



國立臺灣大學電機資訊學院電機工程研究所

碩士論文

Graduate Institute of Electrical Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master's Thesis

環形的群體協議中，快速且空間高效的自我穩定領導
者選舉

Fast and space-efficient self-stabilizing leader election in
population protocols on rings

吳政宏

Cheng-Hung Wu

指導教授：陳和麟 博士

Advisor: Ho-Lin Chen, Ph.D.

中華民國 113 年 7 月

July, 2024

國立臺灣大學碩士學位論文

口試委員會審定書

MASTER'S THESIS ACCEPTANCE CERTIFICATE NATIONAL TAIWAN UNIVERSITY

環形的群體協議中，快速且空間高效的自我穩定領導者選舉
Fast and space-efficient self-stabilizing leader election in population
protocols on rings

本論文係 吳政宏 (姓名) R11921036 (學號) 在國立臺灣大學
電機工程學系完成之碩士學位論文，於民國 113 年 7 月 17 日承下列
考試委員審查通過及口試及格，特此證明。

The undersigned, appointed by the Department of Electrical Engineering on 17 (date) 7 (month)
2024 (year) have examined a Master's thesis entitled above presented by Cheng-Hung Wu
(name) R11921036 (student ID) candidate and hereby certify that it is worthy of
acceptance.

口試委員 Oral examination committee:

陳子仲

(指導教授 Advisor)

蔡孟宇

孫志

呂將

系主任 Director:

李建模





Acknowledgements

這篇論文的完成，首先要感謝指導教授陳和麟老師。在題目的選擇上老師給予了完全的自由，對於學生的任何疑問也願意耐心的進行討論。每次與老師的討論都可以得到很多研究的啟發，論文中不少的設計也都是與老師討論後的成果。在這樣的研究環境以及老師豐富的研究經驗，這篇論文才得以順利的完成。再來我想要感謝來參加口試的口試委員：呂學一老師、于天立老師、蔡孟宗老師，感謝老師們抽空來參加口試，並對論文提供意見及指教。

也感謝實驗室的同學們，雖然大家做的研究主題都不一樣，但我們還是互相鼓勵及幫助，最後才能順利的完成學位口試。最後想感謝我的家人，有家人的支持我才能在學習的路上走到現在。

吳政宏

2024.07.17





摘要

在本文中，我們提出了一種能夠在有向環上解決自穩定領導者選舉問題的群體協議。給定對群體規模 n 的知識 $\psi = \lceil \log \log n \rceil + O(1)$ ，該協議能夠以高機率在步內選出唯一的領導者。此後，群體將永遠保持這個唯一的領導者。該協議使用 $\text{polyloglog}(n)$ 個狀態。與現有文獻相比，我們的協議保持了快速收斂速度，並進一步減少了所使用的狀態數量。

關鍵字：分散式系統、群體協議、領導者選舉、自我穩定、環





Abstract

In this paper, we propose a population protocol that solves self-stabilizing leader election on directed rings. Given a knowledge $\psi = \lceil \log \log n \rceil + O(1)$ about the population size n , the protocol elects a unique leader within $O(n^2 \log n \log \log n)$ steps with high probability. Thereafter, the population keeps the unique leader forever. The protocol uses $\text{polyloglog}(n)$ states. Compared to the current literature, our protocol maintains a fast convergence speed and further reduces the number of states used.

Keywords: Distributed System, Population Protocols, Leader Election, Self-stabilization, Rings

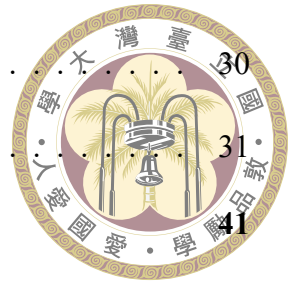




Contents

	Page
Verification Letter from the Oral Examination Committee	i
Acknowledgements	iii
摘要	v
Abstract	vii
Contents	ix
List of Tables	xi
Chapter 1 Introduction	1
Chapter 2 Preliminary	7
Chapter 3 SSLE on Rings	11
3.1 Overview	11
3.2 Cleaning	13
3.3 Creating Leaders	13
3.3.1 Generating Signals	17
3.3.2 Cleaning partial signals	18
3.3.3 Signal Comparison	22
3.3.4 Signal Checking	26
3.4 Suppressing	29

3.5	Eliminating Leaders	30
3.6	Stabilizing	31
Chapter 4	Correctness and Convergence time	41
4.1	Safe Configurations	41
4.1.1	Convergence	43
Chapter 5	Conclusion	57
References		59
Appendix A — CHERNOFF BOUNDS		65





List of Tables

1.1 Self-Stabilizing Leader Election on Rings (Convergence time is shown in the expected number of steps)	4
---	---

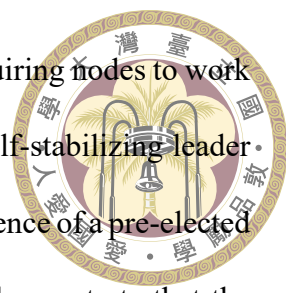




Chapter 1 Introduction

Population protocols [5] are an asynchronous and anonymous distributed computing model used to represent systems composed of simple units, such as sensor networks or chemical reaction networks [14, 22]. In the model of a population protocol, there is a set of finite-state automata called agents. A pair of agents makes interaction (pairwise communication) to update their states. Each agent is anonymous, so during interactions, an agent cannot distinguish between two agents with the same state. In each time step, one pair of agents is chosen by the scheduler to have an interaction. The behavior of the scheduler cannot be predicted. Although the behavior of the scheduler is uncontrollable, assuming that the scheduler uniformly chooses a pair of agents at random enables us to estimate the computational time complexity of the population protocol. Many tasks in distributed computing have been studied in population protocols, such as majority [6, 4, 2], counting [10, 8] and leader election [3, 1, 18, 11].

A distributed algorithm is self-stabilizing [17] if it consistently converges to the desired configuration, no matter the initial configuration of the system. A self-stabilizing algorithm can recover from any transient failure, ensuring reliable computation. Self-stabilization is important in the population protocol model because this model typically represents a distributed system composed of many inexpensive and unreliable nodes, requiring robust fault tolerance.

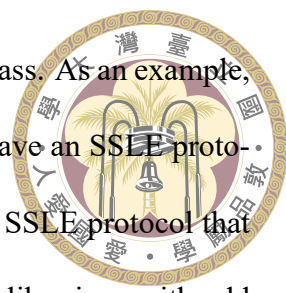


Leader election is a fundamental task in distributed systems, requiring nodes to work together to select one node as the leader. In population protocols, self-stabilizing leader election (SSLE) is important because many protocols assume the existence of a pre-elected leader, as indicated in several studies [5, 6, 7]. Angluin et al. [6] demonstrate that the presence of a unique leader allows for the efficient resolution of all semi-linear predicates. This serves as the motivation for studying SSLE in population protocols. However, solving self-stabilizing leader election without any assumptions is impossible [7, 13, 30]. To overcome this impossibility, many studies introduce assumptions about the model, restricting network topology or relaxing protocol requirements.

Several studies [13, 11, 30] assume that each agent has knowledge of the exact number of agents. Based on this assumption, Cai et al. [13] developed an SSLE protocol specifically designed for complete graphs. Burman et al. [12] introduced time-optimal protocols under the same conditions. Sudo et al. [30] gave an SSLE protocol suitable for arbitrary graphs.

Several studies use *oracles* to overcome the impossibility of SSLE. Fischer et al. [19] introduced an oracle called the eventual detector $\Omega^?$, which eventually detect the presence or absence of a leader. They also presented two protocols to solve SSLE on complete graphs and rings of all sizes using the oracle. Beauquier et al. [9] gave an SSLE protocol for arbitrary graphs, using a composition of two copies of $\Omega^?$.

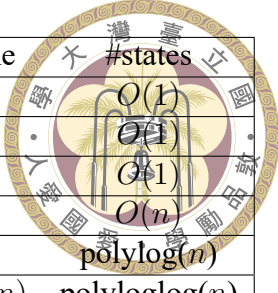
Several studies restrict the topology of a graph to overcome the impossibility of SSLE. A class $\mathcal{G}$ of graphs is called simple if there does not exist a graph in $\mathcal{G}$ that contains two disjoint subgraphs that are also in $\mathcal{G}$. Angluin et al. [7] proved that no SSLE protocol exists that can work for all graphs in any non-simple class. If we focus on a class



of simple graph, there may exist SSLE protocol for all graphs in the class. As an example, the class of rings is typically considered simple. Fischer et al. [19] gave an SSLE protocol for general rings using oracle $\Omega?$. Angluin et al. [7] proposed an SSLE protocol that works in any ring whose size is not a multiple of a certain integer k , like rings with odd numbers of agents, and left SSLE on rings of all sizes without any oracle or knowledge as an open problem. This question remained unresolved until Chen and Chen [15] gave an SSLE protocol for general rings. These three protocols in [19, 7, 15] use a constant number of states per agent. The expected convergence times of the protocols in [19, 7] are $\Theta(n^3)$ where n is the number of agents, while the convergence time of the protocol in [15] is exponentially long. Yokota et al. [31] gave a time-optimal SSLE protocol, which uses $O(n)$ states per agent. Yokota et al. [32] gave a near time-optimal SSLE protocol, which uses $\text{polylog}(n)$ states per agent. Chen and Chen [16] extended the protocol in [15] so that it can solve SSLE on arbitrary regular graphs.

Several studies [20, 23, 24, 25, 26, 27, 28, 29] have relaxed the original self-stabilization requirement and introduced loosely-stabilizing leader election protocols. Specifically, these studies allow a population to temporarily diverge from the problem's specification (such as having a unique leader) after sustaining the specification for a prolonged period. This concept was first introduced by [25]. The protocols proposed by [20, 23, 25, 29] are designed for complete graphs, while those by [24, 26, 27, 28] are applicable to arbitrary graphs.

Our result: The main result of this paper is a population protocol P_{PLL} that can solve SSLE on directed rings of all sizes. Given an integer $\psi = \lceil \log \log n \rceil + O(1)$, P_{PLL} elects a unique leader in $O(n^2 \log n \log \log n)$ steps with high probability and uses $\text{polyloglog}(n)$ states per agent. This work further reduces the required number of states of



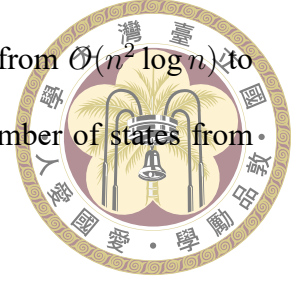
	Assumption	Convergence Time	#states
[7]	n is not a multiple of a given k	$\Theta(n^3)$	$O(1)$
[19]	oracle $\Omega?$	$\Theta(n^3)$	$O(1)$
[15]	none	exponential	$O(1)$
[31]	knowledge $\psi = \lceil \log n \rceil + O(1)$	$\Theta(n^2)$	$O(n)$
[32]	knowledge $\psi = \lceil \log n \rceil + O(1)$	$\Theta(n^2 \log n)$	$\text{polylog}(n)$
this work	knowledge $\psi = \lceil \log \log n \rceil + O(1)$	$\Theta(n^2 \log n \log \log n)$	$\text{polyloglog}(n)$

Table 1.1: Self-Stabilizing Leader Election on Rings (Convergence time is shown in the expected number of steps)

[32] by sacrificing a bit of expected convergence time under the same assumption.

The technical idea of the protocol is to combine the modified mechanism from [7] with the modified mechanism described in [32] to detect the absence of a leader. Like [32], we divide the population into several groups, each assigned an ID. This ID is then used to detect the absence of a leader. In the method in [32], the population is divided into n/ψ groups, with each group's ID in the range $[0, 2^\psi - 1]$, where $\psi = \lceil \log n \rceil + O(1)$ and $2^\psi \geq 2^{\log n} = n$. The entire ring is arranged in a clockwise ascending ID sequence. Because a group's ID can be greater than n/ψ , there will always be a group that does not conform to the sequence. This discrepancy can then be used to check for the absence of a leader. In our work, we modify the pattern of sequence so that the ID only needs to be in the range $[0, 2^\psi - 1]$, where $\psi = \lceil \log \log n \rceil + O(1)$ and $2^\psi \geq 2^{\log \log n} = \log n$. This further restricts the range of IDs, but it is still possible to find a group that does not conform to the modified sequence. However, reducing the number of states prevents us from simply counting to let the agent wait for a period of time at certain moments. As a result, we cannot use the same grouping method as in [32]. Therefore, we adopt the modified protocol in [7] for grouping instead. Moreover, modifications to the sequence pattern prevent us from just comparing the IDs of the two adjacent groups, as in [32]. We need to be able to compare the IDs of two groups that are far apart in the sequence as well. We overcome these challenges by slightly increasing the convergence time. In summary,

compared to [32], our protocol slightly increases convergence time from $O(n^2 \log n)$ to $O(n^2 \log n \log \log n)$ steps but significantly reduces the required number of states from $\text{polylog}(n)$ to $\text{polyloglog}(n)$.



