Optimization of Service Time and Memory Space in a 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 algorithm can deal with any kind of schedulings even unfair ones.

Our 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). Once the system is stabilized the circulation of the sole token is 1-fair (i.e. in every round, every processor obtains the token once). N token circulations are done in at most $0(N^3)$ computation steps where N is the ring size.

The memory space required by our algorithm on each processor $O(M_N)$, M_N being the smallest non divisor of ring size. In [BGJ99a], it was been proven that the minimal memory space required by a self-stabilizing token circulation under any unfair distributed scheduler is $O(M_N)$. Previous randomized self-stabilizing token circulation protocols design to work under unfair distributed schedulers require $O(N)$ memory space or they have not bound on the service time (i.e. duration of a token round). Thus, we present the first protocol having the two major advantages: a bounded service time and optimal in memory space.

Keywords: Distributed algorithm, self-stabilization, mutual exclusion, token circulation, anonymous ring, unfair scheduler, service time.

Résumé Nous présentons un protocole auto-stabilisant de circulation de jeton sur les anneaux anonymes et unidirectionnels. Les processeurs ne peuvent pas être distingués car ils n'ont pas d'identifiant et ils exécutent tous le même algorithme. L'algorithme est auto-stabilisant quelque soit l'ordonnancement des processeurs (même si l'ordonnancement est inéquitable).

L'algorithme que nous présentons est probabiliste. A partir d'une configuration arbitraire (obtenu après une perturbation dans le réseau) il atteint avec une probabilité égale à 1 une configuration légitime (c-a-d une configuration avec un seul jeton).

Une fois que le système est stable, la circulation de l'unique jeton est 1-équitable (durant un tour, chaque processeur obtient une et une seule fois le jeton). N tours du jeton sont réalisés en au plus $0(N^3)$ pas de calcul, N étant la taille de l'anneau.

L'espace mémoire demandé par notre protocole est $O(M_N)$ états par processeur (M_N étant le plus petit entier non diviseur de la taille de l'anneau). Il a été prouvé que l'espace mémoire nécessaire à un protocole auto-stabilisant de circulation de jeton sous n'importe quel ordonnancement est $O(M_N)$ [BGJ99a].

Soit l'espace mémoire demandée par les précédents protocoles est $O(N)$ états par processeur, soit la durée maximale d'une circulation du jeton est non bornée. Nous présentons donc le premier protocole ayant les deux propriétés suivantes: optimal en espace mémoire et ayant un borne supérieure sur la durée d'un tour du jeton.

Mots clés: systèmes réparties, auto-stabilisation, systèmes asynchrones, anonyme processeur, exclusion mutuelle, circulation de jeton, temps de service

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 [Dij74] 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.

In this paper we address the task: token circulation on anonymous rings of any size. We have in mind to obtain solutions both self-stabilizing and providing a good service time. Service time is the maximal time in term of computation steps required by the protocol to perform a token circulation. 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 bidirectionnel anonymous networks have been designed [IJ90, DL00].

[Her90, BCD95] present randomized token circulation protocols on unidirectional rings that stabilize with some type of schedulers (resp. synchronous scheduler and k-bounded scheduler). In [KY97], the first token circulation protocol on unidirectional rings that self-stabilizes under unfair distributed schedulers is presented. [BGJ99b] presents a space optimal token circulation protocol on unidirectional rings that self-stabilizes under unfair distributed schedulers. An adaptation of this protocol that has a better stabilization time is presented in [Ros00]. In [BDLGJ02], the protocol of [BGJ99b] is extended in order to manage any anonymous unidirectional networks.
The protocols of [Her90, BCD95, KY97, BGJ99b, Ros00, DGT00] 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 catches 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 [BCD95, KY97, BGJ99b, Ros00] 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"*.
Datta and al. [DGT00] 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.
Kakugawa and al. in [KY97] have presented a token circulation protocol where the token circulation is not delayed. But this protocol can work only under a weak scheduler (a centralized one). By delaying the token circulation, Kakugawa and al. in [KY02] have adapted their protocol presented in [KY97] to run under any unfair distributed scheduler. The service time is $2N$.
Johnen [Joh02] have presented a protocol where once stabilized, the only token is not delayed or locked; therefore this protocol ensures an optimal service: after N computation steps, each processor has obtained one time the token.

