Service Time of Self-Stabilizing Token Circulation Protocol on Anonymous Unidirectional Rings - Extended Abstract -

Colette Johnen†

Colette Johnen

LRI/UMR 8623 CNRS, Université Paris-Sud
Batiment 490, F-91405 Orsay Cedex
colette@lri.fr, www.lri.fr/∼colette/

We present a self-stabilizing token circulation protocol on unidirectional anonymous rings. The ring size is known by the processors. This protocol does not require processor identifiers, nor distinguished processor (i.e. all processors perform the same code).

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 $O(N^3)$ computation steps where N is the ring size. The memory space required by our protocol on each processor is $O(\lg(M_N))$, M_N being the smallest non divisor of ring size. Thus, we present the first protocol having the two major advantages: the duration of a token circulation is bounded and the protocol is optimal in memory space.

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

1 Introduction and Model

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 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 under any scheduler. 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.

†This work is partial supported by **Dynamo** Action Spécifique du Réseau Thématique Pluridisciplinaire CNRS/STIC Réseaux de Communication

State Model. Each processor p executes an algorithm. The algorithm consists of a set of variables and a finite set of guarded rules (i.e. *guard* $\rightarrow$ *action*). The guard of a rule on p is a boolean expression involving the state of p and the state of its left neighbor. The action of a rule updates the state of the processor that performs the rule. We assume that the rules are executed atomically. A processor, that has a true guard in a configuration c, is said enabled at c. During a *computation step*, one or more enabled processors execute one rule. A *computation* is a sequence of consecutive computation steps. Under a *centralized* scheduler, during a computation step, only one processor performs a rule. Along a *k-bounded* computation, until a processor p is enabled, another processor can perform at most k rules. A *k-bounded* scheduler "produces" only k-bounded computations. There is not restriction on the computations produced by an *unfair* scheduler.

Because, on anonymous networks, without the ability to break symmetry, deterministic self-stabilizing token circulation are impossible, our protocol and previous protocols are randomized. The formal framework we used to prove the converge of our randomized protocols is presented in [Joh02b].

Related works. The randomized token circulation protocols on unidirectional rings of [Her90, BCD95, KY97, BGJ99b, 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.

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 [Her90, BCD95, KY97, BGJ99b], then the upper bounded of the service time is infinite: a processor may never get the token because the token stays forever on the same processor.

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 the first 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 the unfair distributed scheduler. The service time is $2N$. In [Joh02b], we have presented a protocol ensuring an optimal service time: after N computation steps, each processor has obtained one time the token. These protocols [DGT00, KY97, KY02, Joh02b] require on each processor $O(\lg(N))$ bits.

In the figure 1, we compare the existing self-stabilizing token circulation protocols for anonymous unidirectional rings. All these protocols are randomized. In all existing protocols, the processors need to know the ring size. The model of computation is always the state model.

Fig. 1: Comparison of self-stabilizing token circulation protocols

Token circulation Protocol	**Memory Space**	**Upper bound on number of computation steps for N token circulations**	**scheduler required**
first protocol in [KY97]	$O(\lg(N))$	optimal : N^2	centralized
second protocol in [KY97]	$O(\lg(N))$	unbounded	unfair distributed
[DGT00]	$O(\lg(N))$	N^4	unfair distributed
[KY02]	$O(\lg(N))$	$2N^2$	unfair distributed
[Joh02b]	$O(\lg(N))$	optimal : N^2	unfair distributed
[Her90]	$O(\lg M_N))$	unbounded	synchronous
[BCD95]	$O(\lg M_N))$	unbounded	k-bounded
Protocol *SS_TC_Weak*	$O(\lg M_N))$	$O(k \cdot N^2)$	k-bounded
[BGJ99b], [Ros00]	$O(\lg M_N))$	unbounded	unfair distributed
[Ros00]	$O(\lg M_N))$	unbounded	unfair distributed
Protocol *SS_TC*	$O(\lg M_N))$	N	synchronous
Protocol *SS_TC*	$O(\lg M_N))$	$3N^3$	unfair distributed

Our contribution. Our protocols (called SS_TC_weak and SS_TC) require only $O(\lg(M_N))$ memory space on each processor, M_N being the smallest non divisor of ring size N. For instance, on a ring of odd size $M_N = 2$. The value of M_N is less than 11 for every ring whose the size is not a multiple of $2520 = 8 \times 9 \times 5 \times 7$. 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 the unfair distributed scheduler is $O(\lg(M_N))$. Thus, our protocols are optimal in memory space.

