Gradual Stabilization under τ-Dynamics

Karine Altisen, Stéphane Devismes, Anaïs Durand, Franck Petit

Gradual Stabilization under τ-Dynamics

Karine Altisen,[§] Stéphane Devismes,[§] Anaïs Durand,[§] and Franck Petit[†]

[§]VERIMAG UMR 5104, Université Grenoble Alpes, France

[†]LIP6 UMR 7606, INRIA, UPMC Sorbonne Universités, France

Abstract

We introduce the notion of *gradually stabilizing* algorithm as any self-stabilizing algorithm achieving the following additional feature. Assuming that at most τ *dynamic steps* occur starting from a legitimate configuration, a gradually stabilizing algorithm first quickly recover 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 fully recovering to its initial (strong) specification.

We illustrate this new property by considering three variants of a synchronization problem respectively called *strong*, *weak*, and *partial weak* unison. We propose a self-stabilizing algorithm which is also gradually stabilizing in the sense that after one dynamic step from a legitimate configuration, it immediately satisfies the specification of partial weak unison, then converges to the specification of weak unison in at most one round, and finally retrieves the specification of strong unison 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 and μ is a parameter which should be greater than or equal to sum of n (the initial size of the network) and $\#J$ (an upper bound on the number of processes that join the system during the dynamic step).

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

Contents

1 Introduction

In 1974, Dijkstra [10] introduced *self-stabilization*, a general paradigm to enable the design of distributed systems tolerating *any* finite number of transient faults. 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 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 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 [2] and more generally for non-static problems [14] (such as token passing) 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. Consequently, recent research focuses on ensuring 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 given. For example, the notion of k-stabilization [3] guarantees that the system recovers 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 [15], 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* [12] 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 $O(1)$ rounds), and a safety predicate, called a *passage* predicate, should be satisfied along the stabilization phase.

Contribution In this paper, we introduce a specialization of self-stabilization called *gradual stabilization*. A gradually stabilizing algorithm is a self-stabilizing algorithm with the following additional feature. Assuming that at most τ *dynamic steps*[1] occur starting from a legitimate configuration, a gradually stabilizing algorithm 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 fully recovering to its initial (strong) specification. 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 weak* 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) 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 defined partial weak unison as a property dedicated to dynamic systems. It 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 which works with any period $\alpha > 4$ in any anonymous connected network. It assumes the knowledge of two values μ and β, where μ is any upper bound on n, and β should divide α and be greater than μ^2. Our algorithm 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 this algorithm to make it gradually stabilizing assuming at most one dynamic step. In particular, the parameter μ should now be greater than or equal to $n + \#J$, where $\#J$ is an upper bound on the number of processes that join the system during the dynamic step. Notice that these slight modifications lead to increase the stabilization time by one round. This new version is gradually stabilizing because after one dynamic step from a configuration which is legitimate for the strong unison,

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

it immediately satisfies the specification of partial weak unison, then converges to the specification of weak unison in at most one round, and finally retrieves the specification of strong unison 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. The dynamic step may contain several topological events (*i.e.*, link and/or process additions and/or removals). However, we require that, after those topological changes, the network should stay connected and if $\alpha > 4$, every process which joins the system should be linked to at least one process already in the system before the dynamic step. We show that this condition, called UnderLocalControl, is necessary.

Related Work Gradual stabilization is related to two other stronger forms of self-stabilization, namely, *safe-converging self-stabilization* [20] and *superstabilization* [12]. 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, this gradual convergence is stronger than ours as it should be ensured after any step of transient faults,[2] 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 [20], approximately minimum weakly connected dominating sets [22], approximately minimum connected dominating sets [21, 23], and minimal (f, g)-alliances [8]. However, to the best of our knowledge, 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 [12, 5, 4], and coloring [12]. However, notice that there exist few superstabilizing algorithms for non-static problems, such as mutual exclusion [17, 24].

We use the general term *unison* to name several close problems also known in the lit-

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

erature as *phase or barrier synchronization* problems. There exists many self-stabilizing algorithms for the strong as well as weak unison problem, *e.g.*, [16, 1, 18, 27, 19, 7, 28]. Now, to the best of our knowledge, until now, no self-stabilizing solution for such problems proposes specific convergence properties in case of topological changes. Self-stabilizing strong unison was first considered in synchronous anonymous networks. Particular topologies were considered in [18] (rings) and [27] (trees). Gouda and Herman [16] 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 [1]. In [28], 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 [19]. The first self-stabilizing asynchronous weak unison for general graphs was proposed by Couvreur *et al.* [9]. However, no complexity analysis was given. Another solution which stabilizes in $O(n)$ rounds has been proposed by Boulinier *et al.* in [7]. Finally, Boulinier proposed in his PhD thesis a parametric solution which generalizes both the solutions of [9] and [7]. 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 show that condition UnderLocalControl is necessary to obtain our gradually stabilizing solution. 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. Some useful results from [6] are recalled in the appendix.

2 Preliminaries

We consider the *locally shared memory model* introduced by Dijkstra [10] enriched with the notion of topological changes. Thereupon, we follow an approach similar to the one used by Dolev in the context of superstabilization [11].

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 along 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 defines a *distributed algorithm*. In the distributed algorithm $\mathcal{A}$, the local algorithm of 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, if at least one action is enabled at p, p is said to be enabled. 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 $(G_i, V_i \rightarrow \mathcal{S})$. $G_i = (V_i, E_i)$ 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 processes of V_i in γ_i: $\forall p, q \in V_i$, $\{p, q\} \in E_i \Leftrightarrow p \in q.\mathcal{N}$ in γ_i. $V_i \rightarrow \mathcal{S}$ is a function which associates a state to any process of V_i. For sake of simplicity, 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.

Executions. The dynamicity and asynchronism of the system is materialized by an adversary, called the *daemon*. When the system is in a configuration γ, the daemon first chooses between

- selecting enabled processes to make them performing actions— in this case, a *computation step* is made, and
- modifying the topology — in this case, the system suffers from a *dynamic step*.

The set of all possible computation (resp. dynamic) steps induces a binary relation over configurations noted $\mapsto_c \subseteq \mathcal{C} \times \mathcal{C}$ (resp. $\mapsto_d \subseteq \mathcal{C} \times \mathcal{C}$). Let $\mapsto = \mapsto_d \cup \mapsto_c$ be the binary relation defining all possible *steps*.

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 $G_0 = G = (V, E)$; we also note $\mathcal{D}$ the diameter of G and we recall that $|V_0| = |V| = n$. Moreover, we define $\mathcal{E}^\tau$ the set of maximal executions which contain at most τ dynamic steps. A maximal 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 $\mathcal{E}^i \subseteq \mathcal{E}^j$. For any subset of configurations $X \subseteq \mathcal{C}$, we denote by $\mathcal{E}^\tau_X$ the set of all executions in $\mathcal{E}^\tau$ taht start from a configuration of X, *i.e.*, $\mathcal{E}^\tau_X = \{(\gamma_i)_{i \geq 0} \in \mathcal{E}^\tau : \gamma_0 \in X\}$.

Computation 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 computation 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, $\gamma_i \mapsto_c \gamma_{i+1}$ with, in particular, $G_i = G_{i+1}$.

Dynamic steps. Let $\gamma_i \mapsto_d \gamma_{i+1}$ be a dynamic step. We have $G_{i+1} \neq G_i$, whereas the state of every process in $V_i \cap V_{i+1}$ remains unchanged, *i.e.*, $\forall p \in V_i \cap V_{i+1}$, $\gamma_{i+1}(p) = \gamma_i(p)$. Precisely, $\gamma_i \mapsto_d \gamma_{i+1}$ contains a finite number of events 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 the variables of p to a particular state, called *bootstate*. This bootstrap is executed 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 $\gamma_x \mapsto \gamma_{x+1}$, it is still in bootstate. Hence $\forall j \in \{i+1, \ldots, x\}, p \in New_j$, but $p \notin New_{x+1}$.

  We assume that there are at most $\#J$ joins in the system during a dynamic step.

- Process p can also *leave* the system or *crash*, *i.e.*, $p \in V_i \wedge p \notin V_{i+1}$.

- Finally, some communication links can *appear* (resp. *disappear*) between two processes p and q, *i.e.*, $\{p, q\} \notin E_i \wedge \{p, q\} \in E_{i+1}$ (resp. $\{p, q\} \in E_i \wedge \{p, q\} \notin E_{i+1}$).

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

Daemon. As previously explained, executions are driven by a daemon. In this paper, we assume the daemon is *distributed* and *unfair*. In any computation step, a distributed daemon must select at least one enabled process (maybe more). An unfair daemon has no fairness constraint, *i.e.*, it might never select a process during any computation step unless it is the only enabled one. Moreover, at each configuration an unfair daemon freely chooses between making a computation or dynamic step, except if the configuration is terminal; in this latter case, only dynamic steps can be chosen.

Functional specification and performances. 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* [13]. This latter 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} ($p \notin V_{i+1}$), or p is not enabled in γ_{i+1} and 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 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 $\forall \gamma, \gamma' \in \mathcal{C}, (\gamma \in X \wedge \gamma \mapsto_c \gamma') \Rightarrow \gamma' \in X$. 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 in steps (resp. rounds) from Y to X* is the maximal number of steps (or rounds, respectively) to reach a configuration of X in over every execution of $\mathcal{E}_Y^0$.

