On Implementing Stabilizing Leader Election with Weak Assumptions on Network Dynamics

Karine Altisen, Stéphane Devismes, Anaïs Durand, Colette Johnen, Franck Petit

On Implementing Stabilizing Leader Election with Weak Assumptions on Network Dynamics *

Karine Altisen, Stéphane Devismes, Anaïs Durand, Colette Johnen, and Franck Petit

October 27, 2020

Abstract

In this paper, we consider self-stabilization and its weakened form called pseudo-stabilization. We study conditions under which (pseudo- and self-) stabilizing leader election is solvable in networks subject to frequent topological changes. To model such an high dynamics, we use the dynamic graph (DG) paradigm and study a taxonomy of nine important DG classes.

Our results show that self-stabilizing leader election can be achieved only in the classes where all processes are sources. However, we also show that, among those classes, the convergence time of pseudo- and so self-stabilizing solutions can only be bounded in the class where all sources are actually timely.

Furthermore, even pseudo-stabilizing leader election cannot be solved in all remaining classes, except in the class where at least one process is a timely source. We illustrate this latter claim by proposing a pseudo-stabilizing leader election algorithm for this class. We also show that in this class, the convergence time of pseudo-stabilizing leader election algorithms cannot be bounded. Nevertheless, we show that our solution is speculative since its convergence time can be bounded when the dynamics is not too erratic, precisely when all processes are timely sources.

Keywords: leader election, dynamic graphs, self-stabilization, pseudo-stabilization, speculation, sources, sinks, timely sources, timely sinks.

1 Introduction

Leader election is a fundamental problem in distributed computing. Notably, it is often used as a basic building block in the design of more complex crucial tasks such as spanning tree constructions, broadcasts, and convergecasts. Consequently, leader election is commonly used as a benchmark problem to explore new models or environments.

Modern networks —*e.g.*, MANET (*Mobile Ad-Hoc Networks*), VANET (*Vehicular Ad-Hoc Networks*), and DTN (*Delay-Tolerant Networks*)— are prone to both *faults* and *frequent alteration of their topology* (*i.e.*, the addition or the removal of communication links or nodes). Several works aim at capturing such a network dynamics using graph-based models. In [10, 21], the network dynamics is represented as a sequence of digraphs called *evolving or dynamic graphs*. In [9], the topological evolution of the network is modeled by a *Time-Varying Graphs* (TVGs, for short). A TVG consists of a (fixed) digraph and a presence function which indicates whether or not a given arc of the digraph exists at a given time. The way a network is connected

*This study has been partially supported by the French ANR projects ANR-16-CE40-0023 (DESCARTES) and ANR-16 CE25-0009-03 (ESTATE).

over time is of prime interest to develop algorithmic solutions. On the other hand, an algorithmic solution should be as general as possible and so not focus on a single dynamic pattern. Therefore, dynamic patterns are often classified according to the temporal characteristics of edge presence they satisfy; see, *e.g.*, [9]. Such taxonomies are always proposed together with possibility/impossibility results related to the strength of classes that compose them.

In [2], we have initiated research to unify *fault-tolerance* and *endurance to high-frequency of topological changes* by proposing self-stabilizing algorithms for the leader election problem in three important classes of dynamic networks, modeled using the TVG paradigm. In the present paper, we explore conditions on the network dynamics under which the (deterministic) stabilizing leader election is solvable. The term "stabilization" covers here both *self-stabilization* [13] and its weakened deterministic variant called *pseudo-stabilization* [7] (*n.b.*, by definition, any self-stabilizing algorithm is pseudo-stabilizing, but the reverse is not necessarily true). Essentially, these two properties express the ability of a distributed algorithm to withstand transient faults, *i.e.*, faults (such as memory corruption) that occur at an unpredictable time, but do not result in a permanent hardware damage and whose frequency is low. Indeed, starting from an arbitrary configuration (which may be the result of transient faults), a self-stabilizing algorithm makes a distributed system reach within finite time a configuration from which all possible execution suffixes satisfy the intended specification. In contrast, an algorithm is pseudo-stabilizing if all its executions, starting from arbitrary configurations, have a suffix satisfying the intended specification. Up to now, these two stabilizing properties have been mainly studied in static systems, *i.e.*, assuming a network with a fixed topology. In highly dynamic networks, the design of stabilizing solutions is challenging since the network topology continuously evolves over the time. Notably, stabilization should be achieved in spite of the high-frequency of topological changes occurring throughout the convergence.

As is shown by this paper, pseudo- (and so self-) stabilization is often simply impossible to achieve when the dynamics is too erratic. Furthermore, when stabilization is possible, the convergence time of stabilizing solutions is often unboundable. In such cases, *speculation* [20] makes sense. Indeed, an algorithm is speculative whenever it satisfies its requirements for all executions, but also exhibits significantly better performances in a subset of more probable executions. Speculative self-stabilization has been initially investigated in static networks [18]. Yet, recently, we have proposed speculative self-stabilizing solutions for dynamic networks [2]: when the convergence time cannot be bounded in a very general class, we try to exhibit an important subclass where it can be.

Contribution. In this paper, we study conditions under which stabilizing leader election can be solved in highly dynamic identified message passing systems. We model the network dynamics using the dynamic graph (DG for short) paradigm [10]. Moreover, we assume that processes can synchronously communicate using local broadcast primitives: at each round, every process can send a common message to its unknown set of current neighbors (if any).

Due to the intrinsic absence of termination detection, stabilizing leader election is close to the *eventual leader election* problem [12] (usually denoted by Ω) studied in crash-prone partially synchronous, yet static and fully connected, systems. In this problem, all correct processes should eventually always designate the same correct process as leader [1]. We propose here to study the stabilizing leader election problem in DG classes defined by analogy with the classes of partially synchronous systems investigated for Ω. In those latter classes, the notions of source, sink, and timeliness are central. We first accommodate these concepts to the DG paradigm. Roughly speaking, in a DG, a *source* is a process which is infinitely often able to reach all other ones by flooding. It is a *quasi-timely source* if, using flooding, it can infinitely often reach all other processes within some bounded time Δ. Finally, a timely source is a source that can always reach (still by

flooding) all other processes within some bounded time Δ. Conversely, a sink is a process which is infinitely often reachable by all other ones using flooding. *Quasi-timely* and *timely sinks* are defined similarly to the quasi-timely and timely sources. Using these notions we consider a taxonomy of nine DG classes:

- $\mathcal{J}_{1,*}$, $\mathcal{J}^Q_{1,*}(\Delta)$, and $\mathcal{J}^B_{1,*}(\Delta)$ contain every DG where at least one (*a priori* unknown) process is a source, a quasi-timely source, and a timely source, respectively.
- $\mathcal{J}_{*,1}$, $\mathcal{J}^Q_{*,1}(\Delta)$, and $\mathcal{J}^B_{*,1}(\Delta)$ contain every DG where at least one (*a priori* unknown) process is a sink, a quasi-timely sink, and a timely sink, respectively.
- Finally, $\mathcal{J}_{*,*}$, $\mathcal{J}^Q_{*,*}(\Delta)$, and $\mathcal{J}^B_{*,*}(\Delta)$ (the three classes studied in [2]) contain every DG where all processes are sources, quasi-timely sources, and timely sources, respectively.

The hierarchy among all classes of this taxonomy is presented in Figure 2, page 7.

We first study the possibility of designing stabilizing leader election in those classes. Our results are summarized in Figure 1. From [2], we already know that self-stabilizing leader election can be achieved in $\mathcal{J}_{*,*}$, $\mathcal{J}^Q_{*,*}(\Delta)$, and $\mathcal{J}^B_{*,*}(\Delta)$. However, in this paper, we show that those three classes are actually the only ones of our taxonomy where it is possible. Moreover, we show that, among those classes, the convergence time of pseudo- and so self-stabilizing solutions can only be bounded in $\mathcal{J}^B_{*,*}(\Delta)$. By the way, we proposed an asymptotically time-optimal self-stabilizing leader election algorithm for $\mathcal{J}^B_{*,*}(\Delta)$ in [2].

Furthermore, we establish that in the remaining classes, even pseudo-stabilizing leader election is not possible, except in the class $\mathcal{J}^B_{1,*}(\Delta)$. For this latter class, we propose a pseudo-stabilizing leader election algorithm, called Algorithm $\mathcal{LE}$. Now, we also show that, in general, convergence time of pseudo-stabilizing leader election algorithms cannot be bounded in $\mathcal{J}^B_{1,*}(\Delta)$. Nevertheless, we show that Algorithm $\mathcal{LE}$ is speculative in the sense that its convergence time in $\mathcal{J}^B_{*,*}(\Delta) \subset \mathcal{J}^B_{1,*}(\Delta)$ is at most $6\Delta + 2$ rounds.

Finally, we give a preliminary result on the space complexity of any pseudo-stabilizing leader election solution in $\mathcal{J}^B_{1,*}(\Delta)$, the class where only pseudo-stabilization can be achieved for the leader election problem. Namely, we show that the memory requirement of any pseudo-stabilizing leader election algorithm for this class is finite only if it depends on Δ.

Related Work. Ensuring convergence in highly dynamic networks regardless of the initial configuration is challenging, even impossible in many cases [6]. However, perhaps surprisingly, a number of self-stabilizing works, *e.g.*, [11, 14, 17] (*n.b.*, [11] deals with leader election) are advertised as solutions suited for dynamic networks. Actually, those works propose self-stabilizing algorithms dedicated to arbitrary network topologies and do not propose any specific patch to handle topological changes. Consequently, they tolerate topological changes only if they are eventually detected locally at involved processes and if the frequency of such events is low enough. Indeed, in such a case, topological changes can be seen as transient faults. Furthermore, several approaches derived from self-stabilization, *e.g.*, *superstabilization* [16] and *gradual stabilization* [3], aims at additionally providing countermeasures to efficiently treat topological changes when they are both spatially and timely sparse (*i.e.*, transient). Overall, all these approaches become totally ineffective when the frequency of topological changes drastically increases.

Actually and to the best of our knowledge, only few self-stabilizing works [2, 5, 8, 15] deal with a highly dynamic context. The present paper is a follow-up of [2], in which we propose self-stabilizing solutions for the leader election in three important classes of dynamic networks. Here we complete the panorama by studying more general classes. All other aformentionned works, *i.e.*, [5, 8, 15], use widely different models and assumptions than ours, as explained below.

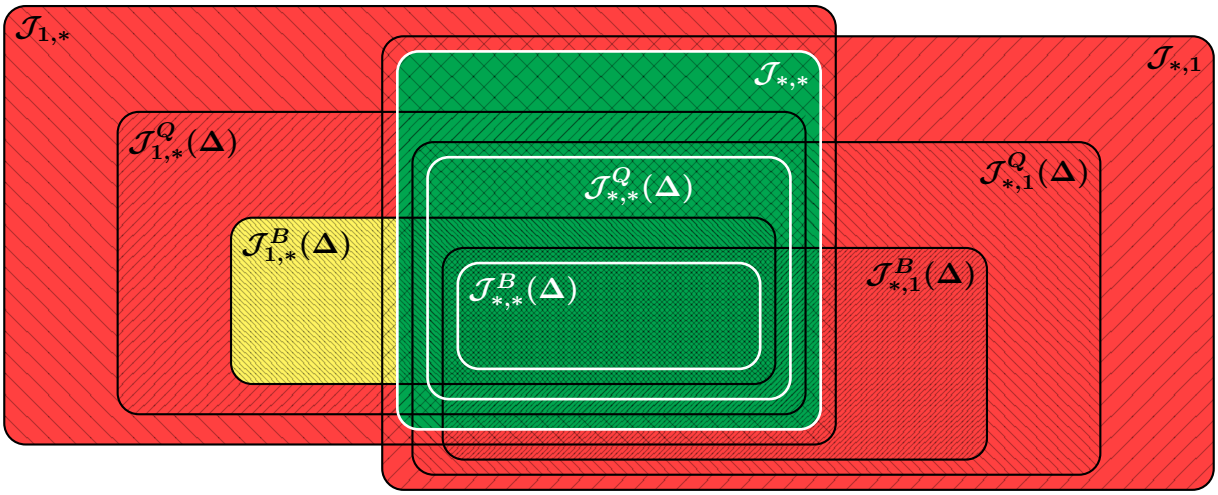

Figure 1: Stabilizing leader election: summary of the results. Pseudo- (and so self-) stabilizing leader election is impossible in the red area. Self- (and so pseudo-) stabilizing leader election is possible in the green area. In the yellow area, only pseudo-stabilization can be achieved.

