C_0 \in \mathcal{S}'_{\mathrm{color}}$ and $\Xi_{\mathcal{P}_{\mathrm{BC}}}(C_0) = C_0, C_1, \dots$. For any $x \in \{\mathtt{LF}, \mathtt{KL}, \mathrm{E}, \mathrm{V}\}$, and for any $y \ge \mathtt{t}_{\mathrm{BC}}/2$ such that y is no more than the maximum value of the domain of $\mathtt{timer}_x$, when $\exists v \in V : C_0(v).\mathtt{timer}_x \ge y$ holds, the probability that $\forall u \in V : C_{2m\tau}(u).\mathtt{timer}_x > y - \mathtt{t}_{\mathrm{BC}}/2$ holds is at least $1 - 2ne^{-\tau}$.*

▶ **Lemma 10.** *Let $C_0 \in \mathcal{S}'_{\mathrm{color}}$ and $\Xi_{\mathcal{P}_{\mathrm{BC}}}(C_0) = C_0, C_1, \dots$. For any integer $\lambda > 0$, and any integer x satisfying $\mathtt{t}_{\mathrm{BC}} \le x \le 2\mathtt{t}_{\mathrm{BC}}$, if $\forall v \in V : C_0(v).\mathtt{timer}_{\mathtt{LF}} \ge x \land \forall i \in [0, \lambda-1], \exists v \in V : C_{2mi\tau}(v).\mathtt{timer}_{\mathtt{LF}} \ge x$ holds, then $\Pr(\forall j \in [0, 2m\lambda\tau], \forall v \in V : C_j(v).\mathtt{timer}_{\mathtt{LF}} > x - \mathtt{t}_{\mathrm{BC}} \land C_{2m\lambda\tau}(v).\mathtt{timer}_{\mathtt{LF}} \ge x - \mathtt{t}_{\mathrm{BC}}/2) \ge 1 - 3\lambda ne^{-\tau}$ holds.*

Proof. Since there exists an agent u satisfying $u.\mathtt{timer}_{\mathtt{LF}} \ge x$ in C_0, the probability that every agent's $\mathtt{timer}_{\mathtt{LF}} \ge x - \mathtt{t}_{\mathrm{BC}}/2$ holds in $C_{2m\tau}$ is at least $1 - 2ne^{-\tau}$ from Lemma 9. Since there is every agent u satisfying $u.\mathtt{timer}_{\mathtt{LF}} \ge x - \mathtt{t}_{\mathrm{BC}}/2$ in C_0, $\Pr(\forall j \in [0, 2m\tau], \forall v \in V : C_j(v).\mathtt{timer}_{\mathtt{LF}} > x - \mathtt{t}_{\mathrm{BC}}) \ge 1 - ne^{-\tau}$ holds from Lemma 8. Thus, $\Pr(\forall j \in [0, 2m\tau], \forall v \in V : C_j(v).\mathtt{timer}_{\mathtt{LF}} > x - \mathtt{t}_{\mathrm{BC}} \land C_{2m\tau}(v).\mathtt{timer}_{\mathtt{LF}} \ge x - \mathtt{t}_{\mathrm{BC}}/2) \ge 1 - 3ne^{-\tau}$ holds by the union bound. Repeating this λ times, we get $\Pr(\forall j \in [0, 2m\lambda\tau], \forall v \in V : C_j(v) > x - \mathtt{t}_{\mathrm{BC}} \land C_{2m\lambda\tau}(v).\mathtt{timer}_{\mathtt{LF}} \ge x - \mathtt{t}_{\mathrm{BC}}/2) \ge 1 - 3\lambda ne^{-\tau}$ by the union bound. ◀

We can prove Lemma 11 by the same way of Lemma 10, and Lemma 12 by assigning $\lambda = 1$ to Lemma 10. Lemma 13, 14, and 15 analyzes the probability that configuration keep some condition for some interval.

