Trade-off between time, space workload: the case of the self-stabilizing unisson

Stéphane Devismes, David Ilcinkas, Colette Johnen, Frédéric Mazoit

Energy efficient, Time optimal Self-stabilizing Asynchronous Unison using little memory

Stéphane Devismes

Laboratoire MIS, Université de Picardie,
33 rue Saint Leu - 80039 Amiens cedex 1, France

David Ilcinkas Colette Johnen Frédéric Mazoit

LaBRI, Université de Bordeaux, 351 cours de la Libération, F-33405 Talence cedex, France

Abstract

We presents a self-stabilizing algorithm for the unison problem which achieves an efficient trade-off between time, workload and space in a weak model.

Our algorithm is defined in the atomic-state model and works in a simplified version of the *stone age* model in which networks are anonymous and local ports are unlabelled. It makes no assumption on the daemon and thus stabilizes under the weakest one: the distributed unfair daemon. Assuming a period $B \geq 2D + 2$, our algorithm stabilizes in at most $2D - 2$ rounds and $O(\min(n^2B, n^3))$ moves, while using $\lceil \log B \rceil + 2$ bits per node where D is the network diameter and n the number of nodes. In particular and to the best of our knowledge, it is the first self-stabilizing unison for arbitrary anonymous networks achieving an asymptotically optimal stabilization time in rounds using a bounded memory at each node.

Finally, we show that our solution allows to efficiently simulate synchronous self-stabilizing algorithms in an asynchronous environment. This provides new state-of-the-art algorithm solving both the leader election and the spanning tree construction problem in any identified connected network which, to the best of our knowledge, beat all known solutions in the literature.

Email Adresses: `stephane.devismes@u-picardie.fr` (Stéphane Devismes), `david.ilcinkas@labri.fr` (David Ilcinkas), `johnen@labri.fr` (Colette Johnen), `frederic.mazoit@labri.fr` (Frédéric Mazoit)

1 Introduction

Context. *Self-stabilization* is a general non-masking and lightweight fault tolerance paradigm [Dij74, ADDP19]. Precisely, a distributed system achieving this property inherently tolerates *any* finite number of transient faults.[1] Indeed, starting from an arbitrary configuration, which may be the result of such faults, a self-stabilizing system recovers *within finite time*, and without any external intervention, a so-called *legitimate configuration* from which its specification is satisfied.

The difficulty of achieving fault tolerance in distributed systems mainly relies on their asynchronous aspect. The impossibility of achieving consensus in an asynchronous system in spite of at most one process crash [FLP85] is a famous example illustrating this fact. Thus, fault tolerance, and in particular self-stabilization, often requires some kind of barrier synchronization to control the asynchronism of the system by making processes progress roughly at the same speed.

In that spirit, the *asynchronous unison problem* (unison for short) is a basic yet fundamental problem that helps the design of asynchronous distributed systems, especially self-stabilizing ones. The unison problem consists in maintaining a local clock at each node; the domain of clocks being infinite or bounded. Each node should increment its own clock infinitely often.[2] Furthermore, the safety property of the unison requires the difference between the clocks of any two neighbors to always be at most one increment. Notice that this problem can be trivially generalized (as done here) by constraining increments at each node p to the satisfaction of some local predicate $P(p)$ (*n.b.*, we retrieve the initial problem if $P(p) \equiv true$).

Unison has numerous applications, especially in self-stabilization. Among others, it can be used to simulate synchronous systems in asynchronous environments [AD17, DDL19], free an asynchronous system from its fairness assumption (using the cross-over composition) [BGJ01], facilitate the termination detection [BJLBP22], or achieve infimum computation and local resource allocation [BP08].

In this paper, we consider the unison problem in the most commonly used model of the self-stabilizing area: the *atomic-state* model [Dij74, ADDP19]. This model is a locally-shared memory model with composite atomicity: the

[1] A transient fault occurs at an unpredictable time, but does not result in a permanent hardware damage. Moreover, as opposed to intermittent faults, the frequency of transient faults is considered to be low.

[2] In case the clock values are bounded, increments are modulo some value B, called the *period*.

state of each node is stored into registers and these registers can be directly read by neighboring nodes; moreover, in one atomic step, a node can read its state and that of its neighbors, perform some local computations, and update its state. In the atomic-state model, asynchrony is materialized by an adversary called *daemon* that restricts the set of possible executions. We consider here the weakest (*i.e.*, the most general) daemon: the *distributed unfair daemon*.

Self-stabilizing algorithms are mainly compared according to their *stabilization time*, *i.e.*, the worst-case time to reach a legitimate configuration starting from an arbitrary one. In the atomic-state model, stabilization time can be evaluated in terms of rounds and moves. Rounds [CDPV02] capture the execution time according to the speed of the slowest nodes. Moves count the number of local state updates. So, the move complexity is rather a measure of work than a measure of time.

It turns out that obtaining efficient stabilization time both in rounds and steps is a difficult issue. Usually, techniques to design an algorithm achieving a stabilization time polynomial in moves usually makes its rounds complexity inherently linear in n, the number of nodes; see, e.g., [CDV09, ACD$^+$17, DJ19, DIJ22]. Conversely, achieving the asymptotic optimality in rounds, *i.e.*, $O(D)$ where D is the network diameter, most of the time makes the stabilization time in moves exponential; see, *e.g.*, [DJ16, GHIJ19]. In a best-effort spirit, Cournier *et al.* [CRV19] have proposed to study what they call *fully-polynomial* stabilizing solutions, *i.e.*, stabilizing algorithms whose round complexity is polynomial on the network diameter and move complexity is polynomial on the network size.[3]

Contribution. We propose the first fully-polynomial self-stabilizing unison in the atomic-state model assuming a distributed unfair daemon. This algorithm works in an anonymous network of arbitrary topology. Moreover, it does not require any local port labeling at nodes. In that sense, the computational model we use is a simplified version of the *stone age* model of Emek and Wattenhofer [EW13].

To the best our our knowledge, this is a first fully-polynomial self-stabilizing algorithm solving a *dynamic problem*.[4] This is also the first

[3]Actually, in [CRV19], authors consider atomic steps instead of moves. However, these two time units essentially measure the same thing: the workload. By the way, the number of moves and the number of atomic steps are closely related: if an execution e contains x steps, then the number y of moves in e satisfies $x \leq y \leq n \cdot x$.

[4]As opposed to a *static problem* that defines a task of calculating a function that depends on the system in which it is evaluated [Tix06].

self-stabilizing unison for arbitrary anonymous network achieving an asymptotically optimal stabilization time in rounds (*i.e.*, $O(D)$) using a bounded memory at each node.

In more detail, assuming a period $B \geq 2D + 2$, our solution stabilizes in at most $2D - 2$ rounds and $O(\min(n^2B, n^3))$ moves using $O(\log B)$ bits per node. Overall, our unison achieves an outstanding trade-off between time, workload, and space.

We also analyze the efficiency of our algorithm to simulate any synchronous self-stabilizing algorithm in an asynchronous environment (under the unfair daemon). If the input synchronous self-stabilizing algorithm is *silent*[5] and stabilizes in at most T synchronous rounds, then its simulation is also silent and self-stabilizing; moreover, its stabilization time is at most $5D + 3T$ rounds and $O(\min(n^2B, n^3)) + nT$ moves using $O(M + \log(B))$ bits per node, where M is the memory requirement of the input algorithm.

An important consequence of this latter result is that one can easily obtain the state-of-the-art leader election and BFS spanning tree construction of the literature for asynchronous identified and arbitrary connected networks simply by simulating the synchronous algorithm of Kravchik and Kutten [KK13]. Precisely, by simulating this algorithm using our unison, we obtain a stabilization time in $O(D)$ rounds and $O(\min(n^2B, n^3))$ moves using $O(\log(N))$ bits per node, where N is any upper bound on n. To the best of our knowledge, there was no such an efficient solution until now in the literature.

Related Work. The asynchronous unison studied here is a variant of the *synchronous unison* problem proposed by Even and Rajsbaum [ER90]. This latter problem is dedicated to synchronous system and requires all clocks increment infinitely often and to become eventually fully synchronized. In [ER90], Even and Rajsbaum consider this problem in a non-fault-tolerant context, yet assuming that nodes do not necessarily start at the same time.

Gouda and Herman [GH90] have proposed the first self-stabilizing synchronous unison. Their algorithm works in anonymous synchronous systems of arbitrary connected topology using infinite clocks. A solution working with the same settings, yet implementing bounded clocks, is proposed in [ADG91]. The asynchronous self-stabilizing unison problem in synchronous setting has been studied in [KK13], an algorithm converging in $O(D)$ with bounded memory is presented.

[5] In the atomic-state model, a self-stabilizing algorithm is silent if all its executions terminate.

Johnen *et al.* investigated the asynchronous self-stabilizing unison in oriented trees in [JADT02]. The first self-stabilizing asynchronous unison for general graphs was proposed by Couvreur *et al.* [CFG92] in the link-register model (a locally-shared memory model without composite atomicity). However, no complexity analysis was given. Another solution which stabilizes in $O(n)$ rounds has been proposed by Boulinier *et al.* [BPV04] in the atomic-state model assuming a distributed unfair daemon. Its move complexity is shown in [DP12] to be in $O(Dn^3 + \alpha n^2)$, where α is a parameter of the algorithm that should satisfies $\alpha \geq L-2$, where L is the length of the longest hole in the network. Boulinier proposes in his PhD thesis a parametric solution which generalizes both the solutions of [CFG92] and [BPV04]. In particular, the complexity analysis of this latter algorithm reveals an upper bound in $O(D.n)$ rounds on the stabilization time of the atomic-state model version of the Couvreur *et al.*'s algorithm.

Awerbuch *et al.* [AKM$^+$93] proposes a self-stabilizing unison (called clock synchronizer in their paper) that stabilizes in $O(D)$ rounds using an unbounded state space. The move complexity of their solution is not analyzed. An asynchronous self-stabilizing unison algorithm is given in [DJ19]. It stabilizes in $O(n)$ rounds and $O(\Delta.n^2)$ moves using unbounded local memories. Emek and Keren present in the stone age model [EK21] a self-stabilizing unison that stabilizes in $O(\mathcal{D}^3)$ rounds, where $\mathcal{D}$ is an upper bound on D known by all nodes. Their solution requires $O(\log(\mathcal{D}))$ bits per nodes. Moreover, since node activations are assumed to be fair, the move complexity cannot be bounded.