In [5], authors consider the self-stabilizing exploration of a *highly dynamic ring* by a cohort of synchronous robots equipped with visibility sensors, moving actuators, yet no communication capabilities. Contrary to [5], our classes never enforce the network to have a particular topology at a given time. In [8], Cai *et al.* tackles the self-stabilizing leader election problem in highly dynamic systems through the population protocol model. In this model, communications are achieved by atomic rendezvous between pairs of anonymous processes, where ties are nondeterministically broken. The local broadcast primitive we use here is weaker. Moreover, authors assume global fairness, meaning that every configuration which is infinitely often reachable is actually infinitely often reached. We do not make such an assumption here. Finally, Dolev *et al.* [15] assume the system is equipped with a routing algorithm, which allows any two processes to communicate, providing that the sender knows the identifier of the receiver. This blackbox protocol abstracts the dynamics of the system: the dynamics makes it fair lossy, non-FIFO, and duplication-prone. Moreover, the channel capacity is assumed to be bounded. Based on this weak routing algorithm, they build a stronger routing protocol which is reliable, FIFO, and which prevents duplication. Again, the communication primitive we assume here is drastically weaker. As a matter of facts, in the several classes studied here all-to-all communication is simply impossible.

Roadmap. In Section 2, we present our taxonomy of DG classes and the computational model. In the same section, we define the two stabilizing variants we will consider as well as the leader election problem. Section 3 is dedicated to our impossibility results and lower bounds. Our speculative pseudo-stabilizing leader election algorithm for $\mathcal{J}^{B}_{1,*}(\Delta)$ is proposed in Section 4. We make concluding remarks in Section 6.

2 Preliminaries

2.1 Dynamic Graphs

Dynamicity refers to the structure of communication links between processes that evolves over the time. We model this using the so-called *dynamic graphs* [10].

In a dynamic graph, the set of vertices (representing processes of the network) is fixed, but the set of edges may evolve. So, a dynamic graph is a sequence of graphs gathering the same set of vertices, but where the set of edges can be different from one graph to another. In such a model, the notion of "static" path used in classical graph theory should be adapted as path over time, called here *journey*, since the existence of a given edge depends on the instant we consider. Accordingly, the classical notions of distance and diameter should be re-engineered as *temporal distance* and *temporal diameter*, respectively.

The next paragraph provides the formal definitions of dynamic graph, journey, temporal distance, and temporal diameter. Next, we will define the important classes of dynamic graphs that will be studied in this paper.

2.1.1 Dynamic Graph Model [10]

For any directed graph G, we denote by $V(G)$ its vertex set and by $E(G)$ its set of oriented edges. A *dynamic graph $\mathcal{G}$ with vertex set V* (DG for short) is an infinite sequence of directed loopless graphs $G_1, G_2, ...$ such that $V(G_i) = V$, for every $i \in \mathbb{N}^*$. Notice that our definition of DG is close to the model, called *evolving graphs*, defined in [21]. For every $i \in \mathbb{N}^*$, we denote by $\mathcal{G}_{i\triangleright}$ the dynamic graph $G_i, G_{i+1}, ...$ with vertex set V, *i.e.*, the suffix of $\mathcal{G}$ starting from position i.

A *journey* in $\mathcal{G}$ from Vertex p to Vertex q is a finite non-empty sequence of pairs $\mathcal{J} = (e_1, t_1), (e_2, t_2), ..., (e_k, t_k)$ where $\forall i \in \{1, \ldots, k\}$, $e_i = (p_i, q_i) \in E(G_{t_i})$, $i < k \Rightarrow q_i = p_{i+1} \wedge t_i < t_{i+1}$, $p_1 = p$, and $q_k = q$. We respectively denote by $departure(\mathcal{J})$ and $arrival(\mathcal{J})$ the *starting position* t_1 and the *arrival position* t_k of $\mathcal{J}$. The *temporal length* of $\mathcal{J}$ is equal to $arrival(\mathcal{J}) - departure(\mathcal{J}) + 1$. We denote by $\mathcal{J}_{\mathcal{G}}(p, q)$ the set of journeys from p to q in $\mathcal{G}$. We let $\overset{\mathcal{G}}{\rightsquigarrow}$ to be the binary relation over V such that $p \overset{\mathcal{G}}{\rightsquigarrow} q$ if and only if $p = q$ or there exists a journey from p to q in $\mathcal{G}$.

The *temporal distance* from p to q in $\mathcal{G}$, $\hat{d}_{\mathcal{G}}(p, q)$, is defined as follows: $\hat{d}_{\mathcal{G}}(p, q) = 0$, if $p = q$, $\hat{d}_{\mathcal{G}}(p, q) = \min\{arrival(\mathcal{J}) : \mathcal{J} \in \mathcal{J}_{\mathcal{G}}(p, q)\}$ otherwise (by convention, we let $\min \emptyset = +\infty$).

The *temporal distance from p to q in $\mathcal{G}$ at position $i \in \mathbb{N}^*$*, $\hat{d}_{\mathcal{G},i}(p, q)$, is the temporal distance from p to q in $\mathcal{G}_{i\triangleright}$, *i.e.*, $\hat{d}_{\mathcal{G}_{i\triangleright}}(p, q)$. Similarly, the *temporal diameter* in $\mathcal{G}$ at position $i \in \mathbb{N}^*$ is the maximum temporal distance at position i between any two vertices in $\mathcal{G}$.

For all aforementioned notations, we omit the subscript $\mathcal{G}$ when it is clear from the context.

2.1.2 Dynamic Graph Classes

A class of dynamic graphs is defined as a particular set of dynamic graphs. A DG class $\mathcal{C}$ is said to be *recurring* if and only if for every DG $\mathcal{G} \in \mathcal{C}$, we have $\forall i \in \mathbb{N}^*$, $\mathcal{G}_{i\triangleright} \in \mathcal{C}$; in other words, every recurring DG class is suffix-closed.

We will consider the nine recurring DG classes in this paper. They are formally defined in Table 1, 2, and 3.

The first three classes (Table 1) contain dynamic graphs with at least one (*a priori* unknown) *(temporal) source*, *i.e.*, a vertex from which every vertex can be reached infinitely often *via* a journey. These classes are said to be "one to all" or classes with a source, and are indexed by $1, *$. They actually differ by the

timing guarantees on journeys they offer. In the first class, except from the existence of a source, there is no guarantees. The second class contains a *timely source*,[1] in the sense that its temporal distance to any vertex is always bounded by some value $\Delta \in \mathbb{N}^*$; the class name is then written with the superscript B. The third class guarantees that there is a source such that at any time and for any vertex, we have wait for some (finite yet unbounded) time before the source can reach this vertex within a bounded temporal distance. Hence, we say that the distance is quasi-bounded and so the source is a *quasi-timely source*; the class name then contains the superscript Q.

The second set of three classes (see Table 2) contains dynamic graphs with at least one *(temporal) sink*, namely a vertex that can be reached infinitely often by any other vertices *via* a journey; These classes are said to be "all to one" or classes with a sink, and are indexed by $*, 1$. Similarly to the classes with a source, they differ by their timing guarantees on journeys they offer.

The last three classes (see Table 3) contain dynamic graphs where every vertex is a source (and so a sink). Hence, they are called "all to all" and indexed by $*, *$. Using the same classification as for the previous classes, these three classes have been already defined in [9], [19] and [2] respectively.

Class $\mathcal{J}_{1,*}$	At least one vertex, called a *(temporal) source*, can reach all the other vertices infinitely often through a journey: $\mathcal{G} \in \mathcal{J}_{1,*}$ iff $\exists src \in V, \forall p \in V, \forall i \in \mathbb{N}^*, src \overset{\mathcal{G}_{i\triangleright}}{\rightsquigarrow} p$
Class $\mathcal{J}^B_{1,*}(\Delta)$	At least one vertex, called a *timely source*, is always at temporal distance at most Δ from all other vertices: $\mathcal{G} \in \mathcal{J}^B_{1,*}(\Delta)$ iff $\exists src \in V, \forall p \in V, \forall i \in \mathbb{N}^*, \hat{d}_{\mathcal{G},i}(src, p) \leq \Delta$
Class $\mathcal{J}^Q_{1,*}(\Delta)$	At least one vertex, called a *quasi-timely source*, is infinitely often at temporal distance at most Δ from each other vertex: $\mathcal{G} \in \mathcal{J}^Q_{1,*}(\Delta)$ iff $\exists src \in V, \forall p \in V, \forall i \in \mathbb{N}^*, \exists j \geq i, \hat{d}_{\mathcal{G},j}(src, p) \leq \Delta$

Table 1: Classes with a source – indexed by $1, *$ ($\mathcal{G}$ is a dynamic graph with vertex set V and $\Delta \in \mathbb{N}^*$).

Class $\mathcal{J}_{*,1}$	At least one vertex, called a *sink*, can be reached by all the other vertices infinitely often through a journey: $\mathcal{G} \in \mathcal{J}_{*,1}$ iff $\exists snk \in V, \forall p \in V, \forall i \in \mathbb{N}^*, p \overset{\mathcal{G}_{i\triangleright}}{\rightsquigarrow} snk$
Class $\mathcal{J}^B_{*,1}(\Delta)$	Every vertex is always at temporal distance at most Δ of at least one given vertex, called a *timely sink*: $\mathcal{G} \in \mathcal{J}^B_{*,1}(\Delta)$ iff $\exists snk \in V, \forall p \in V, \forall i \in \mathbb{N}^*, \hat{d}_{\mathcal{G},i}(p, snk) \leq \Delta$
Class $\mathcal{J}^Q_{*,1}(\Delta)$	Every vertex is infinitely often at temporal distance at most Δ from at least one given vertex, called a *quasi-timely sink*: $\mathcal{G} \in \mathcal{J}^Q_{*,1}(\Delta)$ iff $\exists snk \in V, \forall p \in V, \forall i \in \mathbb{N}^*, \exists j \geq i, \hat{d}_{\mathcal{G},j}(p, snk) \leq \Delta$

Table 2: Classes with a sink – indexed by $*, 1$ ($\mathcal{G}$ is a dynamic graph with vertex set V and $\Delta \in \mathbb{N}^*$).

Remark 1. *By definition, for every $x \in \{1, *; *, 1; *, *\}$ and $y \in \{B, Q\}$, for every $\Delta' \geq \Delta$, we have*

$$\mathcal{J}^y_x(\Delta) \implies \mathcal{J}^y_x(\Delta')$$

[1] We have chosen this name by analogy with timely sources in partially synchronous systems [12].

Class $\mathcal{J}_{*,*}$	Every vertex can always reach all the others through a journey: $\mathcal{G} \in \mathcal{J}_{*,*}$ iff $\forall p \in V, \forall q \in V, \forall i \in \mathbb{N}^*, p \overset{\mathcal{G}_{i\triangleright}}{\rightsquigarrow} q$
Class $\mathcal{J}^B_{*,*}(\Delta)$	Every vertex is always at temporal distance at most Δ from all other vertices: $\mathcal{G} \in \mathcal{J}^B_{*,*}(\Delta)$ iff $\forall p \in V, \forall q \in V, \forall i \in \mathbb{N}^*, \hat{d}_{\mathcal{G},i}(p,q) \leq \Delta$
Class $\mathcal{J}^Q_{*,*}(\Delta)$	Every vertex is infinitely often at temporal distance at most Δ from each other vertex: $\mathcal{G} \in \mathcal{J}^Q_{*,*}(\Delta)$ iff $\forall p, q \in V, \forall i \in \mathbb{N}^*, \exists j \geq i, \hat{d}_{\mathcal{G},j}(p,q) \leq \Delta$

Table 3: Classes where every vertex is a sink and a source – indexed by $*, *$ with $\mathcal{G}$ a dynamic graph with vertex set V and $\Delta \in \mathbb{N}^*$.

By definition, for every $\Delta \in \mathbb{N}^*$, we have the inclusions depicted in Figure 2. Furthermore, as stated by the theorem below, there is no inclusion between any other pair of classes. In particular, this means that inclusions in Figure 2 are strict.

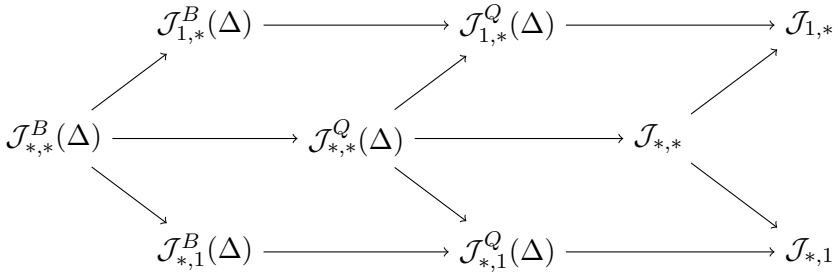

Figure 2: Hierarchy of the studied DG classes. $A \to B$ means that $A \subset B$.

