Brief Announcement: Logarithmic Expected-Time Leader Election in Population Protocol Model

Yuichi Sudo
y-sudou@ist.osaka-u.ac.jp
Osaka University
Suita, Osaka, Japan

Fukuhito Ooshita
f-ooshita@is.naist.jp
Nara Institute of Science and
Technology
Ikoma, Nara, Japan

Taisuke Izumi
t-izumi@nitech.ac.jp
Nagoya Institute of Technology
Nagoya, Aichi, Japan

Hirotsugu Kakugawa
kakugawa@rins.ryukoku.ac.jp
Ryukoku University
Otsu, Shiga, Japan

Toshimitsu Masuzawa
masuzawa@ist.osaka-u.ac.jp
Osaka University
Suita, Osaka, Japan

ABSTRACT

In this paper, we present a 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 the population size n such that $m \geq \log_2 n$ and $m = O(\log n)$, this protocol guarantees that exactly one leader is elected and the unique leader is kept forever thereafter.

CCS CONCEPTS

• Theory of computation → Distributed algorithms.

KEYWORDS

population protocols, leader election

ACM Reference Format:

Yuichi Sudo, Fukuhito Ooshita, Taisuke Izumi, Hirotsugu Kakugawa, and Toshimitsu Masuzawa. 2019. Brief Announcement: Logarithmic Expected-Time Leader Election in Population Protocol Model. In *2019 ACM Symposium on Principles of Distributed Computing (PODC '19), July 29-August 2, 2019, Toronto, ON, Canada*. ACM, New York, NY, USA, 3 pages. <https://doi.org/10.1145/3293611.3331585>

1 INTRODUCTION

We consider the *population protocol* (PP) model [5] in this paper. A network called *population* consists of a large number of finite-state automata, called *agents*. Agents make *interactions* (i.e., pairwise communication) each other by which they update their states. The interactions are opportunistic, that is, they are unpredictable. Agents are strongly anonymous: they do not have identifiers and they cannot distinguish their neighbors with the same states. As with the majority of studies on population protocols, we assume

that the network of agents is a complete graph and that the scheduler selects an interacting pair of agents at each step uniformly at random.

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

There have been many works which study the leader election problem in the PP model (Table 1). Angluin et al. [5] gave the first leader election protocol in 2004, which stabilizes in $O(n)$ parallel time 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 [8] showed that any constant space protocol requires linear parallel time to elect a unique leader. Alistarh and Gelashvili [3] made a breakthrough in 2015; they achieve poly-logarithmic stabilization time ($O(\log^3 n)$ parallel time) by increasing the number of states from $O(1)$ to only $O(\log^3 n)$. Thereafter, the stabilization time has been improved by many studies [2, 7, 10–12]. Gąsieniec et al. [11] gave a state-of-art protocol that converges in $O(\log n \cdot \log \log n)$ parallel time in expectation with only $O(\log \log n)$ states. Its space complexity is optimal; Alistarh et al. [1] shows that any poly-logarithmic parallel time algorithm requires $\Omega(\log \log n)$ states. Michail et al. [12] gave a protocol with $O(\log n)$ parallel time in expectation but with a linear number of states. Those protocols with non-constant number of states [1–3, 7, 10, 11] are not *uniform*, that is, they require some rough knowledge of n . For example, in the protocol of [10], a $\Theta(\log \log n)$ value must be hard-coded to set the maximum value of one variable (named l in that paper). One can find detailed information about the leader election in the PP model in two survey papers [4, 9].

Our Contribution. In this paper, we present the *first* leader election protocol P_{LL} with logarithmic stabilization time and with a sub-polynomial number of states. Specifically, the proposed protocol P_{LL} stabilizes in $O(\log n)$ parallel time and uses only $O(\log n)$ states per agent. Compared to the state of art protocol [11], P_{LL} achieves shorter (and the fastest in the literature) stabilization time

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

PODC '19, July 29-August 2, 2019, Toronto, ON, Canada

© 2019 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-6217-7/19/07.

<https://doi.org/10.1145/3293611.3331585>

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

	States	Stabilization Time
[5]	$O(1)$	$O(n)$
[3]	$O(\log^3 n)$	$O(\log^3 n)$
[1]	$O(\log^2 n)$	$O(\log^{5.3} n \cdot \log \log n)$
[7]	$O(\log^2 n)$	$O(\log^2 n)$
[2]	$O(\log n)$	$O(\log^2 n)$
[10]	$O(\log \log n)$	$O(\log^2 n)$
[11]	$O(\log \log n)$	$O(\log n \cdot \log \log n)$
[12]	$O(n)$	$O(\log n)$
This work	$O(\log n)$	$O(\log n)$

but uses larger space of each agent. Compared to [12], P_{LL} achieves drastically small space while maintaining the same stabilization time. The protocol P_{LL} is non-uniform as with the existing non-constant space protocols; it requires a rough knowledge m of n such that $m \geq \log_2 n$ and $m = \Theta(\log n)$.