In [DIJM23], we propose a algorithm that transforms any terminating synchronous algorithms into an asynchronous silent self-stabilizing fully-polynomial algorithm. The memory requirement of the produced algorithm is in $O(T \times M)$ bits per nodes, where T and M are the time and space complexities of the input algorithm. This transformer thus cannot practically buid solutions for dynamic problems such as unison. Moreover, although it works on a strictly smaller class of algorithms, the synchronizer of the current paper has similar round and move complexities as the transformer of [DIJM23] while having a much better memory requirement.

Roadmap. The rest of the paper is organized as follows. The next section is dedicated to the computational model and basic definitions. In Section 3, we present our unison algorithm, prove its self-stabilization, and study its time complexity. In Section 4 deals with the simulation of synchronous self-stabilizing algorithms in an asynchronous environment using our unison

algorithm.

2 Preliminaries

2.1 Networks

We consider *distributed systems* made of $n \geq 1$ interconnected nodes. Each node can directly communicate through channels with a subset of other nodes, called its neighbors. We assume that the network is connected and that communication is bidirectional.

More formally, we model the topology by a connected simple graph $G = (V, E)$, where V is the set of *nodes* and E is the set of *edges*. If $\{p, q\}$ is an edge, then q is a *neighbor* of p. We denote by $N(p)$ the set of neighbors of p.

A *path* is a finite sequence $P = p_0 p_1 \cdots p_l$ of nodes such that consecutive nodes in P are neighbors. We say that P is *from* p_0 *to* p_l. The *length* of the path P is the number l. Since we assume that G is *connected*, then for every pair of nodes p and q, there exists a path from p to q. We can thus define the *distance* between two nodes p and q to be the minimum length of a path from p to q. The *diameter* D of G is the maximum distance between nodes of G.

2.2 Computational Model: the Atomic-state Model

Our algorithm runs on a variant of the *atomic-state model* [ADDP19] in which nodes communicate using a finite number of locally shared registers, called *variables*. The *state* of a node is defined by the values of its local variables. A *configuration* of the system is a vector consisting of the states of each node.

In one indivisible move, a node p reads its own variables and the set of states of its neighbors. Our algorithm is described as a finite set of *rules* of the form $label : guard \rightarrow action$. Labels are only used to identify rules in the reasoning. A *guard* is a Boolean predicate involving the state of the node and the set of states of its neighbors. The *action* part of a rule updates the state of the node. A rule can be executed only if its guard evaluates to *true*; in this case, the rule is said to be *enabled*. By extension, a node is said to be enabled if at least one of its rules is enabled. We denote by $Enabled(\gamma)$ the subset of nodes that are enabled in configuration γ.

In the model, executions proceed as follows. Given a configuration γ with $Enabled(\gamma) \neq \emptyset$, a so-called *daemon* selects a nonempty set $\mathcal{X} \subseteq Enabled(\gamma)$;

then every node of $\mathcal{X}$ *atomically* executes one of its enabled rules, leading to a new configuration γ'. The atomic transition from γ to γ' is called a *step*. We also say that each node of $\mathcal{X}$ executes an *action* or simply a *move* during the step from γ to γ'. The possible steps induce a binary relation over $\mathcal{C}$, denoted by $\mapsto$. An *execution* is a maximal sequence of configurations $e = \gamma^0\gamma^1 \cdots \gamma^i \cdots$ such that $\gamma^{i-1} \mapsto \gamma^i$ for all $i > 0$. The term "maximal" means that the execution is either infinite, or ends at a *terminal* configuration γ^f with $Enabled(\gamma^f) = \emptyset$. An algorithm which does not admit any infinite execution is called *silent*.

As explained before, each step from a configuration to another is driven by a daemon. We define a daemon as a predicate over executions. We say that an execution e is *an execution under the daemon* S if $S(e)$ holds. In this paper we assume that the daemon is *distributed* and *unfair*, meaning that it has no constraints, except that at each step it must select a nonempty set of enabled nodes. It might, for example, never select a specific enabled node unless it is the only enabled node.

We use two units of measurement to evaluate the time complexity: *moves* and *rounds*. The definition of a round uses the concept of *neutralization*: a node p is *neutralized* during a step $\gamma^i \mapsto \gamma^{i+1}$, if p is enabled in γ^i but not in configuration γ^{i+1}, and does not execute any action in the step $\gamma^i \mapsto \gamma^{i+1}$. Then, the rounds are inductively defined as follows. The first round of an execution $e = \gamma^0\gamma^1 \cdots$ is the minimal prefix e' such that every node that is enabled in γ^0 either executes a rule or is neutralized during a step of e'. If e' is finite, then let e'' be the suffix of e that starts from the last configuration of e'; the second round of e is the first round of e'', and so on and so forth.

The *stabilization time* of a self-stabilizing algorithm is the maximum time (in moves or rounds) over every execution possible under the considered daemon (starting from any initial configuration) to reach a legitimate configuration.

3 A unison algorithm

3.1 The algorithm

Data structures. Let $B \geq 2D + 2$ be an integer. Each node p maintains a single variable $p.v$ of datatype $Pairs = \{(C, x) \mid x \in [-B, B[\} \cup \{(E, x) \mid x \in [-B, 0[\}$. In the algorithm, $p.v$ will be accessed and modified implicitly as follows:

- $p.s$, called the *status* of p, will denote the left field of the pair $p.v$,

- $p.c$, called the *clock* of p, will denote the right field of the pair $p.v$.

For example, if $p.v = (s, c)$, then $p.s = s$ and $p.c = c$. Furthermore, any assigment $p.s := s$ (resp., $p.c := c$) should be understood as $p.v := (s, p.c)$ (resp., $p.v := (p.s, c)$). Finally, a node p such that $p.s = C$ is said to be *correct*; otherwise it is an *erroneous* node (in other words, a node in error).

We define the infix function $+_B$ as follows:

$$\begin{array}{rcll} B-1 +_B 1 & = & 0 & \\ n +_B 1 & = & n+1 & \text{if } n \neq B-1 \\ n +_B (m+1) & = & (n +_B m) +_B 1 & \end{array}$$

We also define a distance δ_B:

$$\begin{array}{rcll} \delta_B(n,n) & = & 0 & \\ \delta_B(n, n +_B 1) & = & 1 & \\ \delta_B(n +_B 1, n) & = & 1 & \\ \delta_B(n,m) & = & 2 & \text{otherwise.} \end{array}$$

If $\gamma^0\gamma^1\cdots$ is an execution, we respectively denote by $p.s^i$ and $p.c^i$ the value of $p.s$ and $p.c$ in γ^i.

Some predicates. Although they are a bit misleading because they suggest that a node can access its neighbors directly, we use the following notations:

$$\begin{array}{lrcl} \texttt{Macro1} & \exists q \in N(p), \mathrm{Pred}(\texttt{st}_q) & := & \exists \texttt{st} \in \{\texttt{st}_q \mid q \in N(p)\}, \mathrm{Pred}(\texttt{st}) \\ \texttt{Macro2} & \forall q \in N(p), \mathrm{Pred}(\texttt{st}_q) & := & \forall \texttt{st} \in \{\texttt{st}_q \mid q \in N(p)\}, \mathrm{Pred}(\texttt{st}) \end{array}$$

$$\begin{array}{rcl} \mathit{root}(p) & := & (p.s = E \wedge \neg(\exists q \in N(p),\ q.s = E \wedge q.c < p.c)) \\ & \vee & (p.s = C \wedge \exists q \in N(p),\ p.c < q.c \wedge \delta_B(q.c, p.c) \geq 2) \\ \mathit{activeRoot}(p) & := & \mathit{root}(p) \wedge (p.c \neq -B \vee p.s = C) \\ \mathit{errorPropag}(p,i) & := & \exists q \in N(p),\ q.s = E \wedge q.c < i < p.c \\ \mathit{canClearE}(p) & := & p.s = E \\ & \wedge & \forall q \in N(p),\ (q.c \in \{p.c-1, p.c, p.c+1\} \wedge \\ & & \qquad (q.c \neq p.c+1 \vee q.s = C)) \\ \mathit{unisonMove}(p) & := & p.s = C \wedge \forall q \in N(p),\ q.c \in \{p.c, p.c +_B 1\} \end{array}$$

The rules. We rarely use a unison algorithm alone. It is merely a tool to help another algorithm. It thus makes sense that our algorithm depends on some properties which are external to the unison algorithm and its variables. Our algorithm uses a predicate P_{aux} which is not defined. As a matter of fact, its influence on the analysis of the algorithm is very limited. We will specialize this predicate in Section 4 when using our unison algorithm as a synchronizer.

- $R_R : activeRoot(p) \longrightarrow p.c := -B \ ; \ v.s := E$
- $R_P(i) : errorPropag(p,i) \longrightarrow p.c := i \ ; \ p.s := E$
- $R_C : canClearE(p) \longrightarrow p.s := C$
- $R_U : unisonMove(p) \wedge$
  $(P_{\text{aux}}(p) \vee \exists q \in N(p), q.c = p.c +_B 1) \longrightarrow p.c := p.c +_B 1$

We set the following priorities:

- R_R has the highest priority.
- $R_P(i)$ has a higher priority than $R_P(i+l)$ for $l > 0$.
- R_C and R_U have the lowest priority.

A node p is a *root* if $root(p)$. In the following, an *error rule* is either the rule R_R or a rule $R_P(i)$.

The *legitimate* configurations are the configurations in which the only rule which can be executed is the rule R_U. Another equivalent characterization of legitimate configurations will be given in Section 3.3.

The following remark is quite important. Since, when encountering an error, the clock of a node becomes negative, and since no nodes in error can have a non-negative clock, it is natural to expect the "error recovery phase" to correspond to the time zone $[-B, 0[$, and the interval $[0, B[$ to correspond to the "legitimate configurations". This would suggest a round complexity of $\Omega(B)$. But this intuition is false. If a configuration γ is such that $p.s = C$ and $p.c = -B$ for every node p, then γ is a legitimate configuration.

