

Memory space requirements for self-stabilizing leader election protocols

Joffroy Beauquier, Maria Gradinariu, Colette Johnen

L.R.I./C.N.R.S., Université de Paris-Sud,
bat 490, 91405 Orsay Cedex, France
jb,mariag,colette@lri.fr

Abstract

We study the memory requirements of self-stabilizing leader election (SSLE) protocols. We are mainly interested in two types of systems: anonymous systems and id-based systems. We consider two classes of protocols: deterministic ones and randomized ones.

We prove that a non-constant lower bound on the memory space is required by a SSLE protocol on unidirectional, anonymous rings (even if the protocol is randomized).

We show that, if there is a deterministic protocol solving a problem on id-based systems where the processor memory space is constant and the id-values are not bounded then there is a deterministic protocol on anonymous systems using constant memory space that solves the same problem. Thus impossibility results on anonymous rings (i.e. one may design a deterministic SSLE protocol, only on prime size rings, under a centralized daemon) can be extended to those kinds of id-based rings. Nevertheless, it is possible to design a silent and deterministic SSLE protocol requiring constant memory space on unidirectional, id-based rings where the id-values are bounded. We present such a protocol.

We also present a randomized SSLE protocol and a token circulation protocol under an unfair, distributed daemon on anonymous and unidirectional rings of any size. We give a lower bound on memory space requirement proving that these protocols are space optimal. The memory space required is constant on average.

Keyword: self-stabilization, leader election, mutual exclusion, decidability, memory space requirement.

1 Introduction

A main requirement for the design of distributed systems is transparency. Roughly speaking, transparency means that an users do not see the network and they interact with the system as if it was a reliable centralized uni-processor system. Transparency takes several forms. Here, we are interested in two of them: transparency to dynamic evolutions and transparency to failures. The former means that the user does not see any change when the network topology

evolves, for instance when sites are added or/and other ones disconnected. The latter means that, if a failure appears, the user is not supposed to notice any major change in the quality of service he receives.

The two particular failures that we consider, in this paper, are memory and channel corruptions. The framework that deals with these types of failures is known as self-stabilization [9]. The self-stabilization approach is basically non-masking: after a failure, the system is allowed to temporarily exhibit bad behavior, but it must behave correctly within a short period of time without external intervention.

A protocol needs only *constant space* if the memory space needed at each processor is constant per link. A constant space protocol is transparent to dynamic evolutions: the local protocol implementation on a processor never has to be modified. Assuming that link and processors contains memory space, when links or processors are added to the network, they will bring enough memory space to the protocol. Thus the protocol will work properly for all network sizes. A self-stabilizing protocol that uses constant memory space per link can be implemented by processors, and links that are uniformly manufactured. On the other hand, a protocol whose required memory space depends on the network size is not transparent: more memory will eventually have to be allocated to every network processor if the network keeps growing. This has to be done no matter how much memory space was initially allocated to a processor.

In this paper, we study the possibility of designing self-stabilizing leader election (SSLE) protocols on rings requiring constant space. We are mainly interested in two types of systems: anonymous systems (where processors do not have identifier) and id-based systems. In the latter type of systems, processors have distinct hardware identities that cannot be corrupted. We consider two classes of protocols: deterministic ones and randomized ones.

Deterministic SSLE protocols on anonymous rings. A *centralized daemon* is a scheduler which determines the *single* processor that will perform an action when several processors are enabled. A more powerful daemon is a *distributed* one: it can choose one or more processors at each computation step. On an anonymous ring, a SSLE protocol can be designed only (1) if the ring size is a prime number and if (2) the daemon is centralized. In [16], Itkis, Lin, and Simon presented a deterministic SSLE protocol under a centralized daemon on bidirectional prime-size rings requiring constant memory space. We prove that such a protocol on unidirectional rings requires at least N states per processor (where

N is the ring size).

Deterministic SSLE protocols on id-based networks. In an id-based system, each processor has a distinct hardware identity. The processor ids are stored in a non-volatile read-only memory and thus they cannot be corrupted, unlike processor states which are stored in a regular memory. The arguments used in the case of anonymous rings to prove the impossibility of SSLE protocols do not hold. In fact, there are several deterministic SSLE protocols on id-based systems where the memory space requirement is not constant (i.e. at least N states per processor) [1], [2], [11], and [3]. Roughly, the general idea of these protocols is that each processor keeps in its regular memory the id-value of its candidate for the leadership. Processors exchange id-values of their candidates and they will eventually get an agreement on the leader.

On id-based systems, in case of a constant space protocol, a processor cannot “store” the id-value of its leader candidate. The regular memory cannot contain an id-value. We show that if a deterministic protocol using constant memory space can solve a problem on id-based systems without a bound on the identifier values, then there is a deterministic protocol using constant memory space that solves the same problem, on anonymous systems. That is a surprising result: an id-based system is not more powerful than an anonymous system. The main idea of our proof is that on such a system, each processor is a state machine. Different processors can be represented by different state machines, but the number of different machines is bounded.

Therefore, we prove that impossibility results for SSLE protocols on anonymous systems using constant memory space can be applied to id-based systems without a bound on the identifier values. Thus, we have shown that, even on id-based networks, in most cases, it is not possible to design an algorithm requiring constant memory space.

A self-stabilizing protocol is *silent* if, once the system is stabilized, processors do not change their state. Processors only check that neighbors' states have not been corrupted. The silence property of self-stabilizing protocols is a desirable property in terms of simplicity and communication overhead. In [12], Dolev, Gouda and Schneider have shown that the memory requirement of a silent SSLE leader election protocol is $O(\lg N)$ in the general case. We exhibit a particular case where it is possible to design a silent algorithm requiring constant memory space: id-based rings with bounded id-values. We present an algorithm performing this task. We conjecture that it is the only case where a silent protocol using constant memory space can be designed.

