Self-Stabilizing Leader Election in Optimal Space

Ajoy K. Datta, Lawrence L. Larmore, and Priyanka Vemula

School of Computer Science, University of Nevada Las Vegas

Abstract. A silent self-stabilizing asynchronous distributed algorithm, SSLE, for the leader election problem, in a connected unoriented network with unique IDs, is given. SSLE uses $O(\log n)$ space per process and stabilizes in $O(n)$ rounds, where n is the number of processes in the network.

Key words: distributed algorithm, leader election, self-stabilization.

1 Introduction

In this paper, we give a self-stabilizing silent asynchronous distributed algorithm for the *leader election problem*, where all process in a network must agree on which one of them is the leader. A self-stabilizing system, regardless of the initial states of the processes and initial messages in the links, is guaranteed to converge to the intended behavior in finite time; the algorithm is also called *silent* if eventually all execution halts [4, 5].

1.1 Related Work

Arora and Gouda [2] present a silent leader election algorithm in the shared memory model. Their algorithm requires $O(N)$ rounds and $O(\log N)$ space, where N is a given upper bound on n, the size of the network. Dolev and Herman [6] give a non-silent leader election algorithm in the shared memory model. This algorithm takes $O(diam)$ rounds, where *diam* is the diameter of the network, and uses $O(N \log N)$ space. Awerbuch *et al.*[3] solve the leader election problem in the message passing model. Their algorithm takes $O(diam)$ rounds and uses $O(\log D \log N)$ space, where D is a given upper bound on the diameter.

Afek and Bremler [1] introduce the concept of *power supply* which they use to construct an algorithm for the leader election problem in the message passing model. Their algorithm takes $O(n)$ time and uses $O(\log n)$ bits per process. Our algorithm SSLE is partially inspired by Afek and Bremler's algorithm.

1.2 Contributions

We present a self-stabilizing algorithm, SSLE, for the leader election algorithm, in the composite atomicity model of computation. The space complexity of our algorithm is $O(\log n)$ bits per process, and the time complexity is $O(n)$. SSLE does not require knowledge of any upper bounds on n or *diam*.

More precisely. The time complexity of SSLE is actually $O(simp)$, where *simp* is defined to be the length of the longest simple path in the network; hence $simp \leq n-1$. Afek and Bremler's algorithm [1] also takes $O(simp)$ rounds.

1.3 Outline of Paper

In Section 2, we describe our model of computation. In Section 3, we give our self-stabilizing algorithm, SSLE. In Section 4, we give a sketch of the proof of the correctness and time complexity of SSLE. Section 5 concludes the paper.

2 Preliminaries

We are given a connected undirected network, $G = (V, E)$ of $|V| = n$ processes, where $n \geq 2$. Each process P has a unique ID, $P.id$, of *ID type*, which could be any ordered type, but which we take to be non-negative integer. We assume the *shared memory model* of computation introduced in [4]. In this model, a process P can read its own registers and those of its neighbors, but can write only to its own registers.

The *state* of a process is defined by the values of its registers. A *configuration* of the network is a function from processes to states; if γ is the current configuration, then $\gamma(P)$ is the current state of each process P. An *execution* of $\mathcal{A}$ is a sequence of states $e = \gamma_0 \mapsto \gamma_1 \mapsto \ldots \mapsto \gamma_i \ldots$, where $\gamma_i \mapsto \gamma_{i+1}$ means that it is possible for the network to change from configuration γ_i to configuration γ_{i+1} in one step. We say that an execution is *maximal* if it is infinite, or if it ends at a *sink*, *i.e.,* a configuration from which no execution is possible.

The *program* of each process consists of a finite set of *actions* of the following form: $< label >:: < guard > \longrightarrow < statement >$. The *guard* of an action in the program of a process P is a Boolean expression involving the registers of P and its neighbors. The *statement* of an action of P updates one or more variables of P. An action can be executed only if it is *enabled*, *i.e.,* its guard evaluates to true. A process is said to be enabled if at least one of its actions is enabled. A *step* $\gamma_i \mapsto \gamma_{i+1}$ consists of one or more enabled processes executing an action. The evaluations of all guards and executions of all statements of those actions are presumed to take place in one atomic step; this model is called *composite atomicity* [5].

We assume that each transition from a configuration to another is driven by a *scheduler*, also called a *daemon*. If one or more processes are enabled, the daemon *selects* at least one of these enabled processes to execute an action. We assume that the daemon is also *weakly fair*, meaning that, if a process P is continuously enabled, P must eventually be selected by the daemon.

We say that a process P is *neutralized* in the computation step $\gamma_i \mapsto \gamma_{i+1}$ if P is enabled in γ_i and not enabled in γ_{i+1}, but does not execute any action between these two configurations. The neutralization of a process represents the following situation: at least one neighbor of P changes its state between γ_i and γ_{i+1}, and this change effectively makes the guard of all actions of P false.

We use the notion of *round* [5], which captures the speed of the slowest process in an execution. We say that a finite execution $\varrho = \gamma_i \mapsto \gamma_{i+1} \mapsto \ldots \mapsto \gamma_j$ is a *round* if the following two conditions hold:

1. Every process P that is enabled at γ_i either executes or becomes neutralized during some step of ϱ.
2. The execution $\gamma_i \mapsto \ldots \mapsto \gamma_{j-1}$ does not satisfy condition 1.