3.2 Preliminary results

Lemma 1. *Let $\gamma^a \mapsto \gamma^b$ be a step. If p is a root in γ^b, then it also is in γ^a.*

Proof. Suppose by contradiction that p is a root in γ^b and not a root in γ^a.

We consider two cases.

- Suppose that $p.s^b = E$. Thus there exists no $q \in N(p)$ such that $q.s^b = E$ and $q.c^b < p.c^b$.

  If $p.s^a = E$ and no $q \in N(p)$ is such that $q.s^a = E$ and $q.c^a < p.c^a$, then p is a root in γ^a, a contradiction.

  We claim that in all remaining cases, p executes an error rule in $\gamma^a \mapsto \gamma^b$. Indeed,

  - if $p.s^a = E$ and there exists $q \in N(p)$ such that $q.s^a = E$ and $q.c^a < p.c^a$, then p cannot execute the rule R_U, q cannot execute the rules R_U or R_C, and thus $q.s^b = E$. We have $q.c^b \leq q.c^a < p.c^a$. So if $p.c^b \geq p.c^a$, then p is not a root in γ^b. Thus p must execute an error rule in $\gamma^a \mapsto \gamma^b$.
  - if $p.s^a = C$, then p must also execute an error rule in $\gamma^a \mapsto \gamma^b$.

  Now two cases arise.

  - If p executes the rule R_R, then p is a root in γ^a, a contradiction.
  - If p executes a rule $R_P(i)$ in $\gamma^a \mapsto \gamma^b$, then there exists $r \in N(p)$ such that $r.c^a = i - 1$ and $r.s^a = E$. But since $r.s^a = E$, r cannot execute the rule R_U, and because of p, r cannot execute the rule R_C. Thus $r.s^b = E$ and $r.c^b < p.c^b$, which contradicts the hypothesis.

- Suppose that $p.s^b = C$. Thus, there exists $q \in N(p)$ such that $p.c^b < q.c^b$ and $\delta_B(p.c^b, q.c^b) \geq 2$. Note that this implies that $q.c^b \geq p.c^b + 2$.

  Since p does not execute an error rule in $\gamma^a \mapsto \gamma^b$, either $p.c^b = p.c^a$ or $p.c^b = p.c^a +_B 1$.

  - Suppose that $p.c^b = p.c^a$. Let us study what happens during $\gamma^a \mapsto \gamma^b$.
    - If q executes the rule R_R, then $q.c^b = -B$, which contradicts the fact that $p.c^b < q.c^b$.
    - If q executes the rule R_U, then it means that $q.c^a \leq p.c^a$. And since $p.s^b = C$, p does not execute an error rule and thus $q.c^b \leq p.c^b + 1$, a contradiction.
    - If q executes no rules or the rule R_C, then $q.c^a = q.c^b$, and since $q.c^a \geq p.c^a + 2$, p cannot execute the rule R_C. Thus, we have $p.s^a = C$, which implies that p is also a root in γ^a, a contradiction.

* If q executes a rule R_P, then $q.c^a > q.c^b \geq p.c^b + 2 = p.c^a + 2$. Thus, we have $q.c^a > p.c^a + 2$, which prevents p from executing the rule R_C. Thus $p.s^a = C$, and since p is not a root in γ^a, $\delta_B(p.c^a, q.c^a) \leq 1$, a contradiction.

– Suppose that $p.c^b = p.c^a +_B 1$. Since $B - 1 \geq q.c^b \geq p.c^b + 2$, $p.c^b = p.c^a + 1$. This implies that p executes the rule R_U during $\gamma^a \mapsto \gamma^b$, and thus $q.c^a \in \{p.c^a, p.c^a +_B 1\} = \{p.c^a, p.c^a + 1\}$. If q executes the rule R_U during $\gamma^a \mapsto \gamma^b$, then $q.c^a = p.c^a$. In this case, we have $q.c^b = p.c^b$, a contradiction. Otherwise, we have $q.c^b \leq p.c^a + 1 = p.c^b$, again a contradiction.

□

Lemma 2. *Let $\gamma^a \mapsto \gamma^b$ be a step, and let r be a root in γ^a which executes the rule R_C during $\gamma^a \mapsto \gamma^b$. Then $r.c^a = -B$ and r is not a root in γ^b.*

Proof. Since R_R has a higher priority than R_C, the guard of R_R is false at r in γ^a. So, as r is a root in γ^a, we necessarily have $r.c^a = -B$.

Then, since r executes the rule R_C during $\gamma^a \mapsto \gamma^b$, we have $r.s^b = C$. Moreover, to allow r to execute the rule R_C, every $q \in N(r)$ should satisfy $q.c^a \leq -B + 1$. Now, as $r.c^a = -B$, no $q \in N(r)$ with $q.c^a = -B + 1$ can execute the rule R_U in $\gamma^a \mapsto \gamma^b$. All this implies that $r.s^b = C$, and for every $q \in N(p)$, $\delta_B(q.c^b, p.c^b) \leq 1$. So, $root(r)$ is false in γ^b, *i.e.*, r is not a root in γ^b. □

A path $P = p_0 p_1 \cdots p_l$ in G is *decreasing* in a configuration γ if for each $0 \leq i < l$, $p_i.c > p_{i+1}.c$. Moreover, P is an *E-path* if it is decreasing, all its nodes are in error, and its last node is a root.

Lemma 3. *Let γ be a configuration. Any node p in error is the first node of an E-path.*

Proof. We prove our lemma by induction on $p.c$. If $p.c = -B$, then p is a root and $P = p_0$ satisfies the required conditions.

Suppose that $p.c > -B$. If p is a root, then $P = p_0$ satisfies the required conditions. Otherwise, there exists $q \in N(v)$ such that $q.c < p.c$ and $q.s = E$. By induction, there exists an E-path P' starting at q. We can add p at the beginning of P' to obtain a path P which satisfies all required conditions. □

3.3 Legitimate configurations

A configuration γ is said to be *almost clean* if

- every root r satisfies $r.c = -B$ and $r.s = E$, and
- every two neighbors p and q satisfy $\delta_B(p.c, q.c) \leq 1$.

Lemma 4. *A configuration is almost clean if and only if no nodes can execute an error rule.*

Proof. Suppose that γ is almost clean. Since every root r is such that $r.c = -B$ and $r.s = E$, no nodes can execute the rule R_R, and since every neighbors p and q are such that $\delta_B(p.c, q.c) \leq 1$, no nodes can execute a rule R_P.

Conversely, suppose that γ is not almost clean. A root r verifying $r.c > -B$ or $r.s = C$ can execute the rule R_R. Let p and q be two neighbors. Assume, without loss of generality, that $p.c \leq q.c$. If $\delta_B(p.c, q.c) \geq 2$, then either $p.s = E$ and q can execute a rule R_P, or $p.s = C$ and p can execute the rule R_R. □

Lemma 5. *Let $\gamma^a \mapsto \gamma^b$ be a step. If γ^a is almost clean, then so is γ^b.*

Proof. Assume, for the purpose of contradiction, that γ^a is almost clean and γ^b is not.

At least one of the following two cases occurs, by Lemma 4.

- Some root r can execute the rule R_R in γ^b (i.e., $r.c^b > -B$ or $r.s^b = C$).

  First, by Lemma 1, r is a root in γ^a, and since γ^a is almost clean, $r.c^a = -B$ and $r.s^a = E$. Thus, either r executes no rules in $\gamma^a \mapsto \gamma^b$, which is a contradiction with $r.c^b > -B$ or $r.s^b = C$, or r executes the rule R_C and r is not a root in γ^b by Lemma 2, which also leads to a contradiction.

- Some node p can execute a rule R_P in γ^b. There exists $q \in N(p)$ such that $B - 1 \geq p.c^b \geq q.c^b + 2$ and $q.s^b = E$. Since γ^a is almost clean, no error rules are executed in the step $\gamma^a \mapsto \gamma^b$. Thus $q.s^a = E$ and q executes no rules in $\gamma^a \mapsto \gamma^b$, so $q.c^a = q.c^b$. Moreover, $\delta_B(p.c^a, q.c^a) \leq 1$. This implies that p must execute the rule R_U. But then $p.c^a \geq q.c^a + 1$ as $p.c^b = p.c^a + 1 \geq q.c^b + 2 = q.c^a + 2$. Thus $q.c^a \notin \{p.c^a, p.c^a +_B 1\}$, which forbids p from executing the rule R_U, a contradiction.

□

Lemma 6. *In any almost clean configuration γ, there exists $c \in [0, B[$ such that for any p, $p.c \neq c$.*

Proof. Suppose that for all $c \in [0, B[$, there exists p such that $p.c = c$. Hence, there is a node p whose clock value is D ($p.c = D$) in γ. We can prove by induction on l that any node q at distance at most l from p has a clock value in $[D-l, D+l]$. We conclude that no node p is such that $p.c = 2D+1 \leq B-1$, a contradiction. □

Lemma 7. *Let γ be an almost clean configuration. There exists $c_{\min} \in [-B, B[$ and $\Delta_c \leq D$ such that $\{p.c \mid p \in V\} = \{c_{\min} +_B i \mid 0 \leq i \leq \Delta_c\}$.*

Proof. We consider two cases.

- Suppose that there exists p such that $p.c < 0$. Let $c_{\min} = \min(p.c \mid p \in V)$, and let Δ_c be the minimum natural integer such that no node q is such that $q.c = c_{\min} + \Delta_c + 1$ (Δ_c exists by Lemma 6).

- Suppose that no node p is such that $p.c < 0$. By Lemma 6, there exists $c \in [0, B[$ which is not the clock value of any node. Since clock values are non-negative, there exists a minimum i such that $c_{\min} = c +_B i$ is a clock value of a node p. We choose Δ_c minimum such that no node q is such that $q.c = c_{\min} +_B (\Delta_c + 1)$.