Randomized SSLE protocols on anonymous networks. Finally, we present a space optimal randomized SSLE protocol under an unfair and distributed daemon for anonymous and unidirectional rings of any size. A daemon is *fair* if a processor that is continuously enabled starting some point in a computation will be eventually chosen by the daemon. There is no restriction on the scheduling in the case of an unfair daemon except that it has to choose processors that can perform an action. To design our protocol, we have shown a protocol-compiler that transforms any self-stabilizing protocol on rings under fair daemons into a self-stabilizing protocol under unfair daemons. Our compiler allows us to obtain a space optimal randomized token circulation protocol under unfair daemons. Only one token circulation protocol that self-stabilizes under unfair daemons was previously known

[18]. This protocol requires $O(\lg N)$ bits per processor. The space complexity of our token circulation and leader election protocols are $O(\lg m_N)$ bits per processor where m_N is the smallest integer not dividing N . Notice that the value of m_N is constant on average. For example, on odd size rings, 4 (resp. 2) bits per processor are necessary and sufficient for leader election (resp. token circulation). We prove the optimality of our algorithms.

Israeli and Jalfon [14] proved a logarithmic lower bound on space for self-stabilizing mutual exclusion algorithms in the case of a restrictive model: a processor has a token if and only if it may perform an action. In [4], Awerbuch and Ostrovsky presented a SSLE randomized algorithm on bidirectional networks requiring $(\lg^* N)$ states. Their initial algorithm requires N states, but they proposed a data structure to store distributively the N -size variables. Itlis and Levin presented in the appendix of [15] another data-structure based on a Thue-Morse sequence requiring $O(1)$ bits per edge to store in a distributed manner variable having N possible values. The two last algorithms require bidirectional networks.

If the “deadlock freedom” property (there is no deadlock) is guaranteed externally, a randomized SSLE constant space protocol on rings in the message passing model was presented by Mayer, Ofek, Ostrovsky, and Yung, in [20]. In [6], Beauquier, Cordier and Delaët have presented a token circulation protocol on anonymous and unidirectional rings that self-stabilizes under a particular class of daemons (the memory space required is also $O(\lg m_N)$ bits). A randomized protocol-compiler that transforms a self-stabilizing protocol on bidirectional anonymous rings to a protocol on synchronous unidirectional and anonymous rings, is presented in [21]; the memory space required by the compiler is constant.

The paper is organized as follows. The formal model is described in section 2. We study the SSLE problem on anonymous rings in section 3 and on id-based systems in section 4. In section 5, we consider randomized protocols.

2 Model

Distributed System. A *distributed system* is a connected graph $Sys = (V, E)$ where V is the set of processors ($|V| = N$) and E is the set of communication links. A communication link connects two processors. The links are directed. Processor p can know the state of p' if and only if p' is a neighbor of p . The degree of p (denoted by d_p) is the number of neighbors of p .

System topology. In a *general system*, there is no restriction on the topology of the distributed system and this topology is unknown. In this paper, we are mostly interested in systems having a ring topology. On a *bidirectional ring*, processors can receive information from two processors. On a *unidirectional ring*, a processor can receive information only from one processor, called its predecessor. A processor sends information only to one processor, called its successor.

States and Configurations. A *configuration* of a distributed system $Sys = (V, E)$ is a vector of the states of all processors of Sys . We denote by \mathcal{S} the set of processor states. A *local configuration* is the part of a configuration that can be “seen” by a processor (i.e. its state and the state of its neighbors). A configuration is *symmetrical* if all processors

have the same local configuration. The set of configurations of Sys is denoted by \mathcal{C} .

Actions. Each processor executes a protocol. The protocol consists of a set of variables and a finite set of guarded actions (i.e. guard \rightarrow statement). The guard of an action on p is a boolean expression involving the state of p and the state of its neighbors. The statement of an action updates the state of the processor that performs the action. A processor that has a true guard in a configuration c , is said enabled at c . We assume that the actions are executed atomically.

On unidirectional rings, an action is denoted by $s_0s_1 \rightarrow s_0s_2$ (s_0 is the required state on the predecessor of p , s_1 is the required state on p and s_2 is the new state of p after performing the action).

Computations. During a *computation step*, one or more processors execute one action. A *computation e* of a protocol \mathcal{P} is a sequence of configurations c_1, c_2, \dots such that for $i = 1, 2, \dots$, the configuration c_{i+1} is reached from c_i by one computation step. c_1 is called the *initial configuration* of e . A computation is said to be *maximal* if the sequence is either infinite or it is finite and no processor is enabled in the final configuration. The set of maximal computations of \mathcal{P} in a system Sys whose initial configurations are elements of $B \subset \mathcal{C}$ is denoted \mathcal{E}_B . \mathcal{E} is the set of all possible maximal computations (i.e. $\mathcal{E} = \mathcal{E}_\mathcal{C}$). A problem is a predicate defined on computations.

Daemons. The *central daemon* is a scheduler which determines what single processor makes a step when several processors are enabled. By extension, the *distributed daemon* [7] is a scheduler which chooses an arbitrary subset of the enabled processors. A daemon is *fair* if if a processor that is continuously enabled starting some point in a computation will be eventually chosen by the daemon. There is no restriction on the scheduling in the case of an unfair distributed daemon except that it has to choose enabled processors. A *synchronous daemon* [13] is a scheduler which “chooses” all enabled processors. In this case, the scheduler has no choice. A daemon is *k-bounded* if and only if when a processor p is continuously enabled, any other processor p' performs at most k actions before p performs an action. A bounded daemon is fair, but the converse is not true.

