JOURNAL OF AEROSPACE COMPUTING, INFORMATION, AND COMMUNICATION
Vol. 3, October 2006

Observing Locally Self-stabilization in a Probabilistic Way

Joffroy Beauquier,* Laurence Pilard,† and Brigitte Rozoy‡
LRI - CNRS UMR 8623, Université Paris-Sud, 91405 Orsay, France

A self-stabilizing algorithm cannot detect by itself that stabilization has been reached. For overcoming this drawback Lin and Simon introduced the notion of an external observer: a set of processes, one being located at each node, whose role is to detect stabilization. Furthermore, Beauquier, Pilard and Rozoy introduced the notion of a local observer: a single observing entity located at a unique node. This entity does not detect false stabilization, eventually detects stabilization, and does not interfere with the observed algorithm. We introduce here the notion of probabilistic observer which realizes the conditions above with probability 1. We show that computing the size of an anonymous ring with a synchronous self-stabilizing algorithm cannot be observed deterministically. We prove that some synchronous self-stabilizing solution to this problem can be observed probabilistically.

I. Introduction

THE notion of self-stabilization was introduced by Dijkstra[1]. An algorithm was defined as self-stabilizing when "regardless of its initial state, it is guaranteed to arrive at a legitimate state in a finite number of steps". Such a property is very desirable for any distributed algorithm, because after any unexpected perturbation modifying the memory state, the algorithm eventually recovers and returns to a legitimate state, without any outside intervention.

Dijkstra's notion of self-stabilization, which originally had a very narrow scope of application, is proving to encompass a formal and unified approach to fault-tolerance under a model of transient failures for distributed algorithms[2,3].

It has been objected to the self-stabilizing approach that 1) a self-stabilizing algorithm only eventually recovers, involving that during some time the behaviour is not correct, 2) a process can never know whether or not the algorithm is stabilized[2].

There is little that you can do against the first point because it is inherently bounded to the very definition of self-stabilization.

There is a few paper dealing with the second issue, even if there is a lot of works dealing with control of distributed algorithms such as snapshot computations or predicate detection techniques[4–13]. All these papers present algorithms which either do not deal with failures, or with crash failures only, or do not give any safety condition about the result after a failure. In this paper, we present a method of self-stabilization detection such that when stabilization is detected after a failure, the system is stabilized.

The only solutions satisfying this safety condition are the important paper by Lin and Simon[14] and the paper[15]. Obviously, no detection of stabilization from the inside is possible since any local variable used for that purpose could be corrupted. Meanwhile it is perfectly feasible to detect stabilization from the outside (for instance and although it is just a theoretical remark, when a bound on the number of steps before stabilization is known, simply by counting). "From the outside" can be replaced by "from the inside but using stable memory" (memory not subject to failures).

Received 2 September 2005; revision received 20 March 2006; accepted for publication 5 June 2006.

* beauquier@lri.fr
† pilard@lri.fr
‡ rozoy@lri.fr

By restricting their attention to ring networks, Lin and Simon[14] propose a new model, in which it is meaningful to say that a process knows that the ring is stable. This model introduces the notion of a distributed observer, located at each node of the network. This observer is responsible for detecting stabilization and does not influence the self-stabilizing protocol. As the observer involves, at each node, the presence of a stable memory, such a distributed observer is suited only for small local area networks, in which strong security and reliability can be ensured. It is unrealistic for large or heterogeneous networks.

In[15], a local observer has been introduced. This local observer is located at only one node of the network, then only one node has to dispose of some stable memory. The local observer is responsible for detecting stabilization and does not influence the self-stabilizing protocol. In[15], it is proven that if there exists a synchronous self-stabilizing distributed solution for some problem in a distinguished network, then there exists a synchronous self-stabilizing distributed solution, not necessarily the same, for the same problem, that can be observed by a local observer.

In this paper, we raise the question of determining whether or not the stabilization detection is feasible by a local observer in an anonymous and synchronous network. We prove that there exists a self-stabilizing algorithm for some problem P, for which there is no deterministic observation. Then we introduce the notion of local and probabilistic observer and we prove that such an observer can detect the stabilization of another self-stabilizing algorithm solving P.

The plan of the paper is the following. First, we describe the distributed systems and we introduce the formal definition of a local and probabilistic observer. Then, we prove that the problem of determining the size of a synchronous, anonymous and one-way ring cannot be observed by a local deterministic observer. Next, we propose a self-stabilizing algorithm which computes the size of a synchronous, anonymous and one-way ring. Finally, we prove that this algorithm can be observed by a local and probabilistic observer.

The present paper is a full version of[16].

II. Model

In this section, we define a distributed algorithm and we state what it means for a distributed algorithm to be self-stabilizing. We use the classical model for distributed algorithms of[3] and the notion of a local observer of[15]. We recall the definition of a deterministic observer and we define a probabilistic observer.

A. Distributed Algorithm

Definition 2.1 (Distributed algorithm). *A distributed algorithm $\mathcal{A} = (C, \lambda, T)$ is an automaton, where C is the set of all states (called* configurations*) of $\mathcal{A}$, λ is the set of labels (called* actions*) and T is the set of all transitions of $\mathcal{A}$: $T \subset C \times \lambda \times C$.*

In a probabilistic distributed algorithm, there is a probabilistic law on the output of a transition such that in each configuration the sum of the probabilities associated to each outgoing transition with the same action is 1.

Definition 2.2 (Execution). *An execution of $\mathcal{A}$, noted $e = c_1 a_1 c_2 a_2 \ldots$ is a maximal sequence of alternating configurations and actions of $\mathcal{A}$ where (c_i, a_i, c_{i+1}) is a transition of $\mathcal{A}$.*

The sequence is maximal if it is either infinite, or it is finite but there is no outgoing transition from the last configuration.

We use a *synchronous* mode of communication, in which a global clock that generates an infinite sequence of pulses, equally spaced in time, is connected to all the processes in the network. The time interval between two consecutive pulses of the clock is a *round*. At the beginning of each round, each process decides, according to its state, what messages to send and on which links to send them. Each process then executes a finite number of internal actions, and finally receives any messages send to it by any of its neighbors in this round.

We consider *anonymous* networks in which processes have no identifiers and the same code. In such networks, it is impossible to distinguish between two distinct processes. We also use the notion of *distinguished* network that is a network where exactly one process can be distinguished from the others.

The formal definition of a self-stabilizing probabilistic algorithm can be state using the formulation of Lynch and Segala[17,18]. This formulation uses the notion of *probability space* and *cone*.

Definition 2.3 (Probability space). *A probability space is a triplet* $(\Omega, \mathcal{F}, Pr)$ *where* Ω *is a set,* $\mathcal{F}$ *is a collection of subsets of* Ω *that is closed under complement and countable union and such that* $\Omega \in \mathcal{F}$*, and* Pr *is a function from* $\mathcal{F}$ *to* $[0, 1]$ *such that* $Pr(\Omega) = 1$ *and for any collection* $\{C_i\}_i$ *of at most countably many pairwise disjoint elements of* $\mathcal{F}$*,* $Pr(\cup_i C_i) = \sum_i Pr(C_i)$*.*

The set Ω contains the objects that we want to analyze: the set of executions of a probabilistic distributed algorithm. The set $\mathcal{F}$ contains the subsets of Ω that we can measure. Finally, Pr is a function that associates a measure with each element of $\mathcal{F}$.

Definition 2.4 (Cone). *Let* $\mathcal{A}$ *be a probabilistic distributed algorithm. A cone* V_h *of* $\mathcal{A}$ *is the set of all* $\mathcal{A}$*'s executions with the common prefix* h*.* h *is called the history of the cone. We note* $|h|$ *the length of* h*.*

In this paper, we consider that specifications are predicate over configurations.

In the two following definitions, the convergence and correctness properties are probabilistic as in[19].

Furthermore, in the following definition, these notations are used: let $\mathcal{A}$ be a probabilistic distributed algorithm, $\mathcal{SP}$ be the specification of $\mathcal{A}$, and $\mathcal{C}$ be a subset of the set of configurations of $\mathcal{A}$. We note $\mathcal{E}_{\mathcal{C}}$ the set of maximal executions of $\mathcal{A}$ reaching a configuration in $\mathcal{C}$ and we note $\mathcal{E}_{\mathcal{C}.\mathcal{SP}}$ the set of maximal executions of $\mathcal{A}$ reaching a configuration in $\mathcal{C}$ and then from this configuration verifying $\mathcal{SP}$. Finally, we note V_h a cone of $\mathcal{A}$. We consider the probability space $(\Omega, \mathcal{F}, Pr)$ where Ω is the whole executions of $\mathcal{A}$ and $\mathcal{F}$ is the closure of Ω under the operation of definition 2.5.

Definition 2.5. *A probabilistic distributed algorithm* $\mathcal{A}$ *is self-stabilizing for a specification* $\mathcal{SP}$ *if and only if there exists a sub-set* $\mathcal{C}_L$ *(legitimate configurations) of the set of the configurations of* $\mathcal{A}$ *such that:*

- *(probabilistic convergence) The probability of the set of maximal executions of* $\mathcal{A}$ *reaching a configuration in* $\mathcal{C}_L$ *is equal to 1, i.e. the probability for* $\mathcal{E}_{\mathcal{C}_L}$ *is equal to 1. Formally:*

$$\lim_{|h|\to+\infty} Pr(V_h \cap \mathcal{E}_{\mathcal{C}_L}) = 1$$

- *(probabilistic correctness) The probability of the set of maximal executions of* $\mathcal{A}$ *reaching a configuration in* $\mathcal{C}_L$ *and then from this legitimate configuration verifying* $\mathcal{SP}$ *is equal to 1,* i.e. *the probability for* $\mathcal{E}_{\mathcal{C}_L.\mathcal{SP}}$ *is equal to 1. Formally:*

$$\lim_{|h|\to+\infty} Pr(V_h \cap \mathcal{E}_{\mathcal{SP}}) = 1$$

For a sake of clarity, we will use for the proof a simpler but equivalent definition of probabilistic self-stabilization.

Let e be an execution of $\mathcal{A}$ that is not maximal. We note $last(e)$, the last configuration of e and $|e|$ the length of e. We say that e' is a continuation of e if and only if there exists an execution E of $\mathcal{A}$ such as ee' is a prefix of E.

