Time-Optimal Self-Stabilizing Leader Election in Population Protocols

Janna Burman
janna.burman@lri.fr
Université Paris-Saclay, CNRS

Ho-Lin Chen
holinchen@ntu.edu.tw
National Taiwan University

Hsueh-Ping Chen
r07921034@ntu.edu.tw
National Taiwan University

David Doty
doty@ucdavis.edu
University of California, Davis

Thomas Nowak
thomas.nowak@lri.fr
Université Paris-Saclay, CNRS

Eric Severson
eseverson@ucdavis.edu
University of California, Davis

Chuan Xu
chuan.xu@inria.fr
Inria Sophia-Antipolis

ABSTRACT

We consider the standard population protocol model, where (*a priori*) indistinguishable and anonymous agents interact in pairs according to uniformly random scheduling. The *self-stabilizing leader election* problem requires the protocol to converge on a single leader agent from *any* possible initial configuration. We initiate the study of time complexity of population protocols solving this problem in its original setting: with probability 1, in a complete communication graph. The only previously known protocol by Cai, Izumi, and Wada [Theor. Comput. Syst. 50] runs in expected parallel time $\Theta(n^2)$ and has the optimal number of n states in a population of n agents. The existing protocol has the additional property that it becomes silent, i.e., the agents' states eventually stop changing.

Observing that any silent protocol solving self-stabilizing leader election requires $\Omega(n)$ expected parallel time, we introduce a silent protocol that uses optimal $O(n)$ parallel time and states. Without any silence constraints, we show that it is possible to solve self-stabilizing leader election in asymptotically optimal expected parallel time of $O(\log n)$, but using at least exponential states (a quasipolynomial number of bits). All of our protocols (and also that of Cai et al.) work by solving the more difficult *ranking* problem: assigning agents the ranks $1, \ldots, n$.

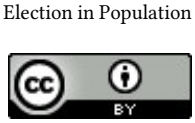
CCS CONCEPTS

• **Theory of computation → Distributed algorithms**.

KEYWORDS

leader election; self-stabilization; population protocols

ACM Reference Format:
Janna Burman, Ho-Lin Chen, Hsueh-Ping Chen, David Doty, Thomas Nowak, Eric Severson, and Chuan Xu. 2021. Time-Optimal Self-Stabilizing Leader Election in Population Protocols. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing (PODC '21), July 26–30, 2021, Virtual Event, Italy.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3465084.3467898



PODC '21, July 26–30, 2021, Virtual Event, Italy.

ACM ISBN 978-1-4503-8548-0/21/07.
https://doi.org/10.1145/3465084.3467898

ACKNOWLEDGMENTS

We warmly thank anonymous reviewers for their detailed comments. Doty and Severson were supported by NSF award 1900931 and CAREER award 1844976. Ho-Lin and Hsueh-Ping were supported by MOST (Taiwan) grant number 107-2221-E-002-031-MY3. Nowak was supported by the CNRS project ABIDE.

1 INTRODUCTION

Population protocols [8] are a popular and well established model of distributed computing, originally motivated by passively mobile sensor networks. However, it also models population dynamics from various areas such as trust and rumor propagation in social networks [28], game theory dynamics [16], chemical reactions [38, 53], and gene regulatory networks [18]. Population protocols are a special-case variant of Petri nets and vector addition systems [34].

This model considers computational *agents* with no ability to control their schedule of communication. They are *a priori* anonymous, indistinguishable, and mobile: interacting in pairs asynchronously and unpredictably. At each step a pair of agents to interact is chosen uniformly at random. Each agent observes the other's state, updating its own according to the transition function. A *configuration* describes the global system state: the state of each of the n agents. The sequence of visited configurations describes a particular *execution* of the protocol. The goal of the protocol is to reach a desired set of configurations with probability 1.

It is common in population protocols to measure space/memory complexity by counting the potential number of states each agent can have.[1] The model originally used constant-state protocols, i.e., the state set is independent of the population size n [8]. Recent studies relax this assumption and allow the number of states to depend on n, adding computational power to the model [17, 39, 45], improving time complexity [2, 37, 51], or tolerating faults [22, 39, 44]. In the current work, for tolerating any number of transient

[1]The base-2 logarithm of this quantity is the standard space complexity: the number of bits required to represent each state (e.g., polynomial states = logarithmic space).

faults (in the framework of self-stabilization), such relaxation is necessary [22, 57] (see details below and Theorem 2.1).

Leader election. In the *leader election* problem, the protocol should reach a configuration C with only one agent marked as a "leader", where all configurations reachable from C also have a single leader. When this happens, the protocol's execution is said to have *stabilized*.[2] The time complexity of a protocol is measured by *parallel time*, the number of interactions until stabilization, divided by the number of agents n.[3]

Leader election is an important paradigm in the design of distributed algorithms useful to achieve a well coordinated and efficient behavior in the network. For example, in the context of population protocols, given a leader, protocols can become exponentially faster [9, 13] or compact (using less memory states) [15]. Moreover, some problems, like fault-tolerant counting, naming and bipartition become feasible, assuming a leader [12, 20, 59].

Leader election protocols have been extensively studied in the original setting where all agents start in the same pre-determined state (a non-self-stabilizing case, and in complete interaction graphs, i.e. where any pair of agents can interact). For example, it was shown that the problem cannot be solved in $o(n)$ (parallel) time if agents have only $O(1)$ states [32], an upper bound later improved to $\leq \frac{1}{2}\log\log n$ states [1]. To circumvent this impossibility result, subsequent studies assume a non-constant state space, though relatively small (e.g., $O(\log n)$ or $O(\log\log n)$). Leader election has recently been shown to be solvable with optimal $O(\log n)$ parallel time and $O(\log\log n)$ states [14], improving on recent work meeting this space bound in time $O(\log^2 n)$ [36] and $O(\log n \log\log n)$ [37]. Another work presents a simpler $O(\log n)$-time, $O(\log n)$-state protocol [55]. It may appear obvious that any leader election protocol requires $\Omega(\log n)$ time, but this requires a nontrivial proof [54]. There is also an $O(1)$-space and expected $O(\log^2 n)$-time protocol, but with a positive error probability; and a slower $o(n)$-time (e.g., $\sqrt{n}$) protocol correct with probability 1 [43]. Recent surveys [6, 33] explain the relevant techniques.

Reliable leader election. The current paper studies leader election in the context of *reliability*. What if agents are prone to memory or communication errors? What if errors cannot be directly detected, so agents cannot be re-initialized in response? One can imagine mobile sensor networks for mission critical and safety relevant applications where rapid recovery from faults takes precedence over memory requirements. Imagine applications operating on relatively small sized networks, so that the sensors' memory storage dependent on n is not necessarily an issue. Additionally, n states are provably required to solve our problem [22] (see Theorem 2.1).

We adopt the approach of self-stabilization [10, 29]. A protocol is called *self-stabilizing* if it stabilizes with probability 1 from an *arbitrary* configuration[4] (resulting from any number of transient faults). Non-self-stabilizing (a.k.a., *initialized*) leader election is easily solvable using only one bit of memory per agent by the single transition $(\ell, \ell) \to (\ell, f)$ from an initial configuration of all ℓ's: when two candidate leaders meet, one becomes a follower f. However, this protocol fails (as do nearly all other published leader election protocols) in the self-stabilizing setting from an all-f configuration. Thus, any self-stabilizing leader election (SSLE) protocol must be able not only to reduce multiple potential leaders to one, but also to create new leaders. A challenge is a careful verification of leader absence, to avoid creating excess leaders forever.

Because of this challenge, in any SSLE protocol, agents must know the *exact* population size n, and the number of states must be at least n [22] (Theorem 2.1 in the preliminaries section). Despite the original assumption of constant space, population protocols with linear space (merely $O(\log n)$ bits of memory) may be useful in practice, similarly to distributed algorithms in other models (message passing, radio networks, etc.). One may now imagine such memory-equipped devices communicating in a way as agents do in population protocols [42, 50]. Think of a group of mobile devices (like sensors, drones or smart phones) operating in different types of rescue, military or other monitoring operations (of traffic, pollution, agriculture, wild-life, etc.). Such networks may be expected to operate in harsh inaccessible environments, while being highly reliable and efficient. This requires an efficient "strong" fault-tolerance for automatic recovery provided by self-stabilization. Moreover, even if one considers only protocols with polylog(n) states interesting, it remains an interesting fact that such protocols cannot solve SSLE.

Finally, self-stabilizing algorithms are easier to compose [30, 31]. Composition is in general difficult for population protocols [24, 52], since they lack a mechanism to detect when one computation has finished before beginning another. However, a self-stabilizing protocol S can be composed with a prior computation P, which may have set the states of S in some unknown way before P stabilized, c.f. [10, Section 4], [7, Theorem 3.5].

Problem variants. To circumvent the necessary dependence on population size n, previous work has considered relaxations of the original problem. One approach, which requires agents only to know an upper bound on n, is to relax the requirement of self-stabilization: *loose-stabilization* requires only that a unique leader persists for a long time after a stabilization, but not forever [56]. Other papers study leader election in more general and powerful models than population protocols, which allow extra computational ability not subject to the limitations of the standard model. One such model assumes an external entity, called an *oracle*, giving clues to agents about the existence of leaders [11, 35]. Other generalized models include *mediated population protocols* [47], allowing additional shared memory for every pair of agents, and the k-interaction model [58], where agents interact in groups of size 2 to k.