▶ **Lemma 11.** *Let $C_0 \in \mathcal{S}'_{\mathrm{color}}$ and $\Xi_{\mathcal{P}_{\mathrm{BC}}}(C_0) = C_0, C_1, \dots$. For any integer $\lambda > 0$, and any integer x satisfying $\mathtt{t}_{\mathrm{BC}} \le x \le 2\mathtt{t}_{\mathrm{BC}}$, if $\forall v \in V : C_0(v).\mathtt{timer}_{\mathrm{E}} \ge x \land \forall i \in [0, \lambda-1], \exists v \in V : C_{2mi\tau}(v).\mathtt{timer}_{\mathrm{E}} \ge x$ holds, then $\Pr(\forall i \in [0, 2m\lambda\tau], \forall v \in V : C_i(v).\mathtt{timer}_{\mathrm{E}} > x - \mathtt{t}_{\mathrm{BC}} \land C_{2m\lambda\tau}(v).\mathtt{timer}_{\mathrm{E}} \ge x - \mathtt{t}_{\mathrm{BC}}/2) \ge 1 - 3\lambda ne^{-\tau}$ holds.*

▶ **Lemma 12.** *Let $C_0 \in \mathcal{S}_{\mathrm{LE}}$ and $\Xi_{\mathcal{P}_{\mathrm{BC}}}(C_0) = C_0, C_1, \dots$. $\Pr(\forall i \in [0, 2m\tau] : C_i \in \mathcal{B}_{\mathrm{no}} \land C_{2m\tau} \in \mathcal{LF}_{\mathrm{qua}}) \ge 1 - 3ne^{-\tau}$ holds.*

▶ **Lemma 13.** *Let $C_0 \in \mathcal{S}_{\mathrm{LE}} \cap \mathcal{V}_{\mathrm{clean}}$ and $\Xi_{\mathcal{P}_{\mathrm{BC}}}(C_0) = C_0, C_1, \dots$. $\Pr(\forall i \in [0, 2m\tau] : C_i \in \mathcal{L}_{\mathrm{one}} \land C_{2m\tau} \in \mathcal{S}_{\mathrm{LE}} \cap (\mathcal{V}_{\mathrm{clean}} \cup \mathcal{L}_{\mathrm{v1}} \cap (\mathcal{V}_{\mathrm{make}} \cup \mathcal{V}_{\mathrm{only}}) \cap \mathcal{E}_{\mathrm{half}}) \ge 1 - 5ne^{-\tau}$ holds.*

▶ **Lemma 14.** *Let $C_0 \in \mathcal{S}_{\mathrm{LE}} \cap \mathcal{L}_{\mathrm{v1}} \cap \mathcal{V}_{\mathrm{make}} \cap \mathcal{E}_{\mathrm{half}}$ and $\Xi_{\mathcal{P}_{\mathrm{BC}}}(C_0) = C_0, C_1, \dots$. $\Pr(\forall i \in [0, 2m\tau] : C_i \in \mathcal{L}_{\mathrm{one}} \land C_{2m\tau} \in \mathcal{S}_{\mathrm{LE}} \cap \mathcal{L}_{\mathrm{v1}} \cap (\mathcal{V}_{\mathrm{make}} \cup \mathcal{V}_{\mathrm{only}}) \cap \mathcal{E}_{\mathrm{half}}) \ge 1 - 3ne^{-\tau}$ holds.*

▶ **Lemma 15.** *Let $C_0 \in \mathcal{S}_{\mathrm{LE}} \cap \mathcal{L}_{\mathrm{v1}} \cap \mathcal{V}_{\mathrm{only}} \cap \mathcal{E}_{\mathrm{half}}$ and $\Xi_{\mathcal{P}_{\mathrm{BC}}}(C_0) = C_0, C_1, \dots$. $\Pr(\forall i \in [0, 2m\tau] : C_i \in \mathcal{L}_{\mathrm{one}} \land C_{2m\tau} \in \mathcal{S}_{\mathrm{LE}} \cap (\mathcal{V}_{\mathrm{clean}} \cup \mathcal{L}_{\mathrm{v1}} \cup \mathcal{V}_{\mathrm{only}} \cap \mathcal{E}_{\mathrm{half}}) \ge 1 - 5ne^{-\tau}$ holds.*

▶ **Lemma 16.** $\min_{C \in \mathcal{S}_{\mathrm{LE}}} \mathrm{EHT}_{\mathcal{P}_{\mathrm{BC}}}(C, LE) = \Omega(\tau e^{\tau})$.