Definition 2.6. *A probabilistic distributed algorithm* $\mathcal{A}$ *is self-stabilizing for a specification* $\mathcal{SP}$ *if and only if there exists a sub-set* $\mathcal{C}_L$ *(legitimate configurations) of the set of the configurations of* $\mathcal{A}$ *such that:*

- *(probabilistic convergence) the probability of the set of maximal executions of* $\mathcal{A}$ *reaching a configuration in* $\mathcal{C}_L$ *is equal to 1. Formally, noting* e *an execution of* $\mathcal{A}$ *that is not maximal,*

$$\lim_{|e|\to+\infty} Proba(\text{last}(e) \in \mathcal{C}_L) = 1$$

- *(probabilistic correctness) the probability of the set of maximal executions of* $\mathcal{A}$ *reaching a configuration in* $\mathcal{C}_L$ *and then from this legitimate configuration verifying* $\mathcal{SP}$ *is equal to 1. Formally, noting* e *an execution of* $\mathcal{A}$ *that is not maximal,*

$$\lim_{|e|\to+\infty} Proba(\forall e' \textit{ a continuation of } e : \text{last}(ee') \textit{ verifies } \mathcal{SP}) = 1$$

B. Observer

As described in[15], an observer has the following features. (1) The observer is *located at a process of the network*. If the network is anonymous, the location of the observer is arbitrary. (2) The observer is not allowed to detect stability with any information *depending on the network* (for instance the size). (3) The observer *cannot interfere with the algorithm*, which means that the executions of the algorithm are the same with or without the observer. (4) The observer is *not subject to any type of corruption*. (5) The observer observes the behavior of the local process (sequential sequence of instructions) and tries to *match part of this behavior with some predefined sequences*. (6) The announcement of the stabilization obeys some *safety* and *liveness* conditions.

The observer can be viewed as a mechanism having a predetermined set of sequences of actions as a parameter. The mechanism observes the local behavior of a process and continuously tries to match one of its sequences to the observed behavior. As soon as a matching is performed, the observer *announces* the stabilization.

The observer must satisfy three conditions:

1. Safety. The observer does not *announce* if the algorithm is not stabilized.
2. Liveness. Once the algorithm is stabilized, the observer eventually *announces*.
3. Non-interference. The executions of the algorithm are the same with or without the observer.

Definition 2.7 (Deterministic observer). *If $\mathcal{A}$ is a self-stabilizing algorithm and $\mathcal{O}$ a deterministic observer of $\mathcal{A}$, we have: (safety) as long as $\mathcal{A}$ is not stabilized, $\mathcal{O}$ returns* false*, and (liveness) once $\mathcal{A}$ is stabilized, $\mathcal{O}$ eventually returns* true*. We say that $\mathcal{A}$ is an* observable algorithm*.*

The deterministic and probabilistic observers have different safety condition. A probabilistic observer is defined with a parameter α.

Definition 2.8 (Probabilistic observer). *If $\mathcal{A}$ is a self-stabilizing algorithm and $\mathcal{O}_\alpha$ a probabilistic observer of $\mathcal{A}$, we have:*
(liveness) the observer eventually announces the stabilization with probability one; (safety) $\forall \varepsilon \in]0, 1], \exists \alpha$ used by the observer: Proba(correct announcement by $\mathcal{O}_\alpha$) $> 1 - \varepsilon$.

The liveness property is probabilistic because a probabilistic algorithm observed could never stabilized (with probability 0).

In the safety property, ε is the margin of error of the announcement. For each margin of error ε allowed for the announcement, there exists a value of the parameter α such that the probability for a false announcement is less than ε. α is used to compute predetermined sequences of the observer. Intuitively the smaller ε is, the higher α is.

III. Impossibility Result with a Deterministic Observer

In[15], the following result has been proven: for any problem *pb* in a synchronous and distinguished (presence of a leader) network, we have: if there exists a self-stabilizing algorithm $\mathcal{A}$ solving *pb*, then there exists a self-stabilizing algorithm $\mathcal{B}$ solving *pb* and which is observable in a deterministic way. In this section, we raise the following question: does the result remain true if the network is anonymous? For this purpose, we present a synchronous and anonymous problem which cannot be observed in a deterministic way. This problem is the computation of the size of a synchronous, anonymous and one-way ring.

Theorem 3.1. *Let $\mathcal{A}$ be a self-stabilizing algorithm computing the size of a synchronous, anonymous and one-way ring. There is no deterministic observer for $\mathcal{A}$.*

Proof. The proof is by contradiction and uses a classical technique.

Let $\mathcal{O}bs$ be a deterministic observer of $\mathcal{A}$. Let R be a ring and n be the size of R. We execute $\mathcal{A}$ and $\mathcal{O}bs$ on R. $\mathcal{O}bs$ is located at an arbitrary process of the ring. Let P_0 be this process. For the need of the proof, the processes of R are named as follows: for all i in $[0, n-1]$, P_i is the successor of $P_{(i+1) \bmod n}$.

Let $State_{P_i}$ be the local state of the process $P_i \in R$ at the initial round ($State_{P_i}$ includes all messages contained in the input channel of P_i).

Let e be an execution of $\mathcal{A}$ and $\mathcal{O}bs$ on R, from the configuration: $State_{P_0}, \ldots, State_{P_{n-1}}$. Let r be the round of e where $\mathcal{O}bs$ announces the stabilization.

Now, let R' be a ring and n' be the size of R' such that $n' \geq r+1$ and $n' \neq n$. We execute $\mathcal{A}$ and $\mathcal{O}bs$ on R'. $\mathcal{O}bs$ is located at an arbitrary process of the ring. Let P'_0 be this process. For the need of the proof, the processes of R are named as follows: for all i in $[0, n-1]$, P'_i is the successor of $P'_{(i+1) \bmod n}$.

Let e' be an execution of $\mathcal{A}$ and $\mathcal{O}bs$ on R', with this initialization:

$$\forall i \in [0, n'-1] : State_{P'_{i \bmod n}} = State_{P_i}.$$

We say that $P_i \in R$ is the *associate* of $P'_j \in R'$ if and only if $i = j \bmod n$. For instance, P_0 is the associate of P'_0 and P'_{2n}, if it exists. If P is the associate of P', then P and P' have the same initialization in e and in e'.

If $\mathcal{A}$ is deterministic and since the ring is one-way then all actions executed by a process at a round depend only on the local states of P and of its predecessor. Thus, the observer observes exactly the same actions in P_0 and P'_0 during the first r rounds of e and e'.

If $\mathcal{A}$ is probabilistic, then all actions executed by a process at a round also depend on probabilistic choices in the execution e. In this case, we choose e' such that all probabilistic choices in e are the same than in e'. Thus, the observer sees exactly the same actions in P_0 and P'_0 during the first r rounds of e and e'.

$\mathcal{O}bs$ announces during the round r in e, so $\mathcal{O}bs$ announces during the round r in e'. But the size of R is different from the size of R', thus $\mathcal{O}bs$ makes a false announcement in e'. Therefore $\mathcal{O}bs$ is not an observer for $\mathcal{A}$. ■

IV. Positive Result with a Probabilistic Observer

Let *pb* be the problem of computing the size of a synchronous, anonymous and one-way ring. We proved in section III that if $\mathcal{A}$ is a self-stabilizing algorithm solving *pb*, then it cannot exist any deterministic observer for $\mathcal{A}$. We introduce in this section a self-stabilizing algorithm $\mathcal{RS}$ solving *pb*, which is observable in a probabilistic way. In the first part, we describe this algorithm, then its probabilistic observer.

A. The Algorithm

In the sequel we note n the size of the ring.

1. *Specification of the algorithm*

The algorithm is probabilistic and verifies the following specification $\mathcal{SP}$: (i) each process knows eventually the size of the ring with probability 1, and (ii) from some point, a process that knows the size of the ring does not modify this value with probability 1.

The variable $size_P$ of a process P contains the current ring size value estimated by P. The legitimate configurations $\mathcal{C}_L$ of $\mathcal{RS}$ are the configurations of $\mathcal{RS}$ in which: $\forall P, size_P = n$. A maximal execution e of $\mathcal{RS}$ satisfies the specification $\mathcal{SP}$ if and only if e has a suffix e' containing only legitimate configurations. We note $size_P(last(e'))$ the value of $size_P$ at the configuration $last(e')$.

- (probabilistic convergence) the probability of the set of maximal executions of $\mathcal{RS}$ reaching a configuration in $\mathcal{C}_L$ is equal to 1. Formally, noting e an execution of $\mathcal{RS}$ that is not maximal,

$$\lim_{|e| \to +\infty} \text{Proba}(\forall P : size_P(last(e)) = n) = 1$$

- (probabilistic correctness) the probability of the set of maximal executions of $\mathcal{RS}$ reaching a configuration in $\mathcal{C}_L$ and then from this legitimate configuration verifying $\mathcal{SP}$ is equal to 1. Formally, noting e an execution of $\mathcal{RS}$ that is not maximal,

$$\lim_{|e| \to +\infty} \text{Proba}(\forall e' \text{ a continuation of } e, \forall P : size_P(last(ee')) = n) = 1$$

2. *Basic idea of the algorithm*

Each process will repetitively send tokens around the ring, each of them testing some value for the size. When a process receives a token, the test value is either accepted or rejected by the process. Accepting a value means that

the process considers that this value can be the size of the ring. Rejecting a value means that the process considers that this value cannot be the size of the ring. A rejected value, is not the size of the ring (except for values resulting from a bad initialization of the system). An accepted value is possibly but not necessarily the size of the ring. But in this case we will see there are much more acceptances that refusals. A process chooses a value for being the size of the ring when the number of refusals and the number of acceptances are significantly different.

3. Description of the algorithm

The ring is anonymous, then all processes in the ring execute the same program. Processes communicate by token passing.

When creating a token T, process P assigns a **life time** to T in $life_T$. Then P saves this life time in $life_P$. Now P has to wait the end of this life time for creating another token. For that, the process has a counter cpt_P such that: at the beginning of each round, cpt_P is incremented by 1 and, when the process creates a new token, cpt_P is reset to 0. Thus, when $cpt_P \geq life_P$, the life time of the last token created by P is over. Furthermore, every time P creates a token, the assigned life time of the token is incremented by 1. Thus, as soon as $cpt_P \geq life_P$, a new token is created with a new life time (incremented by 1).

