Introducing Speculation in Self-Stabilization - An Application to Mutual Exclusion

Swan Dubois, Rachid Guerraoui

Introducing Speculation in Self-Stabilization
An Application to Mutual Exclusion

Swan Dubois*
LPD, EPFL, Switzerland

Rachid Guerraoui†
LPD, EPFL, Switzerland

Abstract

Self-stabilization ensures that, after any transient fault, the system recovers in a finite time and eventually exhibits. Speculation consists in guaranteeing that the system satisfies its requirements for any execution but exhibits significantly better performances for a subset of executions that are more probable. A speculative protocol is in this sense supposed to be both robust and efficient in practice.

We introduce the notion of speculative stabilization which we illustrate through the mutual exclusion problem. We then present a novel speculatively stabilizing mutual exclusion protocol. Our protocol is self-stabilizing for any asynchronous execution. We prove that its stabilization time for synchronous executions is $\lceil diam(g)/2 \rceil$ steps (where $diam(g)$ denotes the diameter of the system).

This complexity result is of independent interest. The celebrated mutual exclusion protocol of Dijkstra stabilizes in n steps (where n is the number of processes) in synchronous executions and the question whether the stabilization time could be strictly smaller than the diameter has been open since then (almost 40 years). We show that this is indeed possible for any underlying topology. We also provide a lower bound proof that shows that our new stabilization time of $\lceil diam(g)/2 \rceil$ steps is optimal for synchronous executions, even if asynchronous stabilization is not required.

Keywords: Fault-tolerance; Speculation; Self-stabilization; Mutual exclusion.

1 Introduction

The speculative approach to distributed computing [21, 23, 18, 13, 14] lies on the inherent trade-of between robustness and efficiency. Indeed, we typically require distributed applications to be safe and live under various hostile conditions such as asynchronism, faults, attacks, and contention. This typically leads to high consumption of system resources, *e.g.* time of computation, which is due to the need to perform synchronizations, redundancies or checking.

The speculative approach assumes that, even if degraded conditions are indeed possible, they are less probable than friendly conditions (for example, synchronous executions without faults). The underlying idea is to simultaneously ensure that the protocol is correct whatever the execution is (even in degraded conditions) but to optimize it for a subset of executions that are the most probable in practice. Even if this idea was applied in various contexts, it has never been applied to distributed systems tolerant to transient faults, *i.e.* self-stabilizing systems [8]. In fact, it was

*swan.dubois@epfl.ch
†rachid.guerraoui@epfl.ch

not clear whether self-stabilization and speculation could be even combined because of the specific nature of transient faults, for they could corrupt the state of the entire system. The objective of this paper is to explore this avenue.

Self-stabilization was introduced by Dijkstra [8]. Intuitively, a self-stabilizing system ensures that, after the end of any transient fault, the system reaches in a finite time, without any external help, a correct behavior. In other words, a self-stabilizing system repairs itself from any catastrophic state. Since the seminal work of Dijkstra, self-stabilizing protocols were largely studied (see *e.g.* [9, 24, 16]). The main objective has been to design self-stabilizing systems tolerating asynchronism while reducing the stabilization time, *i.e.*, the worst time needed by the protocol to recover a correct behavior over all executions of the system.

Our contribution is twofold. First, we define a new variation of self-stabilization in which the main measure of complexity, the stabilization time, is regarded as a function of the adversary and not as a single value. Indeed, we associate to each adversary (known as a *scheduler* or *daemon* in self-stabilization) the worst stabilization time of the protocol over the set of executions captured by this adversary. Then, we define a speculatively stabilizing protocol as a protocol that self-stabilizes under a given adversary but that exhibits a significantly better stabilization time under another (and weaker) adversary. In this way, we ensure that the protocol stabilizes in a large set of executions but guarantees efficiency only on a smaller set (the one we speculate more probable in practice). For the sake of simplicity, we present our notion of speculative stabilization for two adversaries. It could be easily extended to an arbitrary number of adversaries.

Although the idea of optimizing the stabilization time for some subclass of executions is new, some self-stabilizing protocols satisfy (somehow by accident) our definition of speculative stabilization. For example, the Dijkstra's mutual exclusion protocol stabilization time falls to n steps (the number of processes) in synchronous executions. The question whether one could do better has been open since then, *i.e.* during almost 40 years. We close the question in this paper through the second contribution of this paper.

Indeed, we present a novel speculatively stabilizing mutual exclusion protocol. We prove that its stabilization time for synchronous executions is $\lceil diam(g)/2 \rceil$ steps (where $diam(g)$ denotes the diameter of the system), which significantly improves the bound of Dijkstra's protocol. We prove that we cannot improve it. Indeed, we present a lower bound result on the stabilization time of mutual exclusion for synchronous executions. This result is of independent interest since it remains true beyond the scope of speculation and holds even for a protocol that does not need to stabilize in asynchronous executions.

Designing our protocol went through addressing two technical challenges. First, we require the stabilization of a global property (the uniqueness of critical section) in a time strictly smaller than the diameter of the system, which is counter-intuitive (even for synchronous executions). Second, the optimization of the stabilization time for synchronous executions must not prevent the stabilization for asynchronous ones.

The key to addressing both challenges was a "reduction" to clock synchronization: more specifically, leveraging the self-stabilizing asynchronous unison protocol of [2] within mutual exclusion. We show that it is sufficient to choose correctly the clock size and to grant the access to critical section upon some clock values to ensure (i) the self-stabilization of the protocol for any asynchronous execution as well as (ii) the optimality of its stabilization time for synchronous ones. This reduction was also, we believe, the key to the genericity of our protocol. Unlike Dijkstra's protocol which assumes an underlying ring shaped communication structure, our protocol runs over

any communication structure.

We could derive our lower bound result for synchronous executions based on the observation that a process can gather information at most at distance d in d steps whatever protocol it executes. Hence, in the worst case, it is impossible to prevent two processes from simultaneously entering a critical section during the first $\lceil diam(g)/2 \rceil$ steps of all executions with a deterministic protocol.

The rest of this paper is organized as follows. Section 2 introduces the model and the definitions used through the paper. Section 3 presents our notion of speculative stabilization. Section 4 presents our mutual exclusion protocol. Section 5 provides our lower bound result. Section 6 ends the paper with some perspectives.

2 Model, Definitions, and Notations

We consider the classical model of distributed systems introduced by Dijkstra [8]. Processes communicate by atomic reading of neighbors' states and the (asynchronous) adversary of the system is captured by an abstraction called *daemon*.

