RAPPORT DE RECHERCHE

L R I

OPTIMAL SELF-STABILIZING MUTUAL EXCLUSION ON SYNCHRONOUS RINGS

DUCHON P / HANUSSE N / TIXEUIL S

Unité Mixte de Recherche 8623
CNRS-Université Paris Sud – LRI

05/2004

Rapport de Recherche N° 1389

CNRS – Université de Paris Sud
Centre d'Orsay
LABORATOIRE DE RECHERCHE EN INFORMATIQUE
Bâtiment 490
91405 ORSAY Cedex (France)

Optimal Self-stabilizing Mutual Exclusion on Synchronous Rings

Philippe Duchon Nicolas Hanusse Sébastien Tixeuil

LaBRI, Université Bordeaux I, 351 Cours de la Libération, 33400 Talence.
duchon,hanusse@labri.fr
LRI – CNRS UMR 8623 et INRIA Projet Grand Large, Université Paris-Sud XI.
tixeuil@lri.fr

Abstract

We propose several self-stabilizing protocols for unidirectional, anonymous, and uniform synchronous rings of arbitrary size, where processors communicate by exchanging messages. When the size of the ring n is unknown, we better the service time by a factor of n (performing the best possible complexity for the stabilization time and the memory consumption). When the memory size is known, we present a protocol that is optimal in memory (constant and independant of n), stabilization time, and service time (both are in $\Theta(n)$).
Keywords: Distributed Algorithms, Mutual Exclusion, Markov Chain, Stabilization Time, Service Time

Résumé

Nous proposons plusieurs algorithmes auto-stabilisants pour des anneaux synchrones, unidirectionels, anonymes et uniformes de taille arbitraire, où les processeurs communiquent en échangeant des messages. Quand la taille de l'anneau est inconnue, nous améliorons le temps de service d'un facteur n (en préservant la meilleure complexité connue pour le temps de stabilisation et la quantité de mémoire utilisée). Quand la taille de l'anneau est connue, nous présentons un algorithme qui est optimal en mémoire (constante et indépendante de n), en temps de stabilisation et en temps de service (tous deux sont en $\Theta(n)$).
Mots-clef : Algorithmes répartis, Exclusion mutuelle, Chaîne de Markov, Temps de stabilisation, Temps de service

Chapter 1

Introduction

The mutual exclusion one is a fundamental problem in the area of distributed computing. Consider a distributed system of n processors. Every processor, from time to time, may need to execute a critical section in which exactly one processor is allowed to use some shared resource. A distributed system solving the mutual exclusion problem must guarantee the following two properties: *(i) Mutual Exclusion*: one and only one processor is allowed to execute its critical section at any time; *(ii) Fairness*: each processor must be able to execute its critical section infinitely often. A classical technique consists in having every processor passing a special message called *token*. Holding a token means that the processor may enter its critical section. This token must satisfy mutual exclusion constraints: be unique and pass infinitely often at each processor.

The concept of self-stabilization was first introduced by Edsger W. Dijkstra in 1974 [5]. It is now considered to be the most general technique to design a system to tolerate arbitrary transient faults. A self-stabilizing system guarantees that starting from an arbitrary state, the system converges to a legal configuration in a finite number of steps and remains in a legal state until another fault occurs (see also [6]). Intuitively, a self-stabilizing token passing protocol for mutual exclusion guarantees that, even if the system is started from a global state where mutual exclusion specification is violated (zero or several tokens are present in the system), then within a finite number of steps, a unique token circulates fairly in the network.

In practise, we only prove that starting from a global state with several tokens, the system reaches in finite time a global state with a unique token. Indeed, it is proved in [11] that when processors communicate by exchanging messages, a *timeout* mechanism is required to spontaneously inject new tokens: if such a mechanism is not available, the system may not be self-stabilizing since it would be deadlocked when started from a message-free initial global state.

1.1 Previous works

A network is *uniform* if every processor executes the same code, and it is *anonymous* if processors do not have identifiers that would enable executing different sections of code. If a protocol works in a uniform and anonymous network, then it works *a fortiori* in a non-uniform or non-anonymous network. Since the first three self-stabilizing protocols about tokens passing presented in the pioneering paper [5], numerous works dealt with the same problem in various contexts: For the case of unidirectional anonymous and uniform rings., see [8, 1, 2, 4, 13, 10, 9].

A self-stabilizing token passing protocol is *transparent* to the application if it does not modify the format or the content of tokens that are exchanged by the application. Such a property is desirable *e.g.* whenever the token's content is used by the application (it is the case in *Token Ring* or *FFDI* networks, where the token also contains a piece of information to be transmitted to the destination). Indeed, a transparent protocol is easier to implement (it does not modify the frame format of the application that can thus be encrypted or compressed) and to integrate in heterogeneous networks (where some parts of the network use a different token passing protocol). Moreover, checking for message integrity can be delegated entirely to the calling application. Among the pre-cited protocols, only [8, 1] and the synchronous protocol of [4] are transparent to the upper layer application. In order to ensure stabilization, the protocols presented in [2, 4, 13, 10, 9] either make use of several types of tokens (thus adding a *type* field to application messages), or add information to each token in order to ensure stabilization.