We define the *round complexity* of an execution to be the number of disjoint rounds in the execution, possibly plus 1 if there are some steps left over.

2.1 Self-Stabilization and Silence

The concept of *self-stabilization* was introduced by Dijkstra [4]. Informally, we say that distributed algorithm is *self-stabilizing* if, starting from a completely arbitrary configuration, the network will eventually reach a legitimate configuration.

More formally, we assume that we are given a *legitimacy predicate* $\mathcal{L}_{\mathcal{A}}$ on configurations. Let $\mathbb{L}_{\mathcal{A}}$ be the set of all *legitimate* configurations, *i.e.,* configurations which satisfy $\mathcal{L}_{\mathcal{A}}$. Then we define $\mathcal{A}$ to be *self-stabilizing* if the following two conditions hold:

1. (Convergence) Every maximal execution contains some member of $\mathbb{L}_{\mathcal{A}}$.
2. (Closure) If an execution e begins at a member of $\mathbb{L}_{\mathcal{A}}$, then all configurations of e are members of $\mathbb{L}_{\mathcal{A}}$.

We say that $\mathcal{A}$ is *silent* if every execution is finite. In other words, starting from an arbitrary configuration, the network will eventually reach a configuration where no process is enabled.

3 The Leader Election Algorithm SSLE

In this section, we present a silent self-stabilizing algorithm, SSLE, that elects the process of minimum ID in the network to be the leader, within $O(n)$ rounds of arbitrary initialization, using $O(\log n)$ space per process.

3.1 A Simplified Algorithm

We first describe a simplified algorithm for the leader election problem. let *Leader* be the process of smallest ID in the network. Let $P.leader$ be a process P's current estimate of the ID of *Leader* and $P.level$ be P's current estimate of its distance to *Leader*.

For convenience, write $P.key = (P.leader, P.level)$, the *key* of P. Keys are ordered lexically, *i.e.,* $P.key < Q.key$ if $P.leader < Q.leader$, or $P.leader = Q.leader$ and $P.level = Q.level$. For any P, let $P.self = (P.id, 0)$, which we call the *self key* of P. $Succ(i, j) = (i, j + 1)$ for any ordered pair (i, j). Let

$Min\_Key\_Nbr(P)$ to be the minimum value of $Q.key$ among all $Q \in \mathcal{N}_P$, where $\mathcal{N}_P$ is the set of neighbors of P.

When the simplified algorithm converges, the following conditions will hold:

C1. $P.key \leq (P.id, 0)$

C2. If $P.key > Min\_Key\_Nbr(P)$,
then $P.key = Succ(Min\_Key\_Nbr(P))$,
else $P.key = (P.id, 0)$.

It follows easily that, if these conditions hold, $P.leader = Leader.id$ for all P, and $P.level$ will be the distance from P to $Leader$, and hence each process is connected to the $Leader$ by the shortest possible path.

Our simplified algorithm has only two actions, as follows:

A1. If $(P.key > P.self) \vee (P.key \leq Min\_Key\_Nbr(P))$,
then $P.key \leftarrow P.self$.
A2. If $Succ(Min\_Key\_Nbr(P)) < P.key \leq P.self$,
then $P.key \leftarrow Succ(Min\_Key\_Nbr(P))$.

If, initially, $P.leader \geq Leader.id$ for all P, the simplified algorithm converges within $diam + 1$ rounds. In this case, $Leader.self = (Leader.id, 0)$ is the smallest possible key. After one round, $Leader.key = Leader.self$, and after $t + 1$ rounds, all processes within distance t of $Leader$ have their final keys.

3.2 The Problem of Fictitious Leaders

The simplified algorithm in Section 3.1 is not self-stabilizing, since because of arbitrary initialization, $P.leader$ could be initialized to a value of ID type which is not the ID of any process in the network. In this case we say that P has a *fictitious leader*. A fictitious leader that is greater than $Leader.id$ is not a problem, but if a fictitious leader is less than $Leader.id$, the network might never get rid of that fictitious ID. We illustrate this possibility with a simple example.

Consider a 2-process network with processes, P_2 and P_3, where $P_i.id = i$, and where initially $P_2.key = (1, 0)$ and $P_3.key = P_3.self = (3, 0)$. Suppose each process executes one action during each round. After one round, $P_2.key = (2, 0)$ and $P_3.key = (1, 1)$. After another round, $P_2.key = (1, 2)$ and $P_3.key = (3, 0)$. After a total of $2t$ rounds, $P_2.key = (1, 2t)$, and $P_3.key = (3, 0)$. Thus, the algorithm never stabilizes.

Using a known upper bound on the diameter. The problem of fictitious leaders can be solved if an upper bound, D, on the diameter of the network is given. Simply replace A1 by A1′:

A1′. If $(P.key > P.self) \vee (P.key \leq Min\_Key\_Nbr(P)) \vee (P.level \geq D)$,
then $P.key \leftarrow P.self$.