Distributed protocol. The distributed system consists of a set of processes that form a communication graph. The processes are vertices in this graph and the set of those vertices is denoted by V. The edges of this graph are pairs of processes that can communicate with each other. Such pairs are neighbors and the set of edges is denoted by E ($E \subseteq V^2$). Hence, $g = (V, E)$ is the communication graph of the distributed system. Each vertex of g has a set of variables, each of them ranges over a fixed domain of values. A state $\gamma(v)$ of a vertex v is the vector of values of all variables of v at a given time. An assignment of values to all variables of the graph is a configuration. The set of configurations of g is denoted by Γ. An action α of g transitions the graph from one configuration to another. The set of actions of g is denoted by A ($A = \{(\gamma, \gamma') | \gamma \in \Gamma, \gamma' \in \Gamma, \gamma \neq \gamma'\}$). A *distributed protocol* π on g is defined as a subset of A that gathers all actions of g allowed by π. The set of distributed protocols on g is denoted by Π ($\Pi = P(A)$ where, for any set S, $P(S)$ denotes the powerset of S).

Execution. Given a graph g, a distributed protocol π on g, an *execution* σ of π on g, starting from a given configuration γ_0, is a maximal sequence of actions of π of the following form $\sigma = (\gamma_0, \gamma_1)(\gamma_1, \gamma_2)(\gamma_2, \gamma_3)\ldots$. An execution is *maximal* if it is either infinite or finite but its last configuration is terminal (that is, there exists no actions of π starting from this configuration). The set of all executions of π on g, starting from all configurations of Γ, is denoted by Σ_π.

Adversary (daemon). Intuitively, a daemon is a restriction on the executions of distributed protocols to be considered possible. For a distributed protocol π, at each configuration γ, a subset of vertices are *enabled*, that is there exists an action of π that modifies their state (formally, $\exists \gamma' \in \Gamma, (\gamma, \gamma') \in \pi, \gamma(v) \neq \gamma'(v)$). The daemon then chooses one of the possible action of π starting from γ (and hence, selects a subset of enabled vertices that are allowed to modify their state during this action). A formal definition follows.

Definition 1 (Daemon)**.** *Given a graph g, a daemon d on g is a function that associates to each distributed protocol π on g a subset of executions of π, that is $d : \pi \in \Pi \longmapsto d(\pi) \in P(\Sigma_\pi)$.*

Given a graph g, a daemon d on g and a distributed protocol π on g, an execution σ of π ($\sigma \in \Sigma_\pi$) is *allowed* by d if and only if $\sigma \in d(\pi)$. Also, given a graph g, a daemon d on g and a distributed protocol π on g, we say that π *runs* on g under d if we consider that the only possible executions of π on g are those allowed by d.

Some classical examples of daemons follow. The unfair distributed daemon [19] (denoted by ud) is the less constrained one because we made no assumption on its choices (any execution of the distributed protocol is allowed). The synchronous daemon [15] (denoted by sd) is the one that selects all enabled vertices in each configuration. The central daemon [8] (denoted by cd) selects only one enabled vertex in each configuration.

This way of viewing daemons as a set of possible executions (for a particular graph g) drives a natural partial order over the set of daemons. For a particular graph g, a daemon d is more powerful than another daemon d' if all executions allowed by d' are also allowed by d. Overall, d has more scheduling choices than d'. A more precise definition follows.

Definition 2 (Partial order over daemons)**.** *For a given graph g, we define the following partial order $\preccurlyeq$ on $\mathcal{D}$: $\forall (d, d') \in \mathcal{D}, d \preccurlyeq d' \Leftrightarrow (\forall \pi \in \Pi, d(\pi) \subseteq d'(\pi))$. If two daemons d and d' satisfy $d \preccurlyeq d'$, we say that d' is more powerful than d.*

For example, the unfair distributed daemon is more powerful than any daemon (in particular the synchronous one). Note that some daemons (for example the synchronous and the central ones) are not comparable. For a more detailed discussion about daemons, the reader is referred to [10].

Further notations. Given a graph g and a distributed protocol π on g, we introduce the following set of notations. First, n denotes the number of vertices of the graph whereas m denotes the number of edges ($n = |V|$ and $m = |E|$). The set of neighbors of a vertex v is denoted by $neig(v)$. The distance between two vertices u and v (that is, the length of a shortest path between u and v in g) is denoted by $dist(g, u, v)$. The diameter of g (that is, the maximal distance between two vertices of g) is denoted by $diam(g)$. For any execution $e = (\gamma_0, \gamma_1)(\gamma_1, \gamma_2)\ldots$, we denote by e_i the prefix of e of length i (that is $e_i = (\gamma_0, \gamma_1)(\gamma_1, \gamma_2)\ldots(\gamma_{i-1}, \gamma_i)$).

Guarded representation of distributed protocols. For the sake of clarity, we do not describe distributed protocols by enumerating all their actions. Instead, we represent distributed protocols using a local description of actions borrowed from [8]. Each vertex has a local protocol consisting of a set of guarded rules of the following form: $< label > \;\; :: \;\; < guard > \;\; \longrightarrow \;\; < action >$. $< label >$ is a name to refer to the rule in the text. $< guard >$ is a predicate that involves variables of the vertex and of its neighbors. This predicate is true if and only if the vertex is enabled in the current configuration. We say that a rule is enabled in a configuration when its guard is evaluated to true in this configuration. $< action >$ is a set of instructions modifying the state of the vertex. This set of instructions must describe the changes of the vertex state if this latter is activated by the daemon.

Self-stabilization. Intuitively, to be self-stabilizing [8], a distributed protocol must satisfy the two following properties: (i) *closure*, that is there exists some configuration from which any execution of the distributed protocol satisfies the specification; and (ii) *convergence*, that is starting from any arbitrary configuration, any execution of the distributed protocol reaches in a finite time a configuration that satisfies the closure property.

Self-stabilization induces fault-tolerance since the initial configuration of the system may be arbitrary because of a burst of transient faults. Then, a self-stabilizing distributed protocol ensures that after a finite time (called the convergence or stabilization time), the distributed protocol recovers on his own a correct behavior (by convergence property) and keeps this correct behavior until there is no faults (by closure property).

Definition 3 (Self-stabilization [8]). *A distributed protocol π is self-stabilizing for specification spec under a daemon d if starting from any arbitrary configuration every execution of $d(\pi)$ contains a configuration from which every execution of $d(\pi)$ satisfies spec.*

For any self-stabilizing distributed protocol π under a daemon d for a specification *spec*, its convergence (or stabilization) time (denoted by $conv\_time(\pi, d)$) is the worst stabilization time (that is, the number of actions required to reach a configuration from which any execution satisfies *spec*) of executions of π allowed by d. Note that, for any self-stabilizing distributed protocol π under a daemon d, π is self-stabilizing under any daemon d' such that $d' \preccurlyeq d$ and $conv\_time(\pi, d') \leq conv\_time(\pi, d)$.