Proof. Let $C_0 \in \mathcal{S}_{\mathrm{LE}}$. $\Pr(C_0, \dots, C_{2m\tau} \in LE \land C_{2m\tau} \in \mathcal{S}_{\mathrm{LE}}) \ge 1 - 5ne^{-\tau} = 1 - O(ne^{-\tau})$ from Lemma 13, Lemma 14, and Lemma 15. Thus, this lemma follows from Lemma 7. ◀

4.3.2 Expected Convergence Time

We first analyze the number of interactions until all timers converge to 0 with high probability. Let $\lambda \mathtt{t}_{\mathrm{BC}}$ be the maximum value of the domain of timers, that is, $\lambda = 1$ for $\mathtt{timer}_{\mathtt{KL}}$ and $\lambda = 2$ for $\mathtt{timer}_{\mathtt{LF}}$, $\mathtt{timer}_{\mathrm{V}}$, and $\mathtt{timer}_{\mathrm{E}}$.

▶ **Lemma 17.** *Let $C_0 \in \mathcal{S'}_{\mathrm{color}}$ and $\Xi_{\mathcal{P}_{\mathrm{BC}}}(C_0) = C_0, C_1, \dots$. For any $x \in \{\mathtt{LF}, \mathtt{KL}, \mathrm{V}, \mathrm{E}\}$, if every agent's $\mathtt{timer}_x$ increases only by Larger Time Propagation (not including setting to a specific value like $\mathtt{t}_{\mathrm{BC}}$ by leaders etc.), the number of interactions until every agent's $\mathtt{timer}_x$ becomes 0 is less than $2340\lambda m\tau \log n$ with probability at least $1 - e^{-\lambda\tau}$.*

Proof. Let $z = \max_{v \in V}(C_i(v).\mathtt{timer}_x)$ $(i > 0)$. From the mechanism of Larger Time Propagation, for every agent v, $C_i(v).\mathtt{timer}_x$ does not become z if $C_{i-1}(v).\mathtt{timer}_x < z$. Thus, when every agent decreases its timer by at least 1 from $C_j, \dots, C_i$ $(0 \leq j < i)$, $\max_{v \in V}(C_i(v).\mathtt{timer}_x) - \max_{v \in V}(C_j(v).\mathtt{timer}_x) \geq 1$ holds (i.e., , the maximum value of $\mathtt{timer}_x$ decreases by at least 1). Let $X \sim \mathrm{Bi}(2m, \delta_v/m)$ be a binomial random variable that represents the number of interactions of an agent v interacts during $2m$ interactions. From Chernoff Bound (Eq. 4.5 in [14]), $\Pr(X \geq \delta_v) \geq \Pr(X > \delta_v) = 1 - \Pr(X \leq \delta_v) = 1 - \Pr(X \leq (1 - 1/2)E[X]) \geq 1 - e^{-\delta_v/8} \geq 1 - e^{-1/4} > 1/5$. Thus, the probability that an agent v interacts no less than δ_v times during $2m$ interactions is at least $1/5$. Let $Y \sim \mathrm{Bi}(\delta_v, 2/\delta_v)$ be a binomial random variable that represents the number of counting interactions that an agent v encounters during δ_v interactions in which an agent v interacts. The probability that $Y = 0$ is $\Pr(Y = 0) = (1 - 2\delta_v)^{\delta_v} \leq e^{-2} < 1/5$. Thus, the probability that an agent v encounters at least one counting interaction is at least $4/5$. Let E_v denote the number of interactions until an agent v decreases its $\mathtt{timer}_x$ by at least 1. Since $E_v \leq 2m + (1 - 4/25)E_v$ holds, $E_v \leq 13m$ holds. By Markov's inequality, the probability that an agent v does not decrease $\mathtt{timer}_x$ during $2E_v$ interactions is no more than $1/2$. Thus, the probability that an agent v does not decrease $\mathtt{timer}_x$ during $4 \log n \cdot E_v$ interactions is no more than n^{-2}. By the union bound, the probability that every agent v does not decrease $\mathtt{timer}_x$ during $4 \log n \cdot E_v$ interactions is no more than n^{-1}. Let A be an event that every agent v decreases $\mathtt{timer}_x$ by at least 1 during $4 \log n \cdot E_v$ interactions. We consider the expected number of times until A succeeds $16\lambda\tau (= \lambda \mathtt{t}_{\mathrm{BC}})$ times using geometric distributions. In other words, for $k \in [1, 16\lambda\tau]$, let $Z_k \sim Geom(p_k)$ be the independent geometric random variable such that $p_k = 1 - 1/n \geq 1/2$. Considering the sum of independent random variables $Z = \sum_{k=1}^{16\lambda\tau} Z_k$. Note that $E[Z] \leq 32\lambda\tau$ holds. From Janson's inequality (Theorem 2.1 in [13]), $\Pr(Z \geq 45\lambda\tau) \leq \Pr(Z \geq 1.4 \cdot 32\lambda\tau) \leq \Pr(Z \geq 1.4 \cdot E[Z]) \leq e^{-p_i E[Z](1.4 - 1 - \log_e 1.4)} \leq e^{-16\lambda\tau(1.4 - 1 - \log_e 1.4)} \leq e^{-\lambda\tau}$. Thus, the expected number of times that A succeeds $16\lambda\tau$ times is less than $45\lambda\tau$ with probability at least $1 - e^{-\lambda\tau}$. Therefore, the number of interactions until all agents' $\mathtt{timer}_x$ becomes 0 is $45\lambda\tau \cdot 4 \log n \cdot E_v \leq 2340\lambda m\tau \log n$ with probability at least $1 - e^{-\lambda\tau}$. ◀