A process also assigns to each created token T a **counter** cpt_T which is initialized to 1. Moreover, at each round, if T is transmitted, then the counter is incremented by 1, otherwise T is destroyed. A process relays all tokens which have not exhausted their life time ($cpt_T < life_T$). Thus, eventually all token disappears and eventually two distinct tokens created by P are not be in the ring during the same round.

Finally, a process assigns to a token a **color**. The process randomly chooses the color between black and white (equiprobability) and stores this color in $color_P$.

In summary, the variables of a process are:

1. $color_P \in$ {black, white}: the color of the last token created by P;
2. $cpt_P \in \mathbb{N}^*$: the number of rounds since P has created its last token. At a round, if P creates a token, then cpt_P is reset to 0, elsewhere cpt_P is incremented by 1;
3. $life_P \in \mathbb{N}^*$: the life time of the last token created by P. When P creates a token, $life_P$ is incremented by 1;
4. $size_P \in \mathbb{N}^*$: the size of the ring computed by P. $size_P$ is the output variable of the algorithm.

The variables of a token $Token(color_T, cpt_T, life_T)$ are:

1. $color_T \in$ {black, white}: the color of the token (constant during the life of T);
2. $cpt_T \in \mathbb{N}^*$: the number of processes visited by the token. cpt_T is initialized to 1, then is incremented by 1 at each round (this variable also counts the number of rounds since T has been created);
3. $life_T \in \mathbb{N}^*$: the life time of the token (constant during the life of T).

Computation of the size of the ring. Between two token creations, a process analyses all tokens that it receives. Each token received by P is either recognized or not. A token is recognized by P if and only if $color_P = color_T$ and $cpt_P = cpt_T$. "P recognizes a token T" means that T is possibly the last token created by P. Note that if P recognizes T and T does not result from a bad initialization of the system, then the last token created by P has the same color as T ($color_P = color_T$) and has been created the same round as T ($cpt_P = cpt_T$). Note that T is not necessarily the last token created by P. On the other hand, if P does not recognize T and T does not result from a bad initialization of the system, then: if $color_P \neq color_T$ then the last token created by P has not the same color as T and then T is not the last token created by P and, in the same way, if $cpt_P \neq cpt_T$ then the last token created by P has not been created the same round as T and then T is not the last token created by P.

In order to compute the size of the ring, a process P has two arrays: $S_P[\,]$ and $F_P[\,]$ in which P counts respectively the number of recognized and not recognized tokens among those received. More precisely, if P receives and recognizes a token T such that $cpt_T = i$, then P *marks a success in* i, *i.e.* P executes the action $S_P[i] := S_P[i] + 1$. Otherwise, if P receives but does not recognize a token T such that $cpt_T = i$, then P *marks a failure in* i, *i.e.* P executes the action $F_P[i] := F_P[i] + 1$.

Then, a process looks at the ratio $R_P[i] = S_P[i]/(S_P[i] + F_P[i])$. It will be proved that $R_P[i]$ is (in probability) around $1/2$ or less than $1/2$ if i is not a multiple of the size, and that $R_P[i]$ is quickly closed to 1 when i is a multiple

of the size. Thus, a process P computes the size of the ring in $size_P$:

$$size_P := \inf\left\{i > 0 : \frac{S_P[i]}{S_P[i] + F_P[i]} \geq 0.9\right\}$$

Figure 1 contains the text of the algorithm executed by processes. Each round a process executes the procedure *Compute-Size()*.

4. Informal explanation of the algorithm

Example 1. Figure 2 shows an example of execution. Let r be a round and P be a process creating a token T at round r. Let us suppose that the value of $life_P$ is 11 at round $r-1$ and that P chooses black for T. Thus, at round r, P initializes its variables: $color_P := \bullet$; $cpt_P := 0$; $life_P := 12$ and P creates $T(\bullet, 1, 12)$. According to the algorithm $\mathcal{RS}$, T is transmitted if and only if $cpt_T < life_T$. Thus T circulates around the ring until round $r + 12$.

At round $r + 5$, P receives T with $cpt_T = cpt_P = 5$ and $color_T = color_P = \bullet$. Thus, at round $r + 5$, P marks a success in 5. At round $r + 10$, P receives T for the second time with $cpt_T = cpt_P = 10$ and $color_T = color_P = \bullet$. Thus, at round $r + 10$, P marks a success in 10.

At round $r + 12$, Q_2 receives T with $cpt_T = life_T$. Thus Q_2 does not transmit T. T disappears from the ring. During all the token circulation, $life_P$ and $life_T$ remains constant, thus during all the token circulation, $life_P = life_T$. Thus, when T disappears, P knows that fact and then, if P does not receive any token during the round, creates a new token.

We call *real-token* a token created by the execution of the function *Create-Token()* and *false-token* a token resulting from a bad initialization.

We note $S_P[i](r)$, the value of the variable $S_P[i]$ at round r. We note $S_P[i]$, the value of the variable $S_P[i]$ at the current round. We use the same notation for all variables.

In the sequel k denotes a strictly positive integer.

Procedure Create-Token() =

$color_P :=$ uniformly at random choice in {white, black}
$cpt_P := 0$
$life_P := life_P + 1$
Send Token($color_P, 1, life_P$) to the successor of P

Procedure Compute-Size() =

$cpt_P := cpt_P + 1$
$m_P :=$ value of the received channel of P

Part 1: P receives a token T.
if $m_P = \text{Token}(color_T, cpt_T, life_T)$ **then**
  (1) P marks a success or a failure
  if $cpt_P = cpt_T \wedge color_P = color_T$ **then** $S_P[cpt_T] := S_P[cpt_T] + 1$
  else $F_P[cpt_T] := F_P[cpt_T] + 1$

  (2) P updates $size_P$.
  $size_P := \inf\left\{i > 0 : \frac{S_P[i]}{S_P[i]+F_P[i]} \geq 0.9\right\}$

  (3) If T does not reach its life time, then P transmits T,
  (4) else if the last token created by P has disappeared then P creates a new token.
  if $cpt_T < life_T$
    then Send Token($color_T, cpt_T + 1, life_T$) to successor of P
    else if $cpt_P \geq life_P$ **then** Create-Token()

Part 2: P does not receive any token.
  if the last token created by P has disappeared then P creates a new token.
else if $cpt_P \geq life_P$ **then** Create-Token()

Fig. 1 The algorithm.

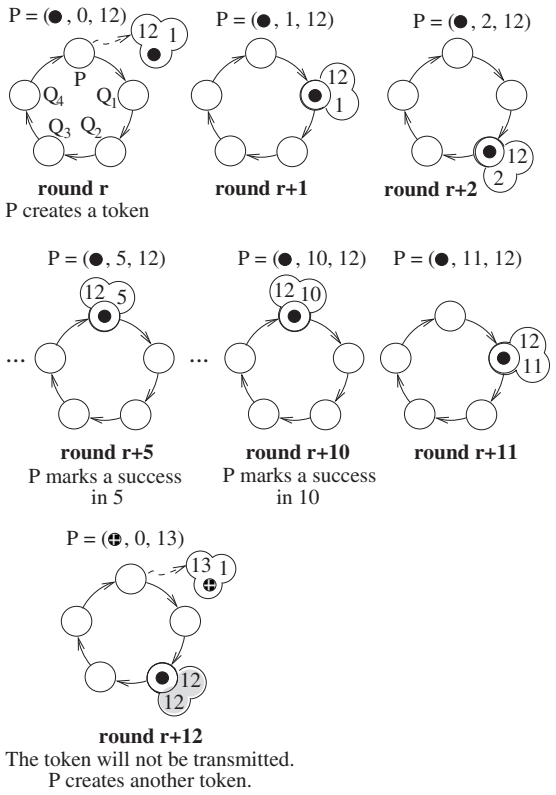

Fig. 2 Behaviours of successes and failures for multiples of the size (⊕ represents the white color).

Ratio for multiples of the size. If n is the size of the ring and if a process P receives a real-token T with $cpt_T = n$, then T has performed one complete circulation around the ring. Thus T has been created by P, $color_P = color_T$ and $cpt_P = cpt_T$ when P receives T (for the first time). Then P recognizes T. In the same way, if a process P receives a real-token T with $cpt_T = kn$, then T has performed k complete circulations around the ring. Thus T has been created by P, $color_P = color_T$ and $cpt_P = cpt_T$ when P receives T (for the k^{th} time). P recognizes T.

Thus, if a process P receives a token T such that $cpt_T = kn$, then:

(i) either T is a real-token, then T has been created by P, $color_P = color_T$ and $cpt_P = cpt_T$, and then P marks a success in cpt_T;

(ii) or T comes from a bad initialization of the system; T is a false-token. Note there are at most n false-tokens, because the capacity of a channel is 1. Moreover, cpt_T is incremented by 1 at each round. Thus for each token T and for all $j \geq 1$, $cpt_T = j$ is true during at most one round in an execution. Therefore P marks something in j at most one time for each false-token and so P marks in kn at most n times during an execution. Thus, P marks at most n failures in kn during an execution.

It will be proved that P creates an infinite number of tokens. Each of them are recognized by P when they return to P after some complete turns of the ring. Thus, P marks an infinite number of successes in kn. Furthermore, we have seen that P marks at most n failures in kn. Thus, for all process P we have:

$$\lim_{r \to +\infty} \left(\frac{S_P[kn](r)}{S_P[kn](r) + F_P[kn](r)} \right) = 1 \geq 0.9$$

and n is the least "kn" that satisfies this relation.

Example 2. Figure 3 shows the same execution as in figure 2, but considers the behavior of several processes. At round r, P creates a token T.

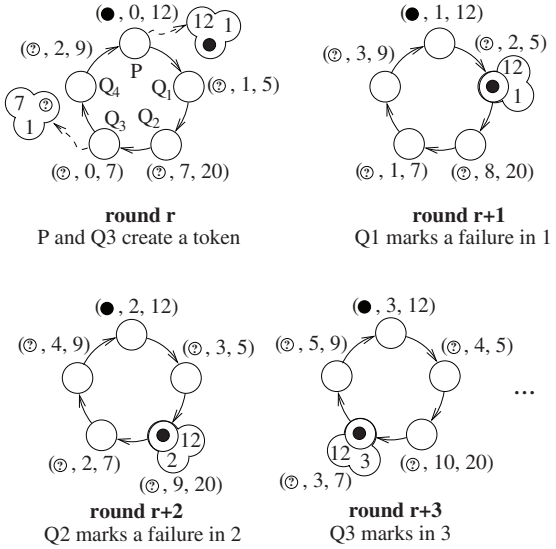

Fig. 3 Behaviours of successes and failures for not multiples of the size (? represents black or white).

