Gradual stabilization

Karine Altisen, Stéphane Devismes, Anaïs Durand, Franck Petit

Gradual Stabilization *

Karine Altisen Stéphane Devismes Anaïs Durand
Franck Petit

Abstract

We consider *dynamic distributed systems*, *i.e.*, distributed systems that can suffer from topological changes over the time. Following the superstabilizing approach, we assume here that topological changes are transient events. In this context, we introduce the notion of *gradual stabilization under* (τ, ρ)*-dynamics* (gradual stabilization, for short). A gradually stabilizing algorithm is a self-stabilizing algorithm with the following additional feature: after up to τ *dynamic steps* of a given type ρ occur starting from a legitimate configuration, it first quickly recovers to a configuration from which a specification offering a minimum quality of service is satisfied. It then gradually converges to specifications offering stronger and stronger safety guarantees until reaching a configuration (1) from which its initial (strong) specification is satisfied again, and (2) where it is ready to achieve gradual convergence again in case of up to τ new dynamic steps of type ρ. A gradually stabilizing algorithm being also self-stabilizing, it still recovers within finite time (yet more slowly) after any other finite number of transient faults, including for example more than τ arbitrary dynamic steps or other failure patterns such as memory corruptions. We illustrate this new property by considering three variants of a synchronization problem respectively called *strong*, *weak*, and *partial* unison. We propose a self-stabilizing unison algorithm which achieves gradual stabilization in the sense that after one dynamic step of a certain type BULCC (such a step may include several topological changes) occurs starting from a configuration which is legitimate for the strong unison, it maintains clocks almost synchronized during the convergence to strong unison: it satisfies partial unison immediately after the dynamic step, then converges in at most one round to weak unison, and finally re-stabilizes to strong unison.

keyword: Self-stabilization, synchronization problems, unison, gradual stabilization, superstabilization, safe-convergence

*This study has been partially supported by the ANR projects DESCARTES (ANR-16-CE40-0023) and ESTATE (ANR-16-CE25-0009). A preliminary version of this paper has been published in the proceedings of Euro-Par 2016 [1].

1 Introduction

In 1974, Dijkstra [2] introduced *self-stabilization*, a general paradigm to enable the design of distributed systems tolerating *any* finite number of transient faults.[1] Consider the first configuration after all transient faults cease. This configuration is arbitrary, but no other transient faults will ever occur from this configuration. By abuse of language, this configuration is referred to as *arbitrary initial configuration* of the system in the literature. Then, a self-stabilizing algorithm (provided that faults have not corrupted its code) guarantees that starting from such an arbitrary initial configuration, the system recovers *within finite time*, without any external intervention, to a so-called *legitimate configuration* from which its specification is satisfied. Thus, self-stabilization makes no hypotheses on the nature (*e.g.*, memory corruptions or topological changes) or extent of transient faults that could hit the system, and the system recovers from the effects of those faults in a unified manner. Such versatility comes at a price, *e.g.*, after transient faults cease, there is a finite period of time, called the *stabilization phase*, during which the safety properties of the system are violated. Hence, self-stabilizing algorithms are mainly compared according to their *stabilization time*, the maximum duration of the stabilization phase. For many problems, the stabilization time is significant, *e.g.*, for synchronization problems [3] and more generally for non-static problems [4] (such as token passing or broadcast) the lower bound is $\Omega(\mathcal{D})$ rounds, where $\mathcal{D}$ is the diameter of the network. By definition, the stabilization time is impacted by worst case scenarios. Now, in most cases, transient faults are sparse and their effect may be superficial. Recent research thus focuses on proposing self-stabilizing algorithms that additionally ensure drastically smaller convergence times in favorable cases.

Defining the number of faults hitting a network using some kind of Hamming distance (the minimal number of processes whose state must be changed in order to recover a legitimate configuration), variants of the self-stabilization paradigm have been defined, *e.g.*, a *time-adaptive* self-stabilizing algorithm [5] additionally guarantees a convergence time in $O(k)$ time-units when the initial configuration is at distance at most k from a legitimate configuration.

The property of *locality* consists in avoiding situations in which a small number of transient faults causes the entire system to be involved in a global convergence activity. Locality is, for example, captured by *fault containing* self-stabilizing algorithms [6], which ensure that when few faults hit the system, the faults are both spatially and temporally contained. "Spatially" means that if only few faults occur, those faults cannot be propagated further than a preset radius around the corrupted processes. "Temporally" means quick stabilization when few faults occur.

Some other approaches consist in providing convergence times *tailored by the type of transient faults*. For example, a *superstabilizing algorithm* [7] is self-stabilizing and has two additional properties when transient faults are limited to a single topological change. Indeed, after adding or removing one link or

[1] Transient faults have low frequency and results in perturbing the state of the system.

process in the network, a superstabilizing algorithm recovers fast (typically $O(1)$ rounds), and a safety predicate, called a *passage* predicate, should be satisfied all along the stabilization phase.

Contribution In this paper, we consider distributed systems that can suffer from topological changes over the time, also referred to as *dynamic distributed systems* in [7]. Following the superstabilizing approach, we assume here that topological changes are transient events. In this context, we introduce a specialization of self-stabilization called *gradual stabilization under* (τ, ρ)*-dynamics*. An algorithm is gradually stabilizing under (τ, ρ)-dynamics if it is self-stabilizing and satisfies the the following additional feature. After up to τ *dynamic steps*[2] of type ρ[3] occur starting from a legitimate configuration, a gradually stabilizing algorithm first quickly recovers a configuration from which a specification offering a minimum quality of service is satisfied. It then gradually converges to specifications offering stronger and stronger safety guarantees until reaching a configuration (1) from which its initial (strong) specification is satisfied again, and (2) where it is ready to achieve gradual convergence again in case of up to τ new dynamic steps of type ρ. Of course, the gradual stabilization makes sense only if the convergence to every intermediate weaker specification is fast.

We illustrate this new property by considering three variants of a synchronization problem respectively called *strong*, *weak*, and *partial* unison. In these problems, each process should maintain a local clock. We restrict here our study to periodic clocks, *i.e.*, all local clocks are integer variables whose domain is $\{0, \ldots, \alpha-1\}$, where $\alpha \geq 2$ is called the *period*. Each process should regularly increment its clock modulo α (liveness) while fulfilling some safety requirements. The safety of strong unison imposes that at most two consecutive clock values exist in any configuration of the system. Weak unison only requires that the difference between clocks of every two neighbors is at most one increment. Finally, we define partial unison as a property dedicated to dynamic systems, which only enforces the difference between clocks of neighboring processes present before the dynamic steps to remain at most one increment.

We propose a self-stabilizing strong unison algorithm $\mathcal{SU}$ which works with any period $\alpha \geq 2$ in an anonymous connected network of n processes. $\mathcal{SU}$ assumes the knowledge of two values μ and β, where μ is any value greater than or equal to $\max(2, n)$, α should divide β, and $\beta > \mu^2$. $\mathcal{SU}$ is designed in the locally shared memory model and assume the distributed unfair daemon, the most general daemon of the model. Its stabilization time is at most $n + (\mu + 1)\mathcal{D} + 1$ rounds, where n (resp. $\mathcal{D}$) is the size (resp. diameter) of the network.

We then slightly modify $\mathcal{SU}$ to make it gradually stabilizing under $(1, \mathsf{BULCC})$-dynamics. In particular, the parameter μ should now be greater than or equal to $\max(2, N)$, where N is a bound on the number of processes existing in any reachable configuration. Our gradually stabilizing variant of $\mathcal{SU}$ is called $\mathcal{DSU}$.

[2] *N.b.*, a dynamic step is a step containing topological changes.

[3] Precisely, ρ is a binary predicate over graphs, representing network topologies, such that $\rho(G, G')$ is true if and only if it is possible for the system to switch from topology G to topology G' in a single (dynamic) step.

Due to the slight modifications, the stabilization time of $\mathcal{DSU}$ is increased by one round compared to the one of $\mathcal{SU}$. The condition BULCC restricts the gradual convergence obligation to dynamic steps, called BULCC-dynamic steps, that fulfill the following conditions. A BULCC-dynamic step may contain several topological events, *i.e.*, link and/or process additions and/or removals. However, after such a step, the network should (1) contains at most N processes, (2) stay connected, and (3) if $\alpha > 3$, every process which joins the system should be linked to at least one process already in the system before the dynamic step, unless all of those have left the system. Condition (1) is necessary to have finite periodic clocks in $\mathcal{DSU}$. We show the necessity of condition (2) to obtain our results whatever the period is, while we proved that condition (3) is necessary for our purposes when the period α is fixed to a value greater than 5. Finally, we exhibit pathological cases for periods 4 and 5, in case we do not assume condition (3).

$\mathcal{DSU}$ is gradually stabilizing because after one BULCC-dynamic step from a configuration which is legitimate for the strong unison, it immediately satisfies the specification of partial unison, then converges to the specification of weak unison in at most one round, and finally retrieves, after at most $(\mu + 1)\mathcal{D}_1 + 1$ additional rounds (where $\mathcal{D}_1$ is the diameter of the network after the dynamic step), a configuration (1) from which the specification of strong unison is satisfied, and (2) where it is ready to achieve gradual convergence again in case of another dynamic step.

Notice that $\mathcal{DSU}$ being also self-stabilizing (by definition), it still converges to a legitimate configuration of the strong unison after the system suffers from arbitrary other kinds of transient fault including, for example, several arbitrary dynamic steps. However, in such cases, there is no safety guarantees during the stabilization phase.

Related Work Gradual stabilization is related to two other stronger forms of self-stabilization, namely, *safe-converging self-stabilization* [8] and *superstabilization* [7]. The goal of a safely converging self-stabilizing algorithm is to first quickly (within $O(1)$ rounds is the usual rule) converge from an arbitrary configuration to a *feasible* legitimate configuration, where a minimum quality of service is guaranteed. Once such a feasible legitimate configuration is reached, the system continues to converge to an *optimal* legitimate configuration, where more stringent conditions are required. Hence, the aim of safe-converging self-stabilization is also to ensure a gradual convergence, but only for two specifications. However, such a gradual convergence is stronger than ours as it should be ensured after any step of transient faults,[4] while the gradual convergence of our property applies after dynamic steps only. Safe convergence is especially interesting for self-stabilizing algorithms that compute optimized data structures, *e.g.*, minimal dominating sets [8], approximately minimum weakly connected dominating sets [9], approximately minimum connected dominating sets [10, 11], and minimal (f, g)-alliances [12]. However, to the best of our knowl-

[4]Such transient faults may include topological changes, but not only.

edge, no safe-converging algorithm for non-static problems, such as unison for example, has been proposed until now.

In superstabilization, like in our approach, fast convergence and the passage predicate should be ensured only if the system was in a legitimate configuration before the topological change occurs. In contrast with our approach, superstabilization ensures fast convergence to the original specification. However, this strong property only considers one dynamic step consisting in only one topological event: the addition or removal of one link or process in the network. Again, superstabilization has been especially studied in the context of static problems, *e.g.*, spanning tree construction [7, 13, 14], and coloring [7]. However, notice that there exist few superstabilizing algorithms for non-static problems in particular topologies, *e.g.*, mutual exclusion in rings [15, 16].

We use the general term *unison* to name several close problems also known in the literature as *phase or barrier synchronization* problems. There exist many self-stabilizing algorithms for the strong as well as weak unison problem, *e.g.*, [17, 18, 19, 20, 21, 22, 23, 24]. However, to the best of our knowledge, until now, no self-stabilizing solution for such problems addresses specific convergence properties in case of topological changes (in particular, no superstabilizing ones). Self-stabilizing strong unison was first considered in synchronous anonymous networks. Particular topologies were considered in [20] (rings) and [21] (trees). Gouda and Herman [18] proposed a self-stabilizing algorithm for strong unison working in anonymous synchronous systems of arbitrary connected topology. However, they considered unbounded clocks. A solution working with the same settings, yet implementing bounded clocks, is proposed in [19]. In [24], an asynchronous self-stabilizing strong unison algorithm is proposed for arbitrary connected rooted networks.

Johnen *et al.* investigated asynchronous self-stabilizing weak unison in oriented trees in [22]. The first self-stabilizing asynchronous weak unison for general graphs was proposed by Couvreur *et al.* [25]. However, no complexity analysis was given. Another solution which stabilizes in $O(n)$ rounds has been proposed by Boulinier *et al.* in [23]. Finally, Boulinier proposed in his PhD thesis a parametric solution which generalizes both the solutions of [25] and [23]. In particular, the complexity analysis of this latter algorithm reveals an upper bound in $O(\mathcal{D}.n)$ rounds on the stabilization time of the Couvreur *et al.*' algorithm.

Roadmap The rest of the paper is organized as follows. In the next section, we define the computational model used in this paper. In Section 3, we recall the formal definition of self-stabilization, and introduce the notion of gradual stabilization. The three variants of the unison problem considered in this paper are defined in Section 4. In Section 5, we justify the condition on dynamic steps we assume for our gradually stabilizing algorithm. We present our self-stabilizing strong unison algorithm in Section 6. The gradually stabilizing variant of this latter algorithm is proposed in Section 7. We make concluding remarks in Section 8.

2 Preliminaries

We consider the *locally shared memory model* introduced by Dijkstra [2] enriched with the notion of topological changes. Thereupon, we follow an approach similar to the one used by Dolev in the context of superstabilization [26].

2.1 Processes

We consider distributed systems made of *anonymous* processes. The system *initially contains* $n > 0$ *processes and its topology is connected*, however it may suffer from topological changes over the time. Each process p can directly communicate with a subset $p.\mathcal{N}$ of other processes, called its *neighbors*. In our context, $p.\mathcal{N}$ can vary over time. Communications are assumed to be *bidirectional*, *i.e.*, for any two processes p and q, $q \in p.\mathcal{N} \Leftrightarrow p \in q.\mathcal{N}$ at any time. Communications are carried out by a finite set of locally shared variables at each process: each process can read its own variables and those of its (current) neighbors, but can only write into its own variables. The *state* of a process is the vector of values of its variables. We denote by $\mathcal{S}$ the set of all possible states of a process.

Each process updates its variables according to a *local algorithm*. The collection of all local algorithms (one per process) defines a *distributed algorithm*. In the distributed algorithm $\mathcal{A}$, the local algorithm of p, noted $\mathcal{A}(p)$, consists of a finite set of *actions* of the following form:

$$\langle \text{ label } \rangle :: \langle \text{ guard } \rangle \rightarrow \langle \text{ statement } \rangle$$

The *labels* are used to identify actions in the reasoning. The *guard* of an action is a Boolean predicate involving variables of p and its neighbors. The *statement* is a sequence of assignments on variables of p. If the guard of some action evaluates to true, then the action is said to be *enabled* at p. By extension, p is said to be enabled if at least one action is enabled at p. An action can be executed only if it is enabled. In this case, the execution of the action consists in executing its statement, atomically.

A *configuration* γ_i of the system is a pair (G_i, f_i). $G_i = (V_i, E_i)$ is a simple undirected graph which represents the topology of the network in configuration γ_i, *i.e.*, V_i is the set of processes that are in the system in γ_i and $E_i \subseteq V_i \times V_i$ represents the communication links between (unordered) pairs of distinct processes of V_i in γ_i: $\forall p, q \in V_i$, $\{p, q\} \in E_i \Leftrightarrow p \in \gamma_i(q).\mathcal{N}$ in γ_i. $f_i : V_i \rightarrow \mathcal{S}$ is a function which associates a state to each process of V_i. We denote by $\gamma_i(p)$ the state of process $p \in V_i$ in configuration γ_i. Moreover, $\gamma_i(p).x$ denotes the value of the x-variable at process p in configuration γ_i. We denote by $\mathcal{C}$ the set of all possible configurations.

2.2 Steps

The dynamicity and asynchronism of the system are materialized by the (indeterministic) choices of an adversary, called *daemon*, made at each step of the

execution. To perform a *step* from a configuration γ_i, the daemon can

- activate some processes (of V_i) that are enabled in γ_i — each activated process executes one of its enabled actions according to its own state and that of its neighbors in γ_i, and/or
- modify the topology.

Activation of enabled processes and/or topology modifications are done atomically, leading to a new configuration γ_{i+1}. The set of all possible steps induces a binary relation over configurations noted $\mapsto \subseteq \mathcal{C} \times \mathcal{C}$.

We distinguish two kinds of steps, *i.e.*, $\mapsto$ is partitioned into $\mapsto_s$ and $\mapsto_d$. Relation $\mapsto_s$ define all possible *static steps*, *i.e.*, all steps consisting in activations of enabled processes only. Relation $\mapsto_d$ define all possible *dynamic steps*, *i.e.*, all steps containing topological changes, and possibly some process activations.

2.2.1 Static Steps

Let γ_i be a configuration. Let $Enabled(\gamma_i)$ be the set of enabled processes in γ_i. The daemon can choose to make a static step from γ_i only if $Enabled(\gamma_i) \neq \emptyset$. In this case, it first selects a non-empty subset S of $Enabled(\gamma_i)$. Next, every process $p \in S$ *atomically* executes one of its enabled actions, leading the system to a new configuration, say γ_{i+1}. In this case, $\gamma_i \mapsto_s \gamma_{i+1}$ with, in particular, $G_i = G_{i+1}$.

2.2.2 Dynamic Steps

Let $\gamma_i \mapsto_d \gamma_{i+1}$ be a dynamic step. We have in particular that $G_i \neq G_{i+1}$. Precisely, the step $\gamma_i \mapsto_d \gamma_{i+1}$ contains a finite number of topological events and maybe some process activations (like in static steps). Each topological event is of the following types.

- A process p can *join* the system, *i.e.*, $p \notin V_i \wedge p \in V_{i+1}$. This event is denoted by $join_p$ and triggers the atomic execution of a specific action, called *bootstrap*, which initializes p to a particular state, called *bootstate*, meaning that the output of p is meaningless for now. This bootstrap is executed instantly, without any communication. We denote by New_k the set of processes which are in bootstate in γ_k. When p joins the system in $\gamma_i \mapsto_d \gamma_{i+1}$, we have $p \in New_{i+1}$, but $p \notin New_i$. Moreover, until p executes its very first action, say in step $\gamma_x \mapsto \gamma_{x+1}$, it is still in bootstate. Hence $\forall j \in \{i+1, \ldots, x\}, p \in New_j$, but $p \notin New_{x+1}$.
- A process p can also *leave* the system, *i.e.*, $p \in V_i \wedge p \notin V_{i+1}$.
- Finally, some communication links can *appear* or *disappear* between two different processes.

Several joins, leaves, as well as link appearances and disappearances can be made in the same step $\gamma_i \mapsto_d \gamma_{i+1}$.

We might make assumptions on possible dynamic steps, *i.e.*, restrict the set of possible dynamic steps *w.r.t.* the possible topological changes. To that goal, we will define a binary predicate ρ over graphs, called a *dynamic pattern*. Let $\mapsto_d^\rho = \{(\gamma_i, \gamma_{i+1}) \in \mathcal{C} \times \mathcal{C} : \gamma_i \mapsto_d \gamma_{i+1} \wedge \rho(G_i, G_{i+1})\}$ be the subrelation of $\mapsto_d$ induced by ρ. Every step in $\mapsto_d^\rho$ is called a *ρ-dynamic step*.

2.3 Executions

An *execution* is any sequence of configurations $\gamma_0, \gamma_1, \ldots$ such that G_0 is connected and $\forall i \geq 0$, $\gamma_i \mapsto \gamma_{i+1}$. For sake of simplicity, we note $G_0 = G = (V, E)$, the initial topology of the network; we also note $\mathcal{D}$ the diameter of G and we recall that $|V| = n$. Moreover, we define $\mathcal{E}^\tau$ the set of maximal executions which contain at most τ dynamic steps. Any execution $e \in \mathcal{E}^\tau$ is either infinite, or ends in a so-called *terminal* configuration, where all processes in the system are disabled. The set of all possible maximal executions is therefore equal to $\mathcal{E} = \cup_{\tau \geq 0} \mathcal{E}^\tau$. Notice that $\forall i, j \in \mathbb{N}$, $i \leq j$ implies $\mathcal{E}^i \subseteq \mathcal{E}^j$. Let ρ be a dynamic pattern, we denote by $\mathcal{E}^{\tau,\rho}$ the subset of all executions in $\mathcal{E}^\tau$ where all dynamic steps follow the dynamic pattern ρ, *i.e.*, $\mathcal{E}^{\tau,\rho} = \{(\gamma_i)_{i\geq 0} \in \mathcal{E}^\tau : \forall i > 0, \gamma_{i-1} \mapsto_d \gamma_i \Rightarrow \gamma_{i-1} \mapsto_d^\rho \gamma_i\}$. For any subset of configurations $X \subseteq \mathcal{C}$, we denote by $\mathcal{E}_X^\tau$ (resp. $\mathcal{E}_X^{\tau,\rho}$) the set of all executions in $\mathcal{E}^\tau$ (resp. $\mathcal{E}^{\tau,\rho}$) that start from a configuration of X, *i.e.*, $\mathcal{E}_X^\tau = \{(\gamma_i)_{i\geq 0} \in \mathcal{E}^\tau : \gamma_0 \in X\}$ (resp. $\mathcal{E}_X^{\tau,\rho} = \{(\gamma_i)_{i\geq 0} \in \mathcal{E}^{\tau,\rho} : \gamma_0 \in X\}$).