While this paper considers only the complete graph (the most difficult case), other work considers protocols that assume a particular non-complete graph topology. In rings and regular graphs with constant degree, SSLE is feasible even with only a constant state space [10, 25, 26, 60]. In another recent related work [57], the authors study the feasibility requirements of SSLE in arbitrary graphs, as well as the problem of *ranking* that we also study (see below). They show how to adapt protocols in [11, 22] into protocols for an arbitrary (and unknown) connected graph topology (without any oracles, but knowing n).

[2] Some protocols [36, 43] stabilize with probability 1, but converge (elect a unique leader) long before stabilizing (become unable to change the number of leaders). In our protocols these two events typically coincide.

[3] This captures the intuition that interactions happen in parallel, defining the time scale so that each agent participates in $O(1)$ interactions per time unit on average.

[4] For a self-stabilizing protocol, it is equivalent to consider probability 1 and fixed probability $p > 0$ of correctness; See Section 2.

1.1 Contribution

We initiate the study of the limits of time efficiency or the time/space trade-offs for SSLE in the standard population protocol model, in the complete interaction graph. The most related protocol, of Cai, Izumi, and Wada [22] (Silent-n-state-SSR, Protocol 1), given for complete graphs, uses exactly n states and $\Theta(n^2)$ expected parallel time , exponentially slower than the polylog(n)-time non-self-stabilizing existing solutions [14, 36, 37, 43, 55]. Our main results are two faster protocols, each making a different time/space tradeoff.

Our protocols, along with that of [22], are summarized in Table 1. These main results are later proven as Theorem 4.1 and Theorem 5.1. Both expected time and high-probability time are analyzed. Any *silent* protocol (one guaranteed to reach a configuration where no agent subsequently changes states) must use $\Omega(n)$ parallel time in expectation (Observation 2.2). This lower bound has helped to guide our search for sublinear-time protocols, since it rules out ideas that, if they worked, would be silent. Thus Optimal-Silent-SSR is time- and space-optimal for the class of silent protocols.

Sublinear-Time-SSR is actually a family of sublinear time protocols that, depending on a parameter H that can be set to an integer between 1 and $\Theta(\log n)$, causes the algorithm's running time to lie somewhere in $O(\sqrt{n})$ and $O(\log n)$, while using more states the larger H is; setting $H = \Theta(\log n)$ gives the time-optimal $O(\log n)$ time protocol. However, even with $H = 1$, it requires exponential states. It remains open to find a sublinear-time SSLE protocol that uses sub-exponential states. We note that any protocol solving SSLE requires $\Omega(\log n)$ time: from any configuration where all n agents are leaders, by a coupon collector argument, it takes $\Omega(\log n)$ time for $n-1$ of them to interact and become followers. (This argument uses the self-stabilizing assumption that "all-leaders" is a valid initial configuration; otherwise, for *initialized* leader election, it requires considerably more care to prove an $\Omega(\log n)$ time lower bound [55].)

For some intuition behind the parameterized running times for Sublinear-Time-SSR, the protocol works by detecting "name collisions" between agents, communicated via paths of length $H+1$. For example, $H = 0$ corresponds to the simple linear-time algorithm that relies on two agents s, a with the same name directly interacting, i.e., the path $s \to a$. $H = 1$ means that s first interacts with a third agent b, who then interacts with a, i.e., the path $s \to b \to a$. To analyze the time for this process to detect a name collision, consider the following "bounded epidemic" protocol. The "source" agent s that starts the epidemic is in state 0, and all others are in state ∞, and they interact by $i, j \to i, i+1$ whenever $i < j$. The time τ_k is the first time some target agent a has state $\leq k$. In other words, this agent has heard the epidemic via a path from the source of length at most k. We have $\mathbb{E}[\tau_1] = O(n)$, since a must meet s directly. An iterative process can then show $\mathbb{E}[\tau_2] = O(\sqrt{n})$, and more generally $\mathbb{E}[\tau_k] = O(kn^{1/k})$. τ_n is the hitting time for the standard epidemic process, since the path from any agent to the source can be at most n. However, with high probability, the epidemic process will reach each agent via a path of length $O(\log n)$, so it follows that $\tau_k = O(\log n)$ if $k = \Omega(\log n)$, so setting $H = \Theta(\log n)$ will detect this name collisions in $O(\log n)$ time.

All protocols in the table solve a more difficult problem than leader election: *ranking* the agents by assigning them the IDs $1, \ldots, n$. Ranking is helpful for SSLE because it gives a deterministic way to detect the absence of a state (such as the leader state). If any rank is absent, the pigeonhole principle ensures multiple agents have the same rank, reducing the task of absence detection to that of collision detection.

Collision detection is accomplished easily in $O(n)$ time by waiting for the colliding agents to meet, which is done by Optimal-Silent-SSR. Achieving stable collision detection in optimal $O(\log n)$ time is key to our fast protocol Sublinear-Time-SSR. This collision detection problem is interesting in its own right, see Conclusion.

Ranking is similar to the *naming* problem of assigning each agent a unique "name" (ID) [20, 46], but is strictly stronger since each agent furthermore knows the order of its name relative to those of other agents. Naming is related to leader election: if each agent can determine whether its name is "smallest" in the population, then the unique agent with the smallest name becomes the leader. However, it may not be straightforward to determine whether some agent exists with a smaller name; much of the logic in the faster ranking algorithm Sublinear-Time-SSR is devoted to propagating the set of names of other agents while determining whether the adversary has planted "ghost" names in this set that do not actually belong to any agent. On the other hand, any ranking algorithm automatically solves both the naming and leader election problems: ranks are unique names, and the agent with rank 1 can be assigned as the leader. (The converse does not hold [21].)

The full version of this paper [21] contains proofs of all results.

2 PRELIMINARIES

We write $\mathbb{N} = \{1, 2, \ldots\}$ and $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$. The term $\ln k$ denotes the natural logarithm of k. $H_k = \sum_{i=1}^{k} \frac{1}{i}$ denotes the kth harmonic number, with $H_k \sim \ln k$, where $f(k) \sim g(k)$ denotes that $\lim_{k\to\infty} \frac{f(k)}{g(k)} = 1$. We omit floors or ceilings (which are asymptotically negligible) when writing $\ln n$ to describe a quantity that should be integer-valued. Throughout this paper, by convention n denotes the *population size* n, the number of agents. We say an event E happens *with high probability (WHP)* if $\mathbb{P}[\neg E] = O(1/n)$.

If a self-stabilizing protocol stabilizes with high probability, then we can make this high probability bound $1 - O(1/n^c)$ for any constant c. This is because in the low probability of an error, we can repeat the argument, using the current configuration as the initial configuration. Each of these potential repetitions gives a new "epoch", where the Markovian property of the model ensures the events of stabilizing in each epoch are independent. Thus the protocol will stabilize after at most c of these "epochs" with probability $1 - O(1/n^c)$. By the same argument, if a self-stabilizing protocol can stabilize with any positive probability $p > 0$, it will eventually stabilize with probability 1.

Model. We consider population protocols [8] defined on a collection $\mathcal{A}$ of n indistinguishable agents, also called a population. We assume a complete communication graph over $\mathcal{A}$, meaning that every pair of agents can interact. Each agent has a set $\mathbf{S}$ of local states. At each discrete step of a protocol, a *probabilistic scheduler* picks randomly an ordered pair of agents from $\mathcal{A}$ to interact. During an interaction, the two agents mutually observe their states

Table 1: Overview of time and space (number of states) complexities of self-stabilizing leader election protocols (which all also solve ranking). For the silent protocols, the silence time also obeys the stated upper bound. Times are measured as parallel time until stabilization both in expectation and with high probability (WHP is defined as probability $1 - O(1/n)$, but implies a guarantee for any $1 - O(1/n^c)$, see Section 2). Entries marked with * are asymptotically optimal in their class (silent/non-silent); see Observation 2.2. The final two rows really describe the same protocol Sublinear-Time-SSR; it is parameterized by the positive integer H; setting $H = \Theta(\log n)$ gives the time-optimal $O(\log n)$ time protocol.

protocol	expected time	WHP time	states	silent
Silent-n-state-SSR [22]	$\Theta(n^2)$	$\Theta(n^2)$	$^* n$	yes
Optimal-Silent-SSR (Sec. 4)	$^* \Theta(n)$	$^* \Theta(n \log n)$	$^* O(n)$	yes
Sublinear-Time-SSR (Sec. 5)	$^* \Theta(\log n)$	$^* \Theta(\log n)$	$\exp\left(O(n^{\log n} \cdot \log n)\right)$	no
Sublinear-Time-SSR (Sec. 5)	$\Theta(H \cdot n^{\frac{1}{H+1}})$	$\Theta(\log n \cdot n^{\frac{1}{H+1}})$	$\Theta(n^{\Theta(n^H)} \log n)$	no

and update them according to a probabilistic[5] transition function $\mathbf{T} : \mathbf{S} \times \mathbf{S} \to \text{Dist}(\mathbf{S} \times \mathbf{S})$ where $\text{Dist}(X)$ denotes the set of probability distributions on X.