Suppose that Q_3 creates a token T' at round r, with $life_{T'} = 7$. Suppose that Q_1, Q_2 and Q_4 do not create any token at round r. At round r, we have: $Q_1 = (?, 1, 5)$, $Q_2 = (?, 7, 20)$, $Q_3 = (?, 0, 7)$ and $Q_4 = (?, 2, 9)$. We will see that Q_1, Q_2 and Q_4 mark failures when they receive T, but not necessarily Q_3.

For a sake of clarity, starting from round $r + 1$, figure 3 only shows the token T created by P.

At round $r + 1$, process Q_1 receives T with $cpt_T \neq cpt_{Q_1}$ Thus whatever the color of Q_1 is, Q_1 marks a failure in 1. Note that $cpt_T \neq cpt_{Q_1}$ is due to the fact that P and Q_1 have not created their last token at the same round. In the same way, at round $r + 2$, process Q_2 receives T with $cpt_T \neq cpt_{Q_2}$. Thus whatever the color of Q_2 is, Q_2 marks a failure in 2. At round $r + 3$, process Q_3 receives T with $cpt_T = cpt_{Q_3}$. This equality is due to the fact that P and Q_3 have created their last token at the same round. Now, the color of Q_3 determines if Q_3 marks a success or a failure when it receives T. If $color_{Q_3} = \bullet$, then Q_3 marks a success in 3, else a failure. Note that, when Q_3 created its last token, there was a probability $1/2$ for Q_3 to choose the same color as P, *i.e.* black. Thus, when Q_3 receives T, there is a probability $1/2$ for Q_3 to mark a success and the same probability to mark a failure.

Ratio for not multiples of the size. If n is the size of the ring and if a process P receives a real-token T with $cpt_T \neq kn$, then T has not performed an entire number of turns around the ring. Then T has not been created by P. Let $Q \neq P$ be the creator of T. When P receives T: either $color_P \neq color_T$ and thus P does not recognize T, or $color_P = color_T$ and thus P recognizes T if and only if $cpt_T = cpt_P$. As P and Q have chosen their colors independently, the probability of having $color_P = color_T$ is equal to $1/2$. Thus, when P receives a token T such that cpt_T is not a multiple of n, the idea is that:

$$\text{Proba}(P \text{ marks a success in } cpt_T) \leq 1/2 \leq \text{Proba}(P \text{ marks a failure in } cpt_T)$$

Thus, for all process P and $\forall i \in \mathbb{N}^*$ such that i is not a multiple of n, the ratio $S_P[i]/S_P[i] + F_P[i]$ is smaller than $1/2$, thus we have:

$$\text{Proba}\left(\frac{S_P[i](r)}{S_P[i](r) + F_P[i](r)} \geq 0.9\right) \xrightarrow{r \to +\infty} 0$$

Conclusion. Let us recall that a process P computes the size of the ring in its variable $size_P$:

$$size_P := \inf\left\{i > 0 : \frac{S_P[i]}{S_P[i] + F_P[i]} \geq 0.9\right\}$$

Then the ratio allows us to distinguish between the multiples of the size and the other values. Indeed, only multiples of the size have a ratio value greater than or equal to 0.9. On the other hand, the *inf* allows us to choose the smallest multiple which is the right size.

5. Remark

The algorithm $\mathcal{RS}$ uses an infinite state space. This feature is inherent to any observable self-stabilizing algorithm computing the size of an anonymous ring. Consider for instance the classical token circulation algorithm of Herman[19] (the network must be odd size). One could think that after stabilization (and only a finite memory is used for that) the computation of the size is easy, since it suffices for a process to count the last visit time of a token. But after stabilization, this time is only approximately $2n$ (which is the expected service of time) but not exactly $2n$. Then in order to compute the size, a process has to memorize (using an infinite state space) the different counted time and analyses their frequencies, in the same way as we do in algorithm $\mathcal{RS}$.

B. Proof of the Algorithm

In the proof, we use the following notation: $R_P[i] = S_P[i]/S_P[i] + F_P[i]$

1. Progression of each process

We will first prove that each process creates indefinitely tokens with increasing life time. Recall that cpt_P is the number of rounds since P has created its last token and that $life_P$ is the life time that P gave to its last token.

Lemma 4.1. *Let P be a process. For each round r and for each $l \geq 1$: if $life_P(r) = l$, then $\exists r' > r$ such that $life_P(r') = l + 1$.*

The idea of the proof is to remark that a process ceases to create new tokens only if it is prevented to do that by another process, *i.e.* it receives a new token at each round. We prove that this situation never appears.

Proof. The proof is by contradiction.

Note that all tokens have a bounded life time, thus eventually all false-tokens disappear. Let r_0 be the first round in which the ring no longer contains any false-token. Let l be an integer and let r be a round such that $life_P(r) = l$. If $r < r_0$ then $life_P(r) \leq life_P(r_0)$.

- If $life_P(r) < life_P(r_0)$ then we are done with r' the first round such as $life_P(r') = 1 + l \leq life_P(r_0)$.
- If $life_P(r) = life_P(r_0)$ then let us start again with $r = r_0$

Thus we can suppose that $r \geq r_0$.

If P never created any token from the round r_0, P has necessarily received a token at each round after round r_0. Since all tokens eventually disappear, there exists at least one process in the ring which creates an infinite number of tokens. Let $\mathbb{P}_\infty$ be the set of processes which create an infinite number of tokens and let $\mathbb{P}_{\not\infty}$ be the set of processes which create a finite number of tokens. We have: $1 \leq |\mathbb{P}_\infty| \leq n - 1$ and $1 \leq |\mathbb{P}_{\not\infty}| \leq n - 1$.

Let $r_1 \geq r_0$ be the first round in which (i) the ring does not contain any token created by a process in $\mathbb{P}_\infty$ and (ii) processes in $\mathbb{P}_{\not\infty}$ no longer create token.

If $Q \in \mathbb{P}_\infty$, Q creates an infinite number of tokens, thus we have:

$$\begin{aligned}
&\forall l, \exists r_Q \geq r_1, \forall r \geq r_Q : life_Q(r) > l \\
&\quad \Longrightarrow \exists r_Q \geq r_1, \forall r \geq r_Q : life_Q(r) > n^n + 1 \quad \text{with } l = n^n + 1 \\
&\quad \Longrightarrow \exists r \geq r_1, \forall r' \geq r, \forall Q \in \mathbb{P}_\infty : life_Q(r\prime) > n^n + 1 \quad \text{with } r = \max\{r_Q : Q \in \mathbb{P}_\infty\}
\end{aligned}$$

Let $r_2 \geq r_1$ be a round such as: $\forall r \geq r_2, \forall Q \in \mathbb{P}_\infty : life_Q(r) > n^n + 1$

Between rounds r_2 and $r_2 + n^n$, we have: $\forall P \in \mathbb{P}_{\not\infty}$, no token of P circulates in the ring, and $\forall Q \in \mathbb{P}_\infty$, Q has created at most one token.

But $|\mathbb{P}_\infty| \leq n - 1$.

Thus, between rounds r_2 and $r_2 + n^n$, there exists a sequence of n rounds in which no token is created and at most $n - 1$ tokens circulate in the ring.

Therefore, during this sequence of n rounds, there exists at least one round during which P does not receive any token, and then, during this round, P creates a token. Contradiction. ■

Corollary 4.1. *$\forall P$ a process:*

1. *$\forall l \in \mathbb{N}^*$, P creates an infinite number of tokens T such that $life_T \geq l$;*
2. *$\forall l \in \mathbb{N}^*$, P receives an infinite number of tokens T such that $cpt_T = l$.*

Proof.
Point 1 is clear as, by lemma 4.1, $life_P$ indefinitely increases, which means that P creates token T of $life_T = l+1, l+2, l+3, \ldots$
Point 2: As a token T is transmitted as long as the variable cpt_T has not reached the value $life_T$, a token sent by process P' is received by process P at distance d from P' with $cpt_T = d,\ d+n,\ d+2n,\ d+3n, \ldots$ until it reached $life_T$. Thus, for a given l, if $l = d + kn$, P receives token with $cpt_T = l$ sent by P' for $life_T = l,\ l+1,\ l+2,\ l+3, \ldots$ As P' indefinitely increases $life_T$, P indefinitely receives token with $cpt_T = l$. ■

2. Ratio for multiples of the size

Proposition 4.1. *$\forall i \in \mathbb{N}^*$ such that i is a multiple of n, we have:*

$$\lim_{r \to +\infty} R_P[i](r) = 1$$

Proof. Let $i = kn$, with $k \in \mathbb{N}^*$.

If P receives a token T such that $cpt_T = i$, then either (i) T is a false-token (resulting from a bad initialization), then P can mark a failure in i, or (ii) T is a real-token, then P is the creator of T and P marks a success in i.

(i) As the ring contains at most n false-tokens, P marks at most n failures in i.

(ii) Lemma 4.1 involves that each process sends an infinite number of tokens with an increasingly large value of life time. Thus, P marks an infinite number of successes in i.

Thus: $\forall i = kn$, $\lim_{r \to +\infty} R_P[i](r) = 1$ ■

3. Ratio for not-multiples of the size

Lemma 4.2. *$\forall i \in \mathbb{N}^*$ such as i is not a multiple of n, we have: when P receives a real-token T such as $cpt_T = i$:*

$$\text{Proba}(P \text{ marks a success in } i) \leq \text{Proba}(P \text{ marks a failure in } i).$$

Proof. Let $i \in \mathbb{N}^*$ be not multiple of n.