2 LOGARITHMIC LEADER ELECTION

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

QuickElimination(): An execution of this module takes $O(\log n)$ parallel time in expectation. For any $i \geq 2$, exactly i leaders survive an execution of this module 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 this module, 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 $(1 - O(1/\log n)) \cdot \log n + O(1/\log n) \cdot O(\log^2 n) = O(\log n)$ parallel time in expectation.

In the remainder of this paper, we briefly give key ideas to design the three modules satisfying the above guarantees. 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. Due to the lack of space, we omit from this paper a way to implement these key ideas with $O(\log n)$ states per agent. Moreover, in the following description of the key ideas, we assume a kind of global synchronization, for example, we assume that each agent begins an execution of *Tournament()* after all agents finish necessary operations of *QuickElimination()*. We also omit how to implement such synchronization. See the full pre-print paper [13] for those implementations.

2.1 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} . Actually, this module is based on almost the same idea as *the lottery protocol* in [1]. The protocol P_{LL} achieves much faster stabilization time than the lottery protocol thanks to tighter analysis on the number of surviving leaders, which we will see below, and the combination with the other two modules.

First, consider the following game:

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

Let $i \geq 2$ and $j \geq 0$. Consider the situation that exactly i agents observe that their first j coin flips result in head and define $p_{i,j}$ as the probability that all the i agents wins 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 observes that all of their first j coin flips result in head. We define $k_n = 0$ for simplicity. Then, for any $i \geq 0$, we have $\Pr(|\{v \in V \mid s_v = s_{\max}\}| = i) = \sum_{j=0}^{\infty} \Pr(k_i = j) \cdot p_{i,j} \leq 2^{1-i} \sum_{j=0}^{\infty} \Pr(k_i = j) \leq 2^{1-i}$.

Module *QuickElimination()* simulates this game in the PP 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 comes from the definition of the uniformly random scheduler: at each step, v interacts as an initiator with the same probability as the event that v does as an responder. Strictly speaking, this simple simulation of coin flips does not guarantee independence of coin flips by u and v for any distinct $u, v \in V$. See the full pre-print paper [13] for the mechanism to achieve independent coin flips. Each agent v computes and stores s_v on variable $v.\text{level}_Q$ by counting the number of interactions that it participates in as an initiator until it interacts as a responder

for the first time. After every agent v computes s_v on $v.\text{level}_Q$, the maximum value of level_Q , i.e., $s_{\max}$, is propagated from agent to agent via *one-way epidemic* [6], that is, (i) each agent memorizes the largest value of level_Q it has observed, and (ii) the larger value is propagated to the agent with smaller value at every interaction. It is proven in [6] that all agents obtain the largest value within $O(\log n)$ parallel time with high probability by this simple propagation. If agent v knows $s_v < s_{\max}$, v changes $v.\text{leader}$ from *true* to *false*, that is, v becomes a follower. Thus, when one-way epidemic of $s_{\max}$ finishes, only the agents v satisfying $s_v = s_{\max}$ are leaders. From the above discussion, for any $i \geq 2$, the number of such surviving leaders is exactly i with probability at most 2^{1-i} . On the other hand, there are at least one agent v with $s_v = s_{\max}$, thus this module never eliminates all leaders.

Although we have to define the domain of variable level_Q , a logarithmic number of states is sufficient for level_Q because each agent v gets more than $c \lg n$ consecutive heads with probability at most n^{-c} for any $c \geq 1$.

2.2 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 $1 - O(i/\log n)$. The idea of this component is simple. As with the *QuickElimination()*, we use coin flips in *Tournament()*. Every leader v maintains variable $v.\text{rand}$. Initially, $v.\text{rand} = 0$. Every time it has an interaction, it updates $v.\text{rand}$ by $v.\text{rand} \leftarrow 2v.\text{rand} + j$ where j indicates whether v is a responder in the interaction or not, i.e., $j = 0$ if v is an initiator and $j = 1$ if v is a responder. This operation stops when v encounters $\lceil \log_2 m \rceil = O(\log \log n)$ interactions. Thus, when all the i leaders encounter at least $\lceil \log_2 m \rceil$ interactions, for every leader v , $v.\text{rand}$ is a random variable uniformly chosen from $\{0, 1, \dots, 2^{\lceil \log_2 m \rceil} - 1\}$. As with *QuickElimination()*, the maximum value r and is propagated to the whole population via one-way epidemic within $O(\log n)$ parallel time with high probability and only leaders with the maximum value remains leaders in the end of *Tournament()*.

Let $v_1, v_2, \dots, v_i$ be the i leaders that survive *QuickElimination()*. Let $r_1, r_2, \dots, r_i$ be the resulting value of $v_i.\text{rand}$ and define $r_{\max}(j) = \max(r_1, r_2, \dots, r_j)$ for any $j = 1, 2, \dots, i$. Clearly, the number of leaders at the end of *Tournament()* is exactly one if $r_{j+1} \neq r_{\max}(j)$ holds for all $j = 1, 2, \dots, i-1$. By the union bound and from the independence between any two pairs of $r_1, r_2, \dots, r_i$, this holds with probability at least $1 - \sum_{j=1}^{i-1} 2^{-\lceil \log_2 m \rceil} \geq 1 - i/m = 1 - O(i/\log n)$.