Clearly, $\{c_{\min} +_B i \mid 0 \leq i \leq \Delta_c\} \subseteq \{p.c \mid p \in V\}$. Now, equality and the fact that $\Delta_c \leq D$ follow from the fact that G is connected and that, between two consecutive nodes of any path, the clock value can only change by one. □

A configuration is said to be *clean* if it contains no roots. Lemma 1 implies that being clean is a closed property. The following lemma gives an alternative definition of being clean, and as a direct consequence, it implies that clean configurations are also almost clean. It also implies that the legitimate configurations are the clean ones.

Lemma 8. *A configuration is clean if and only if nodes can only execute the rule R_U.*

Proof. Suppose that γ is clean. Since it contains no roots, then no nodes can execute the rule R_R. Since there are no roots, then, by Lemma 3, there are no nodes in error, and thus no nodes can execute a rule R_P or the rule R_C.

Conversely, suppose that nodes can only execute the rule R_U. Then by Lemma 4, γ is almost clean. Therefore γ contains no roots having the status C. To prove that γ does not contain any root in error, it is enough to show that γ contains no nodes in error (Lemma 3). Suppose that in γ one or several nodes are in error. Let p be a node in error having the largest clock value. Since γ is almost clean, every neighbor q of p satisfies $\delta_B(p.c, q.c) \leq 1$.

By definition of p, a neighbor of p in error has a clock value smaller than or equal to $p.c$. Hence, p can execute the rule R_C, a contradiction. □

Lemma 9. *Let $e = \gamma^0\gamma^1\cdots$ be an execution such that γ^0 is clean. In any configuration γ^i of e, if a node p satisfies P_{aux}, then at least one node q can execute the rule R_U in $\gamma^i \mapsto \gamma^{i+1}$.*

Proof. By Lemma 1, the configuration γ^i is also clean (and almost clean as well by Lemma 8). According to Lemma 7, in γ^i, there exists $c_{min} \in [-B, B[$ and $\Delta_c \leq D$ such that $\{p.c \mid p \in V\} = \{c_{min} +_B l \mid 0 \leq l \leq \Delta_c\}$. Moreover, in γ^i, the clock value of every neighbor of any node p such that $p.c = c_{min}$ belongs to $\{c_{min}, c_{min} +_B 1\}$. If $\Delta_c = 0$, then any node which satisfies P_{aux} can execute the rule R_U as all nodes have the same clock value and have status C. Otherwise, there exists a node p with $p.c = c_{min}$ which has a neighbor q such that $q.c = c_{min} +_B 1$, and so p can execute the rule R_U in γ^i. □

3.4 D-paths

Recall that a path $P = p_0p_1\cdots p_l$ in G is *decreasing* in a configuration γ if for each $0 \leq i < l$, $p_i.c > p_{i+1}.c$ and that P is an *E-path* if it is decreasing, all its nodes are in error, and its last node is a root.

We extend these definitions in the following way. A path P is *gently decreasing* if, for each $0 \leq i < l$, we have $p_i.c = p_{i+1}.c + 1$. It is a *D-path* if it is decreasing and there exists $0 \leq j \leq l$ such that

- $P_C = p_0\cdots p_{j-1}$ is a (possibly empty) gently decreasing path of nodes in C,
- $P_E = p_j\cdots p_l$ is an E-path.

We call P_C and P_E the *correct* and *error* parts of P.

Lemma 10. *Let $\gamma^a \mapsto \gamma^b$ be a step, and let P be a D-path in γ^a. For any $p \in P$, node p does not execute the rule R_U at that step, and thus $p.c^b \leq p.c^a$. Moreover, if $p \in P$ is such that $p.s^b = C$, then we have equality.*

Proof. Let $p \in P$. Recall that $p.c$ only increases if p executes the rule R_U.

- If p is the last node of P, then in γ^a, p is a root such that $p.s^a = E$. Thus p cannot execute the rule R_U in $\gamma^a \mapsto \gamma^b$.

- If p is not the last node of P, let q be the next node after p on P. Since P is decreasing in γ^a, $q.c^a < p.c^a$. To be able to execute the rule R_U, we must have $q.c^a \in \{p.c^a, p.c^a +_B 1\}$, which is only possible if $q.c^a = 0$ and $p.c^b = B - 1$. But then the definition of a D-path requires that $q.s^a = E$, and p can execute the rule $R_P(1)$ and thus cannot execute the rule R_U in $\gamma^a \mapsto \gamma^b$.

The first part of the lemma follows. Now if $p.s^b = C$, then p does not execute an error rule in $\gamma^a \mapsto \gamma^b$, and thus $p.c^b \geq p.c^a$, which completes the proof. □

Lemma 11. *Let $\gamma^a \mapsto \gamma^b$ be a step, let $P = p_0 \cdots p_l$ be a decreasing path in γ^a such that*

- *apart from p_l which satisfies $p_l.s^a = E$ and $p_l.s^b = C$, all the nodes of P are in C in both γ^a and γ^b;*
- *in γ^a, $p_0 \cdots p_{l-1}$ is gently decreasing.*

Then P is gently decreasing in γ^b.

Proof. The assumptions imply that p_l executes the rule R_C in the step $\gamma^a \mapsto \gamma^b$. Thus $p_l.c^b = p_l.c^a$.

We claim that, for any $0 \leq i < l$, $p_i.c^b = p_i.c^a$. Indeed, since $p_l.s^a = E$, by Lemma 3, p_l is the first node of an E-path in γ^a that we use to extend P into a D-path P'. The claim then follows by Lemma 10 applied to P'.

The path P is decreasing in γ^a, and in particular $p_{l-1}.c^a > p_l.c^a$. Moreover, p_l executes the rule R_C, and thus we have $p_{l-1}.c^a = p_l.c^a + 1$. As the beginning of the path is gently decreasing by hypothesis, P is gently decreasing in γ^a. Finally, since the clock values of nodes of P are the same in γ^a and in γ^b, the lemma follows. □

Lemma 12. *Let $\gamma^a \mapsto \gamma^b$ be a step. Let p be the first node of a D-path P in γ^a. If at least one node of P is in error in γ^b, then p is the first node of a D-path in γ^b.*

Proof. Let $P = p_0 \cdots p_l$ be a D-path in γ^a and let $p = p_0$. Assume that P contains at least one node in error in γ^b, and let $0 \leq i \leq l$ be minimal such that $p_i.s^b = E$.

Let P' be the possibly empty path $p_0 \cdots p_{i-1}$. Since $p_i.s^b = E$, there exists an E-path $Q = p_i q_1 \cdots q_h$ in γ^b, by Lemma 3. We now claim that $P'' = p_0 \cdots p_i q_1 \cdots q_h$ is a D-path in γ^b whose first node is p.

We first prove that P'' is decreasing. Indeed, by Lemma 10, $p_i.c^b \leq p_i.c^a$ and, for $0 \leq j < i$, $p_j.c^b = p_j.c^a$. Since P is decreasing in γ^a, so is $P' p_i$ in γ^b.

Now $p_i q_1 \cdots q_h$ is an E-path in γ^b and is thus also decreasing which implies that so is P''.

To finish the proof, we must show that P' is gently decreasing in γ^b. Let P_c be the correct part of P in γ^a. Since both P' and P_C are prefixes of P, we have 2 cases:

- Assume that P' is a prefix of P_C. Since P_C is gently decreasing in γ^a, so is P'. And since all nodes of P' are still correct in γ^b, Lemma 10 implies that P' is gently decreasing in γ^b.
- Otherwise, P_C is a strict prefix of P'. Since, in a D-path, at most one node can execute the rule R_C, we have $P' = P_C p_{i-1}$. The fact that P' is gently decreasing in γ^b follows from Lemma 11.

□

Lemma 13. *Let $\gamma^a \mapsto \gamma^b$ be a step, let p be the first node of a D-path, and let r be its root in γ^a. If r is still a root in γ^b, then p is the first node of a D-path in γ^b.*

Proof. If r executes the rule R_C during $\gamma^a \mapsto \gamma^b$, then Lemma 2 implies that r is not a root in γ^b, which is a contradiction. Thus r is in error in γ^b, and the lemma follows from Lemma 12. □

Lemma 14. *Let $\gamma^a \mapsto \gamma^b$ be a step. Let p be the first node of a D-path in γ^a. If no D-paths in γ^b contain p, then $p.c^b \leq -B + n$.*

Proof. Let p be the first node of a D-path P, and let r be the root of P in γ^a.

We claim that, in γ^b, P contains no nodes in error. Indeed, otherwise Lemma 12 implies that p is the first node of a D-path in γ^b, which is a contradiction.

Since, in a D-path, at most one node can execute the rule R_C during a step, then in γ^a, all the nodes of P but r have status C. We can thus apply Lemma 11 and obtain that P is gently decreasing in γ^b, and thus $p.c^b = \mathit{length}(P) + r.c^b$. Since no nodes can appear twice in P, we have $p.c^b \leq r.c^b + n$.

Now since $r.s^b = C$, r executes the rule R_C in $\gamma^a \mapsto \gamma^b$. But then Lemma 2 implies that $r.c^a = -B$, and thus $r.c^b = -B$, and the lemma follows. □

3.5 Bounds on the clock values

Lemma 15. *If $i < j$ and p satisfies $p.c^j > p.c^i + 2D$, then for any q, there exists $i \leq h < j$ such that $q.c^h = p.c^i + D$ and q executes the rule R_U in the step $\gamma^h \mapsto \gamma^{h+1}$.*

Proof. First, notice that $p.c^i + 2D < B - 1$ by hypothesis. Then, we prove by induction on $d(q,p)$ that there exist $i \leq i' < j' \leq j$ such that $q.c^{i'} \leq p.c^i + d(q,p)$ and $p.c^j - d(q,p) \leq q.c^{j'}$.

- If $d(q,p) = 0$, then $q = p$ and $i' = i$ and $j' = j$ do the trick.
- If $d(q,p) > 0$ then let $q' \in N(q)$ be such that $d(q',p) = d(q,p) - 1$. By induction, there exists $i \leq i_1 < j_1 \leq j$ such that $q'.c^{i_1} \leq p.c^i + d(q,p) - 1$ and $p.c^j - d(q,p) + 1 \leq q'.c^{j_1}$.