If P receives a real-token T with $cpt_T = i$, and as i is not a multiple of n, then T has been created by another process, say Q, thus:

$$\begin{aligned}\text{Proba}(color_P = color_T) &= \text{Proba}(P \text{ and } Q \text{ independently choose the same color}) \\ &= 1/4 + 1/4 = 1/2\end{aligned}$$

Consequently:

$$\begin{aligned}\text{Proba}(P \text{marks a success in } i) &= \text{Proba}(color_P = color_T \wedge cpt_P = cpt_T) \\ &\leq \text{Proba}(color_P = color_T) = 1/2\end{aligned}$$

And so, we have:

$$\text{Proba}(P \text{ marks a success in } i) \leq 1/2 \leq \text{Proba}(P \text{ marks a failure in } i)$$ ■

Proposition 4.2. *$\forall i \in \mathbb{N}^*$ such as i is not a multiple of n, we have:*

$$\lim_{r \to +\infty} Proba\,(R_P[i](r) \geq 0.9) = 0$$

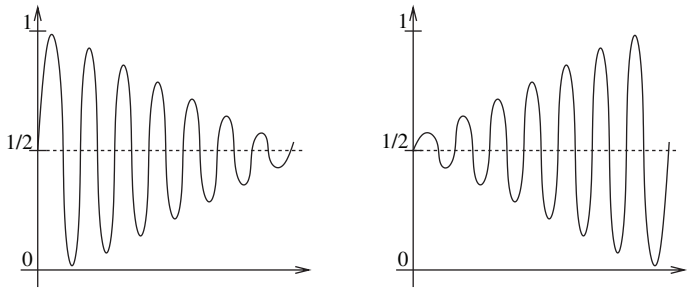

Fig. 4 Variance.

Proof outline.

According to lemma 4.2, if i is not a multiple of n,then the probability for P to mark a failure in i is greater than the probability for P to mark a success in i. Thus the average value of $R_P[i](r)$ is smaller than or equal to 0.5. But the average value is not sufficient to conclude. Indeed, the average value of the two functions drawn in the figure 4 is 0.5, but the variance value of the first function is 0 while the variance value of the second function is $+\infty$. Intuitively, if the ratio is of the form of the first function then the theorem is true, else the theorem is false. Note that, if $R_P[i](r) = 1/2 = 0.5$, then P has marked one success among two marks, and only 8 consecutive successes are enough to set $R_P[i](r)$ above 0.9. But, if $R_P[i](r) = 500/1000 = 0.5$, then P has marked 500 successes among 1000 marks, and 4000 consecutive successes must be marked by P in order to increase $R_P[i](r)$ above 0.9. Thus, when time passes, it is more and more difficult to have $R_P[i](r)$ greater than 0.9.

Using a technique similar to the proof of Bienaymé-Tchebycheff inequality, we prove that Proba$(R_P[i](r) \geq 0.9) \leq 1/8 \cdot (0,4)^3 \cdot N^2$. It is the lemma A.1 given in appendix. This result is more strong that needed for the proposition 4.2, but it will be useful for proposition 4.3 too. The detailed proof of proposition 4.2 is given in appendix: proposition A.1.

The above proposition (4.2) will be used in order to prove the convergence of the algorithm. For the correctness, we need the following proposition (4.3).

Proposition 4.3. *$\forall i \in \mathbb{N}^*$ such as i is not a multiple of n, we have:*

$$\lim_{r \to +\infty} Proba(\forall r' \geq r : R_P[i](r') \geq 0.9) = 0$$

The proof uses a similar argument as for proposition 4.2 and is based on the fact that $(\sum^{\infty} 1/N^2) \sim 1/N$: that's the reason it was needed, in lemma A.1, to obtain a majoring in $1/N^2$ and not $1/N$. The detailed proof is given in appendix: proposition A.2.

4. Conclusion

Theorem 4.1 (Probabilistic convergence). *We note e an execution of $\mathcal{RS}$ that is not maximal.*

$$\lim_{|e| \to +\infty} Proba(\forall P : size_P(\text{last}(e)) = n) = 1;$$

This theorem results from propositions 4.1 and 4.2. We compute a finite multiplication of probabilities to obtain this result. The detailed proof is given in appendix: theorem A.1.

Theorem 4.2 (Probabilistic correctness). *We note e an execution of $\mathcal{RS}$ that is not maximal.*

$$\lim_{|e| \to +\infty} Proba(\forall e' \text{a continuation of } e, \forall P : size_P(\text{last}(ee')) = n) = 1$$

This theorem results from propositions 4.1 and 4.3. We also compute a finite multiplication of probabilities to obtain this result. The detailed proof is given in appendix: theorem A.2.

C. Complexity of the Algorithm

We have computed the time complexity of $\mathcal{RS}$ in the worst case: the stabilization time is in $\max\{O(n^3), O(M^2)\}$, where M is n plus the maximal initial value of any variable of processes. The detailed computation of this complexity is given in the appendix: section B

D. The Observer

In the sequel, we do not make the predefined sequences of the observer explicit (for the sake of simplicity) but we rather describe in a informal way what the observer tries to match. The transcription of this informal observation into formal sequences is given later.

1. Behaviour of the observer.

The observer has two arrays: $S_{\mathcal{O}bs}[\,]$ and $F_{\mathcal{O}bs}[\,]$. $\forall i \geq 1$, $S_{\mathcal{O}bs}[i]$ and $F_{\mathcal{O}bs}[i]$ are initialized to 0. If P is the observed process, then the observer counts in these arrays the number of modifications that P executes in $S_P[\,]$ and $F_P[\,]$. Let $s_P[i]$ and $f_P[i]$ be the initial values of $S_P[i]$ and $F_P[i]$ respectively. Note that the observer never takes into account the values of $s_P[i]$ and $f_P[i]$.

Let n be the size of the ring. If a process P receives a false-token T such that $cpt_T = n$, then P can possibly mark a failure in n, for instance if $color_T \neq color_P$. But that can happen at most n times, because a ring of size n cannot initially contain more than n false-tokens. On the other hand, if P receives a real-token T such that $cpt_T = n$, then P is the creator of T, and marks a success in n. Thus: $\forall r \geq 0, F_P[n](r) \leq f_P[n] + n$.

Since the counter of a token is incremented by 1 at each round, all false-tokens which cause a failure in n will be received during the first n rounds. Then, for each value i, the observer does not take into account the first i tokens arriving at P. For that the observer uses a counter $count_{\mathcal{O}bs}$. This counter is initialized to 0 and is incremented by 1 at each round.

Let P be the observed process. The observer executes:

(a) At each round, $count_{\mathcal{O}bs} := count_{\mathcal{O}bs} + 1$

(b) $\forall i \geq 1$, P marks a success in i and $count_{\mathcal{O}bs} > i \Rightarrow S_{\mathcal{O}bs}[i] := S_{\mathcal{O}bs}[i] + 1$

(c) $\forall i \geq 1$, P marks a failure in i and $count_{\mathcal{O}bs} > i \Rightarrow F_{\mathcal{O}bs}[i] := F_{\mathcal{O}bs}[i] + 1$

The observer announces the stabilization if and only if $size_P$ is such that:

1. $F_{\mathcal{O}bs}[size_P] = 0$ and
2. $\forall i < size_P : F_{\mathcal{O}bs}[i] > 0$ and
3. $S_{\mathcal{O}bs}[size_P] \geq size_P + \alpha$, where α depends on ε.

Clearly, $F_{\mathcal{O}bs}[n] = 0$ is always true. Thus, if $size_P > n$, the second condition is never satisfied. Therefore, the observer cannot announce if $size_P > n$.

If $size_P < n$, the probability for $F_{\mathcal{O}bs}[size_P]$ to satisfy the first condition decreases as the number of $size_P$ tests increases. The third condition forces the observer to wait for $size_P$ to be sufficiently tested before announcing.

This informal description of the observer can be transformed into formal sequences. If P is the observed process, then these sequences are of the form: $s = (a_1, \ldots, a_k)$ such that:

- a_i is an action of $\mathcal{RS}$ for each $i \in [1, \ldots, k]$;
- a_1 is the first action executed from the launch of the observer;
- s contains at least $v + \alpha$ actions $S_P[v] := S_P[v] + 1$;
- s does not contain any action $F_P[v] := F_P[v] + 1$;
- s contains at least one action $F_P[j] := F_P[j] + 1$ for each $j \in [1, \ldots, v[$;
- the last action of s is $size_P := v$.

2. Proof of the observer

Theorem 4.3 (Safety). $\forall \varepsilon \in [0, 1[, \exists \alpha :$ *Proba(false announcement by* $\mathcal{O}bs_\alpha) \leq \varepsilon$

Proof. Let $Prob =$ Proba(false announcement) $\leq \sum_{i=1}^{+\infty}$ Proba(false announcement on i)

Let T be a false token. From round $n + 1$, if T circulates the ring, then $cpt_T > n$. Thus P never marks any failure in n from round $n + 1$. Moreover, according to (a) and (c), the observer only counts the failures marked by P in n

from round $n+1$. Therefore, $F_{\mathcal{O}bs}[n] = 0$ is always true. Then if $size_P > n$ the second condition is never satisfied. Therefore, the observer cannot announce if $size_P > n$ and we have:

if $i > n$, then Proba(false announcement on i) $= 0$

$\Longrightarrow Prob \leq \sum_{i=1}^{n-1}$ Proba(false announcement on i)

If $i < n$ then Proba(false announcement on i) $\leq$ Proba($F_{\mathcal{O}bs}[i] = 0$, with i having been tested at least $i + \alpha$ times) $\leq (1/2)^{i+\alpha}$ (according to the lemma 4.2)

So, we have:

$$Prob \leq \sum_{i=1}^{n-1}(1/2)^{i+\alpha} = (1/2)^{\alpha} * \sum_{i=1}^{n-1}(1/2)^{i} \leq (1/2)^{\alpha}$$

By choosing $\alpha \geq -log\varepsilon/log2$, we obtain: Proba(false announcement) $\leq \varepsilon$ ■

Theorem 4.4 (Liveness). *The observer eventually announces the stabilization with probability* 1.

Proof. Let us prove that: Proba(observer announces on$n$ at the round r) $\xrightarrow{r\to\ +\infty} 1$

First condition: $F_{\mathcal{O}bs}[n] = 0$.

(1) $F_{\mathcal{O}bs}[n] = 0$ is always true.