Self-Stabilization. The protocol \mathcal{P} is self-stabilizing for the problem \mathcal{PR} if and only if there exists a predicate \mathcal{L} defined on configurations such that:

- **convergence** All computations reach a configuration that satisfies \mathcal{L} . Formally:

$$\forall e = (c_1, c_2, \dots) \in \mathcal{E}, \exists n \geq 1, c_n \vdash \mathcal{L}$$
- **correctness** All computations, from \mathcal{L} , satisfy the problem \mathcal{PR} . Formally: $\forall e \in \mathcal{E}_\mathcal{L}, e \vdash \mathcal{PR}$

Here $x \vdash \mathcal{P}$ means that x satisfies the predicate \mathcal{P} .

3 Deterministic SSLE protocols on anonymous rings

In this section, we study the problem of deterministic SSLE on anonymous rings. It is known that there is no deterministic leader election protocol under a distributed daemon; therefore we study systems under a centralized daemon. Dijkstra [10] has pointed out, and Burns and Pachl [8] have proven that there is no self-stabilizing protocol that provides

token circulation under a centralized daemon on anonymous and composite rings. A composite ring is a ring whose size is not a prime number. Their arguments allow us to show that there is no SSLE on composite size rings. Only on prime-size rings, under a centralized daemon, one may design a SSLE protocol.

On unidirectional rings, processors have less information about the configuration than on bidirectional rings. Thus it is more difficult to design a protocol for unidirectional rings. Nevertheless, in [8], Burns and Pachl have designed a deterministic self-stabilizing protocol on anonymous and unidirectional prime-size rings requiring $O(N^2/\log N)$ states per processor. This protocol can be easily transformed to a leader election protocol.

The question addressed here is to determine the minimal amount of memory space needed by a deterministic SSLE leader election protocol on unidirectional prime size rings. We are not claiming that this particular question is important because of applications (a prime-size ring is not a realistic assumption). Rather we seek better insight into the nature of self-stabilization on unidirectional networks.

Notation 1 N -ring is a ring of size N .

Lemma 1 Let \mathcal{P} be a deterministic SSLE protocol requiring only B_N processor states on an anonymous and unidirectional N -ring. There is an integer $k \leq B_N$ and a finite sequence of processor states s_0, s_1, \dots, s_{k-1} satisfying the following property: for all $0 \leq i < k$, each processor has the action: $s_i s_i \rightarrow s_i s_{i \oplus 1}$ where $i \oplus 1 = (i + 1) \bmod k$.

Proof: Consider any configuration of an N -ring where all processors have the same state s . There is no leader, because the configuration is symmetrical. Thus, there is an action defined as $ss \rightarrow ss'$ where $s \neq s'$.

Starting from any state s_0 , there is an infinite sequence of processor states s_0, s_1, s_2, \dots such that, for all i , there is an action of type $s_i s_i \rightarrow s_i s_{i+1}$. As the number of states is bounded by B_N , there are j, l such that $j < l$, $s_j = s_l$, and $l - j \leq B_N$. Rename the processor states as follows: $\forall j \leq i < l, s_i$ is called σ_{i-j} . Let $k = l - j \leq B_N$. The finite sequence of processor states $\sigma_0, \sigma_1, \dots, \sigma_{k-1}$ has the required property. \square

Theorem 1 The number of states per processor required by a deterministic SSLE protocol under a centralized daemon on unidirectional, prime size and anonymous rings is greater than or equal to the ring size.

Proof: Let us assume there is a SSLE protocol under a centralized daemon on unidirectional, prime size and anonymous rings that uses $|S| < N$ states per processor where N is the ring size.

By lemma 1, there is an integer $k \leq |S| < N$ and a finite sequence of processor states s_0, \dots, s_{k-1} such that, for all $0 \leq i < k$, each processor has the following action: $s_i s_i \rightarrow s_i s_{i \oplus 1}$.

Let c_0 be the configuration defined by $(s_0^{N-k-1}, s_0, s_0, s_1, s_2, \dots, s_{k-1})$ on the anonymous N -ring. Such a configuration can be built because $N \geq k + 1$. Study the computation step (called cs_1) where the daemon picks the last processor in state s_0 . The configuration obtained after this step is: $\sigma_1 = (s_0^{N-k-1}, s_0, s_1, s_1, s_2, \dots, s_{k-1})$. For all $i \leq k$, we call σ_i the configuration defined as $(s_0^{N-k-1}, s_0, s_1, s_2, s_3, \dots, s_i, s_i, \dots, s_{k-1})$. Study the computation step, cs_i ($2 \leq i \leq k$), where from σ_{i-1} , the daemon chooses the second processor in state s_{i-1} , the configuration

obtained after cs_i is σ_i . From c_0 , after the computation steps cs_1, cs_2, \dots , and cs_k , the configuration obtained is $c_1 = (s_0^{N-k-1}, s_0, s_1, s_2, \dots, s_{k-1}, s_0)$. Note that c_1 is the same configuration as c_0 after applying a right-shift of processor states.

Let c_h be the configuration $(s_0^{N-k-h+1}, s_1, s_2, s_3, \dots, s_{k-1}, s_0^h)$ where $h < N - k + 1$. Let c_{N-k+1} be the configuration $(s_1, s_2, \dots, s_{k-1}, s_0^{N-k+1})$. Let c_h be the configuration $(s_{h-N+k}, s_{h-N+k+1}, \dots, s_{k-1}, s_0^{N-k+1}, s_1, \dots, s_{h-N+k-1})$ where $N - k + 1 < h < N$. Define $c_N = c_0$.