2.4 Daemon

As previously explained, executions are driven by a daemon. We assume the daemon is *distributed* and *unfair*. In a static step, a distributed daemon must activate at least one enabled process (maybe more). In a dynamic step, a distributed daemon can activate 0, 1, or several enabled processes and decide of the topological changes. An unfair daemon has no fairness constraint, *i.e.*, it might never select a process p during any step unless in the case of a static step from a configuration where p is the only enabled process. Moreover, at each configuration, an unfair daemon freely chooses between making a static or dynamic step, except if no more process is enabled; in this latter case, only a dynamic step containing no process activation can be chosen.

2.5 Functional Specification and Metrics

A distributed algorithm $\mathcal{A}$ is designed to ensure some functional properties called its *specification*. A specification SP is a predicate over $\mathcal{E}$.

We measure the time complexity of our algorithms in terms of *rounds* [27], which expresses the execution time according to the speed of the slowest process. The first round of an execution $e = (\gamma_i)_{i\geq 0}$ is the minimal prefix e' of e such that every enabled process in γ_0 either executes an action or is *neutralized* (defined below). Let γ_j be the last configuration of e', the second round of e is the first round of $e'' = (\gamma_i)_{i\geq j}$, and so forth.

Neutralized means that a process p is enabled in a configuration γ_i but either p is no more in the system in the next configuration γ_{i+1}, or p is not enabled in γ_{i+1} but does not execute any action during the step $\gamma_i \mapsto \gamma_{i+1}$.

3 Stabilization

3.1 Self-stabilization

Below we recall the definitions of some notions classically used in self-stabilization. Notice that all these notions are defined by only considering executions free of topological changes, yet starting from an arbitrary configuration. Indeed, self-stabilization considers the system immediately after the transient faults cease. So, the system is initially observed from an arbitrary configuration reached due to the occurrence of transient faults (including some topological changes maybe), but from which no faults (in particular, no topological changes) will ever occur.

Let $\mathcal{A}$ be a distributed algorithm. Let $X, Y \subseteq \mathcal{C}$ be two subsets of configurations. X is *closed* under $\mathcal{A}$ *if and only if* $\forall \gamma, \gamma' \in \mathcal{C}, (\gamma \in X \wedge \gamma \mapsto_s \gamma') \Rightarrow \gamma' \in X$. Y *converges to* X under $\mathcal{A}$ *if and only if* $\forall e \in \mathcal{E}_Y^0, \exists \gamma \in e$ such that $\gamma \in X$. $\mathcal{A}$ *stabilizes* from Y to a specification SP by X *if and only if*

- X is closed under $\mathcal{A}$,
- Y converges to X under $\mathcal{A}$,
- and $\forall e \in \mathcal{E}_X^0$, $SP(e)$.

Moreover, the *convergence time (of $\mathcal{A}$) in steps (resp. rounds) from Y to X* is the maximal number of steps (or rounds, resp.) to reach a configuration of X in every execution of $\mathcal{E}_Y^0$.

Self-stabilization has been defined by Dijkstra in 1974 [2] as follows: a distributed algorithm $\mathcal{A}$ is *self-stabilizing* for a specification SP *if and only if* $\exists \mathcal{L} \subseteq \mathcal{C}$, such that $\mathcal{A}$ stabilizes from $\mathcal{C}$ to SP by $\mathcal{L}$.

$\mathcal{L}$ (resp. $\mathcal{C} \setminus \mathcal{L}$) is then said to be a set of *legitimate configurations* (resp. *illegitimate configurations*) *w.r.t.* SP. The *stabilization time* of $\mathcal{A}$ is then the convergence time from $\mathcal{C}$ to $\mathcal{L}$.

3.2 Gradual Stabilization under (τ, ρ)-Dynamics

Below, we introduce a specialization of self-stabilization called *gradual stabilization under (τ, ρ)-dynamics*. The overall idea behind this concept is to design self-stabilizing algorithms that ensure additional properties (stronger than "simple" eventual convergence) when the system suffer from topological changes. Initially observe the system from a legitimate configuration, and assume that up to τ ρ-dynamic steps occur. The very first configuration after those steps may be illegitimate, but this configuration is usually far from being arbitrary. In that situation, the goal of gradual stabilization is to first quickly recover a configuration from which a weaker specification offering a minimum quality of

service is satisfied and then make the system gradually re-stabilizes to stronger and stronger specifications, until fully recovering its initial (strong) specification. Of course, the gradual stabilization makes sense only if the convergence to every intermediate weaker specification is fast and each of those weak specifications offers a useful interest.

3.2.1 Definition

Let $\tau \geq 0$. For every execution $e = (\gamma_i)_{i \geq 0} \in \mathcal{E}^\tau$ (*i.e.*, e contains at most τ dynamic steps), we note $\gamma_{fst(e)}$ the first configuration of e after the last dynamic step. Formally, $fst(e) = \min\{i : (\gamma_j)_{j \geq i} \in \mathcal{E}^0\}$. For any subset E of $\mathcal{E}^\tau$, let $FC(E) = \{\gamma_{fst(e)} : e \in E\}$ be the set of all configurations that can be reached after the last dynamic step in executions of E.

Let $SP_1, SP_2, \ldots, SP_k$, be an ordered sequence of specifications. Let B_1, B_2, $\ldots$, B_k be (asymptotic) complexity bounds such that $B_1 \leq B_2 \leq \cdots \leq B_k$. Let ρ be a dynamic pattern.

A distributed algorithm $\mathcal{A}$ is *gradually stabilizing under* (τ, ρ)*-dynamics* for $(SP_1 \bullet B_1, SP_2 \bullet B_2, \ldots, SP_k \bullet B_k)$ *if and only if* $\exists \mathcal{L}_1, \ldots, \mathcal{L}_k \subseteq \mathcal{C}$ such that

1. $\mathcal{A}$ stabilizes from $\mathcal{C}$ to SP_k by $\mathcal{L}_k$.
2. $\forall i \in \{1, \ldots, k\}$,
   - $\mathcal{A}$ stabilizes from $FC(\mathcal{E}^{\tau,\rho}_{\mathcal{L}_k})$ to SP_i by $\mathcal{L}_i$, and
   - the convergence time in rounds from $FC(\mathcal{E}^{\tau,\rho}_{\mathcal{L}_k})$ to $\mathcal{L}_i$ is bounded by B_i.

The first point ensures that a gradually stabilizing algorithm is still self-stabilizing for its strongest specification. Hence, its performances can be also evaluated at the light of its stabilization time. Indeed, it captures the maximal convergence time of the gradually stabilizing algorithm after the system suffers from an arbitrary finite number of transient faults (those faults may include an unbounded number of arbitrary dynamic steps, for example).

The second point means that after up to τ ρ-dynamic steps from a configuration that is legitimate *w.r.t.* the strongest specification SP_k, the algorithm *gradually converges* to each specification SP_i with $i \in \{1, \ldots, k\}$ in at most B_i rounds.

Note that B_k captures a complexity similar to the *fault gap* in fault-containing algorithms [6]: assume a period $P1$ of up to τ ρ-dynamic steps starting from a legitimate configuration of $\mathcal{L}_k$; B_k represents the necessary fault-free interval after $P1$ and before the next period $P2$ of at most τ ρ-dynamic steps so that the system converges to a legitimate configuration of $\mathcal{L}_k$ and so becomes ready again to achieve gradual convergence after $P2$.

3.2.2 Related Properties

Gradual stabilization is related to two other stronger forms of self-stabilization: *safe-converging self-stabilization* [8] and *superstabilization* [7].

The goal of a *safely converging self-stabilizing algorithm* is to first quickly (within $O(1)$ rounds is the usual rule) converge from any arbitrary configuration to a *feasible* legitimate configuration, where a minimum quality of service is guaranteed. Once such a feasible legitimate configuration is reached, the system continues to converge to an *optimal* legitimate configuration, where more stringent conditions are required. Hence, the aim of safe-converging self-stabilization is also to ensure a gradual convergence, but for two specifications. However, this kind of gradual convergence should be ensured after any step of transient faults (such transient faults can include topological changes, but not only), while the gradual convergence of our property applies after dynamic steps only.

A *superstabilizing algorithm* is self-stabilizing and has two additional properties. In presence of a single topological change (adding or removing one link or process in the network), it recovers fast (typically $O(1)$), and a safety predicate, called a *passage* predicate, should be satisfied all along the stabilization phase. Like in our approach, fast convergence, captured by the notion of *superstabilization time*, and the passage predicate should be ensured only if the system was in a legitimate configuration before the topological change occurs. In contrast with our approach, superstabilization consists in only *one* dynamic step satisfying a very restrictive dynamic pattern, noted here ρ_1: only one topological event, *i.e.*, the addition or removal of one link or process in the network. A superstabilizing algorithm for a specification SP_1 can be seen as an algorithm which is gradually stabilizing under $(1, \rho_1)$-dynamics for $(SP_0 \bullet 0, SP_1 \bullet f)$ where SP_0 is the passage predicate and f is the superstabilization time.

4 Unison

We consider three close synchronization problems included here under the general term of *unison*. In these problems, each process should maintain a local *clock*. We restrict here our study to periodic clocks: α, called the *period* of the clocks, should be greater than or equal to 2. The aim is to make all local clocks regularly incrementing (modulo α) in a finite set of integer values $\{0, \ldots, \alpha - 1\}$ while fulfilling some safety requirements.

All these problems require the same liveness property which means that whenever a clock has a value in $\{0, \ldots, \alpha - 1\}$, it should eventually increment.

Definition 1 (Liveness of the Unison) *An execution $e = (\gamma_i)_{i \geq 0}$ satisfies the liveness property $LIVE$* if and only if $\forall \gamma_i \in e, \forall p \in V_i, \forall x \in \{0, \ldots, \alpha - 1\}$, $\gamma_i(p).clock = x \Rightarrow \exists j > i, (\forall k \in \{i+1, \ldots, j-1\}, p \in V_k \wedge \gamma_k(p).clock = x) \wedge (p \in V_j \wedge \gamma_j(p).clock = (x + 1) \mod \alpha)$.

The three versions of unison we consider are respectively named *strong*, *weak*, and *partial* unison, and differ by their safety property. *Strong unison* is also known as the *phase* or *barrier* synchronization problem [28, 29]. The *weak unison* appeared first in [25] under the name of *asynchronous unison*. We define the *partial unison* as a straightforward variant of the weak unison suited for dynamic systems.

Definition 2 (Safety of the Partial Unison) *An execution $e = (\gamma_i)_{i\geq 0}$ satisfies the safety property $SAFE_{\mathsf{PU}}$* if and only if *$\forall \gamma_i \in e$, the following conditions holds*

- $\forall p \in V_i \setminus New_i, \gamma_i(p).clock \in \{0, \ldots, \alpha - 1\}$ *and*
- $\forall p \in V_i \setminus New_i, \forall q \in \gamma_i(p).\mathcal{N} \setminus New_i$, $\gamma_i(p).clock \in \{\gamma_i(q).clock, (\gamma_i(q).clock + 1) \mod \alpha, (\gamma_i(q).clock - 1) \mod \alpha\}$, *meaning that the clocks of any two neighbors which are not in bootstate*[5] *differ from at most one increment (modulo α).*

Definition 3 (Safety of the Weak Unison) *An execution $e = (\gamma_i)_{i\geq 0}$ satisfies the safety property $SAFE_{\mathsf{WU}}$* if and only if

- *$\forall \gamma_i \in e$, $New_i = \emptyset$, meaning that no process is in bootstate and*
- *$SAFE_{\mathsf{PU}}(e)$ holds.*

In the next definition, we use the following notation: for every configuration γ_i, let $CV(\gamma_i) = \{\gamma_i(p).clock : p \in V_i\}$ be the set of clock values present in configuration γ_i.

Definition 4 (Safety of the Strong Unison) *An execution $e = (\gamma_i)_{i\geq 0}$ satisfies the safety property $SAFE_{\mathsf{SU}}$* if and only if *$\forall \gamma_i \in e$, the following conditions holds*

- *$New_i = \emptyset$, meaning that no process is in bootstate,*
- *$\forall p \in V_i, \gamma_i(p).clock \in \{0, \ldots, \alpha - 1\}$, and*
- *$|CV(\gamma_i)| \leq 2 \wedge (CV(\gamma_i) = \{x, y\} \Rightarrow x = (y + 1) \mod \alpha \ \vee \ y = (x + 1) \mod \alpha)$, meaning that there exists at most two different clock values, and if so, these two values are consecutive (modulo α).*

Specification 1 (Partial Unison) *An execution e satisfies the specification SP_{PU} of the* partial unison if and only if $LIVE(e) \wedge SAFE_{\mathsf{PU}}(e)$.

Specification 2 (Weak Unison) *An execution e satisfies the specification SP_{WU} of the* weak unison if and only if $LIVE(e) \wedge SAFE_{\mathsf{WU}}(e)$.

Specification 3 (Strong Unison) *An execution e satisfies the specification SP_{SU} of the* strong unison if and only if $LIVE(e) \wedge SAFE_{\mathsf{SU}}(e)$.

The property below sum up the straightforward relationship between the three variants of unison we consider here.

Property 1 *For every execution e, we have $SP_{\mathsf{SU}}(e) \Rightarrow SP_{\mathsf{WU}}(e) \Rightarrow SP_{\mathsf{PU}}(e)$.*

[5] Recall that while a process is in bootstate, it has not taken any step and so its output, here its clock value, is meaningless.

5 Conditions on the Dynamic Pattern

In Section 6, we provide Algorithm $\mathcal{DSU}$ which is gradually stabilizing under $(1, \mathsf{BULCC})$-dynamics for $(SP_{\mathsf{PU}} \bullet 0, SP_{\mathsf{WU}} \bullet 1, SP_{\mathsf{SU}} \bullet B)$ (where B is a given complexity bound), starting from any arbitrary anonymous (initially connected) network and assuming the distributed unfair daemon. The dynamic pattern BULCC (defined on page 33) requires, in particular, that graphs remain *Connected* (*i.e.*, the dynamic pattern C below) and, if the period α of unison is greater than 3, then the condition *Under Local Control* (*i.e.*, the dynamic pattern ULC below) should hold:

- $\mathsf{C}(G_i, G_j) \equiv$ if graph G_i is connected, then graph G_j is connected.

  Notice that as the initial topology of the system is assumed to be connected, the topology is always connected along any execution of $\mathcal{E}^{1,\mathsf{C}}$.

- $\mathsf{ULC}(G_i, G_j) \equiv$ if $V_i \cap V_j \neq \emptyset$ and G_i is connected, then $V_i \cap V_j$ is a dominating set of G_j.[6]

  ULC permits to prevent a notable desynchronization of clocks. Namely, if not all processes leave the system during a dynamic step $\gamma \mapsto_d \gamma'$ from an initially connected topology, then every process that joins the system during that dynamic step is required to be "under the control of" (that is, linked to) at least one process which exists in both γ and γ'.

We now study the necessity of conditions C and ULC. We first show that the assumption C on dynamic steps is necessary whatever the value of the period α is (Theorem 1). We then show that the dynamic pattern ULC is necessary for any period $\alpha > 5$ (Theorem 2), while our algorithm shows that ULC is not necessary for each period $\alpha < 4$. For remaining cases (periods 4 and 5), our answer is partial, as we show that there are pathological cases among possible dynamic steps which satisfy C but not ULC (Theorem 4 and Corollary 1) that make any algorithm fails to solve our problem. In particular, for the case $\alpha = 5$, we exhibit an important class of such pathological dynamic steps (Theorem 3).

General Proof Context To prove the above results, we assume from now on the existence of a deterministic algorithm $\mathcal{A}$ which is gradually stabilizing under $(1,\rho)$-dynamics for $(SP_{\mathsf{PU}} \bullet 0, SP_{\mathsf{WU}} \bullet 1, SP_{\mathsf{SU}} \bullet B)$ starting from any arbitrary anonymous (initially connected) network and assuming the distributed unfair daemon, where ρ is a given dynamic pattern and B is any (asymptotic) strictly positive complexity bound. Hence, our proofs consist in showing properties that ρ should satisfy (*w.r.t.* dynamic patterns C and ULC) in order to prevent Algorithm $\mathcal{A}$ from failing. In the sequel, we also denote by $\mathcal{L}^{\mathcal{A}}_{\mathsf{SU}}$ the legitimate configurations of $\mathcal{A}$ *w.r.t.* specification SP_{SU}.

[6]Recall that a *dominating set* of the graph $G = (V, E)$ is any subset D of V such that every node not in D is adjacent to at least one member of D.

5.1 Connectivity

Theorem 1 *For every graph G and G', we have $\rho(G, G') \Rightarrow \mathcal{C}(G, G')$.*

Proof. Assume, by the contradiction, that there exists two graphs G_i and G_j such that $\rho(G_i, G_j)$, G_i is connected, and G_j is disconnected. Then, there is an execution $e = (\gamma_i)_{i\geq 0} \in \mathcal{E}^{(1,\rho)}_{\mathcal{L}^{\mathcal{A}}_{\mathsf{SU}}}$ such that $G_0 = G_i$ and $G_{fst(e)} = G_j$. Let A and B be two connected components of $G_{fst(e)}$. By definition, there exists $j \geq fst(e)$ such that $\gamma_j \in \mathcal{L}^{\mathcal{A}}_{\mathsf{SU}}$ and A and B are defined in all configurations $(\gamma_i)_{i\geq j}$. From γ_j, all processes regularly increment their clocks in both A and B by the liveness property of strong unison. Now, as no process of B is linked to any process of A, the behavior of processes in B has no impact on processes in A and vice versa. So, liveness implies, in particular, that there always exists enabled processes in A. Consequently, there exists a possible execution of $\mathcal{E}^{(1,\rho)}_{\mathcal{L}^{\mathcal{A}}_{\mathsf{SU}}}$ prefixed by $\gamma_0 \dots \gamma_j$ where the distributed unfair daemon only selects processes in A from γ_j, hence violating the liveness property of strong unison, a contradiction. □

5.2 Under Local Control

5.2.1 Technical Results

The following property states that, whenever $\alpha > 3$, once a legitimate configuration of the strong unison is reached, the system necessarily goes through a configuration where all clocks have the same value between any two increments at the same process.

Property 2 *Assume $\alpha > 3$. For every $(\gamma_i)_{i\geq 0} \in \mathcal{E}^{0}_{\mathcal{L}^{\mathcal{A}}_{\mathsf{SU}}}$, for every process p, for every $k \in \{0, ..., \alpha - 1\}$, for every $i \geq 0$, if p increments its clock from k to $(k + 1) \mod \alpha$ in $\gamma_i \mapsto_s \gamma_{i+1}$ and $\exists j > i + 1$ such that $\gamma_j(p).clock = (k + 2) \mod \alpha$, then there exists $x \in \{i + 1, ..., j - 1\}$, such that all clocks have value $(k + 1) \mod \alpha$ in γ_x.*

Proof. Let $(\gamma_i)_{i\geq 0} \in \mathcal{E}^{0}_{\mathcal{L}^{\mathcal{A}}_{\mathsf{SU}}}$ and p be a process. Let $k \in \{0, ..., \alpha - 1\}$ and $i \geq 0$ such that p increments its clock from k to $(k + 1) \mod \alpha$ in $\gamma_i \mapsto_s \gamma_{i+1}$ and $\exists j > i + 1$ such that $\gamma_j(p).clock = (k + 2) \mod \alpha$.

