Service Time Optimal Self-Stabilizing Token Circulation Protocol on Anonymous Unidirectional Rings

Colette Johnen

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

Abstract

We present a self-stabilizing token circulation protocol on unidirectional anonymous rings. This protocol does not required processor identifiers, no distinguished processor (i.e. all processors perform the same algorithm).

The protocol is a randomized self-stabilizing, meaning that starting from an arbitrary configuration (in response to an arbitrary perturbation modifying the memory state), it reaches (with probability 1) a legitimate configuration (i.e. a configuration with only one token in the network).

All previous randomized self-stabilizing token circulation protocols design to work under unfair distributed schedulers have the same drawback: once stabilized, the service time is slow (in the best case, it is bounded by $2N$ where N is the ring size).

Once stabilized, our protocol provides an optimal service: after N computation steps, each processor has obtained one time the token.

The protocol can be used to implement a fair distributed mutual exclusion in any ring topology network.

Keywords: *distributed protocol, fault-tolerant, mutual exclusion, self-stabilization, anonymous ring, token circulation, unfair scheduler, service time.*

1. Introduction

Robustness is one of the most important requirements of modern distributed systems. Various types of faults are likely to occur at various parts of the system. These systems go through the transient faults because they are exposed to constant change of their environment.

The concept of self-stabilization [10] is the most general technique to design a system to tolerate arbitrary transient faults. A self-stabilizing system, regardless of the initial states of the processors and initial messages in the links, is guaranteed to converge to the intended behavior in finite time.

Mutual exclusion is a fundamental task for the management of distributed system. A solution to the problem of mutual exclusion is to implement a token circulation, the processor having the token is granted access to the critical resource. Therefore, lot of research works had been done in order to design efficient self-stabilizing token circulation protocols. Dijkstra gave a number of self-stabilizing protocols for token circulation on a bidirectional ring, assuming the existence of a leader [10, 11]. Several deterministic self-stabilizing token passing protocols for different topologies have been proposed in the literature: [2, 10, 15] for a ring; [5, 16, 17] for a linear array of processors, [26, 12, 29] for tree networks, and [19, 23, 30, 22, 9] for general networks. All these protocols assume the existence of a distinguished processor that performs some specific tasks. Deterministic protocols for leader election and token circulation are possible in a ring of N anonymous processors only when N is prime under a centralized scheduler [10, 6] (i.e. during a computation step a single processor, changes its state). Such protocols are presented in [6, 27, 21, 14].

In this paper, we address the following task: "token circulation on anonymous and unidirectional rings of any size". We have in mind to obtain solutions both self-stabilizing and service optimal. Because, on anonymous networks, without the ability to break symmetry, deterministic self-stabilizing token circulation are impossible, our protocol is randomized.

Related works. Based on the random walks techniques, self-stabilizing randomized token circulation protocols on

bidirectional anonymous networks have been designed [20, 13]. Mayer and al [28] have proposed a self-stabilizing randomized round-robin token management protocol on bidirectional rings that reduces an arbitrary plurality of tokens (i.e. one or more) to a single token. Once stabilized, a round-robin requires $O(N)$ computation steps.

In [18, 1], it is presented randomized token circulation protocols on unidirectional rings that stabilize with some type of schedulers (resp. synchronous scheduler and k-bounded scheduler). In [24], the first token circulation protocol on unidirectional rings that self-stabilizes under unfair distributed schedulers is designed. Beauquier and al in [4] presents a space optimal token circulation protocol on unidirectional rings that is self-stabilizes under unfair distributed schedulers. An adaptation of this protocol that has a better stabilization time is given in [31]. In [3], the protocol of [4] is extended in order to manage any anonymous unidirectional networks.

The previous protocols on unidirectional anonymous networks under distributed schedulers [18, 1, 24, 4, 31, 8] are all based on the same technique: "to randomly retard the token circulation". A processor having a token randomly decides to pass or not the token. Under any scheduler, there is a probability one that one token T moves faster that the other tokens, thus T will eventually catch up the others tokens and will eliminate them.
The drawback of this technique is the service time. Once the ring is stabilized (i.e. there is only one token in the ring), the only token also delays its moves. If the delay is unbounded [1, 24, 4, 31] the upper bounded of the service time is infinite: a processor may never get the token because the token stay forever on the same processor. More precisely these protocols are only weakly self-stabilizing for the specification *"one token fairly circulates in the ring"*.
First, Datta and al. [8] have adapted this technique to guarantee an upper bounded and an average bounded of the service time (the both are $O(N^3)$): the protocol ensures a boundary to the slowness of a token move. Their protocol requires $O(lg(N))$ memory space on each processor.
By delaying the token circulation, Kakugawa and al. in [25] have adapted their protocol presented in [24] to run under unfair distributed schedulers. The service time is $2N$. The required memory space on each processor is $O(lg(N))$.

Our contribution. We present a self-stabilizing token circulation protocol for anonymous unidirectional rings of any size under unfair distributed schedulers. Our protocol is not based on the technique "to randomly delay the token circulation". Our idea is to lock forever a token T when the ring has several tokens. The other tokens keeping on circulate, one of the other tokens will eventually catch up T: the tokens number decreases. A protocol based on the same idea is presented in [24]. This protocol requires a centralized scheduler. Under a centralized scheduler, at each computation step, only one processor performs an action. We have adapted the protocol of [24]. to get a protocol that deals with distributed schedulers. At each computation step, a subset of enabled processors perform an action.
Our protocol does not require fairness property from the scheduler. On the contrary, under any scheduler, the obtained computation is N-fair even during the stabilization period (i.e. between two actions of a processor, another processor performs at most N actions).
The protocol [25] requires that each processor knows exactly the ring size. Our protocol only requires that each processor knows an upper bound on the ring size (called B). Thus, our protocol works on dynamic rings (the ring size may increase or decrease assuming the ring size does not go beyond the upper bound).
As the protocols of [8] and [25], our protocol is self-stabilizing for the specification *"one token fairly circulates in the ring"*. Our protocol have the same space complexity that the protocols of [8] and [25].
The protocols of [8] and [25] have an upper bounded of the service time $O(N^3)$ and $2N$ respectively. Once our protocol stabilized, the only token is not delayed or locked; therefore our protocol provides an optimal service: after N computation steps, each processor has obtained one time the token. Our protocol is the first one that never delays the token circulation.
We give a complete formal proof of correctness and convergence of our protocol.