Notice that the previous protocols and our protocol require that each processor have some knowledge about the ring size: the value of M_N for the protocols of [Her90, BCD95, BGJ99b, Ros00, DGT00, BGJDL02] and our protocol; the ring size for the protocols of [KY97, KY02]; and an upper bound on the ring size for the protocol of [Joh02b].

The stabilization time of the Protocol SS_TC is similar to the one of protocols [DGT00, KY02, Joh02b]: $O(N^3)$ computation steps (the proof is out the scope of this paper).

The protocol SS_TC_weak works properly only under the k-bounded scheduler. The protocol SS_TC can deal with any kind of schedules even unfair ones. Once stabilized, the service time depends only of the scheduler. Under any scheduler, N token circulations in SS_TC (resp. SS_TC_weak) require at most $O(N^3)$ (resp. $k \cdot N$) computation steps. Thus, we present the first space optimal self-stabilizing token circulation protocols that has an upper bound on the service time. The service time is not optimal. A question is still opened: "there is or not a self-stabilizing protocol such that it has an optimal service time and it is optimal in memory space". We conjecture that such a protocol does not exist.

2 Token circulation protocol under the k-bounded scheduler

Protocol 2.1 SS_TC_Weak : Bound Service Time & Space optimal token circulation protocol

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 $Pass\_Mark_p$ else $c_p := (c_p + 1 + random(0,1)) \bmod 3$
$R_3 :: \neg Mark_p \wedge Token_p \rightarrow c_p := c_{l_p}$

We present SS_TC_Weak, a self-stabilizing token circulation protocol for anonymous unidirectional rings of any size under the k-bounded scheduler.

A processor holds a $Mark$ if and only if it verifies the predicate $Mark$: its value is not the value of its left neighbor plus one (modulo M_N). A processor having a $Mark$ randomly decides to keep its $Mark$ or to pass it to its right neighbor (rule R_1 and R_2). A processor passes its $Mark$ by changing its value (it takes the value of its left neighbor plus one).

Definition 2.1 *The $L1$ predicate on configurations is "one and only one processor has a Mark". A configuration verifying $L1$ is said semi-legitimate.*

Observation 2.1 *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.
The number of Marks cannot increase.
In the Protocol SS_TC_Weak under any scheduler, the predicate $L1$ is closed.

Informal theorem: Under a k-bounded schedule, from every configuration the probability to reach a semi-legitimate configuration is 1.

Informal proof: Let c be a semi-legitimate configuration where there are several *Marks*. Let $p_1, p_2, \ldots p_m$ be the processors having a *Mark* in c. 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: (i) the $M1$ *Mark* moves from p_1 to p_2; (ii) $M2$ does not move during sc_1.

Under a k-bounded schedule, we prove that cs_1 can achieve in less than $d_1(k(m-1)+1)$ computation steps with a probability greater than $\frac{1}{2}^{d_1(k+1)}$.

By repeating $m-1$ times the scenario sc_1 the system reaches a semi-legitimate configuration. □

Thus, a processor will be eventually distinguished in the ring: the processor verifying the *Mark* predicate. The distinguished processor changes from time to time when the *Mark* moves. The first layer of our protocol is a self-stabilizing "pseudo leader election".

The second layer is a self-stabilizing token circulation on semi-uniform rings. 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). Its new color is randomly computed. Notice that a processor holds a *Token* if and only if it verifies the predicate *Token*.