Assume, by the contradiction, that there is a process q such that $\gamma_i(q).clock = (k - 1) \mod \alpha$. As the daemon is distributed and unfair, there is a possible static step from γ_i where p moves, but not q leading to a configuration where $q.clock = (k - 1) \mod \alpha$ and $p.clock = (k + 1) \mod \alpha$. This configuration violates the safety of SP_{SU}. Hence, there exists an execution of $\mathcal{E}^{0}_{\mathcal{L}^{\mathcal{A}}_{\mathsf{SU}}}$ which does not satisfy SP_{SU}, a contradiction.

Hence, $\forall q \in V, \gamma_i(q).clock \in \{k, (k + 1) \mod \alpha\}$, by the safety of SP_{SU}. Similarly to the previous case, while there are processes whose clock value is k, no process (in particular p) can increment its clock from $(k + 1) \mod \alpha$ to $(k + 2) \mod \alpha$. Hence, between γ_{i+1} (included) and γ_{j-1} (included), there

exists a configuration where all processes have clock value $(k+1) \mod \alpha$, since $\gamma_j(p).clock = (k+2) \mod \alpha$. □

Since $\mathcal{A}$ is gradually stabilizing under (1,ρ)-dynamics for $(SP_{\mathsf{PU}} \bullet 0, SP_{\mathsf{WU}} \bullet 1, SP_{\mathsf{SU}} \bullet B)$, follows.

Remark 1 *Every execution in $\mathcal{E}^{1,\rho}$ is infinite.*

Lemma 1 *Let $\gamma_i \mapsto_d^\rho \gamma_{i+1}$ be a ρ-dynamic step such that $\gamma_i \in \mathcal{L}_{\mathsf{SU}}^{\mathcal{A}}$ and G_i is connected. For every process $p \in New_{i+1}$, p is enabled in γ_{i+1} and if p moves, then in the next configuration, p is not in bootstate and $p.clock \in \{0, \ldots, \alpha-1\}$.*

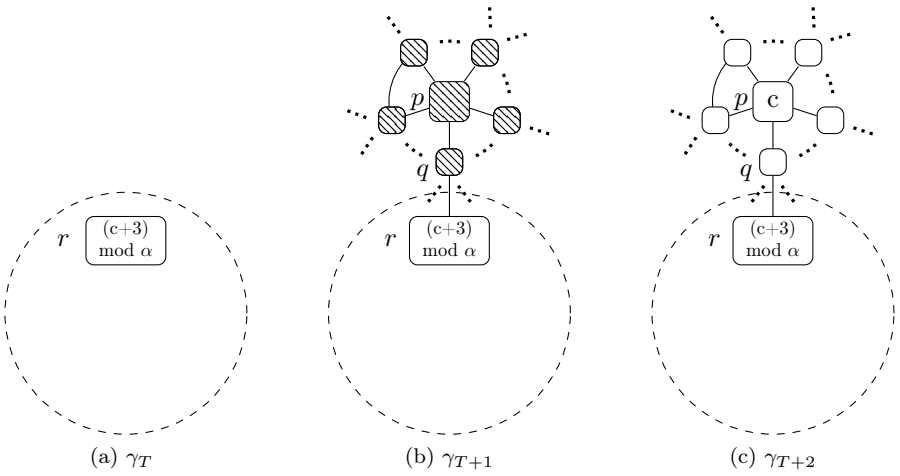
Proof. As $\gamma_i \in \mathcal{L}_{\mathsf{SU}}^{\mathcal{A}}$ and G_i is connected, there is an execution $\mathcal{E}^{1,\rho}_{\mathcal{L}_{\mathsf{SU}}^{\mathcal{A}}}$ prefixed by $\gamma_i\gamma_{i+1}$. Moreover, there are enabled processes in γ_{i+1}, by Remark 1 and the fact that no more dynamic step occurs from γ_{i+1}. Assume that the daemon makes a synchronous static step from γ_{i+1}. The step $\gamma_{i+1} \mapsto_s \gamma_{i+2}$ actually corresponds to a complete round, by definition. So, the execution suffix from γ_{i+2} should satisfy the specification of the weak unison (any execution of $\mathcal{E}^{1,\rho}_{\mathcal{L}_{\mathsf{SU}}^{\mathcal{A}}}$ prefixed by $\gamma_i\gamma_{i+1}$ should converge in one round from $\gamma_{fst(e)} = \gamma_{i+1}$ to a configuration that is legitimate *w.r.t.* SP_{WU}). Now, if, by the contradiction, p is disabled in γ_{i+1}, or p is not in bootstate in γ_{i+2}, or $\gamma_{i+2}(p).clock \notin \{0, \ldots, \alpha-1\}$, then the safety of the weak unison is violated in γ_{i+2}, a contradiction. □

Lemma 2 *Let $c \in \{0, ..., \alpha-1\}$, G be a connected graph of at least two nodes, and r_1 and r_2 be two nodes of G. If $\alpha > 3$, then there exists an execution $e \in \mathcal{E}^0_{\mathcal{L}_{\mathsf{SU}}^{\mathcal{A}}}$ on the graph G which contains a configuration γ_T where r_1 and r_2 have two different clock values, one being $c \mod \alpha$ and the other $(c+1) \mod \alpha$.*

Proof. Consider an execution e' in $\mathcal{E}^0_{\mathcal{L}_{\mathsf{SU}}^{\mathcal{A}}}$ on the graph G. The specification of the strong unison is satisfied in e' and by liveness and Property 2, there is a configuration γ_S in e' where every clock equals $c \mod \alpha$. By liveness again, from γ_S, eventually there is a step where either r_1, or r_2, or both increments to $(c+1) \mod \alpha$. Consider the first step $\gamma_{z-1} \mapsto_s \gamma_z$ after γ_S, where either r_1, or r_2, or both increments to $(c+1) \mod \alpha$. In the two first cases, let $\gamma_T = \gamma_z$ and $e = e'$. For the last case, consider an execution e'' of $\mathcal{E}^0_{\mathcal{L}_{\mathsf{SU}}^{\mathcal{A}}}$ with the prefix $\gamma_0 \ldots \gamma_{z-1}$ common to e', but only r_1 moves in the step from γ_{z-1}. Let γ_T be the configuration reached by this latter step and $e = e''$. In either cases, r_1 and r_2 have two different clock values in γ_T, one being $c \mod \alpha$ and the other $(c+1) \mod \alpha$. □

Lemma 3 *Let G be any connected graph. There exists $\gamma_i \in \mathcal{L}_{\mathsf{SU}}^{\mathcal{A}}$ such that $G_i = G$.*

Proof. $\mathcal{A}$ being designed for arbitrary initially connected networks, there exists at least one execution $e = (\gamma_i)_{i \geq 0} \in \mathcal{E}^0$, where $G_i = G$, $\forall i \geq 0$. By hypothesis, at least one configuration of e belongs to $\mathcal{L}_{\mathsf{SU}}^{\mathcal{A}}$. □



(a) γ_T (b) γ_{T+1} (c) γ_{T+2}

Figure 1: Execution e'' in the proof of Theorem 2. The hachured nodes are in bootstate. The value inside the node is the value of its clock. If there is no value, its clock value is meaningless.

5.2.2 Main Results

Theorem 2 *If $\alpha > 5$, then for every graphs G and G', we have $\rho(G, G') \Rightarrow \mathsf{C}(G, G') \wedge \mathsf{ULC}(G, G')$.*

Proof. We illustrate the following proof with Figure 1. Assume $\alpha > 5$ and let G_{x-1} and G_x be two graphs such that $\rho(G_{x-1}, G_x)$. By Theorem 1, $\mathsf{C}(G_{x-1}, G_x)$ holds. So, assume, by the contradiction, that $\neg\mathsf{ULC}(G_{x-1}, G_x)$. Then, $\mathsf{C}(G_{x-1}, G_x) \wedge \neg\mathsf{ULC}(G_{x-1}, G_x)$ implies, in particular, that both G_{x-1} and G_x are connected.

By Lemma 3, there exists a configuration $\gamma_{x-1} \in \mathcal{L}^{\mathcal{A}}_{\mathsf{SU}}$, whose topology is G_{x-1}. Consider now the configuration γ_x of topology G_x, such that $\gamma_{x-1} \mapsto^{\rho}_{d} \gamma_x$ is a ρ-dynamic step that contains no process activation. Then, since G_x is connected and $V_{x-1} \cap V_x \neq \emptyset$ is not a dominating set, we have: $\exists p \in V_x \setminus V_{x-1}$ such that (1) $\forall v \in \gamma_x(p).\mathcal{N}$, $v \in V_x \setminus V_{x-1}$ and (2) there is a process $q \in \gamma_x(p).\mathcal{N}$ which has at least one neighbor in $V_{x-1} \cap V_x$, say r. Moreover, p and its neighbors (in particular q) are in bootstate in γ_x. So, by Lemma 1, they all are enabled in γ_x and if they move, they will be no more in bootstate and their clock value will belong to $\{0, \ldots, 4\}$ in the configuration that follows γ_x. Let c be the clock value of p in the next configuration, if p moves.

By the liveness property of the strong unison, there exists an execution e in $\mathcal{E}^{0}_{\mathcal{L}^{\mathcal{A}}_{\mathsf{SU}}}$ on the graph G_{x-1} (*n.b.*, G_{x-1} is connected) which contains a configuration γ_T where r has clock value $(c+3) \mod \alpha$, see Figure 1a.

Consider now another execution $e' \in \mathcal{E}^{1,\rho}_{\mathcal{L}^{\mathcal{A}}_{\mathsf{SU}}}$ having a prefix common to e until γ_T. Assume that the unfair daemon introduces a ρ-dynamic step (it is possible

since there was no dynamic step until now). Since $G_T = G_{x-1}$, the daemon can choose a step $\gamma_T \mapsto_d^{\rho} \gamma_{T+1}$, where no process moves and $G_{T+1} = G_x$. Now, $\forall v \in V_{T+1} \setminus V_T$, $\gamma_{T+1}(v) = \gamma_x(v)$, so again, in γ_{T+1}, p and all its neighbors (in particular q) are in bootstate and enabled. Moreover, if they move, they will be not in bootstate and their clock value will belong to $\{0, \ldots, 4\}$ in γ_{T+2}, by Lemma 1. Moreover, p is in the same situation as in γ_x, so if it moves, its clock is equal to c in γ_{T+2}. Then, r is still a neighbor of q which is still not in bootstate and still with clock value $(c+3) \mod 5$, see Figure 1b. By definition, since strong unison is satisfied in γ_T (by assumption), the partial unison necessarily holds all along the suffix of e' starting at γ_{T+1}. Assume that the daemon exactly selects p and its neighbors in the next static step $\gamma_{T+1} \mapsto_s \gamma_{T+2}$. In γ_{T+2} (Figure 1c), r is still not in bootstate and its clock is still equal to $(c+3) \mod 5$, since it did not move. Moreover, p is no more in bootstate and its clock equals c. Now, in γ_{T+2}, q is no more in bootstate and its clock value belongs to $\{0, \ldots, 4\}$. That clock value should differ of at most one increment $(\mod 5)$ from the clocks of p and r since partial unison holds in γ_{T+1} and all subsequent configurations. If the clock of q equals:

- c or $(c+1) \mod \alpha$, the difference between the clocks of q and r is at least 2 increments $(\mod \alpha)$,
- $(c+2) \mod \alpha$, $(c+3) \mod \alpha$, $(c+4) \mod \alpha$, the difference between the clocks of q and p is at least 2 increments $(\mod \alpha)$,
- any value in $\{0, ..., \alpha-1\} \setminus \{c, (c+1) \mod \alpha, (c+2) \mod \alpha, (c+3) \mod \alpha, (c+4) \mod \alpha\}$, the difference between the clocks of q and r is at least 2 increments $(\mod \alpha)$.

Hence, the safety of partial unison is necessarily violated in the configuration γ_{T+2} of e', a contradiction. □

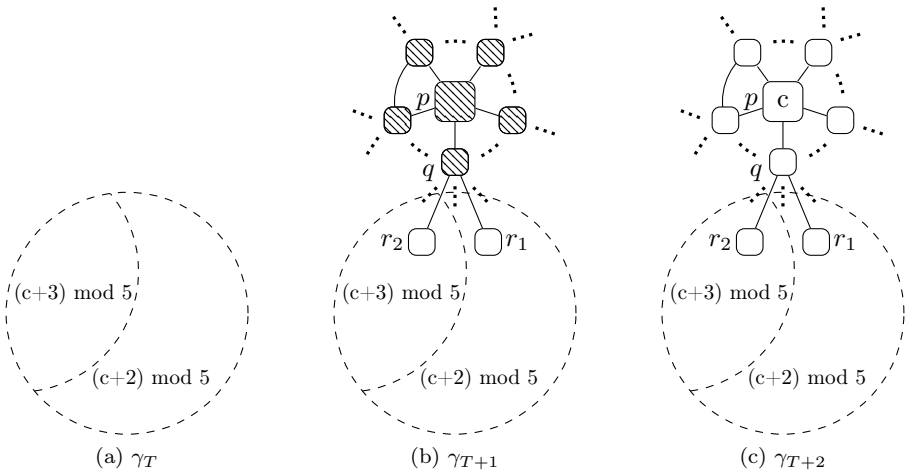
We now focus on dynamic patterns for which C is true but ULC is false and that cannot be included into ρ, unless the specification of $\mathcal{A}$ is violated. Such a pattern is defined below and will be used for the case $\alpha = 5$.

Let ζ be the dynamic pattern such that for every two graphs G_i and G_j, $\zeta(G_i, G_j)$ *if and only if* the following conditions hold:

- both G_i and G_j are connected,
- $|V_i \cap V_j| \geq 2$, and
- $\exists p \in V_j \setminus V_i$ such that $\gamma_j(p).\mathcal{N} \cap V_i = \emptyset$ and $\exists q \in \gamma_j(p).\mathcal{N}, |\gamma_j(q).\mathcal{N} \cap V_i| \geq 2$.

Theorem 3 *If $\alpha = 5$, then for every graphs G and G', we have $\zeta(G, G') \Rightarrow \neg\rho(G, G')$.*

Proof. We illustrate the following proof with Figure 2. Assume, by the contradiction, that $\alpha = 5$ but there exists two graphs G_{x-1} and G_x such that $\zeta(G_{x-1}, G_x)$ and $\rho(G_{x-1}, G_x)$.



(a) γ_T (b) γ_{T+1} (c) γ_{T+2}

Figure 2: Execution e' in the proof of Theorem 3. The hachured nodes are in bootstate. The value inside the node is the value of its clock. If there is no value, its clock value is meaningless.

By Lemma 3, there exists a configuration $\gamma_{x-1} \in \mathcal{L}^{\mathcal{A}}_{\mathsf{SU}}$ (*n.b.*, G_{x-1} is connected, by definition). Consider now the configuration γ_x of topology G_x such that $\gamma_{x-1} \mapsto^{\rho}_{d} \gamma_x$, $\gamma_{x-1} \mapsto^{\zeta}_{d} \gamma_x$, and no process is activated between γ_{x-1} and γ_x.

Let p and q be two nodes such that (1) $p \in V_x \setminus V_{x-1}$, (2) $q \in V_x \setminus V_{x-1}$ and $q \in \gamma_x(p).\mathcal{N}$, (3) $\forall v \in \gamma_x(p).\mathcal{N}$, $v \in V_x \setminus V_{x-1}$, (4) q has at least two neighbors r_1 and r_2 belonging to $V_x \cap V_{x-1}$ (by definition of ζ, p, q, r_1, and r_2 necessarily exist). Then, p and its neighbors (in particular q) are in bootstate in γ_x. So, by Lemma 1, they all are enabled in γ_x and if they move, they will be not in bootstate and their clock values will belong to $\{0, \ldots, 4\}$ in the configuration that follows γ_x. Let c be the clock value of p in the next configuration, if p moves.

By the liveness of the strong unison and Lemma 2, there exists an execution e in $\mathcal{E}^{0}_{\mathcal{L}^{\mathcal{A}}_{\mathsf{SU}}}$ on the graph G_{x-1} which contains a configuration γ_T where r_1 and r_2 are not in bootstate and have two different clock values, one being $(c+2) \mod 5$ and the other $(c+3) \mod 5$. Without the loss of generality, assume that $\gamma_T(r_1).clock = (c+2) \mod 5$ and $\gamma_T(r_2).clock = (c+3) \mod 5$, see Figure 2a.

Consider now another execution $e' \in \mathcal{E}^{1,\rho}_{\mathcal{L}^{\mathcal{A}}_{\mathsf{SU}}}$ having a prefix common to e until γ_T. Assume that the unfair daemon introduces a ρ-dynamic step (it is possible since there was no dynamic step until now). Since $G_T = G_{x-1}$, the daemon can choose a step $\gamma_T \mapsto^{\rho}_{d} \gamma_{T+1}$, where no process moves and $G_{T+1} = G_x$. Now, $\forall v \in V_{T+1} \setminus V_T$, $\gamma_{T+1}(v) = \gamma_x(v)$, so again, in γ_{T+1}, p and all its neighbors (in particular q) are in bootstate and enabled. Moreover, if they move, they will not be in bootstate and their clock values will belong to $\{0, \ldots, 4\}$ in γ_{T+2}, by

Lemma 1. Moreover, p is in the same situation as in γ_x, so if it moves, its clock is equal to c in γ_{T+2}. Then, r_1 and r_2 are both neighbors of q which are still not in bootstate and still with clock values $(c+2) \mod 5$ and $(c+3) \mod 5$, see Figure 2b. By definition, since strong unison is satisfied in γ_T (by assumption), the partial unison necessarily holds all along the suffix of e' starting at γ_{T+1}. Assume that the daemon exactly selects p and its neighbors in the next static step $\gamma_{T+1} \mapsto_s \gamma_{T+2}$. In γ_{T+2} (Figure 2c), r_1 and r_2 are still not in bootstate and their clocks are still respectively equal to $(c+2) \mod 5$ and $(c+3) \mod 5$, since they did not move. Moreover, p is no more in bootstate and its clock equals c. Now, in γ_{T+2}, q is no more in bootstate and its clock value belongs to $\{0, \ldots, 4\}$. That clock value should differ of at most one increment $(\mod 5)$ from the clocks of p, r_1, and r_2 since partial unison holds in γ_{T+1} and all subsequent configurations. If the clock of q equals:

- c or $(c+1) \mod 5$, the difference between the clocks of q and r_2 is at least 2 increments $(\mod 5)$,
- $(c+2) \mod 5$ or $(c+3) \mod 5$, the difference between the clocks of q and p is at least 2 increments $(\mod 5)$,
- $(c+4) \mod 5$, the difference between the clocks of q and r_1 is 2 increments $(\mod 5)$.

Hence, the safety of partial unison is necessarily violated in the configuration γ_{T+2} of e', a contradiction. □

The previous theorem states that no ρ-dynamic step can satisfy ζ, unless $\mathcal{A}$ fails. Now, by definition, for every graphs G and G', $\zeta(G, G') \Rightarrow \mathsf{C}(G, G') \wedge \neg\mathsf{ULC}(G, G')$. Hence, the following corollary holds.

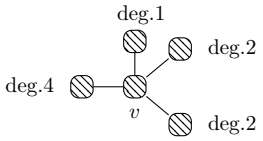
Corollary 1 *If $\alpha = 5$, then there exist graphs G and G' such that $\mathsf{C}(G, G') \wedge \neg\mathsf{ULC}(G, G') \wedge \neg\rho(G, G')$.*

The theorem below provides the same kind of results as Corollary 1 for $\alpha = 4$.

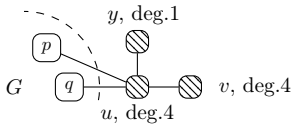
Theorem 4 *If $\alpha = 4$, then there exist graphs G and G' such that $\mathsf{C}(G, G') \wedge \neg\mathsf{ULC}(G, G') \wedge \neg\rho(G, G')$.*

Proof. We illustrate the following proof with Figure 3. Assume, by the contradiction, that $\alpha = 4$ but for every two graphs G and G' we have $\neg\mathsf{C}(G, G') \vee \mathsf{ULC}(G, G') \vee \rho(G, G')$, *i.e.*, $\mathsf{C}(G, G') \wedge \neg\mathsf{ULC}(G, G') \Rightarrow \rho(G, G')$.