Our contribution. We present a self-stabilizing token circulation protocol for anonymous unidirectional ring of any size under unfair distributed scheduler. Our protocol is not based on the technique "to randomly delay the token circulation".
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. two actions of a processor, another processor performs at most N actions).
Once stabilized, the service time depends only of the scheduler. Under a synchroneous scheduler, a token circulation is done in N computation steps (the optimal time). Under any scheduler, N token circulations require at most $O(N^3)$ computation steps.
As the protocols of [DGT00], [KY02] and [Joh02], our protocol is self-stabilizing for the specification *"one token fairly circulates in the ring"*. The protocol of [DGT00], [KY02] and [Joh02] have an upper bounded on the service time (a token round takes respectively $O(N^3)$, $2N$, or N computation steps). The protocols of [DGT00], [KY02] and [Joh02] require $O(log(N))$ memory space on each processor. Our protocol requires only $O(M_N)$ memory space on each processor, M_N being the smallest non divisor of ring size N (M_N is constant on the average). In [BGJ99a], it was been proven that the minimal memory space required by a self-stabilizing token circulation under any unfair distributed scheduler is $O(M_N)$. Thus, our protocol is optimal in memory space.
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. In section 3, we present SS_TC_Weak: a weak version of our protocol that can deal only with weak schedulers (k-bounded ones). The self-stabilization SS_TC_Weak is proven in section 4. In section 5, we present SS_TC: our self-stabilizing token circulation protocol for unidirectional anonymous rings of any size. The later protocol can deal with any unfair distributed scheduler.

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. In the sequel of our paper, all computation is assumed to be maximal.

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.

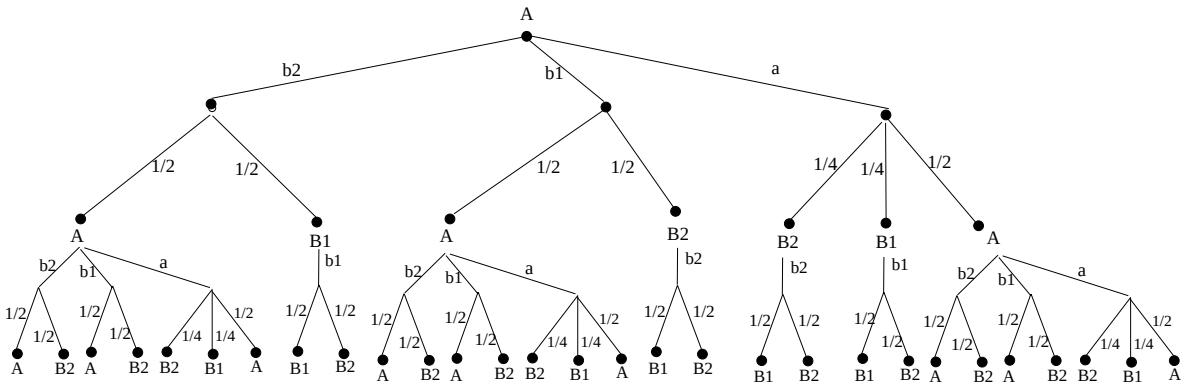

Figure 1: **The beginning of A-tree of the distributed system $TC2$**

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 $\rightarrow$ 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 [BCD95] on the unidirectional ring of size 2. Each processor has the same rule:

$Token_p \rightarrow$ 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 [BGJ99b]. 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 [Seg95]. 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.a) where $h1 = (A, a, B2)$ is $1/4$; the measure of $st2$-cone $\mathcal{C}_{h2}$ (figure 2.b) where $h2 = (A, b1, B2)$ is $1/2$.

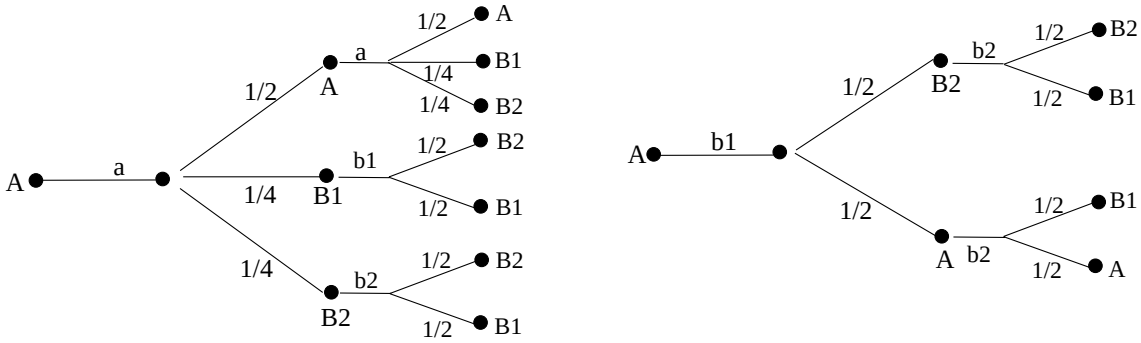

Figure 2: **The beginning of two $TC2$ strategy $st1$**

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 distributed* 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 $P_{st}(unfair\ computations) = \epsilon$. A k-bounded scheduler is the set of strategies that contain only k-bounded computations (i.e. any computation verifying the following property "until a processor p is enabled another processor can perform at most k actions").