Definition 2.2 *The legitimate predicate $L2$ on configuration is "one and only one processor has a Mark and one and only one processor has a Token".*

Observation 2.2 *Let c be a semi-legitimate configuration. In c, there is at least one processor that has a Token.*
Along any computation from any semi-legitimate configuration, the number of Tokens cannot increase.
In the Protocol SS_TC_Weak under any scheduler, the predicate $L2$ is closed.

Informal theorem: Under a k-bounded schedule, from every semi-legitimate configuration, the probability to reach a legitimate configuration is 1.

Informal proof: Let c be a semi-legitimate configuration where there are several *Tokens*. Let $p_1, p_2, \ldots p_m$ be the processors having a *Token* in c. We name p_0 the processor having the *Mark*. Let p_1 (resp. p_2) be the first processor at the p_0 left (resp. p_1 left) having a *Token* in c. We name $T1$ (resp. $T2$) the *Token* in p_1 (resp. p_2).

We call sc_1 the following scenario in 3 steps where co is the color of the $T2$ *Token*: (preamble step) the $T1$ *Token* reaches p_0; (first step) p_0 does not take the co color during its R_2 action; and (second step) the $T2$ *Token* reaches p_0. During sc_1 the *Mark* does not move.

The scenario sc_1 is illustrated in the figure 2. At the end of sc_1, $T2$ has vanished. The $T2$ should be on p_0 the marked processor but the color of p_0 is not the color of its left neighbor p_l: p_0 has not a *Token*. Therefore at the end of sc_1, the number of *Tokens* has decreased.