To reduce the number of cases in the proof, we start by fixing a local proof environment, without loss of generality. To that goal, we consider a configuration γ_i in $\mathcal{L}^{\mathcal{A}}_{\mathsf{SU}}$ such that G_i is connected and contains at least one node. Consider also any ρ-dynamic step, $\gamma_i \mapsto^{\rho}_{d} \gamma_{i+1}$, that adds five nodes u, v, w, x and y to G_i in such way that the neighbors of v in G_{i+1} is $\{u, w, x, y\}$ and the respective degrees of u, w, x, and y are 1, 2, 2, and 4, see for instance Figure 3.(d)). Notice that from its local point of view, v cannot distinguish configuration γ_{i+1} from any other configuration resulting from the addition of v and its neighbors, since



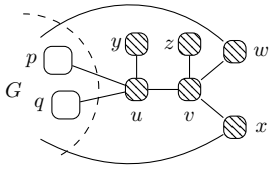
(a) After adding v and its neighbors, v is enabled and $v.clock$ is set to 0 if v moves.



(b) A dynamic step in $\mathcal{F}(G)$. Claim 0: u is enabled and the next value of $u.clock$ is completely determined by the local states of p and q.

	$p.clock$	$q.clock$	$u.clock$
Claim 1	2	3	3
Claim 2	1	2	1
Claim 3	2	2	1 or 3

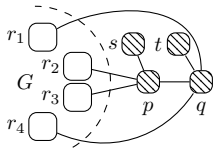
(c) Claim 1, 2, 3: after adding u, v, y, u is enabled and $u.clock$ will be fixed by $p.clock$ and $q.clock$



(d) Proof of Claim 1, 2, 3

	γ_x	γ_{x+1}		γ_y	γ_{y+1}		γ_z	γ_{z+1}
r_1	**1**	2		2	**2**		**2**	3
r_4	1	**2**		2	**2**		2	**3**
	1	2	$\cdots$	2	2	$\cdots$	2	3
	$\vdots$	$\vdots$		$\vdots$	$\vdots$		$\vdots$	$\vdots$
r_2	1	**2**		**2**	2		**2**	3
r_3	1	**2**		**2**	2		**2**	3
p		1	$\cdots$	1	3		3	

(e) Claim 3, 4, 5, and end of the proof.



(f) Proof of Claim 3.

Figure 3: Proof of Theorem 4. The hachured nodes are in bootstate. The value inside a node is its clock value. If no value is given, then the clock value is meaningless. Notation "deg." stands for degree.

v and all its neighbors are in bootstate. In γ_{i+1}, due to Lemma 1, v is enabled and if v moves, then $v.clock \in \{0, ..., 3\}$ in the next configuration. Without the loss of generality, we fix the value to 0. Hence, follows.

Local Proof Environment: Let v be a node surrounded by 4 neighbors having respectively degree 1, 2, 2 and 4 such that v and its neighbors are all in *bootstate*. In such a configuration, v is enabled and if v moves in the next step, then v sets $v.clock$ to 0. See Figure 3.(a). ■

Let $G = (V, E)$ be any connected graph of at least two nodes, p, q. Let $\mathcal{F}(G)$ be the family of graphs $G' = (V', E')$ obtained by applying a dynamic step on G such that (1) $\mathsf{C}(G, G')$ holds, (2) $V \subseteq V'$, (3) $\{u, y, v\} \subseteq V' \setminus V$, and (4) E' contains at least all links in E plus the following additional links.

- y is a 1-degree node linked to u;
- u has degree 4, it is linked to y, v, and two nodes of V; and
- v has degree 4 and is, in particular, linked to u.

See Figure 3.(b). Notice that for every G' in $\mathcal{F}(G)$, $\neg\mathsf{ULC}(G, G')$ holds, due to node y. Hence, any dynamic step that transforms G into G' is a ρ-dynamic step.

Let $\gamma \in \mathcal{L}^{\mathcal{A}}_{\mathsf{SU}}$ whose topology is G. Let $\gamma \mapsto^{\rho}_{d} \gamma'$ be a ρ-dynamic step where no process executes and that transform G into $G' \in \mathcal{F}(G)$.

Claim 0: In γ', process u is enabled (by Lemma 1) and, if it executes, the new value of $u.clock$ is completely determined by $\gamma(p)$ and $\gamma(q)$.

Claim 1: If p and q respectively have clock value 2 and 3 in configuration γ, then for every $G' \in \mathcal{F}(G)$, u is enabled in γ' and if it executes, $u.clock$ has value 3 in next configuration.

Proof of Claim 1: By Claim 0, u is enabled at γ' and its next clock value is fully determined by $\gamma(p)$ and $\gamma(q)$ whatever the graph G' of $\mathcal{F}(G)$. So, to determine this value, it is sufficient to compute it from a particular graph G' of $\mathcal{F}(G)$. We build this graph as follows: $V \subseteq V'$, $\{u, v, w, x, y, z\} \subseteq V' \setminus V$, and E' contains all links in E plus the following additional links.

- u has four neighbors: v, y, p and q,
- v has four neighbors: u, w, x, and z,
- w has two neighbors: v and a node in V,
- x has two neighbors: v and a node in V, and
- y and z have degree one.

See Figure 3.(d). Notice that, by definition, $G' \in \mathcal{F}(G)$.

We consider γ as the first configuration of an execution in $\mathcal{E}^{1,\rho}_{\mathcal{L}^{\mathcal{A}}_{\mathsf{SU}}}$. Then, the first step of the execution is the step $\gamma \mapsto^{\rho}_{d} \gamma'$. Hence in γ', u and v are both enabled (see the local proof environment and Claim 0): assume that the daemon

exactly selects u and v for next static step $\gamma' \mapsto_s \gamma''$. In γ'', the states of p and q have not changed, v and u are no more in bootstate and $v.clock = 0$, from the local proof environment. The clock value $u.clock$ should differ from the clocks of p, q, and v by at most one increment (mod 4) since partial unison holds in γ' and γ''. So, u necessarily has clock value 3 in γ''. ■

Using a similar reasoning, we obtain the following two claims.

Claim 2: If p and q respectively have clock value 1 and 2 at configuration γ, then for every $G' \in \mathcal{F}(G)$, u is enabled in γ' and if it executes, $u.clock$ has value 1 in next configuration.

Claim 3: If p and q have both clock value 2 in configuration γ, then for every $G' \in \mathcal{F}(G)$, u is enabled in γ' and if it executes, $u.clock$ has value either 1 or 3 in next configuration.

Consider now any regular[7] connected graph $G = (V, E)$ of at least four nodes, r_1, r_2, r_3 and r_4. Let $e = (\gamma_i)_{i \geq 0} \in \mathcal{E}^{0,\rho}_{\mathcal{L}^{\mathcal{A}}_{\mathsf{SU}}}$ be a *synchronous* execution of the algorithm on graph G, such that in γ_0 every process has exactly same state. As the execution is synchronous, the algorithm deterministic, and the graph regular, this property is invariant all along the execution: in every configuration γ_i of e, $\forall p \in V, \gamma_i(p) = \gamma_i(r_1)$.

Now, by hypothesis, there exists a configuration γ_i in e such that $\gamma_i \in \mathcal{L}^{\mathcal{A}}_{\mathsf{SU}}$. From γ_i, every clock in the graph regularly increments (modulo 4). We denote by $\gamma_x \mapsto_s \gamma_{x+1}$ some step in e with $x > i$ such that clock value increments from 1 in γ_x to 2 in γ_{x+1}. Moreover, let $\gamma_z \mapsto_s \gamma_{z+1}$ the next step in e where clocks increment again, namely clocks increment from 2 in γ_z to 3 in γ_{z+1}. See Figure 3.(e). Notice that in each configuration between γ_x (included) and γ_{z+1} (included) every process has the same state. Moreover, in all configurations between γ_{x+1} (included) and γ_z (included) all processes have clock value 2.

For every $k \in \{x+1, ..., z\}$, we build the execution $e_k \in \mathcal{E}^{1,\rho}_{\mathcal{L}^{\mathcal{A}}_{\mathsf{SU}}}$ such that e and e_k have the same prefix $\gamma_0, ..., \gamma_k$. But, in γ_k, e_k suffers from a dynamic step $\gamma_k \mapsto_d \gamma'_k$ built as follows: no process executes, no node or edge disappears, yet the pattern of Figure 3.(f) is added; namely this step built a graph $G' = (V', E')$ such that $V \subseteq V'$, $V' \setminus V = \{p, q, s, t\}$, E' contains all links of E plus the following additional links.

- p has four neighbors: r_2, r_3, q, s,
- q has four neighbors: r_1, r_4, p, t,
- s has one neighbor: p, and
- t has one neighbor: q.

Again, notice that, by definition, $G' \in \mathcal{F}(G)$. Hence any dynamic step that transforms G into G' is a ρ-dynamic step.

For every $k \in \{x+1, ..., z\}$, p and q are enabled in γ'_k (by Lemma 1). We note $\gamma'_k \mapsto_s \gamma''_k$ the next static step after the dynamic step where p and q are

[7] Regular means that all nodes have the same degree.

the only nodes activated by the daemon. In the following, we are interesting in the value of $\gamma''_k(p)$ for $k \in \{x+1, ..., z\}$. Note that for all those values, Claim 3 applies, hence $\gamma''_k(p)$ is either 1 or 3.

Claim 4: $\gamma''_{x+1}(p) = 1$.

Proof of Claim 4: We consider the execution $e'_{x+1} \in \mathcal{E}^{1,\rho}_{\mathcal{L}^{\mathcal{A}}_{\mathsf{SU}}}$ with prefix $\gamma_0, ..., \gamma_x$. In γ_x, we introduce a non-synchronous static step $\gamma_x \mapsto_s \vartheta_{x+1}$ such that every process but r_1 are activated. Hence $\vartheta_{x+1}(r_1) = \gamma_x(r_1)$ (in particular the clock is 1) and $\vartheta_{x+1}(n) = \gamma_{x+1}(n)$ for every $n \neq r_1$ (with in particular a clock value equal to 2). The next step of e'_{x+1} is a ρ-dynamic step $\vartheta_{x+1} \mapsto^{\rho}_{d} \vartheta'_{x+1}$ that transforms G into G' and activates no process; again p and q are enabled in ϑ'_{x+1} and the next step is a static step $\vartheta'_{x+1} \mapsto_s \vartheta''_{x+1}$ where the daemon uniquely activates p and q. Claim 2 applies to q: q is enabled, and $\vartheta''_{x+1}(q).clock = 1$. Claim 3 applies to p and $\vartheta''_{x+1}(p).clock$ is either 1 or 3. Hence, to satisfy the partial unison in ϑ''_{x+1}, $\vartheta''_{x+1}(p).clock$ is necessarily equal to 1.

Now, back to execution e_{x+1}, Claim 0 applies to p in step $\gamma'_{x+1} \mapsto_s \gamma''_{x+1}$: $\gamma''_{x+1}(p)$ is fully determined by $\gamma_{x+1}(r_2)$ and $\gamma_{x+1}(r_3)$. As $\gamma_{x+1}(r_2)$ (resp., $\gamma_{x+1}(r_3)$) has been obtained by executing the local algorithm of r_2 (resp., r_3) and as $\vartheta'_{x+1}(r_2)$ (resp., $\vartheta'_{x+1}(r_3)$) has been obtained exactly the same way, they are equal. Hence $\vartheta''_{x+1}(p) = \gamma''_{x+1}(p) = 1$. ■

Claim 5: $\gamma''_z(p) = 3$.

Proof of Claim 5: We consider the execution $e'_z \in \mathcal{E}^{1,\rho}_{\mathcal{L}^{\mathcal{A}}_{\mathsf{SU}}}$ with prefix $\gamma_0, ..., \gamma_z$. At γ_z, we introduce a non-synchronous static step $\gamma_z \mapsto_s \vartheta_{z+1}$ such that process r_4 only has been activated. Hence $\vartheta_{z+1}(r_4) = \gamma_{z+1}(r_4)$ (in particular the clock is 3) and $\vartheta_{x+1}(n) = \gamma_z(n)$ for every $n \neq r_4$ (with clocks equal to 2). The next step of e'_z is a ρ-dynamic step $\vartheta_{z+1} \mapsto^{\rho}_{d} \vartheta'_{z+1}$ that transforms G into G' and activates no process; again p and q are enabled at ϑ'_{z+1} and next step is a static step $\vartheta'_{z+1} \mapsto_s \vartheta''_{z+1}$ where the daemon uniquely activates p and q. Claim 1 applies to q: q is enabled, hence $\vartheta''_{z+1}(q).clock = 1$. Claim 3 applies to p and $\vartheta''_{z+1}(p).clock$ is either 1 or 3. Hence, to satisfy the partial unison in ϑ''_{z+1}, $\vartheta''_{z+1}(p).clock$ is necessarily equal to 1.

Now, back to execution e_z, Claim 0 applies to p in step $\gamma'_z \mapsto_s \gamma''_z$: $\gamma''_z(p)$ is fully determined by $\gamma_z(r_2)$ and $\gamma_z(r_3)$. As $\gamma_z(r_2)$ (resp., $\gamma_z(r_3)$) has been obtained by executing the local algorithm of r_2 (resp., r_3) and as $\vartheta'_{z+1}(r_2)$ (resp., $\vartheta'_{z+1}(r_3)$) has been obtained exactly the same way, they are equal. Hence $\vartheta''_{z+1}(p) = \gamma''_{z+1}(p) = 3$. ■

By Claims 3-5, the sequence of values $(\gamma''_i(p).clock)_{i \in \{x+1,...,z\}}$ is only consists of values 1 and 3, starting with 1 and ending with 3. Hence, there exists an index $y \in \{x+1, z-1\}$ for which the value switches from 1 to 3, *i.e.*, $\gamma''_y(p) = 1$ and $\gamma''_{y+1}(p) = 3$.

We consider the execution $e'_y \in \mathcal{E}^{1,\rho}_{\mathcal{L}^{\mathcal{A}}_{\mathsf{SU}}}$ with prefix $\gamma_0, ..., \gamma_y$. In γ_y, we introduce a non-synchronous static step $\gamma_y \mapsto_s \vartheta_{y+1}$ such that all processes are activated, except r_2 and r_3. Hence, $\vartheta_{y+1}(r_2) = \gamma_y(r_2)$, $\vartheta_{y+1}(r_3) = \gamma_y(r_3)$, and $\vartheta_{y+1}(r) = \gamma_{y+1}(r)$ for every $r \notin \{r_2, r_3\}$. The next step of e'_y is a ρ-dynamic step $\vartheta_{y+1} \mapsto^{\rho}_{d} \vartheta'_{y+1}$ that transforms G into G' and activates no process; again p and q are

enabled in ϑ'_{y+1} and the next step is a static step $\vartheta'_{y+1} \mapsto_s \vartheta''_{y+1}$ where the daemon uniquely activates p and q.

Claim 0 applies to p (resp., q) in step $\vartheta'_{y+1} \mapsto_s \vartheta''_{y+1}$: $\vartheta''_{y+1}(p)$ is fully determined by $\vartheta_{y+1}(r_2) = \gamma_y(r_2)$ and $\vartheta_{y+1}(r_3) = \gamma_y(r_3)$ (resp., $\vartheta_{y+1}(r_1) = \gamma_{y+1}(r_1)$ and $\vartheta_{y+1}(r_1) = \gamma_{y+1}(r_1)$). Hence, $\vartheta''_{y+1}(p) = \gamma''_y(p) = 1$ and $\vartheta''_{y+1}(q) = \gamma''_y(p) = 3$. This contradicts the fact that the partial unison holds in γ''_y. □

6 Self-Stabilizing Strong Unison

In this section, we propose an algorithm, called $\mathcal{SU}$, which is self-stabilizing for the strong unison problem in any arbitrary connected anonymous network. This algorithm works for any period $\alpha \geq 2$ (recall that the problem is undefined for $\alpha < 2$) and is based on an algorithm previously proposed by Boulinier in [17]. This latter is self-stabilizing for the weak unison problem and works for any period $\beta > n^2$, where n is the number of processes. We first recall the algorithm of Boulinier, called here Algorithm $\mathcal{WU}$, in Subsection 6.1. Notice that the notation used in this algorithm will be also applicable to our algorithms. We present Algorithm $\mathcal{SU}$, its proof of correctness, and its complexity analysis in Subsection 6.2. Algorithms $\mathcal{WU}$ and $\mathcal{SU}$ being only self-stabilizing, all their executions contain no topological change, yet start from arbitrary configurations. Consequently, the topology of the network consists in a connected graph $G = (V, E)$ of n nodes which is fixed all along the execution.[8] Moreover, no bootstate has to be defined. Recall that $\mathcal{D}$ is the diameter of G.

6.1 Algorithm $\mathcal{WU}$

Algorithm $\mathcal{WU}$, see Algorithm 1 for its formal code, has been proposed by Boulinier in his PhD thesis [17]. Actually, it is a generalization of the self-stabilizing weak unison algorithm proposed by Couvreur *et al.* [25]. In Algorithm $\mathcal{WU}$, each process p is endowed with a clock variable $p.t \in \{0, \ldots, \beta - 1\}$, where β is its period. β should be greater than n^2. The algorithm also uses another constant, noted μ, which should satisfy $n \leq \mu < \frac{\beta}{2}$.

Notations We define the *delay* between two integer values x and y by the function $d_\beta(x, y) = \min\big((x - y) \mod \beta, (y - x) \mod \beta\big)$. Then, let $\preceq_{\beta,\mu}$ be the relation such that for every two integer values x and y, $x \preceq_{\beta,\mu} y \equiv \big((y - x) \mod \beta\big) \leq \mu$. (*N.b.*, $\preceq_{\beta,\mu}$ is only defined for $\mu < \frac{\beta}{2}$, see Definition 14 in [17].)

6.1.1 The Algorithm

Two actions are used to maintain the clock $p.t$ at each process p. When the delay between $p.t$ and the clocks of some neighbors is greater than one, but the

[8] Precisely, for both $\mathcal{WU}$ and $\mathcal{SU}$, we have $\forall \gamma_i \in \mathcal{C}$, $G_i = G$.

maximum delay is not too big (that is, does not exceed μ), then it is possible to "normally" converge, using Action $\mathcal{WU}$-N, to a configuration where the delay between those clocks is at most one by incrementing the clocks of the most behind processes among p and its neighbors. Moreover, once legitimacy is achieved, p can "normally" increment its clock still using Action $\mathcal{WU}$-N when it is on time or one increment late with all its neighbors. In contrast, if the delay is too big (that is, the delay between the clocks of p and one of its neighbors is more than μ) and the clock of p is not yet reset, then p should reset its clock to 0 using Action $\mathcal{WU}$-R.

Algorithm 1 $\mathcal{WU}$, for every process p

Parameters:
β: any positive integer such that $\beta > n^2$
μ: any positive integer such that $n \leq \mu < \frac{\beta}{2}$

Variable:
$p.t \in \{0, \ldots, \beta - 1\}$

Actions:

$\mathcal{WU}$-N	::	$\forall q \in p.\mathcal{N}, p.t \preceq_{\beta,\mu} q.t$	$\rightarrow$	$p.t \leftarrow (p.t + 1) \mod \beta$
$\mathcal{WU}$-R	::	$\exists q \in p.\mathcal{N}, d_\beta\big(p.t, q.t\big) > \mu \wedge p.t \neq 0$	$\rightarrow$	$p.t \leftarrow 0$

From [17], we have the following theorem.

Theorem 5 *Algorithm $\mathcal{WU}$ is self-stabilizing for SP_{wu} (the specification of weak unison) in an arbitrary connected network assuming a distributed unfair daemon. Its set of legitimate configurations is*

$$\mathcal{L}_{\mathsf{wu}} = \{\gamma \in \mathcal{C} : \forall p \in V, \forall q \in \gamma(p).\mathcal{N}, d_\beta\big(\gamma(p).t, \gamma(q).t\big) \leq 1\}$$

Its stabilization time is at most $n + \mu\mathcal{D}$ rounds, where n (resp. $\mathcal{D}$) is the size (resp. diameter) of the network and μ is a parameter satisfying $n \leq \mu < \frac{\beta}{2}$.