3 Speculative Stabilization

Intuitively, a speculative protocol ensures the correctness in a large set of executions but is optimized for some scenarios that are speculated to be more frequent (maybe at the price of worst performance in less frequent cases).

Regarding self-stabilization, the most common measure of complexity is the stabilization time. Accordingly, we choose to define a speculatively stabilizing protocol as a self-stabilizing protocol under a given daemon that exhibits a significantly better stabilization time under a weaker daemon (the latter gathers scenarios that are speculated to be more frequent). We can now define our notion of speculative stabilization.

Definition 4 (Speculative Stabilization). *For two daemons d and d' satisfying $d' \prec d$, a distributed protocol π is (d, d', f)-speculatively stabilizing for specification spec if: (i) π is self-stabilizing for spec under d; and (ii) f is a function on g satisfying $\frac{conv\_time(\pi,d)}{conv\_time(\pi,d')} \in \Omega(f)$.*

We restrict ourselves for two daemons here for the sake of clarity. We can easily extend this definition to an arbitrary number of daemons (as long as they are comparable). For instance, we can say that a distributed protocol π is (d, d_1, d_2, f_1, f_2)-speculatively stabilizing (with $d_1 \prec d$ and $d_2 \prec d$) if it is both (d, d_1, f_1)-speculatively stabilizing and (d, d_2, f_2)-speculatively stabilizing.

Still for the sake of simplicity, we say in the following that a distributed protocol π is d-speculatively stabilizing for specification *spec* if there exists a daemon $d \neq ud$ such that π is (ud, d, f)-speculatively stabilizing for specification *spec* with $f > 1$. In other words, a d-speculatively stabilizing distributed protocol is self-stabilizing under the unfair distributed daemon (and hence always guarantees convergence) but is optimized for a given subclass of executions described by d.

Examples. Although the idea of speculation approaches in self-stabilization has not been yet precisely defined, there exists some examples of self-stabilizing distributed protocols in the literature that turn out to be speculative. We survey some of them in the following.

The seminal work of Dijkstra [8] introduced self-stabilization in the context of mutual exclusion. His celebrated protocol operates only on rings. It is in fact $(ud, sd, g \mapsto n)$-speculatively stabilizing

since it stabilizes upon $\Theta(n^2)$ steps under the unfair distributed daemon and it is easy to see that it needs only n steps to stabilize under the synchronous daemon. The well-known $min+1$ protocol of [17] is $(ud, sd, g \mapsto n^2/diam(g))$-speculatively stabilizing for BFS spanning tree construction. Its stabilization time is in $\Theta(n^2)$ steps under the unfair distributed daemon while it is in $\Theta(diam(g))$ steps under the synchronous daemon. Another example is the self-stabilizing maximal matching protocol of [22]. This protocol is $(ud, sd, g \mapsto m/n)$-speculatively stabilizing: its stabilization time is 4n+2m (respectively 2n+1) steps under the unfair distributed (respectively synchronous) daemon.

4 A new Mutual Exclusion Protocol

Mutual exclusion was classically adopted as a benchmark in self-stabilization under various settings [8, 20, 11, 5, 1]. Intuitively, it consists in ensuring that each vertex can enter infinitely often in critical section and there is never two vertices simultaneously in the critical section. Using such a distributed protocol, vertices can for example access shared resources without conflict.

Our contribution in this context is a novel self-stabilizing distributed protocol for mutual exclusion under the unfair distributed daemon that moreover exhibits optimal convergence time under the synchronous daemon. Contrary to the Dijkstra's protocol, our protocol supports any underlying communication structure (we do not assume that the communication graph is reduced to a ring). Thanks to speculation, our protocol is ideal for environment in which we speculate that most of the executions are synchronous.

We adopt the following specification of mutual exclusion. For each vertex v, we define a predicate $privileged_v$ (over variables of v and possibly of its neighbors). We say that a vertex v is privileged in a configuration γ if and only if $privileged_v = true$ in γ. If a vertex v is privileged in a configuration γ and v is activated during an action (γ, γ'), then v executes its critical section during this action. We can now specify the mutual exclusion problem as follows.

Specification 1 (Mutual exclusion $spec_{ME}$)**.** *An execution e satisfies $spec_{ME}$ if at most one vertex is privileged in any configuration of e (safety) and any vertex infinitely often executes its critical section in e (liveness).*

The rest of this section is organized as follows. Section 4.1 overviews our protocol. Section 4.2 proves the correctness of our protocol under the unfair distributed daemon. Section 4.3 analyzes its stabilization time under the synchronous and the unfair distributed daemon.

4.1 Speculatively Stabilizing Mutual Exclusion

As we restrict ourselves to deterministic protocols, we know by [4] that, to ensure mutual exclusion, we must assume a system with identities (that is, each vertex has a distinct identifier). Indeed, we know by [4] that the problem does not admit deterministic solution on uniform (*i.e.* without identifiers) rings of composite size. Without loss of generality, we assume that the set of identities (denoted by ID) is equals to $\{0, 1, \ldots, n-1\}$ (if this assumption is not satisfied, it is easy to define a mapping of identities satisfying it).

Our protocol is based upon an existing self-stabilizing distributed protocol for the asynchronous unison problem [12, 6]. This problem consists in ensuring, under the unfair distributed daemon, some synchronization guarantees on vertices' clocks. More precisely, each vertex has a register r_v that contains a clock value. A clock is a bounded set enhanced with an incrementation function.

Intuitively, an asynchronous unison protocol ensures that the difference between neighbors' registers is bounded and that each register is infinitely often incremented.

In the following, we give the definition of this problem and the solution proposed in [2] from which we derive our mutual exclusion protocol.

Clock. A bounded clock $\mathcal{X} = (C, \phi)$ is a bounded set $C = cherry(\alpha, K)$ (parametrized with two integers $\alpha \geq 1$ and $K \geq 2$) enhanced with an incrementation function ϕ defined as follows.