Second condition: $\forall i < n : F_{\mathcal{O}bs}[i] > 0$

According to corollary 4.1, $\forall i \in \mathbb{N}^* : i < n$, P receives an infinite number of tokens T such that $cpt_T = i$. Moreover, according to lemma 4.2, if P receives a real-token T such that $cpt_T < n$, then the probability for P to mark a success in cpt_T is less than the probability for P to mark a failure in cpt_T. Therefore:

(2) Proba($\forall i < n : F_{\mathcal{O}bs}[i](r) > 0) \xrightarrow{r\to+\infty} 1$

Third condition: $S_{\mathcal{O}bs}[n] \geq n + \alpha$

According to corollary 4.1, P creates at least $2n+\alpha$ tokens with a life time $\geq n$, thus: P marks at least $2n+\alpha$ successes in n and the observer marks at least $n+\alpha$ successes in n. Therefore:

(3) eventually, $(S_{\mathcal{O}bs}[n] \geq n+\alpha)$

According to (1), (2) and (3), we have: Proba(observer announces on n) $\xrightarrow{r\to+\infty} 1$

Thus the observer eventually announces the stabilization with probability 1. ■

The liveness property of the observer is weakened because of probabilistic correctness of algorithm. Indeed, in some executions stabilization is reached, then lost, then reached and so on. What is guaranteed is that the probabilistic measure of such executions is 0. But what should announce an observer with a deterministic liveness condition? Obviously, he could not announce that the algorithm is stabilized. He could not never announce either. Hence the probability liveness condition.

V. Conclusion

In this paper, we introduce the notion of a local and probabilistic observer for self-stabilizing algorithms. Our result is that, if the network is uniform and synchronous, then some problems having a self-stabilizing solution do not have any self-stabilizing solution that can be observed by a local and deterministic observer, but have a self-stabilizing solution that can be observed by a local and probabilistic observer. Computing the size of the ring is not only a particular example, but the first step for extending the probabilistic observation to a larger class of problems. The reason why is that, once the size is known for sure (in fact almost sure), it is easier to observe self-stabilizing snapshots, leading to the observation of more complex stabilizations.

A. Proof of the Algorithm

We introduce here notations we used in order to perform the proof.

In the sequel, we note i a value such as : $i \in \mathbb{N}^*$ and i is not a multiple of n.

Let P be a process. Let $g_{i,k}$ be the *Bernouilli variable*, where k is the number of real-tokens T received by P since the beginning of the execution and such that $cpt_T = i$:

$g_{i,k} = 0$ if and only if P marks a failure when P receives T;

$g_{i,k} = 1$ if and only if P marks a success when P receives T;
Let $p_{i,k}$ be the probability to have $g_{i,k} = 1$. According to the lemma 4.2, $p_{i,k} \leq 1/2$.

Let $M_S[i]$ be the number of (false) successes marked by P in i at the reception of a false-token ($cpt_T = i$) and let $M_F[i]$ be the number of (false) failures marked by P in i at the reception of a false-token ($cpt_T = i$). Let $M[i] = M_S[i] + M_F[i]$.

We claim that: $M[i] \leq n$. Indeed, as the counter of a token is incremented by 1 each round then, for a given false-token T and for a given value i, P received at most one time the token T with $cpt_T = i$. Furthermore, a synchronous ring of size n can contains at most n false-tokens at the initialization. Then, for a given value i, P received at most n false-tokens T with $cpt_T = i$. Thus, $M[i] \leq n$.

Let $s_P[i]$ and $f_P[i]$ be the initial values of $S_P[i]$ and $F_P[i]$.

let r_0 be the first round in which the ring no longer contains any false-token and r be a round such that $r \geq r_0$.

At the round r, if the value i has been tested N times, *i.e.* P has received N real-tokens T such that $cpt_T = i$ since the beginning of the execution, then:

$$S_P[i](r) = s_P[i] + M_S[i] + \sum_{k=1}^{N} g_{i,k} \quad \text{and} \quad F_P[i](r) = f_P[i] + M_F[i] + N - \sum_{k=1}^{N} g_{i,k}$$

Therefore, at the round r, if the value i has been tested N times, then:

$$\frac{S_P[i](r)}{S_P[i](r) + F_P[i](r)} = \frac{s_P[i] + M_S[i] + \sum_{k=1}^{N} g_{i,k}}{s_P[i] + f_P[i] + M[i] + N}$$

We note:

$$L_P[i](N) = \frac{s_P[i] + M_S[i] + \sum_{k=1}^{N} g_{i,k}}{s_P[i] + f_P[i] + M[i] + N} \quad \text{and } Q[i](N) = \frac{\sum_{k=1}^{N} g_{i,k}}{N}$$

As $M[i]$ is finite (and constant) and $s_P[i]$ and $f_P[i]$ are constants, the behaviours of $L_P[i](N)$ and $Q[i](N)$ are the same when $N \to +\infty$, so we analyse Q_N.

Lemma A.1. *$\forall i \in \mathbb{N}^*$ such as i is not a multiple of n and $\forall N \in \mathbb{N}^*$:*

$$Proba(Q[i](N) \geq 0.9) \leq \frac{1}{(0.4)^3 .8.N^2}$$

Proof. For more information about formulas of probabilities we have used in the proof, see for instance[20].

In this proof, we note i a value such as : $i \in \mathbb{N}^*$ and i is not a multiple of n.

We note $\mathbb{E}(Q[i](N))$, the average of $Q[i](N)$.

$$\mathbb{E}(Q[i](N)) = \mathbb{E}\left(\frac{\sum_{k=1}^{N} g_{i,k}}{N}\right) = \frac{1}{N}\sum_{k=1}^{N} \mathbb{E}(g_{i,k}) = \frac{1}{N}\sum_{k=1}^{N} p_{i,k} \leq \frac{1}{N}\frac{N}{2} = \frac{1}{2}$$

In the sequel, we use a technique similar of the proof of the inequality of Bienaymé-Tchebycheff, but with a moment of order 3.

$$\begin{aligned} \text{Proba}(Q[i](N) \geq 0.9) &= \text{Proba}(Q[i](N) - 1/2 \geq 0.4) \\ &\leq \text{Proba}\left(|Q[i](N) - 1/2| \geq 0.4\right) \\ &= \text{Proba}\left(|Q[i](N) - 1/2|^3 \geq (0.4)^3\right) \end{aligned}$$

Let $f = |Q[i](N) - 1/2|^3$; f takes its value in a set I.

Let $A = \{x \in I : (f = x) \wedge (f \geq (0.4)^3)\}$, $B = \left\{x \in I : (f = x) \wedge \left(f < (0.4)^3\right)\right\}$ and p_x be the probability to have $f = x$.

By definition, we have:

$$\mathbb{E}(f) = \sum_{x \in I} x \cdot p_x = \sum_{x \in A} x \cdot p_x + \sum_{x \in B} x \cdot p_x$$

Thus,

$$\begin{aligned}
\mathbb{E}(f) &\geq \sum_{x \in A} x \cdot p_x \text{ since all values are positive} \\
&\geq (0.4)^3 \sum_{x \in A} p_x \text{ since } x \in A \Longrightarrow x \geq (0.4)^3 \\
&\geq (0.4)^3 \cdot \text{Proba}(f \geq (0.4)^3) \text{ since } \sum_{x \in A} p_x = \text{Proba}(f \geq (0.4)^3) \text{ by definition}
\end{aligned}$$

Thus, we obtain:

$$\text{Proba}(Q[i](N) \geq 0.9) \leq \text{Proba}(f \geq (0.4)^3) \leq \mathbb{E}(f)/(0.4)^3 = \mathbb{E}(|Q[i](N) - 1/2|^3)/(0.4)^3$$

$$\begin{aligned}
\mathbb{E}\left(\left|Q[i](N) - \frac{1}{2}\right|^3\right) &= \mathbb{E}\left(\left|\frac{1}{N}\sum_{k=1}^{N} g_{i,k} - \frac{1}{2}\right|^3\right) = \mathbb{E}\left(\frac{1}{N^3}\left|\sum_{k=1}^{N} g_{i,k} - \frac{N}{2}\right|^3\right) \\
&= \frac{1}{N^3}\,\mathbb{E}\left(\left|\sum_{k=1}^{N}\left(g_{i,k} - \frac{1}{2}\right)\right|^3\right) && \text{using the linearity of the average} \\
&\leq \frac{1}{N^3}\,\mathbb{E}\left(\sum_{k=1}^{N}\left|g_{i,k} - \frac{1}{2}\right|^3\right) && \text{applying the triangular inequality to the cube of the sum and the fact that the average is a growing function;} \\
&\leq \frac{1}{N^3}\sum_{k=1}^{N}\left(\mathbb{E}\left|g_{i,k} - \frac{1}{2}\right|^3\right) && \text{using the linearity of the average} \\
&= \frac{1}{N^3}\sum_{k=1}^{N}\frac{1}{8} = \frac{1}{8N^2} && \text{as } 1/8 \text{ is the moment of order 3 of a Bernouilli variable of parameter } 1/2
\end{aligned}$$

Therefore,

$$\text{Proba}(Q[i](N) \geq 0.9) \leq \frac{1}{(0.4)^3 \cdot 8 \cdot N^2}$$

■

Proposition A.1. *$\forall i \in \mathbb{N}^*$ such as i is not a multiple of n, we have*:

$$\lim_{r \to +\infty} \textit{Proba}\,(R_P[i](r) \geq 0.9) = 0$$

Proof. In this proof, we note i a value such as : $i \in \mathbb{N}^*$ and i is not a multiple of n.

Let P be a process receiving a token $T = \textit{Token}(\textit{color}_T, \textit{cpt}_T, \textit{life}_T)$ with $\textit{cpt}_T = i$.

According to lemma A.1, we have:

$$\text{Proba}(Q[i](N) \geq 0.9) \leq \frac{1}{(0.4)^3.8.N^2}$$

$$\frac{1}{(0.4)^3.8.N^2} \xrightarrow{N\to+\infty} 0 \Longrightarrow \text{Proba}(Q[i](N) \geq 0.9) \xrightarrow{N\to+\infty} 0$$

$$\Longrightarrow \text{Proba}(L_P[i](N) \geq 0.9) \xrightarrow{N\to+\infty} 0$$

According to corollary 4.1, the number of tokens received by P increases indefinitely with the round, thus N goes to infinity with r and we can conclude:

$$\text{Proba}\left(\frac{S_P[i](r)}{S_P[i](r) + F_P[i](r)} \geq 0.9\right) \xrightarrow{r\to+\infty} 0$$

■

Proposition A.2. $\forall i \in \mathbb{N}^*$ *such as* i *is not a multiple of* n*, we have*:

$$\lim_{r\to+\infty} Proba(\forall r' \geq r\colon R_P[i](r') \geq 0.9) = 0$$

Proof. In this proof, we note i a value such as : $i \in \mathbb{N}^*$ and i is not a multiple of n.