By definition, $\mathcal{D} < n$, consequently we have:

Remark 2 *Once Algorithm $\mathcal{WU}$ has stabilized, the delay between t-clocks of any two arbitrary far processes is at most $n - 1$.*

Some other useful results from [17] about Algorithm $\mathcal{WU}$ are recalled below.

6.1.2 Results from [17]

Algorithm $\mathcal{WU}$ is an instance of the parametric algorithm GAU in [17]: $\mathcal{WU} = GAU(\beta, 0, \mu)$. The following five lemmas (4-8) are used to establish the self-stabilization of $\mathcal{WU}$ for SP_{wu} using the set of legitimate configurations $\mathcal{L}_{\mathsf{wu}}$. The proof of self-stabilization is actually divided into several steps. The first step (Lemma 5) consists in showing the convergence of $\mathcal{WU}$ from $\mathcal{C}$ to C_μ, where C_μ is the set of configurations where the delay between the clocks of two neighbors is at most μ, *i.e.*,

$$C_\mu = \{\gamma \in \mathcal{C} : \forall p \in V, \forall q \in \gamma(p).\mathcal{N}, d_\beta\big(\gamma(p).t, \gamma(q).t\big) \leq \mu\}$$

C_μ is shown to be closed under $\mathcal{WU}$ in Lemma 4. (Notice that $\mathcal{L}_{\mathsf{wu}} \subseteq C_\mu$.) The liveness part of SP_{wu} (the clock $p.t$ of every process p goes through each value in $\{0, \ldots, \beta-1\}$ in increasing order infinitely often) is shown for every execution starting from C_μ in Lemma 6.

Lemma 4 (Property 8 in [17]) *C_μ is closed under $\mathcal{WU}$.*

Lemma 5 (Theorem 56 in [17]) *If $n \leq \mu < \frac{\beta}{2}$, then $\forall e \in \mathcal{E}^0$, $\exists \gamma \in e$ such that $\gamma \in C_\mu$.*

Lemma 6 (Theorem 21 in [17]) *If $\beta > n^2$, then $\forall e \in \mathcal{E}^0_{C_\mu}$, e satisfies the liveness part of SP_{wu}.*

Then, the second step consists of showing closure of $\mathcal{L}_{\mathsf{wu}}$ under $\mathcal{WU}$ (Lemma 7) and the convergence from C_μ to $\mathcal{L}_{\mathsf{wu}}$ (Lemma 8). Regarding the correctness, the safety part of SP_{wu} is ensured by definition of $\mathcal{L}_{\mathsf{wu}}$, whereas the liveness part is already ensured by Lemma 6. Precisely:

Lemma 7 (Property 2 in [17]) *$\mathcal{L}_{\mathsf{wu}}$ is closed under $\mathcal{WU}$.*

Lemma 8 (Theorems 29 in [17]) *If $\beta > n^2$ and $\mu < \frac{\beta}{2}$, then $\forall e \in \mathcal{E}^0_{C_\mu}$, $\exists \gamma \in e$ such that $\gamma \in \mathcal{L}_{\mathsf{wu}}$.*

Some performances of Algorithm $\mathcal{WU}$ are recalled in Theorems 6 and 7 below.

Theorem 6 (Theorem 61 in [17]) *If $n \leq \mu < \frac{\beta}{2}$, the convergence time of $\mathcal{WU}$ from $\mathcal{C}$ to C_μ is at most n rounds.*

Theorem 7 (Theorems 20 and 28 in [17]) *If $\beta > n^2$ and $\mu < \frac{\beta}{2}$, the convergence time of $\mathcal{WU}$ from C_μ to $\mathcal{L}_{\mathsf{wu}}$ is at most $\mu\mathcal{D}$ rounds.*

Finally, Lemma 9 below is a technical result about the values of t-variables.

Lemma 9 (Theorem 20, Lemma 22, Proposition 25, and Property 27 in [17]) *If $\beta > n^2$ and $\beta > 2\mu$, then $\forall e = (\gamma_i)_{i \geq 0} \in \mathcal{E}^0_{C_\mu}$, there exists a so-called* shifting function *$f : \mathcal{C} \times V \to \mathbb{Z}$ such that $\forall i \geq 0$, $\forall p, q \in V$,*

- *$\forall 0 \leq i \leq j$, $f(\gamma_i, p) \leq f(\gamma_j, p)$,*
- *$p.t \preceq_{\beta,\mu} q.t$ if and only if $f(\gamma_i, p) \leq f(\gamma_i, q)$,*
- *$\forall i \geq 0$, $\forall p \in V$, $f(\gamma_i, p) \mod \beta = \gamma_i(p).t$, and*
- *$|f(\gamma_i, p) - f(\gamma_i, q)| = d_\beta\big(\gamma_i(p).t, \gamma_i(q).t\big)$.*

6.1.3 Complexity Analysis

Let C_μ be the set of configurations where the delay between two neighboring clocks is at most μ. Below, we prove in Lemma 10 (resp. Lemma 11) a bound on the time required to ensure that all t-variables have incremented k times. This bound holds since the system has reached a configuration of C_μ (resp. $\mathcal{L}_{\mathsf{wu}}$).

Lemma 10 *$\forall k \geq 1$, $\forall e \in \mathcal{E}^0_{C_\mu}$, every process p increments $p.t$ executing $\mathcal{WU}$-N at least k times every $\mu\mathcal{D} + k$ rounds, where $\mathcal{D}$ is the diameter of the network.*

Proof. Let $k \geq 1$. Let $e = (\gamma_i)_{i\geq 0} \in \mathcal{E}^0_{C_\mu}$. Using Lemma 9, there is a shifting function f such that $\forall i \geq 0$, $\forall p, q \in V$, $|f(\gamma_i, p) - f(\gamma_i, q)| \leq \mu\mathcal{D}$. For every $i \geq 0$, we note $f^{\min}_{\gamma_i} = \min\{f(\gamma_i, x) : x \in V\}$. Action $\mathcal{WU}$-N is enabled in γ_i at every process $x \in V$ for which $f(\gamma_i, x) = f^{\min}_{\gamma_i}$. So, after one round, every such a process x has incremented its t-variable (by $\mathcal{WU}$-N) at least once. Let γ_j be the first configuration after one round. Then, $f^{\min}_{\gamma_j} \geq f^{\min}_{\gamma_i} + 1$. We now consider γ_d to be the first configuration after $\mu\mathcal{D} + k$ rounds, starting from γ_i. Using the same arguments as for γ_j inductively, we have $f^{\min}_{\gamma_d} \geq f^{\min}_{\gamma_i} + \mu\mathcal{D} + k$ $(*)$.

Let p be a process in V. By definitions of f and $f^{\min}_{\gamma_i}$, we have that $f^{\min}_{\gamma_i} \leq f(\gamma_i, p) \leq f^{\min}_{\gamma_i} + \mu\mathcal{D}$ $(**)$. Assume now that p increments $\sharp incr < k$ times $p.t$ between γ_i and γ_d. Then

$$\begin{aligned} f(\gamma_d, p) = f(\gamma_i, p) + \sharp incr &< f(\gamma_i, p) + k \text{ (assumption on } \#incr) \\ &\leq f^{\min}_{\gamma_i} + \mu\mathcal{D} + k, \text{ by } (**) \\ &\leq f^{\min}_{\gamma_d}, \text{ by } (*) \end{aligned}$$

So, p satisfies $f(\gamma_d, p) < f^{\min}_{\gamma_d}$, a contradiction. □

Lemma 11 *$\forall k \geq 1$, $\forall e \in \mathcal{E}^0_{\mathcal{L}_{\mathsf{wu}}}$, every process p increments its clock $p.t$ executing action $\mathcal{WU}$-N at least k times every $\mathcal{D} + k$ rounds, where $\mathcal{D}$ is the diameter of the network.*

Proof. The proof of this lemma is exactly the same as the one of Lemma 10, yet replacing C_μ by $\mathcal{L}_{\mathsf{wu}}$ and $\mu\mathcal{D}$ by $\mathcal{D}$. □

6.2 Algorithm $\mathcal{SU}$

In this subsection, we still assume a non-dynamic context (no topological change) and we use the notations defined in Subsection 6.1. Algorithm $\mathcal{SU}$ is a straightforward adaptation of Algorithm $\mathcal{WU}$. More precisely, Algorithm $\mathcal{SU}$ maintains two clocks at each process p. The first one, $p.t \in \{0, \ldots, \beta - 1\}$, is called the *internal clock* and is maintained exactly as in Algorithm $\mathcal{WU}$. Then, $p.t$ is used as an internal pulse machine to increment a second, yet actual, clock of Algorithm $\mathcal{SU}$ $p.c \in \{0, \ldots, \alpha - 1\}$, also referred to as *external clock*.

Algorithm $\mathcal{SU}$ (see Algorithm 2), is designed for any period $\alpha \geq 2$. Its actions $\mathcal{SU}$-N and $\mathcal{SU}$-R are identical to actions $\mathcal{WU}$-N and $\mathcal{WU}$-R of Algorithm $\mathcal{WU}$,

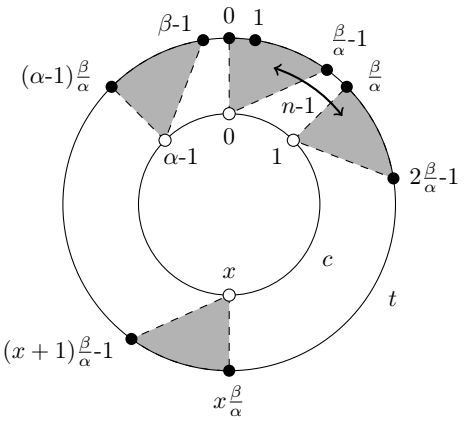


Figure 4: Relationship between variables t and c.

except that we add the computation of the external c-clock in their respective statement.

We already know that Algorithm $\mathcal{WU}$ stabilizes to a configuration from which t-clocks regularly increment while preserving a bounded delay of at most one between two neighboring processes, and so of at most $n-1$ between any two processes (see Remark 2). Algorithm $\mathcal{SU}$ implements the same mechanism to maintain $p.t$ at each process p and computes $p.c$ from $p.t$ as a normalization operation from clock values in $\{0,\ldots,\beta-1\}$ to $\{0,\ldots,\alpha-1\}$: each time the value of $p.t$ is modified, $p.c$ is updated to $\left\lfloor \frac{\alpha}{\beta} p.t \right\rfloor$. Hence, we can set β in such way that $K = \frac{\beta}{\alpha}$ is greater than or equal to n (here, we chose $K > \mu \geq n$ for sake of simplicity) to ensure that, when the delay between any two t-clocks is at most $n-1$, the delay between any two c-clocks is at most one, see Figure 4. Furthermore, the liveness of $\mathcal{WU}$ ensures that every t-clock increments infinitely often, hence so do c-clocks.

Remark 3 *Since $\beta > \mu^2$ and $\mu \geq 2$, we have $\beta \geq 2\mu$.*

Remark 4 *By construction and from Remark 3, all results on t-clocks in Algorithm $\mathcal{WU}$ also holds for t-clocks in Algorithm $\mathcal{SU}$.*

Theorem 8 below states that Algorithm $\mathcal{SU}$ is self-stabilizing for the strong unison problem. We detail the proof of this intuitive result in the sequel.

Theorem 8 *Algorithm $\mathcal{SU}$ is self-stabilizing for SP_{SU} (the specification of the strong unison) in any arbitrary connected anonymous network assuming a distributed unfair daemon. Its stabilization time is at most $n+(\mu+1)\mathcal{D}+1$ rounds, where n (resp. $\mathcal{D}$) is the size (resp. diameter) of the network and μ is a parameter satisfying $\mu \geq \max(n,2)$.*

Algorithm 2 $\mathcal{SU}$, for every process p

Parameters:

α: any positive integer such that $\alpha \geq 2$
μ: any positive integer such that $\mu \geq \max(n, 2)$
β: any positive integer such that $\beta > \mu^2$ and $\exists K$ such that $K > \mu$ and $\beta = K\alpha$

Variables:

$p.c \in \{0, \ldots, \alpha - 1\}$
$p.t \in \{0, \ldots, \beta - 1\}$

Actions:

$\mathcal{SU}$-N	::	$\forall q \in p.\mathcal{N}, p.t \preceq_{\beta,\mu} q.t$	$\rightarrow$	$p.t \leftarrow (p.t + 1) \mod \beta$
				$p.c \leftarrow \left\lfloor \frac{\alpha}{\beta} p.t \right\rfloor$
$\mathcal{SU}$-R	::	$\exists q \in p.\mathcal{N}, d_\beta\big(p.t, q.t\big) > \mu$	$\rightarrow$	$p.t \leftarrow 0$
		$\wedge\ p.t \neq 0$		$p.c \leftarrow 0$

6.2.1 Correctness

We first define a set of legitimate configurations *w.r.t.* specification SP_{SU} (Definition 5). Then, we prove the closure and convergence *w.r.t.* those legitimate configurations (see Lemmas 12 and 13). Afterwards, we prove the correctness *w.r.t.* specification SP_{SU} in any execution starting in a legitimate configuration, namely, safety is shown in Lemma 16 and liveness is proven in Lemma 17.

Definition 5 (Legitimate Configurations of $\mathcal{SU}$ *w.r.t.* SP_{SU}) *A configuration γ of $\mathcal{SU}$ is* legitimate w.r.t. SP_{SU} if and only if

1. $\forall p \in V$, $\forall q \in \gamma(p).\mathcal{N}$, $d_\beta\big(\gamma(p).t, \gamma(q).t\big) \leq 1$.
2. $\forall p \in V$, $\gamma(p).c = \left\lfloor \frac{\alpha}{\beta} \gamma(p).t \right\rfloor$.

We denote by $\mathcal{L}_{\mathsf{SU}}$ the set of legitimate configurations of $\mathcal{SU}$ w.r.t. *SP_{SU}.*

By definition, $\mu \geq n > 0$, hence from Definition 5, follows.

Remark 5 *In any legitimate configuration $\gamma \in \mathcal{L}_{\mathsf{SU}}$, $\forall p, q \in V$, $d_\beta\big(\gamma(p).t, \gamma(q).t\big) \leq \mu$.*

Lemma 12 (Closure) *$\mathcal{L}_{\mathsf{SU}}$ is closed under $\mathcal{SU}$.*

Proof. First, from Theorem 5 and Remark 4, note that the set of legitimate configurations defined for $\mathcal{WU}$ is also closed for $\mathcal{SU}$. Hence we only have to check closure for the second constraint of Definition 5, the one on c-variables.

Let $\gamma \in \mathcal{L}_{\mathsf{SU}}$ be a legitimate configuration of $\mathcal{SU}$ and let $\gamma \mapsto_s \gamma'$ be a static step of $\mathcal{SU}$. Let $p \in V$. As $\gamma \in \mathcal{L}_{\mathsf{SU}}$, $\gamma(p).c = \left\lfloor \frac{\alpha}{\beta} \gamma(p).t \right\rfloor$. Either p does not execute any action during step $\gamma \mapsto_s \gamma'$, or p executes $\mathcal{SU}$-N or $\mathcal{SU}$-R. These two actions update $p.c$ according to the new value of $p.t$. Hence, $\gamma'(p).c = \left\lfloor \frac{\alpha}{\beta} \gamma'(p).t \right\rfloor$. □

Lemma 13 (Convergence) *$\mathcal{C}$ (the set of all possible configurations) converges to $\mathcal{L}_{\mathsf{SU}}$ under $\mathcal{SU}$.*

Proof. From Theorem 5 and Remark 4, the set of legitimate configurations for $\mathcal{WU}$ is also reached in a finite number of steps for $\mathcal{SU}$. Hence, we only have to check that the second constraint (the one on c-variables) is also achievable within a finite number of steps.

Again by Theorem 5 and Remark 4, liveness of Specification SP_{WU} is ensured by $\mathcal{WU}$ and therefore by $\mathcal{SU}$. Hence, after stabilization, each process p updates its internal clock $p.t$ within a finite time; meanwhile $p.c$ is also updated to $\left\lfloor \frac{\alpha}{\beta} p.t \right\rfloor$. □

Remarks 6, 7 and Lemma 14 are technical results on the values of t- and c-variables that will be used to prove that the safety of Specification SP_{SU} is achieved in any execution that starts from a legitimate configuration. For all these lemmas, we assume that α, β, K are positive numbers that satisfies the constraint declared on the **Parameters** section of Algorithm $\mathcal{SU}$, namely $\beta = K\alpha$.

Remark 6 *Let $x \in \{0, \ldots, \alpha-1\}$ and $\xi \in \{0, \ldots, \frac{\beta}{\alpha}-1\}$. The following equality holds: $\left\lfloor \frac{\alpha}{\beta} \left(x\frac{\beta}{\alpha} + \xi \right) \right\rfloor = x$.*

Remark 7 *Let $x_1, x_2 \in \{0, \ldots, \alpha-1\}$ and $\xi_1, \xi_2 \in \{0, \ldots, \frac{\beta}{\alpha}-1\}$. The following assertion holds: $x_1\frac{\beta}{\alpha} + \xi_1 \leq x_2\frac{\beta}{\alpha} + \xi_2 \Rightarrow x_1 \leq x_2$*

We apply Remarks 6 and 7 by instantiating the value of the internal clock t with $x\frac{\beta}{\alpha} + \xi$. Since the value of the external clock c is computed as $\left\lfloor \frac{\alpha}{\beta} t \right\rfloor$ in Algorithm 2, we have $c = x$. Now, if we chose β (period of internal clocks) such that it can be written as $\beta = K\alpha$ with K a positive integer, the value of $c = \left\lfloor \frac{\alpha}{\beta} t \right\rfloor$ is always a non negative integer which evolves according to $t = c\frac{\beta}{\alpha} + \xi$ as shown in Figure 4.

Lemma 14 *Let $t_1, t_2 \in \{0, ..., \beta - 1\}$. The following assertion holds:*

$$\forall d < K,\ \ d_\beta(t_1, t_2) \leq d \Rightarrow d_\alpha\Big(\left\lfloor \frac{\alpha}{\beta} t_1 \right\rfloor, \left\lfloor \frac{\alpha}{\beta} t_2 \right\rfloor \Big) \leq 1$$

Proof. Let $t_1, t_2 \in \{0, ..., \beta - 1\}$ such that $d_\beta(t_1, t_2) \leq d$. Recall that $K = \frac{\beta}{\alpha}$. We write t_1 and t_2 as $t_1 = x_1K + \xi_1$ and $t_2 = x_2K + \xi_2$ where $x_1, x_2 \in \{0, \ldots, \alpha - 1\}$ (resp. $\xi_1, \xi_2 \in \{0, \ldots, K - 1\}$) are the quotients (resp. remainders) of the Euclidean division of t_1, t_2 by K. From Remark 6, we have that $\lfloor t_1/K \rfloor = x_1$ and $\lfloor t_2/K \rfloor = x_2$.

Assume, by contradiction, that $d_\alpha(x_1, x_2) > 1$. By definition, this means that $\min\big((x_1 - x_2) \mod \alpha, (x_2 - x_1) \mod \alpha\big) > 1$. This implies that both $(x_1 - x_2) \mod \alpha > 1$ and $(x_2 - x_1) \mod \alpha > 1$. As $d_\beta(t_1, t_2) \leq d$, $\min\big((t_1 - t_2) \mod \beta, (t_2 - t_1) \mod \beta\big) \leq d$. Without loss of generality, assume that $(t_1 - t_2) \mod \beta \leq d$. There are two cases:

1. If $t_1 \geq t_2$, then $(t_1 - t_2) \mod \beta = t_1 - t_2$. So, $t_1 - t_2 \leq d$.

   Now, as $t_1 \geq t_2$, $x_1 \geq x_2$ by Remark 7. Hence $x_1 - x_2 = (x_1 - x_2) \mod \alpha > 1$. As x_1 and x_2 are natural numbers, this implies that $x_1 - x_2 \geq 2$. We rewrite the inequality as $x_1K + \xi_1 - x_2K - \xi_2 \geq 2K + \xi_1 - \xi_2$. Since $\xi_1, \xi_2 \in \{0, \ldots, K-1\}$, we have $-K < \xi_1 - \xi_2 < K$ and therefore $x_1K + \xi_1 - x_2K - \xi_2 > K > d$. Hence, $t_1 - t_2 > d$, a contradiction.