Outline. The model for randomized self-stabilizing protocols is presented in section 2. The protocol is presented in section 3. The proof of the protocol correctness and self-stabilization is given in section 4.

2. Model

Abstract model. A non deterministic distributed system is represented in the abstract model of *transition systems*. A *distributed system* is a tuple $DS = (\mathcal{C}, T, \Sigma, \mathcal{I})$ where $\mathcal{C}$ is the set of all system configurations; Σ is the finite alphabet. For any letter α of Σ, T_α is a transition function of $\mathcal{C}$ to $\mathcal{C}$ subsets. $\mathcal{I}$ is a $\mathcal{C}$ subset called the initial configurations. We said that there is a *transition* from c of label α if $T_\alpha(c) \neq \phi$. The outputs of the transition $T_\alpha(c)$ are the configurations of the set $T_\alpha(c)$. In a randomized distributed system, there is a probabilistic law on the outputs of a transition. Let $TC2$ be the distributed system defined as $(\{A, B1, B2\}, T, \{a, b1, b2\}, \{A, B1, B2\})$ where $T_a(A) = \{A, B1, B2\}$; $T_{b1}(A) = \{A, B2\}$; $T_{b2}(A) = \{A, B1\}$; $T_{b1}(B1) = T_{b2}(B2) = \{B1, B2\}$ and $T_{b2}(B1) = T_{b1}(B2) = \phi$. The probabilistic law associates to the transition $T_a(A)$ is $1/2$ for the configuration A

and $1/4$ for $B1$ and $B2$. The probability laws associated to other transitions are $1/2$ for each transition output.

A *computation step* is a pair of configurations (c_i, c_j) where c_j is an output of a transition starting from c_i. A *computation* e of DS is a sequence of consecutive computation steps $e = (c_0, c_1), (c_1, c_2) \ldots$ where $c_0 \in \mathcal{I}$. A computation is *maximal*, if the computation is either infinite, or finite and the final configuration is a deadlock. $e_x = (A, A)^x (A, B1)((B1, B2)(B2, B1))^*$ is a maximal computation of $TC2$ for any value of x.

Let c be an initial configuration of a distributed system. The *c-tree* is the tree composed of all maximal computations whose initial configuration is c. The computation *forest* of a distributed system $(\mathcal{C}, T, \Sigma, \mathcal{I})$ is the set of all c-trees where $c \in \mathcal{I}$. The figure 1 illustrates the notion of tree.

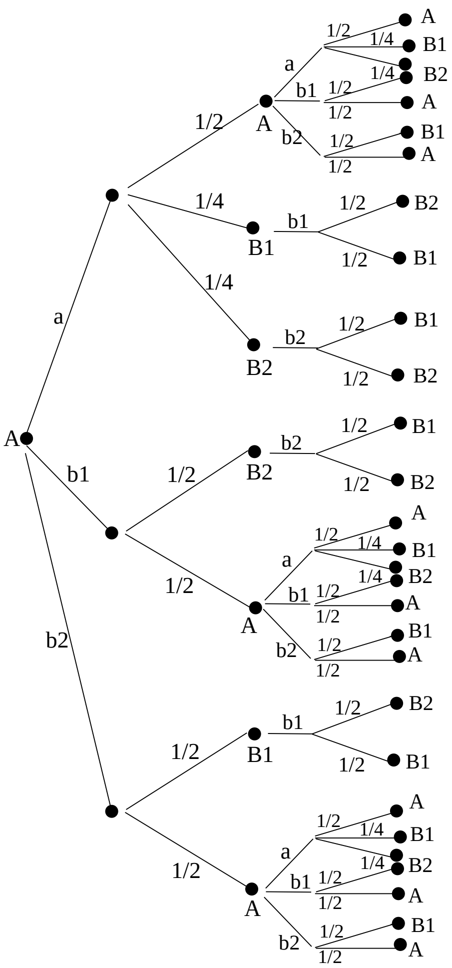

Figure 1. The beginning of A-tree of the distributed system $TC2$

Interpretation. In fact, the distributed system is a networks of processors ($Proc$) computing protocol P. A protocol has a collection of variables (internal and/or field) and has a code part. A processor communicates only with its neighbors (a subset of $Proc$). Communication among neighbors is carried out by field variables.

The state of a processor is the collection of values of the process's variables (internal or field). A configuration of a distributed system is a vector of processor states. A *local configuration* is the part of a configuration that can be "seen" by a processor (i.e. its state and the field variables of its neighbors).

The code is a finite set of guarded rules:

(i.e. label:: guard $\to$ action).

The guard of a rule on p is a boolean expression involving p local configuration. The action of a p rule updates the p state. If the action is randomized, several statements are possible, and each of them has some probability. A processor p is *enabled* at a configuration c, if the rule guard of p is satisfied in c.

Computation step versus transition. Let c be a configuration, and CH be a subset of enabled processors at c. We denote by $< c : CH >$ the set of configurations that are reachable from c after that the processors of CH have performed an action. A computation step has three elements: (1) an initial configuration: c, (2) a set of enabled processors: CH, and (3) a configuration of $< c : CH >$. The computation steps can be interpreted in terms of transitions in the abstract model: $< c : CH >$ is the output configurations of the abstract transition $T_{CH}(c)$ (in this abstract model, the alphabet letters represents the subsets of $Proc$).

For instance, $TC2$ is the system transition representing the weak self-stabilizing token circulation [1] on the unidirectional ring of size 2. Each processor has the same rule:

$Token_p \to$ if $(random(0, 1) = 0)$ then $Pass\_Token_p$.

A is the configuration where all processors have a token. $B1$ (resp. $B2$) is the configuration where only the processor $p1$ (resp. $p2$) has a token. The letter a represents the processor subset $\{p1, p2\}$, the letter $b1$ (resp. $b2$) represents the processor subset $\{p1\}$ (resp. $\{p2\}$).