For example: $c_2 = (s_0^{N-k-1}, s_1, s_2, \dots, s_{k-1}, s_0^2) = (s_2, s_3, \dots, s_{k-1}, s_0^{N-k+1}, s_1)$, and $c_{N-1} = (s_{k-1}, s_0^{N-k+1}, s_1, \dots, s_{k-2})$.

We have built a computation that reaches c_1 from c_0 ; in the same way, we build a computation that reaches c_{h+1} from c_h where $h < N$. Thus a cyclic and infinite computation, e , is built that goes through the following configurations $(c_0, c_1, \dots, c_{N-1})^*$.

Along e , all processors have the same behavior: they get the same states and they execute the same actions. We have built a maximal computation where no processor is elected. \square

Remark 1 *The following deterministic and silent identity assignment protocol on the unidirectional, anonymous 3-ring under a centralized daemon is self-stabilizing. Each processor has one variable v , taking values in $\{0, 1, 2\}$, and one action: $v, v \rightarrow v, (v + 1) \bmod 3$. This protocol elects a leader (specifically, the processor whose the variable value is 0). The protocol is space optimal using 3 states per processor.*

Remark 2 *Subsequently, Jaap-Henk Hoepman (private communication) has applied this proof technique to self-stabilizing token circulation on anonymous and unidirectional rings. He has obtained a lower bound of $(N - 1)/2$ states per processor.*

4 Deterministic SSLE protocols on id-based systems

In this section, we study the possibility of designing deterministic SSLE protocols on id-based systems using constant memory

Definition 1 *An id-based system is said to have a constant bound on its identifiers if and only if there exists an integer k such that id-values are in the set $\{0, 1, 2, \dots, N + k\}$.*

In the first sub-section, we prove that the results on deterministic protocol on anonymous systems can be applied to deterministic protocol on id-based systems without a bound on the identifier values. In the second sub-section, we present a deterministic SSLE protocol that elects a leader on id-based rings (with a constant bound on its identifier) using constant memory space. The protocol is silent.

4.1 Id-based system versus anonymous system

Theorem 2 *If there is a deterministic protocol on id-based systems using constant space to solve a problem \mathcal{PR} (without a bound on the identifier values), then there is a deterministic protocol on anonymous systems using constant space to solve the problem \mathcal{PR} .*

Proof: The idea of the proof is to build a protocol, named \mathcal{P}_u , on anonymous systems from a protocol, named \mathcal{P}_{id} , on id-based systems

In \mathcal{P}_{id} , processors are deterministic. With the same local configuration, a processor has always the same “behavior”, it always gets the same state, if it is chosen by the daemon. For a processor p with k neighbors, there is a deterministic behavior function. The deterministic behavior function ($bf_{(p,k)}: \mathcal{S}^{k+1} \rightarrow \mathcal{S} \cup \{\perp\}$), given p 's local configuration (p state and neighbors' states), says whether p can perform an action and, if so, gives the new state of p that results from the action. \perp means that p cannot perform any action in that local configuration. A processor p uses the same deterministic behavior function, whatever the current system topology, as long as it keeps the same degree value (k).

As the memory space required by \mathcal{P}_{id} is bounded; the number of processor states is also bounded. Thus, the number of distinct behavior functions for processor having k neighbors is bounded by $(|S| + 1)^{|S|^{k+1}}$.

There is at least one behavior function for processor having k neighbors that is executed by an infinite number of processors, because the number of identifiers is unbounded and the number of behavior functions is bounded. Let us call BF_k one of these behavior functions.

The protocol \mathcal{P}_u is built as follows: each processor having k neighbors executes the deterministic behavior function BF_k .

Whatever the anonymous system Sys_u is, one may build an id-based system (processors having distinct identifiers) Sys_{id} having the same topology as Sys_u and such that all processors in Sys_{id} that have k neighbors execute the deterministic behavior function BF_k . This construction may be done because there is no bound on the identifier values in Sys_{id} .

The set of maximal computations of \mathcal{P}_{id} in Sys_{id} is equal to the set of maximal computations of \mathcal{P}_u in Sys_u . If the protocol \mathcal{P}_{id} is able to solve the problem \mathcal{PR} from the initial configuration c , in Sys_u then \mathcal{P}_u is also able to solve the problem \mathcal{PR} from c , in Sys_u . \square

Corollary 1 *If there is a deterministic self-stabilizing protocol on id-based systems using constant space to solve a problem \mathcal{PR} (without a bound on the identifier values) then there is a deterministic self-stabilizing protocol on anonymous systems using constant space to solve the problem \mathcal{PR} .*

Remark 3 *On id-based rings (without a bound on the identifier values), a deterministic SSLE protocol requiring constant memory space can be designed only if (i) the ring size is a prime number, (ii) the daemon is centralized, and (iii) the ring is bidirectional.*

4.2 Deterministic SSLE protocol on Id-based rings with a bound on identifier values

Theorem 3 *There is a deterministic, silent SSLE protocol using constant memory space on unidirectional, id-based rings with a constant bound on its identifier values.*

Proof: Let $N+k$ be the bound on the id-values (N being the ring size and k being a constant). We present a deterministic, silent and self-stabilizing protocol electing the processor having the smallest id-value. Notice that the smallest id-value in any ring is less than $k + 2$. Thus, only a processor whose id-value is less than $k + 2$ may have the smallest id-value in the ring. We call these processors *potential-leaders*. There are at most $k + 2$ potential-leaders in a ring. The goal of the protocol is that each processor knows the id-values

of the potential-leaders that are in the ring. In that way, processors will agree on the smallest id-value.