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 [Her90, BCD95, KY97, BGJ99b, Ros00] 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, $P_{st}(\{e \text{ has a suffix that verifies } SP\}) = 1$.

Based on previous works on the probabilistic automata (see [Seg95], [SL94], [WSS94]) [BGJ99b] 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 [CM88] (for the deterministic systems) and [Seg95] (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, δ, 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 [BGJ99b] 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 *[BGJ99b] 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 Token circulation protocol under k-bounded schedulers

We present SS_TC_Weak, a self-stabilizing token circulation protocol for anonymous unidirectional rings of any size under k-bounded schedulers. A k-bounded scheduler selectes any strategy containing only k-bounded computations. In a k-bounded computation, until a processor p is enabled another processor can perform at most k actions. SS_TC_Weak is the conjunction of two protocol layers.

The first layer is the self-stabilizing token circulation protocol under k-bounded scheduler presented by Beauquier, Cordier and Delaët in [BCD95]. A processor having a token (called Mark) randomly decides to keep its mark or to pass it to its right neighbor (rule R_1 and R_2). Unfortunately, there is not upper bounded on the time needed by a Mark to perform a round. This protocol allows to

Protocol 3.1 SS_TC_Weak : Bound Service Time & Space optimal token circulation protocol

Field variables :

mr_p (the Mark value) is an integer bounded by M_N

c_p (the color value) takes value in $\{0, 1, 2\}$

Macros (l_p is p's left neighbor):

$Pass\_Mark_p = mr_p := (mr_{l_p} + 1) \bmod M_N$

Predicates :

$Mark_p \equiv mr_p \neq (mr_{l_p} + 1) \bmod M_N$

$Token_p \equiv (mark_p \wedge c_p = c_{l_p}) \vee (\neg mark_p \wedge c_p \neq c_{l_p})$

Rules :

$R_1 :: Mark_p \wedge \neg Token_p \rightarrow$
If $(random(0,1) = 0)$ then $\{Pass\_Mark_p; c_p := c_{l_p}\}$

$R_2 :: Mark_p \wedge Token_p \rightarrow$
If $(random(0,1) = 0)$ then $\{Pass\_Mark_p; c_p := c_{l_p}\}$
else $c_p := (c_p + 1 + random(0,1)) \bmod 3$

$R_3 :: \neg Mark_p \wedge Token_p \rightarrow c_p := c_{l_p}$

---

distinct one processor in the ring : the processor verifying the $Mark$ predicate. The distinguished processor changes time to time when the Mark moves.

The second layer is a self-stabilizing token circulation on semi-uniform rings of any size. The distinguished processor does not execute the same algorithm as the other processors. A standard processor (a processor that does not verify the $Mark$ predicate) has a token when it does not have the same color has its left neighbor. A standard processor passes its token by taking the color of its left neighbor (rule R_3). Unlike the others, the distinguished processor (processor that verifying the $Mark$ predicate) has a token when it does have the same color has its left neighbor; it passes the token by changing its color (rule R_2). In [Dij74], Dijkstra has presented a self-stabilizing token circulation based on this idea requiring $N + 1$ colors. In [Her92], Herman have presented an adaptation of the Dijkstra protocol requiring only 3 colors. The space improvement is possible because the distinguished processor randomly chooses its new color. To prove his protocol, Herman assumes that the scheduler is central and randomized: in any configuration, the scheduler randomly chooses one enabled processor. We have adapted the protocol of [Her92] to deal with a moving distinguished processor. We give a prove of our protocol under any k-bounded scheduler.

The service time of SS_TC_Weak does not depend on the output of random variables but only on the scheduling. Under a synchronous schedule a round requires only N steps. (section 4.3).

4 Self-stabilization Proof of SS_TC_Weak

We prove that the protocol 3.1 under a k-bounded scheduler converges to a legitimate configuration.

Definition 4.1 *A processor holds a token iff it verifies the predicate Token.*

Let T be a token held by p. The color of T is the color of the p's left neighbor.

The $L1$ predicate on configurations is defined by one and only one processor verifies the $Mark$ predicate. A configuration verifying $L1$ is said semi-legitimate.

The legitimate predicate L on configuration is : one processor and only processor has a Mark and one and only one processor verifies the Token predicate.

4.1 Convergence of SS_TC_Weak to a semi-legitimate configuration

In this section, we prove that $L1$ is an attractor. First we prove that $L1$ predicate is closed. Then, we prove that in strategy under a k-bounded scheduler, the probability to reach a semi-legitimate configuration is 1. The convergence proof to a semi-legitimate is very similar to the convergence proof of protocol [BCD95] under a k-bounded scheduler given in [BGJ99b]. We present the proof only for the sake of completeness.