Given a finite population $\mathcal{A}$ and state set $\mathbf{S}$, we define a *configuration* C as a mapping $C : \mathcal{A} \to \mathbf{S}$. Given a starting configuration C_0, we define the corresponding *execution* as a sequence $(C_t)_{t \geq 0}$ of random configurations where each C_{t+1} is obtained from C_t by applying $\mathbf{T}$ on the states of a uniform random ordered pair of agents (a, b), i.e., $C_{t+1}(a), C_{t+1}(b) = \mathbf{T}(C_t(a), C_t(b))$ and $C_{t+1}(x) = C_t(x)$ for all $x \in \mathcal{A} \setminus \{a, b\}$. We use the word *time* to mean the number of interactions divided by n (the number of agents), a.k.a. *parallel time*.

Pseudocode conventions. We describe states of agents by several *fields*, using fixed-width font to refer to a field such as `field`. As a convention, we denote by $a.\texttt{field}(t)$, when used outside of pseudocode, the value of `field` in agent a at the end of the t^{th} interaction, omitting "$a.$" and/or "(t)" when the agent and/or interaction is clear from context. Constant values are displayed in a sans serif front such as Yes/No. When two agents a and b interact, we describe the update of each of them using pseudocode, where we refer to `field` of agent $i \in \{a, b\}$ as $i.\texttt{field}$.

In each interaction, one agent is randomly chosen by the scheduler to be the "initiator" and the other the "responder". Most interactions are symmetric, so we do not explicitly label the initiator and responder unless an asymmetric interaction is required.[6]

A special type of field is called a *role*, used in some of our protocols to optimize space usage and limit the types of states accessible to an adversarial initial condition. If an agent has several fields each from a certain set, then that agent's potential set of states is the cross product of all the sets for each field, i.e., adding a field from a set of size k multiplies the number of states by k. A role is used to *partition* the state space: different roles correspond to different sets of fields, so switching roles amounts to deleting the fields from the previous role. Thus the total number of states is obtained by *adding* the number of states in each role.

Convergence and stabilization. Population protocols have some problem-dependent notion of "correct" configurations. (For example, a configuration with a single leader is "correct" for leader election.) A configuration C is *stably correct* if every configuration reachable from C is correct. An execution $\mathcal{E} = (C_0, C_1, \ldots)$ is picked at random according to the scheduler explained above. We say $\mathcal{E}$ *converges* (respectively, *stabilizes)* at interaction $i \in \mathbb{N}$ if C_{i-1} is not correct (resp., stably correct) and for all $j \geq i$, C_j is correct (resp., stably correct). The *(parallel) convergence/stabilization time* of a protocol is defined as the number of iterations to converge/stabilize, divided by n. Convergence can happen strictly before stabilization, although a protocol with a bounded number of states converges from a configuration C with probability $p \in [0, 1]$ if and only if it stabilizes from C with probability p. For a computational task T equipped with some definition of "correct", we say that a protocol *stably computes T with probability p* if, with probability p, it stabilizes (equivalently, converges).

Leader election and ranking. The two tasks we study in this paper are self-stabilizing *leader election* (SSLE) and *ranking* (SSR). For both, the self-stabilizing requirement states that from any configuration, a stably correct configuration must be reached with probability 1. For leader election, each agent has a field `leader` with potential values {Yes, No}, and a *correct* configuration is defined where exactly one agent a has $a.\texttt{leader} = \text{Yes}$.[7] For ranking, each agent has a field `rank` with potential values $\{1, \ldots, n\}$, and a *correct* configuration is defined as one where, for each $r \in \{1, \ldots, n\}$, exactly one agent a has $a.\texttt{rank} = r$. As noted in Sec. 1, any protocol solving SSR also solves SSLE by assigning `leader` to Yes if and only if `rank` = 1; for brevity we omit the `leader` bit from our protocols and focus solely on the ranking problem.

SSLE Protocol from [22]. Protocol 1 shows the original SSLE protocol from [22]. We display it here to introduce our pseudocode style and make it clear that this protocol is also solving ranking.[8]

The convergence proofs in [22] did not consider our definition of parallel time via the uniform random scheduler. Thus we also include proofs that Silent-n-state-SSR stabilizes in $\Theta(n^2)$ time, in

[5]Note that we allow randomness in the transitions for ease of presentation. All our protocols can be made deterministic by standard synthetic coin techniques without changing time or space bounds.

[6]It is also possible to make all transitions symmetric using standard "synthetic coin" techniques.

[7]We do not stipulate the stricter requirement that one agent stays the leader, rather than letting the `leader` = Yes bit swap among agents, but we claim these problems are equivalent due to the complete communication graph. A protocol solving SSLE can also "immobilize" the unique `leader` = Yes bit by replacing any transition $(x, y) \to (w, z)$, where $x.\texttt{leader} = z.\texttt{leader} = \text{Yes}$ and $y.\texttt{leader} = w.\texttt{leader} = \text{No}$, with $(x, y) \to (z, w)$.

[8]Their state set $\{0, \ldots, n-1\}$ from [22] is clearly equivalent to our formal definition of a $\texttt{rank} \in \{1, \ldots, n\}$, but simplifies the modular arithmetic.

expectation and WHP [21] . It is straightforward to argue an $\Omega(n^2)$ time lower bound from a configuration with 2 agents at rank = 0, 0 agents at rank = $n-1$, and 1 agent at every other rank. This requires $n-1$ consecutive "bottleneck" transitions, each moving an agent up by one rank starting at 0. Each takes expected time $\Theta(n)$ since two specific agents (the two with the same rank) must interact directly. Our arguments for a $O(n^2)$ time upper bound give a separate proof of correctness from that in [22], reasoning about a barrier rank that is never crossed.

Protocol 1 Silent-n-state-SSR, for initiator a interacting with responder b
Fields: rank $\in \{0, \ldots, n-1\}$

1: **if** a.rank = b.rank **then**
2: $\quad b$.rank $\leftarrow (b$.rank + 1) mod n

Cai, Izumi, and Wada [22] show that the state complexity of this protocol is optimal. A protocol is *strongly nonuniform* if, for any $n_1 < n_2$, a different set of transitions is used for populations of size n_1 and those of size n_2 (intuitively, the agents hardcode the exact value n).

Theorem 2.1 ([22]). *Any population protocol solving SSLE has $\geq n$ states and is strongly nonuniform.*

It is worth seeing why any SSLE protocol must be strongly nonuniform. Suppose the same transitions are used in population sizes $n_1 < n_2$. By identifying in a single-leader population of size n_2 any subpopulation of size n_1 that does not contain the leader, sufficiently many interactions strictly within the subpopulation must eventually produce a second leader. Thus the full population cannot be stable. These conflicting requirements to both produce a new leader from a leaderless configuration, but also make sure the single-leader configuration is stable, is the key new challenge of leader election in the self-stabilizing setting. Protocols solving SSLE circumvent this error by using knowledge of the exact population size n.

Silent protocols. A configuration C is *silent* if no transition is applicable to it (put another way, every pair of states present in C has only a null transition that does not alter the configuration). A self-stabilizing protocol is *silent* if, with probability 1, it reaches a silent configuration from every configuration. Since convergence time $\leq$ stabilization time $\leq$ silence time, the following bound applies to all three.

Observation 2.2. *Any silent SSLE protocol has $\Omega(n)$ expected convergence time and for any $\alpha > 0$, probability $\geq \frac{1}{2}n^{-3\alpha}$ to require $\geq \alpha n \ln n$ convergence time.*

For example, letting $\alpha = 1/3$, with probability $\geq \frac{1}{2n}$ the protocol requires $\geq \frac{1}{3}n \ln n$ time.

Proof. Let C be a silent configuration with a single agent in a leader state ℓ. Let C' be the configuration obtained by picking an arbitrary non-leader agent in C and setting its state also to ℓ. Since C is silent and the states in C' are a subset of those in C, no state in C' other than ℓ can interact nontrivially with ℓ. So the two ℓ's in C' must interact to reduce the count of ℓ. The number of interactions for this to happen is geometric with $\mathbb{P}[\text{success}] = 1/\binom{n}{2} = \frac{2}{n(n-1)} < 3/n^2$, so expected time $\geq n/3$ and for any $\alpha > 0$, at least $\alpha n^2 \ln n$ interactions ($\alpha n \ln n$ time) are required with probability at least

$$\left(1 - 3/n^2\right)^{\alpha n^2 \ln n} \geq \frac{1}{2}e^{-3\alpha \ln n} = \frac{1}{2}n^{-3\alpha}. \qquad \square$$

Probabilistic tools. An important foundational process is the *two-way epidemic process* for efficiently propagating a piece of information from a single agent to the whole population.

We also consider a generalization, the *roll call process*, where every agent simultaneously propagates a unique piece of information (its *name*). We build upon bounds from [48] to show this process is also efficient (only 1.5 times slower than the original epidemic process). This process appears in two of our protocols, but also gives upper bounds on the time needed for any parallel information propagation, since after the roll call process completes, every agent has had a chance to "hear from" every other agent. The analysis of these two processes gives tight large deviation bounds with specific constants. While getting these precise constants was more than what is strictly necessary for the proofs in this work, the analysis of these processes may be of independent interest. This roll call process has been independently analyzed in [19, 23, 40, 49].