A major criterion to evaluate the efficiency of self-stabilizing protocols is its *stabilization time* (or *convergence time*), noted T, that is the time needed to go from a global state with an arbitrary number of tokens to a global state with a unique token. As proved in [3], it is impossible to solve the problem of self-stabilizing token passing in a n-sized unidirectional anonymous and uniform ring using a deterministic protocol (except when n is prime, which is not a realistic assumption). Thus, the aforementioned solutions are probabilistic. Among those, [8, 1] do not provide any stabilization time calculus, and the expected stabilization times of [2, 4, 10, 9] are of order n^3, where n denotes the size of the ring, and that of [13] is of order n^2. It is obvious that the stabilization time for any algorithm is $\Omega(n)$: if the system is started from a configuration with two opposite tokens, at least $n/2$ time units are needed for a token to catch up with the other.

Another evaluation criterion is the *service time*, *i.e.* the time, in stabilized phase, between two tokens passing at the same processor. This criterion is important to evaluate the performance of the protocol when there are no faults and thus the overhead compared to a non stabilizing protocol. In a system with n processors, the service time is $\Omega(n)$, since if every processor waits the least amount of time, it waits as long as every other. The service time is not calculated in [8, 1, 2], and it is respectively of order n^3, n^2, n, and n in [4, 13, 10, 9].

1.2 Our contribution

We propose several self-stabilizing protocols for synchronous, anonymous, uniform and unidirectional ring networks, in which processors communicate by exchanging messages. The first two protocols are the transpositions in a message passing model of the protocols of [8, 1] (for the **0Memory** protocol) and [4] (for the **1bitMemory** protocol) that were using a shared memory model. By a tight complexity analysis, we prove that the equivalent of [8, 1] stabilizes in expected time $\Theta(n^2)$, and that its service time is $\Theta(n)$ (those complexities were not calculated in the quoted papers). Then, we show that the equivalent of the synchronous protocol of [4] converges in expected time $\Theta(n^2)$ (it was only proved $O(n^3)$ in the original paper) and has a service time of $\Theta(n)$. paper). The technique used for proving the **1bitMemory** protocol may be of independant interest for proving other complex probabilistic distributed algorithms using Markov chains.

Then, we propose several new protocols based on the notion of *speed reducer*. Each processor may declare itself to be a speed reducer and slow down token it receives with some probability. According to the power that is given to the speed reducer (*i.e.* the quantity of memory it has), the complexity results are different, but both new protocols we present have an expected convergence time, noted $\mathbb{E}(T)$, and an expected service time in $\Theta(n)$.

Protocol	Knowledge of n	Stabilization time	Service time * certain	Memory	Uniform	Transparent
[2]	yes	$\Theta(n^3)$	$O(n^3)$	$O(\log(n))$	yes	no
[9]	yes	$O(n^3)$	$^\star O(n)$	$O(\log(n))$	yes	no
[10]	yes	$O(n^3)$	$^\star O(n)$	$O(\log(n))$	yes	no
[13]	no	$\Theta(n^2)$	$O(n^2)$	$O(1)$	yes	no
0Memory	no	$\Theta(n^2)$	$\Theta(n)$	0	yes	yes
1bitMemory	no	$\Theta(n^2)$	$^\star\Theta(n)$	$O(1)$	yes	yes
SpeedReducer1	yes	$\Theta(n)$	$\Theta(n)$	$O(\log(n))$	yes	yes
SpeedReducer2	yes	$\Theta(n)$	$\Theta(n)$	$O(1)$	yes	yes

Figure 1.1: Results summary

All the protocols we present (see Figure 1.1) are transparent to the upper layer application. For some protocols (*i.e.* [9, 10],**1bitMemory**), not only the expected service time is upper bounded, but the bound is also certain (processors are ensured that the upper bound cannot be broken in any execution). When the size of the ring is unknown, the **1bitMemory** protocol is the most interesting (since the service time is certain). When the size of the ring is known, the **SpeedReducer2** protocol is the most interesting, being optimal according to the three evaluation criteria (memory consumption, stabilization time, and service time).

Chapter 2

Preliminaries

2.1 Execution model

We assume that processors are organized as a unidirectional ring of size n and communicate by exchanging messages. A processor may only receive messages from its predecessor and may only send messages to its successor. We consider a synchronous system where every processor takes an action at each *pulse* of a global clock. While processors are synchronized, there exists no global absolute time, only time pulses. Such a phase where all processors take action is called *round*. We also assume that any processor is able to query its underlying communication layer to know if a message (for example, a token) was delivered.

In this paper, we consider the problem of token passing. An algorithm is self-stabilizing for the token passing task if, starting from any configuration, after finite time there remains a unique token that pass forever at every processor. In the context of message passing systems, it is well known [11] that at least one timeout mechanism must be assumed in order to cope with initial configurations where no tokens are present. It is also assumed that those timeouts are activated at most once, and only at the begining of an execution. Thus, designing a self-stabilizing token passing algorithm is reduced to designing an algorithm that reduces tokens so that eventually, a single token remains. To ensure that the system never ends up in a deadlocked (*i.e.* message free) situation, all previous algorithms (and ours) use the technique of *token merging*: when two or more tokens are present at the same time at a single processor, all tokens are merged into one.