Let c be any integer. Denote by $\overline{c}$ the unique element in $[0, \ldots, K-1]$ such that $c = \overline{c} \bmod K$. We define the distance $d_K(c, c') = min\{\overline{c - c'}, \overline{c' - c}\}$ on $[0, \ldots, K-1]$. Two integers c and c' are said to be locally comparable if and only if $d_K(a, b) \leq 1$. We then define the local order relation $\leq_l$ as follows: $c \leq_l c'$ if and only if $0 \leq \overline{c' - c} \leq 1$. Let us define $cherry(\alpha, K) = \{-\alpha, \ldots, 0, \ldots, K-1\}$. Let ϕ be the function defined by:

$$\phi : c \in cherry(\alpha, K) \mapsto \begin{cases} (c+1) & \text{if } c < 0 \\ (c+1) \bmod K & \text{otherwise} \end{cases}$$

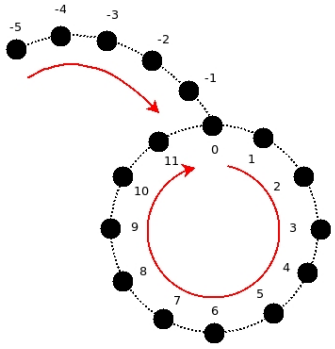

Figure 1: A bounded clock $\mathcal{X} = (cherry(\alpha, K), \phi)$ with $\alpha = 5$ and $K = 12$.

The pair $\mathcal{X} = (cherry(\alpha, K), \phi)$ is called a bounded clock of initial value α and of size K (see Figure 1). We say that a clock value $c \in cherry(\alpha, K)$ is incremented when this value is replaced by $\phi(c)$. A reset on $\mathcal{X}$ consists of an operation replacing any value of $cherry(\alpha, K) \setminus \{-\alpha\}$ by $-\alpha$. Let $init_{\mathcal{X}} = \{-\alpha, \ldots, 0\}$ and $stab_{\mathcal{X}} = \{0, \ldots, K-1\}$ be the set of initial values and correct values respectively. Let us denote $init^*_{\mathcal{X}} = init_{\mathcal{X}} \setminus \{0\}$, $stab^*_{\mathcal{X}} = stab_{\mathcal{X}} \setminus \{0\}$, and $\leq_{init}$ the usual total order on $init_{\mathcal{X}}$.

Asynchronous unison. Given a distributed system in which each vertex v has a register r_v taken a value of a bounded clock $\mathcal{X} = (C, \phi)$ with $C = cherry(\alpha, K)$, we define a legitimate configuration for asynchronous unison as a configuration satisfying: $\forall v \in V, \forall u \in neig(v), (r_v \in stab_{\mathcal{X}}) \wedge (r_u \in stab_{\mathcal{X}}) \wedge (d_K(r_v, r_u) \leq 1)$. In other words, a legitimate configuration is a configuration in which each clock value is a correct one and the drift between neighbors' registers is bounded by 1. We denote by Γ_1 the set of legitimate configurations for asynchronous unison. Note that we have, for any configuration of Γ_1 and any pair of vertices, (u, v), $d_K(r_u, r_v) \leq diam(g)$ by definition. We can now specify the problem.

Specification 2 (Asynchronous unison $spec_{AU}$)**.** *An execution e satisfies $spec_{AU}$ if every configuration of e belongs to Γ_1 (safety) and the clock value of each vertex is infinitely often incremented in e (liveness).*

In [2], the authors propose a self-stabilizing asynchronous unison distributed protocol in any anonymous distributed system under the unfair distributed daemon. The main idea of this protocol is to reset the clock value of each vertex that detects any local safety violation (that is, whenever some neighbor that has a not locally comparable clock value). Otherwise, a vertex is allowed to increment its clock (of initial or of correct value) only if this latter has locally the smallest value.

Algorithm 1 $\mathcal{SSME}$: Mutual exclusion protocol for vertex v.

Constants:

$id_v \in ID$	:	identity of v
$n \in \mathbb{N}$	:	number of vertices of the communication graph
$diam(g) \in \mathbb{N}$	:	diameter of the communication graph
$\mathcal{X} = (cherry(n, (2.n-1)(diam(g)+1)+2), \phi)$	:	clock of v

Variable:

$r_v \in \mathcal{X}$: register of v

Predicates:

$$
\begin{array}{lcl}
privileged_v & \equiv & (r_v = 2.n + 2.diam(g).id_v) \\
correct_v(u) & \equiv & (r_v \in stab_{\mathcal{X}}) \wedge (r_u \in stab_{\mathcal{X}}) \wedge (d_K(r_v, r_u) \leq 1) \\
allCorrect_v & \equiv & \forall u \in neig(v), correct_v(u) \\
normalStep_v & \equiv & allCorrect_v \wedge (\forall u \in neig(v), r_v \leq_l r_u) \\
convergeStep_v & \equiv & r_v \in init^*_{\mathcal{X}} \wedge \forall u \in neig(v), (r_u \in init_{\mathcal{X}} \wedge r_v \leq_{init} r_u) \\
resetInit_v & \equiv & \neg allCorrect_v \wedge (r_v \notin init_{\mathcal{X}})
\end{array}
$$

Rules:

$$
\begin{array}{lcll}
NA & :: & normalStep_v & \longrightarrow \quad r_v := \phi(r_v) \\
CA & :: & convergeStep_v & \longrightarrow \quad r_v := \phi(r_v) \\
RA & :: & resetInit_v & \longrightarrow \quad r_v := -n
\end{array}
$$

The choice of parameters α and K are crucial. In particular, to make the protocol self-stabilizing for any anonymous communication graph g under the unfair distributed daemon, the parameters must satisfy $\alpha \geq hole(g) - 2$ and $K > cyclo(g)$, where $hole(g)$ and $cyclo(g)$ are two constants related to the topology of g. Namely, $hole(g)$ is the length of a longest hole in g (*i.e.* the longest chordless cycle), if g contains a cycle, 2 otherwise. $cyclo(g)$ is the cyclomatic characteristic of g (*i.e.* the length of the maximal cycle of the shortest maximal cycle basis of g), if g contains a cycle, 2 otherwise. Actually, [2] shows that taking $\alpha \geq hole(g) - 2$ ensures that the protocol recovers in finite time a configuration in Γ_1. Then, taking $K > cyclo(g)$ ensures that each vertex increments its local clock infinitely often. Note that, by definition, $hole(g)$ and $cyclo(g)$ are bounded by n.

The mutual exclusion protocol. The main idea behind our protocol is to execute the asynchronous unison of [2], presented earlier, with a particular bounded clock and then to grant the privilege to a vertex only when its clock reaches some value. The clock size must be sufficiently large to ensure that at most one vertex is privileged in any configuration of Γ_1. If the definition of the predicate *privileged* guarantees this property, then the correctness of our mutual exclusion protocol follows from the one of the underlying asynchronous unison.

More specifically, we choose a bounded clock $\mathcal{X} = (cherry(\alpha, K), \phi)$ with $\alpha = n$ and $K = (2.n-1)(diam(g)+1)+2$ and we define $privileged_v \equiv (r_v = 2.n+2.diam(g).id_v)$. In particular, note that we have : $privileged_{v_0} \equiv (r_{v_0} = 2.n)$ and $privileged_{v_{n-1}} \equiv (r_{v_{n-1}} = (2.n-2)(diam(g)+1)+2)$.