On the other hand, an execution of *Tournament()* never eliminates all leaders since there always at least one leader v_j that satisfies $r_j = r_{\max}(i)$.

2.3 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()* at the beginning of an execution of *BackUp()*. Every leader v maintains $v.\text{level}_B$. Initially, $v.\text{level}_B = 0$. Every leader v repeats the following procedure until $v.\text{level}_B$ reaches $5m$ or v becomes a follower. First, v makes a coin flip. If the result is head, v increments $v.\text{level}_B$ by one. If the

result is tail, v does nothing. Then, v waits for sufficiently long but logarithmic parallel time so that the maximum level_B propagates to the whole population via one-way epidemic. If it observes larger value in the epidemic, it becomes a follower. 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 $1 \leq j < 5m$. Consider the first time that level_B of some leader, say v , reaches j . Let $V' \subseteq V$ be the set of leaders at that time. By the definition of the above procedure, every $u \in V'$ other than v satisfies $u.\text{level}_B < j$, and u makes a coin flip at most once with high probability until the maximum value j is propagated from v to u . If the result of the one coin flip is tail, u becomes a follower. Therefore, with probability at least $1/2 - O(n^{-1}) > 1/3$, no less than half of leaders in $V' \setminus v$ becomes followers, that is, the number of leaders decreases to at most $1 + \lfloor |V'|/2 \rfloor$. Thus, we can show that with probability $1 - O(n^{-1})$, exactly one leader survives when all leaders' level_B s reach $5m$ within $O(m \log n)$. Even if multiple leaders survive at that time, a unique leader is elected within the next $O(n)$ parallel time in expectation because when two leaders meet, one of them becomes a follower. Therefore, the total expected parallel time to elect a unique leader is $O(m \log n) + O(n^{-1}) \cdot O(n) = O(\log^2 n)$.

ACKNOWLEDGMENTS

This work was supported by JSPS KAKENHI Grant Numbers 17K19977, 18K18000, 18K11167, and 19K11826 and Japan Science and Technology Agency (JST) SICORP.

REFERENCES

- [1] Dan Alistarh, James Aspnes, David Eisenstat, Rati Gelashvili, and Ronald L Rivest. 2017. Time-space trade-offs in population protocols. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2560–2579.
- [2] Dan Alistarh, James Aspnes, and Rati Gelashvili. 2018. Space-optimal majority in population protocols. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2221–2239.
- [3] Dan Alistarh and Rati Gelashvili. 2015. Polylogarithmic-time leader election in population protocols. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming*. Springer, 479–491.
- [4] Dan Alistarh and Rati Gelashvili. 2018. Recent algorithmic advances in population protocols. *ACM SIGACT News* 49, 3 (2018), 63–73.
- [5] D. Angluin, J. Aspnes, Z. Diamadi, M.J. Fischer, and R. Peralta. 2006. Computation in networks of passively mobile finite-state sensors. *Distributed Computing* 18, 4 (2006), 235–253.
- [6] Dana Angluin, James Aspnes, and David Eisenstat. 2008. Fast computation by population protocols with a leader. *Distributed Computing* 21, 3 (2008), 183–199.
- [7] Andreas Bilke, Colin Cooper, Robert Elsässer, and Tomasz Radzik. 2017. Brief Announcement: Population Protocols for Leader Election and Exact Majority with $O(\log^2 n)$ States and $O(\log^2 n)$ Convergence Time. In *Proceedings of the 38th ACM Symposium on Principles of Distributed Computing*. Springer, 451–453.
- [8] David Doty and David Soloveichik. 2018. Stable leader election in population protocols requires linear time. *Distributed Computing* 31, 4 (2018), 257–271.
- [9] Robert Elsässer and Tomasz Radzik. 2018. Recent results in population protocols for exact majority and leader election. *Bulletin of EATCS* 3, 126 (2018).
- [10] Leszek Gąsieniec and Grzegorz Stachowiak. 2018. Fast space optimal leader election in population protocols. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2653–2667.
- [11] Leszek Gąsieniec, Grzegorz Stachowiak, and Przemysław Uznański. 2018. Almost logarithmic-time space optimal leader election in population protocols. *arXiv preprint arXiv:1802.06867* (2018).
- [12] Othon Michail, Paul G Spirakis, and Michail Theofilatos. 2018. Simple and Fast Approximate Counting and Leader Election in Populations. In *International Symposium on Stabilizing, Safety, and Security of Distributed Systems*. Springer, 154–169.
- [13] Yuichi Sudo, Fukuhito Ooshita, Taisuke Izumi, Hirotugu Kakugawa, and Toshimitsu Masuzawa. 2018. Logarithmic Expected-Time Leader Election in Population Protocol Model. (2018). arXiv:cs.DC/1812.11309