2.2 Markov Chains

We follow the terminology of [12] about Markov Chains. The classical hypothesis can be used since the network has a synchronous behaviour; for an asynchronous setting, see [7]. Let $P_{n\times n}$ be a stochastic matrix, that is the sum of every line is equal to 1. A *discrete Markov Chain*, denoted by $(X_t)_{t\leq 0}$ on a set of states X is a sequence of random variables $X_0, X_1, \ldots$ with $X_i \in X$ and so that X_{i+1} only depends on X_i and $\mathbb{P}r(X_{i+1} = y | X_i = x) =$

$p_{x,y}$. The matrix P is called the *transition probability matrix.*

A node x *leads* to a node y if $\exists j \geq i, \mathbb{P}r(X_j = y | X_i = x) > 0$. A state y is an *absorbing* state if y does not lead to any other state. The *expected hitting time* or *hitting time* $\mathbb{E}_x^y$ is the average number of steps starting from node x to reach node y for the first time.

We will make use of the following theorem for Markov chains :

Theorem 1 *The vector of hitting times* $\mathbb{E}^t = (\mathbb{E}_x^t : x \in V)$ *is the minimal non-negative solution of the following system of linear equations :*

$$\begin{cases} \mathbb{E}_t^t = 0 \\ \mathbb{E}_x^t = 1 + \sum_{y \neq t} p_{x,y} \mathbb{E}_y^t \text{for} x \in V \end{cases}$$

and the following lemma:

Lemma 2 (cf. [12] page 5) *Let* (X_t) *be a Markov Chain of two states* $\{1, 2\}$ *with transition matrix* P *defined by:* $P = \begin{pmatrix} 1-\alpha & \alpha \\ \beta & 1-\beta \end{pmatrix}$ *then*

$$\mathbb{P}r(X_t = 1 | X_0 = 1) = \begin{cases} \frac{\beta}{\alpha+\beta} + \frac{\alpha}{\alpha+\beta}(1-\alpha-\beta)^t & \text{for} \alpha + \beta > 0 \\ 1 & \text{for} \alpha = \beta = 0 \end{cases}$$

Chapter 3

Algorithms

The first protocol we propose is equivalent to a random walk of tokens in the ring, and can be seen as a transposition in the message passing model of the protocols of [8, 1]:

3.1 0Memory protocol

Each processor i executes the following code: at each pulse, if i is currently holding a token, it transmits it to its successor with probability p, and keeps it with probability $1-p$. Remark that the protocol runs without memory and without a knowledge of n.

Applying Theorem 1 to a specific Markov chain, we obtain a useful Lemma for the analysis of **0Memory** protocol :

Lemma 3 *Let C_d be a chain of $d+1$ states $0, 1, \ldots, d$ and $q \in]0, 1/2]$. If state 0 is absorbing and the transition matrix is of the form :*

$$\begin{cases} p_{i,i-1} = p_{i,i+1} = q \text{ for } 1 \leq i \leq d-1 \\ p_{i,i} = 1-2q \text{ for } 1 \leq i \leq d-1 \\ p_{d,d} = 1-q \end{cases}$$

then the hitting time to state 0 starting from state i is $\mathbb{E}_i^0 = \frac{i}{2q}(2d-i+1)$.

Proof. We make a use of Theorem 1 for the computation of $\mathbb{E}_i^0$. We have

$$\begin{cases} \mathbb{E}_1^0 = 1 + (1-2q)\mathbb{E}_1^0 + q\mathbb{E}_2^0 \\ \mathbb{E}_i^0 = 1 + q\mathbb{E}_{i-1}^0 + (1-2q)E_i^0 + q\mathbb{E}_{i+1}^0 \text{ for } 2 \leq i \leq d-1 \\ \mathbb{E}_d^0 = 1 + (1-q)\mathbb{E}_d^0 + q\mathbb{E}_{d-1}^0 \end{cases}$$

Noting that $\mathbb{E}_i^0 = \sum_{j=1}^{i} \mathbb{E}_j^{j-1}$, we are interested by $\mathbb{E}_j^{j-1}$ for $1 \leq j \leq d$. Therefore,

$\mathbb{E}_d^{d-1} = 1 + (1-q)\mathbb{E}_d^{d-1} = 1/q$ and

$$\begin{aligned}
\mathbb{E}_j^{j-1} &= 1 + (1-2q)\mathbb{E}_j^{j-1} + q\mathbb{E}_{j+1}^{j-1} \\
&= 1 + (1-2q)\mathbb{E}_j^{j-1} + q(\mathbb{E}_{j+1}^{j} + \mathbb{E}_j^{j-1}) \\
&= 1/q + \mathbb{E}_{j+1}^{j} \\
&= \frac{d-j}{q}
\end{aligned}$$

This implies that $\mathbb{E}_i^0 = \sum_{j=1}^{i}(d-j)/q = \frac{1}{q}(di - \frac{i(i-1)}{2})$. ■

Theorem 4 *In a unidirectional n-sized ring containing an arbitrary number k of tokens ($k \geq 2$), the stabilization time of protocol **0Memory** is $\frac{n^2}{8p(1-p)} < \mathbb{E}(T) < \frac{n^2}{2p(1-p)}(\frac{\pi^2}{6} - 1) + \frac{n \log n}{p(1-p)}$. For constant p, $\mathbb{E}(T) = \Theta(n^2)$.*

Proof. For any $k \geq 2$, the evolution of the ring with exactly k tokens under the **0Memory** protocol can be described by a Markov chain $\mathcal{S}_k$ whose state space is the set of k-tuples of positive integers whose sum is equal to n (these integers represent the distances betweeen successive tokens on the ring), with an additional state $\delta = (0, \ldots, 0)$ to represent transitions to a configuration with fewer than k tokens. To prove the upper bound of the theorem, we will prove an upper bound on the hitting time of this state δ, independently of the initial state.