By induction, it can be shown that if t rounds have elapsed since initialization, and if a process P has a fictitious leader, then $P.level \geq t$. Thus, after $D+1$ rounds have elapsed, there will be no fictitious leader in the network. After at most *diam* additional rounds, the algorithm converges. This method is similar to the Arora and Gouda's algorithm [2].

3.3 Formal Definition of SSLE

SSLE solves the fictitious leader problem by introducing *color waves*.

In SSLE, each process P has the following variables.

- $P.parent \in \mathcal{N}_P \cup \{P\}$, the *parent* of P.
- $P.key = (P.leader, P.level)$, the *key* of P, where $P.leader$ is of ID type, and $P.level$ is a non-negative integer.
- $P.color \in \{0, 1\}$.
- $P.done$, Boolean.

We also define the following functions on keys:

- $Succ(i, j) = (i, j + 1)$
- $(i, j) < (k, \ell) \equiv (i < k) \ \vee \ ((i = k) \wedge (j < \ell))$, *i.e.,* lexical order on keys.

Each process P has the following functions, which can be evaluated by P.

- $Is\_True\_Root(P) \equiv (P.parent = P) \wedge (P.key = (P.id, 0))$, P is a *true root*.
- $Is\_True\_Chld(P) \equiv (P.key = Succ(P.parent.key)) \wedge (P.leader < P.id)$, P is a *true child*.
- $Is\_False\_Root(P) \equiv \neg Is\_True\_Root(P) \wedge \neg Is\_True\_Chld(P)$, P is a *false root*.
- $Is\_Root(P) \equiv Is\_True\_Root(P) \vee Is\_False\_Root(P)$, P is a *root*.
- $Min\_Key\_Nbr(P) = \min\{Q.id : Q \in \mathcal{N}_P\}$, the minimum key of any neighbor.
- $Can\_Improve(P) \equiv Succ(Min\_Key\_Nbr(P)) < P.key$, there is a neighbor of P that would be a better parent than its current parent.
- $Can\_Attach(P) \equiv \exists Q \in \mathcal{N}_P : (Q.key = Min\_Key\_Nbr(P)) \wedge (Q.color = 1)$, there is a process that P can attach to that is better than its current parent.
- $Best\_Nbr(P)$ = a neighbor $Q \in \mathcal{N}_P$ such that $Q.key = Min\_Key\_Nbr(P)$ and $Q.color = 1$. In case there is more than one choice, pick the one of lowest ID. In case there is none, $Best\_Nbr(P)$ is undefined.
- $Chldrn(P) = \{Q \in \mathcal{N}_P : (Q.parent = P) \wedge Is\_True\_Chld(Q)\}$, the *true children* of P.
- $False\_Chldrn(P) = \{Q \in \mathcal{N}_P : (Q.parent = P) \wedge (Is\_False\_Root(Q))\}$, the *false children* of P.

- $Done(P) \equiv (\forall Q \in \mathcal{N}_P : Q.key \leq Succ(P.key)) \wedge$
  $(\forall Q \in Chldrn(P) : Q.done)$

Table 1: Actions of SSLE

A1 priority 1	Attach	$Is\_True\_Root(P)$ $Can\_Attach(P)$ $False\_Chldrn(P) = \emptyset$	$\longrightarrow$	$P.parent \leftarrow Best\_Nbr(P)$ $P.key \leftarrow Succ(Best\_Nbr(P))$ $P.color \leftarrow 0$ $P.done \leftarrow Done(P)$
A2 priority 1	Reset False Root	$Is\_False\_Root(P)$	$\longrightarrow$	$P.key \leftarrow (P.id, 0)$ $P.parent \leftarrow P$ $P.color \leftarrow 0$ $P.done \leftarrow Done(P)$
A3 priority 1	Detach True Child	$Is\_True\_Chld(P)$ $Can\_Improve(P)$	$\longrightarrow$	$P.key \leftarrow (P.id, 0)$ $P.parent \leftarrow P$ $P.color \leftarrow 0$ $P.done \leftarrow Done(P)$
A4 priority 2	Color 1	$P.color = 0$ $P.parent.color = 0$ $\forall Q \in Chldrn(P) : Q.color = 1$ $\neg Is\_True\_Root(P) \vee \neg P.done$	$\longrightarrow$	$P.color \leftarrow 1$ $P.done \leftarrow Done(P)$
A5 priority 2	Color 0	$P.color = 1$ $P.parent.color = 1$ $\forall Q \in Chldrn(P) : Q.color = 0$ $\neg Is\_True\_Root(P) \vee \neg P.done$ $\forall Q \in \mathcal{N}_P : Q.key \leq Succ(P.key)$	$\longrightarrow$	$P.color \leftarrow 0$ $P.done \leftarrow Done(P)$
A6 priority 3	Update Done	$P.done \not\equiv Done(P)$	$\longrightarrow$	$P.done \leftarrow Done(P)$

We give the table of actions of SSLE in Table 1. The name of each action is listed in the first column, along with its priority number. The guard of each action is the conjunction of up to four *clauses*, listed in the third column. In order for an action to be enabled, its guard must be true, and no action with a lower priority number may be enabled.

We refer to Actions A2 and A3 as *reset actions*. We refer to Actions A1, A2, and A3 as *structure actions*. We refer to Actions A4 and A5 as *color actions*.