A very basic protocol would be that each potential-leader broadcasts its id-value and each processor keeps these id-values in a set (or keeps the smallest id it has seen). In the case of an initial configuration where all sets are empty, all processors will eventually store the id-values of potential-leaders that are actually in the ring. Unfortunately, this basic protocol is not a self-stabilizing protocol. In the initial configuration where all processor sets contain the id-value 0, all processors will agree on the value 0, although it is possible that no processor in the ring has the id-value 0.

In our algorithm, a processor p stores id-values of potential-leaders in a array of $k + 2$ elements (denoted F_p). A potential-leader copies the F array of its predecessor after (i) left-shifts the F array elements (the first value is withdrawn), (ii) adds its own id-value at the end of the array. Eventually, by this action (\mathcal{B}_1), all initial values in the F arrays are removed, and the order of id-values in F_p have a specific meaning: $F_p[k + 2 - i]$ contains the id-value of the i -th previous potential-leader of p in the ring.

A processor that is not a potential-leader only copies the F array of its predecessor (\mathcal{A}_1) and decides that is not the leader (\mathcal{A}_2)

The protocol is silent: once each processor has a correct F array (according to the F array of its predecessor), no one will change its F array. Then, each potential-leader decides if it is the leader or not (\mathcal{B}_2 or \mathcal{B}_3).

When a processor that is not a potential-leader is removed or added to the ring, no F array is changed. When a processor p that is a potential-leader is added to the ring, it adds its id-value in its F array. Then, its successor will update its F array, and eventually every processor will update its F array. When a processor p (a potential-leader) is removed from the ring, its previous successor updates its F array (the F array of its predecessor has changed). Eventually all processors will update their F array and the id-value of p will be removed from all F arrays.

We prove the algorithm by induction (the complete proof of the protocol can be found in [17]). First, we prove that the last value of the F_p array will eventually be correct (i.e. on each potential-leader p , $F_p[k + 2]$ contains the id-value of the previous potential-leader of p in the ring) and then will stay correct on every processor.

We finish the proof by showing that if on any processor p , the l last values of F_p array are and stay correct (i.e. if p is a potential-leader then $\forall 0 \leq i \leq l - 1$, $F_p[k + 2 - i]$ contains the id-value of the i -th previous potential-leader of p in the ring) then the $l + 1$ last values of F_p array will be eventually correct and will stay correct.

By induction, any potential-leader p , F_p array will eventually be correct and will stay correct. Once the F arrays are stabilized, the election is easily completed. \square

5 Randomized SSLE protocols on anonymous rings

The question addressed now, is to determine the minimal memory space needed by a randomized SSLE protocol on unidirectional rings. The proofs about memory space requirement for deterministic SSLE protocols obviously do not hold for randomized SSLE protocols. Exhibiting a particular incorrect behavior does not imply that the set of incorrect behaviors has a non-zero probability. Correctly proving

Leader election algorithm on rings with a constant bound: k

Variables on p :

F_p is an array of $k + 2$ elements taking values in $[0, k + 2]$.
 Ld_p is a boolean.

Notation on p :

lp is the predecessor of p .
 F_{lp} is the value of F on lp .
 id_p is the value of the p identifier.

Predicates on p :

$Next(p, p') \equiv (0 \leq i < k + 1 : F_p[i] = F_{p'}[i + 1]) \wedge (F_p[k + 1] = id_p)$.
 $Following(p) \equiv Next(p, lp)$.

Macro on p :

$Update(p) : 0 \leq i < k + 1 : F_p[i] := F_{lp}[i + 1];$
 $F_p[k + 1] := id_p$.

Actions on p :

$\mathcal{A}_1: (id_p > k + 1) \wedge (F_p \neq F_{lp}) \longrightarrow F_p := F_{lp}$.
 $\mathcal{A}_2: (id_p > k + 1) \wedge (Ld_p = 1) \longrightarrow Ld_p := 0$.

$\mathcal{B}_1: (id_p \leq k + 1) \wedge \neg Following(p) \longrightarrow Update(p)$
 $\mathcal{B}_2: (id_p \leq k + 1) \wedge Following(p) \wedge id_p$ is not the smallest value in $F_p \wedge Ld_p = 1 \longrightarrow Ld_p := 0$
 $\mathcal{B}_3: (id_p \leq k + 1) \wedge Following(p) \wedge id_p$ is the smallest value in $F_p \wedge Ld_p = 0 \longrightarrow Ld_p := 1$

a randomized impossibility result requires the development of a model where a probability distribution on subsets of behaviors has been defined. This is not so simple, because this model has to consider the daemon as an adversary, and clearly separate what is not deterministic (the daemon) and what is randomized (the protocol). There is no place here to develop such a model; that is left to the complete paper. Some useful elements appear in [22] and [5]. Let us just summarize here what is needed for proving such an impossibility result: for an infinite sequence of allowed choices of the daemon, the subset of incorrect computations (according to the specification of the problem) has a strictly positive measure.

First, we need some specific definitions:

Definition 2 *Let s and s' be two processor states. (s, s') is a local deadlock if and only if p cannot perform any action when the state of a processor p is s' and the state of its predecessor is s .*

Notation 2 *Let m_N be the smallest integer that does not divide the ring size N .*

Theorem 4 *The number of states per processor required by a randomized SSLE protocol under an unfair daemon on anonymous and unidirectional rings is greater than $(m_N - 2)/2$.*

Proof: Let \mathcal{P} be a randomized SSLE protocol on anonymous and unidirectional rings. Assume that \mathcal{P} requires $l < (m_N - 2)/2$ processor states on an anonymous and unidirectional N -ring, and that \mathcal{P} can elect a leader under an unfair daemon. Let $S = \{\sigma_1, \sigma_2, \dots, \sigma_l\}$ be the set of processors states. Two cases are possibles:

1. $\forall \sigma \in S, \exists \sigma' \in S$ such that (σ, σ') is a local deadlock.