  Now $q'.c^{j_1} - q'.c^{i_1} \geq p.c^j - p.c^i - 2(d(q,p)-1) > 2D - 2(d(q,p)-1)$. So, $q'.c^{j_1} - q'.c^{i_1} > 2$. Thus, there exists $i_1 \leq i' < j_1$ such that $q'.c^{i'} = q'.c^{i_1}$ and q' executes the rule R_U in $\gamma^{i'} \mapsto \gamma^{i'+1}$. Since q is a neighbor of q', we have $q.c^{i'} \leq q'.c^{i'} + 1 = q'.c^{i_1} + 1 \leq p.c^i + d(q,p)$.

  Now since $q'.c^{i'+1} + 2 \leq q'.c^{j_1}$, there exists $i' < j' < j_1$ such that q' executes the rule R_U in $\gamma^{j'} \mapsto \gamma^{j'+1}$ and $q'.c^{j'+1} = q'.c^{j_1}$. Since q is a neighbor of q', we have $q.c^{j'} \geq q'.c^{j'} = q'.c^{j_1} - 1 \geq p.c^j - d(q,p)$, which finishes the proof of our induction.

Let q be any node. Let $i \leq i' < j' \leq j$ such that $q.c^{i'} \leq p.c^i + d(q,p) \leq p.c^i + D$ and $q.c^{j'} \geq p.c^j - d(q,p) \geq p.c^j - D > p.c^i + D$. There exists $i' \leq h < j'$ such that $q.c^h = p.c^i + D$ and q executes the rule R_U in the step $\gamma^h \mapsto \gamma^{h+1}$. □

Lemma 16. *Suppose that γ^h is not clean. For any node p and any $i < j \leq h$, $p.c^j - p.c^i \leq 2D$.*

Proof. Let r be a root in γ^h and let $i < j \leq h$. By Lemma 1, r is a root in every configuration γ^l with $l \leq h$, and since no roots can execute the rule R_U, the lemma follows from Lemma 15. □

3.6 Move complexity

In this section, we analyze the move complexity of our algorithm. To do so, we fix an execution $e = \gamma^0\gamma^1\cdots$ and study the rules a given node executes in it. Since these rules do not appear explicitly in an execution, we propose to use a proxy for them.

A pair (p, i) is a *move* if p executes a rule in $\gamma^i \mapsto \gamma^{i+1}$. This move is a *$U$-move* if the rule is R_U, a *C-move* if the rule is R_C, a *R-move* if the rule is R_R, and a *$P(i)$-move* if the rule is $R_P(i)$. Since a node p executes at most one rule in a given step, the number of steps in which a given node executes a rule is the number of its moves.

Let S_i be the set of roots in γ^i. Lemma 1 states that for each $i > 0$, $S_i \subseteq S_{i-1}$. Since γ^0 contains at most n roots, there are $l \leq n$ steps $\gamma^{i-1} \mapsto \gamma^i$ for which $S_i \subset S_{i-1}$. Let $r_1, r_2, \ldots, r_l$ be the sequence of increasing indices such that $\forall i \in [1, l], S_{r_i} \subset S_{r_i - 1}$. This sequence gives the following *decomposition* of e into segments.

- The *first segment* is the sequence $\gamma^0 \cdots \gamma^{r_1}$.
- For $1 < i \leq l$ the *i-th segment* is the sequence $\gamma^{r_{i-1}} \cdots \gamma^{r_i}$.
- The *last segment* is the sequence $\gamma^{r_l} \cdots$.

A segment is said to be *clean* if its first configuration is clean. If the first configuration of a segment has a root, then the segment is said to be *unclean*. According to Lemma 1, if the first configuration of a segment is clean then the other configurations of the execution are clean. So, there is at most one clean segment, the last one, in any execution.

R-moves.

Lemma 17. *A node p executes at most one R-move.*

Proof. Let p be a node. We have three cases.

- If p executes no R-moves, it executes at most one R-move.
- If p executes a R-move and no moves after the first R-move, then p executes only one R-move.
- Otherwise, let (p, i) be the first R-move (thus $p.c^{i+1} = -B$ and $p.s^{i+1} = E$), and let (p, j) be the first move which follows. Consequently, (p, j) is necessarily a C-move. The result then follows from Lemmas 2 and 1.

□

U-moves. Note here that the predicate P_{aux} can only prevent a node from executing the rule R_U. Hence, since we consider distributed unfair daemons, an execution with any predicate P_{aux} is a valid execution with the predicate $P_{\text{aux}} = \mathit{true}$ while the configuration is not clean. We therefore consider in this part of the analysis that $P_{\text{aux}} = \mathit{true}$.

Lemma 18. *Let s be a segment. All U-moves done by p during s are done consecutively before the first error rule executed by p during s (if it exists).*

Proof. By definition of the rules R_U and R_C, U-moves of p are done consecutively before the first error rule executed by p. According to Lemma 3, after p executes an error rule, p is the first node of an E-path, and thus of a D-path, by definition. Lemma 13 implies that p remains in a D-path until the end of s. Hence, p no more executes the rule R_U in s, by Lemma 10, and we are done. □

To compute the move complexity, we must, in particular, compute the total number of moves in unclean segments. By definition, the rules R_R, R_P and R_C can only appear in unclean segments.

Lemma 19. *Let s be an unclean segment. A node p executes the rule R_U at most $2D$ times during s.*

Proof. By definition of s, there is a node r that is a root all along s. We now show, by induction on d, that every node p at distance $d \leq D$ from r executes at most $2d$ U-moves in s.

Base Case: If $d = 0$, then $p = r$. Now, r cannot execute a U-move during s.

Induction Step: Assume that p is at distance $d > 0$ from r. Let $q \in N(p)$ such that q is at distance $d - 1$ from r. By Lemma 18, if p, resp. q, changes its clock value during s, it does so by first executing a (possibly empty) sequence of U-moves, and then by executing a (possibly empty) sequence of error moves. By induction hypothesis, q executes $x \leq 2(d-1)$ U-moves in s. To prove the induction step, it is sufficient to prove that p does not execute more than $x + 2$ steps during s.

For the purpose of contradiction, assume that p executes at least $x + 3$ U-moves in s. Let c_p be the clock value of p just before its first U-move in s. There are $x + 3$ integers $t_1 < t_2 \cdots < t_{x+3}$ such that (p, t_i) is a U-move in s setting $p.c$ to the value $c_p +_B i$. By definition of the rule R_U, we must have $q.c^{t_i} \in \{c_p +_B (i-1), c_p +_B i\}$.

We claim that for any $1 \leq i \leq x+3$, node q has executed at least $i-2$, resp. $i-1$, U-moves between the beginning of the segment and γ^{t_i} when $q.c^{t_i} = c_p +_B (i-1)$, resp. $q.c^{t_i} = c_p +_B i$. We prove this claim by induction on i. The base case $i = 1$ is trivial. Assume that the property holds for $i \geq 1$ and let us consider the different cases. If $q.c^{t_{i+1}} = q.c^{t_i}$, then $q.c^{t_i} = c_p +_B i$ and we immediately have the desired property by induction hypothesis. Otherwise, we have $q.c^{t_{i+1}} = q.c^{t_i} +_B j$, with j being 1 or 2. Since $B \geq 4$, the value $q.c^{t_{i+1}}$ is either non-negative, or larger than $q.c^{t_i}$. Since executing an error rule always decreases the clock value, and sets it to a negative value, q cannot use any error rule to obtain for the first time the clock value $q.c^{t_{i+1}}$ from configuration γ^{t_i}. Therefore, q must perform at least j U-moves between γ^{t_i} and $\gamma^{t_{i+1}}$. Still by induction hypothesis, we thus obtain the desired property also in this case, which concludes the proof of the claim. Using it with $i = x+3$ allows us to obtain the expected contradiction, hence proving the overall induction step.

The lemma directly follows from the overall induction. □

Lemma 20. *A node p has at most $2Dn$ U-moves in the unclean segments.*

Proof. By Lemma 19, p executes the rule R_U at most $2D$ times in an unclean segment. Since there are at most n unclean segments (Lemma 1), the lemma follows. □

P-moves with B. We bound the number of P moves in 2 ways: using B, and without using B.

Lemma 21. *A node can have at most nB P-moves.*

Proof. Let p be a node. In a clean segment, p cannot execute a rule R_P. In an unclean segment, by Lemma 18, once p executes a P-move, it cannot execute the rule R_U anymore. Each time p executes the rule R_P, the variable $p.c$ decreases by at least one and takes a value in $[-B, 0[$. Hence, p can only execute B P-moves in an unclean segment. Since there are at most n unclean segments (Lemma 1), the lemma follows. □

P-moves without B. We now need several definitions.

We say that a P-move (p, t) *causes* another P-move (p', t') if

- $p' \in N(p)$, $t' > t$,

- for some l, (p', t') is a $P(l)$-move and (p, t) is a $P(l-1)$-move, and
- for any $t < k < t'$, (p, k) is not a move.

If a node p is in error in some configuration γ^i, this often happens because of some previous P-move (p, t). Moreover, what allowed (p, t) is some $q \in N(p)$ which is in error in γ^{t-1}. Finally, the reason why q is in error in γ^{t-1} is because of some previous move and so on. This motivates the following definition: a *causality chain* is a sequence $C = (p_0, t_0)(p_1, t_1) \dots (p_l, t_l)$ such that

- for each $0 \leq i < l$, (p_i, t_i) causes (p_{i+1}, t_{i+1});
- no (p, t) causes (p_0, t_0).

By construction, any P-move is the last element of a causality chain but the causality chain may not be unique.

We classify the P-move of p in 3 types.

- (p, i) is of Type 1 if there exists a P-move (p, j) with $j > i$ such that $p.c^{i+1} = p.c^{j+1}$.
- (p, i) is of Type 2 otherwise. And we subdivide Type 2 P-moves in
  - Type 2a. if at least one causality chain $C = (p_0, t_0) \dots (p_l, t_l)$ ending in (p, i) does not contain a repeated node. More formally, for any $0 \leq i < j \leq l$, $p_i \neq p_j$.
  - Type 2b. otherwise.

Our goal is to separately bound the number of P-moves of each type that a node can execute.