In this paper, we assume the population is oriented, i.e. a directed ring. However, in [32], by adding a sub-protocol using $O(1)$ states, the population can be oriented in $O(n^2 \log n)$ steps with probability $1 - O(1/n)$. We can remove the assumption of orientation using the same method without affecting the time or number of states of our protocol.





Chapter 2 Preliminary

The definition of the population protocol is derived from [5], but here we use the notations and settings from [32].

A *population* is a weakly connected directed graph $G(V, E)$. Each $v \in V$ represents an agent. If $(u, v) \in E$, agents u and v are allowed to interact with each other, with u serving as an initiator and v serving as a responder. In this paper, we consider the population represented by a directed ring, i.e. we consider that $V = \{u_0, u_1, \dots, u_{n-1}\}$ and $E = \{(u_i, u_{(i+1) \bmod n}) \mid i = 0, 1, \dots, n-1\}$. The indices of the agents are used solely for simplicity of description; the agents themselves do not have access to their indices. We call $u_{(i-1) \bmod n}$ and $u_{(i+1) \bmod n}$ the left neighbor and the right neighbor of u_i , respectively. We denote “left to right” direction by clockwise direction, and “right to left” direction by counter-clockwise direction.

A population protocol $P(Q, Y, T, \pi_{out})$ consists of four components: the finite set Q of states, the finite set Y of outputs, the transition function $T : Q \times Q \rightarrow Q \times Q$ and the output function $Q \rightarrow Y$. When an interaction between two agents occurs, the transition function T determine their the next states based on their current states. The output of an agent is determined by the output function π_{out} : the output of an agent with state $q \in Q$ is $\pi_{out}(q)$. We assume that all agents have a common knowledge $\psi = \lceil \log \log n \rceil + O(1)$.

Consequently, the components Q , T and π_{out} may depend on ψ .

A configuration of a population protocol is a function $C : V \rightarrow Q$ that specifies the states of all agents. We say that configuration C changes to C' by an interaction $e = (u, v)$ denoted by $C \xrightarrow{e} C'$ if we have $(C'(u), C'(v)) = T(C(u), C(v))$ and $C(w) = C'(w)$ for all $w \in V \setminus \{u, v\}$. We simply write $C \rightarrow C'$ if there exists $e \in E$ such that $C \xrightarrow{e} C'$. We say that a configuration C' is reachable from C if there exists a sequence of configurations $C_0, C_1, \dots, C_k$ such that $C = C_0$, $C' = C_k$ and $C_i \rightarrow C_{i+1}$ for all $i = 0, 1, \dots, k - 1$. We also say that a set $\mathcal{C}$ of configurations is closed if no configuration outside $\mathcal{C}$ is reachable from any configuration in $\mathcal{C}$.

A scheduler determines which interaction occurs at each time step. We consider the uniformly random scheduler $\Gamma = \Gamma_0, \Gamma_1, \dots$ each $\Gamma_t \in E$ is a random variable such that $\Pr(\Gamma_t = (u_i, u_{(i+1) \bmod n})) = 1/n$ for $t \geq 0$ and $i = 0, 1, \dots, n - 1$. Each Γ_t represents the interaction that occurs at time step t . Given an initial configuration C_0 , the execution of protocol P under Γ is defined as $\Xi_P(C_0, \Gamma) = C_0, C_1, \dots$ such that $C_t \xrightarrow{\Gamma_t} C_{t+1}$ for all $t \geq 0$. We denote $\Xi_P(C_0, \Gamma)$ simply by $\Xi_P(C_0)$ when no confusion occurs.

In this paper, we solve self-stabilizing leader election problem. To simplify, we give the same definition of a self-stabilizing leader election protocol in [32] rather than separately defining self-stabilization and the leader election problem.

Definition 1 (Self-stabilizing Leader Election). [32, Definition 2.1] For any protocol P , we say that a configuration C of P is safe if (i) exactly one agent outputs L (leader) and all other agents output F (follower) in C , and (ii) at every configuration reachable from C , all agents keep the same outputs as those in C . A protocol P is a self-stabilizing leader election (SSLE) protocol if $\Xi_P(C_0, \Gamma)$ reaches a safe configuration with probability 1 for

arbitrary initial configuration C_0 .



If an interaction sequence s consists of l interactions, we say that s has a length of l .

Lemma 2.0.1. [32, LEMMA 2.3] *Let s be a sequence of interactions with length l . Then, s occurs within nl steps in expectation. For any $c \geq 1$, s occurs within $O(cn(l + \log n))$ steps with probability $1 - n^{-c}$.*

We use Lemma 2.0.1 to describe within how many steps an interaction sequence will occur.

Lemma 2.0.2. [32, LEMMA 2.4] *For any SSLE protocol P and a function $f : \mathbb{N} \rightarrow \mathbb{R}$, the convergence time (i.e., the number of steps to reach a safe configuration) of P is $O(f(n))$ steps in expectation if the convergence time of P is bounded by $f(n)$ steps with probability $1 - O(1/n)$.*





Chapter 3 SSLE on Rings

In this section, we propose a population protocol, called P_{PLL} , which can solve SSLE on a directed ring. The execution of P_{PLL} will bring the population to a safe configuration within $O(n^2 \log n \log \log n)$ steps with high probability. Once the system reaches a safe configuration, the population will have a unique leader and keep this unique leader forever.

The execution of P_{PLL} is divided into several phases. We first give a high level overview to the protocol and each of those phases in 3.1.

3.1 Overview

The protocol has three main goals: (i) create leaders if there are none, (ii) decrease the number of leaders to one, and (iii) stabilize the population so that there is only one leader. The execution of the protocol is divided into five phases.

In phase 1, before creating leaders, we need to ensure that, if there is one or more leaders, the number of leaders does not decrease to zero. When there are one or more leaders, they fire bullets to eliminate each other until only one leader remains. The bullets fired by leaders will not eliminate all leaders. However, there may be some bullets in the initial configuration that could potentially eliminate all leaders. These bullets may

cause issues in the execution of subsequent phases. Therefore, in phase 1, we first enter a configuration, called clean configuration, where no such bullet will appear to ensure correct execution in the subsequent phases. After entering the clean configuration, the execution can proceed to phase 2.



In phase 2, the protocol creates leaders if there is no leader in the configuration. The protocol creates leaders in two different ways, which will be introduced in section 3.3. If there is at least one leader in the configuration, the execution can proceed to phase 3.

In phase 3, there will be at least one leader in the configuration, and to prevent the creation of other leaders, the leader will send special commands, called reset commands, to the entire population. When agents receive a reset command, the time to create a leader will be extended. We call the population is suppressed if all agents have received at least one reset command. When the population is suppressed, we can ensure that, over a sufficiently long period, no leaders will be created with high probability. After the population is suppressed, the execution can proceed to phase 4.

In phase 4, the leaders will eliminate each other. We use a strategy similar to [32] to decrease the number of leaders to one, but not to zero. Since the population is suppressed, no new leaders will be created for a sufficiently long period with high probability. Eventually, over the sufficiently long period, there is only one leader with high probability. When only one leader remains, the execution can proceed to phase 5.

In phase 5, when there is only one leader left, this leader will transform the current configuration into a terminal configuration called *perfect* configuration. In a *perfect* configuration, there is at least one leader and no other leaders will be created. When the configuration is *perfect* with only one leader, the leader election is completed.

The execution of P_{PLL} consists of five phases: cleaning, creating leaders, suppressing, eliminating leaders, and stabilizing. We will describe these phases sequentially from section 3.2 to section 3.6.



3.2 Cleaning

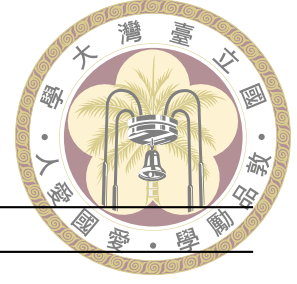
In this section, we explain how to enter clean configuration in phase 1.

In phase 1, we want all bullets that could potentially eliminate all leaders disappear. A configuration that no such bullet will appear is called clean configuration. The definition of clean configuration will be provided in section 4.1. The method to enter a clean configuration is to wait for all such bullets to traverse the ring. The detailed behavior of bullets is related to the strategy to eliminate leaders, which will be explained in section 3.5. After the all these bullets traverse the ring, the population will enter a clean configuration. We will prove it in Lemma 4.1.2.

3.3 Creating Leaders

In this section, we explain how to create leaders.

The protocol creates leaders in two different ways. First, we use the modified SSLE protocol P_ψ in [7], called P'_ψ , as a sub-protocol of P_{PLL} to create leaders. In P'_ψ , each agent maintains a variable *label* $\in [0, \psi - 1]$. We call a pair of adjacent agents (u, v) , where u is on the left of v , a *barrier* if $v.\text{label} \neq (u.\text{label} + 1) \bmod \psi$. Each barrier moves clockwise by changing the label of the agent on the the right of the barrier. When the agent on the right of the barrier is a leader, the barrier stops moving. We call this situation



Algorithm 1: $P_{PLL}(l, r)$

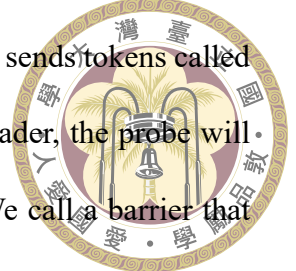
Input: l is the initiator, r is the responder

Variables:

$leader \in \{0, 1\}, probe \in \{0, 1\}, phase \in \{0, 1\},$
 $bit \in \{0, 1\}, wait \in \{0, 1\},$
 $state \in \{produce, done, depart, none,$
 $congest, lock, compare1, compare2,$
 $eliminate_signal, CHECK_ready, equal,$
 $larger, less, toss, toss_done\}$
 $path \in \{\perp\} \cup ([0, \psi - 1] \times \{0, 1\}), // \perp \text{ means no information}$
 $reset \in \{0, 1\}, h \in \{0, 1\}, carry \in \{0, 1\},$
 $count \in [0, \kappa], // \kappa = O(1)$
 $path2 \in \{\perp\} \cup ([0, \psi - 1] \times \{0, 1\} \times [0, \psi] \times [1, 6]),$
 $path3 \in \{\perp\} \cup ([0, \psi - 1] \times \{0, 1\} \times \{0, 1\}),$
 $path4 \in \{\perp\} \cup ([0, \psi - 1] \times \{0, 1\}),$
 $path5 \in \{\perp\} \cup ([0, \psi - 1] \times \{0, 1\}),$
 $p \in \{0, 1\}, p_carry \in \{0, 1\},$
 $reset_cmd \in \{0, 1\}, echo \in \{0, 1\},$
 $signal_B \in \{0, 1\}, bullet \in \{0, 1, 2\}, shield \in \{0, 1\},$
 $write_state \in \{depart, ready, increment_WS,$
 $increment_WS_done, equal, larger, less,$
 $produce, done,$
 $overwrite, move_flip, move, increment_group\}$
 $write \in \{overwrite, move_flip, move, increment_group, none\},$
 $last \in \{0, 1\}, last_set \in \{0, 1\}, last_unset \in \{0, 1\},$
 $b_carry \in \{0, 1\}, gen_ws \in \{0, 1\}, ws_echo \in \{0, 1\},$
 $signal_{ws} \in \{0, 1\}$

- 1 $P'_\psi(l, r)$
 - 2 GENSIG(l, r)
 - 3 CLEAN_SIG(l, r)
 - 4 ELIMINATE_SIG(l, r)
 - 5 SIGMOVE(l, r)
 - 6 CHECK(l, r)
 - 7 DISAPPEAR(l, r)
 - 8 COMPARE(l, r)
 - 9 SUPPRESS_SIG(l, r)
 - 10 EliminateLeaders'(l, r)
 - 11 WRITE_SEQ(l, r)
-

”the barrier being blocked by a leader”. The barrier also continuously sends tokens called *probe* which move counterclockwise. When a probe encounters a leader, the probe will be eliminated. When a probe meets a barrier, a leader is created. We call a barrier that is not blocked by a leader a free barrier. We will prove that a free barrier disappears or becomes barrier blocked by a leader within $O(n^2)$ steps with high probability in Lemma 4.1.3. Therefore, P'_ψ can directly create leaders within $O(n^2)$ steps with high probability when the population size n is not a multiple of ψ .