In the case of a deterministic protocol, a computation step is totally defined by the initial configuration and the set of enabled processors. But in the case of randomized protocol, the final configuration depends on the output of each processor action. Therefore, in the case of randomized protocols, the computation step has a fourth characteristic element: the probabilistic value associated to the computation step. This value depends on the probabilistic law of the random variable of each processor involved in the computation step.

Strategy. Clearly, no probabilistic space can be directly based on computation tree structure. Specific subtrees can equipped with a probabilistic space: the strategies [4]. The formal strategy definition is given below.

Definition 2.1 *Let DS be a distributed system. Let Tr be a tree of DS. A DS strategy is a subtree of Tr where at a node, there is only one outgoing transition.*

The basic notion used to define a probabilistic space on the computations of a given strategy st is the cone. Cones have been introduced in [32]. A *cone* C_h of st is the set of all st's computations with the common prefix h. $length(h)$ is the number of computation steps in h. The measure of a st-cone $\mathcal{C}_h$ is the measure of the prefix h (i.e., the product of the probability of every computation step occurring in h). For instance, the measure of $st1$-cone $\mathcal{C}_{h1}$ (figure 2) where $h1 = (A, a, B2)$ is $1/4$; the measure of $st2$-cone $\mathcal{C}_{h2}$ (figure 3) where $h2 = (A, b1, B2)$ is $1/2$.

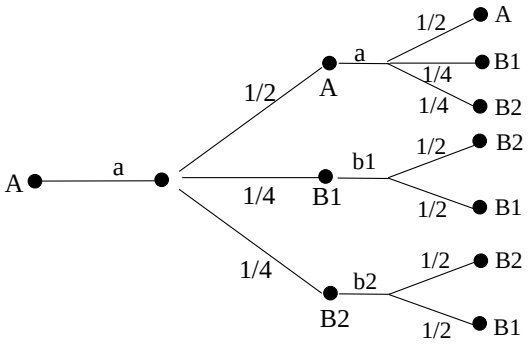

Figure 2. The beginning of the $TC2$ strategy $st1$

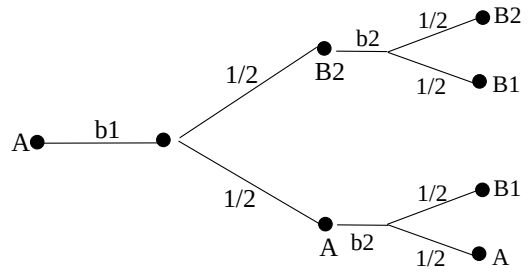

Figure 3. The beginning of the $TC2$ strategy $st2$

Scheduler. Basically, a scheduler is intended to be an abstraction of the external non-determinism. Because the effect of the environment is unknown in advance, the scheduler notion must be able to formalize any external behavior. Defining a scheduler in some operational way - at that point of the computation the scheduler has such or such choice - raises the problem to define exactly in function of what the choice is made. If the choice depends of the actual configuration and of the history, the generality of the scheduler is restricted. How, for instance, express that the scheduler must be fair. That is the reason why in a deterministic system, we define a scheduler as a predicate (subset) on infinite computations. In our framework, the key notion is the strategy. Formally, a *scheduler* is completely defined by the set of strategies which it may "produce".

Definition 2.2 *Let DS be a distributed system. A scheduler D is a set of DS strategies.*

Let DS be a distributed system. The *unfair* scheduler is the set of all DS strategies. The *fair* scheduler is the set of DS strategies that contain only fair computations. The *ϵ-fair* scheduler is the set of DS strategies st such that $P_{st}(unfair\ computations) = \epsilon$.

2.1. Self-stabilization of a randomized protocol

In this section, we define the self-stabilization for randomized protocols with respect to the probabilistic model.

Notation 2.1 *Let st be a strategy of a distributed system DS performing a protocol P under a scheduler D. Let PR be a predicate over configurations. The notation $c \vdash PR$ means that the configuration c verifies the predicate PR. We note by $\mathcal{EPR}_{st}$ the set of st computations reaching a configuration that satisfies the predicate PR.*

Definition 2.3 (Predicate closure) *Let L be a predicate defined on configurations of a distributed system DS. L is closed if any computation step cs from any configuration $c1$ that verifies L reaches a configuration $c2$ that verifies L.*

A problem specification SP is a predicate on computations; for instance the specification of the *leader problem* is "the system has and will always have one and only one leader; the leadership does not move". The definition of self-stabilization of DS under the scheduler D for a specification SP required the definition of a predicate on configurations (legitimate predicate) L. If the DS converge to L and verifies the L-correctness property then DS is a self-stabilizing system for SP under D. In a deterministic system, the convergence property is "all computations under a scheduler D reach a legitimate configuration". In a randomized system the definition of convergence property is probabilistic: "the probability to reach a legitimate configuration is 1 in any strategy of the scheduler D". The L-correctness property is deterministic in any distributed system (deterministic or not): "any computation from a legitimate configuration c (i.e. $c \vdash L$) satisfies the specification SP".

Definition 2.4 (Probabilistic convergence) *Let L be a predicate defined on configurations. A randomized distributed system DS executing the protocol P under a scheduler D converges to L iff: In any strategy st of DS under D the probability of the set of computations reaching L is equal to 1. Formally, $\forall st$ of DS under D, $P_{st}(\mathcal{EL}_{st}) = 1$.*

Definition 2.5 (Probabilistic Self-stabilization) *A randomized distributed system DS executing the protocol P under a scheduler D is self-stabilizing for a specification SP (predicate on the computations), if there exists a predicate on configuration L such that DS converges to L and verifies the following property:*

- **correctness** *$\forall st$ of DS under D, $\forall e \in st$, if $e \in \mathcal{EL}$ then e has a suffix that verifies SP.*

The protocols [18, 1, 24, 4, 31] are only weakly self-stabilizing for the specification *"one and only one token fairly circulates in the ring"*. Because, some computations from any "legitimate" configuration are not correct. It its always possible that "a token stays forever on the same processor". Fortunately, the probability of a such event is 0 in any strategy.