2. If $t_1 < t_2$, then $(t_1 - t_2) \mod \beta = \beta + t_1 - t_2$. So, $\beta + t_1 - t_2 \leq d$.

   Now, as $t_1 < t_2$, $x_1 \leq x_2$ by Remark 7. Hence $(x_1 - x_2) \mod \alpha = \alpha + x_1 - x_2 > 1$. As x_1 and x_2 are natural numbers, this implies that $\alpha + x_1 - x_2 \geq 2$. We rewrite the inequality as $\beta + x_1K + \xi_1 - x_2K - \xi_2 \geq 2K + \xi_1 - \xi_2$. Since $\xi_1, \xi_2 \in \{0, \ldots, K-1\}$, we have $-K < \xi_1 - \xi_2 < K$ and therefore $\beta + x_1K + \xi_1 - x_2K - \xi_2 > K > d$. Hence, $\beta + t_1 - t_2 > d$, a contradiction.

□

As previous remarks, Lemma 14 will be used with the internal clock $t = c\frac{\beta}{\alpha} + \xi$: it expresses that once internal clocks have stabilized at a delay smaller than d, external clocks are at delay smaller than 1. We now prove that Algorithm 2 achieves the safety and liveness properties of SP_{SU} in any execution starting from a legitimate configuration.

Remark 8 (Safety for $\alpha = 2$)) *Assume $\alpha = 2$. Every execution $e \in \mathcal{E}^0_{\mathcal{L}_{\mathsf{SU}}}$ satisfies the safety of SP_{SU}. Indeed, there is only two possible values of clock, so there is at most two (consecutive) values of clock in the network.*

Lemma 15 (Safety for $\alpha = 3$) *Assume $\alpha = 3$. Every execution $e \in \mathcal{E}^0_{\mathcal{L}_{\mathsf{SU}}}$ satisfies the safety of SP_{SU}.*

Proof. If the number of nodes in the network is smaller than 3, trivially there is no more than two different values for clock c. Otherwise, let $\gamma \in \mathcal{L}_{\mathsf{SU}}$ be a legitimate configuration *w.r.t.* SP_{SU} under $\mathcal{SU}$. Assume, by contradiction, that there are more than two different values of variable c in γ: $\exists p_0, p_1, p_2 \in V$ such that $\gamma(p_0).c = 0$, $\gamma(p_1).c = 1$ and $\gamma(p_2).c = 2$. We denote $t_j = \gamma(p_j).t$ for all $j \in \{0, 1, 2\}$ and using Remark 6, there exists $\xi_0, \xi_1, \xi_2 \in \{0, ..., \beta/3 - 1\}$ such that $t_0 = \xi_0$, $t_1 = \beta/3 + \xi_1$ and $t_2 = 2\beta/3 + \xi_2$ (see Figure 5).

As γ is legitimate *w.r.t.* SP_{SU}, the delay (d_β) between any two internal clocks c is upper bounded by $n - 1$ (Remark 2) and using the assumption also upper bounded by $K = \beta/3$ (strict upper bound). So in particular

$$\begin{aligned} d_\beta(t_0, t_1) &= \beta/3 + \xi_1 - \xi_0 < n < K = \beta/3 \\ d_\beta(t_1, t_2) &= \beta/3 + \xi_2 - \xi_1 < n < K = \beta/3 \\ d_\beta(t_2, t_0) &= \beta/3 + \xi_0 - \xi_2 < n < K = \beta/3 \end{aligned}$$

It comes that $\xi_1 < \xi_0 < \xi_2 < \xi_1$, a contradiction.

Finally, as the set $\mathcal{L}_{\mathsf{SU}}$ is closed (Lemma 12), we are done. □

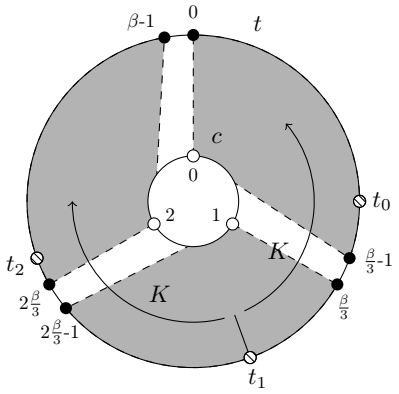


Figure 5: Relative positions of t_0, t_1, and t_2.

Lemma 16 (Safety for $\alpha > 3$) *Assumes $\alpha > 3$. Every execution $e \in \mathcal{E}^0_{\mathcal{L}_{\mathsf{SU}}}$ satisfies the safety of SP_{SU}.*

Proof. Let $\gamma \in \mathcal{L}_{\mathsf{SU}}$: the delay ($\beta$) between any two internal clocks t in γ is upper bounded by $n-1$ and for any process, $p \in V$, $\gamma(p).c = \left\lfloor \frac{\alpha}{\beta}\gamma(p).t \right\rfloor$. Hence, using Lemma 14 with $d = n-1 < K$, we have $\forall p, q \in V$, $d_\alpha\big(\gamma(p).c, \gamma(q).c\big) \leq 1$. As $\alpha > 3$, this proves that the variables c in γ have at most two different consecutive values.

Finally, as the set $\mathcal{L}_{\mathsf{SU}}$ is closed (Lemma 12), we are done. □

Lemma 17 (Liveness) *Every execution $e \in \mathcal{E}^0_{\mathcal{L}_{\mathsf{SU}}}$ satisfies the liveness of SP_{SU}.*

Proof. Let $e = (\gamma_i)_{i\geq 0} \in \mathcal{E}^0_{\mathcal{L}_{\mathsf{SU}}}$. Let p be a process. γ_0 is a legitimate configuration of $\mathcal{WU}$ so p increments infinitely often $p.t$ using $\mathcal{SU}$-N (by Theorem 5 and Remark 4). So $p.t$ goes through each integer value between 0 and $\beta - 1$ infinitely often (in increasing order). Hence, by Remark 6, $p.c$ is incremented infinitely often and goes through each integer value between 0 and $\alpha - 1$ (in increasing order). □

Proof of Theorem 8. Lemma 12 (closure), Lemma 13 (convergence), Lemmas 16-17 and Remark 8 (correctness) prove that Algorithm $\mathcal{SU}$ is self-stabilizing for SP_{SU} in any arbitrary connected anonymous network assuming a distributed unfair daemon. □

6.2.2 Complexity Analysis

We now give some complexity results about Algorithm $\mathcal{SU}$. Precisely, a bound on the stabilization time of $\mathcal{SU}$ is given in Theorem 9. Then, a delay between any two consecutive clocks increments, which holds once $\mathcal{SU}$ has stabilized, is given in Theorem 10.

Theorem 9 *The stabilization time of $\mathcal{SU}$ to $\mathcal{L}_{\mathsf{SU}}$ is at most $n + (\mu + 1)\mathcal{D} + 1$ rounds, where n (resp. $\mathcal{D}$) is the size (resp. diameter) of the network and μ is a parameter satisfying $\mu \geq \max(n, 2)$.*

Proof. Let $(\gamma_i)_{i\geq 0} \in \mathcal{E}^0$. The behavior of the t-variables in $\mathcal{SU}$ is similar to that of $\mathcal{WU}$ (Remark 4), which stabilizes in at most $n + \mu\mathcal{D}$ rounds (see Theorems 6 and 7) to weak unison. So, in $n + \mu\mathcal{D}$ rounds, the delay between the t-clocks of any two arbitrary far processes is at most $n - 1$ (Remark 2). If c-variables are well-calculated according to t-variables, *i.e.*, if $c = \left\lfloor \frac{\alpha}{\beta} t \right\rfloor$, then the delay between the c-clocks of any two arbitrary far processes is at most 1 (Lemma 14). In at most $\mathcal{D} + 1$ additional rounds, each process executes $\mathcal{SU}$-N (Lemma 11) and updates its c-variable according to its t-variable. Hence, in at most $n + (\mu + 1)\mathcal{D} + 1$ rounds, the system reaches a legitimate configuration. □

Theorem 10 *After convergence of $\mathcal{SU}$ to $\mathcal{L}_{\mathsf{SU}}$, each process p increments its clock $p.c$ at least once every $\mathcal{D} + \frac{\beta}{\alpha}$ rounds, where $\mathcal{D}$ is the diameter of the network.*

Proof. If $\mathcal{DSU}$ converged to $\mathcal{L}_{\mathsf{SU}}$, by Remark 4 and Lemma 11, after $\mathcal{D} + \frac{\beta}{\alpha}$ rounds, p increments $p.t$ at least $\frac{\beta}{\alpha}$ times. Now, by Remark 6, if a t-variable is incremented $\frac{\beta}{\alpha}$ times, then its corresponding c-variable is incremented once. □

7 Gradually Stabilizing Strong Unison

We now propose Algorithm $\mathcal{DSU}$ (Algorithm 3), a gradually stabilizing variant of Algorithm $\mathcal{SU}$. First, to maintain a finite period for internal clocks, we need to assume that the number of processes in any reachable configuration never exceeds some bound $N \geq n$. Indeed, in compliance with Algorithm $\mathcal{SU}$, the parameter μ in Algorithm $\mathcal{DSU}$ should be fixed to a value greater than or equal to the maximum between 2 and N. Then, according to the results shown in Section 5 (Theorems 1-2 and Corollary 1), we consider the following dynamic pattern:

$$\mathsf{BULCC}(G_i, G_j) \equiv |V_j| \leq N \wedge (\alpha > 3 \Rightarrow \mathsf{ULC}(G_i, G_j)) \wedge \mathsf{C}(G_i, G_j)$$

BULCC stands for *Bounded number of nodes, Under Local Control, and Connected*. Precisely, after any BULCC-dynamic step (such a step may include several topological events) from a configuration of $\mathcal{L}_{\mathsf{SU}}$ (the set of legitimate configurations *w.r.t.* strong unison) $\mathcal{DSU}$ maintains (external) clocks almost synchronized during the convergence to strong unison, since it immediately satisfies partial weak unison, then converges in at most one round to weak unison, and finally re-stabilizes to strong unison.

In the following, we present in Subsection 7.1 the general principles of the convergence of $\mathcal{DSU}$ after a BULCC-dynamic step occurs from a configuration of $\mathcal{L}_{\mathsf{SU}}$. Then, we show the gradual stabilization of $\mathcal{DSU}$ in Subsection 7.2.

Algorithm 3 $\mathcal{DSU}$, for every process p

Parameters:

α: any positive integer such that $\alpha \geq 2$
μ: any positive integer such that $\mu \geq \max(N, 2)$
β: any positive integer such that $\beta > \mu^2$, and $\exists K$ such that $K > \mu$ and $\beta = K\alpha$

Variables:

$p.c \in \{0, \ldots, \alpha - 1\} \cup \{\bot\}$
$p.t \in \{0, \ldots, \beta - 1\}$

Predicates:

$$
\begin{array}{lcl}
Locked_p & \equiv & p.c = \bot \vee \exists q \in p.\mathcal{N}, q.c = \bot \\
NormalStep_p & \equiv & \neg Locked_p \wedge \forall q \in p.\mathcal{N}, p.t \preceq_{\beta,\mu} q.t \\
ResetStep_p & \equiv & \neg Locked_p \wedge \big(\exists q \in p.\mathcal{N}, d_\beta\big(p.t, q.t\big) > \mu \wedge\ p.t \neq 0\big) \\
JoinStep_p & \equiv & p.c = \bot
\end{array}
$$

Actions:

$$
\begin{array}{lcll}
\mathcal{DSU}\text{-}N & :: & NormalStep_p & \rightarrow \quad p.t \leftarrow (p.t + 1) \mod \beta \\
 & & & \quad\ \ p.c \leftarrow \left\lfloor \frac{\alpha}{\beta} p.t \right\rfloor \\
\mathcal{DSU}\text{-}R & :: & ResetStep_p & \rightarrow \quad p.t \leftarrow 0 \\
 & & & \quad\ \ p.c \leftarrow 0 \\
\mathcal{DSU}\text{-}J & :: & JoinStep_p & \rightarrow \quad p.t \leftarrow MinTime_p \\
 & & & \quad\ \ p.c \leftarrow \left\lfloor \frac{\alpha}{\beta} p.t \right\rfloor \\
bootstrap & :: & join_p & \rightarrow \quad p.t \leftarrow 0 \\
 & & & \quad\ \ p.c \leftarrow \bot
\end{array}
$$

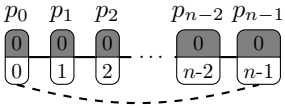


Figure 6: Link addition.

7.1 Algorithm $\mathcal{DSU}$

We now explain step by step how we modify Algorithm $\mathcal{SU}$ to obtain our gradually stabilizing algorithm, $\mathcal{DSU}$. We consider any BULCC-dynamic step $\gamma_i \mapsto_d^{\mathsf{BULCC}} \gamma_{i+1}$ such that $\gamma_i \in \mathcal{L}_{\mathsf{SU}}$, *i.e.*, the set of legitimate configurations *w.r.t.* strong unison. Since, $\mathcal{L}_{\mathsf{SU}}$ is closed (Lemma 12), the set of configurations reachable from $\mathcal{L}_{\mathsf{SU}}$ after one BULCC-dynamic step (which may also include process activations) is the same as the one reachable from $\mathcal{L}_{\mathsf{SU}}$ after BULCC-dynamic step made of topological events only (see Theorem 13, page 42). At the light of this result, we consider, without the loss of generality, no process moves during $\gamma_i \mapsto_d^{\mathsf{BULCC}} \gamma_{i+1}$.

Assume first that $\gamma_i \mapsto_d^{\mathsf{BULCC}} \gamma_{i+1}$ contains link additions only. Adding a link (see the dashed link in Figure 6) can break the safety of weak unison on internal clocks. Indeed, it may create a delay greater than one between two new neighboring t-clocks. Nevertheless, the delay between any two t-clocks remains bounded by $n - 1$ (recall that n is the number of processes initially in

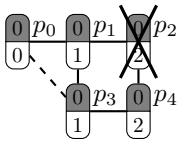
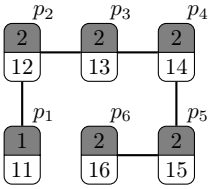
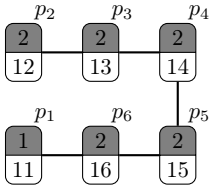


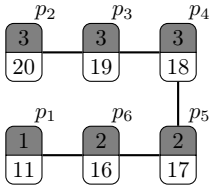
Figure 7: Removals.



(a) Initial configuration satisfying strong unison.



(b) After one dynamic step: link $\{p_1, p_2\}$ disappears and link $\{p_1, p_6\}$ is created.



(c) Some steps later, strong unison is violated.

Figure 8: Example of execution where one link is added and another is removed: $\mu = 6$, $\alpha = 7$, and $\beta = 42$.

the network), consequently, no process will reset its t-clock (Figure 6 shows a worst case). Moreover, c-clocks still satisfy strong unison immediately after the link addition. Besides, since increments are constrained by neighboring clocks, adding links only reinforces those constraints. Thus, the delay between internal clocks of arbitrary far processes remains bounded by $n-1$, and so strong unison remains satisfied, in all subsequent static steps. Consider again the example in Figure 6: before $\gamma_i \mapsto_d^{\mathsf{BULCC}} \gamma_{i+1}$, p_{n-1} had only to wait until p_{n-2} increments $p_{n-2}.t$ in order to be able to increment its own t-clock; yet after the step, it also has to wait for p_0 until its internal clock reaches at least $n-1$.

Assume now that $\gamma_i \mapsto_d^{\mathsf{BULCC}} \gamma_{i+1}$ contains process and link removals only. By definition of BULCC, the network remains connected. Hence, constraints between (still existing) neighbors are maintained: the delay between t-clocks of two neighbors remains bounded by one, see the example in Figure 7: process p_2 and link $\{p_0, p_3\}$ are removed. So, weak unison on t-clocks remains satisfied and so is strong unison on c-clocks.

Consider now a more complex scenario, where $\gamma_i \mapsto_d^{\mathsf{BULCC}} \gamma_{i+1}$ contains link additions as well as process and/or link removals. Figure 8 shows an example of such a scenario, where safety of strong unison is violated. As above, the addition of link $\{p_1, p_6\}$ in Figure 8(b) leads to a delay between t-clocks of these two (new) neighbors which is greater than one (here 5). However, the removal of link $\{p_1, p_2\}$, also in Figure 8(b), relaxes the neighborhood constraint on p_2: p_2 can now increment without waiting for p_1. Consequently, executing Algorithm $\mathcal{SU}$ does not ensure that the delay between t-clocks of any two arbitrary far processes remains bounded by $n-1$, *e.g.*, after several static steps from Figure 8(b), the system can reach Figure 8(c), where the delay between p_1 and p_2 is 9 while

$n-1=5$. Since c-clock values are computed from t-clock values, we also cannot guarantee that there is at most two consecutive c-clock values in the system, *e.g.*, in Figure 8(c) we have: $p_1.c=1$, $p_6.c=2$, and $p_2.c=3$. Again, in the worst case scenario, after $\gamma_i \mapsto_d^{\mathsf{BULCC}} \gamma_{i+1}$, the delay between two neighboring t-clocks is bounded by $n-1$. Moreover, t-clocks being computed like in Algorithm $\mathcal{WU}$, we can use two of its useful properties (see [17]): (1) when the delay between every pair of neighboring t-clocks is at most μ with $\mu \geq n$, the delay between these clocks remains bounded by μ because processes never reset; (2) furthermore, from such configurations, the system converges to a configuration from which the delay between the t-clocks of every two neighbors is at most one. So, keeping $\mu \geq n$, processes will not reset after that BULCC-dynamic step and the delay between any two neighboring t-clocks will monotonically decrease from at most $n-1$ to at most one. Consequently, the delay between any two neighboring c-clocks (which are computed from t-clocks) will stay at most one, *i.e.*, weak unison will be satisfied all along the convergence to strong unison.

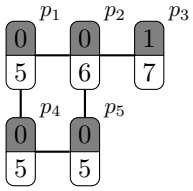
Assume now that a process p joins the system during $\gamma_i \mapsto_d^{\mathsf{BULCC}} \gamma_{i+1}$. The event $join_p$ occurs and triggers the new specific action *bootstrap* that initialized p to its bootstate: p sets $p.c$ to a specific *bootstate* value, noted $\perp$ (meaning that its output is currently undefined), and $p.t$ to 0. By definition and from the previous discussion, the system immediately satisfies partial unison since it only depends on processes that were in the system before the BULCC-dynamic step. Now, to ensure that weak unison holds within a round, we add the action $\mathcal{DSU}$-J which is enabled as soon as the process is in bootstate. This action initializes the two clocks of p according to the minimum t-clock value of its neighbors that are not in bootstate, if any. To that goal, we use the function $MinTime_p$ given below.

$$\begin{aligned} MinTime_p \;=\; & 0 \textbf{ if } \forall q \in p.\mathcal{N}, q.c = \perp\textbf{;} \\ & \min\{q.t : q \in p.\mathcal{N} \wedge q.c \neq \perp\} \textbf{ otherwise.} \end{aligned}$$

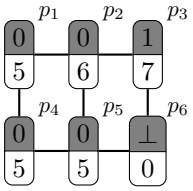
The value of $p.c$ is then computed according to the value of $p.t$. Remark that $MinTime_p$ returns 0 when p and all its neighbors have their respective c-clock equal to $\perp$: if the BULCC-dynamic step replaces all nodes by new ones, then the system reaches in a configuration where all c-clocks are equal to $\perp$, and $\mathcal{DSU}$ still ensures gradual stabilization in this case.

Then, to prevent the unfair daemon from blocking the convergence to a configuration containing no $\perp$-values, we should also forbid processes with non-$\perp$ c-clock values to increment while there are c-clocks with $\perp$-values in their neighborhood. So, we define the predicate *Locked* which holds for a given process p when either $p.c=\perp$, or at least one of its neighbors q satisfies $q.c=\perp$. We then enforce the guard of both normal and reset actions, so that no *Locked* process can execute them. See actions $\mathcal{DSU}$-N and $\mathcal{DSU}$-R. This ensures that t-clocks are initialized first by Action $\mathcal{DSU}$-J, before any value in their neighborhood increments.

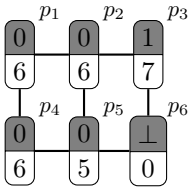
Finally, notice that all the previous explanation relies on the fact that, once the system recovers from process additions (*i.e.*, once no $\perp$ value remains), the



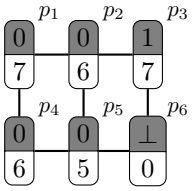
(a) Initial configuration satisfying strong unison.