The pseudocode of P'_ψ is given in Algorithm 2.

Algorithm 2: $P'_\psi(l, r)$

Input: l is the initiator, r is the responder

```

1 if  $r.label \neq (l.label + 1) \bmod \psi$  then // A barrier edge
    // A probe hits a barrier
2   if  $r.probe = 1$  then
3      $r.leader \leftarrow 1$ 
4      $r.label \leftarrow 0$ 
5      $r.probe \leftarrow 0$ 
    // generate probe
6   if  $l.phase = 0$  then
7      $l.phase \leftarrow 1$ 
8     if  $l.leader = 0$  then
9        $l.probe \leftarrow 1$ 
    // advance barrier(clockwise)
    // barrier is blocked by a leader
10  if  $r.leader = 0$  then
11     $r.label \leftarrow (l.label + 1) \bmod \psi$ 
12     $r.phase \leftarrow 0$ 
13 else // A non-barrier edge
    // advance probe (counterclockwise)
14  if  $r.probe = 1$  then
15     $l.probe \leftarrow 1$ 
16     $r.probe \leftarrow 0$ 

```

However, if n is a multiple of ψ , P'_ψ may not be able to create leaders. In such a scenario, there must be no barrier on the ring, which means that all pairs of adjacent agents (u, v) must satisfy that $v.label = (u.label + 1) \bmod \psi$. We use this property to design the

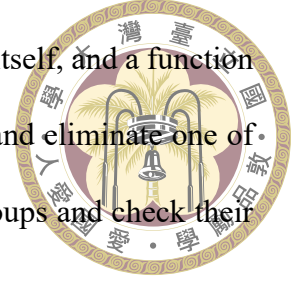


second method for creating a leader. We use a method similar to [7] to divide the population into several groups but with smaller group size and embed a bit in each agent. We call the ψ adjacent agents labeled from 0 to $\psi - 1$ a group and divide the population into n/ψ groups. Each agent maintains a variable $bit \in \{0, 1\}$. All agents within the same group collectively maintain a bit string that serves as the ID for their group. We define the ID of a group $G = u_0, u_1, \dots, u_{\psi-1}$ as $ID(G) = \sum_{i=0}^{\psi-1} b_i \cdot 2^{\psi-1-i}$ where $b_i = u_i.bit$. We call a configuration *perfect* if, when starting from any group without a leader and writing down the ID sequence of groups clockwise until the first and last IDs of the sequence are the same, there exists another group having a greater ID or a leader positioned between the first and last groups of the sequence. If there is a sequence that does not satisfy the conditions, we call it an imperfection. When an imperfection is found, a leader will be created. We check whether the current configuration is *perfect* to create leaders when there is no leader.

To implement this checking process, each group sends a signal carrying its ID, which consists of ψ sub-signals. The signal moves clockwise and compares its ID with the IDs of other groups it encounters. When a signal encounters a group with a greater ID, the signal will be eliminated; when a signal encounters a group with a smaller ID, the signal passes the group and moves to the next group. An imperfection is found if the IDs of signal and group are the same.

However, this method will encounter some issues. First, there may be incomplete signals missing some sub-signals, referred to as partial signals, in the initial configuration. Partial signals may cause issues in the comparing process. Second, while a signal is moving, it may encounter another signal in front of it, blocking its path. This can happen throughout the entire ring, which means that every signal is blocked. We design two func-

tions to resolve these issues: a function that partial signals can clean itself, and a function that the blocked signal can compare its ID with the one blocking it and eliminate one of them. After resolving these issues, the signals can move to other groups and check their IDs. We also design a function for this checking.



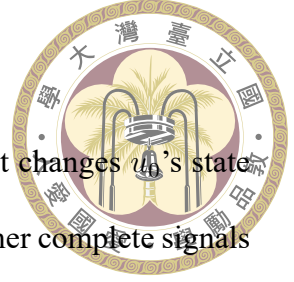
Therefore, the process of this phase is divided into four parts: generating signals, clean partial signals, signals comparison, and signal checking. We will describe these parts sequentially from section 3.3.1 to section 3.3.4.

3.3.1 Generating Signals

In this section, we explain how a group generates a signal.

A group first waits for a certain period before generating a signal. We will explain the reasons for waiting in section 3.4. After the waiting period, the group checks whether there is any signal within the group. If there is a signal within the group, the group waits until the signal exits. After that, the group will generate a signal.

We design a function GENSIG to implement this process. Let a group $G = u_0, u_1, \dots, u_{\psi-1}$, a signal $S = s_0, s_1, \dots, s_{\psi-1}$. GENSIG first waits for a certain period $\Theta(n)$ parallel time. We design a function WAIT to implement the waiting process. WAIT uses the similar method in [32], which implements the lottery game [1]. Each agent u_i in G maintains a variable $h_i \in \{0, 1\}$. We define $hits(G) = \sum_{i=0}^{\psi-1} h_i \cdot 2^i$. Agent u_0 increases h_0 by one when it interacts with its right neighbor. If h_0 overflows, it will carry over to h_1 . Similarly, if h_i overflows, it will carry over to h_{i+1} . However, if agent u_1 interacts with its left neighbor, it resets h_i for all $i \in [0, \psi - 1]$. When $h_{\psi-1}$ overflows, it increases a counter *count* by one. The waiting period ends when *count* = κ where κ is a constant.



The pseudocode of WAIT is given in Algorithm 3.

After waiting, GENSIG starts executing from u_0 . GENSIG first changes u_0 's state to the production state *produce*. If an agent is in the state *produce*, other complete signals are not allowed to move into this agent. The state *produce* will be propagated from left to right by the interaction between u_i and u_{i+1} if $u_i.state = produce$. The propagation of *produce* ends when $u_{\psi-1}$ enters the state *produce*, then the next phase of GENSIG begins. Starting from $u_{\psi-1}$, it will generate a sub-signal $s_{\psi-1}$ and switch to the state *done*. Similar to the state *produce*, the state *done* will be propagated from right to left by the interaction between u_i and u_{i-1} . After u_i propagates the state *done* to u_{i-1} , the sub-signal s_i can begin to move forward. The process of GENSIG ends after u_0 generates sub-signal s_0 .

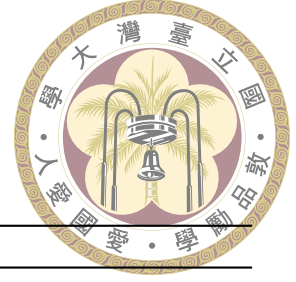
The pseudocode of GENSIG is given in Algorithm 4.

The length of interaction sequence that GENSIG generates signal is exactly $(\psi + \psi) = 2\psi$, which is the length of the process that u_1 propagates state *produce* to u_ψ and u_ψ propagates state *done* to u_1 . Therefore, this interaction sequence occurs within at most $O(n\psi) = O(n \log \log n)$ steps with probability $1 - O(1/n)$ by Lemma 2.0.1.

3.3.2 Cleaning partial signals

In this section, we explain how to clean partial signals.

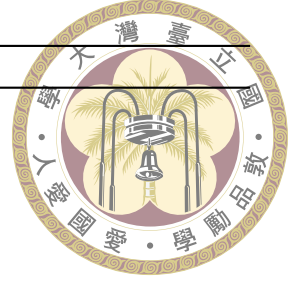
The idea of the cleaning process is to let each sub-signal confirm its preceding and succeeding sub-signals' labels. If a mismatched label is found, it indicates that the signal is a partial signal.



Algorithm 3: WAIT(l, r)

Input: l is the initiator, r is the responder

```
1 if  $r.label = 0$  then
  | //  $u_0$  interacts with its left neighbor
2    $r.reset \leftarrow 1$ 
3    $r.h \leftarrow 0$ 
4 if  $l.label = 0$  then
  | //  $u_0$  interacts with its right neighbor
5   if  $l.h = 1$  then
6     |  $l.h \leftarrow 0$ 
7     |  $l.carry \leftarrow 1$ 
8   else
9     |  $l.h \leftarrow 1$ 
  |
  | // carry over
10 if  $l.carry = 1$  then
11   if  $r.h = 1$  then
12     |  $r.h \leftarrow 0$ 
13     | if  $r.label = \psi - 1$  then
14       |  $r.count \leftarrow r.count + 1$ 
15     | else
16       |  $r.carry \leftarrow 1$ 
17   else
18     |  $r.h \leftarrow 1$ 
  |
  | // reset hit
19 if  $l.reset = 1$  then
20   if  $r.label \neq \psi - 1$  then
21     |  $r.reset \leftarrow 1$ 
22    $l.reset = 0$ 
23    $r.h = 0$ 
  |
  | // waiting process is done
24 if  $r.label = \psi - 1 \wedge r.count = \kappa$  then
25   |  $r.wait \leftarrow 0$ 
26 if  $r.wait = 0 \wedge l.label \neq \psi - 1$  then
27   |  $l.wait = 0$ 
28   |  $r.wait = 1$ 
```



Algorithm 4: GENSIG(l, r)

Input: l is the initiator, r is the responder

```

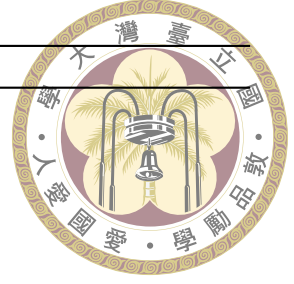
1 WAIT( $l, r$ )
2 if  $l.leader = 0 \wedge l.label = 0 \wedge l.wait = 0 \wedge l.path = \perp$  then
3    $l.state \leftarrow produce$ 
4 if  $r.label \neq 0 \wedge l.state = produce \wedge r.path = \perp$  then
5    $r.state \leftarrow produce$ 
6 if  $r.label = \psi - 1 \wedge r.state = produce$  then
7    $r.state \leftarrow done$ 
8    $r.path \leftarrow (r.label, r.bit)$ 
9 if  $l.label \neq \psi - 1 \wedge l.state = produce \wedge r.state = done$  then
10   $l.state \leftarrow done$ 
11   $l.path \leftarrow (l.label, l.bit)$ 
12   $r.state \leftarrow depart$ 
13 if  $l.label = 0 \wedge l.state = done$  then
14   $l.state \leftarrow depart$ 
15   $l.wait \leftarrow 1$ 

```

We design a function CLEAN_SIG to implement this process. Each sub-signal sends connect signals forward and backward to confirm whether the label of the preceding and succeeding sub-signals is matched. But the sub-signal with label 0 does not need to confirm forward, and similarly, the sub-signal with label $\psi - 1$ does not need to confirm backward. If a mismatch in labels is detected, the signal will be eliminated.

This connect signals can move forward or backward continuously, but it only needs to move at most 2ψ steps to confirm whether it is a partial signal. This is related to the movement of signals. When a signal moves to the next group, each sub-signal will wait until all sub-signals have reached their corresponding positions in this group. Therefore, the maximum distance between each pair of adjacent sub-signals is at most 2ψ .

The pseudocode of CLEAN_SIG is given in Algorithm 5.



Algorithm 5: CLEAN_SIG(l, r)

Input: l is the initiator, r is the responder

```

1 for  $u \in \{l, r\}$  do // generate connect
2   if  $u.path \neq \perp$  then
3     if  $u.path[1] \neq 0$  then
4        $u.path4 \leftarrow (u.path[1], 1)$ 
5     if  $u.path[1] \neq \psi - 1$  then
6        $u.path5 \leftarrow (u.path[1], 1)$ 
7 if  $r.path4 \neq \perp \wedge r.path4[1] \neq -1 \wedge l.path \neq \perp$  then // check
8   if  $r.path4[1] \neq l.path[1] + 1$  then // wrong connection, send a
    clean command back
9      $r.path5 \leftarrow (-1, 1 - r.path4[2])$ 
10     $r.path4 \leftarrow \perp$ 
11 if  $l.path5 \neq \perp \wedge l.path5[1] \neq -1 \wedge r.path \neq \perp$  then
12   if  $l.path5[1] \neq r.path[1] - 1$  then // wrong connection, send a
    clean command back
13      $l.path4 \leftarrow (-1, 1 - r.path5[2])$ 
14     $l.path5 \leftarrow \perp$ 
15 if  $r.path4 \neq \perp$  then // propagate connect to left
16    $l.path4 \leftarrow r.path4$ 
17    $r.path4 \leftarrow \perp$ 
18   if  $l.label = 0$  then
19     if  $l.path4[2] = 1$  then
20        $l.path4[2] \leftarrow 0$ 
21     else
22        $l.path4 \leftarrow \perp$ 
23 if  $l.path5 \neq \perp$  then // propagate connect to right
24    $r.path5 \leftarrow l.path5$ 
25    $l.path5 \leftarrow \perp$ 
26   if  $r.label = \psi - 1$  then
27     if  $r.path5[2] = 1$  then
28        $r.path5[2] \leftarrow 0$ 
29     else
30        $r.path5 \leftarrow \perp$ 
31 for  $u \in \{l, r\}$  do // clean signal
32   if  $u.path \neq \perp$  then
33     if  $u.path4[1] = -1$  then
34       if  $u.path[1] = 0$  then
35          $u.path4 \leftarrow \perp$ 
36        $u.path \leftarrow \perp$ 
37     if  $u.path5[1] = -1$  then
38       if  $u.path[1] = \psi - 1$  then
39          $u.path5 \leftarrow \perp$ 
40        $u.path \leftarrow \perp$ 

```

3.3.3 Signal Comparison

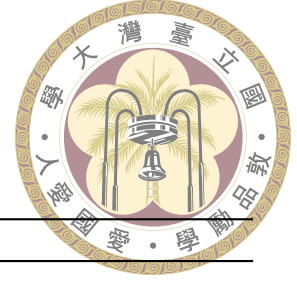


In this section, we explain how the blocked signal compares its ID with the one blocking it.