Definition 2.6 (Weak Probabilistic Self-stabilization) *A randomized distributed system DS executing the protocol P under a scheduler D is weakly self-stabilizing for a specification SP, if DS verifies the following property:*

- **probabilistic correctness** *$\forall st$ of DS under D, $P_{st}(\{e$ has a suffix that verifies $SP\}) = 1$.*

Based on previous works on the probabilistic automata (see [32], [33], [34]) [4] presents a detailed framework for proving self-stabilization of probabilistic distributed systems. A key notion is *local convergence* denoted LC. The LC property is a progress statement as those presented in [7] (for the deterministic systems) and [32] (for the probabilistic systems). Informally, the $LC(PR1, PR2, D, \epsilon)$ property for a randomized self-stabilizing system means that starting in a configuration satisfying PR_1, the system will reach a configuration which satisfies PR_2, in less that D computation steps with a probability greater than ϵ. Formally the *local convergence* property is defined as follows:

Definition 2.7 (Local Convergence) *Let st be a strategy, PR_1 and PR_2 be two predicates on configurations, where PR_1 is a closed predicate. Let δ be a positive probability and D a positive integer. Let $\mathcal{C}_h$ be a st-cone with $last(h) \vdash PR_1$ and let M be the set of sub-cones $\mathcal{C}_h$ of the cone $\mathcal{C}_h$ such that for every sub-cone $\mathcal{C}_{h'}$: $last(h') \vdash PR_2$ and $length(h') - length(h) \leq D$. The cone $\mathcal{C}_h$ satisfies $LC(PR_1, PR_2, \delta, D)$ if and only if $Pr_{st}(\bigcup_{\mathcal{C}_{h'} \in M} \mathcal{C}_{h'}) \geq \delta$.*

Now, if in strategy st, there exist $\delta_{st} > 0$ and $D_{st} \geq 1$ such that any st-cone, $\mathcal{C}_h$ with $last(h) \vdash PR_1$, satisfies $LC(PR_1, PR_2, \delta_{st}, D_{st})$, then the main theorem of the framework presented in [4] states that the probability of the set of computations of st reaching configurations satisfying both PR_1 and PR_2 is 1. Formally:

Theorem 2.1 *[4] Let st be a strategy. Let PR_1 and PR_2 be closed predicates on configurations such that $Pr_{st}(\mathcal{EPR}_1) = 1$. If $\exists \delta_{st} > 0$ and $\exists D_{st} \geq 1$ such that any st-cone $\mathcal{C}_h$ with $last(h) \vdash PR_1$, satisfies the LC $(PR_1, PR_2, \delta_{st}, D_{st})$ property, then $Pr_{st}(\mathcal{EPR}) = 1$, where $PR = PR_1 \wedge PR_2$.*

3. Self-stabilizing token circulation protocol

We present a self-stabilizing token circulation protocol for anonymous unidirectional rings of any size under unfair distributed schedulers (protocol 3.1).

To converge to a configuration where there is only one regular token, our protocol locks forever a regular token T when the ring has several regular tokens. The other regular tokens keeping on circulate, one of the other regular tokens will eventually catch up T: the regular tokens number decreases. [24] have presented a protocol based on the same idea, but their protocol requires a centralized scheduler.

Protocol 3.1 Optimal time service token circulation protocol for processor p

---

Field variables:

v_p (the current value) is an integer bounded by $B > N$ (N being the ring size)

r_p (the random signature) of p is in $\{0, 1\}$

Macros (l_p is p's left neighbor):

$Pass\_Token_p = v_p := (v_{l_p} + 1) \bmod B$

Predicates:

$Token_p \equiv v_p \neq (v_{l_p} + 1) \bmod B$

$Blocked_p \equiv (v_p = 0) \wedge (r_{l_p} > r_p)$

$Wrong\_Token_i \equiv \neg Token_p \wedge (r_p \neq r_{l_p}) \wedge (v_p \neq 0)$

$New\_seg_p \equiv (v_{l_p} = B - 1)$

The Moving rules:

$M_1 :: Token_p \wedge \neg Blocked_p \wedge \neg New\_seg_p \rightarrow$
$\quad Pass\_Token_p; r_p := r_{l_p}$

$M_2 :: Token_p \wedge \neg Blocked_p \wedge New\_seg_p \rightarrow$
$\quad Pass\_Token_p; r_p := random(0, 1)$

The Correcting rule:

$C_3 :: Wrong\_Token_p \rightarrow r_p := r_{l_p}$

The difficulty of this technique is to find a local configuration that indicates that several regular tokens are in the ring. Let us analyze the behavior of our protocol when there

is only one regular token and no wrong token. Every $B-1$ moves, the regular token T randomly changes its signature values (M_2 rule). We name p the processor that has performed the M_2 action. After this action $v_p = 0$ and r_p is equal to the signature value of T. The signature value of T does not change during the next $B-1$ moves of T (M_1 rule). Assuming that T is the only token in the ring, the next time that p gets a regular token: (1) this token is T; (2) the signature of T has not changed in the mi-time (T has performed $N-1 < B-1$ moves). Therefore we have, $r_l = r_p$. We conclude that if $r_l \neq r_p$, then T is not the only token in the ring. The local configuration of p ($(v_p = 0) \wedge (r_l \neq r_p)$) indicates that several processors have a token - regular or not -.

To avoid deadlock configuration, the regular token on p is locked (i.e. p is not enabled) only if $((v_p = 0) \wedge (r_l > r_p))$.

Once, there is only one regular token in the ring; this token is never locked it will freely circulate without delay. The service time is optimal (N computation steps) - formally proven in section 4.1.

Observation 3.1 *Assume that during a computation step cs of protocol 3.1, a processor p performs an action. The processor p is not more enabled at the end of cs, unless it left neighbor has also performed an action during cs. Therefore, along any computation of protocol 3.1, the number of enabled processors cannot increase whatever may happen.*

Definition 3.1 *A processor holds a regular token iff it verifies the predicate $Token$. A processor holds a wrong token iff it verifies the predicate $Wrong\_Token$.*

Let T be a regular token holds by p. The value of T is the value of the p's left neighbor. The signature of T is the value of the p's left neighbor.