Then, there is a infinite sequence s_1, s_2, s_3, \dots , such that, $\forall i > 0, (s_i, s_{i+1})$ is a local deadlock. As the size of S is l , there are two integers such that $s_i = s_j$ and $0 < j - i \leq l < (m_N - 2)/2$. $\exists x \in \mathcal{N} : x \cdot (j - i) = N$ because $j - i < m_N$. Let us call seq the sequence of processor states: $s_i, s_{i+1}, s_{i+2}, \dots, s_{j-1}$. Study the deadlock configuration where seq is repeated x times. This configuration is denoted by seq^x .

There is no leader in the deadlock configuration: for any processor, there is at least another processor that has exactly the same local configuration. Thus \mathcal{P} is not a SSLE protocol.

2. $\exists s \in S, \forall s' \in S, (s, s')$ is not a local deadlock.

As $2 \cdot (l + 1) < m_N, \exists x \in \mathcal{N} : x \cdot 2(l + 1) = N$. Let us call seq' the sequence of processor states $s, \sigma_1, s, \sigma_2, s, \sigma_3, s, \sigma_4, \dots, s, \sigma_l$. Consider the configuration $(s, \sigma_1, seq')^x$. Notice that $x \geq 2$.

From this configuration, consider the maximal computations in which the unfair demon always selects the second processor in the ring. This is possible because $\forall s' \in S (s, s')$ is not a local deadlock. During these computations, the reached configurations have this pattern: $s, s', seq', (s, \sigma_1, seq')^{x-1}$

In any reachable configuration, for any processor there is at least one other processor that has exactly the same local configuration. From a given configuration, for a sequence of choices of the unfair daemon, the measure of the set of computations which do not elect a unique leader is 1. Thus \mathcal{P} is not a SSLE protocol under an unfair daemon. \square

Remark 4 *The number of states per processor required by a randomized self-stabilizing token circulation protocol under an unfair daemon on anonymous and unidirectional N -rings is greater than $(m_N - 2)/2$. The proof is similar to the proof of theorem 4.*

Remark 5 *In [19], Lin and Simon have found a similar bound $(\sqrt{m_N})$ to the self-stabilization of a phase clock on uniform unidirectional rings under a synchronous daemon.*

Remark 6 *There is no randomized SSLE protocol under an unfair daemon (distributed or centralized) that requires constant memory space.*

The number of bits required by a randomized SSLE protocol under an unfair daemon on anonymous and unidirectional rings of any size N is $\Omega(\lg m_N)$ bits. In the next section, we will present a space optimal SSLE protocol.

5.1 A randomized protocol for anonymous rings

In section 5.1.1, we present a protocol-compiler that transforms any self-stabilizing protocol on unidirectional rings under a fair daemon into a self-stabilizing protocol under an unfair daemon. This compiler allows us to obtain a space optimal randomized token circulation protocol under an unfair daemon on anonymous and unidirectional rings which we present in section 5.1.2. Finally, in section 5.1.3, a space optimal randomized SSLE protocol under an unfair daemon for anonymous and unidirectional rings of any size is presented.

In this extended abstract, the proof of correctness will be omitted.

Deterministic Token circulation on anonymous rings

Variable on p :

dt_p is a variable taking values in $[0, m_N]$.
(the deterministic token)

Notation on p :

lp is the predecessor of p .

Predicate on p :

$Deterministic_token(p) \equiv dt_p - dt_{lp} \neq 1 \pmod{m_N}$

Macro on p :

$Pass_Deterministic_token(p) : dt_p := (dt_{lp} + 1) \pmod{m_N}$

Action on p :

$\mathcal{A}_1 : Deterministic_token(p) \longrightarrow Pass_Deterministic_token(p)$

5.1.1 Protocols under a fair versus an unfair daemon

Deterministic token circulation The basic protocol performs deterministic token circulations. The token circulations verify the following properties:

1. There is always at least one deterministic token in a configuration.
2. a processor has a deterministic token if and only if the processor is enabled (on the processor, an action guard is satisfied).
3. When a processor performs an action, it passes its token to its successor.
4. When a deterministic token catches up with another token, one of them disappears.

We present a deterministic token circulation algorithm that verifies the required properties. The deterministic tokens circulation requires a variable with value less than m_N (at first, the convention to represent a token was proposed in [6]).

Note that the daemon can prevent the disappearance of any of the initial tokens. We call these tokens “deterministic tokens”, because they are ruled by a deterministic protocol. Along any computation, the difference between the number of p 's actions with the number of p' 's actions is bounded by the number of deterministic tokens between p and p' (this number is bounded by the distance between p and p'). Thus, this difference is bounded by N .

Protocol-compiler Let \mathcal{P} be a self-stabilizing protocol under a bounded (resp. fair) daemon on anonymous and unidirectional rings to solve a problem \mathcal{PR} .