Lemma 22. *There are at most as many P-moves of type 1 as there are U-moves in the unclean segments.*

Proof. Suppose that (p, i) and (p, j) are both $P(l)$-moves with $i < j$. This means that $p.c^{i+1} = p.c^{j+1} = l$. For (p, j) to be possible, $p.c$ has to go from l in γ^{i+1} to being strictly greater than l in γ^j. This implies that there exists $i < k < j$ such that (p, k) is a U-move with $p.c^k = l$.

Thus, if we associate to each (p, i) of type 1 the U-move (p, j) such that $p.c^{i+1} = p.c^j$ with $j > i$ minimum, then no 2 distinct P-moves correspond to the same U-move. This implies that p has at most as many P-moves of type 1 as it has U-moves in unclean segments. □

Remark that, by definition, two P-moves (p, i) and (p, j) of type 2 are such that $p.c^{i+1} \neq p.c^{j+1}$. To count the number of P-moves (p, i) of type 2, we thus count the number of values that $p.c^{i+1}$ can take.

Lemma 23. *A node p can have at most $n(n+1)$ P-moves of type 2a.*

Proof. Let (p, i) be a P-move of type 2a, and let $C = (p_0, t_0) \dots (p_l, t_l)$ be a corresponding causality chain. We have

- $(p, i) = (p_l, t_l)$
- for any $0 \leq i < j \leq l$, $p_i \neq p_j$.

Clearly, $l < n$ and $p_l.c^{t_l+1} = l + p_0.c^{t_0+1}$. Let $r \in N(p_0)$ be such that $r.s^{t_0} = E$ and $p_0.c^{t_0+1} = r.c^{t_0} + 1$. Since no P-move causes (p_0, t_0), two cases arise:

- the last move of r before t_0 is an R-move in which case $r.c^{t_0} = -B$,
- r executes no rule before t_0 in which case $r.c^{t_0} = r.c^0$.

Thus $p_0.c^{t_0+1}$ can take at most $n+1$ distinct values. The lemma now follows from the fact that l can take at most n distinct values. □

Lemma 24. *A node p can have at most $2n + 2D$ type 2b P-moves.*

Proof. Let (p, i) be a P-move of type 2b, and let $C = (p_0, t_0) \dots (p_l, t_l)$ be a causality chain such that $(p, i) = (p_l, t_l)$.

By definition, there exists $0 \leq i < j \leq l$ such that $p_i = p_j$. Choose such a $i_0 = i$ and $j_0 = j$ with j_0 maximum. We thus have that for any $j_0 \leq i < j \leq l$, $p_i \neq p_j$ and thus $l - j_0 < n$. Let $q = p_{j_0} = p_{i_0}$.

Now $p_l.c^{l+1} = q.c^{j_0+1} + (l - j_0)$. To prove the lemma, it is thus enough to show that $q.c^{j_0+1} \leq n + 2D$.

We have that $q.s^{i_0+1} = E$, thus, by Lemma 3, q is the first node of an E-path, and thus of a D-path in γ^{i_0+1}.

Since $q.c^{j_0+1} > q.c^{i_0+1}$, q executes a U-move (q, k) for $i_0 < k < j_0$. By Lemma 10, q belongs to no D-path in γ^k. There thus exists $i_0 \leq k' < k$ such that q belongs to a D-path in $\gamma^{k'}$ and to no D-path in $\gamma^{k'+1}$. By Lemma 14, $q.c^{k'+1} \leq n$.

Since q is in error in γ^{j_0}, by Lemma 3, q belongs to an E-path in j_0. There thus exists a root r in γ^{j_0}. By Lemma 16, $q.c^{j_0+1} \leq n + 2D$. The lemma follows. □

Lemmas 21-24 directly imply the following lemma.

Lemma 25. *During an execution, there are at most $O(n^3)$ P-moves.*

C-moves.

Lemma 26. *During an execution, the number of C-moves is at most the number of P-moves plus n.*

Proof. Between two C-moves, a node p must execute an error move.

But since, after a C-move, p is can no longer be a root (by Lemmas 1 and 2), p cannot execute a C-move before an R-move. Thus p can execute at most one more C-move than its number of P-moves. □

The move complexity theorem. The following theorem is a direct corollary of Lemmas 17, 20, 25, and 26.

Theorem 1. *Our algorithm converges in $O(\min(n^2B, n^3))$ moves.*

3.7 Round complexity

Throughout this section, we consider an arbitrary execution $e = \gamma^0 \cdots$. For all $i \geq 1$, we denote by γ^{h_i} the last configuration of the i^{th} round (*n.b.*, e is finite, by Theorem 1, so there is no infinite round in e and from the last configuration of e, rounds are empty). We also let $\gamma^{h_0} = \gamma^0$.

In the first $D+1$ rounds, nodes execute error rules to "correct" the initial configuration. During the $D+1$ next rounds, all nodes go back to the correct state. The predicate P_{aux} has no influence on results of this section as R_U executions along e do not impact our analysis.

The "error broadcast phase".

Lemma 27. *For any $h \geq h_{D+1}$, in γ^h, for any root r, we have $r.S = E$ and $p.c \leq -B + d(r,p)$, for any node p.*

Proof. If γ^h contains no root, then the lemma holds. Otherwise, let r be any such root. By Lemma 1, r is also a root in all γ^i with $i \leq h$.

We first prove that $r.c^{h'} = -B$ and $r.s^{h'} = E$ for any h' with $h_1 \leq h' \leq h$. This claim will establish the first part of the lemma and the base case of the next induction.

First, during the first round, while $r.c \neq -B$ or $r.s \neq E$, r is enabled for R_R. Hence, by definition of a round and Rule R_R, there is a configuration in the first round where $r.c = -B$ and $r.s = E$. From such a configuration, the next rule r may execute is R_C. Now, by executing R_C, r is not a root anymore, by Lemmas 1-2. So, r cannot execute R_C before the system reaches

Configuration γ^h. Hence, for any h' with $h_1 \leq h' \leq h$, $r.c^{h'} = -B$ and $r.s^{h'} = E$.

We now prove by induction on $j \geq 1$ that for all nodes p such that $d(p,r) < j$, $p.c^{h'} \leq -B + d(r,p)$ with $h_j \leq h' \leq h$.

If $j = 1$, then $p = r$ and the base case is trivial from the previous claim. Suppose now that $j > 1$. Let p be such that $d(r,p) = j$, and let $q \in N(p)$ be such that $d(r,q) = j-1$. By induction hypothesis, we have $q.c^{h'} \leq -B+j-1$ with $h_{j-1} \leq h' \leq h$.

We first prove that there exists h' such that $h_{j-1} \leq h' \leq h_j$ such that $p.c^{h'} \leq -B+j$. To do so, assume, by the contradiction, that for every h' with $h_{j-1} \leq h' \leq h_j$, $p.c^{h'} > -B + j$, which implies that $p.c^{h'} \geq q.c^{h'} + 2$. From the previous claim, we also know that p is not a root in any $\gamma^{h'}$. Assume that $q.s^{h'} = C$ for some h' with $h_{j-1} \leq h' \leq h_j$. Then, q is a root in $\gamma^{h'}$ and in γ^0, by Lemma 1. From the previous claim, we know that $q.s^{h''} = E$ for any h'' such that $h_1 \leq h'' \leq h$. Now, since $j - 1 \geq 1$, we obtain a contradiction. Thus, $q.s^{h'} = E$ and $p.c^{h'} \geq q.c^{h'} + 2$ for any h' with $h_{j-1} \leq h' \leq h_j$. Hence, p is enabled for executing $R_P(i)$ with $i \leq q.c^{h'} + 1 \leq -B + j$ in every configuration $\gamma^{h'}$. By definition of a round and Rules $R_P(i)$, there exists a configuration h'' with $h_{j-1} \leq h'' \leq h_j$, where $p.c^{h'} \leq -B+j$, a contradiction.

Finally, recall that $q.c^{h''} \leq -B + j - 1$ for every h'' such that $h' \leq h'' \leq h_{D+1}$, by induction hypothesis. So, $p.c^{h''} \leq -B + j$ since p cannot execute R_U. Hence, we are done with the induction and the lemma holds. □

Lemma 28. *For any $h \geq h_{D+1}$, γ^h is almost clean.*

Proof. By Lemma 5, we only need to show that $\gamma^{h_{D+1}}$ is almost clean. To do so and according to Lemma 4, we now show that no node can execute an error rule in $\gamma^{h_{D+1}}$. The fact that no node can execute the rule R_R in h_{D+1} follows from Lemma 27. Assume that in $\gamma^{h_{D+1}}$ a node p verifies $errorPropag(p,i)$. There exists $q \in N(p)$ such that $q.c < p.c - 1$ and $q.s = E$. By Lemma 3, there exists a E-path P of length l from q to a root r. This path implies that $q.c \geq r.c + l \geq r.c + d(r,q)$. But then $p.c > q.c + 1 \geq r.c + d(r,q) + 1 \geq r.c + d(r,p)$, which contradicts Lemma 27. Hence, we conclude that no node verifies $errorPropag(p,i)$ in $\gamma^{h_{D+1}}$, and we are done. □

The "error cleaning phase".

Lemma 29. *For any $h \geq h_{2D+2}$, γ^h is clean.*

Proof. We prove by induction on $0 \leq i \leq D+1$ that for any $j \geq h_{D+1+i}$, γ^j contains no E-path of length $> D - i$. According to Lemma 28, γ^j is almost clean.

Suppose that $i = 0$. In $\gamma^{h_{D+1}}$, a root is in a E-path (by definition of almost clean). If $\gamma^{h_{D+1}}$ contains no E-path, then $\gamma^{h_{D+1}}$ is clean as it contains no root. Otherwise, let P be a E-path. Let p and r be its first and last node, and let l be its length. By definition of a E-path, $p.c - r.c \geq l \geq d(p,r)$. By Lemma 27, $p.c - r.c \leq d(p,r)$. We thus have that $l = d(p,r) \leq D$. The base case thus holds.