Theorem 1. *$\forall \Delta \in \mathbb{N}^*$, the inclusions given in Figure 2 hold. These inclusions are strict and there is no inclusion between any other pair of classes.*

Proof. First, for every $\Delta \in \mathbb{N}^*$, the inclusions in Figure 2 hold by definition of the classes. Now, regarding non-inclusions, the proof is split into three parts. Each number in Figure 3 refers to one of the part.

Let Δ be a positive number and $V = \{v_1, ..., v_n\}$ be a set of $n > 1$ vertices.

$\not\subseteq$ **(1)** *This first part of the proof compares DGs with a source (resp. a sink) against all to all DGs.*

Let $S = (V, E_S)$ be a out-directed star graph, *i.e.*, $E_S = \{(v_1, v_i) : i \in \{2, ..., n\}\}$. Let $T = (V, E_T)$ be a out-directed star graph, *i.e.*, $E_T = \{(v_i, v_1) : i \in \{2, ..., n\}\}$. See Figure 4 for illustrative examples.

We consider the dynamic graph $\mathcal{G}_{(1S)} = S, S, ...$ with vertex set V.

Obviously, $\mathcal{G}_{(1S)} \in \mathcal{J}_{1,*}, \mathcal{J}^Q_{1,*}(\Delta)$ and $\mathcal{J}^B_{1,*}(\Delta)$, since at any time, v_1 can directly reach any other vertex using one single edge.

But $\mathcal{G}_{(1S)} \notin \mathcal{J}_{*,*}, \mathcal{J}^Q_{*,*}(\Delta), \mathcal{J}^B_{*,*}(\Delta)$, indeed v_1 can never be reached.

Moreover, by transitivity, $\mathcal{G}_{(1S)} \notin \mathcal{J}_{*,1}, \mathcal{J}^Q_{*,1}(\Delta), \mathcal{J}^B_{*,1}(\Delta)$. Hence, the non-inclusions between those classes.

	$\mathcal{J}^B_{1,*}(\Delta)$	$\mathcal{J}^B_{*,*}(\Delta)$	$\mathcal{J}^B_{*,1}(\Delta)$	$\mathcal{J}^Q_{1,*}(\Delta)$	$\mathcal{J}^Q_{*,*}(\Delta)$	$\mathcal{J}^Q_{*,1}(\Delta)$	$\mathcal{J}_{1,*}$	$\mathcal{J}_{*,*}$	$\mathcal{J}_{*,1}$
$\mathcal{J}^B_{1,*}(\Delta)$	—	$\not\subset$ (1)	$\not\subset$ (1)	$\subset$	$\not\subset$ (1)	$\not\subset$ (1)	$\subset$	$\not\subset$ (1)	$\not\subset$ (1)
$\mathcal{J}^B_{*,*}(\Delta)$	$\subset$	—	$\subset$	$\subset$	$\subset$	$\subset$	$\subset$	$\subset$	$\subset$
$\mathcal{J}^B_{*,1}(\Delta)$	$\not\subset$ (1)	$\not\subset$ (1)	—	$\not\subset$ (1)	$\not\subset$ (1)	$\subset$	$\subset$	$\not\subset$ (1)	$\not\subset$ (1)
$\mathcal{J}^Q_{1,*}(\Delta)$	$\not\subset$ (2)	$\not\subset$ (1)	$\not\subset$ (1)	—	$\not\subset$ (1)	$\not\subset$ (1)	$\subset$	$\not\subset$ (1)	$\not\subset$ (1)
$\mathcal{J}^Q_{*,*}(\Delta)$	$\not\subset$ (2)	$\not\subset$ (2)	$\not\subset$ (2)	$\subset$	—	$\subset$	$\subset$	$\subset$	$\subset$
$\mathcal{J}^Q_{*,1}(\Delta)$	$\not\subset$ (1)	$\not\subset$ (1)	$\not\subset$ (2)	$\not\subset$ (1)	$\not\subset$ (1)	—	$\not\subset$ (1)	$\not\subset$ (1)	$\subset$
$\mathcal{J}_{1,*}$	$\not\subset$ (3)	$\not\subset$ (1)	$\not\subset$ (1)	$\not\subset$ (3)	$\not\subset$ (1)	$\not\subset$ (1)	—	$\not\subset$ (1)	$\not\subset$ (1)
$\mathcal{J}_{*,*}$	$\not\subset$ (3)	$\not\subset$ (3)	$\not\subset$ (3)	$\not\subset$ (3)	$\not\subset$ (3)	$\not\subset$ (3)	$\subset$	—	$\subset$
$\mathcal{J}_{*,1}$	$\not\subset$ (1)	$\not\subset$ (1)	$\not\subset$ (3)	$\not\subset$ (1)	$\not\subset$ (1)	$\not\subset$ (3)	$\not\subset$ (1)	$\not\subset$ (1)	—

Figure 3: Relations between classes

Figure 4: The star graph S with a source and the star graph T with a sink.

Similarly, we consider the dynamic graph $\mathcal{G}_{(1T)} = T, T, ...$ with vertex set V.

Obviously, $\mathcal{G}_{(1T)} \in \mathcal{J}_{*,1}, \mathcal{J}^Q_{*,1}(\Delta), \mathcal{J}^B_{*,1}(\Delta)$. But, $\mathcal{G}_{(1T)} \notin \mathcal{J}_{*,*}, \mathcal{J}^Q_{*,*}(\Delta), \mathcal{J}^B_{*,*}(\Delta)$. Moreover, by transitivity, $\mathcal{G}_{(1T)} \notin \mathcal{J}_{1,*}, \mathcal{J}^Q_{1,*}(\Delta), \mathcal{J}^B_{1,*}(\Delta)$.

$\not\subset$ **(2)** *This second part of the proof compares classes with a quasi-bounded distance against classes with a bounded distance (i.e., B against Q).*

Let $\mathcal{G}_{(2)} = G_1, G_2, ...$ be the dynamic graph with vertex set V where $\forall i \in \mathbb{N}^*$, if $\exists j \in \mathbb{N}$ such that $i = 2^j$, then G_i is fully connected; G_i has no edge otherwise.

Obviously, $\mathcal{G}_{(2)} \in \mathcal{J}^Q_{*,*}(\Delta), \mathcal{J}^Q_{1,*}(\Delta), \mathcal{J}^Q_{*,1}(\Delta)$, since at position 2^j, every vertex can directly reach every other one using one single edge.

But, $\mathcal{G}_{(2)} \notin \mathcal{J}^B_{*,*}(\Delta), \mathcal{J}^B_{*,1}(\Delta), \mathcal{J}^B_{1,*}(\Delta)$ because waiting for the next index, which is a power of 2, is longer and longer and so cannot be bounded.

$\not\subset$ **(3)** *The third part of the proof compares recurrent dynamic graphs against dynamic graphs with bounded or quasi-bounded distances.*

Consider the following edges: $e_i = (v_i, v_{i+1})$, for $i = 1, ..., n-1$ and $e_n = (v_n, v_1)$. Notice that those edges shape a unidirectional ring.

Let $\mathcal{G}_{(3)} = G_1, G_2, ...$ be the dynamic graph with vertex set V, where $\forall i \in \mathbb{N}^*$, G_i contains 0 or one edge; precisely, for every $j \in \mathbb{N}$, G_{2^j} contains the edge $e_{(j \mod n)+1}$; G_i contains no edge otherwise (*i.e.*, when i cannot be written as a power of 2).

Obviously, $\mathcal{G}_{(3)} \in \mathcal{J}_{*,*}, \mathcal{J}_{*,1}, \mathcal{J}_{1,*}$, because every edge of the ring always eventually appears.

However, $\mathcal{G}_{(3)} \notin \mathcal{J}^Q_{*,*}(\Delta), \mathcal{J}^Q_{*,1}(\Delta), \mathcal{J}^Q_{1,*}(\Delta)$. Indeed, the temporal length of journeys linking any two non-consecutive vertices increases over time and so cannot be bounded. Moreover, by transitivity, $\mathcal{G}_{(3)} \notin \mathcal{J}^B_{*,*}(\Delta), \mathcal{J}^B_{*,1}(\Delta), \mathcal{J}^B_{1,*}(\Delta)$.

□

2.2 Computational Model

We consider the computational model defined in [4, 10]. We assume a *distributed system* made of a set V of n *processes*. Each process has a local memory, a local sequential and deterministic algorithm, and message exchange capabilities. We assume that each process p has a unique identifier (ID for short).The identifier of p is denoted by $id(p)$ and taken in an arbitrary domain $IDSET$ totally ordered by $<$. Processes are assumed to communicate by message passing through an interconnected network that evolves over the time. The dynamic topology of the network is then conveniently modeled by a dynamic graph $\mathcal{G} = G_1, G_2, ...$ with vertex set V, *i.e.*, the the set of processes. Processes execute their local algorithms in *synchronous rounds*. For every $i \in \mathbb{N}^*$, the communication network at *Round* i is defined by G_i, *i.e.*, the graph at position i in $\mathcal{G}$. $\forall p \in V, \forall i \in \mathbb{N}^*$, we denote by $\mathcal{IN}(p)^i = \{q \in V : (q, p) \in E(G_i)\}$ the set of p's incoming neighbors at Round i. $\mathcal{IN}(p)^i$ is assumed to be unknown by p, whatever be the value of i.

A *distributed algorithm* $\mathcal{A}$ is a collection of n local algorithms $\mathcal{A}(p)$, one per process $p \in V$ (*n.b.*, different processes may have different codes). At any round, each process has a state. The *state* of each process $p \in V$ in $\mathcal{A}$ is defined by the values of its variables in $\mathcal{A}(p)$. Some variables may be constant in which case their values are predefined. We assume that any algorithm $\mathcal{A}$ designed for a given class of dynamic graph $\mathcal{C}$ satisfies the following property of *well-formedness*: for all $\mathcal{G}, \mathcal{G}' \in \mathcal{C}$ with vertex sets V and V' respectively, if $|V| = |V'|$, then every process $p \in V \cap V'$ executes the same local program $\mathcal{A}(p)$ (in particular, with the same set of local states) whether p is running in $\mathcal{G}$ or $\mathcal{G}'$. This property claims that an algorithm depends only on (1) characteristics that are global to the class in which it runs (*e.g.*, Δ for $\mathcal{J}^B_{1,*}(\Delta)$), (2) process identifiers, and (3) maybe the number of processes.

A *configuration* of $\mathcal{A}$ for V is a vector of n components $\gamma = (s_1, s_2, \ldots, s_n)$, where s_1 to s_n represent the states of the processes in V. Let γ_1 be the initial configuration of $\mathcal{A}$ for V. For any (synchronous) round $i \geq 1$, the system moves from the current configuration γ_i to some configuration γ_{i+1}, where γ_i (resp. γ_{i+1}) is referred to as the configuration *at the beginning (resp. end) of Round* i. Such a move is atomically performed by every process $p \in V$ according to the following three steps, defined in its local algorithm $\mathcal{A}(p)$:

1. p sends a message consisting of all or a part of its local state in γ_i using the primitive `SEND()`,
2. using Primitive `RECEIVE()`, p receives all messages sent by processes in $\mathcal{IN}(p)^i$, and
3. p computes its state in γ_{i+1}.

An *execution* of a distributed algorithm $\mathcal{A}$ in the dynamic graph $\mathcal{G} = G_1, G_2, ...$ is an infinite sequence of configurations $\gamma_1, \gamma_2, \ldots$ of $\mathcal{A}$ for V such that $\forall i > 0$, γ_{i+1} is obtained by executing a synchronous round of $\mathcal{A}$ on γ_i based on the communication network at Round i, *i.e.*, the graph G_i.

2.3 Stabilizing Leader Election in Recurring DG Classes

We have initiated research on self-stabilization in highly dynamic identified message-passing systems in [2]. Notably, we have adapted the definition of self-stabilization to handle recurring DG classes. We recall this definition below. Then, we accommodate the definition of pseudo-stabilization, that was initially defined in the static context, to recurring DG classes. Finally, we define what we mean by self- and pseudo- stabilizing leader election.

Stabilization in Recurring DG Classes. Let $\mathcal{A}$ be a distributed algorithm, SP be a specification (*i.e.*, a predicate over configuration sequences), and $\mathcal{C}$ be a recurring DG class.