Observation 4.1 *For any configuration, there is at least one processor that verifies the $Mark$ predicate because M_N does not divide the ring size N. Any processor p having a Mark is enabled. There is not deadlock configuration.*

Lemma 4.1 *In the protocol SS_TC_Weak under any scheduler, the predicate $L1$ is closed.*

Proof: During a computation step, either the processor holding a Mark (i.e. verifying the $Mark$ predicate) does not pass the Mark, or it passes its Mark. In the first case, it stays the only marked processor; in the seconde case, its neighbor becomes the new marked processor. □

Lemma 4.2 *Let k be an integer. Let st be a strategy of the protocol SS_TC_Weak under a k-bounded scheduler. Any cone of st satisfies LC ($true$, $L1$, $\frac{1}{2}^{kN^2+N^2}$, kN^3+N^2).*

Proof: Let C_h be a cone of st. Assume that in last(h), there are $m > 1$ marked processors. Let $p_1, p_2, \ldots p_m$ be the processors having a Mark in last(h). We name $M1$ (resp. $M2$) the Mark in p_1 (resp. p_2). Let d_1 be the distance between $M1$ and $M2$.

Let sc_1 be the following scenario : during the last step of sc_1 (i) the R_1 Mark moves from p_1 to p_1''s right neighbor; and (ii) $M2$ does not move during sc_1. At the end of sc_1, the distance between $M1$ and $M2$ has decreased

We study the sub-cone $\mathcal{C}_{hh_1}$ of $\mathcal{C}_h$ where : h_1 ends with an action of p_1 where it passes its Mark or by the merging of two Marks. h_1 contains at most one action of p_1; and during h_1 the processor p_2 keeps its Mark. h_1 realizes the sc_1 scenario or two Marks merge. As the scheduler is k-bounded, h_1 exists and other processors perform at most k actions in h_1. Each time that p_2 performs an action in h_1, its random variable takes the value 1; when p_1 performs its action, its random variable takes the value 0. We have $|h_1| \leq k(N-1)+1$ and $P_{st}(C_{hh_1}) \geq P_{st}(C_h).\frac{1}{2}^{k+1}$.

By repeating d_1 times the scenario sc_1, the system reaches a configuration where $M1$ has merged with $M2$. Let $C_{hh'}$ be a sub-cone of C_h, where h' realizes d_1 times the scenario sc_1 or two Marks merge. In last(h'), there are at most $m-1$ Marks. We have $|h'| \leq d_1(k(N-1)+1) \leq kN^2+N$, and $P_{st}(C_{hh'}) \geq P_{st}(C_h).\frac{1}{2}^{d_1(k+1)} \geq P_{st}(C_h).\frac{1}{2}^{N(k+1)}$.

By repeating $m-1$ times the scenario sc, the system reaches a configuration where there is one Mark. Let C_{hH} be a sub-cone of C_h, where H realizes $m-1$ times the scenario sc. We have $|H| \leq (m-1)(kN^2+N) \leq kN^3+N^2$, and $P_{st}(C_{hH}) \geq P_{st}(C_h).\frac{1}{2}^{(m-1)(kN^2+N)} \geq P_{st}(C_h).\frac{1}{2}^{N^2(k+1)}$. □

From Theorem 2.1 and Lemma 4.2, we get :

Theorem 4.1 *In any strategy st of protocol 3.1 under any k-bounded scheduler, the probability of the set of computations reaching $L1$ is 1.*

4.2 Convergence of SS_TC_Weak to a legitimate configuration

In this section, we study only the computations from a semi-legitimate configuration. We will prove that under a k-bounded scheduler, the probability to reach a legitimate configuration from a semi-legitimate configuration is 1, in any strategy. We also prove the the predicate L is closed.

Observation 4.2 *After that a processor performs an action during a computation step cs, it does not have a Token unless it left neighbor has changed its state during cs.*

Let p be a processor that does not have the color of its left neighbor. p has a Token or a Mark.

Lemma 4.3 *Let c be a configuration c that verifies $L1$. In c, there is at least one processor that has a Token.*

Proof: If all processors have the same color then the processor having the Mark has a Token.