Observation 3.2 *As $B > N$, there is always a regular token in the ring.*

A processor having a token (regular or wrong) passes its token to its right neighbor when it performs an action. When a processor holding a token receives another token, the two tokens merge into a single token or the tokens annihilate each other.

4. Self-stabilizing proof

We prove that the protocol 3.1 is a self-stabilizing protocol for the specification SP: The token circulations are service optimal. A token circulation is service optimal iff (i) the circulation is 1-fair (i.e. in every round, every processor obtains the token once) and (ii) the round duration is exactly N computation steps.

4.1. Correctness proof

Definition 4.1 *We name W the weight function $\mathcal{C} \longrightarrow \mathcal{N}^2$ defined as: $W(c) =$ (number of regular tokens in c, number of wrong tokens in c).*

L is the following predicate on configurations:
$L_c \equiv (W(c) = (1,0))$.

L is the legitimate predicate. We will prove that any computation whose the initial configuration verifies L, verifies the specification SP. First we will prove that L is closed and that all computations are infinite.

Observation 4.1 *Let c be a configuration of protocol 3.1 such that $W(c) = (k,l)$. We have $l \leq N-1$ and $k \leq N$. In the configuration c, the number of enabled processors is inferior or equal to $k+l$.*

Let c' be the configuration reached from c after a computation step. If $W(c') = (k',l')$ then $k' \leq k$ (the number of regular tokens cannot increase).

Lemma 4.1 *Let c be a configuration of protocol 3.1 on an anonymous and unidirectional ring such that $W(c) = (k,l)$. Let c' be a configuration reached from c after a computation step cs. Assume that $W(c') = (k',l')$. If $l' > l$ then $k' < k$ otherwise $k' \leq k$.*

Proof: According to observation 4.1, We have $k' \leq k$.

Assume that during cs there is creation of a wrong token. We name p the processor having this wrong token. There is creation iff p has a wrong token in c' and not in c, and p's left neighbor (q) has not performed the Correcting rule during cs. q has performed a Moving rule during cs; otherwise p would not verify the $Wrong\_Token$ predicate in c'. As p has not a regular token in c', then p was holding a regular token in c and p has not performed an action during cs. Thus, a regular token has cautgh up another one during cs. We have $k' < k$. The figure 4 illustrates the situaton where a wrong token is created. □

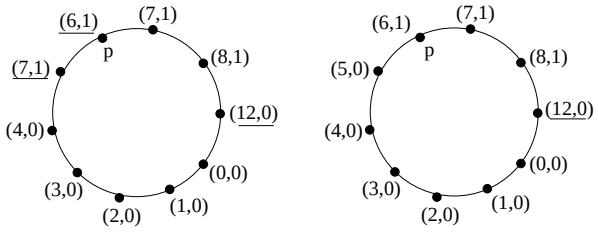

Figure 4. Creation of a wrong token

Corollary 4.1 *L is closed in the protocol 3.1 on an anonymous and unidirectional ring.*

Proof: Let c be a configuration verifying L. Let c' be a configuration reached after a computation step from c. We have $W(c') = (k', l')$ According to observations 3.2 and 4.1, $k' = 1$. According to lemma 4.1, $l' = 0$. We have $W(c') = (1, 0)$ □

Lemma 4.2 *There is not deadlock configuration in the protocol 3.1 on an anonymous and unidirectional ring.*

Proof: Let c be a configuration of protocol 3.1. Assume that c is a deadlock. In c, a processor p_0 has a regular token (observation 3.2). In c, a processor having a regular token has the value 0. As the distance between two regular tokens is inferior to $N < B$. All processors having the value 0 must have a regular token. In c, no processor has a wrong token; therefore all processors between two regular tokens have the same signature value. Let p_0, p_1, ..., p_n be the finite processors series such that (i) $\forall i \in [0, n-1]$, p_i holds a regular token and no processor between p_i and p_{i+1} has a regular token; and (ii) $p_0 = p_n$. We have $n \geq 1$ because, there is always a regular token in the ring (observation 3.2). $\forall i \in [0, n-1]$, We have $r_{p_i} = r_{l_{p_i}} > r_{p_i}$ where l_{p_i} is the left neighbor of p_{i+1}. We conclude that $r_{p_0} > r_{p_0}$. There is a contradiction because $p_0 = p_n$. □

Theorem 4.1 *Let e be a computation of protocol 3.1 on an anonymous and unidirectional ring whose the initial configuration verifying L. Along e, SP is verified.*

Proof: According to lemma 4.2, e is infinite. According to corollary 4.1, all configurations of e verifying L. In each computation step of e, the processor p having the regular token is the only enabled processor (observation 4.1). p performs a Moving action to pass the token in the ring to its right neighbor. After N computation steps, the regular token has terminated a round. □

We have proven that once L is reached SP is verified. In the following section, we will prove that in any strategy, the probability to reach a configuration verifying L is 1.

4.2. Convergence proof

First, we present three properties verify by any computation of the protocol 3.1. Secondly, we prove that any computation from a configuration c where $W(c) = (k, l+1)$ reaches a configuration c' such that $W(c) > W(c')$, in less than $(B - 1 + 3N)N$ computation steps. Third, in any strategy, in any cone $\mathcal{C}_h$ such that $W(c) = (k+1, 0)$, the probability to reduce the configuration weight in less than $(2B + 4N)N$ steps is greater than $1/2^5$. Finally, we conclude that in any strategy under any scheduler, the probability to reach a configuration verifying L is 1.