Definition 1 (Self-stabilization)**.** *An algorithm $\mathcal{A}$ is* self-stabilizing *for SP on $\mathcal{C}$ if and only if for every set of processes V, there exists a subset of configurations $\mathcal{L}$ of $\mathcal{A}$ for V, called* legitimate configurations*, such that for every $\mathcal{G} \in \mathcal{C}$ with set of processes V,*

1. *for every configuration γ of $\mathcal{A}$ for V, every execution of $\mathcal{A}$ in $\mathcal{G}$ starting from γ contains a legitimate configuration $\gamma' \in \mathcal{L}$ (*Convergence*), and*
2. *for every legitimate configuration $\gamma \in \mathcal{L}$ and every execution e in $\mathcal{G}$ starting from γ, $SP(e)$ holds (*Correctness*).*

The length of the stabilization phase *of an execution e is the length of its maximum prefix containing no legitimate configuration. The* stabilization time *in rounds is the maximum length of a stabilization phase over all possible executions.*

Notice that, as we usually do for static networks, we expect here to provide self-stabilizing algorithms working in wide classes of systems. Consequently, in the previous definition, the predefined set of legitimate configurations only depend on the algorithm, $\mathcal{A}$, and the class of DGs, $\mathcal{C}$; it does not depend on the specific (and *a priori* unknown) dynamic graph of the class on which the algorithm will be actually deployed.

Burns *et al.* define in [7], in the context of static networks, a weak variant of self-stabilization called *pseudo-stabilization*. Intuitively, an algorithm is pseudo-stabilizing if all its executions (starting from arbitrary configurations) have a suffix satisfying the intended specification. In [7], pseudo-stabilization has been shown to be strictly weaker than self-stabilization in terms of expressive power. That is, there are systems and problems for which pseudo-stabilizing solutions do exist but self-stabilizing ones do not. As a matter of facts, they show that the data-link problem can be pseudo-stabilizingly, yet not self-stabilizingly, solved in static message passing systems where the link capacity is unbounded and the process memories are finite. Consequently, in general we cannot define a set of legitimate configurations for a pseudo-stabilizing solution. In other words, this notion is irrelevant in pseudo-stabilization.

Below, we give a definition of pseudo-stabilization in the context of recurring DG classes.

Definition 2 (Pseudo-stabilization)**.** *$\mathcal{A}$ is* pseudo-stabilizing *for SP on $\mathcal{C}$ if and only if for every set of processes V, every $\mathcal{G} \in \mathcal{C}$ with set of processes V, and every configuration γ of $\mathcal{A}$ for V, every execution of $\mathcal{A}$ in $\mathcal{G}$ starting from γ contains a suffix satisfying SP.*

The length of the pseudo-stabilization phase *of an execution $\gamma_1, \gamma_2, \ldots$ is the minimum index i such that $S(\gamma_{i+1}, \gamma_{i+2}, \ldots)$ holds. The* pseudo-stabilization time *in rounds is the maximum length of a pseudo-stabilization phase over all possible executions.*

Remark 2. *By definition, if $\mathcal{A}$ is* self-stabilizing *for SP on $\mathcal{C}$, then $\mathcal{A}$ is also* pseudo-stabilizing *for SP on $\mathcal{C}$ (but the reverse is not necessarily true).*

In particular, the length of the stabilization phase and pseudo-stabilization phase are the same in any self-stabilizing algorithm; and so are their stabilization time and pseudo-stabilization time.

Stabilizing Leader Election. The *leader election* problem consists in distinguishing a single process in the system. In identified networks, the election usually consists in making the processes agree on one of the identifiers held by processes. The identifier of the elected process is then stored at each process p as an output variable, denoted here by $lid(p)$.

In the following, we call *fake ID* any value $v \in IDSET$ (recall that $IDSET$ is the definition domain of the identifiers) such that v is not assigned as a process identifier in the system, *i.e.*, there is no process $p \in V$

such that $id(p) = v$. In the stabilizing context, the output variables lid may be initially corrupted; in particular some of them may be initially assigned to *fake IDs*. Despite such fake IDs, the goal of a self-stabilizing algorithm is to make the system converge to a configuration from which a unique process is forever adopted as leader by all processes, *i.e.*, $\exists p \in V$ such that $\forall q \in V, lid(q) = id(p)$ forever. Similarly, the goal of pseudo-stabilizing algorithm is to ensure the existence of an execution suffix where a unique process is forever adopted as leader by all processes. Hence, the *leader election specification* SP_{LE} can be formulated as follows: a sequence of configurations $\gamma_1, \gamma_2, \ldots$ satisfies SP_{LE} if and only if $\exists p \in V$ such that $\forall i \geq 1$, $\forall q \in V$, the value of $lid(q)$ in Configuration γ_i is $id(p)$.

In the sequel, we say that an algorithm is a *self- (resp. pseudo-) stabilizing leader election algorithm* for the recurring DG class $\mathcal{C}$ if and only if it is self- (resp. pseudo-) stabilizing for SP_{LE} on $\mathcal{C}$.

3 Impossibility Results

In this section, we present several impossibility results. We will first show that there are important recurring DG classes where pseudo- or only self- stabilizing leader election cannot be deterministically solved. Then, we will discuss about time and space complexities of stabilizing leader election algorithms in remaining cases.

All results of this section are based on the notion of *indistinguishable* executions [12]. The principle of this proof scheme is as follows. First, we assume, by the contradiction, that a deterministic leader election algorithm has a given property (related to pseudo- or self- stabilization) in a particular DG class. Then, we consider two sets of processes of same cardinality that differ by only few processes (in the usual case, they differ by only one process). We exhibit two particular executions, one for each set, on two well-chosen DGs of the class. The stabilizing election succeeds in the first one. Moreover, all or a part of processes common to both sets behave exactly the same in both executions: we chose the DG and the initial configuration of the second execution in such way that these processes start with the same local states and receive the sames messages at the same times in both executions; justifying then the term "indistinguishable". Now, the behavior of those processes make the stabilizing election fail in the second execution, leading to a contradiction.

3.1 Impossibility Results for the Design of Stabilizing Leader Election

On the impossibility of solving deterministic self-stabilizing leader election in DG classes with a source. In this first part, we show that no deterministic algorithm can solve the self-stabilizing leader election in classes with a source. To that goal, we first introduce some DG and helpful intermediate results that will be used in the impossibility proofs. Then, we prove the result for Class $\mathcal{J}^B_{1,*}(\Delta)$ and finally extend it to other classes with a source as a direct consequence of Theorem 1 and Figure 2.

Definition 3. *Let $PK(X, y)$ be the directed graph with vertex set X and edge set $\{(p, q) : p \in X \setminus \{y\} \wedge q \in X \wedge p \neq q\}$. $PK(X, y)$ is quasi-complete in the sense that only links outgoing from y are missing.*

Let $\mathcal{PK}(X, y)$ be the dynamic graph $PK(X, y), PK(X, y), \ldots$ with vertex set X such that $|X| \geq 2$ and $y \in X$.

Remark 3. *By definition, every dynamic graph $\mathcal{PK}(X, y)$ belongs to $\mathcal{J}^B_{1,*}(\Delta)$, for every $\Delta \in \mathbb{N}^*$. Indeed, all processes, except y, are timely sources that can always reach all other processes in one round. In contrast, y can never transmit information to any other process.*

The following lemma formally establishes that if there is at least two processes in $\mathcal{PK}(V,p)$, p cannot be finally elected since the other processes have no mean to detect that $id(p)$ is not a fake ID. Indeed, by definition of $\mathcal{PK}(V,p)$, they never receive any message from p.

Lemma 1. *Let V be a set of at least two processes. Let $p \in V$ and $\Delta \in \mathbb{N}^*$. Let $\mathcal{A}$ be any deterministic pseudo-stabilizing leader election algorithm for $\mathcal{J}^B_{1,*}(\Delta)$. Let γ_1 be any configuration of $\mathcal{A}$ for V where $\forall q \in V$, $lid(p) = id(p)$.*

Then, in the execution $e = \gamma_1, \gamma_2, ...$ of $\mathcal{A}$ in $\mathcal{PK}(V,p)$, at least one process eventually modifies the value of its lid variable.

Proof. Assume, by the contradiction, that $\forall i \in \mathbb{N}^*, \forall q \in V$, we have $lid(q) = id(p)$ in γ_i.

Let v be any process such that $v \notin V$ (in particular, $id(v) \neq id(q), \forall q \in V$). Let $V' = V \setminus \{p\} \cup \{v\}$ and γ'_1 be any configuration of $\mathcal{A}$ for V' such that

1. v has any local state in γ'_1, and
2. $\forall q \in V' \setminus \{v\}$, the local state of q is the same in γ'_1 and γ_1.

The only difference between γ_1 and γ'_1 is that p has been replaced by v (with an arbitrary local state). We now consider the execution $e' = \gamma'_1, \gamma'_2, ...$ of $\mathcal{A}$ in $\mathcal{PK}(V',v)$ (recall that $\mathcal{PK}(V',v) \in \mathcal{J}^B_{1,*}(\Delta)$; see Remark 3).

Claim 1.*: *$\forall q \in V' \setminus \{v\}, \forall i \in \mathbb{N}^*$, q has the same local state in γ'_i and γ_i.*

> *Proof of the claim:* By induction on i. The base case $i = 1$ is trivial, by construction of γ'_1. Let $i \geq 1$. By induction hypothesis, $\forall q \in V' \setminus \{v\}$, q has the same local state in γ'_i and γ_i. Let x be any process in $V' \setminus \{v\}$. Since by construction $V' \setminus \{v\} = V \setminus \{p\}$, the in-neighborhood of x is $V' \setminus \{v\}$ at the beginning of Round i both in e and e'. So, x receives the same set of messages and takes the same state in Round i of both e and e', since $\mathcal{A}$ is deterministic. Hence, $\forall q \in V' \setminus \{v\}$, q has the same local state in γ'_{i+1} and γ_{i+1}.

By Claim 1.*, $\forall q \in V' \setminus \{v\}$, $lid(q)$ is constantly equal to $id(p)$ in e'. Now, $p \notin V'$ (in other words, $id(p)$ is a fake ID for V'). So, e' has no suffix satisfying SP_{LE}: $\mathcal{A}$ is not a pseudo-stabilizing leader election algorithm for $\mathcal{J}^B_{1,*}(\Delta)$, a contradiction. □

The next theorem is a direct consequence of Lemma 1. Indeed, consider for the purpose of contradiction a legitimate configuration γ. Then, on process, say ℓ, is elected in that configuration. Now, during the execution starting from γ in $\mathcal{PK}(V,\ell)$, at least one process will change is lid variable, leading to a contradiction.

Theorem 2. *Let $\Delta \in \mathbb{N}^*$. There is no deterministic self-stabilizing leader election algorithm for $\mathcal{J}^B_{1,*}(\Delta)$.*

Proof. We proceed by contradiction. Let $\mathcal{A}$ be a deterministic self-stabilizing leader election algorithm for $\mathcal{J}^B_{1,*}(\Delta)$. Let V be any set of at least two processes. Let γ_1 be any legitimate configuration of $\mathcal{A}$ for $\mathcal{J}^B_{1,*}(\Delta)$. Let $\ell \in V$ be the elected leader in γ_1, *i.e.*, $\forall q \in V$, $lid(q) = id(\ell)$ in γ_1. Consider now the execution $e = \gamma_1, \gamma_2, ...$ of $\mathcal{A}$ in $\mathcal{PK}(V,\ell)$ (recall that $\mathcal{PK}(V,\ell) \in \mathcal{J}^B_{1,*}(\Delta)$; see Remark 3). By Remark 2, Lemma 1 applies: there exist a process $p \in V$ and a round $i \in \mathbb{N}^*$, such that $lid(p) = id(\ell)$ at the beginning of Round i, but $lid(p) \neq id(\ell)$ at the end of Round i. Hence, $SP_{LE}(e)$ does not hold, a contradiction to the correctness property of Definition 1. □

From Theorem 1 and Figure 2, we have the following corollaries.

Corollary 1. *Let $\Delta \in \mathbb{N}^*$. There is no deterministic self-stabilizing leader election algorithm for $\mathcal{J}^Q_{1,*}(\Delta)$.*

Corollary 2. *There is no deterministic self-stabilizing leader election algorithm for $\mathcal{J}_{1,*}$.*

On the impossibility of solving deterministic pseudo-stabilizing leader election in $\mathcal{J}^Q_{1,*}(\Delta)$ and $\mathcal{J}_{1,*}$. We now show that no deterministic algorithm solves the pseudo-stabilizing leader election in Classes $\mathcal{J}^Q_{1,*}(\Delta)$ and $\mathcal{J}_{1,*}$. We first prove the result for Class $\mathcal{J}^Q_{1,*}(\Delta)$ and then extend it to $\mathcal{J}_{1,*}$ as a direct consequence of Theorem 1 and Figure 2.

In the next theorem we proceed by the contradiction. The principle is to consider a set of at least two processes and to construct on the fly an execution together with its associated DG of $\mathcal{J}^Q_{1,*}(\Delta)$ in such a way that the leader election fails. We start in arbitrary configuration and let the pseudo-stabilizing leader election algorithm execute in a complete graph until a leader is elected (it will happen since the algorithm is assumed to be pseudo-stabilizing). Once a leader, say ℓ, is elected, we disturb the network by making the execution continue in the graph $\mathcal{PK}(V, \ell)$ until a process changes its leader (from the previous results, we know that it will happen) and then switch back to a complete graph, and so on. Hence, there is never a permanent leader in the execution. Moreover, since the constructed graph contains infinitely many complete graphs, it trivially belongs to $\mathcal{J}^Q_{1,*}(\Delta)$, for every $\Delta \in \mathbb{N}^*$.

