Observing locally self-stabilization in a probabilistic way

Joffroy Beauquier, Laurence Pilard, and Brigitte Rozoy

Laboratoire de Recherche en Informatique - CNRS
Université Paris-Sud - Bât. 490
91405 Orsay Cedex, France
beauquier@lri.fr - (0)1.69.15.67.96
pilard@lri.fr (0)1.69.15.66.34
rozoy@lri.fr (0)1.69.15.66.09

Regular Track

Abstract

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 an unique node. This entity is not allowed to detect false stabilization, must eventually detect that stabilization is reached, and must not interfere with the observed algorithm.

We introduce here the notion of probabilistic observer which realizes the conditions above only 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.

1 Introduction

The notion of self-stabilization was introduced by Dijkstra [Dij74]. He defined an algorithm 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 [Dol00,Tel94].

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 [Dol00].

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

The only studies dealing with the second issue are the important paper by Lin and Simon [LS92] and the paper [BPR04]. 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).

By restricting their attention to ring networks, Lin and Simon in [LS92] 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 [BPR04], 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 [BPR04], 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 introduce 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.

2 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 [Tel94] and the notion of a local observer of [BPR04]. We recall the definition of a deterministic observer and we define a probabilistic observer.

Distributed algorithm. A distributed algorithm $\mathcal{A} = (C, A)$ is an automaton, where C is the set of all states (called *configurations*) of $\mathcal{A}$ and A is the set of all transitions (called *actions*) of $\mathcal{A}$. An execution of $\mathcal{A}$, noted $e = c_1 a_1 c_2 a_2 \ldots$ is a maximal sequence of configurations and actions of $\mathcal{A}$ such that c_{i+1} is reached from c_i by the execution of the action a_i. The sequence is maximal if it is either infinite, or it is finite but no action of $\mathcal{A}$ is enabled in the last configuration. All computations considered in this paper are assumed to be maximal.

An algorithm $\mathcal{A}$ is self-stabilizing for a specification $\mathcal{SP}$ if and only if there exists a sub-set $\mathcal{C}_{\mathcal{L}}$ of the set of the configurations of $\mathcal{A}$ such that: (i) every execution of $\mathcal{A}$ contains at least one configuration in $\mathcal{C}_{\mathcal{L}}$, (ii) every execution of $\mathcal{A}$ with an initial configuration in $\mathcal{C}_{\mathcal{L}}$ verifies $\mathcal{SP}$ and $\mathcal{C}_{\mathcal{L}}$ is closed. We call $\mathcal{C}_{\mathcal{L}}$ the set of legitimate configurations of $\mathcal{A}$.

We use the definition of a probabilistic self-stabilizing algorithm defined in [BGJ99].

A probabilistic algorithm $\mathcal{A}$ is self-stabilizing for a specification $\mathcal{SP}$ if and only if there exists a sub-set $\mathcal{C}_{\mathcal{L}}$ of the set of the configurations of $\mathcal{A}$ such that: (i) the probability for an execution to reach a configuration in $\mathcal{C}_{\mathcal{L}}$ is equal to 1, (ii) the probability for an execution from a configuration in $\mathcal{C}_{\mathcal{L}}$ to stay in $\mathcal{C}_{\mathcal{L}}$ is equal to 1.

Observer. As it is described in [BPR04], 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 stablilization.

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 21 (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 and liveness conditions. A probabilistic observer is defined with a parameter α.

Definition 22 (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 announces eventually the stabilization with probability 1; (safety) $\forall \varepsilon \in]0,1], \exists \alpha$ used by the observer: Proba(correct announcement by $\mathcal{O}_\alpha$) $> 1 - \varepsilon$.*

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.

3 Impossibility result with a deterministic observer

In [BPR04], 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. (Synchronous means that a computation proceeds by rounds. In each round, each process is activated. Anonymous means that processes are indistinguishable: no id's, same code).

Theorem 31 *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.
Proof by contradiction.
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 sees 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 $\mathcal{A}$.
□

4 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 3, 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 and observable in a probabilistic way. In the first part, we describe this algorithm, then its probabilistic observer.