If two signals are adjacent, and the signal in front is not moving, the signal behind will compare their IDs. The one with the smaller ID will be eliminated. If the IDs are equal, the one behind will be eliminated.

We design a function COMPARE to implement this process. COMPARE compares bits from MSB to LSB one by one so that it can determine which one is larger by comparing only one bit at a time. Let the signal $S = s_0, s_1, \dots, s_{\psi-1}$ be behind and the signal $S' = s'_0, s'_1, \dots, s'_{\psi-1}$ be in front. Before comparison, it is necessary to stop the movement of S' , otherwise, even if there is no congestion, adjacent signals will start comparing as soon as they meet. S will first put S' into the *lock* state. After S' is locked, COMPARE begins to compare their IDs bit by bit. Let $b_i = s_i.bit$ and $b'_i = s'_i.bit$ where $i \in [0, \psi-1]$. In the case of $b_i = b'_i$, the comparison of the next bit (b_{i+1} and b'_{i+1}) begins. If b_ψ and b'_ψ are equal, eliminate S . In the case of $b_i > b'_i$, since the ID of S' is smaller, S' will be eliminated. In the case of $b_i < b'_i$, since the ID of S' is greater, S will be eliminated. The process of signal elimination is implemented by the function ELIMINATE_SIG. Assuming that at position s'_i it is decided to eliminate S , s'_i will transmit the information about eliminating S to s_ψ . Then, elimination will start from s_ψ and continue one by one until s_1 is eliminated. If the one to be eliminated is S' , then s'_i will transmit this information to s'_ψ and use the same method to eliminate S' .

The pseudocode of COMPARE and ELIMINATE_SIG are given in Algorithm 6 and 9, respectively.



Algorithm 6: COMPARE(l, r)

Input: l is the initiator, r is the responder

```
1 if  $l.path = \perp$  then
2   |  $l.state \leftarrow none$ 
3 else
4   | if  $r.path \neq \perp$  then
5     | CMP_UPDATE( $l, r$ )
6     | if  $l.path2 \neq \perp \vee r.path2 \neq \perp$  then
7       | COMPARE_BIT( $l, r$ )
```

Algorithm 7: CMP_UPDATE(l, r)

Input: l is the initiator, r is the responder

```
1 if  $l.label = 0$  then
2   |  $l.state \leftarrow congest$ 
3 if  $l.label = \psi - 1 \wedge l.state = congest$  then
4   |  $r.state \leftarrow lock$ 
5 else if  $r.label = \psi - 1 \wedge l.state = lock$  then
6   |  $r.state \leftarrow compare1$ 
7 else if  $l.label = \psi - 1 \wedge r.state = compare1$  then
8   |  $l.state \leftarrow compare2$ 
9 else
10  | if  $l.state \in \{congest, lock\}$  then
11    |  $r.state \leftarrow l.state$ 
12  | else if  $r.state \in \{compare1, compare2\}$  then
13    |  $l.state \leftarrow r.state$ 
14 if  $l.label = 0 \wedge l.state = compare2$  then
15  |  $l.path2 \leftarrow (l.path[1], l.path[2], \psi, 1)$ 
```



Algorithm 8: COMPARE_BIT(l, r)

Input: l is the initiator, r is the responder

```

// compare two bits
// path2[4]:
// 1: move right
// 2: move left (compare the next bit)
// 3: eliminate the left signal
// 4: eliminate the right signal
1 if r.path2[3] = 0 then
2   if r.bit > r.path2[2] then
3     // eliminate S
4     r.path2 ← (0, 0, 0, 3)
5   else if r.bit < r.path2[2] then
6     // eliminate S'
7     r.path2 ← (0, 0, 0, 4)
8   else
9     if r.label =  $\psi$  then
10      // eliminate S
11      r.path2 ← (0, 0, 0, 3)
12    else
13      // back to S then compare the next bit
14      r.path2 ← (0, 0,  $\psi - 1$ , 2)
15  // compare the next bit
16 if l.path2[4] = 2  $\wedge$  l.path2[3] = 0 then
17   l.path2 ← (l.path[1], l.path[2],  $\psi$ , 1)

```

Algorithm 9: ELIMINATE_SIG(l, r)

Input: l is the initiator, r is the responder

```

1 if r.state = eliminate_signal then
2   if l.label  $\neq \psi - 1$  then
3     l.state ← eliminate_signal
4   r.path ←  $\perp$ 
5   r.state ← none

```



Algorithm 10: SIGMOVE(l, r)

Input: l is the initiator, r is the responder

// move the generated signal

```

1 if  $l.path \neq \perp \wedge r.path = \perp$  then
2   if  $l.state \notin \{lock, compare1, CHECK\_ready\}$  then
3     if  $r.state \neq produce \vee l.path[1] \neq \psi - 1$  then
4        $r.path \leftarrow l.path$ 
5        $l.path \leftarrow \perp$ 
6        $l.state \leftarrow none$ 

// move the bit comparison information
7 if  $l.path2[4] = 1 \wedge l.path2[3] \neq 0$  then
8    $r.path2 \leftarrow l.path2$ 
9    $r.path2[3] \leftarrow r.path2[3] - 1$ 
10   $l.path2 \leftarrow \perp$ 
11 if  $r.path2[4] = 2 \wedge r.path2[3] \neq 0$  then
12   $l.path2 \leftarrow r.path2$ 
13   $l.path2[3] \leftarrow l.path2[3] - 1$ 
14   $r.path2 \leftarrow \perp$ 
15 if  $r.path2[4] \in \{3, 6\}$  then
16   $l.path2 \leftarrow r.path2$ 
17   $r.path2 \leftarrow \perp$ 
18 if  $l.path2[4] \in \{4, 5\}$  then
19   $r.path2 \leftarrow l.path2$ 
20   $l.path2 \leftarrow \perp$ 

// control the state after comparing bits
21 for  $u \in \{l, r\}$  do
22   // eliminate signal
23   if  $u.label = \psi - 1 \wedge u.path2[4] \in \{3, 4\}$  then
24      $u.state \leftarrow eliminate\_signal$ 
25   // signal starts moving
26   if  $u.label = \psi - 1 \wedge u.path2[4] = 5$  then
27      $u.state \leftarrow depart$ 
28   // the lottery game after checking
29   if  $u.label = 0 \wedge u.path2[4] = 6$  then
30      $u.state \leftarrow toss$ 

28 if  $r.state = depart$  then
29   if  $l.label < \psi - 1 \wedge l.state = CHECK\_ready$  then
30      $l.state \leftarrow depart$ 

```

3.3.4 Signal Checking



In this section, we explain how a signal compares its ID with the group it arrives at.

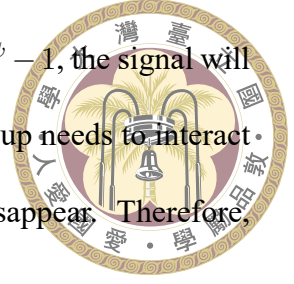
When a signal $S = s_0, s_1, \dots, s_{\psi-1}$ moves to a group $G = u_0, u_1, \dots, u_{\psi-1}$, they will compare their IDs. If the ID of S is smaller, S will be eliminated. If the ID of S is greater, S will move to the next group. If the ID of S and the ID of G are equal, which means that an imperfection is found, we create a leader at the position where the imperfection is located.

We design a function CHECK to implement this process. CHECK begins when each sub-signal is on the corresponding agent (s_i is on u_i). Before the comparison starts, the group G enters the state *check* to ensure no other signal wants to start COMPARE with S . Similar to COMPARE, we use the bit by bit comparison method to compare their IDs. In the case of $ID(S) < ID(G)$, S will be eliminated. In the case of $ID(S) > ID(G)$, S continues to move forward. In the case of $ID(S) = ID(G)$, a leader will be created.

However, there may be a signal with an ID that is greater than all IDs in the ring. This could lead to the signal continuously eliminating other signals without creating a leader. To avoid this situation, we design an additional mechanism where the signal will disappear with some probability after being compared with the group.

We design a function DISAPPEAR to implement this process. DISAPPEAR begins when the ID of the signal is greater than the ID of the group. Similar to WAIT, the group implement a lottery game. Each agent u_i in G maintains a variable $p_i \in \{0, 1\}$. We define $P(G) = \sum_{i=0}^{\psi-1} p_i \cdot 2^i$. The group G increases $P(G)$ like WAIT but stop increasing when u_0 interact with its left neighbor. When the increasing stops, the group decides whether

the signal should disappear based on the value of $P(G)$. If $P(G) = 2^\psi - 1$, the signal will disappear, otherwise, the signal will move to the next group. The group needs to interact continuously with its right neighbor 2^ψ times to make the signal disappear. Therefore, there is a $1/2^{2^\psi} = 1/n$ probability that the signal will disappear.



The pseudocode of CHECK and DISAPPEAR are given in Algorithm 11 and 13, respectively.

Algorithm 11: CHECK(l, r)

Input: l is the initiator, r is the responder

```

1 for  $u \in \{l, r\}$  do
2   if  $u.label = u.path[1] \wedge u.state \notin \{depart, equal, larger, less\}$  then
3      $u.state \leftarrow CHECK\_ready$ 
4 if  $l.label = 0 \wedge l.path[1] = 0 \wedge l.state = CHECK\_ready$  then
5    $l.state \leftarrow CHECK\_BIT(l.bit, l.path[2])$ 
6 if  $l.label < \psi - 1 \wedge l.state = equal$  then
7   if  $r.state = CHECK\_ready$  then
8      $r.state \leftarrow CHECK\_BIT(r.bit, r.path[2])$ 
9 for  $u \in \{l, r\}$  do
10  if  $u.state = larger$  then
11     $// ID(G) > ID(S)$ , eliminate signal
12     $u.path2 \leftarrow (0, 0, 0, 4)$ 
13  if  $u.state = less$  then
14     $// ID(G) > ID(S)$ ,  $S$  implement a lottery game
15     $u.path2 \leftarrow (0, 0, 0, 6)$ 
     $// ID(G) = ID(S)$ , create leader
16 if  $r.label = \psi - 1 \wedge r.state = equal$  then
17    $r.leader \leftarrow 1$ 

```

Algorithm 12: CHECK_BIT($b1, b2$)

```

1 if  $b1 = b2$  then
2   return  $equal$ 
3 if  $b1 > b2$  then
4   return  $larger$ 
5 if  $b1 < b2$  then
6   return  $less$ 

```



Algorithm 13: DISAPPEAR(l, r)

Input: l is the initiator, r is the responder

```

1 if  $r.state = toss \wedge r.label = 0$  then
  | //  $u_0$  interacts with its left neighbor
2    $r.state \leftarrow toss\_done$ 
3    $r.path2[4] \leftarrow (0, 0, 0, 5)$ 
4    $r.p \leftarrow 0$ 
5 if  $l.state = toss \wedge l.label = 0$  then
  | //  $u_0$  interacts with its right neighbor
6   if  $l.p = 1$  then
7     |  $l.p \leftarrow 0$ 
8     |  $l.p\_carry \leftarrow 1$ 
9   else
10    |  $l.p \leftarrow 1$ 
    |
    | // carry over
11 if  $l.p\_carry = 1$  then
12   if  $r.p = 1$  then
13     |  $r.p \leftarrow 0$ 
14     | if  $r.label = \psi - 1$  then
15       |  $r.state \leftarrow eliminate\_signal$ 
16     | else
17       |  $r.p\_carry \leftarrow 1$ 
18   else
19     |  $r.p \leftarrow 1$ 
  
```

3.4 Suppressing

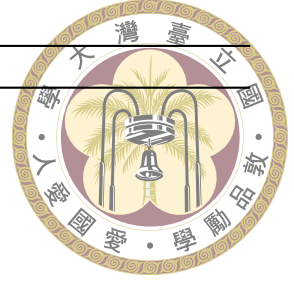


In this section, we explain how a leader suppresses the population.