Our distributed protocol, called $\mathcal{SSME}$ (for $\mathcal{S}$peculatively $\mathcal{S}$tabilizing $\mathcal{M}$utual $\mathcal{E}$xclusion), is described in Algorithm 1. Note that this protocol is identical to the one of [2] except for the size of the clock and the definition of the predicate *privileged* (that does not interfere with the protocol).

We prove in the following that this protocol is self-stabilizing for $spec_{ME}$ under the unfair distributed daemon and exhibits the optimal convergence time under the synchronous one. In other words, we will prove that this protocol is *sd*-speculatively stabilizing for $spec_{ME}$.

4.2 Correctness

We prove here the self-stabilization of $\mathcal{SSME}$ under the unfair distributed daemon.

Theorem 1. *$\mathcal{SSME}$ is a self-stabilizing distributed protocol for $spec_{ME}$ under ufd.*

Proof. As we choose $\alpha = n \geq hole(g) - 2$ and $K = (2.n - 1)(diam(g) + 1) + 2 > n \geq cyclo(g)$, the main result of [2] allows us to deduce that $\mathcal{SSME}$ is a self-stabilizing distributed protocol for $spec_{AU}$ under ufd (recall that the predicate *privileged* does not interfere with the protocol). By definition, this implies that there exists, for any execution e of $\mathcal{SSME}$ under ufd, a suffix e' reached in a finite time that satisfies $spec_{AU}$.

Let γ be a configuration of e' such a vertex v is privileged in γ. Then, by definition, we have $r_v = 2.n + 2.diam(g).id_v$. As γ belongs to e', we can deduce that $\gamma \in \Gamma_1$. Hence, for any vertex $u \in V \setminus \{v\}$, we have $d_K(r_u, r_v) \leq diam(g)$. Then, by definition of the predicate *prvileged*, no other vertex than v can be privileged in γ. We can deduce that the safety of $spec_{ME}$ is satisfied on e'. The liveness of $spec_{ME}$ on e' follows from the one of $spec_{AU}$ and from the definition of the predicate *privileged*.

Hence, for any execution of $\mathcal{SSME}$ under ufd, there exists a suffix reached in a finite time that satisfies $spec_{ME}$, that proves the theorem. □

4.3 Time Complexities

This section analyses the time complexity of our self-stabilizing mutual exclusion protocol. In particular, we provide an upper bound of its stabilization time under the synchronous daemon (see Theorem 2) and under the unfair distributed daemon (see Theorem 3).

Synchronous daemon. We first focus on the stabilization time of $\mathcal{SSME}$ under the synchronous daemon. We need to introduce some notations and definitions.

From now, $e = (\gamma_0, \gamma_1)(\gamma_1, \gamma_2)\ldots$ denotes a synchronous execution of $\mathcal{SSME}$ starting from an arbitrary configuration γ_0. For a configuration γ_i and a vertex v, r_v^i denotes the value of r_v in γ_i.

Definition 5 (Island)**.** *In a configuration γ_i, an island I is a maximal (w.r.t. inclusion) set of vertices such that $I \subsetneq V$ and $\forall (u, v) \in I, u \in neig(v) \Rightarrow correct_v(u)$. A zero-island is an island such that $\exists v \in I, r_v^i = 0$. A non-zero-island is an island such that $\forall v \in I, r_v^i \neq 0$.*

Note that any vertex v that satisfies $r_v \in stab_{\mathcal{X}}$ in a configuration $\gamma \notin \Gamma_1$ belongs by definition to an island (either a zero-island or a non-zero-island) in γ.

Definition 6 (Border and depth of an island)**.** *In a configuration γ_i that contains an island $I \neq \emptyset$, the border of I (denoted by $border(I)$) is defined by $border(I) = \{v \in I | \exists u \in V \setminus I, u \in neig(v)\}$ and the depth of I (denoted by $depth(I)$) is defined by $depth(I) = max\{min\{dist(g, v, u) | u \in border(I)\} | v \in I\}$.*

Then, we have to prove a set of preliminaries lemmas before stating our main theorem.

Lemma 1. *If a vertex v is privileged in a configuration γ_i (with $0 \leq i < diam(g)$), then v cannot execute rules CA and RA in e_i.*

Proof. As the result is obvious for $i = 0$, let γ_i (with $0 < i < diam(g)$) be a configuration such that a vertex v is privileged in γ_i. Then, we have by definition that $r_v^i = 2.n + 2.diam(g).id_v$.

By contradiction, assume that v executes at least once rule CA or RA in e_i. Let j be the biggest integer such that v executes rule CA or RA during action (γ_j, γ_{j+1}) with $j < i$.

Assume that v executes rule RA during (γ_j, γ_{j+1}). Then, we have $r_v^{j+1} = -n$. From this point, only rule CA may be enabled at v but v does not execute it by construction of j. Then, we can deduce that $r_v^i = -n$ that is contradictory.

Hence, we know that v executes rule CA during (γ_j, γ_{j+1}). Consequently, we have $r_v^{j+1} \in init_{\mathcal{X}}$ by construction of the rule. As v can only execute rule NA between γ_{j+1} and γ_i by construction of j, we can deduce that $r_v^i \in init_{\mathcal{X}} \cup \{0, \ldots, 0 + i - (j + 1)\}$. As $0 + i - (j + 1) < diam(g)$, this contradiction proves the result. □

Lemma 2. *If a vertex v is privileged in a configuration γ_i (with $0 \leq i < diam(g)$), then v cannot belong to a zero-island in any configuration of e_i.*

Proof. Let γ_i (with $0 \leq i < diam(g)$) be a configuration such that a vertex v is privileged in γ_i. Then, we have by definition that $r_v^i = 2.n + 2.diam(g).id_v$.

By contradiction, assume that there exists some configurations of e_i such that v belongs to a zero-island. Let j be the biggest integer such that v belongs to a zero-island I in γ_j with $j \leq i$.

By definition of a zero-island, we know that there exists a vertex u in I such that $r_u^j = 0$. As $dist(g, u, v) \leq diam(g)$ and u and v belongs to the same island in γ_j, we have $d_K(r_u^j, r_v^j) \leq diam(g)$. By construction of the clock, we have so $r_v^j \in \{(2.n - 2)(diam(g) + 1) + 3, \ldots, 0, \ldots, diam(g)\}$.