Self-stabilization has been defined by Dijkstra in 1974 [10] as follows: a distributed algorithm $\mathcal{A}$ is *self-stabilizing* for a specification SP if and only if $\exists \mathcal{L} \subseteq \mathcal{C}$, $\mathcal{A}$ 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-Dynamics

Below, we introduce a specialization of self-stabilization called *gradual stabilization*. The main idea behind this concept is the following: if after starting from a legitimate configuration, the system suffers from few topological changes, then the very first configuration after those topological changes may be illegitimate, but this configuration is usually far from being arbitrary. Hence, in such a situation, it may be possible to first quickly recover to a configuration from which a specification offering a minimum quality of service is satisfied. It may be also possible to gradually converge to specifications offering stronger and stronger safety guarantees until fully recovering to the initial (strong) specification. Of course, the gradual stabilization makes sense only if the convergence to every intermediate weaker specification is fast.

Let $\tau \geq 0$. For a given execution $e = (\gamma_i)_{i \geq 0} \in \mathcal{E}^\tau$, let $first(e)$ be the integer such that $\gamma_{first(e)}$ is the first configuration of e after the last topological change. Formally, $first(e)$ is the minimal index such that the suffix of e starting from $first(e)$ contains no dynamic step: $first(e) = \min\{i : (\gamma_j)_{j \geq i} \in \mathcal{E}^0\}$. For any subset of executions $E \subseteq \mathcal{E}^\tau$, let $FC(E) = \{\gamma_{first(e)} : e = (\gamma_i)_{i \geq 0} \in E\}$ ($FC()$ stands for "First Configuration").

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 $B_1 < B_2 < \cdots < B_k$.

A distributed algorithm $\mathcal{A}$ is *gradually stabilizing under τ-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 $\mathcal{L}_k$, *i.e.*, *$\mathcal{A}$ is self-stabilizing for SP_k*.
2. *Starting from a legitimate configuration, after at most τ steps of topological changes, $\mathcal{A}$ gradually converges to every $\mathcal{L}_i$ with $i \in \{1, \ldots, k\}$*, *i.e.*, $\forall i \in \{1, \ldots, k\}$, we have

- $\mathcal{A}$ stabilizes from $FC(\mathcal{E}^{\tau}_{\mathcal{L}_k})$ to SP_i by $\mathcal{L}_i$, and
- the convergence time in rounds from $FC(\mathcal{E}^{\tau}_{\mathcal{L}_k})$ to $\mathcal{L}_i$ is bounded by B_i.

Notice that, by definition, any gradually stabilizing algorithm is also a self-stabilizing algorithm for SP_k. Hence, the performances of any gradually stabilizing algorithm can be also evaluated at the light of its stabilization time.

Gradual stabilization is related to two other stronger forms of self-stabilization: *safe-converging self-stabilization* [20] and *superstabilization* [12].

The goal of a *safely converging self-stabilizing algorithm* is to first quickly (within $O(1)$ rounds is the usual rule) converge 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 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 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 only considers one dynamic step consisting in only one topological event: 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-dynamics for $(SP_0 \bullet 0, SP_1 \bullet f)$ where SP_0 is the passage predicate, f is the superstabilization time and the dynamic step consists of adding or removing one link or process in the network only.

4 Unison

We consider several 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, *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) while fulfilling some safety requirements. Below we define three

versions of the problem respectively named *strong*, *weak*, and *partial weak* unison.

Strong unison defined below is also known as the *phase* or *barrier* synchronization problem [26, 25].

Specification 1 (Strong Unison)**.** An execution e satisfies the specification SP_{SU} of *strong unison* if and only if

- In any configuration $\gamma \in e$, there exists at most two different clock values, and if so, these two values are consecutive (modulo α). *(Safety)*
- Every process increments its clock infinitely often in e. *(Liveness)*

The definition of *weak unison* below appeared first in [9] under the name of *asynchronous unison*.

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

- In any configuration $\gamma \in e$, the clocks of every two neighboring processes differ from at most one increment (modulo α). *(Safety)*
- Every process increments its clock infinitely often in e. *(Liveness)*

Finally, in the context of dynamic systems, a straightforward variant of the weak unison is the following.

Specification 3 (Partial Weak Unison)**.** An execution $e = (\gamma_i)_{i \geq 0}$ satisfies the specification SP_{PU} of *partial weak unison* if and only if

- In any configuration $\gamma_i \in e$, the clocks of any two neighbors which are not in New_i differ from at most one increment (modulo α). *(Safety)*
- Every process increments its clock infinitely often in e. *(Liveness)*

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

Property 1. $SP_{\mathsf{SU}} \Rightarrow SP_{\mathsf{WU}} \Rightarrow SP_{\mathsf{PU}}$.

5 Necessary Condition

Through out this section, we assume the existence of a deterministic algorithm $\mathcal{A}$ which is gradually stabilizing under 1-dynamics for $(SP_{\mathsf{PU}} \bullet 0, SP_{\mathsf{WU}} \bullet 1, SP_{\mathsf{SU}} \bullet B)$ in any arbitrary anonymous network under the distributed unfair daemon, with $B > 1$ be any (asymptotic) complexity bound. Let $\mathcal{L}^{\mathcal{A}}_{\mathsf{SU}}$ be the legitimate configurations of $\mathcal{A}$ *w.r.t.* specification SP_{SU}.

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

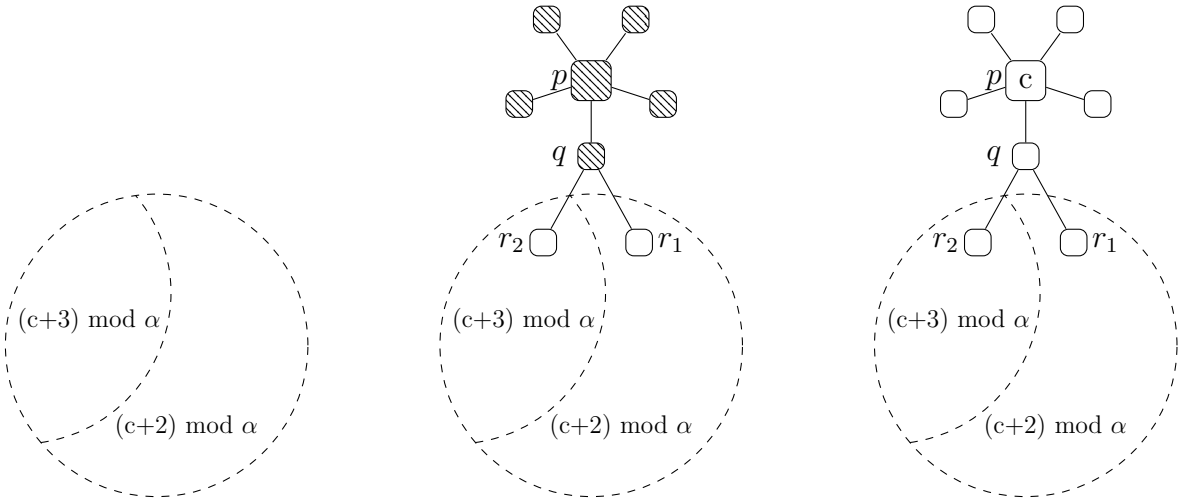
Property 2. *Assume $\alpha > 3$. For every $(\gamma_i)_{i\geq 0} \in \mathcal{E}^0_{\mathcal{L}^{\mathcal{A}}_{\mathsf{SU}}}$, for every process p, for every $k \in \{0, ..., \alpha - 1\}$, for every $i \geq 0$, if p increments its clock from k to $(k+1) \mod \alpha$ in $\gamma_i \mapsto \gamma_{i+1}$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

Assume that there is a process q such that $\gamma_i(q).clock = (k-1) \mod \alpha$. As the execution satisfies SP_{SU}, there exists a step after γ_i in which p increments, due to liveness; but due to safety, q necessarily increments at the same step. Using the daemon, we can now build a possible step where p moves, but not q leading to a configuration where $q.clock = (k-1) \mod \alpha$ and $p.clock = (k+1) \mod \alpha$. Hence, there exists an execution starting from a configuration of $\mathcal{L}^{\mathcal{A}}_{\mathsf{SU}}$ 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 and γ_j there exists a configuration where all processes have clock value $(k+1) \mod \alpha$. □

In the following, we will establish that the property UnderLocalControl given below is a necessary condition for $\mathcal{A}$. The definition of UnderLocalControl uses the notion of *dominating set*: 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. UnderLocalControl captures a condition on the network dynamics which is necessary to prevent a notable desynchronization of clocks. Namely, the network should stay connected and if $\alpha > 4$, every process that joins during the dynamic step $\gamma \mapsto_d \gamma'$ should be "under the control of" (that is, linked to) at least one process which exists in both γ and γ'.



(a) γ_T (b) γ_{T+1} (c) γ_{T+2}

Figure 1: Execution e'' in the proof of Theorem 1. The hachured nodes are in bootstate.

Definition 1 (UnderLocalControl)**.** UnderLocalControl holds if and only if for every execution $e = (\gamma_i)_{i\geq 0} \in \mathcal{E}^1_{\mathcal{L}^{\mathcal{A}}_{\mathsf{SU}}}$,

1. $G_{first(e)}$ is connected, and
2. if $\alpha > 4$, then $V_{first(e)} \setminus New_{first(e)}$ is a dominating set.