4.1 The algorithm

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

Specification of the algorithm

The algorithm is probabilistic as it is defined in [BGJ99] and verifies the following specification $\mathcal{SP}$: (i) each process knows eventually the size of the ring with probability 1, and (ii) 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 $\mathcal{C}_\mathcal{L}$ of $\mathcal{RS}$ are the configurations of $\mathcal{RS}$ in which: $\forall P, size_P = n$. An execution e of $\mathcal{RS}$ satisfies the specification $\mathcal{SP}$ if and only if e has a suffix containing only legitimate configurations.
Let E be the set of all executions of $\mathcal{RS}$, e be an element of E and r_e be a round of e. Let P be a process and let $size_P(r_e)$ be the value of $size_P$ at round r_e.
Convergence : the probability for an execution to reach a legitimate configuration is equal to 1: $\forall e \in E, \text{Proba}(\forall P : size_P(r_e) = n) \xrightarrow{r_e \to +\infty} 1$;
Correctness : the probability for an execution from a legitimate configuration to stay in a legitimate configuration is equal to 1:
$\forall e \in E, \text{Proba}(\forall r'_e > r_e, \forall P : size_P(r'_e) = n) \xrightarrow{r_e \to +\infty} 1$

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 **period of life** to T in $life_T$. Then P saves this period in $life_P$. Now P has to wait the end of this period for creating another token. To do this, the process has a counter cpt_P such that: at the begining 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 period of life of the last token created by P is over. Furthermore, every time P creates a token, the assigned period of life of the token is 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 its counter is incremented by 1. A process relays all tokens which have not exhausted their periods of life ($cpt_T < life_T$). Thus, eventually all token disappears and eventually two distincts tokens created by P cannot be in the ring at 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. $color_P$: the color of the last token created by P;
2. cpt_P: 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. $life_P$: the period of life of the last token created by P. When P creates a token, $life_P$ is incremented by 1;
4. $size_P$: 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. $color_T$: the color of the token (constant during the life of T);

2. cpt_T: 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. $life_T$: the period of life 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.

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 $F_P[i] := F_P[i] + 1$.

Finally, 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 algorithm executed by processes. Each round a process executes the procedure *Compute-Size()*.

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 $T(\bullet, 0, 12)$. 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}
$\text{cpt}_P := 0$
$\text{life}_P := \text{life}_P + 1$
Send Token(color_P, 1, life_P) to the successor of P

Procedure Compute-Size() =

$\text{cpt}_P := \text{cpt}_P + 1$
m_P := value of the received channel of P

Part 1: P receives a token T.
if $\text{m}_P = \text{Token}(\text{color}_T, \text{cpt}_T, \text{life}_T)$ **then**
(1) P marks a success or a failure
if $\text{cpt}_P = \text{cpt}_T \wedge \text{color}_P = \text{color}_T$ **then** $\text{S}_P[\text{cpt}_T] := \text{S}_P[\text{cpt}_T] + 1$
else $\text{F}_P[\text{cpt}_T] := \text{F}_P[\text{cpt}_T] + 1$

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

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

Fig. 1. The algorithm

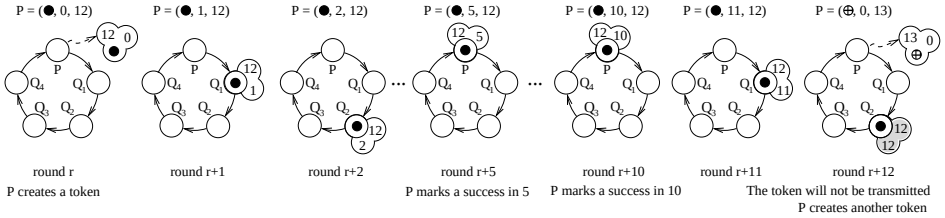

Fig. 2. behaviours of successes and failures for multiples of the size ($\oplus$ represents the white color).

Ratio for multiples of the size. 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

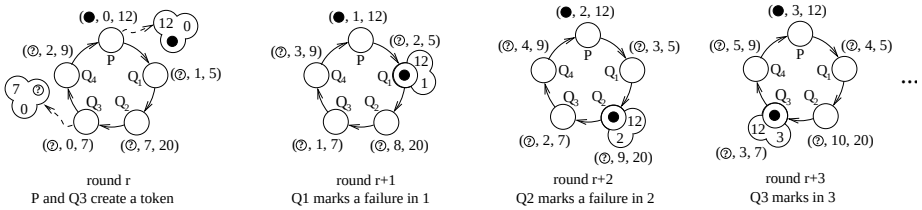

Fig. 3. behaviours of successes and failures for not multiples of the size (? represents black or white).

a real-token T with $cpt_T = kn$, then T has performed k entire traversals of 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 that in the ring, 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.

P creates an infinite number of tokens. Each of them are recognized by P when they return to P after an entire traversal of the ring. Thus, P marks an infinite number of successes in n. Furthermore, we have seen that P marks at most n failures in n. Thus, for all process P we have:

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

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.