3.4 Overview of SSLE

The correct value of $P.key$, and the value it will achieve eventually if the algorithm is correct, is $P.final\_key = (Leader.id, Level(P))$, where $Level(P)$ is the

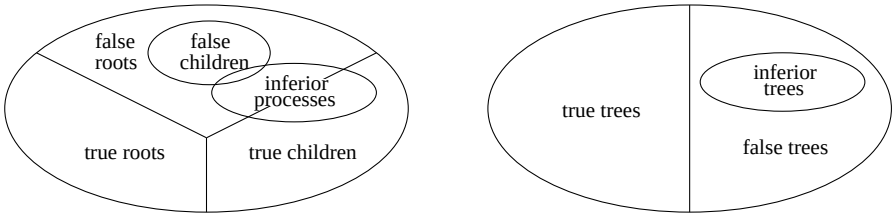

Fig. 1. Relations Among Classes of Processes and Trees

distance from P to *Leader*. If $P.key < P.final\_key$, we say that P is *inferior*. We define an *inferior tree* to be a tree whose root is inferior. All inferior processes belong to inferior trees, and all inferior trees are false trees. The relations between the various sets of processes and trees are indicated in Figure 1.

As the algorithm progresses, processes leave trees and join other trees. When SSLE has stabilized, all processes belong to one true tree rooted at *Leader*. A process can easily detect that it is a false root, but a process that is not a root has no way of knowing whether it is a member of a false tree. The problem we face is that an inferior tree can continue to recruit new leaves, even as it deletes itself starting from the root, and might never disappear.

3.5 Color Waves and Energy

Afek and Bremler solve the fictitious leader problem in their message-passing leader election algorithm [1], by using the concept of "power supply," the idea being that a true root continuously supplies "power" to its tree, allowing it to recruit new processes, whereas false trees will eventually run out of "power" and be unable to recruit. In this paper, we introduce a similar concept. Each process P has a *color*, either 0 or 1. Only processes of color 1 are allowed to recruit new members of the tree, and the new recruits always have color 0. In addition, we allow a process P to change color if $P.parent.color = P.color$, and if all its true children have the opposite color. Processes change colors in convergecast waves starting from the leaves of the trees.

A true root "absorbs" the color waves by alternating its own color, but a false root cannot change color. Thus, in a false tree, color waves, which cannot pass each other, eventually cause *color deadlock*, preventing further growth of the tree.

At the same time a false root is enabled to reset (execute Action A2). Thus, a false tree shrinks every round, but is limited in its growth. Deletion of a false root can break the remainder of its tree into multiple smaller false trees, all with the same leader.

In order to prove that, eventually, all inferior trees will be eliminated, we define the *energy* of a tree, and show that the maximum energy of any false tree decreases every round. For any process P, let

$$\beta(P) = \begin{cases} 1 & \text{if } \mathit{Is\_Root}(P) \wedge (P.\mathit{color} = 0) \\ 2 & \text{if } \mathit{Is\_Root}(P) \wedge (P.\mathit{color} = 1) \\ \beta(P.\mathit{parent}) & \text{if } \mathit{Is\_True\_Chld}(P) \wedge (P.\mathit{color} = 0) \wedge \\ & \quad (P.\mathit{parent}.\mathit{color} = 1) \\ \beta(P.\mathit{parent}) + 2 & \text{otherwise} \end{cases}$$

We define the *energy* of a tree to be the maximum value of $\beta(P)$ for any process P in that tree, and we define B to be the maximum value of the energy of any inferior tree. The energy of a tree can increase in only one way, and that is by its root executing a color action. Thus a true tree can increase its energy, but a false tree, such as an inferior tree, although it can recruit members, cannot increase its energy. Furthermore, the energy of any tree decreases if its root leaves the tree. Thus, since every false root is enabled to execute Action A2, its energy decreases every round. Finally, since no new inferior trees can be created, except by fragmentation of an existing inferior tree, the value of B decreases every round.

Since the energy of any tree cannot exceed $2\mathit{simp}+2$, the time required for all inferior trees to be deleted is $O(\mathit{simp})$. Once there are no more inferior processes in the network, SSLE will stabilize within $O(\mathit{simp})$ additional rounds. There will then be just one tree $\mathcal{T}$, rooted at *Leader*, which will be a breadth-first-search spanning tree of the network.

After it has stabilized, SSLE may not yet be silent, since Actions A4, A5, and A6 may continue to execute. In a convergecast wave starting at the leaves of $\mathcal{T}$, $P.\mathit{done}$ will be set to true for all P. When $\mathit{Leader}.\mathit{done}$ holds, it has received acknowledgment from all other processes that it has been elected leader, and it ceases to change color, because of the fourth clause of the guard of each color action. Within $O(\mathit{diam})$ additional rounds, all other nodes stop changing color as well.

Due to arbitrary initialization, $\mathit{Leader}.\mathit{done}$ could be true even if the algorithm is not finished. In this case, within $O(\mathit{diam})$ time, $\mathit{Leader}.\mathit{done}$ will be set to false, and SSLE will proceed normally.