Consider two successive tokens on the ring. On any given round, each will move forward, independently of the other, with probability p, and stay in place with probability $1-p$. Thus, with probability $p(1-p)$, the distance between them will decrease by 1; with the same probability, it will increase by 1; and, with probability $1 - 2p(1-p)$, the distance will remain the same. Thus, locally, the distance between consecutive tokens follows the same evolution rule as that of the chain $\mathcal{C}_n$ of Lemma 3.

What follows is a formal proof, using the technique of *couplings* of Markov chains, that the expected time it takes for two tokens among k to collide is no longer than the expected time for $\mathcal{C}_{n/k}$ to reach state 0.

For any state $\mathbf{x} = (x^1, \ldots, x^k)$ of $\mathcal{S}_k$, let $m(\mathbf{x}) = \min_i x^i$ denote the minimum distance between two successive tokens, and let $i(\mathbf{x}) = \min\{j : x^j = m(\mathbf{x})\}$ denote the smallest index where this minimum is realized. Let $(X_t)_{t\geq 0}$ denote a realization of the Markov chain $\mathcal{S}_k$. We define a *coupling* $(X_t, Y_t)_{t\geq 0}$ of the Markov chains $\mathcal{S}_k$ and $\mathcal{C}_d$, where $d = \lfloor n/k \rfloor$ and $q = p(1-p)$, as follows :

- $Y_0 = m(X_0)$;
- $Y_{t+1} = \min\{d, Y_t + (X_{t+1}^{i(X_t)} - X_t^{i(X_t)})\}$

In other words, the evolution of Y_t is determined by selecting two tokens that are separated by the minimum distance in X_t, and making the change in Y_t reflect the change in distance between these two tokens (while capping Y_t at d).

A trivial induction on t shows that $Y_t \geq m(X_t)$ holds for all t, so that (X_t) will reach state δ no later than (Y_t) reaches 0. Thus, *the time for $\mathcal{S}_k$ to reach δ (that is, the time during which the ring has exactly k tokens) is stochastically dominated by the time for $\mathcal{C}_d$ to reach* 0. By Lemma 3, the expectation of this time is no longer than

$$\frac{d(d+1)}{2q} \leq \frac{1}{2q}\left(\frac{n^2}{k^2} + \frac{n}{k}\right)$$

Summing over all values of k from 2 to n, we get, for the expected stabilization time T,

$$\begin{aligned}\mathbb{E}(T) &\leq \frac{1}{2p(1-p)} \sum_{k=2}^{n} \frac{n^2}{k^2} + \frac{n}{k} \\ &\leq \frac{1}{2p(1-p)}\left(\left(\frac{\pi^2}{6} - 1\right) n^2 + n\ln(n)\right)\end{aligned}$$

The lower bound comes from the fact that, when $k = 2$, the expected time for $\mathcal{C}_{n/2}$ to reach state 0 from state $n/2$ is at least $\frac{n^2}{8p(1-p)}$. ■

Remark 5 *Our upper bound on the expected convergence time is minimal for $p = 1/2$. The precise study of the **0Memory** protocols show that the convergence time hardly depends on the initial number of tokens: for n high enough and $p = 1/2$, $\mathbb{E}(T) > n^2/2$ for two initial tokens, and $\mathbb{E}(T) < 1.3n^2$ for n tokens.*

The expected service time is exactly n/p. The following protocol, **1bitMemory**, guarantees that the the service time can be upper bounded by $2n$ with probability 1. This allows to ease the implementation of the timeout mechanism by inserting a new token if no token has been encountered for $2n$ rounds.

3.2 1bitMemory protocol

Each processor i executes the following code: at each pulse, if a token was just delivered to i, then i executes the **0Memory** protocol for one pulse. If a token was delivered to i at the previous round, then i transmits the token to its successor on the ring. Processor i needs only one bit of memory to store whether it received a token at the previous round. Remark that the protocol runs with one bit of memory and without a knowledge of n.

The analysis of this protocol leads to the following result:

Theorem 6 *In a unidirectional n-sized ring containing an arbitrary number k of tokens ($k \geq 2$), the service time of protocol **1bitMemory** is between n et $2n$ with probability 1 and the stabilization time is $\mathbb{E}(T) = O(n^2)$ for constant p.*

Proof. The state of the ring with k tokens is described by a Markov chain $\mathcal{S}'_k$, whose states are of the form $\mathbf{x} = (x^1, s^1, \ldots, x^k, s^k)$. In such a state, x^i denotes the distance between tokens i and $i+1$, and $s^i \in \{0, 1\}$ keeps track of the current state of the node holding token i (0 corresponds to a node who just received a token, and 1, to a node that got its token one round ago). If we only look at two consecutive tokens – that is, the part (s^i, x^{i+1}, s^{i+1}) for some i – we can see that this will evolve according to a Markov chain $\mathcal{C}'_d$, part of whose state space and transition probabilities are represented in Figure 3.1.