By Lemma 1, we know that v may execute only rule NA between γ_j and γ_j. Then, we have $r_v^i \in \{(2.n - 2)(diam(g) + 1) + 3, \ldots, 0, \ldots, diam(g) + (i - j)\}$. As $diam(g) + (i - j) < 2.diam(g)$, v cannot be privileged in γ_i (whatever is its identity). This contradiction proves the result. □

Lemma 3. *If a vertex v belongs to a non-zero-island of depth $k \geq 0$ in a configuration γ_i (with $0 < i < diam(g)$), then v belongs either to a non-zero-island of depth greater or equals to $k + 1$ or to a zero-island in γ_{i-1}.*

Proof. Let γ_i (with $0 < i < diam(g)$) be a configuration such that a vertex v belongs to a non-zero-island I of depth $k \geq 0$ in γ_i.

Assume that v does not belongs to any island in γ_{i-1}. In other words, we have $r_v^{i-1} \in init_{\mathcal{X}}^*$. Consequently, v may only execute rule CA during action (γ_{i-1}, γ_i) and we have $r_v^i \in init_{\mathcal{X}}$. This means that v either belongs to a zero-island or does not belong to any island in γ_i. This contradiction shows us that v belongs to an island in γ_{i-1}.

If v belongs to a zero-island in γ_{i-1}, we have the result. Otherwise, assume by contradiction that v belongs to a non-zero island I' such that $depth(I') \leq k$ in γ_{i-1}. By definition of a non-zero-island, all vertices of $border(I')$ are enabled by rule RA in γ_{i-1}. As we consider a synchronous execution, we obtain that I (the non-zero-island that contains v in γ_i) satisfies $depth(I) < k$. This contradiction shows the lemma. □

Lemma 4. *If $\gamma_0 \notin \Gamma_1$, then any vertex v satisfies $r_v^{diam(g)} \in init_{\mathcal{X}} \cup \{(2.n - 2)(diam(g) + 1) + 3, \ldots, 0, \ldots, 2.diam(g) - 1\}$.*

Proof. Assume that $\gamma_0 \notin \Gamma_1$. Then, by definition of Γ_1 and by the construction of the protocol, we know that there exists a set $\emptyset \neq V' \subseteq V$ such that vertices of V' are enabled by rule RA in γ_0. Let v be an arbitrary vertex of V.

If v executes at least once the rule RA during $e_{diam(g)}$, let i be the biggest integer such that v executes rule RA during (γ_i, γ_{i+1}) with $i < diam(g)$. Then, we have $r_v^{i+1} = -n$. As $diam(g) - (i + 1) < n$, we can deduce that v may execute only rule CA between γ_i and $\gamma_{diam(g)}$. Consequently, we have $r_v^{diam(g)} \in init_{\mathcal{X}}$.

If v executes at least once the rule CA but never executes rule RA during $e_{diam(g)}$, let i be the biggest integer such that v executes rule CA during (γ_i, γ_{i+1}) with $i < diam(g)$. Then, we have $r_v^{i+1} \in init_{\mathcal{X}}$. By construction of i, we can deduce that v may execute only rule NA between γ_i and $\gamma_{diam(g)}$. As $diam(g) - (i + 1) < diam(g)$, we have $r_v^{diam(g)} \in init_{\mathcal{X}} \cup \{0, \ldots, diam(g) - 1\}$.

Otherwise (v executes only rule NA during $e_{diam(g)}$), let i be the integer defined by $i = min\{dist(g, v, v')|v' \in V'\}$. Note that $0 < i \leq diam(g)$ by construction (recall that $v \notin V'$). We can deduce that v belongs to a zero-island in γ_i (otherwise, v executes rule RA or CA during (γ_i, γ_{i+1})). By definition of a zero-island, we have then $r_v^i \in \{(2.n - 2)(diam(g) + 1) + 3, \ldots, 0, \ldots diam(g)\}$. As v may execute only rule NA between γ_i and $\gamma_{diam(g)}$ and $diam(g) - i < diam(g)$, we can deduce that $r_v^{diam(g)} \in \{(2.n - 2)(diam(g) + 1) + 3, \ldots, 0, \ldots, 2.diam(g) - 1\}$. □

Theorem 2. $conv\_time(\mathcal{SSME}, sd) \leq \left\lceil \frac{diam(g)}{2} \right\rceil$

Proof. By contradiction, assume that $conv\_time(\mathcal{SSME}, sd) > \left\lceil \frac{diam(g)}{2} \right\rceil$. This means that there exists a configuration γ_0 such that the synchronous execution $e = (\gamma_0, \gamma_1)(\gamma_1, \gamma_2) \ldots$ of $\mathcal{SSME}$ satisfies: there exists an integer $i \geq \left\lceil \frac{diam(g)}{2} \right\rceil$ and two vertices u and v such that u and v are simultaneously privileged in γ_i. Let us study the following cases (note that they are exhaustive):

Case 1: $\left\lceil \frac{diam(g)}{2} \right\rceil \leq i < diam(g)$

By Lemma 1, we know that u may execute only rule NA in e_i. This implies that $\forall j \leq i, r_u^j \in stab_{\mathcal{X}}$ and then $d_K(r_u^i, r_u^0) \leq i$. By the same way, we can prove that $d_K(r_v^i, r_v^0) \leq i$.

If u is privileged in γ_i, this means that $r_u^i \in stab_{\mathcal{X}}$ and $d_K(r_u^i, 0) > diam(g)$. As u and v are simultaneously privileged in γ_i, we have by definition that $d_K(r_u^i, r_v^i) > diam(g)$. This implies that $\gamma_i \notin \Gamma_1$ and that u belongs to a non-zero-island I such that $depth(I) \geq 1$ in γ_i. By recursive application of Lemmas 2 and 3, we deduce that u belongs to a non-zero-island I' such that $depth(I') \geq i + 1 \geq \left\lceil \frac{diam(g)}{2} \right\rceil + 1$ in γ_0. The same property holds for v. As $dist(g, u, v) \leq diam(g)$, we can deduce that u and v belongs to the same non-zero-island in γ_0, that allows us to state $d_K(r_u^0, r_v^0) \leq diam(g)$.

Without loss of generality, assume that $id_u < id_v$. Let us now distinguish the following cases:

If $id_v - id_u \geq 2$, as u and v are simultaneously privileged in γ_i, we have $d_K(r_u^i, r_v^i) \geq 2.n + diam(g) + 1$ (if $id_u = n - 1$ and $id_v = 0$) or $d_K(r_u^i, r_v^i) \geq 4.diam(g)$ (otherwise). Note that in both cases, we have $d_K(r_u^i, r_v^i) \geq 3.diam(g)$. Recall that d_K is a distance. In particular, it must satisfy the triangular inequality. Then, we have $d_K(r_u^i, r_v^i) \leq d_K(r_u^i, r_u^0) + d_K(r_u^0, r_v^0) + d_K(r_u^0, r_v^i)$. By previous result, we obtain that $d_K(r_u^i, r_v^i) \leq diam(g) + 2.i < 3.diam(g)$, that is contradictory.