Let P be a process receiving a token $T = Token(color_T, cpt_T, life_T)$ with $cpt_T = i$.

According to lemma A.1, we have:

$$\text{Proba}\,(Q[i](N) \geq 0.9) \leq \frac{1}{(0.4)^3.8.N^2}$$

$$\text{Proba}(\forall M \geq N : Q[i](M) \geq 0.9) \leq \sum_{M=N}^{+\infty} \text{Proba}(Q[i](M) \geq 0.9)$$

$$\leq \frac{1}{(0.4)^3.8} \sum_{M=N}^{+\infty} \frac{1}{M^2} \sim \frac{1}{(0.4)^3.8.N} \xrightarrow{N\to+\infty} 0$$

$$\text{Proba}(\forall M \geq N\colon Q[i](M) \geq 0.9) \xrightarrow{N\to+\infty} 0 \Longrightarrow \text{Proba}(\forall M \geq N\colon L_P[i](M) \geq 0.9) \xrightarrow{N\to+\infty} 0$$

Again according to corollary 4.1, we have N and r goes to infinity together.

$$\Longrightarrow \text{Proba}\left(\forall r' \geq r\colon \frac{S_P[i](r')}{S_P[i](r') + F_P[i](r')} \geq 0.9\right) \xrightarrow{r\to+\infty} 0$$

■

Theorem A.1 (Probabilistic convergence). *We note* e *an execution of* $\mathcal{RS}$ *that is not maximal.*

$$\lim_{|e|\to+\infty} Proba(\forall P : size_P(\text{last}(e)) = n) = 1;$$

Proof. We note $P_1, \ldots, P_n$, processes of the ring.

The size of the ring is therefore equal to n and a process P will choose this value as soon as $R_P[n] \geq 0.9$ and $R_P[i] < 0.9$ for all i less than n. Thus:

$$\begin{aligned}
&\text{Proba}(\forall \textit{ process } P : \textit{size}_P(r) = n) \\
&= \text{Proba}(\forall k \in [1,n]\text{: } (\forall i \in [1, n-1] : R_{P_k}[i](r) < 0.9) \text{ and } (R_{P_k}[n](r) \geq 0.9)) \\
&\leq \left(\prod_{k=1}^{n} \text{Proba}(\forall i \in [1, n-1]\text{: } R_{P_k}[i](r) < 0.9)\right) * \left(\prod_{k=1}^{n} \text{Proba}(R_{P_k}[n](r) \geq 0.9)\right)
\end{aligned}$$

Let P be a process of the ring. According to proposition 4.2, we have:

$$\forall i \in [1, n-1]\text{: } \forall \varepsilon > 0, \exists R, \forall r > R :: \text{Proba}(R_P[i](r) < 0.9) > 1 - \varepsilon$$

Thus the result follows as finite product of limit. More precisely:

$$\forall \varepsilon_i > 0 \ (i \in [1, n-1]), \exists R = \max\left(\bigcup_{i=1}^{n-1} \{R_i\}\right), \forall r > R : \prod_{i=1}^{n-1} \text{Proba}(R_P[i](r) < 0.9) > \prod_{i=1}^{n-1} (1 - \varepsilon_i)$$

Moreover:

$$\begin{aligned}
&\forall \varepsilon > 0, \exists \varepsilon_1 > 0, \ldots, \exists \varepsilon_{n-1} > 0 : \prod_{i=1}^{n-1} (1 - \varepsilon_i) > 1 - \varepsilon \\
&\Longrightarrow \textbf{(i)} \ \ \forall \varepsilon_k > 0, \exists R_k, \forall r > R_k\text{: } \prod_{i=1}^{n-1} \text{Proba}(R_{P_k}[i](r) < 0.9) > 1 - \varepsilon_k \\
&\Longleftrightarrow \text{Proba}(\forall i \in [1, n-1] : R_{P_k}[i](r) < 0.9) \xrightarrow{r \to +\infty} 1
\end{aligned}$$

According to proposition 4.1, we have:

$$\textbf{(ii)} \ \ \forall \varepsilon'_k > 0, \exists R'_k, \forall r > R'_k\text{: } \text{Proba}(R_{P_k}[n](r) \geq 0.9) > 1 - \varepsilon'_k$$

According to (i) and (ii), we have:

$$\begin{aligned}
&\forall \varepsilon_k > 0 \ (k \in [1,n]), \forall \varepsilon'_k > 0 (k \in [1,n]), \exists R = \max(\cup_{k=1}^{n} \{R_k, R'_k\}), \forall r > R\text{:} \\
&\left(\prod_{k=1}^{n} \text{Proba}(\forall i \in [1, n-1]\text{: } R_{P_k}[i](r) < 0.9)\right) * \left(\prod_{k=1}^{n} \text{Proba}(R_{P_k}[n](r) \geq 0.9)\right) \\
&> \left(\prod_{k=1}^{n} (1 - \varepsilon_k)\right) * \left(\prod_{k=1}^{n} (1 - \varepsilon'_k)\right)
\end{aligned}$$

Moreover:

$$\begin{aligned}
&\forall \varepsilon > 0, \exists \varepsilon_1 > 0, \ldots, \exists \varepsilon_n > 0, \exists \varepsilon'_1 > 0, \ldots, \exists \varepsilon'_n > 0\text{: } \prod_{k=1}^{n} (1 - \varepsilon_k) * \prod_{k=1}^{n} (1 - \varepsilon_k) > 1 - \varepsilon \\
&\Longrightarrow \forall \varepsilon > 0, \exists R, \forall r > R : \text{Proba}(\forall \text{ process} P\text{: } \textit{size}_P(r) = n) > 1 - \varepsilon \\
&\Longleftrightarrow \text{Proba}(\forall \text{ process } P : \textit{size}_P(r) = n) \xrightarrow{r \to +\infty} 1
\end{aligned}$$

if e is an execution of $\mathcal{RS}$ that is not maximal and if $\textit{last}(e)$ is the configuration reached at the round r, then r goes to infinity with $|e|$ and we can conclude:

$$\lim_{|e| \to +\infty} \text{Proba}(\forall P : \textit{size}_P(\textit{last}(e)) = n) = 1$$

■

Theorem A.2 (Probabilistic correctness). *We note e an execution of $\mathcal{RS}$ that is not maximal.*

$$\lim_{|e|\to+\infty} \mathit{Proba}(\forall e' \textit{ a continuation of } e, \forall P : \mathit{size}_P(\text{last}(ee')) = n) = 1$$

Proof. We note $P_1, \ldots, P_n$, processes of the ring.

$$\begin{aligned}\text{Proba}(\forall r' \geq r, \forall \textit{ process } P : \mathit{size}_P(r') = n) = \text{Proba}(\forall k \in [1,n]\text{: } (\forall i \in [1,n-1], \forall r' \geq r\text{: } R_{P_k}[i](r') < 0.9) \text{ and}\\ (\forall r' \geq r\text{: } R_{P_k}[n](r') \geq 0.9))\\ = \left(\prod_{k=1}^{n} \text{Proba}(\forall i \in [1,n-1], \forall r' \geq r : R_{P_k}[i](r') < 0.9)\right)\\ * \left(\prod_{k=1}^{n} \text{Proba}(\forall r' \geq r\text{: } R_{P_k}[n](r') \geq 0.9)\right)\end{aligned}$$

Let P be a process of the ring. According to proposition 4.3, we have:

$$\begin{aligned}&\forall i \in [1,n-1] : \ \forall \varepsilon > 0, \exists R, \forall r > R :: \text{Proba}(\forall r' \geq r : R_P[i](r') < 0.9) > 1-\varepsilon\\ &\quad \Longrightarrow \forall \varepsilon_i > 0 \ (i \in [1,n-1]), \exists R = \max(\cup_{i=1}^{n-1}\{R_i\}), \forall r > R :\\ &\qquad \prod_{i=1}^{n-1} \text{Proba}\left(\forall r' \geq r\text{: } R_P[i](r') < 0.9\right) > \prod_{i=1}^{n-1}(1-\varepsilon_i)\end{aligned}$$

Moreover:

$$\begin{aligned}&\forall \varepsilon > 0, \exists \varepsilon_1 > 0, \ldots, \exists \varepsilon_{n-1} > 0 : \prod_{i=1}^{n-1}(1-\varepsilon_i) > 1-\varepsilon\\ &\quad \Longrightarrow \textbf{(i)} \ \ \forall \varepsilon_k > 0, \exists R_k, \forall r > R_k\text{: } \prod_{i=1}^{n-1} \text{Proba}(\forall r' \geq r\text{:} R_{P_k}[i](r') < 0.9) > 1-\varepsilon_k\\ &\quad \Longleftrightarrow \text{Proba}(\forall i \in [1,n-1], \forall r' \geq r\text{: } R_{P_k}[i](r') < 0.9) \xrightarrow{r\to+\infty} 1\end{aligned}$$

According to proposition 4.1, we have:

$$\textbf{(ii)} \ \ \forall \varepsilon'_k > 0, \exists R'_k, \forall r > R'_k\text{: Proba}(\forall r' \geq r\text{: } R_{P_k}[n](r') \geq 0.9) > 1-\varepsilon'_k$$

According to (i) and (ii), we have:

$$\begin{aligned}&\forall \varepsilon_k > 0 (k \in [1,n]), \forall \varepsilon'_k > 0 (k \in [1,n]), \exists R = \max(\cup_{k=1}^{n}\{R_k, R'_k\}), \forall r > R :\\ &\left(\prod_{k=1}^{n} \text{Proba}(\forall i \in [1,n-1], \forall r' \geq r\text{: } R_{P_k}[i](r') < 0.9)\right) * \left(\prod_{k=1}^{n} \text{Proba}(\forall r' \geq r\text{: } R_{P_k}[n](r') \geq 0.9)\right)\\ &\quad > \left(\prod_{k=1}^{n}(1-\varepsilon_k)\right) * \left(\prod_{k=1}^{n}(1-\varepsilon'_k)\right)\end{aligned}$$

Moreover:

$$\forall \varepsilon > 0, \exists \varepsilon_1 > 0, \ldots, \exists \varepsilon_n > 0, \exists \varepsilon'_1 > 0, \ldots, \exists \varepsilon'_n > 0 : \prod_{k=1}^{n}(1-\varepsilon_k) * \prod_{k=1}^{n}(1-\varepsilon'_k) > 1-\varepsilon$$

$$\Longrightarrow \forall \varepsilon > 0, \exists R, \forall r > R : \text{Proba}(\forall r' \geq r, \forall \text{ process} P : size_P(r') = n) > 1-\varepsilon$$

$$\Longleftrightarrow \text{Proba}(\forall r' \geq r, \forall \text{ process} P : size_P(r') = n) \xrightarrow{r \to +\infty} 1$$

If e is an execution of $\mathcal{RS}$ that is not maximal and if $last(e)$ is the configuration reached at the round r, then r goes to infinity with $|e|$ and we can conclude:

$$\lim_{|e| \to +\infty} \text{Proba}(\forall e' \quad \text{a continuation of } e, \ \forall P : size_P(last(ee')) = n) = 1$$

■

B. Complexity of the Algorithm

In this section, we compute the time complexity of $\mathcal{RS}$ in the worst case. In order to compute this complexity, we need in a first time, to evaluate the time between two receptions and between two creations of a token.