Lemma 18 shows a convergence time.

▶ **Lemma 18.** *Let $C_0 \in \mathcal{S}_{\mathrm{color}}$ and $\Xi_{\mathcal{P}_{\mathrm{BC}}}(C_0) = C_0, C_1, \dots$. The number of interactions until the configuration reaches $\mathcal{S}_{\mathrm{LE}}$ is $O(m\tau \log n)$ with probability $1 - o(1)$.*

▶ **Theorem 19.** *Protocol $\mathcal{P}_{\mathrm{BC}}$ is a randomized $(O(m(n + \tau \log n)), \Omega(\tau e^{\tau}))$-loosely-stabilizing leader election protocol for arbitrary graphs when $\tau \geq \max(2d, \lceil \log N \rceil / 2, 15 + 3 \log n)$ if $\mathcal{P}_{\mathrm{LRU}}$ is used for two-hop coloring.*

▶ **Theorem 20.** *Protocol $\mathcal{P}_{\mathrm{BC}}$ is a deterministic $(O(m(n + \Delta \log N + \tau \log n)), \Omega(\tau e^{\tau}))$-loosely-stabilizing leader election protocol for arbitrary graphs when $\tau \geq \max(2d, \lceil \log N \rceil / 2, 15 + 3 \log n)$ if $\mathcal{P'}_{\mathrm{LRU}}$ is used for two-hop coloring.*

5 Conclusion

New loosely-stabilizing leader election population protocols on arbitrary graphs without identifiers are proposed. One is randomized, and the other is deterministic. The randomized one converges within $O(mN \log n)$ steps, while the deterministic one converges $O(mN \log N)$ steps both in expectations and with high probability. Both protocols hold a unique leader with $\Omega(Ne^{2N})$ expected steps and utilizes $O(\Delta \log N)$ bits of memory. The convergence time is close to the known lower bound of $\Omega(mN)$.

References

1. Dan Alistarh, Rati Gelashvili, and Joel Rybicki. Fast Graphical Population Protocols. In *25th International Conference on Principles of Distributed Systems (OPODIS 2021)*, volume 217, pages 14:1–14:18, 2022. doi:10.4230/LIPICS.OPODIS.2021.14.
2. Dan Alistarh, Joel Rybicki, and Sasha Voitovych. Near-optimal leader election in population protocols on graphs. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, pages 246–256, 2022. doi:10.1145/3519270.3538435.
3. Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4):235–253, 2006. doi:10.1007/S00446-005-0138-3.
4. Dana Angluin, James Aspnes, Michael J. Fischer, and Hong Jiang. Self-stabilizing population protocols. *ACM Trans. Auton. Adapt. Syst.*, 3(4), 2008. doi:10.1145/1452001.1452003.
5. Joffroy Beauquier, Peva Blanchard, and Janna Burman. Self-stabilizing leader election in population protocols over arbitrary communication graphs. In *Principles of Distributed Systems*, pages 38–52, 2013. doi:10.1007/978-3-319-03850-6_4.
6. Petra Berenbrink, George Giakkoupis, and Peter Kling. Optimal time and space leader election in population protocols. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 119–129, 2020. doi:10.1145/3357713.3384312.
7. Janna Burman, Ho-Lin Chen, Hsueh-Ping Chen, David Doty, Thomas Nowak, Eric Severson, and Chuan Xu. Time-optimal self-stabilizing leader election in population protocols. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, pages 33–44, 2021. doi:10.1145/3465084.3467898.
8. Shukai Cai, Taisuke Izumi, and Koichi Wada. How to prove impossibility under global fairness: On space complexity of self-stabilizing leader election on a population protocol model. *Theory of Computing Systems*, 50(3):433–445, 2012. doi:10.1007/S00224-011-9313-Z.
9. Davide Canepa and Maria Gradinariu Potop-Butucaru. Stabilizing leader election in population protocols. Research Report RR-6269, INRIA, 2007.
10. Michael Fischer and Hong Jiang. Self-stabilizing leader election in networks of finite-state anonymous agents. In *Principles of Distributed Systems*, pages 395–409, 2006. doi:10.1007/11945529_28.
11. Leszek Gąsieniec, Grzegorz Stachowiak, and Przemyslaw Uznanski. Almost logarithmic-time space optimal leader election in population protocols. In *The 31st ACM Symposium on Parallelism in Algorithms and Architectures*, pages 93–102, 2019. doi:10.1145/3323165.3323178.
12. Taisuke Izumi. On space and time complexity of loosely-stabilizing leader election. In *Structural Information and Communication Complexity*, pages 299–312, 2015. doi:10.1007/978-3-319-25258-2_21.
13. Svante Janson. Tail bounds for sums of geometric and exponential variables. *Statistics & Probability Letters*, 135:1–6, 2018.
14. Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005. doi:10.1017/CBO9780511813603.