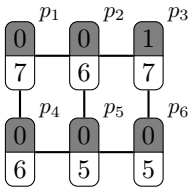
(b) After the dynamic step where process p_6 joins, p_3 and p_5 are locked. p_1 and p_4 are enabled to execute $\mathcal{DSU}$-N.



(c) p_4 is disabled. $\mathcal{DSU}$-N is enabled at p_1.



(d) Now, p_1 is disabled because of p_2 and p_4. p_6 is the only enabled process.



(e) p_6 executes $\mathcal{DSU}$-J and initializes its clocks.

Figure 9: Example of execution where the daemon delays the first step of a new process: $\mu = 6$, $\alpha = 6$, and $\beta = 42$.

algorithm behaves exactly the same as Algorithm $\mathcal{SU}$. Hence, it has to match the assumptions made for $\mathcal{SU}$, in particular, the ones on α and β. However the constraint on μ has to be adapted, since μ should be greater than or equal to the actual number of processes in the network and this number may increase. Now, this number is assumed to be bounded by N. So, we now require that $\mu \geq \max(N, 2)$.

We now consider the example execution of Algorithm $\mathcal{DSU}$ in Figure 9. This execution starts in a configuration legitimate *w.r.t.* the strong unison, see Figure 9(a). Then, one BULCC-dynamic step happens (step (a)$\mapsto$(b)), where a process p_6 joins the system. We now try to delay as long as possible the execution of $\mathcal{DSU}$-J by p_6. In configuration (b), p_3 and p_5, the new neighbors of p_6, are locked. They will remain disabled until p_6 executes $\mathcal{DSU}$-J. p_1 and p_4 execute $\mathcal{DSU}$-N in (b)$\mapsto$(c). Then, p_4 is disabled because of p_5 and p_1 executes $\mathcal{DSU}$-N in (c)$\mapsto$(d). In configuration (d), p_1 is from now on disabled: p_1 must wait until p_2 and p_4 get t-clock value 7. p_6 is the only enabled process, so the unfair daemon has no other choice but selecting p_6 to execute $\mathcal{DSU}$-J in the next step.

7.2 Correctness

7.2.1 Self-stabilization *w.r.t.* SP_{SU}

Remark 9 *By definition, $N \geq n$, so, whenever the parameters α, μ, and β satisfy the constraints of Algorithm $\mathcal{DSU}$, they also satisfy all constraints of Algorithm $\mathcal{SU}$.*

Remark 10 *In $\mathcal{DSU}$, if all c-variables have values different from $\perp$, predicates JoinStep and Locked are false. Furthermore, no action can assign $\perp$ to c during a static step. Consequently, when all c-variables have values different from $\perp$, and as far as no topological change occurs, Algorithms $\mathcal{DSU}$ and $\mathcal{SU}$ are syntactically identical. So, fixing the initial graph and the three parameters α, μ, and β, we obtain that the set of executions $\mathcal{E}^0$ of $\mathcal{SU}$ and the set of executions $\mathcal{E}^0_{NoBot}$ of $\mathcal{DSU}$, where $NoBot = \{\gamma \in \mathcal{C} \ : \ \forall p \in V, \gamma(p.c) \neq \perp\}$, are identical.*

By Remarks 10 and 9, results of Algorithm $\mathcal{SU}$ about $\mathcal{L}_{\mathsf{SU}}$ (see Definition 5 page 29) also hold for Algorithm $\mathcal{DSU}$. Hence, follows.

Lemma 18 (Closure and Correctness of $\mathcal{L}_{\mathsf{SU}}$ under $\mathcal{DSU}$) *$\mathcal{L}_{\mathsf{SU}}$ is closed under $\mathcal{DSU}$, and for every execution $e \in \mathcal{E}^0_{\mathcal{L}_{\mathsf{SU}}}$ under $\mathcal{DSU}$, $SP_{\mathsf{SU}}(e)$.*

Lemma 19 *For any execution $(\gamma_i)_{i\geq 0} \in \mathcal{E}^0$ under $\mathcal{DSU}$, $\exists j \geq 0$ such that $\forall k \geq j$, $\forall p \in V$, $\gamma_k(p).c \neq \perp$.*

Proof.

Let $e = (\gamma_i)_{i\geq 0} \in \mathcal{E}^0$. For any $i \geq 0$, we note $Bottom(\gamma_i) = \{p \in V : \gamma_i(p).c = \perp\}$. As actions $\mathcal{DSU}$-N, $\mathcal{DSU}$-R and $\mathcal{DSU}$-J do not create any $\perp$-value, $\forall i > 0$, $Bottom(\gamma_i) \subseteq Bottom(\gamma_{i-1})$. Now, assume by contradiction that $\exists p \in V$ such that $\forall i \geq 0$, $p \in Bottom(\gamma_i)$. Since, the number of nodes is constant, there is a configuration γ_s, $s \geq 0$, from which no $\perp$-value disappears anymore, *i.e.*, $\forall p \in V$, $p \in Bottom(\gamma_s) \Rightarrow \forall i \geq s, p \in Bottom(\gamma_i)$.

If $Bottom(\gamma_s) = V$, every process is enabled for $\mathcal{DSU}$-J. So, the unfair daemon selects at least one process to execute $\mathcal{DSU}$-J and sets its c-variable to a value different from $\perp$, a contradiction with the definition of γ_s.

Hence there is at least one process that is not in $Bottom(\gamma_s)$. Again, if the only enabled processes are in $Bottom(\gamma_s)$, then the unfair daemon has no other choice but selecting one of them, a contradiction. So, $\forall i \geq s$, there exists a process that is enabled in γ_i but which is not in $Bottom(\gamma_i)$. Remark that this implies in particular that e is an infinite execution.

Now, let consider the subgraph G' of G induced by $V \backslash Bottom(\gamma_s)$. G' is composed of a finite number of connected components and, as e is infinite, there is an infinite number of actions of e executed in (at least) one of these components. Let $G'' = (V'', E'')$ be such a connected component.

Let $e' = (\gamma'_i)_{i\geq 0}$ be the projection of e on G'' and t-variable: $\forall i \geq 0, \forall x \in V''$, $\gamma'_i(x).t = \gamma_i(x).t$. We construct $e'' = (\gamma''_j)_{j\geq 0}$ from e' by removing duplicate configurations with the following inductive schema:

- $\gamma''_0 = \gamma'_0$,
- and, $\forall j > 0$, if $\gamma''_0 \dots \gamma''_j$ represents $\gamma'_0 \dots \gamma'_k$ without duplicate configurations, $\gamma''_{j+1} = \gamma'_{next}$, where $next = \min\{l > k : \gamma'_l \neq \gamma'_k\}$. (Notice that $next$ is always defined as there is an infinite number of actions executed in G''.)

Let $L = \{p \in V'' : \exists q \in Bottom(\gamma_s), \{p, q\} \in E\}$ be the set of processes that are neighbors of a $Bottom(\gamma_s)$ process in G. As G is connected, L is not empty. Furthermore, during the execution e, *Locked* holds forever for processes in L, hence are disabled. As a consequence, in execution e'', no process in L can execute a static step. Now, from Remark 4 and 10, and since γ''_0 contains no $\perp$-value, e'' is also an execution of $\mathcal{WU}$ in graph G''. The fact that existing processes (from the non-empty set L) never increment their clocks during an infinite execution e'' of $\mathcal{WU}$ is a contradiction with the liveness of the weak unison (Specification 2), Remark 9, and Theorem 5 which states that $\mathcal{WU}$ is self-stabilizing for weak unison under an unfair daemon. □

Lemma 20 (Convergence to $\mathcal{L}_{\mathsf{SU}}$) *$\mathcal{C}$ (the set of all possible configurations) converges under $\mathcal{DSU}$ to the set of legitimate configurations $\mathcal{L}_{\mathsf{SU}}$.*

Proof. Let $(\gamma_i)_{i\geq 0} \in \mathcal{E}^0$ under $\mathcal{DSU}$. Using Lemma 19, $\exists j \geq 0$ such that $\forall k \geq j$, $\forall p \in V$, $\gamma_k(p).c \neq \perp$. After γ_j, the execution of the system, $(\gamma_k)_{k\geq j}$, is also a possible execution of $\mathcal{SU}$ (by Remark 10). Hence, it converges to a configuration γ_k $(k \geq j)$ in $\mathcal{L}_{\mathsf{SU}}$, by Remark 9 and Lemma 13. □

Using Lemmas 18 and 20, we can deduce the following theorem.

Theorem 11 (Self-stabilization of $\mathcal{DSU}$ *w.r.t.* strong unison) *Algorithm $\mathcal{DSU}$ is self-stabilizing for SP_{SU} in any arbitrary connected anonymous network assuming a distributed unfair daemon.*

Theorem 12 states the stabilization time of $\mathcal{DSU}$.

Theorem 12 *The stabilization time of $\mathcal{DSU}$ to $\mathcal{L}_{\mathsf{SU}}$ is at most $n+(\mu+1)\mathcal{D}+2$, where n (resp. $\mathcal{D}$) is the size (resp. diameter) of the network, and μ is a parameter satisfying $\mu \geq \max(2, N)$.*

Proof. Let $(\gamma_i)_{i\geq 0} \in \mathcal{E}^0$. If there are some processes p such that $\gamma_0(p).c = \perp$, $\mathcal{DSU}$-J is continuously enabled at p. So, after at most one round $p.c \neq \perp$, for every process p. Afterwards, the behavior of the algorithm is identical to the one of $\mathcal{SU}$ (Remarks 9 and 10), which stabilizes in at most $n + (\mu + 1)\mathcal{D} + 1$ rounds (see Theorem 9). Hence, in at most $n+(\mu+1)\mathcal{D}+2$ rounds, the system reaches a legitimate configuration. □

7.2.2 Immediate Stabilization to SP_{PU} after one BULCC-Dynamic Step from $\mathcal{L}_{\mathsf{SU}}$

Definition 6 (Legitimate Configurations of $\mathcal{DSU}$ *w.r.t.* SP_{PU}) *A configuration γ_i of $\mathcal{DSU}$ is* legitimate w.r.t. SP_{PU} if and only if

1. $\forall p \in V_i,\ \gamma_i(p).c \neq \bot \Rightarrow \gamma_i(p).c = \left\lfloor \frac{\alpha}{\beta}\gamma_i(p).t \right\rfloor$; *and*
2. *if $\alpha > 3$, then the following three additional conditions hold:*
   (a) $\big(\forall p \in V_i, (\gamma_i(p).c = \bot \Rightarrow (\forall q \in \gamma_i(p).\mathcal{N}, \gamma_i(q).c \in \{0, \bot\}))\big) \vee$
   $\big(\forall p \in V_i, (\gamma_i(p).c = \bot \Rightarrow (\exists q \in \gamma_i(p).\mathcal{N}, \gamma_i(q).c \neq \bot))\big)$
   (b) $\forall p \in V_i, \forall q \in \gamma_i(p).\mathcal{N}$,
   $$\gamma_i(p).c \neq \bot \wedge \gamma_i(q).c \neq \bot \Rightarrow d_\beta\big(\gamma_i(p).t, \gamma_i(q).t\big) \leq \mu;$$
   (c) $\forall p, q \in V_i$,
   $$\gamma_i(p).c \neq \bot \wedge (\exists x \in \gamma_i(p).\mathcal{N}, \gamma_i(x).c = \bot) \wedge$$
   $$\gamma_i(q).c \neq \bot \wedge (\exists y \in \gamma_i(q).\mathcal{N}, \gamma_i(y).c = \bot) \Rightarrow$$
   $$d_\beta\big(\gamma_i(p).t, \gamma_i(q).t\big) \leq \mu.$$

We denote by $\mathcal{L}_{\mathsf{PU}}$ the set of legitimate configurations of $\mathcal{DSU}$ w.r.t. SP_{PU}.

Lemma 21 (Closure of $\mathcal{L}_{\mathsf{PU}}$ under $\mathcal{DSU}$) *$\mathcal{L}_{\mathsf{PU}}$ is closed under $\mathcal{DSU}$.*

Proof. Let $\gamma_i \mapsto_s \gamma_{i+1}$ be a static step of $\mathcal{DSU}$ such that $\gamma_i \in \mathcal{L}_{\mathsf{PU}}$. By definition, $\mathcal{DSU}$-R is disabled in γ_i, for all processes: a process can only execute $\mathcal{DSU}$-N or $\mathcal{DSU}$-J depending whether its c-clock is $\bot$ or not.

Point 1 of Definition 6. Let $p \in V_{i+1}$ such that $\gamma_{i+1}(p).c \neq \bot$. Two cases are possible: either p executes no action during $\gamma_i \mapsto_s \gamma_{i+1}$ and the constraint between $p.t$ and $p.c$ has been preserved, or p executes $\mathcal{DSU}$-J or $\mathcal{DSU}$-N. In the latter case, the assignment of the action ensures the constraint.

For the three next points, we assume that $\alpha > 3$ (otherwise, those constraints are trivially preserved).

Point 2a of Definition 6. Since $\gamma_i \in \mathcal{L}_{\mathsf{PU}}$, two cases are possible, by Definition 6.2a.

Assume $\forall p \in V_i, (\gamma_i(p).c = \bot \Rightarrow (\forall q \in \gamma_i(p).\mathcal{N}, \gamma_i(q).c \in \{0, \bot\}))$. Neither $\mathcal{DSU}$-N nor $\mathcal{DSU}$-J sets c to $\bot$. Then, let $p \in V_i$ such that $\gamma_i(p).c = \bot$. Let q be a neighbor of p in γ_i. If $\gamma_i(q).c = 0$, then $\gamma_{i+1}(q).c = 0$, since q is disabled ($Locked_q$ holds in γ_i because of p). If $\gamma_i(q).c = \bot$, then $\gamma_{i+1}(q).c \in \{0, \bot\}$ depending on whether or not q executes $\mathcal{DSU}$-J, since the c-clock values of all its neighbors are 0 or $\bot$, by hypothesis, and since the c-clock values of its non-$\bot$ neighbors are well computed according to their t-clock values, by Definition 6.1. Hence, the constraint is preserved in this case, and we are done.

Otherwise, $\forall p \in V_i, (\gamma_i(p).c = \bot \Rightarrow (\exists q \in \gamma_i(p).\mathcal{N}, \gamma_i(q).c \neq \bot))$. Since neither $\mathcal{DSU}$-N nor $\mathcal{DSU}$-J sets $p.c$ to $\bot$, this constraint is also preserved in this case.

Point 2b of Definition 6. Let p, q be two neighbors such that $\gamma_{i+1}(p).c \neq \bot$ and $\gamma_{i+1}(q).c \neq \bot$.

1. Assume that $\gamma_i(p).c \neq \bot$ and $\gamma_i(q).c \neq \bot$. As $\gamma_i \in \mathcal{L}_{\mathsf{PU}}$, we have $d_\beta\big(\gamma_i(p).t, \gamma_i(q).t\big) \leq \mu$, by Definition 6.2b. So, p and q can only execute $\mathcal{DSU}$-N during $\gamma_i \mapsto_s \gamma_{i+1}$. If both p and q, or none of them, execute $\mathcal{DSU}$-N, the delay between $p.t$ and $q.t$ remains the same. If only one of them, say p, executes $\mathcal{DSU}$-N, $\gamma_i(p).t \preceq_{\beta,\mu} \gamma_i(q).t$. So, either $\gamma_i(p).t = \gamma_i(q).t$ and $d_\beta\big(\gamma_{i+1}(p).t, \gamma_{i+1}(q).t\big) = 1 \leq \mu$, or the increment of $p.t$ decreases the delay between $p.t$ and $q.t$ and again we have $d_\beta\big(\gamma_{i+1}(p).t, \gamma_{i+1}(q).t\big) \leq \mu$.
2. Assume that $\gamma_i(p).c = \bot$ and $\gamma_i(q).c \neq \bot$. Let x be the neighbor of p in γ_i such that $\gamma_i(x).c \neq \bot$ with the minimum t-value (x is defined because $\gamma_i(q).c \neq \bot$). Since q and x are both neighbors of p in γ_i and $\gamma_i \in \mathcal{L}_{\mathsf{PU}}$, $d_\beta\big(\gamma_i(x).t, \gamma_i(q).t\big) \leq \mu$, by Definition 6.2c. Moreover, q is disabled in γ_i because of p ($Locked_q$ holds in γ_i), so $\gamma_{i+1}(q).t = \gamma_i(q).t$. Necessarily, p executes $\mathcal{DSU}$-J in $\gamma_i \mapsto_s \gamma_{i+1}$, since $\gamma_{i+1}(p).c \neq \bot$. Hence, $\gamma_{i+1}(p).t = \gamma_i(x).t$ and $d_\beta\big(\gamma_{i+1}(p).t, \gamma_{i+1}(q).t\big) \leq \mu$.
3. Assume that $\gamma_i(p).c \neq \bot$ and $\gamma_i(q).c = \bot$. This case is similar to the previous one.
4. Assume that $\gamma_i(p).c = \bot$ and $\gamma_i(q).c = \bot$. As $\gamma_{i+1}(p).c \neq \bot$ and $\gamma_{i+1}(q).c \neq \bot$, p and q necessarily move during $\gamma_i \mapsto_s \gamma_{i+1}$. Since $\gamma_i \in \mathcal{L}_{\mathsf{PU}}$, two cases are possible, by Definition 6.2a.

   If $\forall v \in V_i, (\gamma_i(v).c = \bot \Rightarrow (\forall w \in \gamma_i(v).\mathcal{N}, \gamma_i(w).c \in \{0, \bot\}))$, then $\gamma_{i+1}(p).c = \gamma_{i+1}(q).c = 0$ (owing the fact that the c-clock values of all their non-$\bot$ neighbors are well computed according to their t-clock values, by Definition 6.1), and we are done.

   Otherwise, $\forall v \in V_i, (\gamma_i(v).c = \bot \Rightarrow (\exists w \in \gamma_i(v).\mathcal{N}, \gamma_i(w).c \neq \bot))$. So, in γ_i, we have: $\exists x \in p.\mathcal{N}$ such that $\gamma_i(x).c \neq \bot \wedge \gamma_i(x).t = MinTime_p$ and $\exists y \in q.\mathcal{N}$ such that $\gamma_i(y).c \neq \bot \wedge \gamma_i(y).t = MinTime_q$, by Definition 6.2a. Moreover, x and y have neighbors whose c-variables equal $\bot$ (p and q, respectively), we have $d_\beta\big(\gamma_i(x).t, \gamma_i(y).t\big) \leq \mu$, by Definition 6.2c. Since p and q execute $\mathcal{DSU}$-J, $\gamma_{i+1}(p).t = \gamma_i(x).t$ and $\gamma_{i+1}(q).t = \gamma_i(y).t$, and we are done.

Point 2c of Definition 6. Let p, q be two processes such that $\gamma_{i+1}(p).c \neq \bot$, $\exists x \in \gamma_{i+1}(p).\mathcal{N}$ with $\gamma_{i+1}(x).c = \bot$, $\gamma_{i+1}(q).c \neq \bot$, and $\exists y \in \gamma_{i+1}(q).\mathcal{N}$ with $\gamma_{i+1}(y).c = \bot$.

As no enabled action can set variable c to $\bot$, $\gamma_i(x).c = \bot$ and $\gamma_i(y).c = \bot$.

1. Assume that $\gamma_i(p).c \neq \bot$ and $\gamma_i(q).c \neq \bot$. As $\gamma_i \in \mathcal{L}_{\mathbf{PU}}$, $d_\beta\big(\gamma_i(p).t, \gamma_i(q).t\big) \leq \mu$, by Definition 6.2b. Now, p and q are disabled in γ_i because of x and y ($Locked_p$ and $Locked_q$ hold in γ_i). Hence, $d_\beta\big(\gamma_{i+1}(p).t, \gamma_{i+1}(q).t\big) \leq \mu$.

2. Assume that $\gamma_i(p).c = \bot$ and $\gamma_i(q).c \neq \bot$. Since $\gamma_i \in \mathcal{L}_{\mathbf{PU}}$, two cases are possible, by Definition 6.2a.