The compiler modifies the actions of \mathcal{P} in such a way that holding a deterministic token is necessary to perform an action of \mathcal{P} . After an action of \mathcal{P} , the processor has passed the deterministic token to its neighbor. If a processor holds a deterministic token and it does not satisfy any guard of \mathcal{P} it must pass its deterministic token when it is chosen by the daemon. When a processor passes a deterministic token, if it can perform an action of \mathcal{P} , it performs this action. A processor, has to wait at most $N \cdot (N - 1)/2$ computation steps before performing an action: another processor can perform at most N actions during the waiting. Let us call

\mathcal{TC} : Randomized token circulation protocol on anonymous rings under unfair daemon

Variables on p :

dt_p is a variable taking values in $[0, m_N]$.
(the deterministic token)
 t_p is a variable taking values in $[0, m_N]$.
(the circulating token)

Notation on p :

v_{lp} is the value of the variable v on lp .

Predicates on p :

$Deterministic_token(p) \equiv dt_p - dt_{lp} \neq 1 \pmod{m_N}$
 $token(p) \equiv t_p - t_{lp} \neq 1 \pmod{m_N}$

Macro on p :

$Pass_Deterministic_token(p) : dt_p := (dt_{lp} + 1) \pmod{m_N}$
 $Pass_token(p) : t_p := (t_{lp} + 1) \pmod{m_N}$

Actions on p :

$A_1 : Deterministic_token(p) \wedge \neg token(p) \longrightarrow$
 $Pass_Deterministic_token(p)$

$A_2 : Deterministic_token(p) \wedge token(p) \longrightarrow$
 $Pass_Deterministic_token(p);$
if (random(0, 1) = 0) then $Pass_token(p)$

\mathcal{BP} the obtained protocol by combining the deterministic token circulation protocol with \mathcal{P} , as we have explained.

During any computation of \mathcal{BP} - under an unfair daemon -, when a processor p satisfies a guard of \mathcal{P} another processor performs at most N actions of \mathcal{P} , before p performs an action of \mathcal{P} . Therefore, all computations of \mathcal{BP} under an unfair daemon are N -bounded in regard to the \mathcal{P} protocol.

The following theorem is a direct consequence of our protocol-compiler.

Theorem 5 *If there is a self-stabilizing protocol on anonymous and unidirectional rings to solve a problem \mathcal{PR} under a centralized (resp. distributed) fair or bounded daemon, then there is a self-stabilizing protocol on anonymous and unidirectional rings to solve the problem \mathcal{PR} under a centralized (resp. distributed) unfair daemon.*

5.1.2 Randomized token circulation under an unfair daemon

In [6], Beauquier, Cordier and Delaët have presented a randomized self-stabilizing token circulation protocol on anonymous and unidirectional rings. If the daemon is synchronous, as in [13] or bounded as in [6], this protocol converges toward configurations where a single token fairly circulates in the ring. With our protocol-compiler, we get a randomized self-stabilizing token circulation protocol without making any assumption on the daemon. This protocol requires m_N^2 processor states. We called this protocol \mathcal{TC} .

5.1.3 Randomized SSLE under an unfair daemon

The same coding technique as in [6] is used to code leader marks: each processor p has a variable lm_p (having a value

Randomized leader election protocol on anonymous rings under unfair daemon

Variables on p :

dt_p is a variable taking values in $[0, m_N]$.
(the deterministic token)
 lm_p is a variable taking values in $[0, m_N]$.
(the leader mark)
 t_p is a variable taking values in $[0, m_N]$.
(the colored token)
 c_p is the color variable taking values in {blue, green}

Notation on p :

v_{lp} is the value of the variable v on lp .

Predicates on p :

$Det_tok(p) \equiv dt_p - dt_{lp} \neq 1 \pmod{m_N}$
 $Leader(p) \equiv lm_p - lm_{lp} \neq 1 \pmod{m_N}$
 $Col_tok(p) \equiv t_p - t_{lp} \neq 1 \pmod{m_N}$

Macro on p :

$Pass_Det_tok(p) : dt_p := (dt_{lp} + 1) \pmod{m_N}$
 $Pass_Leader(p) : lm_p := (lm_{lp} + 1) \pmod{m_N}$
 $Pass_Col_tok(p) : t_p := (t_{lp} + 1) \pmod{m_N}$

Actions on p :

$A_1 : Det_tok(p) \wedge \neg Col_tok(p) \longrightarrow$
 $Pass_Det_tok(p)$

$A_2 : Det_tok(p) \wedge \neg Leader(p) \wedge Col_tok(p) \longrightarrow$
 $Pass_Det_tok(p);$
if (random(0, 1) = 0) then
{ $c_p := c_{lp}; Pass_Col_tok(p)$ }

$A_3 : Det_tok(p) \wedge Leader(p) \wedge Col_tok(p) \wedge$
 $(c_p \neq c_{lp}) \longrightarrow$
 $Pass_Det_tok(p);$
if (random(0, 1) = 0) then {
 $c_p := c_{lp}; Pass_Col_tok(p); Pass_Leader(p);$ }

$A_4 : Det_tok(p) \wedge Leader(p) \wedge Col_tok(p) \wedge$
 $(c_p = c_{lp}) \longrightarrow$
 $Pass_Det_tok(p);$
if (random(0, 1) = 0) then {
 $c_p := \text{random}(\text{blue}, \text{green}); Pass_Col_tok(p);$ }

less than m_N) a processor has a leader mark if and only if $lm_p - lm_{i_p} \neq m_N$. There is always a leader mark in the ring. Initially, the ring may have several leader marks.

We use blue or green colored tokens that circulate on the ring to destroy all leader marks except one. The colored token circulation is provided by the self-stabilizing protocol \mathcal{TC} . Thus, only one colored token will eventually circulate on the ring for any initial configuration and for any daemon.

The goal of the colored token is to freeze the leader mark when there is only one, but also to ensure the circulation of leader marks when the ring contains several ones.