3 RESETTING SUBPROTOCOL

Propagate-Reset (Protocol 2) is used as a subroutine in both of our protocols Optimal-Silent-SSR (Sec. 4) and Sublinear-Time-SSR (Sec. 5). Intuitively, it provides a way for agents (upon detecting an error that indicates the starting configuration was "illegal" in some way) to "reset" quickly, after which they may be analyzed as though they began from the reset state. For that, the protocol Reset has to be defined for use by Propagate-Reset. We assume that Reset changes the role variable to something different from Resetting. Crucially, after the reset, agents have no information about whether a reset has happened and do not attempt any synchronization to ensure they only reset once, lest the adversary simply sets every agent to believe it has already reset, preventing the necessary reset from ever occurring.[9]

We now define some terms used in the analysis of Propagate-Reset, and their intuition:

If a.role $\neq$ Resetting, then we say a is *computing* (it is executing the outside protocol). Otherwise, for a.role = Resetting, we use three terms. If a.resetcount = $R_{\max}$, we say a is *triggered* (it has just detected an error and initiated this global reset). If a.resetcount > 0 we say a is *propagating* (intuitively this property of positivity spreads by epidemic to restart the whole population; we also consider triggered agents to be propagating). If a.resetcount = 0, we say a is *dormant* (it is waiting for a delay to allow the entire population to become dormant before they start waking up, this prevents an agent from waking up multiple times during one reset).

Likewise, we will refer to a configuration as *fully / partially* propagating (resp. dormant, computing, triggered) if all / some agents are propagating (resp. dormant, computing, triggered).

[9] This is unlike in standard population protocol techniques in which "phase information" is carried in agents indicating whether they are encountering an agent "before" or "after" a new phase starts.

A configuration C is *awakening* if it is the first partially computing configuration reachable from a fully dormant configuration. Protocols that use Propagate-Reset will start their analysis by reasoning about an awakening configuration, which formalizes the idea of having gone through a "clean reset". In an awakening configuration, all agents are dormant except one agent who has just executed Reset. Computing agents will awaken dormant agents by epidemic, so within $O(\log n)$ time, all agents will have executed Reset once and then be back to executing the main algorithm.

Protocol 2 Propagate-Reset(a,b), for Resetting agent a interacting with agent b.

Fields: If `role` = Resetting, `resetcount` $\in \{0, 1, \ldots, R_{\max}\}$ and when `resetcount` = 0 an additional field `delaytimer` $\in \{0, 1, \ldots, D_{\max}\}$.

1: **if** a.`resetcount` > 0 and b.`role` $\neq$ Resetting **then** // *bring b into* Resetting *role*
2: $b \leftarrow$ Resetting, b.`resetcount` $\leftarrow 0$
3: b.`delaytimer` $\leftarrow D_{\max}$
4: **if** b.`role` = Resetting **then** // *change* `resetcount`
5: a.`resetcount`, b.`resetcount` $\leftarrow \max(a$.`resetcount` $- 1, b$.`resetcount` $- 1, 0)$
6: **for** $i \in \{a, b\}$ with i.`role` = Resetting and i.`resetcount` = 0 **do** // *dormant agents*
7: **if** i.`resetcount` just became 0 **then** // *initialize* `delaytimer`
8: i.`delaytimer` $\leftarrow D_{\max}$
9: **else**
10: i.`delaytimer` $\leftarrow i$.`delaytimer` $- 1$
11: **if** i.`delaytimer` = 0 or b.`role` $\neq$ Resetting **then** // *awaken by epidemic*
12: execute Reset(i) // *Reset subroutine provided by main protocol*

We require $R_{\max} = \Omega(\log n)$, and for our protocol will choose the concrete value $R_{\max} = 60 \ln n$. We also require $D_{\max} = \Omega(R_{\max})$. For our $O(\log n)$ time protocol Sublinear-Time-SSR, we have $D_{\max} = \Theta(\log n)$. In Optimal-Silent-SSR, we set $D_{\max} = \Theta(n)$, to give enough time for the dormant agents to do a slow leader election so they finish reset with a unique leader.

Propagate-Reset begins by some agent becoming triggered (`resetcount` $= R_{\max}$). Although introduced for a different purpose, Propagate-Reset is essentially equivalent to a subprotocol used in [5], so we adopt their time analysis to prove it completes in $O(\log n)$ time. Briefly, from a partially triggered configuration, the propagating condition (`resetcount` > 0) spreads by epidemic (in $O(\log n)$ time) . Once the configuration is fully propagating, it becomes fully dormant in $O(\log n)$ time . From the fully dormant configuration, we reach an awakening configuration within $O(\log n)$ time when the first agent executes Reset . Then the instruction to execute Reset spreads by epidemic (in $O(\log n)$ time).

4 LINEAR-TIME, LINEAR-STATE, SILENT PROTOCOL

In this section, we present a silent self-stabilizing ranking protocol, Optimal-Silent-SSR, which achieves asymptotically optimal $O(n)$ time and state complexity. Like Silent-n-state-SSR, there will be a unique stable and silent configuration where every agent has a unique rank, but now a rank collision will trigger our Propagate-Reset, causing the entire population to reset. The key idea behind Optimal-Silent-SSR is to add a large delay $D_{\max} = \Theta(n)$ in the Propagate-Reset, which will ensure that the entire population is dormant for long enough to do a simple slow leader election via $L, L \to L, F$, where all agents set themselves to L upon entering the Resetting role. Thus after the population has undergone a reset, we have a unique leader with high probability. After this reset, we do a linear-time leader-driven ranking, where the ranks correspond to nodes in a full binary tree rooted at the leader. In this ranking algorithm, each agent that has been assigned a rank (starting with the leader) assigns ranks directly to 0, 1, or 2 other agents (depending on its number of children in the tree).

In more detail, each agent can be classified into three roles: Settled, Unsettled, and Resetting. A Settled agent has the field `rank` $\in \{1, 2, ..., n\}$. Each individual role will use $O(n)$ states, and the total state set is the disjoint sum of these roles, for a total of $O(n)$ states. On the other hand, an Unsettled agent has no rank, and it waits for the assignment of a rank from Settled agents.

We use the subprotocol Propagate-Reset described in Section 3 to reset each agent when detecting errors. For Optimal-Silent-SSR, the resetting process is triggered under two different situations. 1) Two Settled agents have an identical rank. The rank conflict can be detected when the two agents interact. 2) An Unsettled agent does not get its rank after $\Theta(n)$ interactions.

During the dormant phase of Propagate-Reset, lasting for $\Theta(n)$ time in this protocol, we do slow leader election via $L, L \to L, F$. Upon awakening (calling Reset), the (likely unique) leader L is Settled with `rank` = 1 and followers F are Unsettled. Thus, after resetting, with high probability there will be exactly one Settled agent with `rank` = 1, and all the other agents are Unsettled. The Settled agent will act as a leader to assign ranks to all Unsettled agents in the following way. At this point the protocol executes an initialized ranking algorithm, similar to others in the renaming literature [3, 4]. Intuitively, a full binary tree forms within the population. Each Settled agent recruits at most two Unsettled agents, assigning them ranks based on its own to guarantee uniqueness. The children of rank i are $2i$ and $2i + 1$; in other words if an agent's rank has binary expansion s, its childrens' ranks have binary expansions $s0$ and $s1$. Since each agent knows the exact population size, each knows whether its rank corresponds to a node with 0, 1, or 2 children in the full binary tree with n nodes. See Figure 1 for an example. This process clearly terminates when all agents are recruited and become settled into different ranks.

Optimal-Silent-SSR takes linear time by the following high-level argument: If there is a rank collision, this is detected in $O(n)$ time. If any agent remains Unsettled without a rank, this is detected via counting up to `errorcount` in $O(n)$ time. Either of these triggers a call to Propagate-Reset.

This reset finishes and reaches a fully computing configuration in $O(n)$ time . There is a constant probability the slow leader election fails (i.e., we end up with multiple leaders), but the expected number of times we must repeat this process before getting a unique leader is constant. The $O(n)$ time of this ranking protocol follows by analyzing each level of the binary tree created across the population:

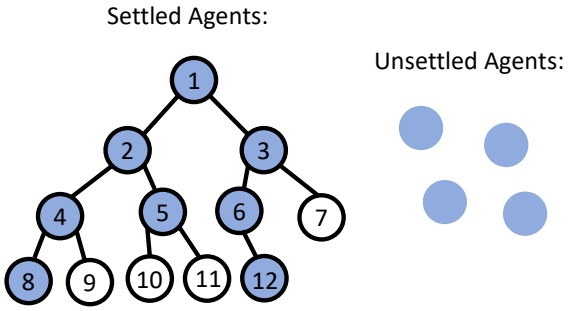


Figure 1: Example of rank assignment in Optimal-Silent-SSR with $n = 12$ agents. There are 8 settled agents on the left (blue circles), with ranks shown. There are 4 ranks in the binary tree left to be filled by the unsettled agents, when they interact with the settled agents with ranks 3,4 or 5. This process completes in expected $\Theta(n)$ time.

each level takes time proportional to the number of nodes in the level, whence the time is proportional to the size of the tree, i.e., $O(n)$.