3.6 Example Execution

In Figure 2, we show the sequence of configurations for an execution of SSLE in one example, where the network consists of six processes in a chain. The IDs of the processes are shown across the top of the figure. Each row shows one configuration. Each process is represented by a box containing three numbers. The leftmost number in the box representing a process P is $P.\mathit{leader}$, the middle number is $P.\mathit{level}$, and the rightmost number is $P.\mathit{color}$. Arrows represent parent pointers. If no arrow is shown from the box representing P, then $P.\mathit{parent} = P$. In this example, *Leader* is the fifth node in the chain, and $\mathit{Leader}.\mathit{id} = 2$. We assume that $P.\mathit{done}$ is initially false for all P.

In our example computation, the initial configuration contains one inferior tree consisting of the first four processes. The other two processes form singleton

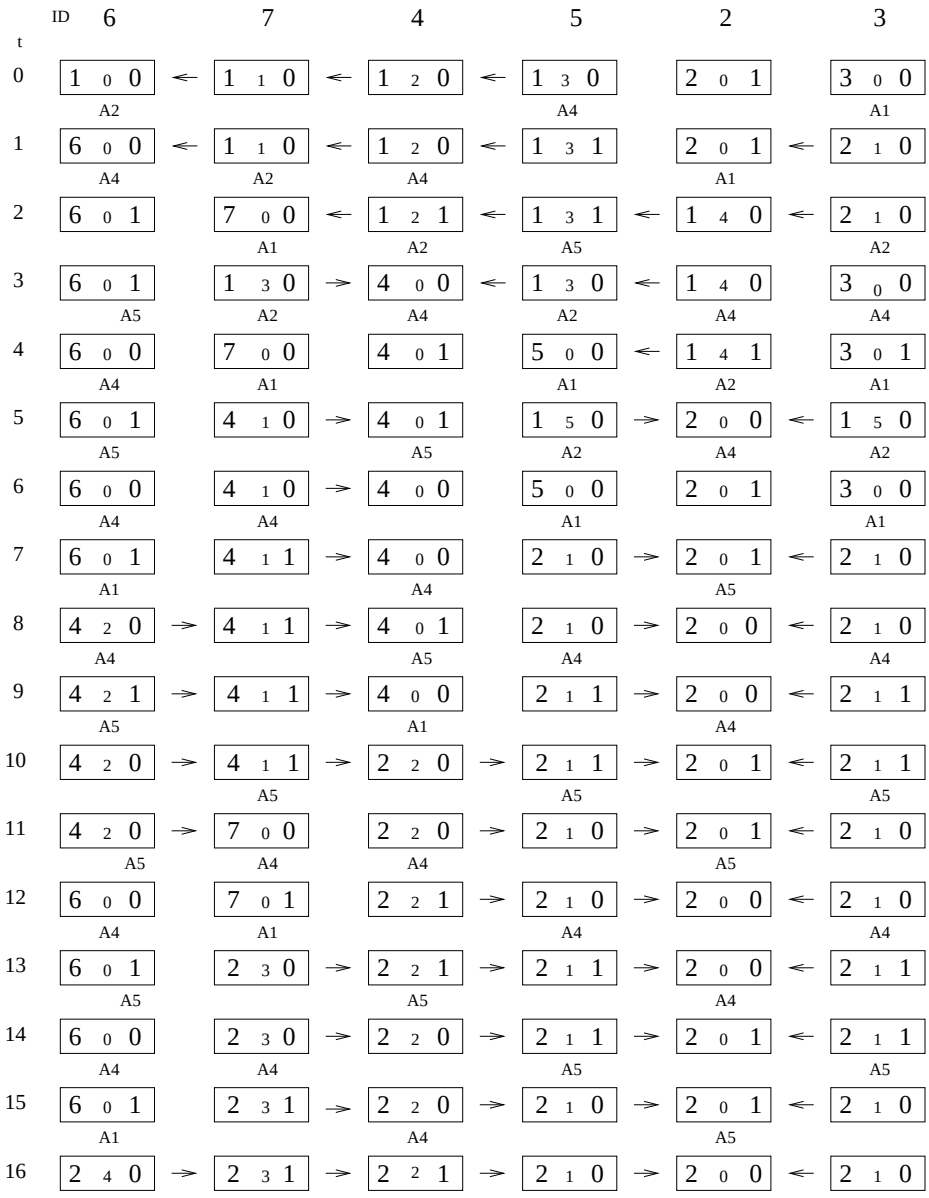

Fig. 2. Example Computation of SSLE **on a Chain of Six Processes**

trees. For simplicity, we will assume that all enabled processes are selected at each step; thus, each round consists of one step. When a process executes an action, the name of that action is shown. For example, during the eighth step, the first process, whose ID is 6, executes Action A1 to join the tree whose leader is 4; changing its *key* from $(6, 0)$ to $(4, 2)$ and changing its *color* from 1 to 0. We do not show Action A6 in the figure, nor do we show the values of $P_i.done$.

As the inferior tree grows to the right, it captures the rightmost two nodes, but also shrinks on the left as its processes executes A2. After six steps, the inferior tree is gone. The tree rooted at *Leader* then grows until it captures all processes after 16 steps. All processes have now chosen 2 as the leader ID, their choices will not change, and SSLE has stabilized.