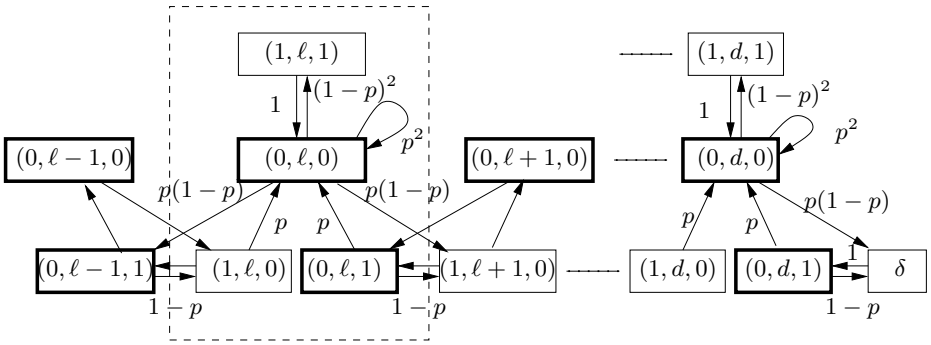

Figure 3.1: Transition probabilities of chain $\mathcal{C}'_d$

Our proof of the upper bound on the expected collision time (in the chain $\mathcal{S}'_k$) is in two steps. In the first part of the proof, we construct a kind of coupling between $\mathcal{S}'_k$ and $\mathcal{C}'_d$ (with the caveat that we have to use a trick where one of the chains in the coupling may have to be "stopped" from time to time while the other "catches up" with it; *i.e.*, until $Y_t = m(X_t)$); this allows us to prove that the expected collision time is lower than the expected hitting time of state $(0, 0, 1)$ in $\mathcal{C}'_d$. In the second step, we use yet another Markov chain to prove an upper bound on this expected hitting time.

First, we need to introduce some notation. If $\mathbf{x} = (x^1, s^1, \ldots, x^k, s^k)$ is some state of $\mathcal{S}'_k$ and $1 \leq i \leq k$, $d_i(\mathbf{x}) = x_i$, and $I_{\min}(\mathbf{x})$ is the smallest index j such that $d_j(\mathbf{x})$ is minimal. Also, $m_i(\mathbf{x}) = (x^i, s^i, x^{i+1})$ is a mapping of the states of $\mathcal{S}'_d$ to the states of $\mathcal{C}'_d$, with the special provision that, if $d_i(\mathbf{x}) > d$, then $m_i(\mathbf{x}) = \delta$. The set of "good" states A of $\mathcal{C}'_d$ is the set of states of the form $(0, \ell, 0)$ or $(0, \ell, 1)$, plus the special state δ; note that any transition from a state *not* in A is guaranteed to lead to a state in A – see Figure 3.1.

The idea of our coupling is as follows: we start with a copy (X_t) of the chain $\mathcal{S}'_k$, and initially select two consecutive tokens in the configuration; the distance between these two tokens will be used to define a copy (Y_t) of the chain $\mathcal{C}_d$. We have to be careful, however, if the two tokens ever get too far apart from each other (that is, their distance grows larger than d). In such a case, we select two "new" tokens with the minimal distance between them, and "stop" the chain (X_t), while still running the chain (Y_t) until it "catches up" with the distance between the two newly selected tokens in (X_t). This last part is only possible because of the special structure of the Markov chain $\mathcal{C}'_d$.

More precisely, our coupling will be a Markov chain $(X_t, Y_t, I_t, T_t)_{t \geq 0}$. Here, T_t indicates the number of transitions (X) has gone through when (Y) has gone through t transitions

(it is a "change of time"); I_t simply keeps track of which two tokens of X_t we are currently "watching". The coupling is constructed as follows : X_0 is any state of $\mathcal{S}'_k$, $T_0 = 0$, $I_0 = I_{\min}(X_0)$, $Y_0 = m_{I_0}(X_0)$; then, the transition from time t to $t+1$ is done as follows :

- if $Y_t = m_{I_t}(X_t)$: this corresponds to the "normal" operation of the coupling. X_{t+1} is selected randomly, according to the evolution law of $\mathcal{S}'_k$, and we set $T_{t+1} = T_t + 1$, $Y_{t+1} = m_{I_t}(X_{t+1})$, then I_{t+1} is determined depending on $d_{I_t}(X_{t+1})$:
  - if $d_{I_t}(X_{t+1}) \leq d$, then $I_{t+1} = I_t$;
  - if $d_{I_t}(X_{t+1}) > d$ (in which case we have $Y_{t+1} = \delta$, then $I_{t+1} = I_{\min}(X_{t+1})$ (that is, I_{t+1} is now the index of a new pair of consecutive tokens in the configuration described by X_{t+1})
- if $Y_t \neq m_{I_t}(X_t)$, then we need to distinguish whether $X_t \in A$ (in which case we "stop" X_t) or not (in which case running X_t for one more round will ensure $X_{t+1} \in A$) :
  - if $X_t \in A$, then we set $X_{t+1} = X_t$, $T_{t+1} = T_t$, $I_{t+1} = I_t$, and select Y_{t+1} randomly, according to the evolution law of $\mathcal{C}'_d$;
  - otherwise, we set $T_{t+1} = T_t$, $I_{t+1} = I_t$, and select X_{t+1} and Y_{t+1} independently, according to the evolution laws of $\mathcal{S}'_k$ and $\mathcal{C}'_d$, respectively.