The main goal of this phase is to prevent the creation of other leaders when there is already a leader in the ring. To achieve this, leaders send special reset commands that move clockwise to each group to reset the waiting time before producing a signal. We call the population is suppressed if all agents have received at least one reset command. When a reset command encounter a signal, the signal will be eliminated. If there is no signal, the comparison of signals and groups will not occur, thus no leader will be created. The reset command requires $\Theta(n)$ parallel time to be sent to every group in the ring. Therefore, we set the waiting time before generating a signal to $\Theta(n)$ parallel time. However, the reset command may exist in some initial configuration without a leader, leading to the situation where a leader will never be created. To resolve this issue, when a group receives a reset command, it sends an echo counterclockwise; when an echo meets a reset command, they eliminate each other.

We design a function `SUPPRESS_SIG` to implement this process. A leader continuously sends the reset command, which moves clockwise. The reset command eliminates all signals it encounters. When a group $G = u_0, u_1, \dots, u_{\psi-1}$ receives a reset command, $u_{\psi-1}$ resets the counter *count* of G to zero. Additionally, when $u_{\psi-1}$ receives a reset command, it sends an echo counterclockwise. If an echo encounters a leader, it will be absorbed. And when an echo encounters a reset command, they will eliminate each other.

The pseudocode of `SUPPRESS_SIG` is given in Algorithm 14.



Algorithm 14: SUPPRESS_SIG(l, r)

Input: l is the initiator, r is the responder

```

1 if  $l.leader = 1$  then
2    $l.reset\_cmd \leftarrow 1$ 
3 if  $l.reset\_cmd = 1$  then
4    $l.reset\_cmd \leftarrow 0$ 
5    $r.reset\_cmd \leftarrow 1$ 
6 if  $r.label = \psi - 1 \wedge r.reset\_cmd = 1$  then
7    $r.count \leftarrow 0$ 
8    $l.echo \leftarrow 1$ 
9 if  $r.echo = 1$  then
10  if  $r.reset\_cmd = 1$  then
11     $r.echo \leftarrow 0$ 
12     $r.reset\_cmd \leftarrow 0$ 
13  else
14     $r.echo \leftarrow 0$ 
15     $l.echo \leftarrow 1$ 
16 if  $l.leader = 1$  then
17    $l.echo \leftarrow 0$ 

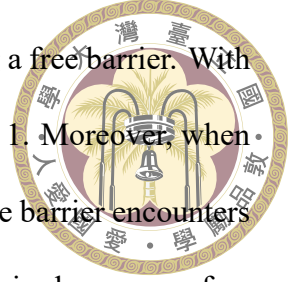
```

3.5 Eliminating Leaders

In this section, we explain how to eliminate redundant leaders.

We use the modified function `EliminateLeaders()` in [32], call `EliminateLeaders'()`, to eliminate redundant leaders. The basic strategy is the same. Leaders may have shields; a leader with a shield will not be killed. A leader can fire two types of bullets: live bullets and dummy bullets. A live bullet can kill a leader without a shield, while a dummy bullet cannot. When a leader fires a live bullet, they also generate a shield. Conversely, when a leader fires a dummy bullet, they simultaneously remove the shield. When a leader decides to fire a bullet, it fires a live bullet or a dummy bullet both with probability $1/2$.

We modify the direction of bullet movement and bullet absence signal movement. Bullets move counterclockwise; bullet absence signals move clockwise. We also set the



bullets and bullet absence signals to be eliminated when encountering a free barrier. With these changes, non-peaceful bullets will be eliminated faster in phase 1. Moreover, when eliminating redundant leaders, there will not be a scenario where a free barrier encounters a probe. When a live bullet kills a leader blocking a barrier, the barrier becomes a free barrier. However, this free barrier will not encounter a probe because its nearest right leader, called rl , will block all probes. Additionally, rl will not be killed because it just fired a live bullet, giving it a shield. The shield will not be removed because the free barrier will block all bullet absence signals. Therefore, the free barrier moves clockwise and is eventually blocked by rl . The original leader elimination function is not affected by barriers. The detailed proof is provided in Lemma 4.1.9 and Lemma 4.1.10. Therefore, `EliminateLeaders'()` can reduce the number of leaders to one, but not to zero.

The pseudocode of `EliminateLeaders'` is given in Algorithm 15.

3.6 Stabilizing

In this section, we show how a leader stabilize the population.

When there is only one leader left, this leader needs to make the configuration *perfect*, as there is still a possibility of creating new leaders in an imperfect configuration. The remaining leader will rewrite the IDs of all groups, making the entire ring become a stable ID sequence. If n is not a multiple of ψ , there will be an incomplete group of size $< \psi$ on the left of the unique leader. We set that when the leader is created, the leader will automatically move to the agent with label 0. And we set a signal to be eliminated when it encounters the leader or the incomplete group on the left of the leader. In this way we don't have to consider this incomplete group. Let the stable ID sequence $S =$



Algorithm 15: EliminateLeaders'(l, r)

Input: l is the initiator, r is the responder

```

1 if  $l.leader = l.signal_B = 1$  then
  | // live bullet
2 |  $(l.bullet, l.shield, l.signal_B) \leftarrow (2, 1, 0)$ 
3 if  $r.leader = r.signal_B = 1$  then
  | // dummy bullet
4 |  $(r.bullet, r.shield, r.signal_B) \leftarrow (1, 0, 0)$ 
5 if  $r.bullet > 0 \wedge l.leader = 1$  then
  | // live bullet kills a leader and live bullet transforms
  | // into dummy bullet
6 | if  $r.bullet = 2 \wedge l.shield = 0$  then
7 | |  $l.leader \leftarrow 0$ 
8 | |  $r.bullet \leftarrow 1$ 
9 else if  $r.bullet > 0 \wedge l.leader = 0$  then
  | // a bullet will be blocked by a barrier
10 | if  $r.label = (l.label + 1) \bmod \psi$  then
11 | |  $l.bullet \leftarrow r.bullet$ 
12 | |  $r.bullet \leftarrow 0$ 
13 | |  $l.signal_B \leftarrow 0$ 
14 | else
15 | |  $r.bullet \leftarrow 0$ 
  | // bullet absence signal will be blocked by a barrier
16 if  $r.label = (l.label + 1) \bmod \psi \vee l.leader = 1$  then
17 |  $r.signal_B = \max(l.signal_B, r.signal_B, l.leader)$ 

```

$s_0, s_1, s_2, \dots, s_{n/\psi-1}$ where s_0 is the ID of the group with the leader. For $i \neq 0$, $s_i = i \cdot 2^x$ where x is the number of trailing zeros of i in binary numeral system. We design a function WRITE_SEQ to write the stable ID sequence. The leader will generate special signals, called overwriting signals, to overwrite the IDs of other groups. The overwriting process is divided into ψ rounds. After ψ rounds of overwriting completed, the process of writing a stable sequence ends, and the leader election is also completed.

In i -th round, the leader will generate overwriting signals with ID i where $i \in [0, \log n - 1]$. Each overwriting signal has a bit $wb \in \{\text{write}, \text{skip}\}$, which is set to *write* when it is generated. The overwriting signals will move clockwise. When there is another overwriting signal ahead, the one behind will wait until the one in front has moved forward before advancing. When an overwriting signal encounters the leader, the overwriting signal will disappear.

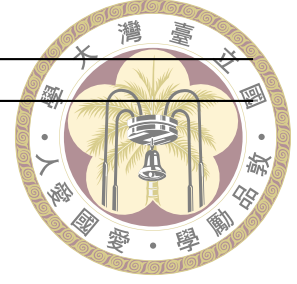
During the process of an overwriting signal WS traversing the ring, it will compare its ID with each encountered group G . In the case that $ID(WS) > ID(G)$, WS skips this group. In the case that $ID(WS) = ID(G)$, if $wb = \text{write}$, wb switches to *skip*; if $wb = \text{skip}$, the ID of this group increases by one and wb switches to *write*. In the case that $ID(WS) < ID(G)$, if $wb = \text{write}$, wb switches to *skip* and $ID(G)$ is overwritten to $ID(WS)$; if $wb = \text{skip}$, wb switches to *write*. After the comparison of IDs, WS continues to move to the next group. When WS finishes traversing and returns to the leader's group, WS will be eliminated and the leader will generate another overwriting signal with $ID = ID(g_0)$ where g_0 is the group that the leader is located. Then the $ID(g_0)$ will be incremented by one. The next round will then begin. This process continues for a total of $\log n$ rounds.

The leader will send overwriting absence signals counterclockwise, similar to the mechanism of firing bullets. When an overwriting absence signal encounters an overwriting signal, the overwriting absence signal will disappear. When an overwriting absence signal encounters the leader, the leader will generate overwriting signals with $ID = ID(g_0)$ where g_0 is the group that the leader is located. Then the $ID(g_0)$ will be incremented by one.

Similar to the reset command, there may be some overwriting signals in the initial configuration. We use the same method that when an overwriting signal leaves a group, it sends an echo counterclockwise. We call this echo the overwriting signal echo. The overwriting signal echo will eliminate all overwriting signals it meets and be eliminated when it meets a leader.

After $O(\log n)$ rounds, the ID sequence starts from the leader's group is a stable ID sequence. Then the leader election is completed.

The pseudocode of WRITE_SEQ is given in Algorithm 16.



Algorithm 16: WRITE_SEQ(l, r)

Input: l is the initiator, r is the responder

// adjust the position of leader

- 1 **if** $r.leader = 1 \wedge r.label \neq 0$ **then**
- 2 $l.leader = 1$
- 3 $r.leader = 0$
- // information of the last group
- 4 GROUP_INFO(l, r)
- 5 **if** $l.leader = 1 \wedge l.gen_ws = 1$ **then**
- 6 GENSIG_WRITESEQ(l, r)
- 7 **for** $u \in \{l, r\}$ **do**
- 8 **if** $u.path3 \neq \perp \wedge u.label = u.path3[1] \wedge u.write_state \neq depart$ **then**
- 9 $u.write_state \leftarrow ready$
- 10 **else**
- 11 $u.write_state \leftarrow none$
- 12 **if** $r.label = 0 \wedge r.write_state = ready$ **then**
- 13 $r.write_state \leftarrow CHECK_BIT(r.bit, r.path3[2])$
- 14 **if** $l.label < \psi - 1 \wedge l.write_state = equal \wedge r.write_state = ready$ **then**
- 15 $r.write_state \leftarrow CHECK_BIT(r.bit, r.path3[2])$
- 16 **for** $u \in \{l, r\}$ **do**
- 17 **if** $u.write_state = larger$ **then**
- 18 // $ID(G) > ID(WS)$
- 19 **if** $u.path3[3] = 1$ **then**
- 20 // overwrite and flip write bit
- 21 $u.write \leftarrow overwrite$
- 22 **else**
- 23 // move forward and flip write bit
- 24 $u.write \leftarrow move_flip$
- 25 **if** $u.write_state = less$ **then**
- 26 // $ID(G) < ID(WS)$, move forward
- 27 $u.write \leftarrow move$
- 28 **if** $r.label = \psi - 1 \wedge r.write_state = equal$ **then**
- 29 // $ID(G) = ID(WS)$
- 30 **if** $r.path3[3] = 1$ **then**
- 31 // move forward and flip write bit
- 32 $r.write \leftarrow move_flip$
- 33 **else**
- 34 // $ID(G) \leftarrow ID(G) + 1$ and flip write bit
- 35 $r.write \leftarrow increment_group$
- // writing actions
- 29 WRITE_ACTION(l, r)
- // move writing signal
- 30 MOVE_WSIG(l, r)



Algorithm 17: GROUP_INFO(l, r)