Theorem 3. *Let $\Delta \in \mathbb{N}^*$. There is no deterministic pseudo-stabilizing leader election algorithm for $\mathcal{J}^Q_{1,*}(\Delta)$.*

Proof. We proceed by the contradiction. Let $\mathcal{A}$ be a pseudo-stabilizing leader election algorithm for $\mathcal{J}^Q_{1,*}(\Delta)$. Let V be a set of at least two processes.

Claim 3.*: *Let γ_1 be a configuration of $\mathcal{A}$ for V where there is a unique leader $\ell \in V$. Let $e = \gamma_1, ...$ the execution e of $\mathcal{A}$ starting from γ_1 in $\mathcal{PK}(V, \ell)$. Then, there exists $i \in \mathbb{N}^*$ such that ℓ is not the leader in γ_i.*

Proof of the claim: Since $\mathcal{J}^B_{1,*}(\Delta) \subset \mathcal{J}^Q_{1,*}(\Delta)$, the claim is immediate from Lemma 1.

Let $\mathcal{G} = G_1, G_2, ...$ be a dynamic graph with vertex set V and $e = \gamma_1, \gamma_2, ...$ be the execution of $\mathcal{A}$ in $\mathcal{G}$ starting from γ_1 constructed as follows: γ_1 is a configuration where there is no unique leader and $G_1 = K(V)$; then, $\forall i \in \mathbb{N}^*$,

1. if there is one and the same leader ℓ in both γ_i and the configuration γ_{i+1} computed from γ_i and G_i, then $G_{i+1} = PK(V, \ell)$;
2. otherwise, $G_{i+1} = K(V)$.

By construction and Claim 3.*, e has no suffix satisfying SP_{LE}. Moreover, by Claim 3.* again, $\mathcal{G}$ contains $K(V)$ infinitely many times. Hence, $\mathcal{G} \in \mathcal{J}^Q_{1,*}(\Delta)$. Consequently, $\mathcal{A}$ is not a deterministic pseudo-stabilizing leader election algorithm for $\mathcal{J}^Q_{1,*}(\Delta)$, a contradiction. □

From Theorem 1 and Figure 2, we have the following corollary.

Corollary 3. *There is no deterministic pseudo-stabilizing leader election algorithm for $\mathcal{J}_{1,*}$.*

On the impossibility of solving deterministic pseudo-stabilizing leader election algorithm for classes with a sink. We now introduce some DGs and intermediate helpful results allowing us to show that no pseudo-stabilizing leader election algorithm for $\mathcal{J}^B_{*,1}(\Delta)$ exists. Then, we extend this result to every classes with a sink and self-stabilizing algorithms (as direct consequences of Theorem 1, Figure 2, and Remark 2).

Definition 4. *Let X be any vertex set such that $|X| \geq 2$; see, for example, the star graph T in Figure 4. For every $y \in X$. Let $S(X, y)$ be the directed graph with vertex set X and edge set $\{(p, y) : p \in X \setminus \{y\}\}$. Let $\mathcal{S}(X, y)$ be the dynamic graph with vertex set X $S(X, y), S(X, y), ...$*

Remark 4. *By definition, every dynamic graph $\mathcal{S}(X, y)$ belongs to $\mathcal{J}^B_{*,1}(\Delta)$, for every $\Delta \in \mathbb{N}^*$. Indeed, y is a timely sink that can always be reached from all other processes in one round. However, y can never transmit information to any other process.*

To show the next theorem, the overall idea is quite simple: if we consider an set V of at least three processes, one process p of V, and the DG $\mathcal{S}(V, p)$, then every leaf of the star (at least two) will eventually choose itself as leader since it has no mean to guess any identifier of some other process. Indeed, using $\mathcal{S}(V, p)$, except p, no process receives any message.

Theorem 4. *Let $\Delta \in \mathbb{N}^*$. There is no deterministic pseudo-stabilizing leader election algorithm for $\mathcal{J}^B_{*,1}(\Delta)$.*

Proof. We proceed by the contradiction. Let V be a set of at least three processes and $p \in V$. Let $\mathcal{A}$ be a deterministic pseudo-stabilizing leader election algorithm for $\mathcal{J}^B_{*,1}(\Delta)$. Let $e = \gamma_1, \gamma_2, ...$ be any execution of $\mathcal{A}$ in $\mathcal{S}(V, p)$.

To obtain the contradiction, we show below that for every process $q \in V \setminus \{p\}$, eventually $q.lid = id(q)$ forever in e. Since $|V \setminus \{p\}| \geq 2$, we immediately obtain the contradiction.

Let $q \in V \setminus \{p\}$. Since $\mathcal{A}$ is a deterministic pseudo-stabilizing leader election algorithm for $\mathcal{J}^B_{*,1}(\Delta)$, there exist $i \in \mathbb{N}^*$ and $v \in V$, such that $\forall j \geq i$, $lid(q) = id(v)$ in γ_j. Assume, by the contradiction, that $v \neq q$. Let w be any process such that $w \notin V$ (in particular, $id(w) \neq id(x), \forall x \in V$). Let $V' = V \setminus \{v\} \cup \{w\}$. Let γ'_1 be any configuration of $\mathcal{A}$ for V' such that

1. w has any local state in γ'_1, and
2. $\forall x \in V' \setminus \{w\}$, the local state of x is the same in γ'_1 and γ_i.

The only difference between γ_i and γ'_1 is that v has been replaced by w (with an arbitrary local state). We now consider the execution $e' = \gamma'_1, \gamma'_2, ...$ of $\mathcal{A}$ in $\mathcal{S}(V', w)$ (recall that $\mathcal{S}(V', w) \in \mathcal{J}^B_{*,1}(\Delta)$; see Remark 4).

Claim 4.*: *$\forall j \in \mathbb{N}^*$, the local state of q is the same in γ'_j and γ_{i+j-1}.*

> *Proof of the claim:* By induction on j. The base case $j = 1$ is trivial, by construction of γ'_1. Let $j \geq 1$. By induction hypothesis, $\gamma'_j(q) = \gamma_{i+j-1}(q)$. By construction, the in-neighborhood of q is empty at the beginning of both Round $i + j - 1$ in e and Round j in e'. So, q receives no message and takes the same state in both Round $i + j - 1$ of e and Round j of e', since $\mathcal{A}$ is deterministic. Hence, the local state of q is the same in γ'_{j+1} and γ_{i+j}.

By Claim 4.*, $lid(q) = id(v)$ forever in e'. Now, $id(v) \notin V'$ (in other words, $id(v)$ is a fake ID for V'). So, e' has no suffix satisfying SP_{LE}: $\mathcal{A}$ is not a deterministic pseudo-stabilizing leader election algorithm for $\mathcal{J}^B_{*,1}(\Delta)$, a contradiction. □

From Theorem 1 and Figure 2, we have the following corollaries.

Corollary 4. *Let $\Delta \in \mathbb{N}^*$. There is no deterministic pseudo-stabilizing leader election algorithm for $\mathcal{J}^Q_{*,1}(\Delta)$.*

Corollary 5. *There is no deterministic pseudo-stabilizing leader election algorithm for $\mathcal{J}_{*,1}$.*

From Remark 2, we have the following corollaries.

Corollary 6. *Let $\Delta \in \mathbb{N}^*$. There is no deterministic self-stabilizing leader election algorithm for $\mathcal{J}^B_{*,1}(\Delta)$.*

Corollary 7. *Let $\Delta \in \mathbb{N}^*$. There is no deterministic self-stabilizing leader election algorithm for $\mathcal{J}^Q_{*,1}(\Delta)$.*

Corollary 8. *There is no deterministic self-stabilizing leader election algorithm for $\mathcal{J}_{*,1}$.*

3.2 Time Complexity

On time complexity of deterministic pseudo-stabilizing leader election in $\mathcal{J}^B_{1,*}(\Delta)$. Below, we show that the length of the pseudo-stabilization phase of any deterministic pseudo-stabilizing leader election algorithm for $\mathcal{J}^B_{1,*}(\Delta)$ cannot be bounded.

Definition 5. *Let $K(X) = (X, E)$ the directed complete graph, i.e., $E = \{(p, q) : p, q \in X \wedge p \neq q\}$. Let $\mathcal{K}(X)$ be the dynamic graph with vertex set X $K(X), K(X), ...$*

For the next theorem, we again proceed by the contradiction. We consider an execution which begins in an arbitrary configuration while the network is complete. The network remains complete until the end of Round $f(n, \Delta)$. By hypothesis, a leader, say ℓ, is elected at that time. Then, we disturb the system by making the execution continue in the graph $\mathcal{PK}(V, \ell)$. Now, in one hand, we know (from previous results) that a process will eventually changes its leader, falsifying the specification after Round $f(n, \Delta)$; and in the other hand any DG starting with a finite number of complete graphs followed by $\mathcal{PK}(V, \ell)$ belong to $\mathcal{J}^B_{1,*}(\Delta)$. Hence, we obtain the contradiction.

Theorem 5. *Let $\Delta \in \mathbb{N}^*$ and $n \geq 2$. Let $\mathcal{A}$ be a deterministic pseudo-stabilizing leader election algorithm for $\mathcal{J}^B_{1,*}(\Delta)$. There exists no function $f : \mathbb{N}^* \times \mathbb{N}^* \to \mathbb{N}$ such that $\forall \mathcal{G} \in \mathcal{J}^B_{1,*}(\Delta)$ with a vertex set of n processes, the length of the pseudo-stabilization phase of every execution of $\mathcal{A}$ in $\mathcal{G}$ is less than or equal to $f(n, \Delta)$.*

Proof. Assume by the contradiction that such a function f exists. By definition, $\mathcal{K}(V) \in \mathcal{J}^B_{1,*}(\Delta)$. Let $\gamma_1, \gamma_2, ...$ be any execution of $\mathcal{A}$ in $\mathcal{K}(V)$. Let $i = f(n, \Delta) + 1$. By hypothesis, there exists $\ell \in V$ such that $\forall p \in V$, $\gamma_i(p).lid = id(\ell)$.

Let e be the execution of $\mathcal{A}$ in $\mathcal{PK}(V, \ell)$ starting from γ_i. By Lemma 1, at least one process eventually modifies the value of its lid variable in e (this in particular means that $SP_{LE}(e)$ does not hold).

Now, $\mathcal{G} = (K(V))^{i-1}, \mathcal{PK}(V, \ell)$, *i.e.*, the sequence made of $i - 1$ graphs $K(V)$ followed by the sequence of graphs in $\mathcal{PK}(V, \ell)$, is a dynamic graph with vertex set V that belongs to $\mathcal{J}^B_{1,*}(\Delta)$. Moreover, $e' = \gamma_1, \gamma_2, ..., \gamma_{i-1}, e$, *i.e.*, the sequence made of $\gamma_1, \gamma_2, ..., \gamma_{i-1}$ followed the sequence of configurations in e, is an execution of $\mathcal{A}$ in $\mathcal{G}$ where $SP_{LE}(e)$ does not hold. Hence, the length of the pseudo-stabilization phase in e' is greater than $f(n, \Delta)$, a contradiction. □

On time complexity of deterministic pseudo-stabilizing leader election in Class $\mathcal{J}^Q_{*,*}(\Delta)$. Below, we show that the length of the pseudo-stabilization phase of a deterministic pseudo-stabilizing leader election algorithm for classes $\mathcal{J}^Q_{*,*}(\Delta)$ cannot be bounded. Since $\mathcal{J}^Q_{*,*}(\Delta) \subset \mathcal{J}_{*,*}$, we can generalize this result to $\mathcal{J}_{*,*}$.

The idea behind the next impossibility result is that if we consider any DG $\mathcal{G}$ of $\mathcal{J}^Q_{*,*}(\Delta)$, then we can construct another DG starting with a finite number of independent sets followed by $\mathcal{G}$. This latter also belongs to $\mathcal{J}^Q_{*,*}(\Delta)$. Now, the length of the finite prefix made of independent sets is unbounded and during this prefix no process receive any message, by definition. Hence, they cannot coordinate together to elect a leader.