Lemma 1. *For every execution $e \in \mathcal{E}^1_{\mathcal{L}^{\mathcal{A}}_{\mathsf{SU}}}$, $G_{first(e)}$ is connected.*

Proof. Assume, by the contradiction, that there is an execution $e = (\gamma_i)_{i\geq 0} \in \mathcal{E}^1_{\mathcal{L}^{\mathcal{A}}_{\mathsf{SU}}}$ such that $G_{first(e)}$ is disconnected. Let A and B be two connected components of $G_{first(e)}$. By definition, there exists $j \geq first(e)$ 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. In particular, there always exists enabled processes in A that increment. 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. Consequently, there exists a possible execution of $\mathcal{E}^1_{\mathcal{L}^{\mathcal{A}}_{\mathsf{SU}}}$ 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. □

Lemma 2. *If $\alpha > 4$, then for every execution $e \in \mathcal{E}^1_{\mathcal{L}^{\mathcal{A}}_{\mathsf{SU}}}$, $V_{first(e)} \setminus New_{first(e)}$ is a dominating set.*

Proof. We illustrate the following proof with Figure 1. Let $e \in (\gamma_i)_{i\geq 0} \in \mathcal{E}^1_{\mathcal{L}^{\mathcal{A}}_{\mathsf{SU}}}$. Let $x = first(e)$. Assume, by the contradiction, that $\alpha > 4$ and G_x is connected, but

$V_x \setminus New_x$ is not a dominating set. This implies that $\exists p \in New_x$ such that $\forall q \in \gamma_x(p).\mathcal{N}$, $q \in New_x$.

First, notice that every process among p and its neighbors are enabled in γ_x to take a clock value in $\{0 \dots \alpha - 1\}$. Indeed, assume that the daemon makes a synchronous step from γ_x, then the step $\gamma_x \mapsto \gamma_{x+1}$ actually corresponds to a complete round, by definition and so γ_{x+1} should be a legitimate configuration of weak unison. Let c be the clock value taken by p if p moves in $\gamma_x \mapsto_c \gamma_{x+1}$.

Consider now another execution e' in $\mathcal{E}^0_{\mathcal{L}^{\mathcal{A}}_{\mathsf{SU}}}$ (with no topological change) on a graph of at least two nodes which contains neither p nor its neighbors in $\gamma_x(p).\mathcal{N}$. Strong unison is satisfied in e' and, as $\alpha > 4$, by Property 2, there is a configuration γ_S in e' where every clock equals $(c+2) \mod \alpha$. From γ_S, there is eventually a step in which at least one process increments its clock to $(c+3) \mod \alpha$. Assume not all processes are activated by the distributed unfair daemon during this step. Then, this step leads to a configuration γ_T where there is exactly two values of clock: $(c+2) \mod \alpha$ and $(c+3) \mod \alpha$, see Figure 1a.

Consider now another execution e'' having a prefix common to e' until γ_T. Assume that the daemon introduces a dynamic step at configuration γ_T. Assume that this step consists in adding p with the same neighborhood as well as two links from q, a neighbor of p, and two already existing nodes r_1 and r_2, such that the clock of r_1 (resp. r_2) equals $(c+2) \mod \alpha$ (resp. $(c+3) \mod \alpha$) in γ_T, see Figure 1b. By definition, since strong unison is satisfied in γ_T (by assumption), the partial weak unison necessarily holds along the suffix of e'' starting at γ_{T+1}.

Process p and its neighbors are in a situation similar to the one in γ_x so they are enabled to take a clock value in $\{0 \dots \alpha - 1\}$, in particular p is enabled to take value c. Assume that the daemon exactly selects p and its neighbors in the next step $\gamma_{T+1} \mapsto \gamma_{T+2}$. In γ_{T+2} (Figure 1c), the clock of r_1 and r_2 are respectively equal to $(c+2) \mod \alpha$ and $(c+3) \mod \alpha$, since they did not move; moreover, the clock of p is equal to c. Now, q also chooses a clock value in $\gamma_{T+1} \mapsto \gamma_{T+2}$ and that clock value should differ of at most one increment from the clocks of p, r_1, and r_2 since partial weak unison holds in γ_{T+1} and all subsequent configurations. As $\alpha > 3$, if the clock of q equals:

- c or $(c+1) \mod \alpha$, the difference between the clocks of q and r_2 is at least 2 increments,
- $(c+2) \mod \alpha$ or $(c+3) \mod \alpha$, the difference between the clocks of q and p is at least 2 increments,
- any value in $\{0 ... \alpha - 1\} \setminus \{c, (c+1) \mod \alpha, (c+2) \mod \alpha, (c+3) \mod \alpha\}$, the difference between the clocks of q and r_1 is at least 2 increments.

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

By Lemmas 1 and 2, follows:

Theorem 1. *An algorithm $\mathcal{A}$ is gradually stabilizing under 1-dynamics for $(SP_{\mathsf{PU}} \bullet 0, SP_{\mathsf{WU}} \bullet 1, SP_{\mathsf{SU}} \bullet B)$ in arbitrary anonymous networks under the distributed unfair daemon with a set of legitimate configurations* w.r.t. *specification SP_{SU} noted $\mathcal{L}^{\mathcal{A}}_{\mathsf{SU}}$ only if* UnderLocalControl *holds.*

6 Self-Stabilizing Strong Unison

In this section, we propose an algorithm which is self-stabilizing for the strong unison problem in any arbitrary connected anonymous network. This algorithm works for any period $\alpha > 4$ and is based on an algorithm previously proposed by Boulinier in [6], this latter is self-stabilizing for the weak unison problem and works for any period $\beta > n^2$, where n is the size of the network. 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 give our self-stabilizing algorithm for the strong unison, Algorithm $\mathcal{SU}$, and its proof of correctness in Subsection 6.2.

6.1 Algorithm $\mathcal{WU}$

Algorithm $\mathcal{WU}$, see Algorithm 1 for its formal code, has been proposed by Boulinier in his PhD thesis [6]. Actually, it is a generalization of the self-stabilizing weak unison algorithm proposed by Couvreur *et al.* [9]. This algorithm being simply self-stabilizing, it only considers executions without any topological change, yet starting from arbitrary configuration. The topology of the network then consists in a connected graph $G = (V, E)$ of n nodes which is fixed all along the execution. Remind that $\mathcal{D}$ is the diameter of G.

Algorithm $\mathcal{WU}$ uses the following notations.

Notations We define the *delay* between two integer values x and y by the function $d_\beta\big(x, y\big) = \min\big((x - y) \mod \beta, (y - x) \mod \beta\big)$. Then, let the order relation $\preceq_{\beta,\mu}$ such that for every two integer values x and y, $x \preceq_{\beta,\mu} y \equiv \big((y - x) \mod \beta\big) \leq \mu$.

Overview In the algorithm, 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 $n \leq \mu \leq \frac{\beta}{2}$. The main idea behind the algorithm is the following. When the delay between the clock of a given process p

and the clocks of some neighbors is greater than one, but the 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 making increment the clocks of the most behind processes among p and 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), then p should reset its clock to 0 (Action $\mathcal{WU}$-R).

The Algorithm Two actions are used to maintain $p.t$ at each process p. When p is in time or late, but not that much, with all its neighbors, *i.e.*, when $\forall q \in p.\mathcal{N}$, $p.t \preceq_{\beta,\mu} q.t$, p can "normally" increment its clock using Action $\mathcal{WU}$-N. When the delay between the clocks of p and one of its neighbors q is too big, *i.e.*, $d_\beta\big(p.t, q.t\big) > \mu$ and the clock of p is not yet reset, then p resets 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 $n \leq \mu \leq \frac{\beta}{2}$

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 [6], we have the following theorem.

Theorem 2. *Algorithm $\mathcal{WU}$ is self-stabilizing for SP_{WU} (specification of weak unison) by the set of legitimate configurations*

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

in an arbitrary connected network assuming a distributed unfair daemon.

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 $n \leq \mu \leq \frac{\beta}{2}$.

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

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

Complexity Analysis. Let C_μ be the set of configurations where the distance between two neighboring clocks is at most μ. Below, we prove in Lemma 3 (resp. Lemma 4) a

bound on the time required to ensure that all t-variables have incremented k times which holds since the system has reached a configuration of C_μ (resp. $\mathcal{L}_{\mathsf{WU}}$)

Lemma 3. *$\forall k \geq 1$, $\forall e \in \mathcal{E}^0_{C_\mu}$, 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 30, $\forall i \geq 0$, there is a function f on processes such that $\forall p \in V, f(\gamma_i, p) \bmod \beta = \gamma_i(p).t$ and $\forall p \in V$, $\forall q \in p.\mathcal{N}$, $d_\beta\big(\gamma_i(p).t, \gamma_i(q).t\big) \leq \mu$. Hence, $\forall p, q \in V$, $|f(\gamma_i, p) - f(\gamma_i, q)| \leq \mu\mathcal{D}$.

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 $\gamma_i(x).t = f(\gamma_i, x) = f^{\min}_{\gamma_i}$. So, after one round, every such a process x has incremented its t-variable (executing action $\mathcal{WU}$-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 inductively, the same arguments as for j, it comes that $f^{\min}_{\gamma_d} \geq f^{\min}_{\gamma_i} + \mu\mathcal{D} + k$ $(*)$.

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 4. *$\forall k \geq 1$, $\forall e \in \mathcal{E}^0_{\mathcal{L}_{\mathsf{WU}}}$, 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 for Lemma 3 when replacing C_μ with $\mathcal{L}_{\mathsf{WU}}$ and $\mu\mathcal{D}$ with $\mathcal{D}$. □

Some other useful results from [6] about Algorithm $\mathcal{WU}$ are recalled in Appendix A.

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 for its formal code, is designed for any period $\alpha > 4$. Its actions $\mathcal{SU}$-N and $\mathcal{SU}$-R are identical to actions $\mathcal{WU}$-N and $\mathcal{WU}$-R of Algorithm $\mathcal{WU}$, 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 1). 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 just need to fix β in such way that $\frac{\beta}{\alpha}$ is greater than or equal to n 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 2 (page 21). Furthermore, the liveness of $\mathcal{WU}$ ensures that every t-clock increments infinitely often, hence so do c-clocks.