If $id_v - id_u = 1$, by construction of γ_i, we have $r_u^i = 2.n + 2.diam(g).id_u > 0$ and $r_v^i = 2.n + 2.diam(g).(id_u + 1)$. Then, we obtain $r_v^i - r_u^i = 2.diam(g)$. Hence, we have $0 < r_u^0 \leq r_u^i < r_v^0 \leq r_v^i$. Then, we can deduce from $r_v^i - r_u^i = 2.diam(g)$ and $r_u^i - r_u^o \geq 0$ that

$r_v^i - r_u^0 \geq 2.diam(g)$. On the other hand, previous results show us that $r_v^0 - r_u^0 \leq diam(g)$ and $r_v^i - r_v^0 < diam(g)$. It follows $r_v^i - r_u^0 < 2.diam(g)$, that is contradictory.

Case 2: $diam(g) \leq i < 2.n + diam(g)$

As u and v are simultaneously privileged in γ_i, we have by definition that $d_K(r_u^i, r_v^i) > diam(g)$. This implies that $\gamma_i \notin \Gamma_1$ and then $\gamma_0 \notin \Gamma_1$ (otherwise, we obtain a contradiction with the closure of $spec_{AU}$).

By Lemma 4, for any vertex w, $r_w^{diam(g)} \in init_{\mathcal{X}} \cup \{(2.n-2)(diam(g)+1)+3, \ldots, 0, \ldots, 2.diam(g)-1\}$. As w may execute at most $i - diam(g) < 2.n$ actions between $\gamma_{diam(g)}$ and γ_i, we can deduce that $r_w^i \in init_{\mathcal{X}} \cup \{(2.n-2)(diam(g)+1)+3, \ldots, 0, \ldots, 2.n+2.diam(g)-1\}$ for any vertex w.

By construction of the clock and the definition of the predicate *privileged*, we can conclude that there is at most one privileged vertex (the one with identity 0) in γ_i, that is contradictory.

Case 3: $i \geq 2.n + diam(g)$

By [3], we know that $\mathcal{SSME}$ stabilizes to $spec_{AU}$ in at most $\alpha + lcp(g) + diam(g)$ steps under the synchronous daemon where $lcp(g)$ denotes the length of the longest elementary chordless path of g. As we have $\alpha = n$ by construction and $lcp(g) \leq n$ by definition, we can deduce that $\mathcal{SSME}$ stabilizes to $spec_{AU}$ in at most $2.n + diam(g)$ steps under the synchronous daemon.

In particular, this implies that $\gamma_i \in \Gamma_1$. Then, using proof of Theorem 1, we obtain a contradiction with the fact that u and v are simultaneously privileged in γ_i.

We thus obtain that $conv\_time(\mathcal{SSME}, sd) \leq \left\lceil \frac{diam(g)}{2} \right\rceil$. □

Unfair distributed daemon. We now interested in the stabilization time of our mutual exclusion protocol under the unfair distributed daemon. Using a previous result from [7], we have the following upper bound:

Theorem 3. $conv\_time(\mathcal{SSME}, ufd) \in O(diam(g).n^3)$

Proof. Remind that the stabilization time of $\mathcal{SSME}$ for $spec_{AU}$ is an upper bound for the one for $spec_{ME}$ whatever the daemon is. The step complexity of this protocol is tricky to exactly compute. As the best of our knowledge, [7] provides the best known upper bound on this step complexity.

The main result of [7] is to prove that $\mathcal{SSME}$ stabilizes in at most $2.diam(g).n^3 + (\alpha+1).n^2 + (\alpha - 2.diam(g)).n$ steps under ufd. Since we chose $\alpha = n$, we have the result. □

5 Synchronous Lower Bound

We prove here a lower bound on the stabilization time of mutual exclusion under a synchronous daemon, showing hereby that our speculatively stabilizing protocol presented in Section 4.1 is in this sense optimal. We introduce some definitions and a lemma.

Definition 7 (Local state)**.** *Given a configuration γ, a vertex v and an integer $0 \leq k \leq diam(g)$, the k-local state of v in γ (denoted by $\gamma_{v,k}$) is the configuration of the communication subgraph $g' = (V', E')$ induced by $V' = \{v' \in V | dist(g, v, v') \leq k\}$ defined by $\forall v' \in V', \gamma_{v,k}(v') = \gamma(v')$.*

Note that $\gamma_{v,0} = \gamma(v)$ by definition.

Definition 8 (Restriction of an execution)**.** *Given an execution $e = (\gamma_0, \gamma_1)(\gamma_1, \gamma_2)\ldots$ and a vertex v, the restriction of e to v (denoted by e_v) is defined by $e_v = (\gamma_0(v), \gamma_1(v))(\gamma_1(v), \gamma_2(v))\ldots$.*

Lemma 5. *For any self-stabilizing distributed protocol π for $spec_{ME}$ under the synchronous daemon and any pair of configuration (γ, γ') such that there exists a vertex v and an integer $1 \leq k \leq diam(g)$ satisfying $\gamma_{v,k} = \gamma'_{v,k}$, the restrictions to v of the prefixes of length k of executions of π starting respectively from γ and γ' are equals.*

Proof. Let π be a self-stabilizing distributed protocol for $spec_{ME}$ under the synchronous daemon and (γ, γ') two configurations such that there exists a vertex v and an integer $1 \leq k \leq diam(g)$ satisfying $\gamma_{v,k} = \gamma'_{v,k}$. We denote by $e = (\gamma, \gamma_1)(\gamma_1, \gamma_2)\ldots$ (respectively $e' = (\gamma', \gamma'_1)(\gamma'_1, \gamma'_2)\ldots$) the synchronous execution of π starting from γ (respectively γ'). We are going to prove the lemma by induction on k.

For $k = 1$, we have $\gamma_{v,1} = \gamma'_{v,1}$, that is the state of v and of its neighbors are identical in γ and γ'. As the daemon is synchronous, we have $(e_1)_v = (e'_1)_v$, that implies the result.