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 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 $cpt_T = cpt_P$. P and Q choosing their color 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, we have:

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

Thus, for all process P and $\forall i \in \mathbb{N}^{*+}$ such that i is not a multiple of n, 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.

4.2 Proof of the algorithm

In the proof, we use the following notation: $R_P[i] = \dfrac{S_P[i]}{S_P[i] + F_P[i]}$

Progression of each process

Lemma 41 *Let P be a process. For each round r and for each $l \geq 1$: if* $\text{cpt}_P(r) = \text{life}_P(r) = l$, *then* $\exists\, r' \geq r$ *such that* $\text{life}_P(r') = l + 1$,

Proof.

By contradition.

Let r_0 be the first round during which the ring does not contain any false token. Note that all tokens have a bounded life, thus eventually all false tokens disappear.

Let P be a process such that $cpt_P(r_0) = life_P(r_0) = l$. Let us suppose that $\forall r \geq r_0, life_P(r) = life_P(r_0)$. 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. $1 \leq |\mathbb{P}_\infty| \leq n-1$ and $1 \leq |\mathbb{P}_{\not\infty}| \leq n-1$.
Let $r_1 \geq r_0$ be the last round where a process in $\mathbb{P}_{\not\infty}$ creates a token.
If $Q \in \mathbb{P}_\infty$, Q creates an infinite number of tokens, thus we have:
$\forall i, \exists r_Q \geq r_1, \forall r \geq r_Q : \mathit{life}_Q(r) > i$
$\Rightarrow \exists r_Q \geq r_1, \forall r \geq r_Q : \mathit{life}_Q(r) > n^n + 1$
$\Rightarrow \exists r \geq r_1, \forall r' \geq r, \forall Q \in \mathbb{P}_\infty : \mathit{life}_Q(r') > n^n + 1$
Let $r_2 \geq r_1$ be a round such as : $\forall r \geq r_2, \forall Q \in \mathbb{P}_\infty : \mathit{life}_Q(r) > n^n + 1$
Between rounds r_2 and $r_2 + n^n$, we have: $\forall P \in \mathbb{P}_{\not\infty}$, P does not create any token, 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 41 *$\forall P$ a process:*

1. *$\forall i \in \mathbb{N}^{*+}$, P creates an infinite number of tokens T such that* $\mathrm{life}_T \geq i$
2. *$\forall i \in \mathbb{N}^{*+}$, P receives an infinite number of tokens T such that* $\mathrm{cpt}_T = i$.

Ratio for multiples of the size

Theorem 41 *$\forall i \in \mathbb{N}^{*+}$ 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, 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 41 involves that each process sends an infinite number of tokens with an increasingly large value of period of life. Thus, P marks an infinite number of successes in i.
Thus: $\forall i = kn, \lim_{r \to +\infty} R_P[i](r) = 1$
□

Ratio for not-multiples of the size

Lemma 42 *$\forall i \in \mathbb{N}^{*+}$ such as i is not a multiple of n, we have: when P receives a token T such as* $\mathrm{cpt}_T = i$*: Proba(P marks a success in i) $\leq$ Proba(P marks a failure in i)*

Proof.

Let $i \in \mathbb{N}^{*+}$ be not multiple of n.
When P receives a false-token T such as $cpt_T = i$:
Proba($color_P = color_T$) = Proba($color_P \neq color_T$) = $\frac{1}{2}$, and
Proba($cpt_P = cpt_T$) $\leq$ Proba($cpt_P \neq cpt_T$)
Thus, Proba(P marks a success in i) $\leq$ Proba(P marks a failure in i)
If P receives a real-token T such as $cpt_T = i$, then T has been created for i rounds by the process $Q \neq P$. Let r be the round where Q created T.
Case (i): If P has created a token T' at round r, then when P receives T, we have: Proba($cpt_P = cpt_T = i$) ≤ 1, and
Proba($color_P = color_T$) = Proba($color_P \neq color_T$) = $\frac{1}{2}$
Thus, Proba(P marks a success in i) $\leq$ Proba(P marks a failure in i)
Case (ii): If P has not created a token at round r, then when P receives T, we have: $cpt_P \neq cpt_T$. Thus, Proba(P marks a success in i) $\leq$ Proba(P marks a failure in i)
□

Theorem 42 *$\forall i \in \mathbb{N}^{*+}$ such as i is not a multiple of n, we have:*

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

Proof Outline.