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

Parameters:

α: any positive integer such that $\alpha > 4$
μ: any positive integer such that $\mu \geq n$
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	::	$\exists q \in p.\mathcal{N}, d_\beta(p.t, q.t) > \mu$	$\rightarrow$	$p.t \leftarrow 0$
		$\wedge\ p.t \neq 0$		$p.c \leftarrow 0$

Remark 2. Notice that $\beta > \mu^2$, so $\beta \geq 2\mu$ when $\mu \geq 2$. Moreover, $\alpha > 4$ and β is a multiple of α, so $\beta \geq 5$. Thus, $\beta \geq 2\mu$ also holds if $\mu = 1$.

Remark 3. By construction and from Remark 2, all results on t-clocks in Algorithm $\mathcal{WU}$ also holds for t-clocks in Algorithm $\mathcal{SU}$.

Theorem 3 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 3. *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 $\mu \geq n$.*

6.2.1 Correctness Proof

We first define a set of legitimate configurations *w.r.t.* the specification SP_{SU} (Definition 2). Then, we prove the closure and convergence *w.r.t.* those legitimate configurations (see Lemmas 5 and 6). Afterwards, we prove correctness *w.r.t.* the specification, namely, safety of SP_{SU} is shown in Lemma 10 and liveness is proved in Lemma 11.

Definition 2 (Legitimate Configurations *w.r.t.* SP_{SU} under Algorithm $\mathcal{SU}$)**.** A configuration γ is *legitimate w.r.t.* SP_{SU} under $\mathcal{SU}$ 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 *w.r.t.* SP_{SU} under $\mathcal{SU}$.

By definition, $\mu \geq n > 0$, hence from Definition 2, follows.

Remark 4. In a 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 5 (Closure)**.** *$\mathcal{L}_{\mathsf{SU}}$ is closed under Algorithm $\mathcal{SU}$.*

Proof. First, from Theorem 2, note that the set of legitimate configurations defined for Algorithm $\mathcal{WU}$ is also closed for Algorithm $\mathcal{SU}$. Hence we only have to check closure for the second constraint of Definition 2, the one on c-variables .

Let $\gamma \in \mathcal{L}_{\mathsf{SU}}$ be a legitimate configuration of Algorithm $\mathcal{SU}$ and let $\gamma \mapsto_c \gamma'$ be a computation step of Algorithm $\mathcal{SU}$. 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 $\gamma \mapsto_c \gamma'$, 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 6 (Convergence)**.** *$\mathcal{C}$ (the set of all possible configurations) converges to $\mathcal{L}_{\mathsf{SU}}$ under Algorithm $\mathcal{SU}$.*

Proof. From Theorem 2, note that the set of legitimate configurations for Algorithm $\mathcal{WU}$ is also reachable in a finite number of steps for Algorithm $\mathcal{SU}$. Hence, again, we only have to check that the second constraint (the one on c-variables) is also achievable within a finite number of steps.

Again using Theorem 2, liveness of Specification SP_{WU} is ensured by Algorithm $\mathcal{WU}$ and therefore by Algorithm $\mathcal{SU}$. Hence, after stabilization, each process p updates its internal clock $p.t$ within a finite time; meanwhile $p.c$ is also updated to $\left\lfloor \frac{\alpha}{\beta}p.t \right\rfloor$. □

Lemmas 7, 8 and 9 are technical results on the values of t- and c- variables that will be used to prove the safety part of Specification SP_{SU}. 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 $\beta = K\alpha$.

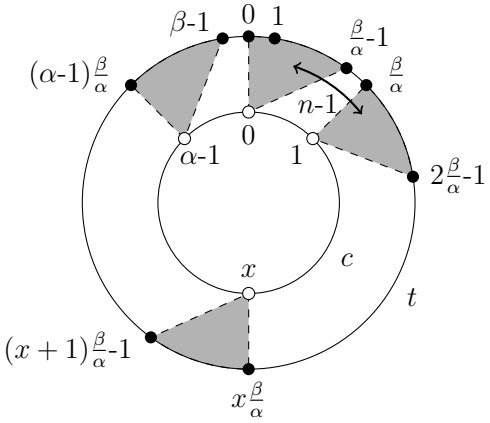


Figure 2: Relationship between variables t and c.

Lemma 7. *Let $x \in \{0, \ldots, \alpha - 1\}$ and $\xi \in \{0, \ldots, \frac{\beta}{\alpha} - 1\}$. The following equality holds: $\left\lfloor \frac{\alpha}{\beta} \left(x\frac{\beta}{\alpha} + \xi \right) \right\rfloor = x$.*

Proof. Let $x \in \{0, \ldots, \alpha - 1\}$ and $\xi \in \{0, \ldots, \frac{\beta}{\alpha} - 1\}$. As $\left\lfloor \frac{\alpha}{\beta} \left(x\frac{\beta}{\alpha} + \xi \right) \right\rfloor = \left\lfloor x + \frac{\alpha}{\beta}\xi \right\rfloor = x + \left\lfloor \frac{\alpha}{\beta}\xi \right\rfloor$ and $\xi \in \{0, \ldots, \frac{\beta}{\alpha} - 1\}$, we have that $0 \leq \frac{\alpha}{\beta}\xi < 1$ and therefore $\left\lfloor \frac{\alpha}{\beta}\xi \right\rfloor = 0$. □

We apply Lemma 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 in Figure 2.

Lemma 8. *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: $x_1\frac{\beta}{\alpha} + \xi_1 \leq x_2\frac{\beta}{\alpha} + \xi_2 \Rightarrow x_1 \leq x_2$*

Proof. Let $x_1, x_2 \in \{0, \ldots, \alpha-1\}$ and $\xi_1, \xi_2 \in \{0, \ldots, \frac{\beta}{\alpha}-1\}$. If $x_1\frac{\beta}{\alpha}+\xi_1 \leq x_2\frac{\beta}{\alpha}+\xi_2$, then we have that $x_1 - x_2 \leq (\xi_1 - \xi_2)/K$. As $\xi_1, \xi_2 \in \{0, \ldots, K-1\}$, we have $(\xi_2 - \xi_1)/K \leq 1 - 1/K$. By transitivity, we obtain that $x_1 - x_2 \leq 1 - 1/K < 1$. As x_1 and x_2 are natural integers, so is their difference; this proves that $x_1 - x_2 \leq 0$ □

Again, Lemma 8 will be used with the internal clock $t = c\frac{\beta}{\alpha} + \xi$: this establishes the monotonic relation between internal and external clocks.

Lemma 9. *Let $t_1, t_2 \in \{0, ..., \beta - 1\}$. The following assertion holds:*

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

Proof. Let $t_1, t_2 \in \{0, ..., \beta - 1\}$ such that $d_\beta\big(t_1, t_2\big) \leq d$. 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 Lemma 7, we have that $\lfloor t_1/K \rfloor = x_1$ and $\lfloor t_2/K \rfloor = x_2$.

Assume, by contradiction, that $d_\alpha\big(x_1, x_2\big) > 1$. 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 $d_\beta\big(t_1, t_2\big) \leq d$, $\min\big((t_1 - t_2) \mod \beta, (t_2 - t_1) \mod \beta\big) \leq d$. 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 Lemma 8. 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 Lemma 8. 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 lemmas, Lemma 9 will be used with the internal clock $t = c\frac{\beta}{\alpha} + \xi$: it expresses that once internal clocks have stabilized at a distance smaller than d, external clocks are at distance smaller than 1. We now prove that Algorithm 2 achieves the safety and liveness properties of SP_{SU}.

Lemma 10 (Safety)**.** *Algorithm $\mathcal{SU}$ achieves the safety of SP_{SU}.*

Proof. Let $\gamma \in \mathcal{L}_{\mathsf{SU}}$: the distance ($\beta$) 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 9 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 $\alpha > 4$, this proves that the variables c in γ have at most two different consecutive values. □

Lemma 11 (Liveness)**.** *Algorithm $\mathcal{SU}$ achieves 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 Action $\mathcal{SU}$-N (by Theorem 2 and Remark 3). So $p.t$ goes through each integer value between 0 and $\beta - 1$ infinitely often (in increasing order). Hence, by Lemma 7, $p.c$ is incremented infinitely often and goes through each integer value between 0 and $\alpha - 1$ (in increasing order). □

Proof of Theorem 3. Lemmas 5 (closure), 6 (convergence), 10 (safety of Specification SP_{SU}), and 11 (liveness of Specification SP_{SU}) 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 4. Then, a delay between any two consecutive clocks increments, which holds once Algorithm $\mathcal{SU}$ has stabilized, is given in Theorem 5.