Protocol 3 Optimal-Silent-SSR, for initiator a interacting with responder b

Fields: `role` $\in$ {Settled, Unsettled, Resetting}
If `role` = Settled, `rank` $\in \{1, ..., n\}$, `children` $\in \{0, 1, 2\}$
If `role` = Unsettled, `errorcount` $\in \{0, 1, ..., E_{\max} = \Theta(n)\}$
If `role` = Resetting, `leader` $\in \{L, F\}$, `resetcount` $\in \{1, \ldots, R_{\max}\}$, `delaytimer` $\in \{0, 1, \ldots, D_{\max} = \Theta(n)\}$

1: **if** a.`role` = Resetting or b.`role` = Resetting **then**
2: execute Propagate-Reset(a,b)
3: **if** a.`leader` = L and b.`leader` = L **then**
4: b.`leader` $\leftarrow F$
5: **if** a.`role` = b.`role` = Settled and a.`rank` = b.`rank` **then**
6: a.`role`, b.`role` $\leftarrow$ Resetting
7: a.`resetcount`, b.`resetcount` $\leftarrow R_{\max}$
8: a.`leader`, b.`leader` $\leftarrow L$
9: **for** $(i, j) \in \{(a, b), (b, a)\}$ **do**
10: **if** i.`role` = Settled, j.`role` = Unsettled, i.`children` < 2, and $2 \cdot i$.`rank` + i.`children` $< n$ **then**
11: j.`role` $\leftarrow$ Settled, j.`children` $\leftarrow 0$
// j becomes a child node of i
12: j.`rank` $\leftarrow 2 \cdot i$.`rank` + i.`children`
13: i.`children` $\leftarrow i$.`children` + 1
14: **for** $i \in \{a, b\}$ **do**
15: **if** i.`role` = Unsettled **then**
16: i.`errorcount` $\leftarrow \max(i$.`errorcount` $- 1, 0)$
17: **if** i.`errorcount` = 0 **then**
18: a.`role`, b.`role` $\leftarrow$ Resetting,
19: a.`resetcount`, b.`resetcount` $\leftarrow R_{\max}$
20: a.`leader`, b.`leader` $\leftarrow L$

The required proofs for Optimal-Silent-SSR are given in [21] and yield the following main results:

Protocol 4 Reset(a) for Optimal-Silent-SSR, for agent a. (Called in line 12 of Propagate-Reset.)

1: **if** a.`leader` = L **then**
2: a.`role` $\leftarrow$ Settled, a.`rank` $\leftarrow 1$, a.`children` $\leftarrow 0$
3: **if** a.`leader` = F **then**
4: a.`role` $\leftarrow$ Unsettled, a.`errorcount` $\leftarrow E_{\max} = \Theta(n)$

Theorem 4.1. *Optimal-Silent-SSR is a silent protocol solving self-stabilizing ranking with $O(n)$ states and $O(n)$ expected time.*

Corollary 4.2. *Optimal-Silent-SSR takes $O(n \log n)$ time with high probability.*

5 LOGARITHMIC-TIME PROTOCOL

In this section, we show a protocol solving SSR, and thus SSLE, in optimal $O(\log n)$ expected time, using a "quasi-exponential" number of states: $\exp\left(O(n^{\log n} \cdot \log n)\right)$. Observation 2.2 shows that to achieve sublinear time, the protocol necessarily must be non-silent: agents change states forever.

5.1 Overview

Intuitively, Sublinear-Time-SSR works as follows. Each agent has a field `name`, a bitstring of length $3 \log_2 n$. The n^3 possible values ensure that if all agents pick a `name` randomly, with high probability, there are no collisions. The set of all `name` values in the population is propagated by epidemic in $O(\log n)$ time in a field called `roster` (which has an exponential number of possible values). Agents update their `rank` (a write-only output field) only when their `roster` field has size n; in this case the agent's `rank` is its `name`'s lexicographical order in the set `roster`.

One source of error is that we can start in a configuration with a "*ghost name*": a name that is in the `roster` set of some agent, but that is not the `name` of any agent. If there are no collisions among actual `name`'s, this error is easy to handle: eventually we will have $|$`roster`$| > n$, indicating that there is a ghost name, triggering Propagate-Reset.[10]

The main challenge is then to detect name collisions. Sublinear-Time-SSR calls a subroutine Detect-Name-Collision that detects whether two agents have the same `name`. If so, we call the same subroutine Propagate-Reset used in Optimal-Silent-SSR, now with $D_{\max} = \Theta(\log n)$ rather than $\Theta(n)$ as in Optimal-Silent-SSR. Upon awakening from Propagate-Reset, agents pick a new name randomly. They use their dormant time, while still in role Resetting, but with `resetcount` = 0 while counting `delaytimer` down to 0, to generate random bits to pick a new random name.

The bulk of the analysis is in devising an $O(\log n)$ time protocol implementing Detect-Name-Collision. The rest of the protocol outside of Detect-Name-Collision is silent: once the protocol stabilizes, no `name` or `rank` field changes. Indeed, we can implement a silent protocol on top of this scheme if we are content with $\Theta(n)$ time: Detect-Name-Collision can be implemented with the

[10] We will introduce a tree data structure that also has all the names. However, it is necessary to keep a separate set `roster` of names for the following reason. The set `roster` is propagated in time $O(\log n)$, whereas in slower variants of our algorithm, the tree takes too long to collect the names. For example, in the $O(\sqrt{n})$ time (and uses less memory) variant, the tree takes time $\Omega(n)$ to populate with all names.

simple rule that checks whether the name fields of the two interacting agents are equal, i.e., direct collision detection. The challenge, therefore, is in implementing Detect-Name-Collision in sublinear time by *indirectly* detecting collisions, without requiring agents with the same name to meet directly. Any method of doing this will necessarily be non-silent.

The protocol Detect-Name-Collision is parameterized to give a tradeoff between stabilizing time and state complexity. For instance, there is a $O(\sqrt{n})$ time protocol that uses a data structure with kn bits for a parameter k, i.e., 2^{kn} possible values. Of course, all of the schemes use at least exponential states, since the field roster has $\approx n^{3n}$ possible values. However, the faster schemes will use even more states than this, and their analysis is more complex. This is discussed in more detail in Section 5.2.

Protocol 5 Sublinear-Time-SSR, for agent a interacting with b.
Fields: role $\in$ {Collecting, Resetting}, name $\in \{0,1\}^{\leq 3\log_2 n}$
If role = Collecting, rank $\in \{1,\dots,n\}$, roster $\subseteq \{0,1\}^{\leq 3\log_2 n}$, |roster| $\leq n$, other fields from Detect-Name-Collision
If role = Resetting, resetcount $\in \{1,\dots,R_{\max}\}$, delaytimer $\in \{0,1,\dots,D_{\max} = \Theta(\log n)\}$

1: **if** a.role = b.role = Collecting **then**
2: **if** Detect-Name-Collision(a,b) or $|a.\text{roster} \cup b.\text{roster}| > n$ **then**
3: a.role, b.role $\leftarrow$ Resetting
4: a.resetcount, b.resetcount $\leftarrow R_{\max}$
5: **else**
6: a.roster, b.roster $\leftarrow a.\text{roster} \cup b.\text{roster}$
7: **if** $|a.\text{roster} \cup b.\text{roster}| = n$ **then**
// *do not set rank until all names collected*
8: **for** $i \in \{a,b\}$ **do**
9: i.rank $\leftarrow$ order of i.name in roster
10: **else** // *some agent is* Resetting
11: execute Propagate-Reset(a,b)
12: **for** $i \in \{a,b\}$ such that $|i.\text{resetcount}| > 0$ **do**
// *clear names while propagating the reset signal*
13: i.name $\leftarrow \varepsilon$
14: **for** $i \in \{a,b\}$ such that i.resetcount = 0 and $|i.\text{name}| < 3\log_2 n$ **do**
15: append a random bit to i.name // *can be derandomized*

Protocol 6 Reset(a) for Sublinear-Time-SSR, for agent a.
(Called in line 12 of Propagate-Reset.)

1: role $\leftarrow$ Collecting
2: roster $\leftarrow$ {name}

5.2 Fast Collision Detection

In Sublinear-Time-SSR, both Propagate-Reset and filling all agents' roster take $O(\log n)$ time, so the time bottleneck is waiting to detect a name collision. If we simply wait for two agents with the same name to meet to detect a collision, this will take $\Theta(n)$ time in the worst case, which would give a $\Theta(n)$ time silent algorithm.

The goal of Detect-Name-Collision is to detect these names collisions in sublinear time. Because of the lower bound of Observation 2.2, this protocol must not be silent. Detect-Name-Collision will have to satisfy two conditions. In order to allow $O(\log n)$ time convergence, from any configuration with a name collision, some agent must detect this collision in $O(\log n)$ time to initiate Propagate-Reset. Second, to ensure the eventual ranked configuration is stable, it must satisfy a safety condition where from a configuration with unique names, no agent will ever think there is a name collision.[11]

As a warm-up to the full $O(\log n)$-time protocol of Detect-Name-Collision, consider the following simpler $O(\sqrt{n})$-time protocol. Each agent keeps a dictionary keyed by names of other agents they have encountered in the population. Whenever agents a and b meet, they generate a random shared value sync $\in \{1,\dots,k\}$, which a stores in its dictionary keyed by the name of b, and b stores in its dictionary keyed by the name of a. If the two agents disagree on this sync value at the beginning of an interaction, they declare a name collision.

From a configuration with two agents a and a' sharing the same name, within $O(\sqrt{n})$ time, some agent b will interact with both a and a' (assume b first interacts with a, then a'). With probability $1 - \frac{1}{k}$, the sync value that b generates with a will disagree with the sync value that a' has with b. So when b then meets a', it is able to detect a name collision. From a configuration with unique names, an invariant is maintained that all pairs of agents agree on their corresponding sync values, giving the required safety property.