Input: l is the initiator, r is the responder

```
1 if  $r.leader = 1$  then
2   if  $r.label \neq (l.label + 1) \bmod \psi$  then
3      $l.last\_set \leftarrow 1$ 
4   else
5      $l.last\_unset \leftarrow 1$ 
6 if  $r.last\_set = 1$  then
7    $r.last \leftarrow 1$ 
8   if  $l.label \neq \psi - 1$  then
9      $l.last\_set \leftarrow 1$ 
10  else
11     $l.last\_unset \leftarrow 1$ 
12 if  $r.last\_unset = 1$  then
13    $r.last \leftarrow 0$ 
14   if  $l.leader \neq 1$  then
15      $l.last\_unset \leftarrow 1$ 
16    $r.last\_unset \leftarrow 0$ 
```



Algorithm 18: GENSIG_WRITESEQ(l, r)

Input: l is the initiator, r is the responder

```

1 if  $l.label = 0 \wedge l.gen\_ws = 1 \wedge l.path3 = \perp$  then
2    $l.write\_state \leftarrow produce$ 
3 if  $r.label \neq 0 \wedge l.write\_state = produce \wedge r.path3 = \perp$  then
4    $r.write\_state \leftarrow produce$ 
5 if  $r.label = \psi - 1 \wedge r.write\_state = produce$  then
6    $r.write\_state \leftarrow done$ 
7    $r.path3 \leftarrow (r.label, r.bit, 1)$ 
8   if  $r.bit = 1$  then
9      $r.b\_carry \leftarrow 1$ 
10   $r.bit \leftarrow 1 - r.bit$ 
11 if  $l.label \neq \psi - 1 \wedge l.write\_state = produce \wedge r.write\_state = done$  then
12    $l.write\_state \leftarrow done$ 
13    $l.path3 \leftarrow (l.label, l.bit, 1)$ 
14   if  $r.b\_carry = 1$  then
15     if  $l.bit = 1$  then
16        $l.b\_carry \leftarrow 1$ 
17        $l.bit \leftarrow 1 - l.bit$ 
18        $r.b\_carry \leftarrow 0$ 
19    $r.write\_state \leftarrow depart$ 
20 if  $l.label = 0 \wedge l.write\_state = done$  then
21    $l.write\_state \leftarrow depart$ 
22    $l.gen\_ws \leftarrow 0$ 

```



Algorithm 19: WRITE_ACTION(l, r)

Input: l is the initiator, r is the responder

```

1 for  $u \in \{l, r\}$  do
2   if  $u.label = \psi - 1$  then
3      $u.write\_state \leftarrow u.write$ 
4      $u.write \leftarrow none$ 
5 if  $l.label < r.label \vee r.label = 0$  then
6   if  $r.write\_state = overwrite$  then
7      $r.bit \leftarrow r.path3[2]$ 
8      $r.path3[3] \leftarrow 1 - r.path3[3]$ 
9   if  $r.write\_state = move\_flip$  then
10     $r.path3[3] \leftarrow 1 - r.path3[3]$ 
11  if  $r.write\_state = increment\_group$  then
12    if  $r.bit = 1$  then
13       $r.bit \leftarrow 0$ 
14    else
15       $r.bit \leftarrow 1$ 
16       $r.write\_state \leftarrow depart$ 
17  if  $r.label \neq 0$  then
18     $l.write\_state \leftarrow r.write\_state$ 
19   $r.write\_state \leftarrow depart$ 

```



Algorithm 20: MOVE_WSIG(l, r)

Input: l is the initiator, r is the responder

```
1 if  $r.leader = 1$  then
2   if  $l.path3[1] = 0$  then
3      $r.gen\_ws \leftarrow 1$ 
4    $l.path3 \leftarrow \perp$ 
5    $l.write \leftarrow none$ 
6    $l.write\_state \leftarrow none$ 
7 if  $l.label = l.path3[1] \wedge l.last \leftarrow 1$  then
8    $l.path3 \leftarrow \perp$ 
9    $l.write \leftarrow none$ 
10   $l.write\_state \leftarrow none$ 
11 if  $l.path3 \neq \perp \wedge r.path3 = \perp \wedge l.write\_state = depart$  then
12   if  $r.ws\_echo \neq 1$  then
13      $r.path3 \leftarrow l.path3$ 
14      $r.write\_state \leftarrow l.write\_state$ 
15      $l.path3 \leftarrow \perp$ 
16      $l.write\_state \leftarrow none$ 
17      $l.write \leftarrow none$  if  $l.label = 1 \wedge l.leader \neq 1$  then
18        $l.ws\_echo \leftarrow 1$ 
19   else
20      $l.path3 \leftarrow \perp$ 
21      $l.write \leftarrow none$ 
22      $l.write\_state \leftarrow none$ 
23 if  $l.label < r.label$  then
24    $r.write \leftarrow l.write$ 
25    $l.write \leftarrow none$ 
26 if  $r.leader = 1$  then
27    $l.signal_{ws} \leftarrow 1$ 
28 if  $r.signal_{ws} = 1$  then
29   if  $l.leader \neq 1$  then
30      $l.signal_{ws} \leftarrow 1$ 
31   else
32      $l.gen\_ws \leftarrow 1$ 
33    $r.signal_{ws} \leftarrow 0$ 
34 if  $r.ws\_echo = 1$  then
35   if  $l.leader = 1$  then
36      $r.ws\_echo \leftarrow 0$ 
37   else
38      $l.ws\_echo \leftarrow 1$ 
39      $r.ws\_echo \leftarrow 0$ 
```





Chapter 4 Correctness and Convergence time

In this section, we prove that P_{PLL} performs SSLE on rings of arbitrary size n given knowledge $\psi = \lceil \log \log n \rceil + O(1)$ and the convergence time of P_{PLL} is $O(n^2 \log n \log \log n)$ steps with probability $1 - O(1/n)$. We first define safe configurations in Section 4.1. Next, we prove that the population reaches a safe configuration within $O(n^2 \log n \log \log n)$ steps with probability $1 - O(1/n)$ starting from arbitrary configuration.

4.1 Safe Configurations

Definition 2 (safe configuration). *A configuration is a safe configuration if it satisfies the following conditions:*

1. *There is only one leader in the configuration.*
2. *There is no non-peaceful bullet in the configuration.*
3. *There is no free barrier in the configuration.*
4. *There is no free overwriting signal in the configuration.*
5. *In the configuration, the ring is a stable ID sequence that starts from the leader.*

We will provide the definition of free overwriting signal before the start of Lemma 4.1.12.



We define the peaceful bullet similarly to [32] but with some additional modifications. We say that a live bullet located in the agent u_i is peaceful when the following conditions satisfied:

1. $d_{RL}(i) \neq \infty$.
2. $u_{i+d_{RL}}.shield = 1$.
3. $\forall j = 0, 1, \dots, d_{RL}(i) : u_{i+j}.signal_B = 0 \wedge u_{i+j+1}.label = (u_{i+j}.label + 1) \bmod \psi$.

We also similarly define 3 sets of configurations C_{PB} , C_{NoLB} and C_{NoBAS} as in [32] :

- C_{PB} : A set of configurations where there is one or more leader and all live bullets are peaceful.
- C_{NoLB} : A set of configurations where there are no live bullets.
- C_{NoBAS} : A set of configurations where there are no bullet absence signal.

We also define the clean configuration C_{clean} in phase 1 as $C_{PB} \cup C_{NoLB} \cup C_{NoBAS}$.

Lemma 4.1.1. *The safe configuration is closed.*

Proof. A configuration is safe if it satisfies the five conditions above. If there is no non-peaceful bullet and only one leader in the configuration, no non-peaceful bullet will be generated. Therefore, if a configuration is safe, condition 2 will be satisfied forever. If there is no free barrier and the ring is a stable ID sequence that starts from the unique

leader, no new leaders will be created. Therefore, if a configuration is safe, condition 1 will be satisfied forever. When condition 1 is always satisfied, no free barrier will appear. Therefore, if a configuration is safe, condition 3 will be satisfied forever. If there is no free overwriting signal and only one leader in the configuration, no free overwriting signal will appear. Therefore, if a configuration is safe, condition 4 will be satisfied forever. When condition 1 and 4 are always satisfied, the ID sequence starts from the leader will not change. Therefore, if a configuration is safe, condition 5 will be satisfied forever. In summary, if a configuration is safe, it will remain safe forever. Therefore, the safe configuration is closed. □



4.1.1 Convergence

In phase 1, we need to ensure that when there is one or more leaders, the number of leaders does not decrease to zero. To achieve this goal, we first bring the population into a configuration $C \in C_{clean}$. Once the population reaches $C \in C_{clean}$, we can ensure that as long as there is at least one leader, the number of leaders will not be reduced to zero. This will be proven in Lemma 4.1.5. Here, we prove that in phase 1, the population enters the clean configuration within $O(n^2)$ steps with probability $1 - O(1/n)$.

Lemma 4.1.2. *From any configuration C_0 , $\Xi_{P_{LL}}(C_0, \Gamma)$ enters C_{clean} within $O(n^2)$ steps with probability $1 - O(1/n)$.*

Proof. Assume that there are one or more live bullets in the current configuration. Let b be any one of these live bullets. We let b traverse the ring. In the traverse, b will either visit all agents or disappear. If b survives after the traverse, it means that b did not encounter a leader or a barrier and cleared all bullet absence signals. At this point, if there is one or

more leaders, these leaders must have been newly created at positions that b has already passed. Therefore, these leaders have shields, and there are no bullet absence signals or barriers from b 's position to the first leader on its left. Thus, b is peaceful. If there is no leader, b must clear all bullet absence signals. Thus, there are no bullet absence signals in the configuration. If b encounters a leader or a barrier, b will be eliminated. The length of interaction sequence for b traversing the ring is n . By Lemma 2.0.1 with $c = 2$, b traverses the ring within $O(n^2)$ steps with probability $1 - O(1/n^2)$. By the union bound, all live bullets traverse the ring within $O(n^2)$ steps with high probability $1 - O(1/n)$. We can conclude that within $O(n^2)$ steps, with probability $1 - O(1/n)$, $\Xi_{P_{LL}}(C, \Gamma)$ enters C_{clean} . $\square$

In phase 2, we need to create at least one leader if there is no leader. Because the population has already entered a clean configuration, once there is a leader, the number of leaders will not be reduced to zero. After phase 2, we can ensure that there is always at least one leader in the population. Here, we will prove that both methods of creating a leader can create a leader within $O(n^2 \log n)$ steps with probability $1 - O(1/n)$.

Lemma 4.1.3. *From any configuration, a free barrier meets a probe or a leader within $O(n^2)$ steps with probability $1 - O(1/n)$.*

Proof. Assume there are one or more free barriers in the current configuration. Each free barrier moves clockwise and sends probes that move counterclockwise. Let fb be any one of these free barriers. We let fb traverse the ring. In this traverse, fb will encounter either a probe, a leader, or another barrier. If fb encounters a probe, then a leader is created and fb becomes a barrier blocked by that leader. If fb encounters a leader, fb becomes a barrier blocked by that leader. If fb encounters another barrier, they either merge into one barrier

or both disappear. When the merged barrier is still a fb , we can treat it as the original fb that takes a step forward. The length of interaction sequence that fb traverses the ring is n . By Lemma 2.0.1, fb traverses the ring within $O(n^2)$ steps with probability $1 - O(1/n)$. We can conclude that within $O(n^2)$ steps, with probability $1 - O(1/n)$, a free barrier meets a probe or a leader. □

Lemma 4.1.4. *In phase 2, when there is no leader, a leader is created within $O(n^2 \log n)$ with probability $1 - O(1/n)$.*

Proof. In phase 2, there are two different ways to create leaders. First, the encounter between a free barrier and a probe will create a leader. This will occur within $O(n^2)$ steps with probability $1 - O(1/n)$, as proven in Lemma 4.1.3.