We encourage the reader to verify that, in this example, $B = 7$ initially, then drops to 5 in the first round, then to 4, then 3, then 2. $B = 1$ after five rounds, and $B = 0$ thenceforth.

Although we only show the first 16 steps, we remark that $Leader.done$ is true after 21 steps. All actions will cease after 25 steps.

4 Proof of SSLE

A *legitimate configuration* for SSLE is a configuration where the following conditions hold.

1. All processes belong to a true tree rooted at *Leader*.
2. If P is any process, then $P.level$ is equal to the length of the shortest path from P to *Leader*.

Recall that $simp \leq n-1$ is the length of the longest simple path in the network. Our main result follows.

Theorem 1. *From arbitrary configuration,* SSLE *is self-stabilizing and silent within* $O(simp)$ *rounds.*

In this section, we sketch the proof of Theorem 1. The proof sketches are intuitive, and only touch lightly on the finer technical details. The complete proof will be given in the full paper.

4.1 Additional Notation

- $||P, Q||$ = the length of the shortest path from process P to process Q.
- $Level(P) = ||P, Leader||$.
- $\mathcal{T}_P$ = the subtree rooted at P of the tree that contains P.

4.2 Elimination of Inferior Processes

Recall that B is the maximum energy of any inferior tree, if there is any; otherwise $B = 0$. By definition of β, we have $B \leq 2simp + 2$. We will show that B decreases during every round. Thus, there will be no inferior trees and hence no inferior processes after $2simp + 2$ rounds have elapsed from initialization.

In the statements and proofs of Lemmas 1 and 2, we will consider just one given step of the execution, $\gamma_{t-1} \mapsto \gamma_t$.

Lemma 1. *If R is a false root at γ_{t-1} and also at γ_t, then the energy of $\mathcal{T}_R$ does not increase during the step.*

Proof. (Sketch.) Since R cannot execute a color action, $\beta(R)$ cannot change. By induction on the length of the parental path from P to R, we can show that, if $P \in \mathcal{T}_R$ both before and after the step, $\beta(P)$ cannot increase. In particular, if $P.color$ changes from 1 to 0, $\beta(P)$ decreases by 2, while $\beta(P)$ is unchanged in all other cases.

Suppose, on the other hand, that P joins $\mathcal{T}_R$ during the step, by attaching to a process $Q \in \mathcal{T}_R$. Then $Q.color = 1$ both before and after the step, and $P.color = 0$ after the step. $\beta(P) = \beta(Q)$ after the step, and $\beta(Q)$ does not change.

Lemma 2. *If R is a false root and $S \in \mathcal{T}_R$ at γ_{t-1}, where $S \neq R$, and if S is a false root at γ_t, then the energy of $\mathcal{T}_S$ at γ_t is less than the energy of $\mathcal{T}_R$ at γ_{t-1}.*

Proof. (Sketch.) During the step, $S.parent$ leaves the tree, making S a root. By induction on the length of the parental chain from S to R, we can prove that $\beta(S)$ at γ_t is less then $\beta(R)$ at γ_{t-1}. Each step of the induction requires examining several cases, depending on the colors of the processes both before and after the step.

The rest of the proof is similar to that of Lemma 1.

Lemma 3. *If $B > 0$, then B decreases during the next round.*

Proof. (Sketch.) By Lemmas 1 and 2, B cannot increase at any step. Since any inferior root is a false root, and every false root is enabled to execute Action A2, every inferior root will reset during the round. By Lemma 2, B will decrease.

Lemma 4. *After $2simp + 2$ rounds have elapsed from initialization, there are no inferior processes.*

Proof. (Sketch.) By the definition of β, $B \leq 2simp + 2$. By Lemma 3, $B = 0$ after $2simp + 2$ rounds. Since every inferior process must belong to an inferior tree, we are done.

4.3 Convergence After Elimination of Inferior Processes

After there are no more inferior processes, *Leader* is a true root within at most one more round, after which *Leader* remains a true root. SSLE then stabilizes within $O(simp)$ additional rounds, as we shall explain in this section.

Although it appears to be intuitively obvious that SSLE will stabilize, we have failed to find a simple proof. Our proof, which will appear in the journal version, uses a complex potential argument.

The complexity of our argument is caused by the fact that only processes whose color is 1 can recruit, and thus recruitment of processes by $\mathcal{T}_{Leader}$ can be delayed if processes are forced to wait to change color. This delay has two rather different causes, making it difficult to obtain a proper potential to measure the maximum number of rounds needed to stabilize.

One source of the delay is *color deadlock*, which we have already discussed. If the sequence of colors of a parental chain in $\mathcal{T}_{Leader}$ is of the form $(01)^*$, *i.e.,* maximally impacted color waves, it is color deadlocked except at the root end. The "traffic jam" is slowly cleared out as *Leader* absorbs the waves by alternating its own color.

Much worse is the delay caused if all processes have color 0. In this case, none of the processes in the tree can recruit. A color wave can only start at the leaves of the tree, which can be very far from the root, and no process in the tree can recruit until that color wave reaches it.