   Assume $\forall v \in V_i, (\gamma_i(v).c = \bot \Rightarrow (\forall w \in \gamma_i(v).\mathcal{N}, \gamma_i(w).c \in \{0, \bot\}))$. Then, since $\gamma_i(y).c = \bot$, $\gamma_i(q).c = 0$. Then, $\gamma_{i+1}(p).c = \gamma_{i+1}(q).c = 0$ (p necessarily moves in $\gamma_i \mapsto_s \gamma_{i+1}$ and gets clock 0 owing the fact that the c-clock values of all its non-$\bot$ neighbors are well computed according to their t-clock values, by Definition 6.1; moreover q is disabled in γ_i since $Locked_q$ hold because of y), and we are done.

   Otherwise, $\forall v \in V_i, (\gamma_i(v).c = \bot \Rightarrow (\exists w \in \gamma_i(v).\mathcal{N}, \gamma_i(w).c \neq \bot))$. Then, $\exists x' \in \gamma_i(p).\mathcal{N}$ such that $\gamma_i(x').c \neq \bot \wedge \gamma_i(x').t = MinTime_p$ in γ_i. By Definition 6.2c, $d_\beta\big(\gamma_i(x').t, \gamma_i(q).t\big) \leq \mu$ because they have neighbors whose c-variables equal $\bot$ (p and y, respectively). Moreover, q is disabled in γ_i because of y: $\gamma_{i+1}(q).t = \gamma_i(q).t$. Finally, $\gamma_{i+1}(p).t = \gamma_i(x').t$ since p executes $\mathcal{DSU}$-J. So, $d_\beta\big(\gamma_{i+1}(p).t, \gamma_{i+1}(q).t\big) \leq \mu$.

3. Assume that $\gamma_i(p).c \neq \bot$ and $\gamma_i(q).c = \bot$. The case is similar to the previous one.

4. Assume that $\gamma_i(p).c = \bot$ and $\gamma_i(q).c = \bot$. Since $\gamma_i \in \mathcal{L}_{\mathbf{PU}}$, two cases are possible, by Definition 6.2a.

   If $\forall v \in V_i, (\gamma_i(v).c = \bot \Rightarrow (\forall w \in \gamma_i(v).\mathcal{N}, \gamma_i(w).c \in \{0, \bot\}))$, then $\gamma_{i+1}(p).c = \gamma_{i+1}(q).c = 0$ (owing the fact that the c-clock values of all their non-$\bot$ neighbors are well computed according to their t-clock values, by Definition 6.1), and we are done.

   Otherwise, $\forall v \in V_i, (\gamma_i(v).c = \bot \Rightarrow (\exists w \in \gamma_i(v).\mathcal{N}, \gamma_i(w).c \neq \bot))$. So, $\exists x' \in \gamma_i(p).\mathcal{N}$ such that $\gamma_i(x').c \neq \bot \wedge \gamma_i(x').t = MinTime_p$ in γ_i and $\exists y' \in \gamma_i(q).\mathcal{N}$ such that $\gamma_i(y').c \neq \bot \wedge \gamma_i(y').t = MinTime_q$ in γ_i. By Definition 6.2c, $d_\beta\big(\gamma_i(x').t, \gamma_i(y').t\big) \leq \mu$ because they have neighbors whose c-variables equal $\bot$ (p and q, respectively). $\gamma_{i+1}(p).t = \gamma_i(x').t$ and $\gamma_{i+1}(q).t = \gamma_i(y').t$ since p and q execute $\mathcal{DSU}$-J. So $d_\beta\big(\gamma_{i+1}(p).t, \gamma_{i+1}(q).t\big) \leq \mu$.

□

Before going into details, we show the following theorem, which allows to simplify proofs and explanations.

Theorem 13 *Let X be a closed set of configurations. Let ρ be any dynamic pattern.*

$$\forall \gamma_i \in \mathcal{C}, (\exists \gamma_j \in X\ \gamma_j \mapsto^\rho_d \gamma_i) \Leftrightarrow (\exists \gamma_k \in X\ \gamma_k \mapsto^\rho_{d_{only}} \gamma_i)$$

where $\mapsto^\rho_{d_{only}}$ is the set of all ρ-dynamic steps containing no process activation.

Proof. Let $\gamma_i \in \mathcal{C}$ such that $\gamma_j \mapsto_d^\rho \gamma_i$ with $\gamma_j \in X$. If $\gamma_j \mapsto_{d_{only}}^\rho \gamma_i$, we are done. Otherwise, let A be the non-empty subset of processes that are activated in $\gamma_j \mapsto_d^\rho \gamma_i$. There exists $\gamma_j \mapsto_s \gamma_u$, where A is activated. As X is closed, $\gamma_u \in X$. Moreover, $\forall x \in G_j \cap G_i$, $x \in G_u$ (since $G_u = G_j$) and $\gamma_u(x) = \gamma_i(x)$. Let $\gamma_u \mapsto_{d_{only}}^\rho \gamma_k$ such that $G_k = G_i$. $\forall x \in G_j \cap G_i$, $x \in G_k$ (since $G_k = G_i$) and $\gamma_k(x) = \gamma_u(x) = \gamma_i(x)$. Moreover, $\forall x \in G_i \setminus G_j$, $x \in G_k$ (since $G_k = G_i$) and $\gamma_k(x) = \gamma_i(x)$ because in both cases, x is in bootstate. Hence, $\gamma_k = \gamma_i$, and we are done.

The second part of the assertion is trivial since, by definition, $\mapsto_{d_{only}}^\rho \subseteq \mapsto_d^\rho$. □

Lemma 22 *Let $\gamma_i \in \mathcal{L}_{\mathsf{SU}}$ and $\gamma_i \mapsto_d^{\mathsf{BULCC}} \gamma_{i+1}$. Then, $\gamma_{i+1} \in \mathcal{L}_{\mathsf{PU}}$.*

Proof. By Theorem 13 and as $\mathcal{L}_{\mathsf{SU}}$ is closed (Lemma 18), we can assume, without the loss of generality that, no process moves during $\gamma_i \mapsto_d^{\mathsf{BULCC}} \gamma_{i+1}$. Then, the lemma is obvious by Definition of BULCC, Remark 5, and Definitions 5-6. □

Lemma 23 (Safety of SP_{PU} in $\mathcal{E}^0_{\mathcal{L}_{\mathsf{PU}}}$) *Every execution $e \in \mathcal{E}^0_{\mathcal{L}_{\mathsf{PU}}}$ satisfies the safety of SP_{PU}.*

Proof. Let $\gamma_i \in \mathcal{L}_{\mathsf{PU}}$. If $\alpha \leq 3$, then by definition, for every two neighbors p and q in γ_i, we have $\big(\gamma_i(p).c \neq \bot \wedge \gamma_i(q).c \neq \bot\big) \Rightarrow d_\alpha\big(\gamma_i(p).c, \gamma_i(q).c\big) \leq 1$.

Assume now that $\alpha > 3$. By Definition 6.2b, for every two neighbors p and q in γ_i we have $\big(\gamma_i(p).c \neq \bot \wedge \gamma_i(q).c \neq \bot\big) \Rightarrow d_\beta\big(\gamma_i(p).t, \gamma_i(q).t\big) \leq \mu$. Furthermore, for every process p, $\gamma_i(p).c \neq \bot \Rightarrow \gamma_i(p).c = \left\lfloor \frac{\alpha}{\beta} \gamma_i(p).t \right\rfloor$. Hence, using Lemma 14 with $d = \mu < K$, $\forall p \in V_i$, $\forall q \in \gamma_i(p).\mathcal{N}$, $\big(\gamma_i(p).c \neq \bot \wedge \gamma_i(q).c \neq \bot\big) \Rightarrow d_\alpha\big(\gamma_i(p).c, \gamma_i(q).c\big) \leq 1$.

Finally, as the set $\mathcal{L}_{\mathsf{PU}}$ is closed (Lemma 21), we are done. □

By Lemmas 19 and 21, we have the following corollary.

Corollary 2 *$\mathcal{DSU}$ converges from $\mathcal{L}_{\mathsf{PU}}$ to $\mathcal{L}_{\mathsf{WU}}$ in a finite time.*

Lemma 24 (Liveness of SP_{PU} in $\mathcal{E}^0_{\mathcal{L}_{\mathsf{PU}}}$) *Every execution $e \in \mathcal{E}^0_{\mathcal{L}_{\mathsf{PU}}}$ satisfies the liveness of SP_{PU}.*

Proof. Let $e = (\gamma_i)_{i \geq 0} \in \mathcal{E}^0_{\mathcal{L}_{\mathsf{PU}}}$. By Corollary 2, there exists $i \geq 0$ such that $\gamma_i \in \mathcal{L}_{\mathsf{WU}}$. By Remarks 4, 9, and 10, we can apply Lemma 6: $p.t$ goes through each integer value between 0 and $\beta - 1$ infinitely often (in increasing order), for every process p. Hence, by Remark 6, for every process p, $p.c$ is incremented infinitely often and goes through each integer value between 0 and $\alpha - 1$ (in increasing order). □

By Lemmas 21-24, we can deduce the following theorem.

Theorem 14 *After one BULCC-dynamic step from a configuration of $\mathcal{L}_{\mathsf{SU}}$, $\mathcal{DSU}$ immediately stabilizes to SP_{PU} by $\mathcal{L}_{\mathsf{PU}}$.*

7.2.3 Stabilization from $\mathcal{L}_{\mathsf{PU}}$ to SP_{WU} by $\mathcal{L}_{\mathsf{WU}}$ in at most one Round

Lemma 25 *$\mathcal{DSU}$ converges from $\mathcal{L}_{\mathsf{PU}}$ to $\mathcal{L}_{\mathsf{WU}}$ in finite time. The convergence time is at most one round.*

Proof. The finite convergence time is proven by Corollary 2. Then, round complexity is trivial since every process in bootstate is continuously enabled to leave its bootstate by $\mathcal{DSU}$-J, and no process can set its c-clock to $\perp$ during a static step. □

By Remarks 4, 9, and 10, results of Algorithm $\mathcal{WU}$ about $\mathcal{L}_{\mathsf{WU}}$[9] also hold for Algorithm $\mathcal{DSU}$. Hence, follows.

Lemma 26 (Closure and Correctness of $\mathcal{L}_{\mathsf{WU}}$ under $\mathcal{DSU}$) *$\mathcal{L}_{\mathsf{WU}}$ is closed under $\mathcal{DSU}$, and for every execution $e \in \mathcal{E}^0_{\mathcal{L}_{\mathsf{WU}}}$ under $\mathcal{DSU}$, $SP_{\mathsf{WU}}(e)$.*

By Lemmas 25-26 and Theorem 14, follows.

Theorem 15 *After one BULCC-dynamic step from a configuration of $\mathcal{L}_{\mathsf{SU}}$, $\mathcal{DSU}$ stabilizes from $\mathcal{L}_{\mathsf{PU}}$ to SP_{WU} by $\mathcal{L}_{\mathsf{WU}}$ in finite time. The convergence time from $\mathcal{L}_{\mathsf{PU}}$ to $\mathcal{L}_{\mathsf{WU}}$ is at most one round.*

7.2.4 Gradual Stabilization after one BULCC-Dynamic Step from $\mathcal{L}_{\mathsf{SU}}$

Lemma 27 *The convergence time from $\mathcal{L}_{\mathsf{WU}}$ to $\mathcal{L}_{\mathsf{SU}}$ is at most $(\mu+1)\mathcal{D}_1+1$ rounds, where $\mathcal{D}_1$ is the diameter of the network after the dynamic step and μ is a parameter satisfying $\mu \geq \max(2, N)$.*

Proof. Let $e = (\gamma_i)_{i\geq 0} \in \mathcal{E}^0_{\mathcal{L}_{\mathsf{WU}}}$. The behavior of the algorithm is similar to the one of $\mathcal{WU}$ (Remarks 4, 9, and 10). By Theorem 7, within at most $\mu\mathcal{D}_1$ rounds the system reaches a configuration from which $\forall q \in p.\mathcal{N}$, $d_\beta\big(p.t, q.t\big) \leq 1$ forever, provided that no dynamic step occurs. By Lemma 11, each process increments its clock within at most $\mathcal{D}_1+1$ additional rounds, from that point the c-variables are well computed according to t-variables. Hence, in at most $(\mu+1)\mathcal{D}_1+1$ rounds, the system reaches $\mathcal{L}_{\mathsf{SU}}$. □

By Theorems 11-15 and Lemma 27, follows.

Theorem 16 *$\mathcal{DSU}$ is gradually stabilizing under $(1, \mathsf{BULCC})$-dynamics for*

$$(SP_{\mathsf{PU}} \bullet 0, SP_{\mathsf{WU}} \bullet 1, SP_{\mathsf{SU}} \bullet (\mu+1)\mathcal{D}_1 + 2)$$

where $\mathcal{D}_1$ is the diameter of the network after the dynamic step and μ is a parameter satisfying $\mu \geq \max(2, N)$.

Theorem 17 establishes a bound on how many rounds are necessary to ensure that a given process increments its c-clock after the convergence to legitimate configurations *w.r.t.* SP_{SU} (resp. SP_{WU}).

[9] See Theorem 5.

Theorem 17 *After convergence of $\mathcal{DSU}$ to $\mathcal{L}_{\mathsf{WU}}$ (resp. $\mathcal{L}_{\mathsf{SU}}$), each process p increments its clock $p.c$ at least once every $\mu\mathcal{D}_1 + \frac{\beta}{\alpha}$ rounds (resp. $\mathcal{D}_1 + \frac{\beta}{\alpha}$ rounds), where $\mathcal{D}_1$ is the diameter of the network after the dynamic step, and α, μ, and β are parameters respectively satisfying $\alpha \geq 2$, $\mu \geq \max(2, N)$, $\beta > \mu^2$, and β is multiple of α.*

Proof. By Remarks 4, 9, and 10, we can use results on $\mathcal{WU}$ for $\mathcal{DSU}$. If $\mathcal{DSU}$ has converged to a configuration $\gamma \in \mathcal{L}_{\mathsf{WU}}$, then $\gamma \in C_\mu$. So, by Lemma 10, after $\mu\mathcal{D}_1 + \frac{\beta}{\alpha}$ rounds, p increments $p.t$ at least $\frac{\beta}{\alpha}$ times. Now, by Remark 6, if t-variable is incremented $\frac{\beta}{\alpha}$ times, c-variable is incremented once.

If $\mathcal{DSU}$ has converged to $\mathcal{L}_{\mathsf{SU}}$, the result of Theorem 10 can be applied (Remarks 9 and 10). So, after $\mathcal{D}_1 + \frac{\beta}{\alpha}$ rounds, p increments $p.c$ at least once. □

8 Conclusion

The apparent seldomness of superstabilizing solutions for non-static problems, such as unison, may suggest the difficulty of obtaining such a strong property and if so, make our notion of gradual stabilization very attractive compared to merely self-stabilizing solutions. For example, in our unison solution, gradual stabilization ensures that processes remain "almost" synchronized during the convergence phase started after one BULCC-dynamic step. Hence, it is worth investigating whether this new paradigm can be applied to other, in particular non-static, problems. Concerning our unison algorithm, the graceful recovery after one dynamic step comes at the price of slowing down the clock increments. The question of limiting this drawback remains open. Finally, it would be interesting to address in future work gradual stabilization for non-static problems in context of more complex dynamic patterns.

References

[1] Karine Altisen, Stéphane Devismes, Anaïs Durand, and Franck Petit. Gradual stabilization under τ-dynamics. In Pierre-François Dutot and Denis Trystram, editors, *Euro-Par 2016: Parallel Processing - 22nd International Conference on Parallel and Distributed Computing, Grenoble, France, August 24-26, 2016, Proceedings*, volume 9833 of *Lecture Notes in Computer Science*, pages 588–602. Springer, 2016.

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

[3] Baruch Awerbuch, Shay Kutten, Yishay Mansour, Boaz Patt-Shamir, and George Varghese. Time optimal self-stabilizing synchronization. In *STOC*, pages 652–661, 1993.

[4] Christophe Genolini and Sébastien Tixeuil. A lower bound on dynamic k-stabilization in asynchronous systems. In *SRDS*, page 212, 2002.

[5] Shay Kutten and Boaz Patt-Shamir. Stabilizing time-adaptive protocols. *Theor. Comput. Sci.*, 220(1):93–111, 1999.

[6] Sukumar Ghosh, Arobinda Gupta, Ted Herman, and Sriram V. Pemmaraju. Fault-containing self-stabilizing distributed protocols. *Distributed Computing*, 20(1):53–73, 2007.

[7] Shlomi Dolev and Ted Herman. Superstabilizing protocols for dynamic distributed systems. *Chicago J. Theor. Comput. Sci.*, 1997, 1997.

[8] Hirotsugu Kakugawa and Toshimitsu Masuzawa. A self-stabilizing minimal dominating set algorithm with safe convergence. In *IPDPS*, pages 8.–, 2006.

[9] Sayaka Kamei and Hirotsugu Kakugawa. A self-stabilizing approximation for the minimum connected dominating set with safe convergence. In *OPODIS*, pages 496–511, 2008.

[10] Sayaka Kamei, Tomoko Izumi, and Yukiko Yamauchi. An asynchronous self-stabilizing approximation for the minimum connected dominating set with safe convergence in unit disk graphs. In *SSS*, pages 251–265, 2013.

[11] Sayaka Kamei and Hirotsugu Kakugawa. A self-stabilizing 6-approximation for the minimum connected dominating set with safe convergence in unit disk graphs. *Theor. Comput. Sci.*, 428:80–90, 2012.

[12] Fabienne Carrier, Ajoy Kumar Datta, Stéphane Devismes, Lawrence L. Larmore, and Yvan Rivierre. Self-stabilizing (f,g)-alliances with safe convergence. *J. Parallel Distrib. Comput.*, 81-82:11–23, 2015.

[13] Lélia Blin, Maria Gradinariu Potop-Butucaru, Stephane Rovedakis, and Sébastien Tixeuil. Loop-free super-stabilizing spanning tree construction. In *SSS*, pages 50–64, 2010.

[14] Lélia Blin, Maria Potop-Butucaru, and Stephane Rovedakis. A superstabilizing log(n)log(n)-approximation algorithm for dynamic steiner trees. *Theor. Comput. Sci.*, 500:90–112, 2013.

[15] Ted Herman. Superstabilizing mutual exclusion. *Distributed Computing*, 13(1):1–17, 2000.

[16] Yoshiaki Katayama, Eiichiro Ueda, Hideo Fujiwara, and Toshimitsu Masuzawa. A latency optimal superstabilizing mutual exclusion protocol in unidirectional rings. *J. Parallel Distrib. Comput.*, 62(5):865–884, 2002.

[17] Christian Boulinier. *L'Unisson*. PhD thesis, Université de Picardie Jules Vernes, France, 2007.

[18] Mohamed G. Gouda and Ted Herman. Stabilizing unison. *Inf. Process. Lett.*, 35(4):171–175, 1990.

[19] Anish Arora, Shlomi Dolev, and Mohamed G. Gouda. Maintaining digital clocks in step. *Parallel Processing Letters*, 1:11–18, 1991.

[20] Shing-Tsaan Huang and Tzong-Jye Liu. Four-state stabilizing phase clock for unidirectional rings of odd size. *Inf. Process. Lett.*, 65(6):325–329, 1998.

[21] Florent Nolot and Vincent Villain. Universal self-stabilizing phase clock protocol with bounded memory. In *IPCCC*, pages 228–235, 2001.

[22] Colette Johnen, Luc Onana Alima, Ajoy Kumar Datta, and Sébastien Tixeuil. Optimal snap-stabilizing neighborhood synchronizer in tree networks. *Parallel Processing Letters*, 12(3-4):327–340, 2002.

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

[24] Chi-Hung Tzeng, Jehn-Ruey Jiang, and Shing-Tsaan Huang. Size-independent self-stabilizing asynchronous phase synchronization in general graphs. *J. Inf. Sci. Eng.*, 26(4):1307–1322, 2010.

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

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

[27] Shlomi Dolev, Amos Israeli, and Shlomo Moran. Uniform Dynamic Self-Stabilizing Leader Election. *IEEE Trans. Parallel Distrib. Syst.*, 8(4):424–440, 1997.

[28] Jayadev Misra. Phase synchronization. *Inf. Process. Lett.*, 38(2):101–105, 1991.

[29] Sandeep S. Kulkarni and Anish Arora. Multitolerant barrier synchronization. *Inf. Process. Lett.*, 64(1):29–36, 1997.