It should be clear that $(Y_n)_{n\geq 0}$ is a faithful copy of the chain $\mathcal{C}'_d$. Next, if we set, for all $n \geq 0$, $\tau_n = \inf\{t : T_t \geq n\}$ (so that we have $T_{\tau_n} = n$ with probability 1, and thus $\tau_n \geq n$), then $(X_{\tau_n})_{n\geq 0}$ is a faithful copy of the chain $\mathcal{S}'_k$. Furthermore, $d_{I_t}(X_t) \leq d(Y_t)$ holds for all t (the only evolution rules where this might conceivably not be maintained are those that apply when $Y_t \neq m_{I_t}(X_t)$; the key observation is that the chain $\mathcal{C}'_d$ cannot go from a state (x, ℓ, y) to a state (x', ℓ', y') with $\ell' < \ell$, without going through states $(0, \ell', 1)$, then $(0, \ell', 0)$).

Thus, if we define hitting times $t_X = \inf\{t \geq 0 : d_{I_{\min}}(X_t) = 0\}$ and $t_Y = \inf\{t \geq 0 : d(Y_t) = 0\}$, then we have $t_X \leq t_Y$, thus $T_{t_X} \leq t_Y$. But the expected value of T_{t_X} is the expected collision time of the chain $\mathcal{S}'_k$, while the expected value of t_Y is the expected hitting time of state $(0, 0, 1)$ for the chain $\mathcal{C}'_d$ (starting from states X_0 and Y_0, respectively), which ends the first part of the proof.

To give an upper bound on the hitting time of $(0, 0, 1)$ for $\mathcal{C}'_d$, we consider the simpler chain $\mathcal{D}_d$, which is exactly the chain $\mathcal{C}'_d$, stopped at each passage in a state of A; the transition probabilities of $\mathcal{D}_d$ are as illustrated in Figure 3.2. Note that, because, in $\mathcal{C}'_d$, states not in A have only states in A as their successors, a copy of $\mathcal{C}'_d$ will make at most twice as many steps as the corresponding (coupled) copy of $\mathcal{D}_d$; this ensures that the expected hitting time of state $(0, 0, 1)$ is at most twice as large in $\mathcal{C}'_d$ as it is in $\mathcal{D}_d$.

For convenience, we rename the states of D_d in the following way : $(0, \ell, 0)$ will be denoted by 2ℓ, $(0, \ell, 1)$ by $2\ell + 1$ and δ by $2d + 2$. For any $\ell \leq 2d + 2$, let us upper bound $\mathbb{E}^1_\ell$ in chain D_d. Since to go from state $j > i$ to state i, we should visit all the states from $j - 1, j - 2, \ldots, i + 1$, we have in D_d that $\mathbb{E}^i_j = \sum_{k=i}^{j-1} \mathbb{E}^{k+1}_k$.

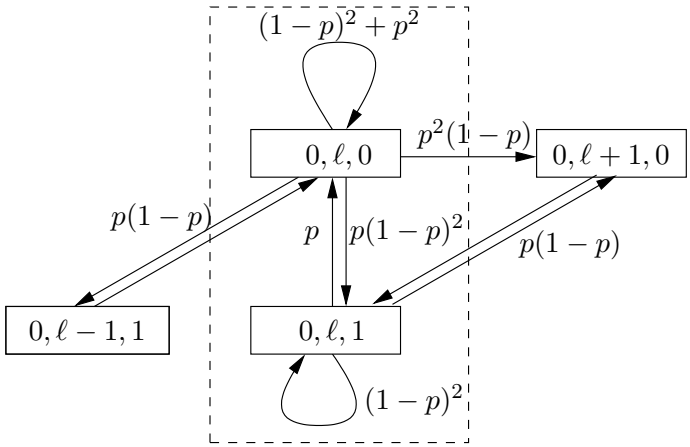

Figure 3.2: Transition probabilities of chain $\mathcal{D}_d$

We make use of Theorem 1 to obtain, for $k < d$:

$$\begin{aligned}
\mathbb{E}_{2k+1}^{2k} &= \frac{1}{p} + (1-p)\mathbb{E}_{2k+2}^{2k+1} \\
\mathbb{E}_{2k}^{2k-1} &= \frac{1}{p(1-p)} + \mathbb{E}_{2k+1}^{2k} + p\mathbb{E}_{2k+2}^{2k+1} \\
&= \frac{2-p}{p(1-p)} + \mathbb{E}_{2k+2}^{2k+1}
\end{aligned}$$