Second, when a signal moves to a group with the same ID, a leader will be created. The most time-consuming scenario for this method is when a signal with the greatest ID needs to traverse the whole ring and return to the group that sent the signal. There are three types of time cost of the scenario: traversing the ring, comparisons with groups, and comparisons with other signals. First, the length of interaction sequence for the signal moving from one group to another is $O(\psi^2)$, meaning that each sub-signal moves ψ steps separately. There are exactly n/ψ groups in the ring. Therefore, the length of interaction sequence is $O(\psi^2) \cdot n/\psi = O(n\psi)$. By Lemma 2.0.1, the signal traversing the ring requires $O(n^2\psi)$ steps with probability $1 - O(1/n)$. Second, the cost of comparisons with groups is that the signal compares with all groups in the ring. The length of interaction sequence for a comparison between a signal and a group is $O(\psi)$. After comparing with a group, the signal may need to implement a lottery game. The length of interaction sequence for a lottery game is $O(2^\psi) = O(\log n)$. In the process that the signal traverses the ring, the

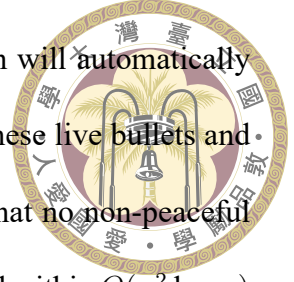
signal compares ID with at most $n\psi$ groups. Therefore, the length of interaction sequence for the signal compares ID with groups in the process of traversing is $(O(\psi) + O(\log n)) \cdot n/\psi = O(n \log n)$. By Lemma 2.0.1, the comparisons with groups require $O(n^2 \log n)$ steps with probability $1 - O(1/n)$. Third, the cost of comparisons with signals is that the signal compares with another signal at each movement from a group to the next group. The length of interaction sequence for a comparison between two signals is $O(\psi^2)$. Because we consider the signal with the greatest ID in the ring, we do not worry about the signal being eliminated. Even if it encounters another signal with the same ID blocking its path, the signal in front of it will still survive and continue to move forward after the comparison is completed. This is equivalent to the original signal moving to the next group. The signal moves from a group to the next group at most n/ψ times. Therefore, the length of interaction sequence for the comparison with signals is $O(\psi^2) \cdot n/\psi = O(n\psi)$. By Lemma 2.0.1, comparisons with other signals require $O(n^2\psi)$ steps with probability $1 - O(1/n)$. In our design, the signal disappears after a comparison with a group with probability $1/n$. But this will at most increase the time required by this scenario by a constant factor. In summary, a leader is created by the second way within $O(n^2\psi) + O(n^2 \log n) + O(n^2\psi) = O(n^2 \log n)$ steps with probability $1 - O(1/n)$.

We can conclude that when there is no leader, a leader is created within $O(n^2 \log n)$ with probability $1 - O(1/n)$. □

Lemma 4.1.5. *From any configuration $C \in C_{clean}$, no non-peaceful bullet will appear after $O(n^2 \log n)$ steps with probability $1 - O(1/n)$.*

Proof. If C is in C_{PB} , the newly generated live bullets are also peaceful. If C is in C_{NoLB} , all live bullets in the future will be newly generated, so they are all peaceful. If C is in

C_{NoBAS} , when a leader is created, all live bullets in the configuration will automatically become peaceful because there is no bullet absence signal between these live bullets and the new leader. Therefore, after a leader is created, we can ensure that no non-peaceful bullet will appear. By Lemma 4.1.4, in phase 2, a leader will be created within $O(n^2 \log n)$ steps with probability $1 - O(1/n)$. We can conclude that after $O(n^2 \log n)$ steps, with probability $1 - O(1/n)$, no non-peaceful bullet will appear. $\square$



We implement the lottery game in each group to increase the waiting time before generating a signal. We use the definition of lottery game and a Lemma in [32].

Definition 3. [32, Definition 3.8] *Let k and l be positive integers. Consider that a player makes independent coin flips repeatedly. The result of each flip is head with probability $1/2$ and tail with probability $1/2$. One round of the lottery game ends each time the player sees a tail or consecutive k heads. If a round ends in the former case (i.e., the round ends with the observation of a tail), the player loses the game in that round; Otherwise, the player wins the game in that round. Define a random variable $W_{LG}(k, l)$ as the number of rounds that the player wins in the first l flips.*

Lemma 4.1.6. [32, LEMMA 3.9] *$\Pr(W_{LG}(k, 4ck \cdot 2^k) \leq 8ck) = 1 - 1/2^{ck}$ holds for any $k \geq 1$ and $c \geq 1$.*

In phase 3, the leaders will prevent the creation of other leaders. This is to ensure that there will not be any concerns about creating a new leader when eliminating redundant leaders, thus ensuring the selection of a unique leader. We first prove that in phase 3, the population will be suppressed within $O(n^2)$ steps with probability $1 - O(1/n)$. Then, we prove that after the population is suppressed, within $O(n^3)$ steps, no signal will be generated with probability $1 - O(1/n)$, which is sufficiently to decrease the number of

leaders to one.

Lemma 4.1.7. *In phase 3, the population will be suppressed within $O(n^2)$ steps with probability $1 - O(1/n)$.*



Proof. In phase 3, the leaders will send reset commands to the population. We call the population is suppressed if all agents have received at least one reset command. We can just wait the reset command traverses the ring then the population is suppressed. By Lemma 2.0.1, after $O(n^2)$ steps, the population will be suppressed with probability $1 - O(1/n)$. $\square$

Lemma 4.1.8. *In phase 4, after the population is suppressed, within $O(n^3)$ steps, no signal will be generated with probability $1 - O(1/n)$.*

Proof. We call the population is suppressed if all agents have received at least one reset command. Reset commands sent by leaders can reset the counters of all groups in $c'n^2$ steps for a constant c' with high probability by Lemma 2.0.1. Therefore, with high probability, the time for a reset group to be reset again is at most $2c'n^2$ steps. In $2c'n^2$ steps, a group has at most $8c'n$ interactions to increase/reset the hit of the group with high probability by Chernoff bound (Lemma A.0.1). A group does not increase its counter to κ in $8c'n$ interactions with probability $1 - O(1/n^3)$ by Lemma 4.1.6 with $k = 2^\psi$ and $c = 3$ for a sufficiently large constant κ . Then by union bound, all groups do not increase their counters to κ in $8c'n$ interactions with probability $1 - O(1/n^2)$. Then by union bound again, all groups do not increase their counters to κ in n times $2c'n^2$ steps with probability $1 - O(1/n)$. Therefore, with probability $1 - O(1/n)$, no signal will be generated in $O(n^3)$ steps, meaning no leaders will be created by the signal. $\square$

We ensure that after the population is suppressed, no leader will be created by the signal over a sufficiently long period. It is necessary to ensure that no leader will be created by the barrier within this period. We say that a free barrier located in the pair of agents u_i and u_{i+1} is safe when the following conditions are satisfied:



1. $d_{RL}(i+1) \neq \infty$.
2. $\forall j = 0, 1, \dots, d_{RL}(i+1) : u_{i+1+j}.\text{signal}_B = 0$.
3. $d_{LL}(i) \neq \infty$.
4. $\forall j = 0, 1, \dots, d_{LL}(i) : u_{i-j}.\text{bullet} \neq 2$.

Simply put, the agents on the right side of the safe barrier up to the first leader will not have bullet absence signals, and the agents on the left side of the safe barrier up to the first leader will not have live bullets. If a free barrier is safe, it will not create leaders with other free barriers.

Lemma 4.1.9. *In phase 4, all free barrier will be safe within $O(n^2)$ steps with probability $1 - O(1/n)$.*

Proof. Assume that there are one or more free barriers in the current configuration. Let fb be any one of these free barriers. In phase 4, there will be one or more leaders in the configuration. Therefore, the first and third conditions are satisfied in phase 4. Also, in phase 4, there is no non-peaceful bullet in the configuration. Therefore, there is no live bullet to the left of fb , so the fourth condition is satisfied.

We can assume that there is only one free barrier between two leaders because if there is more than one free barrier between two leaders, according to Lemma 4.1.3, these

free barriers will send probes to the free barrier on their left and create new leaders within $O(n^2)$ steps with probability $1 - O(1/n)$. This process will stop only when there is at most one free barrier between the two leaders.



As fb moves toward its nearest right leader rl , three types of events will occur. First, fb meets rl and fb is blocked by rl . At this point, fb is no longer a free barrier. Second, before fb meets rl , rl is killed and then a barrier fb' , which was blocked by rl , becomes free. Then fb will receive the probe sent by fb' and create a leader. At this point, fb is no longer a free barrier and there will be no bullet absence signal between fb' and its nearest right leader rl' because rl' has just fired a live bullet to kill rl and cleared the bullet absence signals along the way. Therefore, we can consider that fb has transformed into fb' and satisfies the second condition; then fb' becomes safe.

Third, before fb meets rl , rl is killed but there is no barrier blocked by rl . In this case, there might have been bullet absence signals between fb and rl originally. We use the property where a live bullet, after killing the leader, transforms into a dummy bullet, which clears all the original bullet absence signals. After the dummy bullet clears all original bullet absence signals, there will be no bullet absence signal between fb and its new nearest right leader rl' , then fb becomes safe.

In these three types of events, fb will be safe after it moves to its nearest right leader. A free barrier moves to its nearest right leader within $O(n^2)$ steps with probability $1 - O(1/n^2)$ by Lemma 2.0.1 with $c = 2$. By union bound, all free barrier move to their nearest right leaders within $O(n^2)$ steps with probability $1 - O(1/n)$. We can conclude that in phase 4, all free barriers will be safe within $O(n^2)$ steps with probability $1 - O(1/n)$.

□

Now, we ensure that in phase 4, no leader will be created by both methods of creating leaders within $O(n^3)$ steps with probability $1 - O(1/n)$. Next, we will prove here that in phase 4, the number of leaders will be decreased to one within $O(n^2 \log n)$ steps with probability $1 - O(1/n)$.



Lemma 4.1.10. *In phase 4, there will be only one leader within $O(n^2 \log n)$ steps with probability $1 - O(1/n)$.*

Proof. We use a similar method to [32] to eliminate redundant leaders. The only two differences between our method and theirs are that in our method, bullets and bullet absence signals are blocked by barriers, and after killing a leader, a live bullet will transform into a dummy bullet. We do not need to worry about the creation of a leader during this leader elimination process. In phase 4, the population is suppressed, and we can ensure that no leader will be created by a signal within $O(n^3)$ steps. By Lemma 4.1.9, we can ensure that all free barriers will be safe, so no leader will be created due to the meeting of a barrier and a probe.

Let u_1, u_2 , and u_3 be the leaders in the configuration such that u_3 is the nearest right leader of u_2 , u_2 is the nearest right leader of u_1 . ($u_1 = u_3$ may hold.) Assume that u_2 is a leader blocking a barrier. When u_2 is killed by the live bullet fired by u_3 , the barrier blocked by u_2 becomes a free barrier. u_3 needs to wait this free barrier moves to u_3 , then u_3 can receive a bullet absence signal from u_1 . In addition, u_3 also needs to wait for the live bullet transformed into a dummy bullet to move to u_1 before receiving the bullet absence signal sent by u_1 . Therefore, in our method, the time it takes for a leader to fire a bullet is at most four times that of the original method. Yokota et al. [31] proves that the original method decreases the number of leaders to one within $O(n^2 \log n)$ steps with probability

$1 - O(1/n)$. Therefore, the modified method also decreases the number of leaders to one within $O(n^2 \log n)$ steps with probability $1 - O(1/n)$. Additionally, when there is only one leader left, after $O(n^2)$ steps, there will be no free barriers in the configuration with probability $1 - O(1/n)$. □



In phase 5, the remaining leader must ensure that no other leaders are created thereafter. By Lemma 4.1.10, we can ensure that no leader will be created by barriers. We also need to ensure that no leader is created due to signals. Here, we prove that the remaining leader will make the ring become a stable sequence in $O(n^2 \log n \log \log n)$ steps with probability $1 - O(1/n)$. This ensures that no other leader is created due to signals.

Lemma 4.1.11. *In phase 5, the remaining leader will make the entire ring become a stable ID sequence within $O(n^2 \log n \log \log n)$ steps with probability $1 - O(1/n)$.*

Proof. In phase 5, the process of writing a stable ID sequence will be divided into $\log n$ rounds. In the i -th round, the leader generates a overwriting signal with ID i where $i \in [0, \log n - 1]$. The overwriting signal will traverse the ring and rewrite the IDs of the groups it passes. The time required for this process can be divided into two types: traversing the ring and ID overwriting. First, the length of interaction sequence for overwriting signal moving from one group to another is $O(\psi^2)$. There are exactly n/ψ groups in the ring. Therefore, the length of interaction sequence for traversing the ring is $O(\psi^2) \cdot n/\psi = O(n\psi)$. By Lemma 2.0.1, the overwriting signal traversing the ring requires $O(n^2\psi)$ steps with probability $1 - O(1/n)$. Second, the length of interaction sequence for rewriting the ID of one group is $O(\psi)$, which is the process that check the size relationship of the overwriting signal and the group, and then rewrite the ψ bits in the group. There are exactly n/ψ groups. Therefore, the length of interaction sequence for rewriting the IDs of all

groups is $O(n)$. By Lemma 2.0.1, rewriting the IDs of all groups requires $O(n^2)$ steps with probability $1 - O(1/n)$. In summary, one round will require $O(n^2\psi) + O(n^2) = O(n^2\psi)$ steps. Therefore, the leader writes a stable ID sequence within $O(n^2\psi) \cdot O(\log n) = O(n^2\psi \log n) = O(n^2 \log n \log \log n)$ steps with probability $1 - O(1/n)$.