Theorem 4. *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.*

Proof. Let $(\gamma_i)_{i\geq 0} \in \mathcal{E}^0$. The behavior of the t-variables in Algorithm $\mathcal{SU}$ is similar to that of $\mathcal{WU}$ (Remark 3), which stabilizes in at most $n + \mu\mathcal{D}$ rounds (see Theorems 10 and 11) 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. 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 9). In at most $\mathcal{D} + 1$ additional rounds, each process executes $\mathcal{SU}$-N (Lemma 4) 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 5. *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 $\mathcal{L}^d_{\mathsf{SU}}$, by Remark 3 and Lemma 4, after $\mathcal{D} + \frac{\beta}{\alpha}$ rounds, p increments $p.t$ at least $\frac{\beta}{\alpha}$ times. Now, by Lemma 7, if t-variable is incremented $\frac{\beta}{\alpha}$ times, c-variable is incremented once. □

7 Gradual Stabilization under 1-Dynamics for Strong Unison

We now propose a variant of Algorithm $\mathcal{SU}$, called Algorithm $\mathcal{DSU}$ (see Algorithm 3 for formal code). This latter is still self-stabilizing for strong unison and achieves a

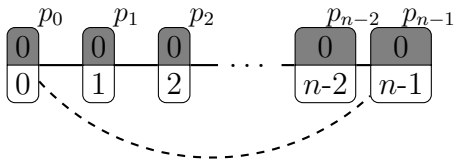


Figure 3: Difference between neighboring t-clocks may become greater than one after adding a link (*e.g.*, the dashed one). The value of c- (resp. t-) variable is in the upper (resp. lower) part of the node.

gradual convergence after one dynamic step. Roughly speaking, after one dynamic step, which may include several topological events, it maintains clocks almost synchronized during the convergence to strong unison. Precisely, after any dynamic step which fulfills condition UnderLocalControl, Algorithm $\mathcal{DSU}$ immediately satisfies partial weak unison, then converges in at most one round to weak unison, and finally re-stabilizes to strong unison.

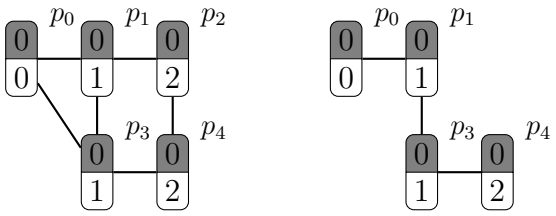
Notice that, after one dynamic step, the graph contains at most $n + \#J$ processes, by definition. Moreover, we denote by $\mathcal{D}_1$ the diameter of the new graph.

7.1 Algorithm $\mathcal{DSU}$

Our solution withstands one dynamic step, which may include several topological events (*i.e.* link or process additions or removals). However, according to Theorem 1, such a dynamic step should satisfy Condition UnderLocalControl. Namely, the graph should stay connected and, as $\alpha > 4$, every process that joins during the dynamic step $\gamma \mapsto_d \gamma'$ should be linked to at least one process which exists in both γ and γ'.

Consider first link additions only. Adding a link can break the safety of weak unison on internal clocks, see for example Figure 3. Indeed, adding a link may create a delay between two (new) neighboring t-clocks greater than one. Nevertheless, the delay between any two t-clocks remains bounded by $n-1$ and, consequently, no process will reset its t-clock (Figure 3 shows a worst case). Moreover, c-clocks still satisfies 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 computation steps in this case. Consider again example in Figure 3: before the dynamic step, 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 has also to wait for p_0.

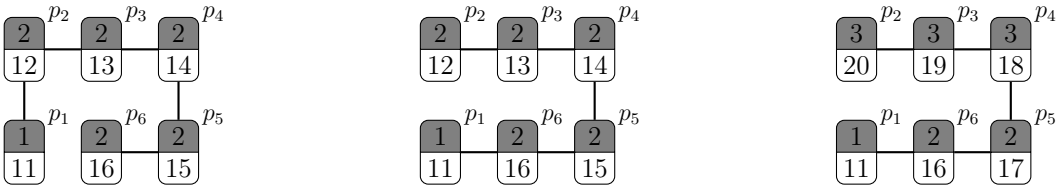
Assume now a dynamic step containing only process and link removals. Due to Condition UnderLocalControl, the network remains connected. Hence, constraints between (still existing) neighbors are maintained: the delay between t-clocks of two neighbors



(a) Initial configuration.

(b) After one dynamic step where p_2 leaves and link $\{p_0, p_3\}$ is removed.

Figure 4: Delay between neighboring t-clocks remains bounded by one after removing processes and/or links.



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

remains bounded by one, see example on Figure 4. So, weak unison on t-clocks remains satisfied and so is strong unison on c-clocks.

Consider now a more complex scenario, where the dynamic step contains link additions as well as process and/or link removals. Figure 5 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 5b leads to a delay between t-clocks of these two (new) neighbors which is greater than one (here 5). However, the removal of link $\{p_1, p_2\}$ 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.*, in Figure 5c, 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 c-clock values in the system, see in Figure 5c $p_1.c = 1$ while $p_2.c = 3$.

Again, in the worst case scenario, after a dynamic step, 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 [6]): (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 one 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 less than or equal to one, *i.e.*, weak unison will be satisfied, all along the convergence to strong unison.

Consider now a process p that joins the system. The event $join_p$ occurs and triggers the specific action *bootstrap* that sets both the clocks $p.t$ and $p.c$ to a specific *bootstate* value, noted $\perp$. Note that by definition and from the previous discussion, the system immediately satisfies partial weak unison since it only depends on processes that were in the system before the 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 clock values in its neighborhood. Precisely, the value of $p.t$ can be chosen among the non-$\perp$ values in its neighborhood, and such values exist by Condition UnderLocalControl. We choose to set $p.t$ to the minimum non-$\perp$ t-clock value in its neighborhood, using the function $MinTime_p$ given below.

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

The value of $p.c$ is then computed according to the value of $p.t$. Notice that $MinTime_p$ returns 0 when p and all its neighbors have their respective t-clock equal to $\perp$. This ensures that Algorithm $\mathcal{DSU}$ remains self-stabilizing (in particular, if the system starts in a configuration where all t-clocks are equal to $\perp$).

To prevent the unfair daemon from blocking the convergence to a configuration containing no $\perp$ values, we should also forbid processes with non-$\perp$ t-clock values to increment while there are $\perp$ t-clock values in their neighborhood. So, we define the predicate *Locked* which holds for a given process p when either $p.t = \perp$, or at least one of its neighbor q satisfies $q.t = \perp$. 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 algorithm behaves exactly the same as Algorithm $\mathcal{SU}$. Hence, it has to match the assumptions made for $\mathcal{SU}$. In particular, the assumptions on α and β remain the same. But 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, the number of processes added in a dynamic step is bounded by $\#J$. So, we require μ to be greater than or equal to $n + \#J$.

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

Parameters:

α: any positive integer such that $\alpha > 4$
μ: any positive integer such that $\mu \geq n + \#J$
β: any positive integer such that $\beta > \mu^2$, and $\exists K$ such that $K > \mu$ and $\beta = K\alpha$

Variables:

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

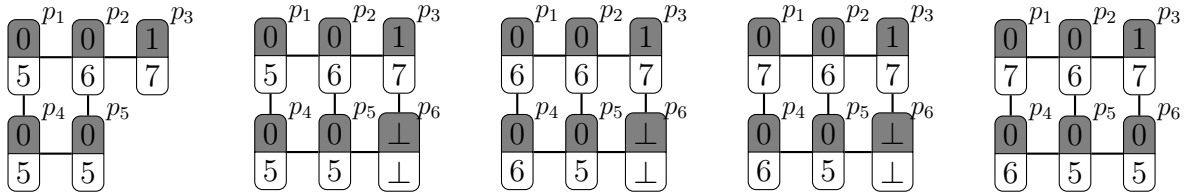
Predicates:

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

Actions:

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

We now consider the example of execution of Algorithm $\mathcal{DSU}$ given in Figure 6. This execution starts in a configuration satisfying strong unison, see Figure 6a. Then, one 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 distributed unfair daemon has no other choice: it selects p_6 to initialize its variables executing $\mathcal{DSU}$-J in (d)$\mapsto$(e).



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

7.2 Proof of Correctness

Proof of self-stabilization *w.r.t.* SP_{SU}.

Remark 5. Looking at Algorithm $\mathcal{DSU}$, if t-variables have values different from $\perp$, the predicates *JoinStep* and *Locked* are false. Furthermore, no action can assign $\perp$ to t. As a consequence,

- when t-variables are initialized with values different from $\perp$,
- as far as no topology change occurs (as far as *join* is false),

Algorithms $\mathcal{DSU}$ and $\mathcal{SU}$ are syntactically the same. This implies in particular, that the set of executions $\mathcal{E}^0$ under $\mathcal{SU}$ and the set of executions $\mathcal{E}^0_{nobot}$ under $\mathcal{DSU}$ are exactly the same, where $nobot = \{\gamma \in \mathcal{C} \ : \ \forall p \in V, \gamma(p.t) \neq \perp\}$.

Definition 3 (Legitimate Configurations *w.r.t.* SP_{SU} under Algorithm $\mathcal{DSU}$)**.** A configuration γ is *legitimate w.r.t.* SP_{SU} under $\mathcal{DSU}$ if and only if

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

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

Remark 6. As $\mathcal{L}^d_{\mathsf{SU}}$ restricts the values of t to non-$\perp$ values, we trivially have the equivalence between $\mathcal{L}^d_{\mathsf{SU}}$ and $\mathcal{L}_{\mathsf{SU}}$: for every configuration γ,

$$\gamma \in \mathcal{L}^d_{\mathsf{SU}} \iff \gamma \in \mathcal{L}_{\mathsf{SU}}$$

Lemma 12 (Closure of $\mathcal{L}^d_{\mathsf{SU}}$ under Algorithm $\mathcal{DSU}$)**.** *The set of configurations $\mathcal{L}^d_{\mathsf{SU}}$ is closed under Algorithm $\mathcal{DSU}$.*

Proof. Let $\gamma \in \mathcal{L}^d_{\mathsf{SU}}$ be a legitimate configuration of Algorithm $\mathcal{DSU}$ and let $\gamma \mapsto_c \gamma'$ be a computation step of Algorithm $\mathcal{DSU}$, from configuration γ. By Remark 6, γ is also in $\mathcal{L}_{\mathsf{SU}}$. But Lemma 5 shows that $\mathcal{L}_{\mathsf{SU}}$ is closed under $\mathcal{SU}$. From Remark 5, the step $\gamma \mapsto_c \gamma'$ is also a step under Algorithm $\mathcal{SU}$ since γ contains no $\perp$. Therefore, γ' is also in $\mathcal{L}_{\mathsf{SU}}$ and in $\mathcal{L}^d_{\mathsf{SU}}$, as well, using again Remark 6. □

Lemma 13. *For any execution $(\gamma_i)_{i\geq 0} \in \mathcal{E}^0$ under $\mathcal{DSU}$, $\exists j \geq 0$ such that $\forall k \geq j$, $\forall p \in V$, $\gamma_k(p).t \neq \perp$.*

Proof. Let $e = (\gamma_i)_{i\geq 0} \in \mathcal{E}^0$. For any $i \geq 0$, we note $Bottom(\gamma_i) = \{p \in V : \gamma_i(p).t = \perp\}$. As actions $\mathcal{DSU}$-N, $\mathcal{DSU}$-R and $\mathcal{DSU}$-J do not create $\perp$ values, $\forall i > 0$, $Bottom(\gamma_i) \subseteq Bottom(\gamma_{i-1})$. Now, assume by contradiction that $\exists p \in V$ such that $\forall i \geq 0$, $p \in Bottom(\gamma_i)$. 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 action $\mathcal{DSU}$-J. So, the unfair daemon selects at least one process to execute action $\mathcal{DSU}$-J and sets its t-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 (no terminal configuration reached).