It turns out that

$$\begin{aligned}
\mathbb{E}_{2k}^{2k-1} &= (d-k)\frac{2-p}{p(1-p)} + \mathbb{E}_{2d}^{2d-1} \\
\mathbb{E}_{2k+1}^{2k} &= \frac{1}{p} + \frac{(d-k-1)(2-p)}{p} + (1-p)\mathbb{E}_{2d}^{2d-1}
\end{aligned}$$

The recurrence can be completed after the computation of $\mathbb{E}_{\delta}^{2d+1}$, $\mathbb{E}_{2d+1}^{2d}$ and $\mathbb{E}_{2d}^{2d-1}$:

$$\begin{aligned}
\mathbb{E}_{\delta}^{2d+1} &= 1 \\
\mathbb{E}_{2d+1}^{2d} &= 1 + (1-p)\mathbb{E}_{\delta}^{2d} \\
&= \frac{2-p}{p} \\
\mathbb{E}_{2d}^{2d-1} &= 1 + \left(p^2 + (1-p)^2\right)\mathbb{E}_{2d}^{2d-1} + p(1-p)\mathbb{E}_{\delta}^{2d-1} \\
&= \frac{3-2p}{p(1-p}
\end{aligned}$$

Since $\mathbb{E}_{\ell}^{1} \leq \mathbb{E}_{\delta}^{1}$, summing $\mathbb{E}_{k}^{k-1}$ for $k = 2 \ldots 2d$, we obtain that:

$$\begin{aligned} t_Y &\leq 2\left(\mathbb{E}_\delta^{2d+1} + \sum_{k=2}^{2d+1} \mathbb{E}_k^{k-1}\right) \\ &\leq d^2\left(\frac{2-p}{p} + \frac{p+2}{p(1-p)}\right) + d\left(\frac{4-3p}{p(1-p)} + \frac{2-p}{p}\right) + 2 + \frac{4-2p}{p} + \frac{6-4p}{p(1-p)} \end{aligned}$$

Subsituying d by $\frac{n}{k}$ and summing t_Y's over all values of k from n downto 2, we obtain that the stabilization time is $O(n^2)$ for constant p. ■

We now consider the possibility that some particular processors have a different behavior from the others, with some probability. We call these processors *speed reducers*: each round, *normal* processors always transmit any token they are holding, while *speed reducers* do not transmit them.

3.3 SpeedReducer1 protocol

Each processor i executes the following code: every n pulses, processor i randomly decides whether it must act as a speed reducer or not. Processor i acts as a speed reducer with probability $1/2n$.

Theorem 7 *In a n-sized ring containing an arbitrary number k of tokens ($k \geq 2$) in its initial state, the protocol **SpeedReducer1** has a stabilization time upper bounded by $(1 + 3e^{3/2})n$.*

Proof. Note that convergence is to occur as soon as, during a lapse of time of length n, a unique processor is a speed reducer. We estimate the average time before this situation happens in order to bound the expected convergence time.

A processor has two possible states: normal or speed reducer. At each round, the configuration of the ring (discounting token positions) can be described by a binary vector of n states. Let C_k be the configuration at time kn and let R_k its number of speed reducers. The probability $\mathbb{P}r(R_k = 1)$ of the existence of a unique speed reducer in the configuration C_k is:

$$\begin{aligned} \mathbb{P}r(R_k = 1) &= \frac{1}{n}\left(1 - \frac{1}{2n}\right)^{n-1}\binom{n}{1} \\ &= e^{(n-1)\ln(1-1/2n)} \\ &\geq e^{\frac{(1-n)}{n}}. \end{aligned}$$

For $x \in]0, 1/2], \ln(1-x) \geq -2x$. This implies that for $n \geq 2, \mathbb{P}r(R) \geq e^{-1/2}$.

Similarly, we have $\mathbb{P}r(R_k = 0)\left(1 - \frac{1}{2n}\right)^n \geq e^{-1/2}$.

Let us define $\mathcal{E}_k$ the event "$R_{k-1} = 0, R_k = 1$ and $R_{k+1} = 0$", and assume $\mathcal{E}_k$ occurs. Let i be the unique speed reducer node at time kn, and let t denote the last time before kn

where it had to decide on its state; $(k-1)n < t \leq kn$. $\mathcal{E}_k$ implies that, from time $(k-1)n$ to time $(k+1)n$, all nodes other than i chose to be *normal* whenever they had to choose. Thus, i is the only speed reducer from time t to time $t+n-1$, and convergence occurs no later than time $t+n-1 < (k+1)n$.

From what has been written previously, $\mathbb{P}r(\mathcal{E}_k) \geq e^{-3/2}$. If J is the smallest integer j such that $\mathcal{E}_{3i+1}$ occurs, J has an expected value no larger than $e^{3/2}$ (the events $(\mathcal{E}_{3i+1})_{i\geq 0}$ are independent, contrary to $(\mathcal{E}_i)_{i\geq 0}$). As a consequence, the expected stabilization time is less than $(1+3e^{3/2})n$. ■

In order to count n pulses, protocol **SpeedReducer1** requires logarithmic memory. The next protocol **SpeedReducer2** does not need that much space, and only uses a constant memory (that is independent of n).