Assume that there are at least two colors in the ring. Let $p1$ be processor having the Mark. Let $p2$ be the first processor at the right of $p1$ that does not have $p1$'s color. $p2$ has a Token or the Mark (Observation 4.2). As there are several colors in the ring $p2 \neq p1$. Thus $p2$ has a Token. □

Lemma 4.4 *Let cs be a computation step from a configuration c that verifies $L1$. Let p be a processor that has a Token after cs. Assume that p has not a Token in c or it has performed a rule. p has a Token after cs iff p's left neighbor has performed rule R_2 or R_3 during cs.*

Proof: We name q the p's left neighbor. If p has a Token in c then (by hypothesis) it has performed an action during cs. If q does not perform an action, during cs then p has not a Token after cs (Observation 4.2). If p does not have a Token in c and q does not perform any action during cs then clearly p has not a Token after cs. Thus q performs anction during cs.

Therefore, q has a Mark or/and a Token in c. We name co the color of q and co' the color of q left neighbor, in c.

Assume that q performed R_1 action during cs. Thus, q has the only Mark in the ring; and $co \neq co'$. We conclude that p has not a mark before cs. Therefore, during cs, either p performs R_3 action or not action. If p does not perform any action then p did not have a Token (by hypothesis) and neither a Mark in c. Therefore the p color is co before and after cs. If p performs action R_3, p has the color co after cs. In any case, after cs, p has the co color.

After cs, q has kept its state (i.e. it has kept its Mark) or it has passed its Mark to p. In the first case, p cannot not have a Token after cs (Observation 4.2). In the last case, q has changed its color during cs to get the color co'. After cs, p has a Mark and does not have the q's color: it does not have a Token. We conclude that q does not perform R_1 during cs.

We have proven that q performs rule during cs. This rule is R_2 or R_3. □

Corollary 4.1 *Let $T1$ be a Token hold by p. Along any computation after any action of p, $T1$ is held by p's right neighbor or its has vanished.*

Lemma 4.5 *Along any computation from any semi-legitimate configuration, the number of Tokens cannot increase.*

Proof: Let cs be a computation step from a configuration c that verifies $L1$. Let q_0 be a processor that has a Token after cs and not before cs. q_0's left neighbor, q_1, has performed rule R_2 or R_3 (Lemma 4.4). Thus, q_1 had a Token before cs. q_1 has still a Token after cs iff q's left neighbor (called q_2) has performed rule R_2 or R_3 during cs (Lemma 4.4).

By induction, we build a finite series of processors of $1 \leq m < N$ q_0, q_1, ... such that (i) $\forall i \in [0, m-1]$, q_i is at the right of q_{i+1} and q_i has a Token after cs; (ii) $\forall i \in [1, m]$,q_i has a Token in c; (iii) q_m has not a Token after cs; (iv) q_0 has not a Token in c. We conclude that along any computation, the number of Tokens cannot increase whatever may happen. □

Lemma 4.6 *L is a closed predicate.*

Proof: $L1$ is closed (Lemma 4.1). There is always a Token in the ring (Lemma 4.3). As, along any computation from any semi-legitimate configuration, the number of Tokens cannot increase (Lemma 4.5). We conclude that L is closed. □

Lemma 4.7 *Let k be an integer. Let st be a strategy of the protocol SS_TC_Weak under a k-bounded scheduler. Any cone of st satisfies LC ($L1$, L, $\frac{1}{2}^{(kN^2+kN+N)}$, kN^2+kN+N^3).*

Proof: Let C_h be a cone of st. Assume that in last(h), there are $m > 1$ Tokens. The Mark is on the processor P. Let p_1 (resp. p_2) be the first processor at the P left (resp. p_1 left) having a Token in last(h). We name $T1$ (resp. $T2$) the Mark in p_1 (resp. p_2). Let d_1 be the distance between $M1$ and P. Let d_2 be the distance between $M2$ and $M1$. We have $d_1 + d_2 \leq n$.

If P does not have a Token, we defined sc_1 has the following scenario where co is the color of the $T2$ Token: (preamble step) the $T1$ Token reaches P; (first step) the P does not takes the co color during the R_2 action. (second step) the $T2$ Token reaches P.
If P does have a Token we define sc_1 has the following scenario where co is the color of the $T1$ Token: (first step) P does not take the co color during the R_2 action; (second step) the $T1$ Token reaches P. In all cases, during cs_1 the Mark does not move. At the end of this scenario, all processors from $p1$ to P_l included have the same color: co. At the end, of cs_1, $T2$ has vanished. The Token $T2$ should be on P the marked processor but the color of P is not the color of its left neighbor P_l: P has not Token.

We study the $C_{hh'}$ sub-cones of C_h, where h' realizes (i) cs_1 or (ii) two Tokens merge. The duration of the preamble step and the first step is at most $(d_1+1)k$ steps under a k-bounded schedule: between two steps of $T1$, P performs at most k actions. Similarly, the duration of the second step is at most $d_2 k$ steps. During cs_1; no Token does catch up its preceding Token; thus any Token performs at most $2N$ steps. Therefore, $|h'| \leq (N+1)k+2Nm$ and $P_{st}(C_{hh'}) \geq P_{st}(C_h).\frac{1}{2}^{(d_1+d_2+1)k+1}$ $\geq P_{st}(C_h).\frac{1}{2}^{(N+2)k+1}$.