Lemma 4.3 *Let e be a computation of protocol 3.1. Along e, a processor p will wait at most N^2 computation steps before performing an action.*

Proof: We study a fragment of e where p does not perform any action, called f. After an action, a processor does not verify a guard rule till its left neighbor does not perform an action. Therefore, between two actions of p its right neighbor can perform at most one action. By induction on the distance from p to q_d, we prove that q_d can perform at most d actions before a p action (d being the distance from p to q_d). Therefore, f contains at most d computation steps where q_d performs an action. Thus f contains at most $N(N-1)/2$ computation steps. □

Lemma 4.4 *Let e be a computation of protocol 3.1. A given regular token T will circulate forever in the ring and it will have infinitely often the value $B-1$ or it will merge with its preceding regular token.*

Proof: In a configuration of c, assume that the regular token T is on the processor p and T has the value x. Along e, p will perform a Moving action before that its left neighbor performs a Moving action (otherwise the T token would merge with its preceding regular token T'). After the p's action, the T token has the value $x+1$ and it is held by the p's right neighbor. By induction, we prove that the T token will get the value $B-1$ or will merge with another token. If the token T never merges with its preceding regular token: T will get the value 0; then, T will eventually get the value $B-1$; and so one. □

Lemma 4.5 *Let e be a computation of protocol 3.1. Along e, if the number of regular tokens does not decrease during $(X+N)N$ computation steps then each token (regular or wrong) has performed at least X moves during these computation steps.*

Proof: Let e' be the $(X+N)N$ first computation steps of e. Assume that along e', the number of regular tokens does not decrease. In e' the number of wrong token cannot increase (lemma 4.1). Let T be a regular token. Let T' be a token (regular or not). We name $m'_e(T')$ the number of moves of T' during e'.

Along e', a regular token $T1$ always stays behind T, otherwise T and $T1$ would merge (the number of regular token would decrease). We have $m'_e(T1) < m'_e(T) + d$ where d is the initial distance between $T1$ and T. If a wrong token $T2$ catchs up a regular token T then $T2$ vanishes (a processor cannot have a wrong and a regular token at the same time). Thus, $m'_e(T2) < m'_e(T) + d$ where d is the initial distance between $T2$ and T.

We conclude that, if $m_e(T) < X$ then $m_e(T') < X + N$. By hypothesis, $length(e') = (X+N)N$ and we have proven that $length(e') = sum_{T:token} m_e(T') < (X+N)N$. There is a contradiction. □

Lemma 4.6 *Let e be a computation of protocol 3.1. Along e, if the number of regular tokens does not decrease during*

$(B-1+3N)N$ *computation steps then there is no wrong token in the ring after these computation steps.*

Proof: Assume that along e, the number of regular tokens does not decrease during $(B-1+3N)N$ computation steps. A regular token T will be preceded by the same regular token T' during these computation steps.

Along this sequence, T will get the value $B-1$ (lemma 4.4) in less than $(B-1+N)N$ computation steps (lemma 4.5). Then, the processor having the T token (called p) will eventually perform the M_2 action to get the value 0 and to pass the T token to its right neighbor (after at most N^2 computation steps - lemma 4.3 -). Until the next action of p, no processor between p and the token T has a wrong token. After at most N^2 computation steps, p will perform an action. We name c' the configuration of the system just before the p action. In c', p has the T' token; thus no processor on the path between T' and T has a wrong token.

No processor inside this path may perform an action. Only processors holding T or T' may perform an action. After their actions, no processor on the path between T' and T has a wrong token. Therefore, at the end of this sequence, no processor has a wrong token. □

Lemma 4.7 *Let st be strategy under a distributed unfair scheduler of protocol 3.1 on an anonymous and unidirectional ring. Let $\mathcal{C}_h$ be a st-cone where $W(last(h)) = (k, l+1)$ Let M the set of sub-cones $\mathcal{C}_{hh'}$ of $\mathcal{C}_h$ such that (i) $lenght(h') \leq (B-1+3N)N$ and (ii) $W(last(hh')) = (k', l')$ such that if $k' = k$ then $l' = 0$. We have $Pr_{st}(M) = Pr_{st}(\mathcal{C}_h)$*

Proof: We study M' the set of sub-cones $\mathcal{C}_{hh'}$ of $\mathcal{C}_h$ such that: (i) in $last(hh')$, no processor has a wrong token; or (ii) two regular tokens have merged during the last computation step of h'. According to lemma 4.6, after at most $(B-1+3N)N$ computation steps, no processor has a wrong token or two regular tokens have merged. Thus any computation of $\mathcal{C}_h$ belongs M. We have $M' = \mathcal{C}_h$. The length of h' is inferior to $(B-1+3N)N$ computation steps. In the first case, $W(last(hh')) = (k, 0)$. In the second case, $W(last(hh')) = (k', l')$ where $k' < k$. $M' = M$. □

Lemma 4.8 *Let st be strategy under a distributed unfair scheduler of protocol 3.1 on an anonymous and unidirectional ring. Let $\mathcal{C}_h$ be a st-cone where $W(last(h)) = (k+1, 0)$. Let M the set of sub-cones $\mathcal{C}_{hh'}$ of $\mathcal{C}_h$ such that (i) $length(h') \leq (2B+4N)N$ and (ii) $W(last(hh')) < (k, N-1)$. We have $Pr_{st}(M) \geq Pr_{st}(\mathcal{C}_h) \times 1/2^5$.*

Proof: We name T a token. We name T' the token that follows T in the ring. $T' \neq T$, because the ring has several tokens. Let cs be the following scenario: (1) a processor holding T' performs the M_2 actions; (2) a processor having the T token (called p) performs the M_2 action; (3) p gets the T' token; (4) finally the left neighbor of p performs an action during the last computations of h'.

We study the sub-cones $\mathcal{C}_{hh'}$ of $\mathcal{C}_h$ where: (i) two tokens merge during the last computation step of h' or (ii) the scenario sc happens. We call M the set of these sub-cones.

According to lemma 4.4, along any computation of $\mathcal{C}_h$, (a) the token T and T' will perform infinitely often the M_2 action and any tokens will circulate forever in the ring or (b) two tokens will merge. Thus any computation of $\mathcal{C}_h$ belongs M. We have $M = \mathcal{C}_h$.

The first stage of scenario sc required at most $(B+N)N$ computation steps (times that the T' token gets the value 0 - lemma 4.5 -). The second stage (times that the T token gets the value 0), required at most $(B+N)N$. The third stage requires at most N^2 steps (time that p performs an action). The last stage requires N^2 steps (time that p's left neighbor performs an action). The length of h' is inferior to $(2B+4N)N$ computation steps.

Assume that along h' the scenario sc does not happen. By definition of h', we have $W(last(hh')) \leq (k, N-1)$.