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 computation step.

Now, from Remark 3 and 5, and since γ_0'' contains no $\perp$ value, e'' is also an execution of Algorithm $\mathcal{WU}$ 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 unison (Specification 2) and Theorem 2 which states that $\mathcal{WU}$ is self-stabilizing for unison under an unfair daemon. □

Lemma 14 (Convergence to $\mathcal{L}^d_{\mathsf{SU}}$)**.** *The whole set of configurations, $\mathcal{C}$, converges under Algorithm $\mathcal{DSU}$ to the set of legitimate configurations $\mathcal{L}^d_{\mathsf{SU}}$.*

Proof. Let $(\gamma_i)_{i\geq 0} \in \mathcal{E}^0$ under $\mathcal{DSU}$. Using Lemma 13, $\exists j \geq 0$ such that $\forall k \geq j$, $\forall p \in V$, $\gamma_k(p).t \neq \perp$. After γ_j, the execution of the system, $(\gamma_k)_{k\geq j}$, is also a possible execution of $\mathcal{SU}$ (see Remark 5). Hence, it converges to a configuration γ_k ($k \geq j$) in $\mathcal{L}_{\mathsf{SU}}$ (Lemma 6). So, using Remark 6, $\gamma_k \in \mathcal{L}^d_{\mathsf{SU}}$. □

Lemma 15 (Correctness of SP_{SU} under $\mathcal{DSU}$)**.** *For any execution $e \in \mathcal{E}^0_{\mathcal{L}^d_{\mathsf{SU}}}$ under $\mathcal{DSU}$, $\mathcal{L}^d_{\mathsf{SU}}(e)$.*

Proof. From Remark 5, every execution in $\mathcal{E}^0_{\mathcal{L}^d_{\mathsf{SU}}}$ under $\mathcal{DSU}$ is also an execution in $\mathcal{E}^0_{\mathcal{L}_{\mathsf{SU}}}$ under $\mathcal{SU}$. Therefore, the correctness is proved in Lemmas 10 and 11. □

Using Lemmas 12, 14 and 15, we can deduce the following theorem:

Theorem 6 (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 7 states the stabilization time of $\mathcal{DSU}$.

Theorem 7. *The stabilization time of $\mathcal{DSU}$ to $\mathcal{L}^d_{\mathsf{SU}}$ 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 $\mu \geq n$.*

Proof. Let $(\gamma_i)_{i\geq 0} \in \mathcal{E}^0$. If there are some processes p such that $\gamma_0(p).t = \perp$, $\mathcal{DSU}$-J is continuously enabled at p. So, in one round $p.t \neq \perp$. Afterwards, the behavior of the algorithm is similar to the one of $\mathcal{SU}$, that stabilizes in at most $n + (\mu + 1)\mathcal{D} + 1$ rounds (see Theorem 4). Hence, in $n + (\mu + 1)\mathcal{D} + 2$ rounds, the system reaches a legitimate configuration. □

Proof of the $SP_{\mathsf{WU}} \bullet 1$ part.

Definition 4 (Legitimate Configurations *w.r.t.* SP_{WU} under Algorithm $\mathcal{DSU}$)**.** A configuration γ is *legitimate w.r.t.* SP_{WU} if and only if

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

We denote by $\mathcal{L}^d_{\mathsf{WU}}$ the set of legitimate configurations *w.r.t.* SP_{WU} under $\mathcal{DSU}$.

Lemma 16 (Closure of $\mathcal{L}^d_{\mathsf{WU}}$ under Algorithm $\mathcal{DSU}$)**.** *The set of configurations $\mathcal{L}^d_{\mathsf{WU}}$ is closed under Algorithm $\mathcal{DSU}$.*

Proof. Notice that the first and third constraints of Definition 4 are closed, as for $\mathcal{L}^d_{\mathsf{SU}}$. As, (1) for every configuration γ, we have that $\gamma \in \mathcal{L}^d_{\mathsf{WU}} \Rightarrow \gamma \in C_\mu$, (2) C_μ is closed under $\mathcal{WU}$ (see Lemma 25) and (3) every computation step of $\mathcal{DSU}$ from $\mathcal{L}^d_{\mathsf{WU}}$ is also a computation step for $\mathcal{WU}$ (see Remarks 3 and 5), the second constraint is also closed under $\mathcal{DSU}$. □

Lemma 17 (Safety of $\mathcal{L}^d_{\mathsf{WU}}$ under $\mathcal{DSU}$)**.** *Safety of SP_{WU} under $\mathcal{DSU}$ is satisfied.*

Proof. Let $\gamma \in \mathcal{L}^d_{\mathsf{WU}}$, $p \in V$, and $q \in \gamma(p).\mathcal{N}$. We have that $d_\beta\big(\gamma(p).t, \gamma(q).t\big) \leq \mu$. and $\gamma(p).c = \left\lfloor \frac{\alpha}{\beta}\gamma(p).t \right\rfloor$. Using Lemma 9 with $d = \mu < K$, we obtain that $d_\alpha\big(\gamma(p).c, \gamma(q).c\big) \leq 1$. □

Lemma 18 (Liveness of $\mathcal{L}^d_{\mathsf{WU}}$)**.** *Liveness of SP_{WU} under $\mathcal{DSU}$ is satisfied.*

Proof. Let $e = (\gamma_i)_{i\geq 0} \in \mathcal{E}^0$ such that $\gamma_0 \in \mathcal{L}^d_{\mathsf{WU}}$. Let p be a process. γ_0 is a legitimate configuration of $\mathcal{WU}$ so p increments infinitely often $p.t$ executing $\mathcal{DSU}$-N (see Lemma 27). Furthermore, actions $\mathcal{DSU}$-R and $\mathcal{DSU}$-J are disabled. So $p.t$ goes through each integer value between 0 and $\beta - 1$ infinitely often (in increasing order). Hence, by Lemma 7, $p.c$ is incremented infinitely often and goes through each integer value between 0 and $\alpha - 1$ (in increasing order). □

Proof of the $SP_{\mathsf{PU}} \bullet 0$ part.

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

a). $\forall p \in V$, $\gamma(p).t = \perp \Rightarrow \big(\exists q \in \gamma(p).\mathcal{N}, \gamma(q).t \neq \perp\big)$.

b). $\forall p \in V, \forall q \in \gamma(p).\mathcal{N},\ \big(\gamma(p).t \neq \bot \wedge \gamma(q).t \neq \bot\big) \Rightarrow \big(d_\beta\big(\gamma(p).t, \gamma(q).t\big) \leq \mu\big)$.

c). $\forall p, q \in V,\ \big(\ \gamma(p).t \neq \bot \wedge (\exists x \in \gamma(p).\mathcal{N}, \gamma(x).t = \bot) \wedge$
$\gamma(q).t \neq \bot \wedge (\exists y \in \gamma(q).\mathcal{N}, \gamma(y).t = \bot)\ \big) \Rightarrow \big(d_\beta\big(\gamma(p).t, \gamma(q).t\big) \leq \mu\big)$.

d). $\forall p \in V,\ \gamma(p).t \neq \bot \Rightarrow \gamma(p).c = \left\lfloor \frac{\alpha}{\beta}\gamma(p).t \right\rfloor$.

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

Lemma 19 (Closure of $\mathcal{L}^d_{\mathsf{PU}}$ under $\mathcal{DSU}$)**.** *The set of configurations $\mathcal{L}^d_{\mathsf{PU}}$ is closed under Algorithm $\mathcal{DSU}$.*

Proof. Let $\gamma \in \mathcal{L}^d_{\mathsf{PU}}$ be a legitimate configuration of Algorithm $\mathcal{DSU}$ and let $\gamma \mapsto_c \gamma'$ be a computation step of Algorithm $\mathcal{DSU}$, from configuration γ. In γ, action $\mathcal{DSU}$-R is disabled for all processes: a process can only execute action $\mathcal{DSU}$-N or $\mathcal{DSU}$-J depending whether its clock is $\bot$ or not.

a). Let $p \in V$ such that $\gamma'(p).t = \bot$. As no action can set $p.t$ to $\bot$, $\gamma(p).t = \bot$ and by Definition 5, $\exists q \in \gamma(p).\mathcal{N}$ such that $\gamma(q).t \neq \bot$. $Locked_q$ holds in γ (because of p). Hence, q is disabled in γ and $\gamma(q).t = \gamma'(q).t \neq \bot$.

b). Let $p \in V$ and $q \in \gamma(p).\mathcal{N}$ such that $\gamma'(p).t \neq \bot$ and $\gamma'(q).t \neq \bot$.