By repeating $m-1$ times the scenario sc, the system reaches a configuration where there is one Token. Let C_{hH} be a sub-cone of C_h, where H realizes $m-1$ times the scenario sc or $m-1$ Tokens have merged. We have $|H| \leq (N+1)mk + 2N^2m \leq kN^2+kN+2N^3$, and $P_{st}(C_{hH}) \geq P_{st}(C_h).\frac{1}{2}^{(kN^2+kN+N)}$. □

From Theorem 2.1 and Lemmas 4.2 and 4.7. we get :

Theorem 4.2 *In any strategy st of protocol 3.1 under any k-bounded scheduler, the probability of the set of computations reaching L is 1.*

4.3 Correctness of SS_TC_Weak

In this section we will prove that SS_TC_Weak protocol satisfies the specification *"one and only one Token fairly circulates in the ring and during a round, every processor obtains the Token exactly one time"* under a k-bounded scheduler.

Theorem 4.3 *Any computation of the protocol 3.1 under any k-bounded scheduler from a legitimate configuration satisfies the specification* "one and only one Token fairly circulates in the ring and during a round, every processor obtains the Token exactly one time"

Proof: In a legitimate configuration there are only one Token.

As L is closed, in any computation from a legitimate configuration there is always one and only one Token. The processor p holding the Token is enabled (R_2 or R_3). Therefore, p waits at most kN computation steps before performing an action under a k-bounded scheduler.

Let cs be the computation step where p performs an action (R_2 or R_3). As p has the only Token. during cs, p's left neighbor does perform action R_2 or R_3. p has not the Token after cs (Lemma 4.4). According to Lemma 4.4, p's right neighbor has the Token after cs.

After at most kN^2 computation steps, the Token has performed a round. Clearly during a round a processor gets exactly one time the Token. □

5 Token circulation protocol under unfair distributed schedulers

In [BGJ99a] is informally presented a protocol-compiler that transforms a self-stabilizing protocol under a k-bounded scheduler into an self-stabilizing protocol under any unfair distributed scheduler. A formal presentation may be found in [BGJ01]. After transformation of the protocol SS_TC_Weak, we get the protocol SS_TC (protocol 5.1).

The compiler modifies the rule $of SS\_TC\_Weak$, in such a way that holding a privilege is necessary to perform an action of SS_TC_Weak. After an action of SS_TC_Weak, the processor has passed the Privilege to its neighbor. If a processor holds a Privilege and it does not satisfy any guard of SS_TC_Weak it must pass its Privilege when it is chosen by the scheduler. When a processor passes a Privilege, if it can perform an action of SS_TC_Weak, it performs this action. We prevent the scheduler from having unfair behaviors: the scheduler cannot avoid choosing a processor. A processor satisfying a guard of SS_TC_Weak has to wait at most $N \cdot (N-1)/2$ computation steps before performing this action: another processor can perform at most N actions during the waiting.

The transformation ensures the following property: in any strategy of the protocol SS_TC under any unfair distributed scheduler, the probability to reach a legitimate configuration (a configuration where there is one Mark and one Token) is 1.

Property 5.1 *The properties of this transformation have been largely studied [BGJ99b], [Ros00], and [BGJ01]. One of the most interesting properties are:*

- *any computation has a suffix where the number of Privilege does not change;*
- *there is at least one Privilege in the ring;*
- *the number of Privileges cannot increase;*
- *The upper bound on the time needed by a Privilege to perform X rounds is $(X+1)N^2$ computation steps.*

Protocol 5.1 SS_TC : Bound Service Time & Space optimal token circulation protocol

Shared variables :

pr_p (the privilege value) is an integer bounded by M_N
mr_p (the Mark value) is an integer bounded by M_N
c_p (the color value) takes value in $\{0, 1, 2\}$

Macros (l_p is p's left neighbor):

$Pass\_privilege_p = pr_p := (pr_{l_p} + 1) \bmod M_N$
$Pass\_Mark_p = mr_p := (mr_{l_p} + 1) \bmod M_N$

Predicates :

$Mark_p \equiv mr_p \neq (mr_{l_p} + 1) \bmod M_N$
$Privilege_p \equiv pr_p \neq (pr_{l_p} + 1) \bmod M_N$
$Token_p \equiv (mark_p \wedge c_p = c_{l_p}) \vee (\neg mark_p \wedge c_p \neq c_{l_p})$

Rules :