3.4 SpeedReducer2 protocol

Each processor i executes the following code: at each pulse, if the state of the processor is *normal*, it becomes a *speed reducer* with probability $q = \frac{1}{n(n-1)}$; otherwise (if the state of the processor is *speed reducer*), it becomes *normal* with probability $p = 1/n$.

Theorem 8 *In a n-sized ring containing an arbitrary number k of tokens ($k \geq 2$) in its initial state, the protocol **SpeedReducer2** has a stabilization time upper bounded by $n(1+\frac{e^4}{e-1})$ and only needs one bit of memory.*

Proof. Let us show that after n rounds, we have a constant probability to have a unique speed reducer during n rounds.

The state of each node evolves independently from the other nodes like a two states Markov chain of transition probabilities $\alpha = \frac{1}{n(n-1)}$ and $\beta = \frac{1}{n}$. Let $\mathcal{E}_{x,t}$ be the event "processor x is in the normal state at time t". From Lemma 2, $\mathbb{P}r(\mathcal{E}_{x,t}) = 1 - \frac{1}{n} + \frac{1}{n}\left(1-\frac{1}{n-1}\right)^t$. For $t \geq n-1$, $1-1/n \leq \mathbb{P}r(\mathcal{E}_{x,t}) \leq 1-1/n+e^{-1}/n$. Let R_t be the number of speed reducers at time t. The probability that there exists a unique speed reducer at time $t \geq n-1$ is:

$$\mathbb{P}r(R_t = 1) \geq \frac{1-e^{-1}}{n}\left(1-\frac{1}{n}\right)^{n-1}\binom{n}{1} \geq \frac{1-e^{-1}}{e}.$$

The probability that a speed reducer stays in the same state during n rounds is $(1-1/n)^n = e^{-1}+o(1)$ and the probability that the $n-1$ other nodes remain normal during n rounds is $(1-\frac{1}{n(n-1)})^{n(n-1)} = e^{-1}+o(1)$.

Therefore, for any configuration of states at time 0, the probability that a node is a unique speed reducer during n rounds between $t=n$ and $t=2n-1$ (implying the total merger between tokens takes place before time $2n$) is asymptotically greater than $p = \frac{e-1}{e^4}$. The same is true every n rounds thereafter until stabilization occurs, so that the probability that stabilization fails to occur in the first $(k+1)n$ rounds is at most p^k; this, in turn, implies that the expected stabilization time is at most $n\left(1+\frac{e^4}{e-1}+o(1)\right)$. ■

Chapter 4

Conclusion

We propose several self-stabilizing protocols for synchronous, anonymous, uniform, and unidirectional ring networks, where processors communicate by exchanging messages. A common quality of all presented algorithm is that they are transparent to the upper layer application (that uses the token passing algorithm to perform *e.g.* critical sections of code). First, we provided tight complexity results about previously known self-stabilizing algorithm ([8, 1, 4]). Then, we provided original algorithms using the notion of a speed reducer. One algorithm, **SpeedReducer1**, is optimal in stabilization time and service time, but require $O(log(n))$ bits of memory per processor. The last algorithm, **SpeedReducer2**, is optimal with respect to all three complexity measures of self-stabilizing token passing algorithms: stabilization time, service time, and memory.

There remains the open question of having an optimal self-stabilizing algorithm for uniform and anonymous unidirectional rings of any size that is optimal in stabilization time, service time, and memory, but also that has a bounded service time on all possible executions.

Bibliography

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

[2] J. Beauquier, M. Gradinariu, and C. Johnen. Memory space requirements for self-stabilizing leader election protocols. In *PODC99 18th Annual ACM Symposium on Principles of Distributed Computing*, pages 199–208, 1999.

[3] J. E. Burns and J. Pachl. Uniform self-stabilizing rings. *ACM Transactions on Programming Languages and Systems*, 11(2):330–344, 1989.

[4] A. K. Datta, M. Gradinariu, and S. Tixeuil. Self-stabilizing mutual exclusion using unfair distributed scheduler. In *IPDPS00 14th International Parallel and Distributed Processing Symposium*, pages 465–470, 2000.

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

[6] S. Dolev. *Self-stabilization*. The MIT Press, 2000.

[7] M. Duflot, L. Fribourg, and C. Picaronny. Finite-state distributed algorithms as markov chains. In *DISC01 Distributed Computing 15th International Symposium, Springer LNCS:2180*, pages 240–255, 2001.

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

[9] C. Johnen. Service time optimal self-stabilizing token circulation protocol on anonymous unidirectional rings. In *SRDS02 21th Symposium on Reliable Distributed Systems*, pages 80–89. IEEE Computer Society Press, 2002.

[10] 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.

[11] S. Katz and K. J. Perry. Self-stabilizing extensions for message-passing systems. *Distributed Computing*, 7(1):17–26, 1993.

[12] J.R. Norris. *Markov Chains*. Cambridge University Press, 1997.

[13] L. Rosaz. Self-stabilizing token circulation on asynchronous uniform unidirectional rings. In *PODC00 19th Annual ACM Symposium on Principles of Distributed Computing*, pages 249–258, 2000.