The actual protocol Detect-Name-Collision is a generalization of this idea. The agents now store a more complicated data structure: a tree whose nodes are labelled by names. See Figure 2 for an example. The root is labelled by the agent's own name, and every root-to-leaf path is *simply labelled*, meaning that each node on the path contains unique names (it is permitted for the same name to appear on multiple nodes in the tree, but neither of these nodes can be an ancestor of the other). Each edge is labelled by a sync value. The intuition is that these paths correspond to *histories*: chains of interactions between agents, where the sync values on the edges were generated by the interaction between that pair of agents. For instance, if a has a path $a \xrightarrow{3} b \xrightarrow{5} c \xrightarrow{7} d$, the interpretation is that when a last met b, a and b generated sync value 3, and in that interaction, b told a that when b last met c, b and c generated sync value 5, and in that interaction, c told b that when c last met d, c and d generated sync value 7. In particular, it could be that c and d have interacted again, generating a different sync value than 7, before a and b interact, but b has not heard about that interaction. See Fig. 2 for an example showing how this information is built up.

The $O(\sqrt{n})$ time algorithm above can be thought of as a tree of depth 1, where each agent stores only the names and sync values of the agents it has directly interacted with. The general algorithm has a tree of depth H, which allows agents to hear about other agents' sync values through longer chains of interactions. In line 3 of Detect-Name-Collision, each agent checks any paths ending at the name of the other agent (the additional fields edge.timer

[11] The initial configuration could have unique names, but with auxiliary data adversarially planted to mislead agents into believing there is a name collision, triggering a reset. So the actual safety condition is more subtle and involves unique-name configurations reachable only after a reset.

Protocol 7 Detect-Name-Collision(a,b) for Sublinear-Time-SSR, for agent a, b.
Fields: tree: depth H, root labelled name, other nodes have node.name ∈ roster. Edges have edge.sync ∈ $\{1, \ldots, S_{\max} = \Theta(n^2)\}$ and edge.timer ∈ $\{0, \ldots, T_H\}$. The parameter $T_H = \Theta(H \cdot n^{1/(H+1)})$ for $H = O(1)$ and $T_H = \Theta(\log n)$ for $H = \Theta(\log n)$ (we need $T_H = \Theta(\tau_{H+1})$).

for $(i, j) \in \{(a, b), (b, a)\}$ **do**
  for every path $(i.e_1, \ldots, i.e_p)$ in i.tree with $i.e_1$.timer, $\ldots$, $i.e_p$.timer > 0 and last node v with v.name $= j$.name **do**
    // *All of i's histories about j that aren't outdated*
    if Check-Path-Consistency$(j, (i.e_1, \ldots, i.e_p)) =$ Inconsistent **then**
      Return True // *collision detected*
$x \leftarrow$ chosen uniformly at random from $\{1, \ldots, S_{\max}\}$ // *Choose new sync value*
for $(i, j) \in \{(a, b), (b, a)\}$ **do** // *Update trees to share new information*
  if i.tree has node v at depth 1 with v.name $= j$.name **then**
    Remove the subtree rooted at v from i.tree
  Add j.tree (to depth $H - 1$) as a subtree of i.tree via new edge e from the root
  e.sync $\leftarrow x$, e.timer $\leftarrow T_H$
for $i \in \{a, b\}$ **do** // *Keep the trees simply labelled*
  remove from i.tree all subtrees with root labelled with i.name
for each edge e in a.tree and b.tree **do**
  e.timer $\leftarrow \max(e$.timer $- 1, 0)$
Return False // *no collision detected*

are a technicality to handle certain adversarial initial conditions). Intuitively, they require the other agent to show information that is logically consistent with this path, formalized in the conditions of Check-Path-Consistency. To detect a name collision between agents a and a', it will now suffice for some agent b to have heard about agent a before meeting a'. With constant probability, the duplicate agent a' will not have any sync values that are logically consistent with this path, and b will declare a collision. Allowing longer paths decreases the time it takes for this information to travel between a and a'. Because the paths that spread information in the epidemic process have length at most $O(\log n)$ with high probability [21], once we take $H = O(\log n)$, in the $O(\log n)$ time it would take for an epidemic starting at a to reach a', some agent will detect a collision in this way.

Protocol 8 Check-Path-Consistency(j,P) for Detect-Name-Collision, for agent j verifying path $P = (i.e_1, \ldots, i.e_p)$

$q \leftarrow \min\{q' \mid \exists(j.e_p, \ldots, j.e_{q'})$ in j.tree$\}$ // $(j.e_p, \ldots, j.e_q)$ *is a root-to-leaf path*
for edge $j.e \in (j.e_p, \ldots, j.e_q)$ **do**
  if $j.e$.sync $= i.e$.sync **then**
    Return True
Return Inconsistent

Detect-Name-Collision works in $O(T_H)$ time, and also satisfies required safety conditions that ensure there are no "false positives" where collisions are detected from configurations with unique names. These results will let us prove the main theorem about the behavior of Sublinear-Time-SSR:

Theorem 5.1. *Sublinear-Time-SSR uses* $\exp\left(O(n^H)\log n\right)$ *states. When* $H = O(1)$, *Sublinear-Time-SSR solves self-stabilizing ranking in expected* $O(H \cdot n^{1/(H+1)})$ *time, and* $O(H \cdot \log n \cdot n^{1/(H+1)})$ *time with high probability* $1 - O(1/n)$. *When* $H = \Theta(\log n)$, *Sublinear-Time-SSR solves self-stabilizing ranking in time* $O(\log n)$, *in expectation and with high probability* $1 - O(1/n)$.

6 CONCLUSION AND PERSPECTIVES

For the first time, we addressed time-space trade-offs of self-stabilizing leader election and ranking in population protocols over complete graphs. We emphasize that solving these problems, while ensuring such a strong form of fault-tolerance, necessitates linear states and strong nonuniformity (Theorem 2.1). Other forms of "strong" fault-tolerance, such as Byzantine-tolerance [39] or loosely-stabilizing leader election with *exponential holding time* (a period of time where a unique leader persists after stabilization) [41, 56], similarly necessitate $\Omega(n)$ states. By contrast, a sublinear number of states suffices for many non-fault-tolerant protocols (cf. [6]) and weaker forms of tolerance, such as loosely-stabilizing leader election with *polynomial* holding time [56] or tolerance to a *constant* number of crashes and transient faults [27].

To conclude, we propose several perspectives.

Time/space tradeoffs. It is open to find a subexponential-state sublinear-time self-stabilizing ranking protocol. Observation 2.2 states that any sublinear time SSR protocol is not silent. Sublinear-Time-SSR is non-silent because it perpetually passes around information about agents' recent interactions with each other, as a way to detect name collisions without requiring the agents with equal names to meet directly. Even when limiting the tree of interactions to depth 1, this results in an exponential number of states, since each agent must maintain a value to associate to every other agent in the population. Thus, a subexponential-state protocol (if based upon fast collision detection) would somehow need to embed enough information in each agent to enable fast collision detection, while somehow allowing the agent to forget "most" of the information about its interactions. Furthermore, our strategy of using the set roster of all names to go from unique names to unique ranks fundamentally requires exponential states.

Ranking vs. leader election. Ranking implies leader election ("automatically"), but the converse does not hold. In the *initialized* case where we can specify an initial state for each agent, it is possible to elect a leader without ranking, using the single transition $\ell, \ell \to \ell, f$ (using too few states for the ranking problem even to be definable). Though any *self-stabilizing* protocol for leader election must use at least n states [22] (Theorem 2.1 here), it is not the case that any SSLE protocol implicitly solves the ranking problem. It would be interesting to discover an SSLE algorithm that is more efficient than our examples because it does not also solve ranking.

Initialized ranking. Consider the ranking problem in a non-self-stabilizing setting. Without the constraint of self-stabilization, there is no longer the issue of ghost names. Compared to self-stabilization,