Suppose that the hypothesis holds for $i \geq 0$. Again, if $\gamma^{h_{D+1+i}}$ contains no E-path, then it is clean. Let P be an E-path in $\gamma^{h_{D+1+i}}$. Let p be the first node of P. Since no nodes can execute an error rule during the round $D+1+i$, then P is also an E-path in $\gamma^{h_{D+i}}$. Moreover, R_C is not enabled on p in $\gamma^{h_{D+i}}$; otherwise, p would have done a move during the round $D+i+1$, and p would not been in a E-path in $\gamma^{h_{D+i+1}}$.

There thus exists $q \in N(p)$ such that $q.c > p.c$ which is in error in $\gamma^{h_{D+i}}$. The path qP is a E-path in $\gamma^{h_{D+i}}$. By induction, the length of qP is at most $D - i$, and thus the length of P is at most $D - (i+1)$. The hypothesis thus holds for $i+1$.

For $h \geq h_{2D+2}$, γ^h contains no E-path, which implies that γ^h is clean. □

The round complexity proof. Lemmas 28 and 29 directly imply that

Theorem 2. *Our algorithm converges in $2D+2$ rounds.*

4 Synchronizer

Using folklore ideas (see, *e.g.*, [AKM+93] and [EK21]), we can use our unison algorithm to simulate any synchronous self-stabilizing algorithm in an asynchronous environment under an unfair daemon. We now study such a simulation.

4.1 Time definition

In everyday life, we have a distinction between the value of a clock (modulo 24 hours) and the time. Both are obviouly linked. We would like to make a similar distinction here.

Let $e = \gamma^0 \cdots$ be an execution such that γ^0 is clean, and so almost clean too. According to Lemma 7, there exists $c_{\min} \in [-B, B[$ and $\Delta \leq D$ such that $\{p.c^0 \mid p \in V\} = \{c_{\min} +_B i \mid 0 \leq i \leq \Delta\}$. The *birth time*

of p is $time^0(p) = i - \Delta$ where i satisfies $p.c^0 = c_{\min} +_B i$. Moreover, $time^{j+1}(p) := time^j(p) + 1$ whenever p executes the rule R_U in $\gamma^j \mapsto \gamma^{j+1}$ (otherwise, $time^{j+1}(p) := time^j(p)$).

An important remark is that if $time(p) = time(q)$, then $p.c = q.c$. Moreover, $unisonMove(p)$ is true if and only if $time(p)$ is a local minimum. Note that the birth time of a node is in $[-D, 0]$.

4.2 The algorithm

We consider a synchronous self-stabilizing algorithm Alg_I which runs in a variant of the atomic-state model which is at least as expressive as the model of our unison algorithm. This means that we should be able to encode the macros `Macro1` and `Macro2` (defined page 8) in the model of Alg_I. In the following, we denote by T the stabilization time of Alg_I (in synchronous settings) and by $Trans(Alg_I)$ the simulation of Alg_I using our unison algorithm.

The basic idea of the simulation is that the execution of Alg_I is driven by the unison algorithm. To that goal, each node p stores its last two states in Alg_I using two additional variables: $p.old$ and $p.curr$. Once the unison algorithm has stabilized, if p is a local minimum (w.r.t. the time of the unison) and is about to increase its clock (by performing Rule R_U), it computes its next state $\widehat{Alg_I}(p)$ in Alg_I. It does so by selecting for each neighbor q the variable $q.curr$ if $p.c = q.c$, and $q.old$ otherwise (*i.e.*, when $q.c = p.c +_B 1$).

We thus modify the rule R_U in the following way:

$$\begin{array}{rcl} R_U : unisonMove(p) & \wedge & (P_{\text{aux}}(p) \vee \exists q \in N(p), q.c = p.c +_B 1) \\ & \longrightarrow & p.old := p.curr; \\ & & p.curr := \widehat{Alg_I}(p); \\ & & p.c := p.c +_B 1 \end{array}$$

Let us consider the execution after the unison has stabilized (*i.e.*, the suffix of the execution starting from the first clean configuration). The time of each node is thus defined. If $time(p)^i = t$, then we set $\mathtt{st}_p^t = p.curr$. Since the state of p changes if and only if its time does, this is well defined. For any positive t, we can then define the configuration η^t of Alg_I in which the state of each node p is $\mathtt{st}_p^t$. The folklore claim is that the sequence $\eta^0 \cdots$ is a synchronous execution of Alg_I.

When $P_{\text{aux}}(p)$ is always *true*, the clocks of the unison constantly change. Therefore, even if Alg_I is silent, its simulation is not. In order to obtain a

silent simulation in such a case, we instantiate the predicate $P_{\text{aux}}(p)$ such that p increments its clock (and thus performs a simulation step) only if the simulation step makes its state change. More precisely, we define two possible predicates as follows:

$$\begin{aligned} P_{\text{greedy}}(p) &= \mathit{true} \\ P_{\text{lazy}}(p) &= p \text{ is enabled in } Alg_I \end{aligned}$$

and say that our synchronizer runs in *greedy mode* if $P_{\text{aux}} = P_{\text{greedy}}$, and that it runs in *lazy mode* if $P_{\text{aux}} = P_{\text{lazy}}$.

4.3 Complexity analysis

Greedy mode. In greedy mode, Lemma 9 implies that the algorithm is never silent. Nevertheless, it is easy to see that, once unison has been reached, all nodes with minimum time can be activated. And since nodes cannot be deactivated unless executing R_U, after one round, the minimum time of a node has increased by at least one. Thus, after $O(D)$ rounds (and $O(\min(n^2B, n^3))$ steps), each round of $Trans(Alg_I)$ simulates at least one round of Alg_I.

Lazy mode.

Lemma 30. *In lazy mode, the maximum time of each node is at most T.*

Proof. Let $T_{\eta^0} \leq T$ be the number of rounds that Alg_I takes to be silent from the clean configuration η^0. Let $e = \eta^0 \cdots$ be an execution starting from η^0. We claim that no node p has the time $T_{\eta^0} + 1$ along e. Let p be any node. By time definition, $time^0(p) \leq 0$.

Suppose that $\eta^i \mapsto \eta^{i+1}$ is such that no node has a time greater than T_{η^0} in η^i. If $time^i(p) \leq T_{\eta^0} - 1$, then $time^{i+1}(p) \leq T_{\eta^0}$. Otherwise, $time^i(p) = T_{\eta^0}$ and no neighbor q of p is such that $q.c = p.c +_B 1$. Moreover, by definition of T_{η^0}, p is not enabled in Alg_I. So p cannot execute the rule R_U and $time^{i+1}(p) = T_{\eta^0}$ in η^{i+1}. □

Lemma 31. *If Alg_I reaches a terminal configuration in at most T synchronous rounds, then $Trans(Alg_I)$ reaches a terminal configuration in at most $nT + nD$ moves from a clean configuration.*

Proof. Let $e = \gamma^0 \cdots$ be an execution starting from a clean configuration. The birth time of a node p is in $[-D, 0]$. No node has a time greater than T along e. So a node executes the rule R_U at most $T + D$ times along e. □

Because of the previous lemma, in lazy mode, if Alg_I is silent, then all executions of $Trans(Alg_I)$ are finite. The round analysis is a bit more involved. We split the analysis in two parts: the number of rounds so that all nodes have positive time, and the additional number of rounds to reach silence.

In the following, we consider a (finite) execution $e = \gamma^0 \cdots \gamma^f$ be an execution where γ^0 clean. As previously, we denote by γ^{h_i} the last configuration of the i^{th} round of e, for any $i \geq 1$, and we let $\gamma^{h_0} = \gamma^0$.

Lemma 32. *In lazy mode, the time of all nodes is positive after at most* $2D$ *rounds.*

Proof. Let s be a node with birth time zero. Let $\lambda(p, i) := 2i + D + d(p, s)$. We prove by induction on $0 \leq j \leq 2D$ that if $i \leq 0$ and $\lambda(p, i) \leq j$, then $time^{h_j}(p) \geq i$.

Suppose that $j = 0$. If $\lambda(p, i) \leq 0$, then $i \leq -d(p, s)$. As γ^0 is clean, we have $time^{h_0}(p) time^0(p) \geq -d(p, s) \geq i$. So, the base case holds.

Suppose that $j > 0$. Let (p, i) be such that $\lambda(p, i) = j$. For any $q \in N[p]$, $\lambda(p, i) - \lambda(q, i - 1) = 2 + d(p, s) - d(q, s) > 0$. So $\lambda(q, i - 1) < j$ and, by induction hypothesis, $time^{h_{j-1}}(q) \geq i - 1$.

Two cases now arise:

- If $time^{h_{j-1}}(p) \geq i$, then we are done.
- If $time^{h_{j-1}}(p) = i - 1$, then $p \neq s$ (recall that $time^0(s) = 0$, and so $time^{h_{j-1}}(s) \geq 0 \geq i$). Then let $q \in N(p)$ be such that $d(q, s) < d(p, s)$. We have $\lambda(q, i) < j$, and thus, by induction hypothesis, $time^{h_{j-1}}(q) \geq i$. This implies that p can execute the rule R_U in $\gamma^{h_{j-1}}$, and thus will have done at last at γ^{h_j}.

□

We now focus on the part of the execution e in which all nodes have a positive time. We denote by $e' = \rho^0 \cdots \rho^f$ this part. From now on, we denote by ρ^{r_i} the last configuration of the i^{th} round in e', for any $i \geq 1$, and we let $\rho^{r_0} = \rho^0$.

We do not know how these times evolve during the rounds of e', nevertheless we know that if no nodes have time $l + 1$ in ρ^i but $time^{i+1}(p) = l + 1$, then p is enabled in Alg_I at η^l. We thus say that any such a node p *may start time* $l + 1$.

If no node are enabled for Alg_I in η^i, then in in η^{i+j} with $j > 0$, no node are enabled in Alg_I. Therefore, if p may start time $i + 1$, then either $i = 0$ or there exists $q \in N[p]$ which may start time i.

This motivates the following definition. A *starting sequence* for ρ^f is a sequence of nodes $s_1 s_2 \cdots s_H$ such that each s_i starts time i, and $s_{i-1} \in N[s_i]$ if $i > 1$. Note that if ρ^0 contains no node which may start time 1, then the algorithm is already silent. Otherwise, ρ^f must contain a starting sequence.