□

Additionally, we need to ensure that this ID sequence remains unchanged once it is completed. Here, we first define a overwriting signal that can change the stable ID sequence as a free overwriting signal. We then prove that all free overwriting signals will not appear after $O(n^2 \log n \log \log n)$ steps with probability $1 - O(1/n)$.

Let the group sequence $G = g_0, g_1, g_2, \dots, g_{n/\psi-1}$ where g_0 is the group in which the leader is located. We say that a overwriting signal WS with located in a group g_i is free when the following conditions are satisfied:

1. $WS.wb = write$.
2. $i \neq x$ where x is the number of trailing zeros of i .
3. $ID(g_i) > ID(WS)$.

Lemma 4.1.12. *In phase 5, no free overwriting signal will appear after $O(n^2 \log n \log \log n)$ steps with probability $1 - O(1/n)$.*

Proof. Assume there are one or more free overwriting signal in the current configuration. Let fws be any one of these free overwriting signals. We let fws traverse the ring. In this traverse, fws will encounter either a overwriting signal echo or a leader.

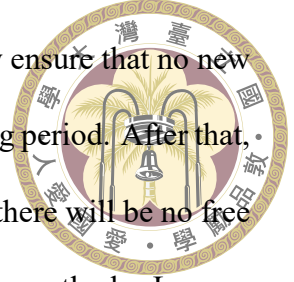
If fws encounters a overwriting signal echo, then it will be eliminated. If fws encounters a leader, then $ID(fws)$ will be increased by one. Until the $ID(fws)$ overflows, at which point fws is no longer a free overwriting signal because, at this moment, fws is equivalent to a new overwriting signal generated by the leader. And all overwriting signals generated by the leader are not free overwriting signals. The length of interaction sequence for traversing the ring is $O(\psi^2) \cdot n/\psi = O(n\psi)$. By Lemma 2.0.1 with $c = 2$, fws traverses the ring within $O(n^2\psi)$ steps with probability $1 - O(1/n^2)$. The $ID(fws)$ overflow requires at most $\log n$ increments. Therefore, fws becomes a non-free overwriting signal within $O(n^2\psi) \cdot \log n = O(n^2 \log n \log \log n)$ steps with probability $1 - O(1/n^2)$. By the union bound, all free overwriting signals become non-free overwriting signals within $O(n^2 \log n \log \log n)$ with probability $1 - O(1/n)$. We can conclude that within $O(n^2 \log n \log \log n)$ steps, no free overwriting signal will appear after $O(n^2 \log n \log \log n)$ steps with probability $1 - O(1/n)$.

□

Theorem 4.1.13. *If knowledge $\psi = \lceil \log \log n \rceil + O(1)$ is given, P_{PLL} is a self-stabilizing leader election protocol for any directed ring such that (i) for any configuration C_0 , $\Xi_{P_{PLL}}(C_0, \Gamma)$ reaches a safe configuration within $O(n^2 \log n \log \log n)$ steps with probability $1 - O(1/n)$, and (ii) each agent uses $\text{polyloglog}(n)$ states.*

Proof. By Lemma 4.1.2, we can ensure that after $O(n^2)$, the population enters a clean configuration with probability $1 - O(1/n)$, then, by Lemma 4.1.4, at least one leader will be created within $O(n^2 \log n)$ steps with probability $1 - O(1/n)$. At this moment, by Lemma 4.1.5, we can ensure that the number of leaders will not be decrease to zero. Next, by Lemma 4.1.8 and 4.1.9, we can ensure that within $O(n^3)$ steps, no leader will

be created with probability $1 - O(1/n)$. But at this time, we can only ensure that no new leaders is created with probability $1 - O(1/n)$ within a sufficiently long period. After that, by Lemma 4.1.10, the number of leaders will be decrease to one and there will be no free barriers after $O(n^2 \log n)$ steps with probability $1 - O(1/n)$. Subsequently, by Lemma 4.1.11, the remaining leader will stabilize the population within $O(n^2 \log n \log \log n)$ steps with probability $1 - O(1/n)$. Once the population has stabilized, we can ensure that no new leader will ever be created thereafter. At this point, we can ensure that (i) there is only one leader (ii) there is no non-peaceful bullet (iii) there is no free barrier (iv) the ring is a stable ID sequence that starts from the leader, meaning that the population is in safe configuration. In summary, for any configuration C_0 , $\Xi_{P_{LL}}(C_0, \Gamma)$ reaches a safe configuration within $O(n^2 \log n \log \log n)$ steps with probability $1 - O(1/n)$. $\square$







Chapter 5 Conclusion

In this paper, we propose an SSLE population protocol on directed rings. Compared to related works with similar assumptions (knowledge of n), our protocol has fast convergence speed and significantly reduces the number of states required. Further reducing the required number of states while maintaining acceptable convergence time under the same assumptions is an direction of future research. Designing an SSLE protocol with a constant number of states and efficient convergence time, or providing a lower bound, remains an interesting open problem.

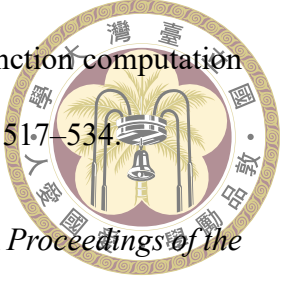




References

- [1] ALISTARH, D., ASPNES, J., EISENSTAT, D., GELASHVILI, R., AND RIVEST, R. L. Time-space trade-offs in population protocols. In *Proceedings of the twenty-eighth annual ACM-SIAM symposium on discrete algorithms* (2017), SIAM, pp. 2560–2579.
- [2] ALISTARH, D., ASPNES, J., AND GELASHVILI, R. Space-optimal majority in population protocols. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms* (2018), SIAM, pp. 2221–2239.
- [3] ALISTARH, D., AND GELASHVILI, R. Polylogarithmic-time leader election in population protocols. In *Automata, Languages, and Programming: 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II 42* (2015), Springer, pp. 479–491.
- [4] ALISTARH, D., GELASHVILI, R., AND VOJNOVIĆ, M. Fast and exact majority in population protocols. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing* (2015), pp. 47–56.
- [5] ANGLUIN, D., ASPNES, J., DIAMADI, Z., FISCHER, M. J., AND PERALTA, R. Computation in networks of passively mobile finite-state sensors. In *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing* (2004), pp. 290–299.

- 
- [6] ANGLUIN, D., ASPNES, J., AND EISENSTAT, D. Fast computation by population protocols with a leader. *Distributed Computing* 21 (2008), 183–199.
- [7] ANGLUIN, D., ASPNES, J., FISCHER, M. J., AND JIANG, H. Self-stabilizing population protocols. *ACM Trans. Auton. Adapt. Syst.* 3, 4 (dec 2008).
- [8] ASPNES, J., BEAUQUIER, J., BURMAN, J., AND SOHIER, D. Time and space optimal counting in population protocols. *arXiv preprint arXiv:1611.07238* (2016).
- [9] BEAUQUIER, J., BLANCHARD, P., AND BURMAN, J. Self-stabilizing leader election in population protocols over arbitrary communication graphs. In *International Conference on Principles of Distributed Systems* (2013), Springer, pp. 38–52.
- [10] BEAUQUIER, J., BURMAN, J., CLAVIERE, S., AND SOHIER, D. Space-optimal counting in population protocols. In *International Symposium on Distributed Computing* (2015), Springer, pp. 631–646.
- [11] BERENBRINK, P., GIAKKOUPIS, G., AND KLING, P. Optimal time and space leader election in population protocols. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing* (2020), pp. 119–129.
- [12] BURMAN, J., CHEN, H.-L., CHEN, H.-P., DOTY, D., NOWAK, T., SEVERSON, E., AND XU, C. Time-optimal self-stabilizing leader election in population protocols. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing* (2021), pp. 33–44.
- [13] CAI, S., IZUMI, T., AND WADA, K. 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 (2012), 433–445.

- 
- [14] CHEN, H.-L., DOTY, D., AND SOLOVEICHIK, D. Deterministic function computation with chemical reaction networks. *Natural computing* 13 (2014), 517–534.
- [15] CHEN, H.-P., AND CHEN, H.-L. Self-stabilizing leader election. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing* (2019), pp. 53–59.
- [16] CHEN, H.-P., AND CHEN, H.-L. Self-stabilizing leader election in regular graphs. In *Proceedings of the 39th Symposium on Principles of Distributed Computing* (2020), pp. 210–217.
- [17] DIJKSTRA, E. W. Self-stabilizing systems in spite of distributed control. *Communications of the ACM* 17, 11 (1974), 643–644.
- [18] DOTY, D., AND SOLOVEICHIK, D. Stable leader election in population protocols requires linear time. *Distributed Computing* 31, 4 (2018), 257–271.
- [19] FISCHER, M., AND JIANG, H. Self-stabilizing leader election in networks of finite-state anonymous agents. In *Principles of Distributed Systems* (Berlin, Heidelberg, 2006), M. M. A. A. Shvartsman, Ed., Springer Berlin Heidelberg, pp. 395–409.
- [20] IZUMI, T. On space and time complexity of loosely-stabilizing leader election. In *International Colloquium on Structural Information and Communication Complexity* (2015), Springer, pp. 299–312.
- [21] MITZENMACHER, M., AND UPFAL, E. *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press, 2017.
- [22] SOLOVEICHIK, D., COOK, M., WINFREE, E., AND BRUCK, J. Computation with finite stochastic chemical reaction networks. *natural computing* 7 (2008), 615–633.

- [23] SUDO, Y., EGUCHI, R., IZUMI, T., AND MASUZAWA, T. Time-optimal loosely-stabilizing leader election in population protocols. *arXiv preprint arXiv:2005.09944*. (2020).



- [24] SUDO, Y., MASUZAWA, T., DATTA, A. K., AND LARMORE, L. L. The same speed timer in population protocols. In *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)* (2016), IEEE, pp. 252–261.
- [25] SUDO, Y., NAKAMURA, J., YAMAUCHI, Y., OOSHITA, F., KAKUGAWA, H., AND MASUZAWA, T. Loosely-stabilizing leader election in a population protocol model. *Theoretical Computer Science* 444 (2012), 100–112.
- [26] SUDO, Y., OOSHITA, F., KAKUGAWA, H., AND MASUZAWA, T. Loosely-stabilizing leader election on arbitrary graphs in population protocols. In *Principles of Distributed Systems: 18th International Conference, OPODIS 2014, Cortina d'Ampezzo, Italy, December 16-19, 2014. Proceedings 18* (2014), Springer, pp. 339–354.
- [27] SUDO, Y., OOSHITA, F., KAKUGAWA, H., AND MASUZAWA, T. Loosely stabilizing leader election on arbitrary graphs in population protocols without identifiers or random numbers. *IEICE TRANSACTIONS on Information and Systems* 103, 3 (2020), 489–499.
- [28] SUDO, Y., OOSHITA, F., KAKUGAWA, H., MASUZAWA, T., DATTA, A. K., AND LARMORE, L. L. Loosely-stabilizing leader election for arbitrary graphs in population protocol model. *IEEE Transactions on Parallel and Distributed Systems* 30, 6 (2018), 1359–1373.

- 
- [29] SUDO, Y., OOSHITA, F., KAKUGAWA, H., MASUZAWA, T., DATTA, A. K., AND LARMORE, L. L. Loosely-stabilizing leader election with polylogarithmic convergence time. *Theoretical Computer Science* 806 (2020), 617–631.
- [30] SUDO, Y., SHIBATA, M., NAKAMURA, J., KIM, Y., AND MASUZAWA, T. Self-stabilizing population protocols with global knowledge. *IEEE Transactions on Parallel and Distributed Systems* 32, 12 (2021), 3011–3023.
- [31] YOKOTA, D., SUDO, Y., AND MASUZAWA, T. Time-optimal self-stabilizing leader election on rings in population protocols. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* E104.A, 12 (Dec. 2021), 1675–1684.
- [32] YOKOTA, D., SUDO, Y., OOSHITA, F., AND MASUZAWA, T. A near time-optimal population protocol for self-stabilizing leader election on rings with a poly-logarithmic number of states. In *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing* (New York, NY, USA, 2023), PODC '23, Association for Computing Machinery, p. 2–12.





Appendix A — CHERNOFF BOUNDS

Lemma A.0.1. [21, Theorems 4.4, 4.5] Let $X_1, \dots, X_s$ be independent Poisson trials, and let $X = \sum_{i=1}^s X_i$. Then

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

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