1. If $\gamma(p).t \neq \bot$ and $\gamma(q).t \neq \bot$, as $\gamma \in \mathcal{L}^d_{\mathsf{PU}}$, $d_\beta\big(\gamma(p).t, \gamma(q).t\big) \leq \mu$. Now, p and q can only execute action $\mathcal{DSU}$-N during $\gamma \mapsto_c \gamma'$. If both p and q, or none of them, execute action $\mathcal{DSU}$-N, the delay between $p.t$ and $q.t$ remains the same. If only one of them, without loss of generality assume p, executes action $\mathcal{DSU}$-N, $p.t \preceq_{\beta,\mu} q.t$ holds in γ. So the increment of $p.t$ decreases the delay between $p.t$ and $q.t$. Hence, $d_\beta\big(\gamma'(p).t, \gamma'(q).t\big) \leq \mu$.
2. If $\gamma(p).t = \bot$ and $\gamma(q).t \neq \bot$, as $\gamma \in \mathcal{L}^d_{\mathsf{PU}}$, $\exists x \in \gamma(p).\mathcal{N}$ such that $\gamma(x).t \neq \bot$. We choose x as such a neighbor of p, with minimum value for t, *i.e.*, $\gamma(x).t = MinTime_p$ in γ. Hence, $d_\beta\big(\gamma(x).t, \gamma(q).t\big) \leq \mu$ because of Definition 5.c): q and x have a (common) neighbor p whose t-variable equals $\bot$. q is disabled in γ because of p ($Locked_q$ holds in γ), hence $\gamma(q).t = \gamma'(q).t$. As $\gamma'(p).t = \gamma(x).t$ (since p executes action $\mathcal{DSU}$-J), $d_\beta\big(\gamma'(p).t, \gamma'(q).t\big) \leq \mu$.
3. If $\gamma(p).t \neq \bot$ and $\gamma(q).t = \bot$, similar to case 2.
4. If $\gamma(p).t = \bot$ and $\gamma(q).t = \bot$, as $\gamma \in \mathcal{L}^d_{\mathsf{PU}}$, $\exists x \in p.\mathcal{N}$ such that $\gamma(x).t = MinTime_p \neq \bot$ in γ and $\exists y \in p.\mathcal{N}$ such that $\gamma(y).t = MinTime_q \neq \bot$ in γ. Because of Definition 5.c), $d_\beta\big(\gamma(x).t, \gamma(y).t\big) \leq \mu$ because they have neighbors whose t-variables equal $\bot$ (p and q, respectively). Since p and q execute action $\mathcal{DSU}$-J, $\gamma'(p).t = \gamma(x).t$ and $\gamma'(q).t = \gamma(y).t$, so $d_\beta\big(\gamma'(p).t, \gamma'(q).t\big) \leq \mu$.

c). Let $p, q \in V$ such that $\gamma'(p).t \neq \bot$, $\exists x \in \gamma'(p).\mathcal{N}$ with $\gamma'(x).t = \bot$, $\gamma'(p).t \neq \bot$, and $\exists y \in \gamma'(q).\mathcal{N}$ with $\gamma'(y).t = \bot$.

As no action can set variable t to $\bot$, $\gamma(x).t = \bot$ and $\gamma(y).t = \bot$.

1. If $\gamma(p).t \neq \bot$ and $\gamma(q).t \neq \bot$, as $\gamma \in \mathcal{L}^d_{\mathsf{PU}}$, $d_\beta\big(\gamma(p).t, \gamma(q).t\big) \leq \mu$. Now, p and q are disabled in γ ($Locked_p$, $Locked_q$) because of x and y, respectively. Hence, $d_\beta\big(\gamma'(p).t, \gamma'(q).t\big) \leq \mu$.
2. If $\gamma(p).t = \bot$ and $\gamma(q).t \neq \bot$, as $\gamma \in \mathcal{L}^d_{\mathsf{PU}}$, $\exists x' \in \gamma(p).\mathcal{N}$ such that $\gamma(x').t = MinTime_p \neq \bot$ in γ. Hence, $d_\beta\big(\gamma(x').t, \gamma(q).t\big) \leq \mu$ because they have neighbors whose t-variables equal $\bot$ (p and y, respectively). q is disabled in γ ($Locked_q$) because of y: $\gamma(q).t = \gamma'(q).t$. And $\gamma'(p).t = \gamma(x').t$ since p executes action $\mathcal{DSU}$-J. So $d_\beta\big(\gamma'(p).t, \gamma'(q).t\big) \leq \mu$.
3. If $\gamma(p).t \neq \bot$ and $\gamma(q).t = \bot$, similar to case 2.
4. If $\gamma(p).t = \bot$ and $\gamma(q).t = \bot$, as $\gamma \in \mathcal{L}^d_{\mathsf{PU}}$, $\exists x' \in \gamma(p).\mathcal{N}$ such that $\gamma(x').t = MinTime_p \neq \bot$ in γ and $\exists y' \in \gamma(p).\mathcal{N}$ such that $\gamma(y').t = MinTime_q \neq \bot$ in γ. Hence, $d_\beta\big(\gamma(x').t, \gamma(y').t\big) \leq \mu$ because they have neighbors whose t-variables equal $\bot$ (p and q, respectively). $\gamma'(p).t = \gamma(x').t$ and $\gamma'(q).t = \gamma(y').t$ since p and q execute action $\mathcal{DSU}$-J. So $d_\beta\big(\gamma'(p).t, \gamma'(q).t\big) \leq \mu$.

d). Let $p \in V$ such that $\gamma'(p).t \neq \bot$. Two cases are possible: either p does no action and the constraint between $p.t$ and $p.c$ is preserved, or p executes an action $\mathcal{DSU}$-N. In the latter case, the assignment of the action ensures the constraint.

□

Lemma 20 (Safety of $\mathcal{L}^d_{\mathsf{PU}}$ under $\mathcal{DSU}$)**.** *Safety of SP_{PU} under $\mathcal{DSU}$ is satisfied.*

Proof. Let $\gamma \in \mathcal{L}^d_{\mathsf{PU}}$. By Definition 5, $\forall p \in V$, $\forall q \in \gamma(p).\mathcal{N}$, $\big(\gamma(p).t \neq \bot \wedge \gamma(q).t \neq \bot\big) \Rightarrow d_\beta\big(\gamma(p).t, \gamma(q).t\big) \leq \mu$. Furthermore, $\forall p \in V$, $\gamma(p).t \neq \bot \Rightarrow \gamma(p).c = \left\lfloor \frac{\alpha}{\beta}\gamma(p).t \right\rfloor$. Hence, using Lemma 9 with $d = \mu < K$, $\forall p \in V$, $\forall q \in \gamma(p).\mathcal{N}$, $\big(\gamma(p).t \neq \bot \wedge \gamma(q).t \neq \bot\big) \Rightarrow d_\alpha\big(\gamma(p).c, \gamma(q).c\big) \leq 1$. □

Lemma 21 (Liveness of $\mathcal{L}^d_{\mathsf{PU}}$)**.** *Liveness of SP_{PU} under $\mathcal{DSU}$ is satisfied.*

Proof. Let $e = (\gamma_i)_{i\geq 0} \in \mathcal{E}^0_{\mathcal{L}^d_{\mathsf{PU}}}$. Using Lemma 13, $\exists i \geq 0$ such that $\forall j \geq i$, $\forall p \in V$, $\gamma_j(p).t \neq \bot$. The legitimate configurations $\mathcal{L}^d_{\mathsf{PU}}$ are closed (Lemma 19), so γ_i is a legitimate configuration *w.r.t.* SP_{PU} and is also a legitimate configuration of $\mathcal{WU}$ so p increments infinitely often $p.t$ executing action $\mathcal{DSU}$-N (Lemma 27). Furthermore, actions $\mathcal{DSU}$-R and $\mathcal{DSU}$-J are disabled. So $p.t$ goes through each integer value between

0 and $\beta - 1$ infinitely often (in increasing order). Hence, by Lemma 7, $p.c$ is incremented infinitely often and goes through each integer value between 0 and $\alpha - 1$ (in increasing order). □

Lemma 22. *$\mathcal{DSU}$ converges from $\mathcal{L}^d_{\mathsf{PU}}$ to $\mathcal{L}^d_{\mathsf{WU}}$ in a finite time. The convergence time is at most one round.*

Proof. Let $(\gamma_i)_{i\geq 0} \in \mathcal{E}^0_{\mathcal{L}^d_{\mathsf{PU}}}$. $\forall p \in V$, such that $\gamma_0(p).t = \perp$, action $\mathcal{DSU}$-J is continuously enabled at p. By Lemma 13, $\exists i > 0$, such that $\forall j \geq i$, $\forall p \in V$, $\gamma_j(p).t \neq \perp$. Hence, $\exists j \in \{1, \ldots, i\}$ such that p executes action $\mathcal{DSU}$-J during $\gamma_{j-1} \mapsto \gamma_j$.