Lemma 33. *e' reaches a terminal configuration in at most $D + 3T - 2$ rounds in lazy mode.*

Proof. If ρ^f contains no starting sequence, then $\rho^f = \rho^0$, and e' indeed reaches a terminal configuration in at most $(D + 3T - 2)$ rounds.

Assume now that ρ^f contains the starting sequence $s_1 \cdots s_T$. We also let $s_i = s_1$, for any $i < 1$. For any node p and $0 \leq i \leq T$, we let $\lambda(p, i) = 3i - 2 + d(p, s_i)$. The lemma is a direct consequence of the following induction.

We now prove by induction on $0 \leq j \leq 3T + D - 2$ that for every p and i such that $\lambda(p, i) \leq j$, we have $time^{r_j}(p) \geq i$.

If $j = 0$, then $i = 0$ and the result is clear.

Suppose that $j > 0$. If no (p, i) such that $\lambda(p, i) = j$ exists, then we are done. Otherwise, let (p, i) be such a pair. For any $q \in N[p]$, $\lambda(p, i) - \lambda(q, i - 1) = 3 + d(p, s_i) - d(q, s_{i-1})$, and thus $\lambda(p, i) - \lambda(q, i - 1) \geq 3 - |d(p, s_i) - d(p, s_{i-1})| - |(d(p, s_{i-1}) - d(q, s_{i-1})|$. Now $|d(p, s_i) - d(p, s_{i-1})| \leq 1$ because $s_{i-1} \in N[s_i]$, and $|(d(p, s_{i-1}) - d(q, s_{i-1})| \leq 1$ because $q \in N[p]$. We thus have $\lambda(q, i - 1) < j$. By induction hypothesis, for any $q \in N[p]$, $time^{r_{j-1}}(q) \geq i - 1$.

Three cases now arise:

- If $time^{r_{j-1}}(p) \geq i$, then we are done.
- If $time^{r_{j-1}}(p) = i - 1$ and $p = s_i$. Then, since s_i may start time i, p can execute the rule R_U in $\rho^{r_{j-1}}$, and thus will have done at last at ρ^{r_j}.
- If $time^{r_{j-1}}(p) = i - 1$ and $p \neq s_i$. Then, let $q \in N(p)$ be such that $d(q, s_i) < d(p, s_i)$. We have $\lambda(q, i) < j$, and thus, by induction hypothesis, $time(q) \geq i$ in $\rho^{r_{j-1}}$. This implies that p can execute the rule R_U in $\rho^{r_{j-1}}$, and thus will have done at last at ρ^{r_j}.

Since $\lambda(p, i) \leq 3T + D - 2$, the lemma follows. □

By Theorems 1 and 2 and Lemmas 32-33, follows.

Theorem 3. *Assumes that Alg_I reaches a terminal configuration in at most T rounds and requires $O(M)$ bits per node. In lazy mode, $Trans(Alg_I)$ reaches a terminal configuration in $O(\min(n^2 B, n^3)) + nT$ moves and at most*

$5D + 3T$ *rounds. Moreover,* $Trans(Alg_I)$ *requires* $O(M + \log(B))$ *bits per node.*

References

[ACD+17] K. Altisen, A. Cournier, S. Devismes, A. Durand, and F. Petit. Self-stabilizing leader election in polynomial steps. *Information and Computation*, 254(3):330 – 366, 2017. doi:10.1016/j.ic.2016.09.002.

[AD17] K. Altisen and S. Devismes. On probabilistic snap-stabilization. *Theor. Comput. Sci.*, 688:49–76, 2017. doi:10.1016/j.tcs.2015.08.001.

[ADDP19] K. Altisen, S. Devismes, S. Dubois, and F. Petit. *Introduction to Distributed Self-Stabilizing Algorithms*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool, 2019. doi:10.2200/S00908ED1V01Y201903DCT015.

[ADG91] A. Arora, S. Dolev, and M. G. Gouda. Maintaining digital clocks in step. *Parallel Processing Letters*, 1:11–18, 1991.

[AKM+93] B. Awerbuch, S. Kutten, Y. Mansour, B. Patt-Shamir, and G. Varghese. Time optimal self-stabilizing synchronization. In *25th Annual Symposium on Theory of Computing, (STOC'93)*, pages 652–661, 1993. doi:10.1145/167088.167256.

[BGJ01] J. Beauquier, M. Gradinariu, and C. Johnen. Cross-over composition - enforcement of fairness under unfair adversary. In *5th International Workshop on Self-Stabilizing Systems, (WSS 2001)*, volume 2194 of *LNCS*, pages 19–34. Springer, 2001. doi:10.1007/3-540-45438-1_2.

[BJLBP22] L. Blin, C. Johnen, G. Le Bouder, and F. Petit. Silent anonymous snap-stabilizing termination detection. In *41st International Symposium on Reliable Distributed Systems, (SRDS'22)*, pages 156–165. IEEE, 2022. doi:10.1109/SRDS55811.2022.00023.

[BP08] C. Boulinier and F. Petit. Self-stabilizing wavelets and rho-hops coordination. In *22nd IEEE International Symposium on Parallel and Distributed Processing, (IPDPS 2008)*, pages 1–8. IEEE, 2008. doi:10.1109/IPDPS.2008.4536130.

[BPV04] C. Boulinier, F. Petit, and V. Villain. When graph theory helps self-stabilization. In *23rd Annual Symposium on Principles of Distributed Computing, (PODC'04)*, pages 150–159, 2004. doi:10.1145/1011767.1011790.

[CDPV02] A. Cournier, A. Datta, F. Petit, and V. Villain. Snap-stabilizing PIF algorithm in arbitrary networks. In *22nd International Conference on Distributed Computing Systems (ICDCS'02)*, pages 199–206. IEEE Computer Society, 2002. doi:10.1109/ICDCS.2002.1022257.

[CDV09] A. Cournier, S. Devismes, and V. Villain. Light enabling snap-stabilization of fundamental protocols. *ACM Transactions on Autonomous and Adaptive Systems*, 4(1):1–27, 2009. doi:10.1145/1462187.1462193.

[CFG92] J.-M. Couvreur, N. Francez, and M. G. Gouda. Asynchronous unison (extended abstract). In *12th International Conference on Distributed Computing Systems, (ICDCS'92)*, pages 486–493, 1992. doi:10.1109/ICDCS.1992.235005.

[CRV19] A. Cournier, S. Rovedakis, and V. Villain. The first fully polynomial stabilizing algorithm for BFS tree construction. *Information and Computation*, 265:26–56, 2019. doi:10.1016/j.ic.2019.01.005.

[DDL19] Ajoy K. Datta, Stéphane Devismes, and Lawrence L. Larmore. A silent self-stabilizing algorithm for the generalized minimal k-dominating set problem. *Theor. Comput. Sci.*, 753:35–63, 2019. doi:10.1016/j.tcs.2018.06.040.

[Dij74] E. W. Dijkstra. Self-stabilization in spite of distributed control. *Communications of the ACM*, 17(11):643–644, 1974. doi:10.1145/361179.361202.

[DIJ22] S. Devismes, D. Ilcinkas, and C. Johnen. Optimized silent self-stabilizing scheme for tree-based constructions. *Algorithmica*, 84(1):85–123, 2022. doi:10.1007/s00453-021-00878-9.

[DIJM23] S. Devismes, D. Ilcinkas, C. Johnen, and F. Mazoit. Making local algorithms efficiently self-stabilizing in arbitrary asynchronous environments. *CoRR*, abs/2307.06635, 2023. arXiv:2307.06635, doi:10.48550/arXiv.2307.06635.

[DJ16] S. Devismes and C. Johnen. Silent self-stabilizing BFS tree algorithms revisited. *Journal on Parallel Distributed Computing*, 97:11–23, 2016. doi:10.1016/j.jpdc.2016.06.003.

[DJ19] S. Devismes and C. Johnen. Self-stabilizing distributed cooperative reset. In *39th International Conference on Distributed Computing Systems, (ICDCS'19)*, pages 379–389, 2019. doi:10.1109/ICDCS.2019.00045.

[DP12] S. Devismes and F. Petit. On efficiency of unison. In *4th Workshop on Theoretical Aspects of Dynamic Distributed Systems, (TADDS'12)*, pages 20–25, 2012. doi:10.1145/2414815.2414820.

[EK21] Y. Emek and E. Keren. A thin self-stabilizing asynchronous unison algorithm with applications to fault tolerant biological networks. In *40nd Symposium on Principles of Distributed Computing, (PODC'21)*, pages 93–102. ACM, 2021. doi:10.1145/3465084.3467922.

[ER90] S. Even and S. Rajsbaum. Unison in distributed networks. In Renato M. Capocelli, editor, *Sequences*, pages 479–487, New York, NY, 1990. Springer New York.

[EW13] Y. Emek and R. Wattenhofer. Stone age distributed computing. In *32nd Symposium on Principles of Distributed Computing, (PODC'13)*, pages 137–146, 2013. doi:10.1145/2484239.2484244.

[FLP85] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985. doi:10.1145/3149.214121.

[GH90] M. G. Gouda and T. Herman. Stabilizing unison. *Inf. Process. Lett.*, 35(4):171–175, 1990.

[GHIJ19] C. Glacet, N. Hanusse, D. Ilcinkas, and C. Johnen. Disconnected components detection and rooted shortest-path tree maintenance in networks. *J. Parallel Distributed Comput.*, 132:299–309, 2019. doi:10.1016/j.jpdc.2019.05.006.

[JADT02] C. Johnen, L. Alima, A. Datta, and S. Tixeuil. Optimal snap-stabilizing neighborhood synchronizer in tree networks. *Parallel Processing Letters*, 12(3-4):327–340, 2002.

[KK13] A. Kravchik and S. Kutten. Time optimal synchronous self stabilizing spanning tree. In *27th International Symposium on Distributed Computing, (DISC'13)*, volume 8205, pages 91–105, 2013. doi:10.1007/978-3-642-41527-2_7.

[Tix06] S. Tixeuil. *Vers l'auto-stabilisation des systèmes à grande échelle*. Habilitation à diriger des recherches, Université Paris Sud - Paris XI, 2006. URL: https://tel.archives-ouvertes.fr/tel-00124848/file/hdr_final.pdf.