We say that a process P is *exact* if $P.key = P.final\_key$. At some step within $O(simp)$ rounds of initialization, every exact process which is a member of $\mathcal{T}_{Leader}$ will have color 1. Neighbors of those processes of color 1 will attach to them by executing structure actions. Within $O(diam)$ rounds after every exact process in $\mathcal{T}_{Leader}$ has had a chance to have color 1, all processes will join $\mathcal{T}_{Leader}$ and become exact.

Potentials The arguments used to prove convergence make use of a potential Σ, whose definition is quite complex.

Let:

$$\mathcal{T} = \mathcal{T}_{Leader}$$

$$\mathcal{T}^\star = \{P \in \mathcal{T} : P.level = Level(P)\}$$

$$\mathcal{T}[1] = \{P \in \mathcal{T} : P.color = 1\}$$

$$\theta(P) = \begin{cases} -\infty & \text{if } P \notin \mathcal{T} \\ 0 & \text{if } P = Leader \\ \theta(P.parent) + 2 & \text{if } P \in \mathcal{T}, P \neq Leader, \\ & \quad \text{and } P.color = P.parent.color \\ \theta(P.parent) - 2 & \text{if } P \in \mathcal{T}, P \neq Leader, \\ & \quad \text{and } P.color \neq P.parent.color \end{cases}$$

$$\epsilon(P,Q) = \begin{cases} \theta(Q)+1 & \text{if } Q \in \mathcal{N}_P \cap \mathcal{T}[1] \text{ and } \mathit{Succ}(Q.\mathit{key}) < P.\mathit{key} \\ & \quad \text{and } \mathit{Is\_True\_Root}(P) \text{ and } \mathit{False\_Chldrn}(P) = \emptyset \\ -\infty & \text{otherwise} \end{cases}$$

$$\epsilon(P) = \max_Q \{\epsilon(P,Q)\}$$

$$\zeta(P,Q) = \begin{cases} \theta(Q)+2 & \text{if } \neg \mathit{Is\_True\_Root}(P) \text{ and } Q \in \mathcal{N}_{P.\mathit{parent}} \cap \mathcal{T}[1] \\ & \quad \text{and } \mathit{Is\_True\_Root}(P.\mathit{parent}) \\ & \quad \text{and } \mathit{Succ}(Q.\mathit{key}) < P.\mathit{parent}.\mathit{key} \\ -\infty & \text{otherwise} \end{cases}$$

$$\zeta(P) = \max_Q \{\zeta(P,Q)\}$$

$$\eta(P,Q) = \begin{cases} \theta(Q)+3 & \text{if } Q \in \mathcal{N}_P \cap \mathcal{T}[1] \text{ and } \neg \mathit{Is\_True\_Root}(P) \\ & \quad \text{and } \mathit{Succ}(Q.\mathit{key}) < P.\mathit{key} \\ -\infty & \text{otherwise} \end{cases}$$

$$\eta(P) = \max_Q \{\eta(P,Q)\}$$

$$\sigma(P) = \max\{\theta(P), \epsilon(P), \zeta(P), \eta(P)\}$$

$$\Sigma = \max\{\sigma(P)\}$$

Note that Σ depends only on the configuration. Chasing definitions, it is fairly easy to verify that $0 \leq \Sigma < 2\mathit{simp} - 1$.

Let γ^* be the first configuration in the execution at which *Leader* is a true root and there are no inferior processes. Let Σ^* be the value of Σ at that configuration.

We omit the proof of the following lemma, which is very technical and several pages long.

Lemma 5. *If Leader is a true root and there are no inferior processes, Then, for any integer $c > 0$, Leader will execute a color action at least c times during the next $\Sigma + 5c - 4$ rounds, provided Leader.done is false during those rounds.*

The color potential. We define a function τ on $\mathcal{T}$, which we call the *color potential*, as follows:

- $\tau(\mathit{Leader})$ = the number of times *Leader* has executed a color action since γ^*.
- If $P \in \mathcal{T}$ and $P \neq \mathit{Leader}$, then

$$\tau(P) = \begin{cases} \tau(P.\mathit{parent}) & \text{if } P.\mathit{color} = P.\mathit{parent}.\mathit{color} \\ \tau(P.\mathit{parent}) + 1 & \text{if } P.\mathit{color} \neq P.\mathit{parent}.\mathit{color} \end{cases}$$

Lemma 6. *Suppose the configuration is good, $P \in \mathcal{T}$, and P remains in $\mathcal{T}$ after the next step. Then, during that step, $\tau(P)$ increases by 1 if P executes a color action, and is unchanged otherwise.*

Proof. By induction on $P.level$. If $P.level = 0$, then $P = Leader$, and we are done by definition of τ. Otherwise, let $Q = P.parent$. Suppose $P.color$ changes. By the guards of the color actions, $P.color = Q.color$ and hence $\tau(P) = \tau(Q)$ before the step, and Q cannot execute a color action during that step. By the inductive hypothesis, $\tau(Q)$ does not change; thus $\tau(P)$ increases by 1, by the definition of τ.

Suppose $Q.color$ changes. By the guards of the color actions, $P.color \neq Q.color$ and hence $\tau(P) = \tau(Q) + 1$ before the step, and P cannot execute a color action during that step. By the inductive hypothesis, $\tau(Q)$ increases by 1; thus $\tau(P) = \tau(Q)$ after the step, by the definition of τ, and hence is unchanged.