Theorem 6. *Let $\Delta \in \mathbb{N}^*$ and $n \geq 2$. Let $\mathcal{A}$ be a deterministic pseudo-stabilizing leader election algorithm for $\mathcal{J}^Q_{*,*}(\Delta)$. There exists no function $f : \mathbb{N}^* \times \mathbb{N}^* \to \mathbb{N}$ such that $\forall \mathcal{G} \in \mathcal{J}^Q_{*,*}(\Delta)$ with a vertex set of n processes, the length of the pseudo-stabilization phase of every execution of $\mathcal{A}$ in $\mathcal{G}$ is less than or equal to $f(n, \Delta)$.*

Proof. Assume by the contradiction that such a function f exists. Let $\mathcal{G}$ be any dynamic graph of $\mathcal{J}^Q_{*,*}(\Delta)$ with vertex set V such that $|V| = n$ and whose prefix of length $f(n, \Delta)$ is only constituted of independent sets (no edge). Such a graph exists in $\mathcal{J}^Q_{*,*}(\Delta)$, by definition.

Let $e = \gamma_1, \gamma_2, ...$ be any execution of $\mathcal{A}$ in $\mathcal{G}$. By hypothesis, there exists a unique leader since $\gamma_{f(n,\Delta)+1}$. Let $\ell \in V$ be that leader process.

Let v be any process such that $v \notin V$ (in particular, $id(v) \neq id(p), \forall p \in V$). Let $V' = V \setminus \{\ell\} \cup \{v\}$. Let γ'_1 be any configuration of $\mathcal{A}$ for V' such that

1. v has any local state in γ'_1, and
2. $\forall p \in V' \setminus \{v\}$, the local state of p is the same in γ'_1 and $\gamma_1(p)$.

The only difference between γ_1 and γ'_1 is that ℓ has been replaced by v (with an arbitrary local state). We now consider the execution $e' = \gamma'_1, \gamma'_2, ...$ of $\mathcal{A}$ in $\mathcal{G}$.

Claim 6.*: *$\forall p \in V' \setminus \{v\}$, $\forall i \in \{1, ..., f(n, \Delta) + 1\}$, the local state of p is the same in γ'_i and γ_i.*

> *Proof of the claim:* By induction on i. The base case $i = 1$ is trivial, by construction of γ'_1. Let $i \in \{1, ..., f(n, \Delta)\}$. By induction hypothesis, $\forall p \in V' \setminus \{v\}$, the local state of p is the same in γ'_i and γ_i. Let q be any process in $V' \setminus \{v\}$. By construction, the in-neighborhood of q is empty at the beginning of Round i both in e and e'. So, q receives no message and takes the same state in Round i of e and e', since $\mathcal{A}$ is deterministic. Hence, the local state of q is the same in γ'_{i+1} and γ_{i+1}.

By Claim 6.*, $\forall q \in V' \setminus \{v\}$, $lid(q) = id(\ell)$ at the beginning of Round $f(n, \Delta) + 1$ in e'. Now, $id(\ell) \notin V'$ (in other words, $id(\ell)$ is a fake ID for V'). So, the suffix of e' starting from $\gamma'_{f(n,\Delta)+1}$ does not satisfy SP_{LE}, *i.e.*, the length of the pseudo-stabilizing phase of e' is greater than $f(n, \Delta)$, a contradiction. □

From Remark 2, we have the following corollary.

Corollary 9. *Let $\Delta \in \mathbb{N}^*$ and $n \geq 2$. Let $\mathcal{A}$ be a deterministic self-stabilizing leader election algorithm for $\mathcal{J}^Q_{*,*}(\Delta)$. There exists no function $f : \mathbb{N}^* \times \mathbb{N}^* \to \mathbb{N}$ such that $\forall \mathcal{G} \in \mathcal{J}^Q_{*,*}(\Delta)$ with a vertex set of n processes, the length of the stabilization phase of every execution of $\mathcal{A}$ in $\mathcal{G}$ is less than or equal to $f(n, \Delta)$.*

Corollary 10. *Let $n \geq 2$. Let $\mathcal{A}$ be a deterministic pseudo-stabilizing leader election algorithm for $\mathcal{J}_{*,*}$. There exists no function $g : \mathbb{N}^* \to \mathbb{N}$ such that $\forall \mathcal{G} \in \mathcal{J}_{*,*}$ with a vertex set of n processes, the length of the pseudo-stabilization phase of every execution of $\mathcal{A}$ in $\mathcal{G}$ is less than or equal to $g(n)$.*

Proof. Assume, by the contradiction, that the corollary is false. Then, there exists a function $g : \mathbb{N}^* \to \mathbb{N}$ such that $\forall \mathcal{G} \in \mathcal{J}_{*,*}$ with a vertex set of n processes, the length of the pseudo-stabilization phase of every execution of $\mathcal{A}$ in $\mathcal{G}$ is less than or equal to $g(n)$. For every $\Delta \in \mathbb{N}^*$, since $\mathcal{J}^Q_{*,*}(\Delta) \subset \mathcal{J}_{*,*}$, this claim also holds for $\mathcal{J}^Q_{*,*}(\Delta)$. Let $f(n, \Delta) = g(n)$. We have then $\forall \mathcal{G} \in \mathcal{J}^Q_{*,*}(\Delta)$ with a vertex set of n processes, the length of the pseudo-stabilization phase of every execution of $\mathcal{A}$ in $\mathcal{G}$ is less than or equal to $f(n, \Delta)$, which contradicts Corollary 9. □

From Remark 2, we have the following corollary.

Corollary 11. *Let $n \geq 2$. Let $\mathcal{A}$ be a deterministic self-stabilizing leader election algorithm for $\mathcal{J}_{*,*}$. There exists no function $f : \mathbb{N}^* \to \mathbb{N}$ such that $\forall \mathcal{G} \in \mathcal{J}_{*,*}$ with a vertex set of n processes, the length of the stabilization phase of every execution of $\mathcal{A}$ in $\mathcal{G}$ is less than or equal to $f(n)$.*

3.3 Memory Requirement

The following theorem shows that, if $\mathcal{A}$ a pseudo-stabilizing leader election algorithm for $\mathcal{J}^B_{1,*}(\Delta)$, then to be finite, the number of configurations of $\mathcal{A}$ for a given set of process V should in particular depend on Δ. As a consequence, the process local memories may be finite only if their size in particular depends on Δ too.

The proof of the next theorem is done by contradiction and follows almost the same scheme as the proof of Theorem 3. The only difference is that we use the assumption which claims that the number of configurations is bounded by $f(n)$ to show that the DG constructed on the fly has never more than $M_0 - 1$ consecutive graphs in $\mathcal{G}$ different from $K(V)$, where $M_0 = \max\{\Delta_0, f(n) + 1\}$. Hence, we obtain a DG that belongs to $\mathcal{J}^B_{1,*}(M_0)$.

Theorem 7. *Let $\mathcal{A}$ be a deterministic distributed algorithm. Let $f : \mathbb{N}^* \to \mathbb{N}$ be any function (that will be used to measure the number of configurations). Let V a set of vertices of size $n \geq 2$.*

For every $\Delta_0 \in \mathbb{N}^*$, there exists $\Delta \geq \Delta_0$ if $\mathcal{A}$ is a pseudo-stabilizing leader election algorithm for $\mathcal{J}^B_{1,*}(\Delta)$, then $\mathcal{A}$ has more than $f(n)$ configurations when executing in a DG of $\mathcal{J}^B_{1,*}(\Delta)$ with V as vertex set.

Proof. We proceed by the contradiction. Let $\Delta_0 \in \mathbb{N}^*$, $f : \mathbb{N}^* \to \mathbb{N}$, V a set of $n \geq 2$ processes, and $\mathcal{A}$ is a deterministic distributed algorithm.

Assume that, $\forall \Delta \geq \Delta_0$, $\mathcal{A}$ is a pseudo-stabilizing leader election algorithm for $\mathcal{J}^B_{1,*}(\Delta)$ and $\mathcal{A}$ has at most $f(n)$ configurations when executing in a DG of $\mathcal{J}^B_{1,*}(\Delta)$ with V as vertex set.

Let $M_0 = \max\{\Delta_0, f(n) + 1\}$.

Claim 7.*: *Let γ_1 be any configuration of $\mathcal{A}$ for V where there is a unique leader $\ell \in V$. Let $\gamma_1, ..., \gamma_{M_0}$ be the prefix of length M_0 of the execution e of $\mathcal{A}$ starting from γ_1 in $\mathcal{PK}(V, \ell)$.*

Then, there exist $i \in \{2, ..., M_0\}$ and $p \in V$ such that $\gamma_i(p).lid \neq id(\ell)$.

Proof of the claim: By definition of M_0, there exist x, y such that $1 \leq x < y \leq M_0$ such that $\gamma_x = \gamma_y$. So, $e = \gamma_1, \gamma_{x-1}, (\gamma_x, ..., \gamma_{y-1})^\omega$ since $\mathcal{A}$ is deterministic and the topology is fixed in $\mathcal{PK}(V, \ell)$. Hence, if the claim is wrong, then ℓ is the unique leader forever in e, contradicting Lemma 1.

Let $\mathcal{G} = G_1, G_2, ...$ be the dynamic graph with vertex set V and $e = \gamma_1, \gamma_2, ...$ be the execution of $\mathcal{A}$ in $\mathcal{G}$ starting from γ_1 constructed as follows. γ_1 is a configuration where there is no unique leader and $G_1 = K(V)$. Then, $\forall i \in \mathbb{N}^*$,

1. if there is one and the same leader ℓ in both γ_i and the configuration γ_{i+1} computed from γ_i and G_i, then $G_{i+1} = PK(V, \ell)$;
2. otherwise, $G_{i+1} = K(V)$.

By construction and Claim 7.*, e has no suffix satisfying SP_{LE}. Moreover, by Claim 7.* again, there is never more than $M_0 - 1$ consecutive graphs in $\mathcal{G}$ different from $K(V)$. Hence, $\mathcal{G} \in \mathcal{J}^B_{1,*}(M_0)$. Consequently, $\mathcal{A}$ is not a deterministic pseudo-stabilizing leader election algorithm for $\mathcal{J}^B_{1,*}(M_0)$, a contradiction since $M_0 \geq \Delta_0$. □

4 Pseudo-stabilizing Leader Election in $\mathcal{J}^B_{1,*}(\Delta)$

We denote by $\mathcal{LE}$ our pseudo-stabilizing leader election algorithm for $\mathcal{J}^B_{1,*}(\Delta)$. Its code is given in Algorithms 1 and 2.

Algorithm 1: Algorithm $\mathcal{LE}$, macros and variables for each process p

Inputs:

$\Delta \in \mathbb{N}^*$	:	upper bound on the temporal distance from a timely source to any other process
$id(p) \in IDSET$	:	ID of p

Variable Type:

$MapType$	:	map containing tuples $\langle id, susp, ttl\rangle \in IDSET \times \mathbb{N} \times \{0, \ldots, \Delta\}$

Local Variables:

$lid(p) \in IDSET$	:	ID of the leader
$Lstable(p) \in MapType$	:	locally stable processes for p
$Gstable(p) \in MapType$	:	globally stable processes collected by p
$msgs(p)$	:	set containing records $\langle id, LSPs, ttl\rangle \in IDSET \times MapType \times \{0, \ldots, \Delta\}$

Macros:

$minSusp(p) \equiv \min\{Gstable(p)[q].susp : q \in Gstable(p)\}$

Overview. The goal of Algorithm $\mathcal{LE}$ is to elect what we call a *stable* process. Intuitively, a process p is stable if eventually all other processes receive (maybe indirectly) information about p at least every Δ rounds.

Basically, in the algorithm all processes initiate a broadcast at each round. So, since there exist timely sources in the class we consider ($\mathcal{J}^B_{1,*}(\Delta)$), there are stable processes in the system, namely those timely sources themselves.

To evaluate the stability of a process p, we use a so-called *counter of suspicion*. The suspicion counter of p is maintained by p itself and is incremented each time p learns that some other process does not consider it as stable. In the following, we call *suspicion value* of p the current value of the suspicion counter of p.

Due to initial inconsistencies, a suspicion counter may be reset once during the first round. Yet, after the first round, each counter is monotonically nondecreasing. Moreover, the counter of each timely source is eventually constant since it is stable, by definition. Finally, each process aims at electing a process with the minimum suspicion value (we use identifiers to break ties) since such a process will be a stable process.

To implement this principle, each process p maintains four variables: $lid(p)$, $msgs(p)$, $Lstable(p)$, and $Gstable(p)$. Variable $lid(p)$ is the output of the algorithm, it eventually contains the identifier of the elected process. The three other variables are more complex data structures. Basically, Process p stores in $msgs(p)$ the messages that it will send at the beginning of the next round; see Line 2. $Lstable(p)$ and $Gstable(p)$ are used to detect stable processes. They are both of type $MapType$.