$M_1 :: Privilege_p \wedge Mark_p \wedge \neg Token_p \rightarrow Pass\_Privilege_p;$
If $(random(0,1) = 0)$ then $\{Pass\_Mark_p; c_p := c_{l_p}\}$
$M_2 :: Privilege_p \wedge Mark_p \wedge Token_p \rightarrow Pass\_Privilege_p;$
If $(random(0,1) = 0)$ then $\{Pass\_Mark_p; c_p := c_{l_p}\}$
else $c_p := (c_p + 1 + random(0,1) \bmod 3)$
$M_3 :: Privilege_p \wedge \neg Mark_p \wedge Token_p \rightarrow Pass\_Privilege_p;\ c_p := c_{l_p}$
$M_4 :: Privilege_p \wedge \neg Mark_p \wedge \neg Token_p \rightarrow Pass\_Privilege_p$

5.1 Service Time of SS_TC

Let T be the Token that will stay forever in the ring. We prove that N T-token circulations require at most $4N^3$ computation steps. We also establishes that any computation from a legitimate configuration has a suffix where a Token circulation requires N steps under a synchronous scheduler.

Lemma 5.1 *Let e be a computation of SS_TC from a semi-legitimate configuration, We name T the Token that will stay forever in the ring.*

Assume that e reaches a configuration where a processor p holds T and a Privilege (called $Pr0$). Then along c, T and $Pr0$ will circulate forever together unless $Pr0$ merges with another privilege.

Proof: Let cs be the computation step where p performs an action (M_2 or M_3). As p has the T Token. During cs, According corollary 4.1, p right neighbor has the Token after cs. Thus after cs, p's right neighbor has the Token and the Privilege $Pr0$ unless p's right neighbor was holding a privilege and the two privileges vanish.

Let cs be a computation step, where p's left neighbor performs an action and p does not. After cs, two Privileges have merged (the Privilege that was on p's left neighbor has caught $Pr0$). During the merging, it is possible that two Privileges vanish. After cs, p has not any privilege. □

Lemma 5.2 *Let T be the Token that will stay forever in the ring. In any computation of SS_TC from a legitimate configuration, N T-rounds take at most $3N^3$ computation steps.*

Proof: Let c be a semi-legitimate configuration with m Privileges. We study a computation from c.

Let p be the processor having the Token T. If p does not have a Privilege then the Token T stays on p until p get a Privilege. p will get a Privilege in less than $2N^2$ computation steps (Property 5.1). Then, p has the Token T and a Privilege.

Now the Token and the Privilege will circulate together until the Privilege merges with another Privilege (Lemma 5.1) and the two Privileges vanish. This event can happen at most $(m-1)/2$ times during any computation from c. After such an event, the Token circulation is delayed by $2N^2$ computation steps (Property 5.1).

Once the Token T is joined with a Privilege, N T-token circulation needs $(N+1)N^2$ computation steps, assuming that there is no Privilege disparition (Property 5.1).

Therefore, N T-round requires, in the worst case, $(m-1)N^2+2N^2+(N+1)N^2 \leq 3N^3$ computation steps. □

Lemma 5.3 *Any computation of SS_TC from a legitimate configuration, has a suffix where a Privilege and the Token are always hold by the same processor.*

Proof: Let c be a legitimate configuration. We study a e computation from c.

e has a suffix e' where no Privilege disapeared. In e', the number of Privileges is constant. Let c be a configuration reached in e'. Let p be the processor having the Token in c. If p does not have a Privilege then the Token stays on p until p get a Privilege. After at most N^2 computation steps, p has the Token and a Privilege. □

Lemma 5.4 *Any computation under any schedulers of SS_TC from a legitimate configuration, has a suffix where a Token circulation requires at most N^2 steps. Any computation under a synchronous scheduler of SS_TC from a legitimate configuration, has a suffix where a Token circulation requires N steps.*

Proof: Let c be a legitimate configuration. We study an e computation from c.

e has a suffix e' where no Privilege disappeared and the Token and a Privilege are always on the same processor. The Token and the Privilege will circulate together forever (Lemma 5.1).

Under any scheduler, a T-round is a Privilege-round. Under any scheduler, X Privilege-rounds require at most $(X+N)N$ computation steps. Under a synchronous scheduler, a $Pr0$ round needs N computation steps. □

5.2 Stabilization Time of SS_TC

In this section, we interest to the time requires on the average by the protocol SS_TC to reach a legitimate configuration in any strategy under any scheduler. This time is called the stabilization time.