If along h' the scenario sc happens. No processor has a wrong token along h' (lemma 4.1): all computation steps of h' contains only Moving actions. Assume that along h', (i) T' always gets the 1 signature when it performs the M_2; (ii) p gets the 0 signature when it performs the M_2 action (when it has the T token); (iii) After (1), the signature of T' will be always 1. After (2), we have: $((r_p = 0) \wedge (v_p = 0))$. After (3), p verifies the predicate $Token \wedge Blocked$. p cannot perform any action until its left neighbor performs an action. At the end of h', two tokens has merged: T' has disappeared. The h' stages are illustrated in the figure 5.

During the first stage, T' will perform one time the M_2 action, during the second stage T' will perform at most 2 times the action M_2 (T' performs at most $B+N$ moves - at most two times the action M_2 -). During the two last stages, p performs no move; thus, T' can perform at most one time the action M_2. The probability of (i), (ii) and (iii) is greater than $1/2^5$. Because along h', T' performs at most 4 times the action M_2. We conclude that the probability that in $last(hh')$, there are less than $k+1$ tokens is greater than $1/2^5$. □

The following corollaries are direct consequences of two preceding lemmas.

Corollary 4.2 *Let st be strategy under a distributed unfair scheduler of protocol 3.1 on an anonymous and unidirectional ring. Let $\mathcal{C}_h$ be a st-cone where $W(last(h)) = (1+k, l)$. Let M the set of sub-cones $\mathcal{C}_{hh'}$ of $\mathcal{C}_h$ such that (i) $length(h') \leq (3B+7N)N$ and (ii) $W(last(hh')) \leq (k, N-1)$. We have $Pr_{st}(M) \geq Pr_{st}(\mathcal{C}_h) \times 1/2^5$.*

Therefore, the average expectation number of computation steps to reduce the number of tokens is less than

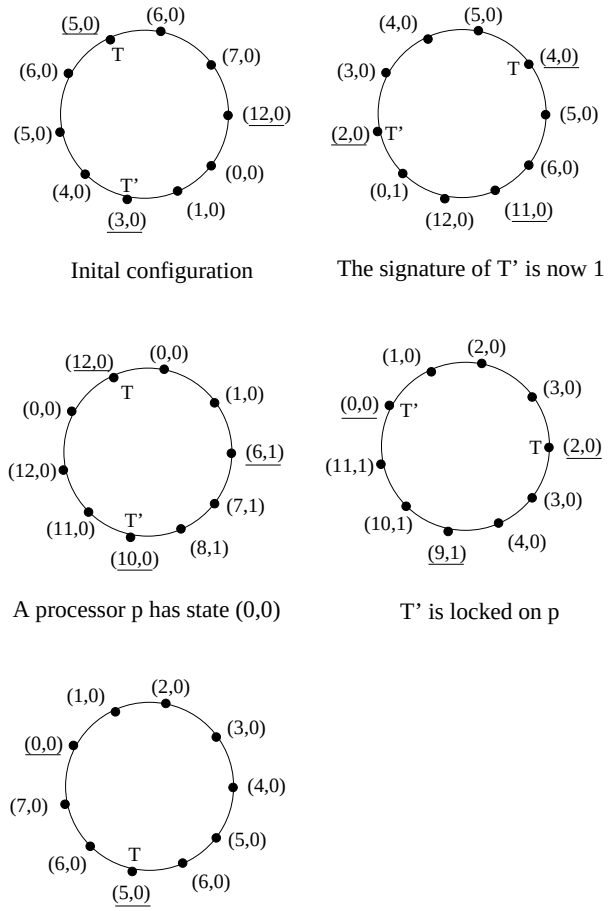

Figure 5. The locking of a regular token under a synchronous scheduler

$160.B.N$.

Corollary 4.3 *Let st be strategy under a distributed unfair scheduler of protocol 3.1 on an anonymous and unidirectional ring. Let $\mathcal{C}_h$ be a st-cone where $W(last(h)) = (1+k, l)$ Let M the set of sub-cones $\mathcal{C}_{hh'}$ of $\mathcal{C}_h$ such that (i) $length(h') \leq (k+1)(3B+7N)N$ and (ii) $W(last(hh')) = (1, 0)$. We have $Pr_{st}(M) \geq Pr_{st}(\mathcal{C}_h) \times 1/2^{5k}$.*

The following theorem is a direct consequence of observation 4.1 and the preceding corollary.

Theorem 4.2 *Let st be strategy under a distributed unfair scheduler of protocol 3.1 on an anonymous and unidirectional ring. Any st-cone $\mathcal{C}_h$ satisfies the LC $(true, L, 1/2^{5N}, (3B + 7N)N^2)$ property.*

From Theorems 2.1 and 4.2, we get:

Corollary 4.4 *In any strategy st of protocol 3.1 on anonymous and unidirectional rings under any unfair distributed scheduler, the probability of the set of computations reaching L is 1.*

5. Conclusion

In this paper, we have presented a randomized self-stabilizing token circulation on anonymous and unidirectional rings of any size. We have given a formal proof of the convergence of the protocol.

Our protocol is the first one that never delays the token circulation. once stabilized, our protocol provides an optimal service time: (i) it implements a *1-fair* token circulation scheme, i.e., in every round, every processor obtains the token once; and (ii) the round duration is exactly N computation steps.

There are only two previous self-stabilizing token circulation on anonymous and unidirectional rings [8, 25] that ensure an upper bounded on the service time (respectively $O(N^3)$ and $2N$).

Similary to the computation of convergence time in [24], we can prove that the expectation time of convergence of our protocol is $0(BN^2)$ computation steps (B is an integer greater than the ring size N). This convergence time is similar to the convergence time of protocols [8, 25]: $O(N^3)$ computation steps.

The memory space required by our protocol and the protocols [8, 25] on each processor is $O(lg(N))$.

References

[1] J. Beauquier, S. Cordier, and S. Delaët. Optimum probabilistic self-stabilization on uniform rings. In *Proceedings of the Second Workshop on Self-Stabilizing Systems*, pages 15.1–15.15, 1995.