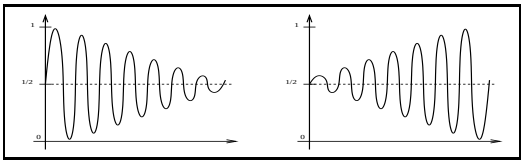

Fig. 4. Variance

According to lemma 42, 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 draw 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($R_P[i](r) \geq 0.9$) $\leq \frac{1}{8.(0,4)^3.N^2}$. The detailed proof, is left for the complete version of the paper.

Theorem 43 *$\forall i \in \mathbb{N}^{*+}$ such as i is not a multiple of n, we have:*

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

Proof Outline. We use a similar argument as for theorem 42. The detailed proof is left for the complete version of the paper.

Conclusion

Theorem 44 (Convergence) *Let E be the set of all possible executions of $\mathcal{RS}$, e be an element of E and r be a round of e.*

$$Proba(\forall\ process\ P : \text{size}_P(r) = n) \xrightarrow{r\ \to\ +\infty} 1$$

Proof Outline. This theorem results from theorems 41 and 42. We compute a finite multiplication of probabilities to obtain this result. The detailed proof is left for the complete version of the paper.

Theorem 45 (Correctness) *Let E be the set of all possible executions of $\mathcal{RS}$, e be an element of E and r be a round of e.*

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

Proof Outline. This theorem results from theorems 41 and 43. We also compute a finite multiplication of probabilities to obtain this result.

The proof is based on the fact that $\left(\sum^{\infty} \frac{1}{N^2}\right) \sim \frac{1}{N}$. The detailed proof is left for the complete version of the paper.

4.3 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 straightforward.

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 considers neither $s_P[i]$, nor $f_P[i]$.

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 mark a failure in n when it receives T, for instance if $color_T \neq color_P$. But P can receive at most n false-tokens like that, because a ring of size n can contain at most n false-tokens at the initialization. On the other hand, if P receives a real-token T such that $cpt_T = n$, then P is the creator of T, then P 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 arrive during the first n rounds. Then, for the value i, the observer does not count the first i tokens that arrived at P. For that the observer has 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.

We have: $F_{\mathcal{O}bs}[n] = 0$ 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.

Proof of the observer

Theorem 46 $\forall \varepsilon \in [0, 1[, \exists \alpha : Proba(\textit{false announcement by } \mathcal{O}bs_\alpha) \leq \varepsilon$

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) $= 0$
$\Rightarrow Prob \leq \sum_{i=1}^{n-1}$ 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)
$\leq \left(\frac{1}{2}\right)^{i+\alpha}$ (according to the lemma 42)
So, we have:

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

By choosing $\alpha \geq -\frac{log\varepsilon}{log2}$, we obtain: Proba(false annoucement) $\leq \varepsilon$
□

Theorem 47 *The observer eventually announces the stabilization with probability 1.*

Proof.
According to theorem 45, we have: Proba $(\forall r' \geq r, \forall P : size_P(r') = n) \xrightarrow{r \to +\infty} 1$
Let us prove that: Proba(observer announces on n) $\xrightarrow{r \to +\infty} 1$
First condition: $F_{\mathcal{O}bs}[n] = 0$.
(*) $F_{\mathcal{O}bs}[n] = 0$ is always true.

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

According to corollary 41, $\forall i \in \mathbb{N}^{*+} : i < n$, P receives an infinite number of tokens T such that $cpt_T = i$. Moreover, according to theorem 42, if P receives a 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:

(**) 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 41, P creates at least $2n+\alpha$ tokens with a period of life $\geq n$, thus: P marks at least $2n+\alpha$ successes in n and the observer marks at least $n+\alpha$ successes in n. Therefore:

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

According to (*), (**) and (***), we have:

Proba(observer announces on n) $\xrightarrow{r \to +\infty} 1$

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

5 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.

References

[BGJ99] J. Beauquier, M. Gradinariu, and C. Johnen. Randomized self-stabilizing and space optimal leader election under arbitrary scheduler on rings. Technical Report 99-1225, Universite Paris Sud, 1999.

[BPR04] Joffroy Beauquier, Laurence Pilard, and Brigitte Rozoy. Observing locally self-stabilization. *Journal of High Speed networks*, 2004.

[Dij74] Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, nov 1974.

[Dol00] Shlomi Dolev. *Self-Stabilization.* MIT Press, Cambridge, MA, 2000.

[LS92] Chengdian Lin and Janos Simon. Observing self-stabilization. In Maurice Herlihy, editor, *Proceedings of the 11th Annual Symposium on Principles of Distributed Computing*, pages 113–124, Vancouver, BC, Canada, August 1992. ACM Press.

[Tel94] Gerard Tel. *Introduction to Distributed Algorithms.* Cambridge Uni Press, Cambridge, 1994.