SS_TC is the conjunction of two protocol layers. The first layer is TC protocol (self-stabilizing Token circulation protocol under unfair distributed schedulers) presented in [BGJ99a]. TC is the transformation by the protocol-compiler presented in [BGJ01] of Token circulation presented protocol [BCD95]. The first layer protocol is a self-stabilizing protocol for the specification "the ring has only one Mark". Rosaz in [Ros00] proves that in any strategy, the average convergence time of protocol in [BGJ99a] is $O(N^3)$ computation steps.

Thus the average time requires by protocol SS_TC to reach a semi-legitimate configuration is $O(N^3)$ computation steps, in any strategy.

Now, we will compute the average time requires by protocol SS_TC to reach a legitimate configuration. First, we will compute the average time for a Token to reach the Mark (i.e. the Mark and the Token are held by the same processor).

Let c a a semi-legitimate configuration where there are several Tokens. Let T be the Token that will stay forever in the ring. We name $T1$ the first Token at the left of the Mark. We name $d1$ the distance from $T1$ to Mark. Assume that between $T1$ and the Mark there are $d2$ Privileges in c. We have $d2 \leq d1 < N$. We name ST_M (resp. ST_T) the number of steps performed by the Mark (resp. the Token $T1$). Rosaz in [Ros00] proves that along any computation from c, $ST_M \leq ST_T + d2$. In the average, the Mark moves from a processor to its right neighbor every two Mark steps. After, $3d1 < 3N$ steps of Token $T1$, in the average the Mark has moved at most $(3d1+d2)/2 \leq 2d1$ times. As the Token $T1$ moves during all its steps, we conclude that $T1$ is held by the marked processor in the average, after 3 rounds of $T1$. During this time, the Token T has performed at most 4 rounds.

The color of Mark was randomly computed (probability 1/2) when the Token that follows $T1$ was held by the marked processor. Therefore, the probability that $T1$ does not have the Mark color is 1/2. In that case the $T1$ vanishes when it reaches the marked processor. Thus, the expectation time to discard a Token is less than 8 T-rounds. The expectation time to reach a legitimate is less than $8N$ T-rounds (i.e. $O(N^3)$ computation steps - Lemma 5.2).

6 Conclusion

We have presented a randomized self-stabilizing token circulation protocol on unidirectional anonymous rings under unfair distributed scheduler. We have given a formal proof of the convergence of the protocol.
Once stabilized, our protocol provides a 1-fair token circulation: during a round each processor has only one time a token.
The memory space required by our protocol on each processor is $O(M_N))$. Our protocol is proven to be space optimal [BGJ99a].
Moreover the service time is always bounded ($O(N^3)$ computation steps to perform N token rounds). Under a synchronous scheduler, in N computation steps, each processor has a token.
The stabilization time is similar to the one of protocols [DGT00, KY02, Joh02]: $O(N^3)$ computation steps.

References

[BCD95] J. Beauquier, S. Cordier, and S. Delaët. Optimum probabilistic self-stabilization on uniform rings. In *WSS95 Proceedings of the Second Workshop on Self-Stabilizing Systems*, pages 15.1–15.15, 1995.

[BDLGJ02] 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.

[BGJ99a] J. Beauquier, M. Gradinariu, and C. Johnen. Memory space requirements for self-stabilizing leader election protocols. In *PODC99 Proceedings of the Eighteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 199–208, 1999.

[BGJ99b] 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.

[BGJ01] J Beauquier, M Gradinariu, and C Johnen. Cross-over composition - enforcement of fairness under unfair adversary. In *WSS01 Proceedings of the Fifth International Workshop on Self-Stabilizing Systems, Springer LNCS:2194*, pages 19–34, 2001.

[CM88] K. Chandy and J. Misra. *Parallel Programs Design: A Foundation*. Addison-Wesley, 1988.

[DGT00] A.K. 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.

[Dij74] E.W. Dijkstra. Self stabilizing systems in spite of distributed control. *Communications of the Association of the Computing Machinery*, 17:643–644, 1974.

[DL00] J. Durand-Lose. Randomized uniform self-stabilizing mutual exclusion. *Information Processing Letters*, 74(5-6):203–207, 2000.

[Her90] T. Herman. Probabilistic self-stabilization. *Information Processing Letters*, 35:63–67, 1990.

[Her92] T Herman. Self-stabilization: randomness to reduce space. *Distributed Computing*, 6:95–98, 1992.

[IJ90] 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.

[Joh02] Colette Johnen. Service time optimal self-stabilizing token circulation protocol on anonymous unidirectional rings. In *SRDS 2002 21st Symposium on Reliable Distributed Systems*. IEEE, October 2002.

[KY97] 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.

[KY02] 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.

[Ros00] 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.

[Seg95] R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, MIT, Departament of Electrical Engineering and Computer Science, 1995.

[SL94] 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.

[WSS94] S.H. Wu, S. A. Smolka, and E.W. 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.