The type $MapType$. The main objective of Algorithm $\mathcal{LE}$ is to identify a set of stable processes. To achieve this, processes exchange information about each process they heard about. This information is stored in data structures of type $MapType$, *i.e.*, *maps* containing tuples of three elements:

- an identifier $id \in IDSET$,
- a natural number $susp$, representing the (maybe outdated) suspicion value of the process identified by id, and

Algorithm 2: Algorithm $\mathcal{LE}$, code for each process p

1: **Repeat Forever**
2: SEND($\{msg \in msgs(p) : msg.ttl > 0 \wedge msg.id \in msg.LSPs\}$)
3: mailbox := RECEIVE()

// Reset
4: **if** $\langle id(p), -, \Delta\rangle \notin Lstable(p)$ **then** insert $\langle id(p), 0, \Delta\rangle$ in $Lstable(p)$
5: **if** $\langle id(p), -, \Delta\rangle \notin Gstable(p) \vee Gstable(p)[id(p)].susp \neq Lstable(p)[id(p)].susp$ **then**
6: insert $Lstable(p)[id(p)]$ in $Gstable(p)$

// Update of the timers
7: **forall** $id \in Lstable(p)$ **do**
8: **if** $id \neq id(p) \wedge Lstable(p)[id].ttl \neq 0$ **then** $Lstable(p)[id].ttl - -$
9: **forall** $id \in Gstable(p)$ **do**
10: **if** $id \neq id(p) \wedge Gstable(p)[id].ttl \neq 0$ **then** $Gstable(p)[id].ttl - -$

// Handling the messages
11: **forall** *tuples* $msg = \langle id, LSPs, ttl\rangle$ *in a message of mailbox* **do**
12: **if** $id \neq id(p)$ **then**
13: **if** $\langle id, -, ttl\rangle \notin msgs(p)$ **then** insert $msg = \langle id, LSPs, ttl\rangle$ in $msgs(p)$
14: **if** $\langle id, -, -\rangle \notin Lstable(p) \vee (\langle id, -, ttl'\rangle \in Lstable(p)$ *with* $ttl' < ttl)$ **then**
15: insert $\langle id, LSPs[id].susp, ttl\rangle$ in $Lstable(p)$;
16: **forall** $\langle id', susp, -\rangle \in LSPs : id' \neq id(p)$ **do**
17: insert $\langle id', susp, \Delta\rangle$ in $Gstable(p)$
18: **if** $id(p) \notin LSPs$ **then** $Lstable(p)[id(p)].susp + +$; $Gstable(p)[id(p)].susp + +$;

// Expired timers
19: **forall** $id \in Lstable(p)$ **do**
20: **if** $Lstable(p)[id].ttl = 0$ **then** remove the tuple associated to id in $Lstable(p)$
21: **forall** $id \in Gstable(p)$ **do**
22: **if** $Gstable(p)[id].ttl = 0$ **then** remove the tuple associated to id in $Gstable(p)$

// Update messages
23: **forall** $msg \in msgs(p)$ **do**
24: **if** $msg.ttl = 0 \vee msg.id \notin msg.LSPs$ **then** remove msg from $msgs(p)$
25: **else** $msg.ttl - -$
26: insert $\langle id(p), Lstable(p), \Delta\rangle$ in $msgs(p)$

// Choice of the leader
27: $lid(p) := \min\{id \in Gstable(p), Gstable(p)[id].susp = minSusp(p)\}$

- a timer value $ttl \in \{0, \ldots, \Delta\}$ (ttl stands for "time to live").

A map of type $MapType$ is indexed by the first element id of each tuple it contains. So, there exists at most one tuple $\langle id, -, - \rangle$ in any variable of type $MapType$, where "$-$" is a wildcard, *i.e.*, "$-$" is any possible value. (For example, in Line 13, $\langle id, -, ttl \rangle$ represents any tuple with id and ttl as first and third fields, whatever the value of the second field is.) We simply denote by $M[id]$ the tuple of index id in the map M. Moreover, for sake of simplicity, we write $id \in M$ to mean that the tuple $M[id]$ exists, *i.e.*, M contains a tuple of index id: $\langle id, -, - \rangle \in M$; see Line 7 for example.

Of course, the insertion function keeps the uniqueness of each index. Assume, for example, that a tuple $\langle id, s, t \rangle$ is inserted into the map M. If $M[id]$ already exists right before the insertion, then $M[id]$ is just refreshed with the new values s and t. Similarly, right after the removal of $M[id]$ from M, M contains no tuple $\langle id, -, - \rangle$.

Locally Stable Processes. Every process p stores into $Lstable(p)$ information —precisely, an identifier, a suspicion value, and a timer value— for each process that is *locally stable* at p, *i.e.*, for each process q that p currently considers as a good candidate for the leader election because it receives information from q at most Δ rounds ago. Variable $Lstable(p)$ is of type $MapType$.

Globally Stable Processes. For every process p, $Gstable(p)$ is also of type $MapType$. The map $Gstable(p)$ contains information —*i.e.*, tuples made of an identifier, a suspicion value, and a timer value— related to *globally stable processes*, meaning processes that are *locally stable* at any process (and so not only at p). Eventually all stable processes (in particular, the timely sources) should forever belong to $Gstable(p)$, for every process p. Moreover, at each round every process selects the value of its output $lid(p)$ among identifiers stored into $Gstable(p)$. Precisely, p chooses an ID in $Gstable(p)$ associated with the minimum suspicion value (we use IDs to break ties); see Line 27 and macro $minSusp(p)$.

Messages. Processes exchange information stored into what we call *records*. Actually, a record R is a tuple $\langle id, LSPs, ttl \rangle$, where $R.id \in IDSET$, $R.LSPs$ is a map of type $MapType$, and $R.ttl \in \{0, \ldots, \Delta\}$ is a timer value. The variable $msgs(p)$ is used to store the records that will be sent by Process p at the beginning of the next round. Actually, $msgs(p)$ is a set (and so not a map) meaning in particular that it may contain several records tagged with the same identifier.

At each round, each process p initiates the broadcast of the record $\langle id(p), Lstable(p), \Delta \rangle$ by inserting it into $msgs(p)$; see Line 26. Notice that in each record $\langle id(p), Lstable(p), \Delta \rangle$ initiated by p, we have $id(p) \in Lstable(p)$ (if necessary, $id(p)$ is added in $Lstable(p)$ at Line 4, and then never removed). Then, the timer in the record is used to relay the record during Δ rounds. The goal of this broadcast is to inform other processes about the current list of processes that are locally stable at p with their associated suspicion value known by p (*n.b.*, those suspicion values may be outdated).

At each round, received records that should be relayed are collected into $msgs(p)$ and their associated timer is decremented; see Lines 13 and 25. Only well-formed records R (*i.e.*, satisfying $R.id \in R.LSPs$) with non-zero timer will be sent during the next round; see Lines 2 and 24. The condition $R.id \in R.LSPs$ allows to eliminate some spurious messages. The timer mechanism, in particular, ensures that records tagged with fake IDs are eventually no more relayed since such records are never initiated.

Management of $Lstable(p)$. Recall that $Lstable(p)$ contains the identifiers and known suspicion values of processes that are currently considered by p to be locally stable. First, p always considers itself as

a locally stable process. Thank to Line 4, a tuple $\langle id(p), -, \Delta \rangle$ forever belongs to $Lstable(p)$ after the first round. Then, p supplies $Lstable(p)$ using records it receives from other processes. Upon receiving a record $\langle id, LSPs, ttl \rangle$, p inserts $\langle id, LSPs[id].susp, ttl \rangle$ into $Lstable(p)$ if either id does not appear in $Lstable(p)$, or ttl is greater than the current timer associated to id in $Lstable(p)$; see Lines 14-15. Notice that $LSPs[id].susp$ is meant to be the suspicion value of the initiator at the time it started to broadcast the record.

The timer values of each record in $Lstable(p)$ is decremented at each round; see Lines 7-8. Every record whose timer value reaches 0 is removed from $Lstable(p)$; see Lines 19-20.

Hence, an identifier id different from $id(p)$ remains in $Lstable(p)$ only if p receives at least every Δ rounds a record $\langle id, -, - \rangle$ initiated by the process whose ID is id. Notice that, since records tagged with fake IDs are eventually never more received, they also eventually vanish from all $Lstable$ maps.

Management of $Gstable(p)$. Similarly to $Lstable(p)$, p always considers itself as a globally stable process. Thank to Line 5, a tuple $\langle id(p), -, \Delta \rangle$ forever belongs to $Gstable(p)$ after the first round. Then, p supplies $Gstable(p)$ with the map $R.LSPs$ attached to records R it receives from other processes. At each round, each identifier different from $id(p)$ present in a map of some received record is meant to be the identifier of a locally stable process at some other process. So, it is meant to identify a globally stable process. Such an identifier and its associated suspicion value is then stored into $Gstable(p)$ with a timer initialized to Δ; see Lines 17-18.

Similarly to $Lstable(p)$, the timer values of each record in $Gstable(p)$ is decremented at each round; see Lines 9-10. Every record whose timer value reaches 0 is removed from $Gstable(p)$; see Lines 21-22.

Hence, an identifier id different from $id(p)$ remains in $Gstable(p)$ only if p receives at least every Δ rounds a record from some other process notifying that a process identified by id is one of its locally stable processes. Remark that, since fake IDs eventually vanish from all $Lstable$ maps, fake IDs also eventually vanish from all $Gstable$ maps.

Suspicion value. The suspicion value of each process p is stored in both $Lstable(p)[id(p)].susp$ and $Gstable(p)[id(p)].susp$ (this choice allows to simplify a bit the algorithm design). Hence, the values $Lstable(p)[id(p)].susp$ and $Gstable(p)[id(p)].susp$ are kept equal; see Lines 5, 6, and 18. Moreover, after the first round, the suspicion value of p is monotonically nondecreasing.

Actually, p increments its suspicion value each time it realizes that it is not part of the locally stable processes of some other process q, *i.e.*, each time it receives a record R initiated by q such that $id(p) \notin R.LSPs$; see Line 18.

Using the broadcasts it initiates at each round, p aims at continuously propagating its latest suspicion value to all other processes.

Fake IDs. We have seen that records tagged with fake IDs are eventually no more sent. Furthermore, by a domino effect, fake IDs are eventually removed from all $Lstable$ and $Gstable$ maps. Actually, we were able to prove that no fake ID exists in the system after at most 4Δ rounds; see Lemma 8 for details.

Pseudo-stabilization. We have seen that every process p collects in $Lstable(p)$ the list of processes that it currently considers to be locally stable, including itself. In particular, an identifier id different from $id(p)$ remains in $Lstable(p)$ only if p received at least every Δ rounds a record $\langle id, -, - \rangle$ initiated by the process identified by id. Hence, if the temporal distance between two processes p and q is always at most Δ, then eventually $id(q) \in Lstable(p)$ forever. A direct consequence of this property is that the identifier of every

timely source eventually constantly belongs to the $Lstable$ map of every process. Actually, we can show that the identifier of each timely source forever belongs to any $Lstable$ map after at most $2\Delta + 1$ rounds; see Lemmas 10 and 11.

$Gstable$ maps are managed similarly to $Lstable$ maps using timers. Each process p forever appears in its own $Gstable$ map after at most one round. Then, p tries to fill it with information (including the suspicion value) about the local stable processes at all other processes. Notice that eventually each received suspicion value is either up-to-date or an old one.

Again, since p receives records initiated by each timely source at least every Δ rounds, almost up to date information about the processes locally stable at each timely source are eventually always present in $Gstable(p)$. Notably, identifiers that are eventually forever present in $Lstable$ maps of all timely sources, in particular their own identifiers, are eventually forever present in each $Gstable$ map (actually, after at most $3\Delta + 2$ rounds by Lemma 12).

Recall that after the first round, the suspicion value of every process is monotonically nondecreasing. Moreover, the suspicion value of a process p is incremented only when p receives a record with a list of locally stable processes that does not include its own identifier. Hence, from the previous discussion, we know that each timely source only increments its suspicion value a finite number of times. Moreover, eventually all processes store forever in their $Gstable$ map both the identifiers and the final suspicion values of all timely sources.

Consider now a process q whose identifier is infinitely often absent from $Gstable(p)$, for some process p. In this case, there are infinitely many period of Δ rounds where p does not receive any record including $id(q)$ into its list of locally stable processes. This means, in particular, that every timely source broadcasts infinitely many records that do not include $id(q)$ into the attached list of locally stable processes. Since initiated by timely sources, such records reach q infinitely often, and consequently the suspicion counter of q grows infinitely often. Moreover, from the previous discussion, if q is also inserted infinitely often in some $Gstable$ map, then its suspicion value in the map grows infinitely often.

Overall, the $Gstable$ map of every process eventually forever contains the identifiers and final suspicion values of all processes (including timely sources) whose suspicion value is eventually constant. Moreover, every other process which regularly appears in the map do so with an associated suspicion value that increases again and again. Now, at the end of each round, each process p selects an identifier in its $Gstable$ map with the minimum attached suspicion value (the order on identifiers may be used to break ties). Hence, all processes eventually forever elect the same process.