[2] J. Beauquier and O. Debas. An optimal self-stabilizing algorithm for mutual exclusion on bidirectional non uniform rings. In *Proceedings of the Second Workshop on Self-Stabilizing Systems*, pages 17.1–17.13, 1995.

[3] J. Beauquier, J. Durand-Lose, M. Gradinariu, and C. Johnen. Token based self-stabilizing uniform algorithms. *Journal of Parallel and Distributed Computing*, 62(5):899–921, May 2002.

[4] J. Beauquier, M. Gradinariu, and C. Johnen. Randomized self-stabilizing and space optimal leader election under arbitrary scheduler on rings. Technical Report 1225, L.R.I, December 1999.

[5] G. Brown, M. Gouda, and C. Wu. Token systems that self-stabilize. *IEEE Transactions on Computers*, 38:845–852, 1989.

[6] J. Burns and J. Pachl. Uniform self-stabilizing rings. *ACM Transactions on Programming Languages and Systems*, 11:330–344, 1989.

[7] K. Chandy and J. Misra. *Parallel Programs Design: A Foundation*. Addison-Wesley, 1988.

[8] A. Datta, M. Gradinariu, and S. Tixeuil. Self-stabilizing mutual exclusion using unfair distributed scheduler. In *IPDPS'2000 Proceedings of the 14th International Parallel and Distributed Processing Symposium*, pages 465–470, 2000.

[9] A. K. Datta, C. Johnen, F. Petit, and V. Villain. Self-stabilizing depth-first token circulation in arbitrary rooted networks. *Distributed Computing*, 13(4):207–218, 2000.

[10] E. Dijkstra. Self stabilizing systems in spite of distributed control. *Communications of the Association of the Computing Machinery*, 17:643–644, 1974.

[11] E. Dijkstra. A belated proof of self-stabilization. *Distributed Computing*, 1:5–6, 1986.

[12] S. Dolev, A. Israeli, and S. Moran. Self-stabilization of dynamic systems assuming only read/write atomicity. *Distributed Computing*, 7:3–16, 1993.

[13] J. Durand-Lose. Randomized uniform self-stabilizing mutual exclusion. *Information Processing Letters*, 74(5-6):203–207, 2000.

[14] F. Fich and C. Johnen. A space optimal, deterministic, self-stabilizing, leader election algorithm for unidirectional rings. In *DISC00 Distributed Computing 15th International Symposium, Springer-Verlag LNCS:2180*, pages 224–239, 2001.

[15] M. Flatebo, A. Datta, and A. Schoone. Self-stabilizing multi-token rings. *Distributed Computing*, 8:133–142, 1994.

[16] S. Ghosh. An alternative solution to a problem on self-stabilization. *ACM Transactions on Programming Languages and Systems*, 15:735–742, 1993.

[17] M. Gouda and F. Haddix. The stabilizing token ring in three bits. *Journal of Parallel and Distributed Computing*, 35:43–48, 1996.

[18] T. Herman. Probabilistic self-stabilization. *Information Processing Letters*, 35:63–67, 1990.

[19] S. Huang and N. Chen. Self-stabilizing depth-first token circulation on networks. *Distributed Computing*, 7:61–66, 1993.

[20] A. Israeli and M. Jalfon. Token management schemes and random walks yield self-stabilizing mutual exclusion. In *PODC90 Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing*, pages 119–131, 1990.

[21] G. Itkis, C. Lin, and J. Simon. Deterministic, constant space, self-stabilizing leader election on uniform rings. In *WDAG95 Distributed Algorithms 9th International Workshop Proceedings, Springer-Verlag LNCS:972*, pages 288–302, 1995.

[22] C. Johnen, G. Alari, J. Beauquier, and A. Datta. Self-stabilizing depth-first token passing on rooted networks. In *WDAG97 Distributed Algorithms 11th International Workshop Proceedings, Springer-Verlag LNCS:1320*, pages 260–274, 1997.

[23] C. Johnen and J. Beauquier. Space-efficient distributed self-stabilizing depth-first token circulation. In *Proceedings of the Second Workshop on Self-Stabilizing Systems*, pages 4.1–4.15, 1995.

[24] H. Kakugawa and M. Yamashita. Uniform and self-stabilizing token rings allowing unfair daemon. *IEEE Transactions on Parallel and Distributed Systems*, 8:154–162, 1997.

[25] H. Kakugawa and M. Yamashita. Uniform and self-stabilizing fair mutual exclusion on unidirectional rings under unfair distributed daemon. *Journal of Parallel and Distributed Computing*, 62(5):885–898, May 2002.

[26] H. Kruijer. Self-stabilization (in spite of distributed control) in tree-structured systems. *Information Processing Letters*, 8:91–95, 1979.

[27] C. Lin and J. Simon. Observing self-stabilization. In *PODC92 Proceedings of the Eleventh Annual ACM Symposium on Principles of Distributed Computing*, pages 113–123, 1992.

[28] A. Mayer, Y. Ofek, R. Ostrovsky, and M. Yung. Self-stabilizing symmetry breaking in constant-space. In *STOC92 Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 667–678, 1992.

[29] F. Petit. Highly space-efficient self-stabilizing depth-first token circulation for trees. In *OPODIS'97, International Conference On Principles Of Distributed Systems Proceedings*, pages 221–235, 1997.

[30] F. Petit and V. Villain. Color optimal self-stabilizing depth-first token circulation. In *I-SPAN'97, Third International Symposium on Parallel Architectures, Algorithms and Networks Proceedings, IEEE Computer Society Press*. IEEE Computer Society Press, 1997. To appear.

[31] L. Rosaz. Self-stabilizing token circulation on asynchronous uniform unidirectional rings. In *PODC00 Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 249–258, 2000.

[32] R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, MIT, Departament of Electrical Engineering and Computer Science, 1995.

[33] R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. In *CONCUR'94 Fifth International Conference Concurrency Theory, Springer-Verlag LNCS:836*, pages 481–496, 1994.

[34] S. Wu, S. A. Smolka, and E. Stark. Composition and behaviors of probabilistic i/o automata. In *CONCUR'94 Fifth International Conference Concurrency Theory, Springer-Verlag LNCS:836*, pages 513–528, 1994.