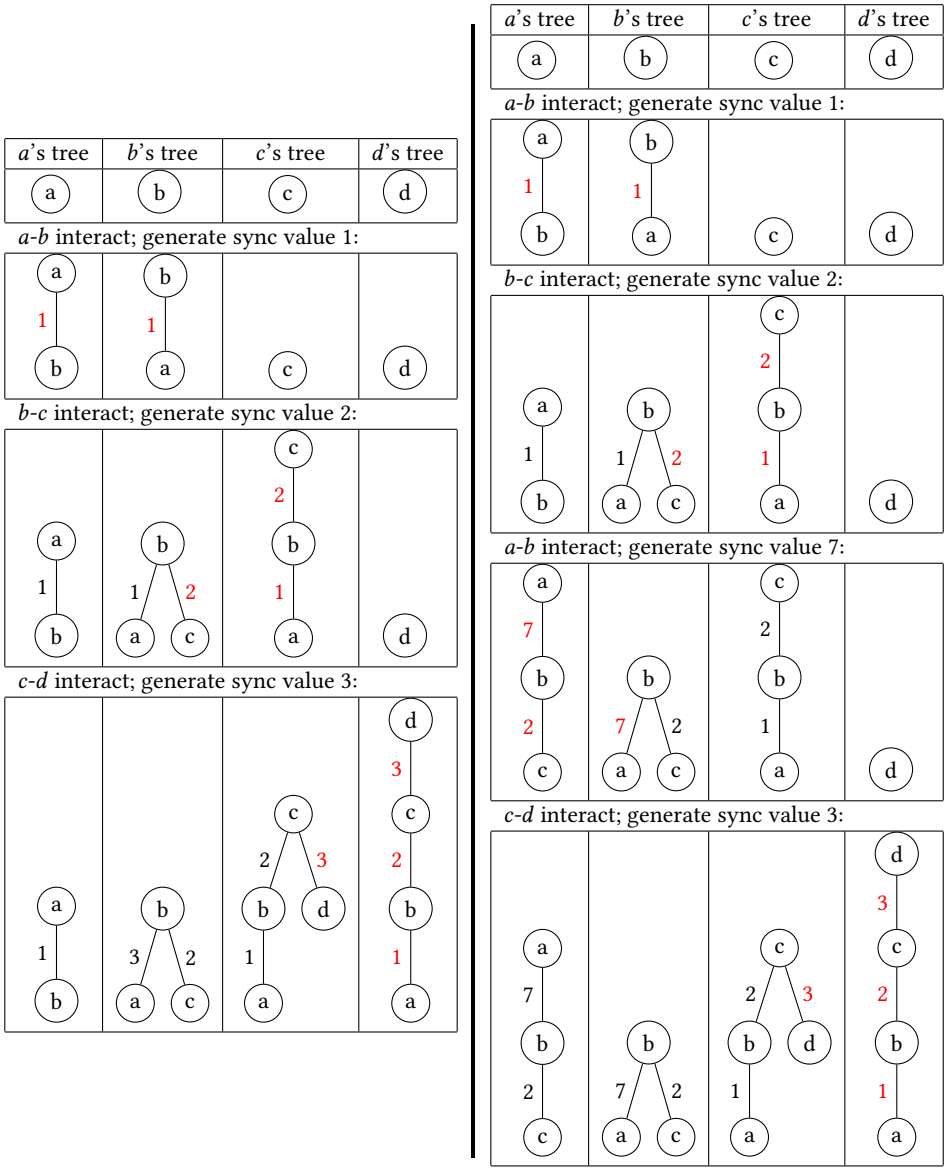


Figure 2: Example executions building up trees in agents, starting from a "clean" configuration with singleton trees. Red sync values are newly generated or communicated in the preceding interaction. As an example of how agents check for consistency, when a and d interact, before updating their trees, d checks any paths p that end with a (here there's just one, $d \xrightarrow{3} c \xrightarrow{2} b \xrightarrow{1} a$) against a's corresponding path, which is a's longest reversed suffix of p. Left: a's reverse suffix is $a \xrightarrow{1} b$, with just a single edge that matches the final sync value in this path p, so Check-Path-Consistency will return True after checking the first edge. Right: a's reverse suffix is $a \xrightarrow{7} b \xrightarrow{2} c$. The first edge $a \xrightarrow{7} b$ does not match d's tree, because agents a and b generated the new sync value 7 in a later interaction. However, in that interaction, a added the edge $b \xrightarrow{2} c$, hearing about the b-c interaction with sync value 2 that matches the path in d's tree. Now Check-Path-Consistency will return True after checking the second edge.

it may be easier to find an *initialized* ranking protocol that still uses polylogarithmic time, but only polynomial states.

Initialized collision detection. The core difficulty of Sublinear-Time-SSR is *collision detection*. It would be interesting to study this problem in the (non-self-stabilizing) setting where an adversary assigns read-only names to each agent, but the read/write memory can be initialized to the same state for each agent. Can a name collision be detected in sublinear time and sub-exponential states?

REFERENCES

[1] D. Alistarh, J. Aspnes, D. Eisenstat, R. Gelashvili, and R. L. Rivest. 2017. Time-Space Trade-offs in Population Protocols. In *SODA*. 2560–2579.

[2] D. Alistarh, J. Aspnes, and R. Gelashvili. 2018. Space-Optimal Majority in Population Protocols. In *SODA*. 2221–2239.

[3] Dan Alistarh, Hagit Attiya, Seth Gilbert, Andrei Giurgiu, and Rachid Guerraoui. 2010. Fast randomized test-and-set and renaming. In *DISC 2010: International Symposium on Distributed Computing*. Springer, 94–108.

[4] Dan Alistarh, Oksana Denysyuk, Luís Rodrigues, and Nir Shavit. 2014. Balls-into-leaves: Sub-logarithmic renaming in synchronous message-passing systems. In *PODC 2014: Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing*. 232–241.

[5] D. Alistarh, B. Dudek, A. Kosowski, D. Soloveichik, and P. Uznanski. 2017. Robust Detection in Leak-Prone Population Protocols. In *DNA*. 155–171. https://doi.org/10.1007/978-3-319-66799-7_11

[6] D. Alistarh and R. Gelashvili. 2018. Recent Algorithmic Advances in Population Protocols. *SIGACT News* 49, 3 (2018), 63–73. https://doi.org/10.1145/3289137.3289150

[7] Talley Amir, James Aspnes, David Doty, Mahsa Eftekhari, and Eric Severson. 2020. Message complexity of population protocols. In *DISC 2020: 34th International Symposium on Distributed Computing (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 179)*, Hagit Attiya (Ed.). Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 6:1–6:18. https://doi.org/10.4230/LIPIcs.DISC.2020.6

[8] D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta. 2006. Computation in networks of passively mobile finite-state sensors. *Distributed Computing* 18, 4 (2006), 235–253.

[9] D. Angluin, J. Aspnes, and D. Eisenstat. 2008. Fast computation by population protocols with a leader. *Distributed Computing* 21, 3 (2008), 183–199.

[10] D. Angluin, J. Aspnes, M. J. Fischer, and H. Jiang. 2005. Self-stabilizing Population Protocols. In *OPODIS*, Vol. 3974. Springer, 103–117. https://doi.org/10.1007/11795490_10

[11] J. Beauquier, P. Blanchard, and J. Burman. 2013. Self-stabilizing Leader Election in Population Protocols over Arbitrary Communication Graphs. In *OPODIS*. 38–52.

[12] J. Beauquier, J. Clement, S. Messika, L. Rosaz, and B. Rozoy. 2007. Self-Stabilizing Counting in Mobile Sensor Networks with a base station. In *DISC*. 63–76.

[13] A. Belleville, D. Doty, and D. Soloveichik. 2017. Hardness of Computing and Approximating Predicates and Functions with Leaderless Population Protocols. In *ICALP*. 141:1–141:14. https://doi.org/10.4230/LIPIcs.ICALP.2017.141

[14] P. Berenbrink, G. Giakkoupis, and P. Kling. 2020. Optimal time and space leader election in population protocols. In *STOC*. ACM, 119–129. https://doi.org/10.1145/3357713.3384312

[15] M. Blondin, J. Esparza, and S. Jaax. 2018. Large Flocks of Small Birds: on the Minimal Size of Population Protocols. In *STACS*. 16:1–16:14. https://doi.org/10.4230/LIPIcs.STACS.2018.16

[16] O. Bournez, J. Chalopin, J. Cohen, and X. Koegler. 2008. Playing With Population Protocols. In *CSP 2008*. 3–15.

[17] O. Bournez, J. Cohen, and M. Rabie. 2018. Homonym Population Protocols. *Theory of Computing Systems* 62, 5 (2018), 1318–1346.

[18] J. M. Bower and H. Bolouri. 2004. *Computational modeling of genetic and biochemical networks*. MIT press.

[19] D. W. Boyd and J. M. Steele. 1979. Random Exchanges of Information. *Journal of Applied Probability* 16, 3 (1979), 657–661. http://www.jstor.org/stable/3213094

[20] J. Burman, J. Beauquier, and D. Sohier. 2019. Space-Optimal Naming in Population Protocols. In *DISC'19*, J. Suomela (Ed.), Vol. 146. 9:1–9:16. https://doi.org/10.4230/LIPIcs.DISC.2019.9

[21] Janna Burman, Ho-Lin Chen, Hsueh-Ping Chen, David Doty, Thomas Nowak, Eric Severson, and Chuan Xu. 2021. *Time-optimal self-stabilizing leader election in population protocols (full version of this paper)*. Technical Report 1907.06068. arXiv. http://arxiv.org/abs/1907.06068

[22] S. Cai, T. Izumi, and K. Wada. 2012. How to prove impossibility under global fairness: On space complexity of self-stabilizing leader election on a population protocol model. *Theory of Computing Systems* 50, 3 (2012), 433–445.

[23] A. Casteigts, M. Raskin, M. Renken, and V. Zamaraev. 2020. Sharp Thresholds in Random Simple Temporal Graphs. arXiv:2011.03738 [cs.DM]

[24] Cameron Chalk, Niels Kornerup, Wyatt Reeves, and David Soloveichik. 2021. Composable rate-independent computation in continuous chemical reaction networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 18, 1 (2021), 250–260. https://doi.org/10.1109/TCBB.2019.2952836 special issue of invited papers from CMSB 2018.

[25] H.-P. Chen and H.-L. Chen. 2019. Self-Stabilizing Leader Election. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing* (Toronto ON, Canada) *(PODC '19)*. Association for Computing Machinery, New York, NY, USA, 53–59. https://doi.org/10.1145/3293611.3331616

[26] H.-P. Chen and H.-L. Chen. 2020. Self-Stabilizing Leader Election in Regular Graphs. In *Proceedings of the 39th Symposium on Principles of Distributed Computing* (Virtual Event, Italy) *(PODC '20)*. Association for Computing Machinery, New York, NY, USA, 210–217. https://doi.org/10.1145/3382734.3405733

[27] C. Delporte-Gallet, H. Fauconnier, R. Guerraoui, and E. Ruppert. 2006. When Birds Die: Making Population Protocols Fault-Tolerant. In *DCOSS*. 51–66.