15 Yuichi Sudo, Ryota Eguchi, Taisuke Izumi, and Toshimitsu Masuzawa. Time-Optimal Loosely-Stabilizing Leader Election in Population Protocols. In *35th International Symposium on Distributed Computing (DISC 2021)*, volume 209, pages 40:1–40:17, 2021. `doi:10.4230/LIPICS.DISC.2021.40`.

16 Yuichi Sudo, Junya Nakamura, Yukiko Yamauchi, Fukuhito Ooshita, Hirotsugu Kakugawa, and Toshimitsu Masuzawa. Loosely-stabilizing leader election in a population protocol model. *Theoretical Computer Science*, 444:100–112, 2012. `doi:10.1016/J.TCS.2012.01.007`.

17 Yuichi Sudo, Fukuhito Ooshita, Taisuke Izumi, Hirotsugu Kakugawa, and Toshimitsu Masuzawa. Time-optimal leader election in population protocols. *IEEE Transactions on Parallel and Distributed Systems*, 31(11):2620–2632, 2020. `doi:10.1109/TPDS.2020.2991771`.

18 Yuichi Sudo, Fukuhito Ooshita, Hirotsugu Kakugawa, and Toshimitsu Masuzawa. Loosely-stabilizing leader election on arbitrary graphs in population protocols. In *Principles of Distributed Systems*, pages 339–354, 2014. `doi:10.1007/978-3-319-14472-6_23`.

19 Yuichi Sudo, Fukuhito Ooshita, Hirotsugu Kakugawa, and Toshimitsu Masuzawa. Loosely stabilizing leader election on arbitrary graphs in population protocols without identifiers or random numbers. *IEICE Transactions on Information and Systems*, E103.D(3):489–499, 2020. `doi:10.1587/TRANSINF.2019FCP0003`.

20 Yuichi Sudo, Fukuhito Ooshita, Hirotsugu Kakugawa, Toshimitsu Masuzawa, Ajoy K. Datta, and Lawrence L. Larmore. Loosely-stabilizing leader election for arbitrary graphs in population protocol model. *IEEE Transactions on Parallel and Distributed Systems*, 30(6), 2019. `doi:10.1109/TPDS.2018.2881125`.

21 Yuichi Sudo, Fukuhito Ooshita, Hirotsugu Kakugawa, Toshimitsu Masuzawa, Ajoy K. Datta, and Lawrence L. Larmore. Loosely-stabilizing leader election with polylogarithmic convergence time. *Theoretical Computer Science*, 806:617–631, 2020. `doi:10.1016/J.TCS.2019.09.034`.

22 Yuichi Sudo, Masahiro Shibata, Junya Nakamura, Yonghwan Kim, and Toshimitsu Masuzawa. Self-stabilizing population protocols with global knowledge. *IEEE Transactions on Parallel and Distributed Systems*, 32(12):3011–3023, 2021. `doi:10.1109/TPDS.2021.3076769`.