So, in at most one round, the system reaches a configuration γ_k, $k \geq 0$, such that $\forall p \in V_k$, $\gamma_k(p).t \neq \perp$. Now, $\gamma_0 \in \mathcal{L}^d_{\mathsf{PU}}$ and $\mathcal{L}^d_{\mathsf{PU}}$ is closed under $\mathcal{DSU}$ (Lemma 19), so $\gamma_k \in \mathcal{L}^d_{\mathsf{PU}}$. As there is no t-variable with $\perp$ value in γ_k, $\gamma_k \in \mathcal{L}^d_{\mathsf{WU}}$. □

Convergence After One Topology Change Step. We now consider executions in $\mathcal{E}^1$ such that $\mathsf{UnderLocalControl}$ holds, namely, after the dynamic step, the graph is still connected and the nodes that were not added design a dominating set.

Lemma 23. *Let $\gamma_i \in \mathcal{L}^d_{\mathsf{SU}}$ be a legitimate configuration under $\mathcal{DSU}$, $\gamma_i \mapsto_d \gamma_{i+1}$ be a dynamic step, such that $\forall p \in V_{i+1}$, $\gamma_{i+1}(p).t = \perp \Rightarrow (\exists q \in \gamma_{i+1}(p).\mathcal{N}, \gamma_{i+1}(q).t \neq \perp)$, then $\gamma_{i+1} \in \mathcal{L}^d_{\mathsf{PU}}$.*

Proof. Let $\gamma_i \mapsto_d \gamma_{i+1}$ be a dynamic step such that $\gamma_i \in \mathcal{L}^d_{\mathsf{SU}}$ and $\forall p \in V_{i+1}$, $\gamma_{i+1}(p).t = \perp \Rightarrow (\exists q \in \gamma_{i+1}(p).\mathcal{N}, \gamma_{i+1}(q).t \neq \perp)$. By Definition 3, $\forall p \in V_i$, $\gamma_i(p).t \neq \perp$, $\gamma_i(p).c = \left\lfloor \frac{\alpha}{\beta}\gamma_i(p).t \right\rfloor$, and $\forall q \in \gamma_i(p).\mathcal{N}$, $d_\beta\big(\gamma_i(p).t, \gamma_i(q).t\big) \leq 1$. Hence $\forall p, q \in V_i$, $d_\beta\big(\gamma_i(p).t, \gamma_i(q).t\big) \leq \mu$.

Now, after the dynamic step, in γ_{i+1}, the state of processes that are in $V_i \cap V_{i+1}$ remains the same. So, $\forall p \in V_{i+1}$, $\forall q \in \gamma_{i+1}(p).\mathcal{N}$, if $\gamma_{i+1}(p).t \neq \perp$ and $\gamma_{i+1}(q).t \neq \perp$, then $p, q \in V_i \cap V_{i+1}$. As $d_\beta\big(\gamma_i(p).t, \gamma_i(q).t\big) \leq \mu$ (Remark 4), $d_\beta\big(\gamma_{i+1}(p).t, \gamma_{i+1}(q).t\big) \leq \mu$.

Finally, $\forall p \in V_{i+1}$, if $\gamma_{i+1}(p).t \neq \perp$, $p \in V_i \cap V_{i+1}$ so $\gamma_{i+1}(p).c = \gamma_i(p).c = \left\lfloor \frac{\alpha}{\beta}\gamma_i(p).t \right\rfloor = \left\lfloor \frac{\alpha}{\beta}\gamma_{i+1}(p).t \right\rfloor$.

Hence, $\gamma_{i+1} \in \mathcal{L}^d_{\mathsf{PU}}$. □

Lemma 24. *$\mathcal{DSU}$ converges from $\mathcal{L}^d_{\mathsf{WU}}$ to $\mathcal{L}^d_{\mathsf{SU}}$ in a finite time. The convergence time is at most $(\mu + 1)\mathcal{D}_1 + 1$ rounds.*

Proof. Let $e = (\gamma_i)_{i\geq 0} \in \mathcal{E}^0_{\mathcal{L}^d_{\mathsf{WU}}}$. The behavior of the algorithm is similar to the one of $\mathcal{WU}$ (Remarks 3 and 5). Furthermore, $\forall p \in V$, $\forall q \in p.\mathcal{N}$, $d_\beta\big(\gamma_0(p).t, \gamma_0(q).t\big) \leq \mu$, so $\gamma_0 \in C_\mu$. By Lemma 29, in a finite time, $\forall p \in V$, $\forall q \in p.\mathcal{N}$, $d_\beta\big(\gamma_0(p).t, \gamma_0(q).t\big) \leq 1$. This convergence lasts at most $\mu\mathcal{D}_1$ rounds (Theorem 11).

The liveness of weak unison is ensured in e (Lemma 18), so each process eventually increments its clock executing $\mathcal{DSU}$-N and updates its c-variable. By Lemma 4, the c-variables are well computed according to t-variables in at most $\mathcal{D}_1 + 1$ additional rounds. Hence, in at most $(\mu + 1)\mathcal{D}_1 + 1$ rounds, the system reaches a $\mathcal{L}^d_{\mathsf{SU}}$. □

Using Theorem 6, and Lemmas 16 to 22, we can conclude:

Theorem 8. *If* UnderLocalControl *is satisfied then Algorithm $\mathcal{DSU}$ is gradually stabilizing under 1-dynamics for $(SP_{\mathsf{PU}} \bullet 0, SP_{\mathsf{WU}} \bullet 1, SP_{\mathsf{SU}} \bullet (\mu+1)\mathcal{D}_1+2)$, where $\mathcal{D}_1$ (resp. $n+\#J$) is the diameter (resp. an upper bound on the size) of the network after the dynamic step and μ is a parameter satisfying $\mu \geq n + \#J$.*

Theorem 9 introduce a result on how often a process increments its clocks since convergence to legitimate configurations *w.r.t.* SP_{SU} or SP_{WU}.

Theorem 9. *After convergence of $\mathcal{DSU}$ to $\mathcal{L}^d_{\mathsf{WU}}$ (resp. $\mathcal{L}^d_{\mathsf{SU}}$), 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).*

Proof. By Remarks 3 and 5, we can use results on $\mathcal{WU}$ for $\mathcal{DSU}$. If $\mathcal{DSU}$ converged to a configuration $\gamma \in \mathcal{L}^d_{\mathsf{WU}}$, then $\gamma \in C_\mu$. So, by Lemma 3, after $\mu\mathcal{D}_1 + \frac{\beta}{\alpha}$ rounds, p increments $p.t$ at least $\frac{\beta}{\alpha}$ times. Now, by Lemma 7, if t-variable is incremented $\frac{\beta}{\alpha}$ times, c-variable is incremented once.

If $\mathcal{DSU}$ converged to $\mathcal{L}^d_{\mathsf{SU}}$, the result of Theorem 5 can be applied (Remark 5). 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 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] Anish Arora, Shlomi Dolev, and Mohamed G. Gouda. Maintaining digital clocks in step. *Parallel Processing Letters*, 1:11–18, 1991.

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

[3] Joffroy Beauquier, Christophe Genolini, and Shay Kutten. k-stabilization of reactive tasks. In *PODC*, page 318, 1998.

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

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

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

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

[8] 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.

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

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

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

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

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

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

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

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

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

[18] 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.

[19] 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.

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

[21] 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.

[22] Sayaka Kamei and Hirotsugu Kakugawa. A self-stabilizing approximation algorithm for the minimum weakly connected dominating set with safe convergence. In *WRAS*, pages 57–67, 2007.

[23] 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.

[24] 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.

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

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

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

[28] 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.

A Results from [6]

In this section, we recall some useful technical results from [6] about Algorithm $\mathcal{WU}$. Algorithm $\mathcal{WU}$ is an instance of the parametric algorithm GAU in [6]: $\mathcal{WU} = GAU(\beta, 0, \mu)$.

The following five lemmas (25-29) are used to establish the self-stabilization of $\mathcal{WU}$ for SP_{WU} by the set of legitimate configurations defined as $\mathcal{L}_{\mathsf{WU}}$, where $\gamma \in \mathcal{L}_{\mathsf{WU}}$ if and only if $\forall p \in V$, $\forall q \in \gamma(p).\mathcal{N}$, $d_\beta\big(\gamma(p).t, \gamma(q).t\big) \leq 1$.

The proof of self-stabilization is divided into several steps. The first step (Lemma 26) consists in showing the convergence of $\mathcal{WU}$ from $\mathcal{C}$ to C_μ, where C_μ is the set of configurations where the distance 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 25. (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 27.

Lemma 25 (Property 8 in [6]). *C_μ is closed under $\mathcal{WU}$.*

Lemma 26 (Theorem 56 in [6]). *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 27 (Theorem 21 in [6]). *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 28) and the convergence from C_μ to $\mathcal{L}_{\mathsf{WU}}$ (Lemma 29). Regarding the correctness, the safety part of SP_{WU} (two neighbor clocks differ from at most 1) is ensured by definition of $\mathcal{L}_{\mathsf{WU}}$, whereas the liveness part is already ensured by Lemma 27. Precisely,

Lemma 28 (Property 2 in [6]). *$\mathcal{L}_{\mathsf{WU}}$ is closed under $\mathcal{WU}$.*

Lemma 29 (Theorems 29 in [6]). *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 already recalled in Theorems 10 and 11 (pages 39 and 39).

Theorem 10 (Theorem 61 in [6]). *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 11 (Theorems 20 and 28 in [6]). *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 30 below is a technical result about the values of t-variables.

Lemma 30 (Theorem 20, Property 27, and Lemma 22 in [6])**.** *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 function f on processes such that*

- $\forall i \geq 0$, $\forall p \in V$, $f(\gamma_i, p) \mod \beta = \gamma_i(p).t$,
- *and* $\forall i \geq 0$, $\forall p, q \in V$, $|f(\gamma_i, p) - f(\gamma_i, q)| = d_\beta\big(\gamma_i(p).t, \gamma_i(q).t\big)$.