[28] Z. Diamadi and M. J. Fischer. 2001. A simple game for the study of trust in distributed systems. *Wuhan University Journal of Natural Sciences* 6, 1 (01 Mar 2001), 72–82. https://doi.org/10.1007/BF03160228

[29] E. W. Dijkstra. 1974. Self-stabilizing Systems in Spite of Distributed Control. *Commun. of the ACM* 17, 11 (Nov. 1974), 643–644.

[30] Shlomi Dolev. 2000. *Self-stabilization*. MIT press.

[31] Shlomi Dolev, Amos Israeli, and Shlomo Moran. 1993. Self-Stabilization of Dynamic Systems Assuming Only Read/Write Atomicity. *Distributed Comput.* 7, 1 (1993), 3–16. https://doi.org/10.1007/BF02278851

[32] D. Doty and D. Soloveichik. 2015. Stable Leader Election in Population Protocols Requires Linear Time. In *DISC*. 602–616.

[33] R. Elsässer and T. Radzik. 2018. Recent Results in Population Protocols for Exact Majority and Leader Election. *Bulletin of the EATCS* 126 (2018). http://bulletin.eatcs.org/index.php/beatcs/article/view/549/546

[34] J. Esparza, P. Ganty, J. Leroux, and R. Majumdar. 2017. Verification of population protocols. *Acta Informatica* 54, 2 (2017), 191–215. https://doi.org/10.1007/s00236-016-0272-3

[35] M. J. Fischer and H. Jiang. 2006. Self-stabilizing Leader Election in Networks of Finite-State Anonymous Agents. In *OPODIS*. 395–409.

[36] L. Gasieniec and G. Stachowiak. 2018. Fast Space Optimal Leader Election in Population Protocols. In *SODA*. 2653–2667. https://doi.org/10.1137/1.9781611975031.169

[37] L. Gasieniec, G. Stachowiak, and P. Uznanski. 2019. Almost Logarithmic-Time Space Optimal Leader Election in Population Protocols. In *SPAA*. 93–102. https://doi.org/10.1145/3323165.3323178

[38] D. T. Gillespie. 1977. Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry* 81 (25) (1977), 2340 – 2361.

[39] R. Guerraoui and E. Ruppert. 2009. Names Trump Malice: Tiny Mobile Agents Can Tolerate Byzantine Failures. In *ICALP (2)*. 484–495.

[40] J. Haigh. 1981. Random Exchanges of Information. *Journal of Applied Probability* 18, 3 (1981), 743–746. http://www.jstor.org/stable/3213330

[41] T. Izumi. 2015. On Space and Time Complexity of Loosely-Stabilizing Leader Election. In *SIROCCO (Lecture Notes in Computer Science, Vol. 9439)*. Springer, 299–312. https://doi.org/10.1007/978-3-319-25258-2_21

[42] D. Johnson, T. Stack, R. Fish, D. Montrallo Flickinger, L. Stoller, R. Ricci, and J. Lepreau. 2006. Mobile Emulab: A Robotic Wireless and Sensor Network Testbed. In *INFOCOM*. IEEE. https://doi.org/10.1109/INFOCOM.2006.182

[43] A. Kosowski and P. Uznanski. 2018. Brief Announcement: Population Protocols Are Fast. In *PODC*. 475–477. https://dl.acm.org/citation.cfm?id=3212788

[44] G. Di Luna, P. Flocchini, T. Izumi, T. Izumi, N. Santoro, and G. Viglietta. 2019. Population protocols with faulty interactions: The impact of a leader. *Theoretical Computer Science* 754 (2019), 35–49. https://doi.org/10.1016/j.tcs.2018.09.005

[45] O. Michail, I. Chatzigiannakis, and P. G. Spirakis. 2011. New Models for Population Protocols. *Synthesis Lectures on Distributed Computing Theory* 2, 1 (2011), 1–156. https://doi.org/10.2200/S00328ED1V01Y201101DCT006 arXiv:https://doi.org/10.2200/S00328ED1V01Y201101DCT006

[46] O. Michail, I. Chatzigiannakis, and P. G. Spirakis. 2013. Naming and Counting in Anonymous Unknown Dynamic Networks. In *SSS*. 281–295. https://doi.org/10.1007/978-3-319-03089-0_20

[47] R. Mizoguchi, H. Ono, S. Kijima, and M. Yamashita. 2012. On space complexity of self-stabilizing leader election in mediated population protocol. *Distributed Computing* 25, 6 (2012), 451–460. https://doi.org/10.1007/s00446-012-0173-9

[48] Y. Mocquard, B. Sericola, S. Robert, and E. Anceaume. 2016. Analysis of the propagation time of a rumour in large-scale distributed systems. In *2016 IEEE 15th International Symposium on Network Computing and Applications (NCA)*. IEEE, 264–271.

[49] J. W. Moon. 1972. Random exchanges of information. *Nieuw Archief voor Wiskunde* 20 (1972), 246—-249.

[50] J. Polastre, J. L. Hill, and D. E. Culler. 2004. Versatile low power media access for wireless sensor networks. In *SenSys*. ACM, 95–107. https://doi.org/10.1145/1031495.1031508

[51] M. Rabie. 2017. Global Versus Local Computations: Fast Computing with Identifiers. In *SIROCCO*. 90–105. https://doi.org/10.1007/978-3-319-72050-0_6

[52] Eric Severson, David Haley, and David Doty. 2020. Composable computation in discrete chemical reaction networks. *Distributed Computing* (2020). to appear. Special issue of invited papers from PODC 2019.

[53] D. Soloveichik, M. Cook, E. Winfree, and J. Bruck. 2008. Computation with finite stochastic chemical reaction networks. *Natural Computing* 7, 4 (2008), 615–633. https://doi.org/10.1007/s11047-008-9067-y

[54] Y. Sudo and T. Masuzawa. 2020. Leader election requires logarithmic time in population protocols. *Parallel Processing Letters* 30, 01 (2020), 2050005.

[55] Y. Sudo, F. Ooshita, T. Izumi, H. Kakugawa, and T. Masuzawa. 2020. Time-Optimal Leader Election in Population Protocols. *IEEE Transactions on Parallel and Distributed Systems* 31, 11 (2020), 2620–2632. https://doi.org/10.1109/TPDS.2020.2991771

[56] Y. Sudo, F. Ooshita, H. Kakugawa, T. Masuzawa, A. K. Datta, and L. L. Larmore. 2020. Loosely-stabilizing leader election with polylogarithmic convergence time. *Theor. Comput. Sci.* 806 (2020), 617–631. https://doi.org/10.1016/j.tcs.2019.09.034
[57] Y. Sudo, M. Shibata, J. Nakamura, Y. Kim, and T. Masuzawa. 2020. The Power of Global Knowledge on Self-stabilizing Population Protocols. In *SIROCCO*, Vol. 12156. Springer, 237–254. https://doi.org/10.1007/978-3-030-54921-3_14
[58] X. Xu, Y. Yamauchi, S. Kijima, and M. Yamashita. 2013. Space Complexity of Self-Stabilizing Leader Election in Population Protocol Based on k-Interaction. In *SSS*. 86–97. https://doi.org/10.1007/978-3-319-03089-0_7
[59] H. Yasumi, F. Ooshita, K. Yamaguchi, and M. Inoue. 2017. Constant-Space Population Protocols for Uniform Bipartition. In *OPODIS 2017*. 19:1–19:17.
[60] D. Yokota, Y. Sudo, and T. Masuzawa. 2020. Time-Optimal Self-stabilizing Leader Election on Rings in Population Protocols. In *SSS*, Vol. 12514. Springer, 301–316. https://doi.org/10.1007/978-3-030-64348-5_24

Preflight Results

Document Information

Title: Time-Optimal Self-Stabilizing Leader Election in Population Protocols
Author: Janna Burman, Ho-Lin Chen, Hsueh-Ping Chen, David Doty, Thomas Nadeau, Robert E. Tarjan, and Chuan Xu
Creator: LaTeX with acmart 2020/04/30 v1.71 Typesetting articles for the Association for Computing Machinery and hyperref 2
Producer: pdfTeX-1.40.21

Preflight Information

Profile: Convert to PDF/A-1b
Version: Qapla' PDF Preflight v2021.1.10
Date: Apr 5, 2022 7:42:58 AM

Legend: (X) - Can NOT be fixed by PDF/A-1b conversion.
(!X) - Could be fixed by PDF/A-1b conversion. User chose to be warned in PDF/A settings.

Page 1 Results

(X) Font widths must be the same in both the font dictionary and the embedded font file.
(X) Font widths must be the same in both the font dictionary and the embedded font file.
(X) Font widths must be the same in both the font dictionary and the embedded font file.
(X) Font widths must be the same in both the font dictionary and the embedded font file.
(X) Font widths must be the same in both the font dictionary and the embedded font file.
(X) Font widths must be the same in both the font dictionary and the embedded font file.
(X) Font widths must be the same in both the font dictionary and the embedded font file.
(X) Font widths must be the same in both the font dictionary and the embedded font file.
(X) Font widths must be the same in both the font dictionary and the embedded font file.