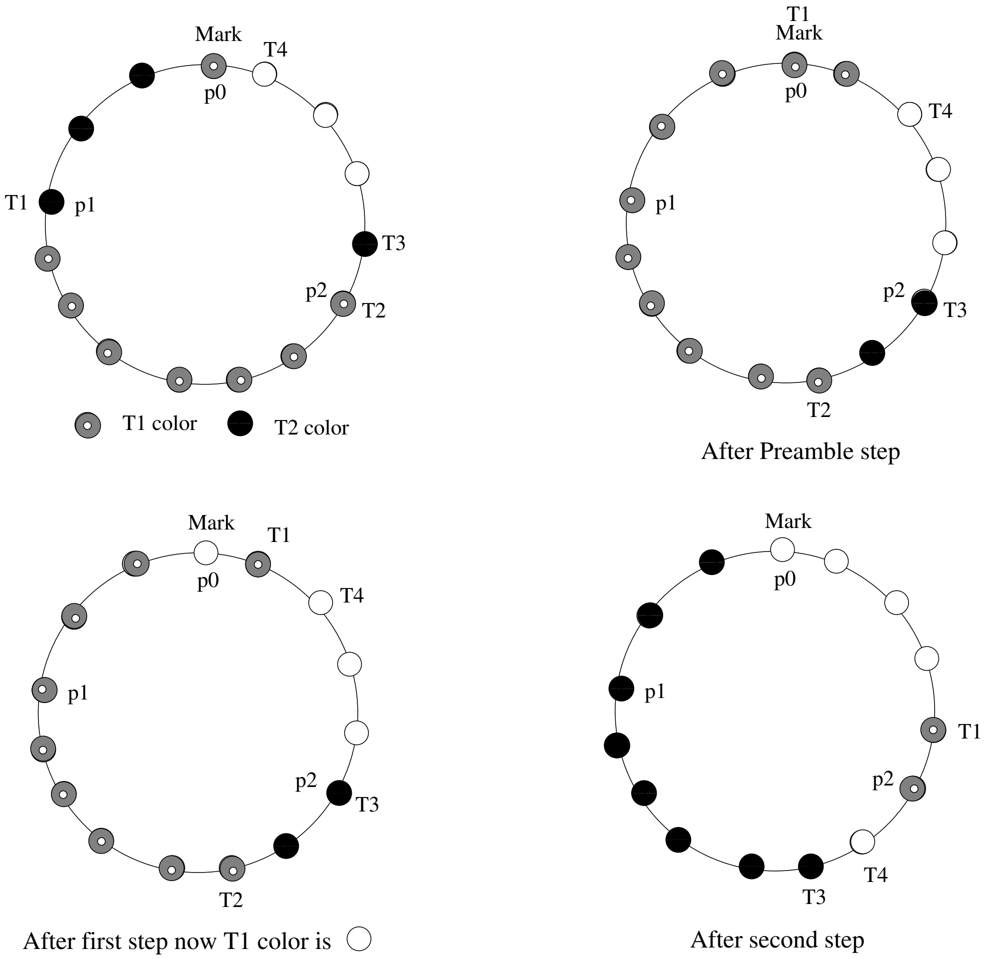

Fig. 2: Token discarding in Protocol *SS_TC_Weak*

Under a k-bounded schedule, we prove that cs_1 can achieve in less than $(2N+1)k+2Nm$ computation steps with a probability greater than $\frac{1}{2}^{(2N+1)k}$.

By repeating $m-1$ times the scenario sc_1 the system reaches a legitimate configuration. □

The formal convergence proof of our protocol can be found in [Joh02a].

3 Token circulation protocol under the unfair distributed scheduler

In [BGJ99a] is informally presented a protocol-compiler that transforms a self-stabilizing protocol under a k-bounded schedule into an self-stabilizing protocol under the 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 3.1).

The compiler modifies the rule of *SS_TC_Weak*, in such a way that a processor enabled in *SS_TC* holds a Privilege. During any action, a processor passes its Privilege to its right neighbor. A chosen processor by the scheduler, will perform the associated *SS_TC_Weak* rule action if it verifies a *SS_TC_Weak* guard. Also, 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 the action rule: another processor can perform at most N rules during the waiting.

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

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

The properties of this transformation have been largely studied [BGJ99b, Ros00, FJ01, BGJ01]. One of the most interesting properties are:

- any computation has a suffix where the number of Privileges 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.

Every processor having a Privilege is enabled. Even once the protocol *SS_TC* is stabilized, several processor may have a Privilege. Therefore, the token circulation speed depends on the processor schedule. Assume that the ring has m Privileges : Pri_1, Pri_2, ..., Pri_m. Let us study the following strictly centralized fair schedule: Pri_1, Pri_2, ..., Pri_m, Pri_1, Pri_2, ..., Pri_m, ... Under this schedule, a token circulation requires N^2 computation steps. Let T be the *Token* that will stay forever in the ring. In [Joh02a], we prove that N token circulations of T require at most $3N^3$ computation steps under all schedules. Under the synchronous scheduler, a token circulation takes N computation steps.

References

[BCD95] 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.

[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.

[BGJDL02] J. Beauquier, M. Gradinariu, C. Johnen, and J. Durand-Lose. Token-based self-stabilization uniform algorithms. *Journal of Parallel and Distributed Computing*, 62(5):899–921, 2002.

[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.

[FJ01] F. E. Fich and C. Johnen. A space optimal, deterministic, self-stabilizing, leader election algorithm for unidirectional rings. In *DISC01 Distributed Computing 15th International Symposium, Springer LNCS:2180*, pages 224–239, 2001.

[Her90] T. Herman. Probabilistic self-stabilization. *Information Processing Letters*, 35(2):63–67, 1990.

[Joh02a] C. Johnen. Optimization of service time and memory space in a self-stabilizing token circulation protocol on anonymous unidirectional rings. Technical Report 1330, L.R.I, September 2002.

[Joh02b] C. Johnen. Service time optimal self-stabilizing token circulation protocol on anonymous unidrectional. In *SRDS 2002 21st Symposium on Reliable Distributed Systems, IEEE Computer Society Press*, pages 80–89, 2002.

[KY97] H. Kakugawa and M. Yamashita. Uniform and self-stabilizing token rings allowing unfair daemon. *IEEE Transactions on Parallel and Distributed Systems*, 8(2):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, 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.