For $k > 1$, assume that the lemma is true for $k-1$. The induction assumption and the synchrony of the daemon allows us to deduce that $(e_{k-1})_v = (e'_{k-1})_v$ and $\forall u \in neig(v), (e_{k-1})_u = (e'_{k-1})_u$. Hence, we have $(\gamma_{k-1})_{v,1} = (\gamma'_{k-1})_{v,1}$. Then, by the same argument than in the case $k = 1$, we deduce that $(\gamma_k)_{v,0} = (\gamma'_k)_{v,0}$, that implies the result. □

Theorem 4. *Any self-stabilizing distributed protocol π for $spec_{ME}$ satisfies $conv\_time(\pi, sd) \geq \left\lceil \frac{diam(g)}{2} \right\rceil$.*

Proof. By contradiction, assume that there exists a self-stabilizing distributed protocol π for $spec_{ME}$ such that $conv\_time(\pi, sd) < \left\lceil \frac{diam(g)}{2} \right\rceil$. For the sake of notation, let us denote $t = conv\_time(\pi, sd)$.

Given an arbitrary communication graph g, choose two vertices u and v such that $dist(g, u, v) = diam(g)$ and an arbitrary configuration γ_0. Denote by $e = (\gamma_0, \gamma_1)(\gamma_1, \gamma_2)\ldots$ the synchronous execution of π starting from γ_0.

By definition, e contains an infinite suffix in which u (respectively v) executes infinitely often its critical section. Hence, there exists a configuration γ_i (respectively γ_j) such that u (respectively v) is privileged in γ_i (respectively γ_j) and $i > t$ (respectively $j > t$).

As $t < \left\lceil \frac{diam(g)}{2} \right\rceil$ and $dist(g, u, v) = diam(g)$, there exists at least one configuration γ'_0 such that $(\gamma'_0)_{u,t} = (\gamma_{i-t})_{u,t}$ and $(\gamma'_0)_{v,t} = (\gamma_{j-t})_{v,t}$. Let $e' = (\gamma'_0, \gamma'_1)(\gamma'_1, \gamma'_2)\ldots$ be the synchronous execution of π starting from γ'_0.

By Lemma 5, we can deduce that the restriction to u of the prefix of length t of e' is the same as the one of the suffix of e starting from γ_{i-t}. In particular, u is privileged in γ'_t. By the same way, we know that v is privileged in γ'_t. This contradiction leads to the result. □

6 Conclusion

This paper studies for the first time the notion of speculation in self-stabilization. As the main measure in this context is the stabilization time, we naturally consider that a speculatively stabilizing protocol is a self-stabilizing protocol for a given adversary that exhibits moreover a better

stabilization time under another (and weaker) adversary. This weaker adversary captures a subset of most probable executions for which the protocol is optimized.

To illustrate this approach, we consider the seminal problem of Dijkstra on self-stabilization: mutual exclusion. We provide a new self-stabilizing mutual exclusion protocol. We prove then that this protocol has an optimal stabilization time in synchronous executions.

Our paper opens a new path of research in self-stabilization by considering the stabilization time of a protocol as a function of the adversary and not as a single value. As a continuation, one could naturally apply our new notion of speculative stabilization to other classical problems of distributed computing and provide speculative protocols for other adversaries than the synchronous one. It may also be interesting to study a composition tool that automatically ensures speculative stabilization.

References

[1] Joffroy Beauquier and Janna Burman. Self-stabilizing mutual exclusion and group mutual exclusion for population protocols with covering. In *OPODIS*, pages 235–250, 2011.

[2] Christian Boulinier, Franck Petit, and Vincent Villain. When graph theory helps self-stabilization. In *PODC*, pages 150–159, 2004.

[3] Christian Boulinier, Franck Petit, and Vincent Villain. Synchronous vs. asynchronous unison. *Algorithmica*, 51(1):61–80, 2008.

[4] James E. Burns and Jan K. Pachl. Uniform self-stabilizing rings. *ACM Trans. Program. Lang. Syst.*, 11(2):330–344, 1989.

[5] Viacheslav Chernoy, Mordechai Shalom, and Shmuel Zaks. A self-stabilizing algorithm with tight bounds for mutual exclusion on a ring. In *DISC*, pages 63–77, 2008.

[6] Jean-Michel Couvreur, Nissim Francez, and Mohamed G. Gouda. Asynchronous unison. In *ICDCS*, pages 486–493, 1992.

[7] Stéphane Devismes and Franck Petit. On efficiency of unison. In *TADDS*, pages 20–25, 2012.

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

[9] Shlomi Dolev. *Self-stabilization.* MIT Press, 2000.

[10] Swan Dubois and Sébastien Tixeuil. A taxonomy of daemons in self-stabilization. *CoRR*, abs/1110.0334, 2011.

[11] Philippe Duchon, Nicolas Hanusse, and Sébastien Tixeuil. Optimal randomized self-stabilizing mutual exclusion on synchronous rings. In *DISC*, pages 216–229, 2004.

[12] Mohamed G. Gouda and Ted Herman. Stabilizing unison. *Information Processing Letters*, 35(4):171–175, 1990.

[13] Rachid Guerraoui, Nikola Knezevic, Vivien Quéma, and Marko Vukolic. The next 700 bft protocols. In *EuroSys*, pages 363–376, 2010.

[14] Rachid Guerraoui, Viktor Kuncak, and Giuliano Losa. Speculative linearizability. In *PLDI*, pages 55–66, 2012.

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

[16] Ted Herman. A comprehensive bibliography on self-stabilization. http://www.cs.uiowa.edu/ftp/selfstab/bibliography/, 2002.

[17] Shing-Tsaan Huang and Nian-Shing Chen. A self-stabilizing algorithm for constructing breadth-first trees. *Information Processing Letters*, 41(2):109–117, 1992.

[18] Prasad Jayanti. Adaptive and efficient abortable mutual exclusion. In *PODC*, pages 295–304, 2003.

[19] Hirotsugu Kakugawa and Masafumi Yamashita. Uniform and self-stabilizing token rings allowing unfair daemon. *IEEE Transactions on Parallel and Distributed Systems*, 8(2):154–162, 1997.

[20] Hirotsugu Kakugawa and Masafumi Yamashita. Uniform and self-stabilizing fair mutual exclusion on unidirectional rings under unfair distributed daemon. *J. Parallel Distrib. Comput.*, 62(5):885–898, 2002.

[21] Butler W. Lampson. Lazy and speculative execution in computer systems. In *ICFP*, pages 1–2, 2008.

[22] Fredrik Manne, Morten Mjelde, Laurence Pilard, and Sébastien Tixeuil. A new self-stabilizing maximal matching algorithm. *Theoretical Computer Science*, 410(14):1336–1345, 2009.

[23] Fernando Pedone. Boosting system performance with optimistic distributed protocols. *IEEE Computer*, 34(12):80–86, 2001.

[24] Sébastien Tixeuil. *Algorithms and Theory of Computation Handbook, Second Edition*, chapter Self-stabilizing Algorithms, pages 26.1–26.45. Chapman & Hall/CRC Applied Algorithms and Data Structures. CRC Press, Taylor & Francis Group, November 2009.