Creation:

If $\forall P$, $life_P \geq n$ then $\forall P$, P receives successive tokens from a given process Q with at least n rounds between these two receptions. Thus, in an interval of n rounds, P receives at least $n-1$ tokens from other processes. Then, as soon as the life of its last created token expires (*i.e.* it disappears from the ring), P must wait at most n rounds before creating another token. Consequently, in "persistent phase" P creates a token every "$n + life_P$" rounds.

If $\exists P$,$life_P < n$ then, in an interval of at most n rounds, at least one process "progresses", *i.e.* creates a token increasing its life by 1: in less than n rounds, the token of P disappears ; it is then in process Q and so Q is "free" at this round, *i.e.* it has no token to forward ; if $cpt_Q \geq life_Q$ then Q creates a token and progresses ; else it does not send anything and the successor of $Q$is free at the next round, thus the creation is made at the latest by P (after n rounds) by induction on the distance between Q and P.

Consequently, after about $O(n^3)$ rounds in the worst case, we have: $\forall P$, $life_P \geq n$.

P is stabilized if at least the following property holds: $n = \inf\{i : R_P[i] \geq 0.9\}$ and still holds forever. We are going to compute the time needed to reach the situation where the property holds (time needed to "forever" verify the property is of the same order).

Time needed to have $\forall P$, $R_P[n] \geq 0.9$:

A process can mark at most n failures in n during the first n rounds of an execution, and then it only marks successes in n.

$R_P[n] \geq 0.9 \Leftrightarrow S_P[n] \geq 0.9\,(S_P[n] + F_P[n]) \Leftrightarrow S_P[n] \geq 9\,F_P[n]$ which is verified if $S_P[n] \geq 9M$, where M is n plus the maximal initial value of any variable of processes.

After about $O(n^3)$ rounds in the worst case, $life_P \geq n$. Let us assume that P progresses then from A to $A+h$, where A is the maximum between n (if $life_P$ is initially lesser than n) and M (else). Since we study the worst case, and since M is greater that n, we assume that $A = M$. For each $life_P = k$ between M and $M+h$, the time D_1 needed for this progression is the life plus at most n rounds, then:

$$D_1 = \sum_{k=M}^{k=M+h} (k+n) = \frac{1}{2}(h+1)(2M+h) + n(h+1)$$

A process marks exactly one success in n for every token it creates with a life greater than or equal to n. Thus, P marks h successes in n during its progression between M and $M+h$. But h must be greater that $9M$, thus the order of h is $O(M)$ and then the order of D_1 is $O(M^2)$.

Consequently, the time needed to have $\forall P$, $R_P[n] \geq 0.9$ is a $O(n^3) + O(M^2)$.

Time needed to have $\forall P, \forall i < n, R_P[i] < 0.9$:

P marks a success or a failure in $i < n$ only if it receives a token. This token necessarily comes from process P_i at distance i. After about $O(n^3)$ rounds in the worst case, processes are in the "persistent phase". Thus P_i waits at most n rounds before creating a token since its last token has disappeared, and then P receives and marks in i once every '$n + life_{P_i}$" rounds, for all i.

Let us consider $S_P[i]/S_P[i] + F_P[i] < 0.9 \quad \Leftrightarrow S_P[i] < 9\,F_P[i]$ with Proba(P marks a success in i) = Proba(P marks a failure in i) $= 1/2$

In a first time, we consider that variables $S_P[i]$ and $F_P[i]$ are initialized to 0, $\forall i$. In this case, we say that "there is stabilization in x rounds at i", if $S_P[i]/S_P[i] + F_P[i] < 0.9$ after x rounds but not before. We obtain the following probabilities for the stabilization in x rounds at i:

x	1	2	3	...	9	10	11	12	...
Proba	$1/2$	$(1/2)^2$	$(1/2)^3$	...	$(1/2)^9$	0	$(1/2)^{11}$	$(1/2)^{12} * 2$	...

Then the average number of rounds before stabilization is $\mathbb{E}(stabi) \leq \sum_{k=1}^{+\infty} k(1/2)^k = 2$

If we change the definition of "there is stabilization in x rounds at i" by $R_P[i] < 0.9$ after x rounds but not before, and during 2 rounds, or during 3 rounds, ..., then $\mathbb{E}(stabi)$ is a little bit greater but still bounded.

Now, we consider that variables $S_P[i]$ and $F_P[i]$ may not be initialized to 0, $\forall i$. In the worst case, we have M bad successes in i due to the initialization of $S_P[i]$ and to the n false tokens. Thus, we must have: $S_P[i] + M/S_P[i] + F_P[i] + M < 0.9 \Leftrightarrow S_P[i]+M < 9\,F_P[i]$. We obtain then an average number of rounds before stabilization in at most $O(2 + M)$.

Consequently, P must arrive in the "persistent phase" (in $O(n^3)$ rounds), then marks $M + 2$ times in i which is done in $O(M^2)$ rounds. Thus the time needed to have $\forall P, \forall i < n, R_P[i] < 0.9$ is in $O(n^3) + O(M^2)$.

Acknowledgments

We would like to thank anonymous referees for their valuable comments that helped to improve this paper.

Conclusion:

The stabilization time of $\mathcal{RS}$ is in $\max\{O(n^3), O(M^2)\}$, where M is n plus the maximal initial value of any variable of processes.

References

[1]Dijkstra, E. W., "Self-stabilizing Systems in Spite of Distributed Control," *Communications of the ACM*, Vol. 17, No. 11, Nov. 1974, pp. 643–644.

[2]Dolev, S., *Self-Stabilization*, MIT Press, Cambridge, MA, 2000, p. 208.

[3]Tel, G., *Introduction to Distributed Algorithms*, Cambridge Univ. Press, Cambridge, 1994.

[4]Chandy, K. M. and Lamport, L., "Distributed Snapshots: Determining Global States of Distributed Systems," *ACM Transactions on Computer Systems (TOCS)*, Vol. 3, No. 1, Feb. 1985, pp. 63–75.

[5]Garg, V. K. and Mitchell, J. R., "Distributed Predicate Detection in a Faulty Environment," *International Conference on Distributed Computing Systems*, 1998, pp. 416–423.

[6]Garg, V. K., "Observation of Global Properties in Distributed Systems," *SEKE*, 1996, pp. 418–425.

[7]Garg, V. K., "Observation and Control for Debugging Distributed Computations," *AADEBUG*, 1997, pp. 1–12.

[8]Gärtner, F. C. and Pleisch, S., "(Im)Possibilities of Predicate Detection in Crash-Affected Systems," *WSS 2001*, Vol. 2194 of *LNCS*, 2001, pp. 98–113.

[9]Gärtner, F. C. and Pleisch, S., "Failure Detection Sequencers: Necessary and Sufficient Information about Failures to Solve Predicate Detection," *DISC*, Springer, 2002, pp. 280–294.

[10]Katz, S. and Perry, K. J., "Self-stabilizing Extensions for Message-Passing Systems," *Distributed Computing*, Vol. 7, No. 1, 1993, pp. 17–26.

[11]Lai, T. H. and Yang, T. H., "On Distributed Snapshots," *Information Processing Letters*, Vol. 25, No. 3, 29 May 1987, pp. 153–158.

[12]Mittal, N., Freiling, F. C., Venkatesan, S., and Penso, L. D., "Efficient Reduction for Wait-Free Termination Detection in a Crash-Prone Distributed System," *DISC*, Lecture Notes in Computer Science, Springer-Verlag, 2005, pp. 93–107.

[13]Prakash, R. and Singhal, M., "Maximal Global Snapshot with Concurrent Initiators," *Proceedings of the 6th Symposium on Parallel and Distributed Processing*, IEEE Computer Society Press, Los Alamitos, CA, USA, Oct. 1994, pp. 344–351.

[14]Lin, C. and Simon, J., "Observing Self-Stabilization," *Proceedings of the 11th Annual Symposium on Principles of Distributed Computing*, edited by M. Herlihy, ACM Press, Vancouver, BC, Canada, Aug. 1992, pp. 113–124.

[15]Beauquier, J., Pilard, L., and Rozoy, B., "Observing Locally Self-stabilization," *Journal of High Speed Networks*, Vol. 14, No. 1, 2004, pp. 3–19.

[16]Beauquier, J., Pilard, L., and Rozoy, B., "Observing Locally Self-stabilization in a Probabilistic Way," *DISC'05: 19th International Symposium on Distributed Computing*, 2005, pp. 399–413.

[17]Segala, R. and Lynch, N., "Probabilistic Simulations for Probabilistic Processes," *CONCUR '94: Concurrency Theory, 5th International Conference*, edited by B. Jonsson and J. Parrow, Vol. 836 of *Lecture Notes in Computer Science*, Springer-Verlag, Uppsala, Sweden, 22–25Aug. 1994, pp. 481–496.

[18]Segala, R., "Modeling and Verification of Randomized Distributed Real-time Systems," Ph.D. thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1995.

[19]Herman, T., "Probabilistic Self-stabilization," *Information Processing Letters*, Vol. 35, No. 2, 29 June 1990, pp. 63–67.

[20]Cormen, T. H., Leiserson, C. E., and Rivest, R. L., *Introduction to Algorithms*, MIT Press and McGraw-Hill Book Company, 6th ed., 1992.

Shlomi Dolev
Associate Editor