Suppose neither P nor Q executes a color action. By the inductive hypothesis, $Q.color$ remains unchanged, and thus $\tau(P)$ remains unchanged, by the definition of τ.

Lemma 7. *Eventually, $\mathcal{T}^\star$ contains every process.*

Proof. By induction on $Level(P)$. $Leader \in \mathcal{T}^\star$ within $2simp + 2$ rounds. Within $simp$ additional rounds, $Leader.done$ is false. Suppose $P \neq Leader$. Pick $Q \in \mathcal{N}_P$ such that $Level(Q) = Level(P) - 1$. By the inductive hypothesis, $Q \in \mathcal{T}^\star$ eventually. By Lemma 5, P will eventually execute Action A5, which implies that $P \in \mathcal{T}^\star$ at that time.

Lemma 8. *Let γ_P be the first good configuration where $P \in \mathcal{T}^\star$. Then $\tau(P) \leq 3Level(P)$ at γ_P.*

Proof. By induction on $Level(P)$. If $Level(P) = 0$, then $P = Leader$, $\gamma_P = \gamma^*$, and we are done, by definition of τ.

Suppose $Level(P) = L \geq 1$. Assume that $\tau(P) > 3L$ at γ_P. Let $Q = P.parent$. Then, $\tau(Q) \geq 3L$ and $Q \in \mathcal{T}^\star$. By the inductive hypothesis, $\tau(Q)$ was at most $3L - 3$ at the configuration γ_Q. After Q changes color two more times, P must have joined $\mathcal{T}^\star$, and $\tau(Q) \leq 3L - 1$ by Lemma 6. Thus, $\tau(P) \leq 3L$ at γ_P, contradiction.

Let B_0 be the value of B at initialization.

Lemma 9. SSLE *stabilizes within* $\mathrm{B}_0 + \Sigma^* + 15(diam)$ *rounds of arbitrary initialization.*

Proof. Let P be any process, and let $L = Level(P)$. By Lemma 3, the configuration γ^* is reached within $\mathrm{B}_0 + 1$ of initialization. Let γ_P be the first configuration after γ^* at which $P \in \mathcal{T}^\star$. By Lemma 8, $\tau(P) \leq 3L$ at γ_P.

Let γ' be the configuration $\mathrm{B}_0 + simp + \Sigma^* + 15L$ rounds after initialization. By Lemma 5, $\tau(Leader) \geq 3L + 1$ at γ'.

Suppose that γ_P occurs after γ'. Then $\tau(P) \geq \tau(Leader) \geq 3L + 1$ at γ_P, contradiction. Since $L \leq diam$, our result follows.

Lemma 10. SSLE *is silent within* $B_0 + simp + \Sigma^* + 18(diam) + 2$ *rounds.*

Proof. Let $L = \max\{Level(P)\}$. By Lemma 9, SSLE stabilizes within $B_0 + simp + \Sigma^* + 15(diam)$ rounds. Within L rounds additional rounds, $Leader.done$ holds. Let $\Theta = \max\{\theta(P) : \theta(P) + 2Level(P) > 0\}$, with the default value $\Theta = -L$ if $\theta(P) + 2Level(P) = 0$ for all P. Let $\Delta = \frac{1}{2}\Theta + L$, which is an integer since θ is even. $\Delta \leq 2L$, and as long as $\Delta > 0$, it must decrease by at least 1 every round, since every process P where $\theta(P) + 2Level(P) > 0$ and $\theta(P) = \Theta$ must execute a color action. When $\Delta = 0$, no further actions can be executed.

Our main result, Theorem 1, follows immediately from Lemma 10.

5 Conclusion

We present a silent self-stabilizing asynchronous distributed algorithm, SSLE, for election of a leader of a network, where processes have unique IDs. The algorithm stabilizes in $O(n)$ rounds, using $O(\log n)$ space per process, and becomes silent after an additional $O(diam)$ rounds, under the weakly fair daemon.

SSLE is also silent and self-stabilizing under the *unfair daemon*. The proof will be given in the journal version.

References

1. Afek, Y., Bremler, A., Self-Stabilizing Unidirectional Network Algorithms by Power-Supply. In 8th Annual ACM Symposium on Discrete Algorithms, 111–120 (1997)
2. Arora, A., Gouda, M.G., Distributed Reset. IEEE Transactions on Computers, 43, 1026–1038 (1994)
3. Awerbuch, B., Kutten, S., Mansour, Y., Patt-Shamir, B., Varghese, G., Time Optimal Self-stabilizing Synchronization. In 25th Annual ACM Symposium on Theory of Computing, 652–661 (1993)
4. Dijkstra, E.W., Self-stabilizing Systems in Spite of Distributed Control. Communications of the Association for Computing Machinery, 17, 643–644 (1974)
5. Dolev, S., Self-Stabilization. The MIT Press, Cambridge (2000)
6. Dolev, S., Herman, T., Superstabilizing Protocols for Dynamic Distributed Systems. Chicago J. Theor. Comput. Sci., 1997-4, 1–40 (1997)