A processor waits for the colored token to check whether or not it holds the single leader mark. When it has the colored token (and a deterministic token), it randomly selects a color and uses the colored token to communicate the color to every processor of the ring. It waits, until receiving a colored token. If the color of the token is the same as the color of the token which it last sent, then it keeps the leader mark and starts the checking again by randomly selecting a color (action \mathcal{A}_4).

Since the color of token is randomly selected, when there are several leader marks in the ring, a processor having a leader mark will eventually received a colored token that does not have the right color (i.e. its color). In this case, the processor randomly passes its leader mark (action \mathcal{A}_3).

As the leader mark speeds are not identical (they depend on a random value); leader marks will catch up other leader marks. Eventually, only one leader mark will stay in the ring.

Once the ring is stabilized, there is one frozen leader mark and one colored token that circulates.

The actions \mathcal{A}_1 and \mathcal{A}_2 ensure circulation of the deterministic tokens and the colored token on processors that do not have a leader mark.

The randomized SSLE protocol uses $2 \cdot m_N^3$ states per processor; each processor needs $O(\lg m_N)$ bits. Note that the value of m_N is constant on average.

References

- [1] Y. Afek, S. Kutten, and M. Yung. Memory-efficient self-stabilization on general networks. In *WDAG90 Distributed Algorithms 4th International Workshop Proceedings, Springer-Verlag LNCS:486*, pages 15–28, 1990.
- [2] S. Aggarwal and S. Kutten. Time optimal self-stabilizing spanning tree algorithm. In *FSTTCS93 Proceedings of the 13th Conference on Foundations of Software Technology and Theoretical Computer Science, Springer-Verlag LNCS:761*, pages 400–410, 1993.
- [3] A. Arora and M.G. Gouda. Distributed reset. *IEEE Transactions on Computers*, 43:1026–1038, 1994.
- [4] B. Awerbuch and R. Ostrovsky. Memory-efficient and self-stabilizing network reset. In *PODC94 Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 254–263, 1994.
- [5] J. Beauquier. Proving self-stabilizing randomized protocols. In Hermes, editor, *OPODIS97 Proceedings of the first international conference On Principles Of Distributed Systems*, pages 279–284, 1997.
- [6] J. Beauquier, S. Cordier, and S. Delaët. Optimum probabilistic self-stabilization on uniform rings. In *Proceedings of the Second Workshop on Self-Stabilizing Systems*, pages 15.1–15.15, 1995.
- [7] J.E. Burns, M.G. Gouda, and R.E. Miller. On relaxing interleaving assumptions. In *Proceedings of the MCC Workshop on Self-Stabilizing Systems, MCC Technical Report No. STP-379-89*, 1989.
- [8] J.E. Burns and J. Pachl. Uniform self-stabilizing rings. *ACM Transactions on Programming Languages and Systems*, 11:330–344, 1989.
- [9] E.W. Dijkstra. Self stabilizing systems in spite of distributed control. *Communications of the Association of the Computing Machinery*, 17:643–644, 1974.
- [10] E.W. Dijkstra. Self-stabilization in spite of distributed control. In *Selected Writings on Computing: A Personal Perspective*, pages 41–46. Springer-Verlag, 1982. (paper's original date is 1973).
- [11] S. Dolev. Optimal time self-stabilization in dynamic systems. In *WDAG93 Distributed Algorithms 7th International Workshop Proceedings, Springer-Verlag LNCS:725*, pages 160–173, 1993.
- [12] S. Dolev, M.G. Gouda, and M. Schneider. Memory requirements for silent stabilization. In *PODC96 Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 27–34, 1996.
- [13] T. Herman. Probabilistic self-stabilization. *Information Processing Letters*, 35:63–67, 1990.
- [14] A. Israeli and M. Jalfon. Token management schemes and random walks yield self-stabilizing mutual exclusion. In *PODC90 Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing*, pages 119–131, 1990.
- [15] G. Itkis and L. Levin. Fast and lean self-stabilizing asynchronous protocols. In *FOCS94 Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science*, pages 226–239, 1994.
- [16] G. Itkis, C. Lin, and J. Simon. Deterministic, constant space, self-stabilizing leader election on uniform rings. In *WDAG95 Distributed Algorithms 9th International Workshop Proceedings, Springer-Verlag LNCS:972*, pages 288–302, 1995.
- [17] C. Johnen. Deterministic, silent and self-stabilizing leader election algorithm on id-based rings. Technical report, Laboratoire de Recherche en Informatique, Université de Paris-Sud, 1998.
- [18] H. Kakugawa and M. Yamashita. Uniform and self-stabilizing token rings allowing unfair daemon. *IEEE Transactions on Parallel and Distributed Systems*, 8:154–162, 1997.
- [19] C. Lin and J. Simon. Possibility and impossibility results for self-stabilizing phase clocks on synchronous rings. In *Proceedings of the Second Workshop on Self-Stabilizing Systems*, pages 10.1–10.15, 1995.
- [20] A. Mayer, Y. Ofek, R. Ostrovsky, and M. Yung. Self-stabilizing symmetry breaking in constant-space. In *STOC92 Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 667–678, 1992.

- [21] A. Mayer, R. Ostrovsky, and M. Yung. Self-stabilizing algorithms for synchronous unidirectional rings. In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA96)*, pages 564–573, 1996.
- [22] A. Pogosyants, R. Segala, and N. Lynch. Verification of the randomized consensus algorithm of Aspnes and Herlihy: a case study. In *WDAG97 Distributed Algorithms 11th International Workshop Proceedings*, Springer-Verlag LNCS:1320, pages 22–36, 1997.