Speculation. The pseudo-stabilization time of Algorithm $\mathcal{LE}$ cannot be bounded in $\mathcal{J}^B_{1,*}(\Delta)$ by Theorem 5. However, Algorithm $\mathcal{LE}$ is speculative in the sense that its pseudo-stabilization time in $\mathcal{J}^B_{*,*}(\Delta) \subset \mathcal{J}^B_{1,*}(\Delta)$ is bounded. This result is due to the fact that all processes are timely sources in $\mathcal{J}^B_{*,*}(\Delta)$.

In more detail, the suspicion value of each process becomes constant after executing at most $2\Delta + 1$ rounds in a DG of $\mathcal{J}^B_{*,*}(\Delta)$; see Lemma 12. Then, we can show that during the next 4Δ rounds, all fake IDs vanish (Lemma 8) and all nodes are inserted forever into all $Gstable$ map (Lemma 10) with their final suspicion values (Lemma 16 and Remark 5.(b)). Hence, at the beginning of the next round (*i.e.*, after at most $6\Delta + 2$ rounds), a unique leader is forever elected by all processes.

5 Correctness

Let $\Delta \in \mathbb{N}^*$. Let $\mathcal{G} = G_1, G_2, ...$ be a dynamic graph with vertex set V of Class $\mathcal{J}^B_{1,*}(\Delta)$. We study the execution of Algorithm $\mathcal{LE}$ in $\mathcal{G}$; we denote this execution by $ex = \gamma_1, \gamma_2, ...$

Let $i \in \mathbb{N}^*$ and p be a process. If $var(p)$ is a variable of p, we denote by $var(p)_i$ the value of $var(p)$ at Configuration γ_i, *i.e.*, at the beginning of Round i.

According to the computational model, the step between Configurations γ_i and γ_{i+1} is performed on the communication network described by the graph G_i. At the end of Round i (which is also the beginning of Round $i+1$), *i.e.*, at Configuration γ_{i+1}, the variable $var(p)$ takes its next value $var(p)_{i+1}$.

Remark 5. *Let $p \in V$. The following remarks are local observations, directly deduced from the code of p.*

(a) If $\langle id(p), -, \Delta\rangle \notin Lstable(p)$, p executes Line 4 and inserts $\langle id(p), 0, \Delta\rangle$ into $Lstable(p)$. Afterwards, this tuple is never removed from $Lstable(p)$ since $Lstable(p)[id(p)].ttl$ never decreases.

Thus, $\forall i > 1$, $id(p) \in Lstable(p)_i$.

(b) If $\langle id(p), -, \Delta\rangle \notin Gstable(p)$ or $Gstable(p)[id(p)].susp \neq Lstable(p)[id(p)].susp$, p executes Line 6 and inserts $Lstable(p)[id(p)]$ into $Gstable(p)$. Afterwards, this tuple is never removed from $Gstable(p)$ since $Gstable(p)[id(p)].ttl$ never decreases. Moreover, each time $Lstable(p)[id(p)].susp$ is incremented, $Gstable(p)[id(p)].susp$ is incremented too, and conversely; see Line 18.

Thus, $\forall i > 1$, $id(p) \in Gstable(p)_i$ and $Gstable(p)_i[id(p)].susp = Lstable(p)_i[id(p)].susp$.

(c) Any message msg such that $msg.id \notin msg.LSPs$ is never sent; see Line 2. Moreover, at each round, every process $q \in V$ removes such messages from $msgs(q)$; Line 24.

Thus, $\forall i > 1$, $\forall msg \in msgs(p)_i$, $msg.id \in msg.LSPs$.

*(d) Let $i \in \mathbb{N}^*$. p receives a message msg during Round i (*i.e.*, msg is in its mailbox at Round i) if and only if there exists a process $q \in \mathcal{IN}(p)^i$ such that $msg \in msg(q)_i$ (at the beginning of Round i), $msg.ttl > 0$ and $msg.id \in msg.LSPs$.*

5.1 Communication Exchanges

We first show some properties on the communication exchanges between processes. Lemma 2 gives conditions to ensure that a message $msg = \langle id, LSPs, \Delta - X\rangle$ in a process mailbox at Round i has been effectively produced by the process identified by id at Round $i - X - 1$. Lemma 3 establishes that the message produced by a given process p at Round $i \in \mathbb{N}^*$ (and so sent at Round $i+1$) is received at Round $i + \hat{d}_{p,i+1}(q)$ by each process q such that $\hat{d}_{p,i+1}(q) \leq \Delta$.

Lemma 2. *Let q be a process in V. Let X and i be two integers such that $0 \leq X < \Delta$ and $i > X + 1$. Let msg be a message such that $msg.ttl = \Delta - X$.*

*If $msg \in msgs(q)_i$ (*i.e.*, at the beginning of Round i), then msg was* initiated *during Round $i - X - 1$ by a process $p \in V$ such that $msg.LSPs = Lstable(p)_{i-X}$ and $msg.id = id(p)$.*

Proof. The fact that if a message msg is initiated (Line 26) by some process p in V during the k-th round with $k \in \mathbb{N}^*$, then $msg.id = id(p)$ and $msg.LSPs = Lstable(p)_{k+1}$ is direct from the code (note that $msg.LSPs$ is never modified).

Hence, it remains to show the first part: precisely, we show by induction on $X \in \{0, ..., \Delta - 1\}$ that $\forall q \in V$, $\forall i > X + 1$, if $msg = \langle id, LSPs, \Delta - X\rangle$ is in $msgs(q)_i$ (*i.e.*, in $msgs(q)$ at the beginning of Round i), then msg was initiated by some process $p \in V$ during Round $i - X - 1$.

Base case ($X = 0$): Let $q \in V$ and $i > 1$. Assume there exists a message $msg \in msgs(q)_i$ such that $msg.ttl = \Delta$.

If a message, msg', is already in $msgs(q)_{i-1}$, *i.e.*, at the beginning of Round $i-1$ or if q received msg' during Round $i-1$ (Line 13), q decrements the associated timer (Line 25) during Round $i-1$. Thus, at the beginning of Round i, the timer associated to msg' is lower than Δ. Since the timer of msg is Δ, the only remaining possibility is that msg was initiated by q (Line 26) during Round $i-1$.

Induction step ($0 < X < \Delta$): Let $q \in V$ and $i > X+1$. Assume there exists a message $msg = \langle id, LSPs, \Delta - X \rangle \in msgs(q)_i$.

Notice that msg was not initiated by q (Line 26) during Round $i-1$, otherwise the associated timer would be Δ. Thus, there are two cases : either q received msg during Round $i-1$ or msg was already in $msgs(q)$ at the beginning of Round $i-1$. In both cases, q decrements the timer associated to the message (Line 25) at the end of Round $i-1$. So:

(a) Either q receives $msg' = \langle id, LSPs, \Delta - X + 1 \rangle$ during Round $i-1$ and msg' was sent by some process q'. Thus, $msg' \in msgs(q')_{i-1}$.

(b) Or, $msg' = \langle id, LSPs, \Delta - X + 1 \rangle \in msgs(q)_{i-1}$.

The induction hypothesis applies in both cases: there is some process $p \in V$ such that $id = id(p)$ that initiated msg' (and so msg) during Round $(i-1) - (X-1) - 1 = i - X - 1$.

□

The next corollary follows from Lemma 2 and Remark 5.(d).

Corollary 12. *Let q be a process in V. Let X and i be integers such that $0 \leq X < \Delta$ and $i > X + 1$.*

If q receives a message $msg = \langle id, LSPs, \Delta - X \rangle$ during Round i, then msg was initiated at Round $i - X - 1$ by some process $p \in V$ such that $LSPs = Lstable(p)_{i-X}$ and $id = id(p)$.

Lemma 3. *$\forall p, q \in V$, $\forall i > 1$, $\forall d \leq \Delta$, $\hat{d}_i(p, q) = d$ implies that*

(a) q receives a message $\langle id(p), Lstable(p)_i, \Delta - d + 1 \rangle$ during Round $i + d - 1$ if $q \neq p$; and

(b) $\langle id(p), Lstable(p)_i, \Delta - d \rangle \in msgs(q)_{i+d}$.

Proof. By induction on $d \in \{0, ..., \Delta\}$.

Base case ($d = 0$): Let $i > 1$. Let $p, q \in V$ such $\hat{d}_i(p, q) = 0$. Then $p = q$, by definition. Process p ends each round, and in particular Round i, by inserting the tuple $\langle id(p), Lstable(p)_i, \Delta \rangle$ into $msgs(p)$; see Line 26. Hence, $\langle id(p), Lstable(p)_i, \Delta \rangle \in msgs(q)_i$.

Induction step ($0 < d \leq \Delta$): Let $i > 1$. Let $p, q \in V$ such $\hat{d}_i(p, q) = d$ (*n.b.*, $p \neq q$). By definition, there is a journey $\mathcal{J} = (e_1, t_1), ...(e_k, t_k) \in \mathcal{J}(p, q)$ such that $t_1 \geq i$, $t_k = i + d - 1$ and $e_k = (q', q)$ for some $q' \in V$. Hence, $\hat{d}_i(p, q') = d' < \hat{d}_i(p, q) = d$ and we can apply the induction hypothesis to q': $\langle id(p), Lstable(p)_i, \Delta - d' \rangle \in msgs(q')_{i+d'}$. During every round from Round $i + d'$ to Round $i + d - 1$, the timer of this message decreases by one at each round (see Line 25). Hence, since the timer is equal to $\Delta - d'$ at the beginning of Round $i + d'$, the timer is equal to $\Delta - d + 1$ at the beginning of Round $i + d - 1$ (indeed, $i + d - 1 - (i + d') = \Delta - d' - (\Delta - d + 1)$).

Notice that, meanwhile and since $\Delta - d + 1 > 0$, the message has not been discarded from $msgs(q')$ (the first and second fields remaining unchanged).

As $e_k = (q', q)$ is an edge of G_{i+d-1} (by definition of $\mathcal{J}$), the timer of the message $(\Delta - d + 1)$ is positive, and $id(p) \in Lstable(p)_i$ (Remark 5.(a)), Remark 5.(d) applies: q receives a message $msg = \langle id(p), Lstable(p)_i, \Delta - d + 1\rangle$ during Round $i + d - 1$; proving *(a)*.

We now look at the algorithm of q during Round $i + d - 1$: it treats the message msg using Line 13.

If $\langle id(p), -, \Delta - d + 1\rangle \notin msgs(q)_{i+d-1}$ then msg is inserted into $msgs(q)$. As $\Delta - d + 1 > 0$ and $id(p) \in Lstable(p)_i$ (Remark 5.(a)), its timer is decreased; see Line 25. Hence, we have $\langle id(p), Lstable(p)_i, \Delta - d\rangle \in msgs(q)_{i+d}$.

Otherwise, there exists some map M such that $\langle id(p), M, \Delta - d + 1\rangle \in msgs(q)_{i+d-1}$. We can apply Lemma 2 with "X"$= d - 1 \in \{0, ..., \Delta - 1\}$ and "i"$= i + d - 1$ since $i + d - 1 > d - 1 + 1$: we deduce that $M = Lstable(p)_{i+d-1-(d-1)} = Lstable(p)_i$. The conclusion is the same as in the former case, and we are done with *(b)*.

□

5.2 Properties on Locally and Globally Stable Processes

A process q is memorized into $Lstable(p)$ at Round $i + \Delta + 1$ (*i.e.*, $q \in Lstable(p)_{i+\Delta+1}$) if and only if a message from q was received by p during the time interval $[i + 1, i + \Delta]$ (Corollary 13 and Lemma 6). A process q is memorized into $Gstable(p)$ at Round $i + \Delta + 1$ (*i.e.*, $q \in Gstable(p)_{i+\Delta+1}$) if and only if a message $\langle -, LSPs, -\rangle$ such that $id(q) \in LSPs$ was received by p during the time interval $[i + 1, i + \Delta]$ (Lemma 5 and Lemma 7).

Lemma 4. *Let $i > 1$. Let p be a process of V. If no tuple $\langle id, -, -\rangle$ is inserted in $Lstable(p)$ by p at Line 15 during Δ consecutive rounds starting at Round i (i.e., from Round i to $i + \Delta - 1$), then $id \notin Lstable(p)_{i+\Delta}$ or $id = id(p)$.*

Proof. Let $i > 1$ and $p \in V$. Let $id \in IDSET$ such that $id \neq id(p)$. Assume that no tuple $\langle id, -, -\rangle$ is inserted into $Lstable(p)$ by p at Line 15 during Round i, ..., $i + \Delta - 1$.

Since $id \neq id(p)$, Line 15 is the only way for p to insert $\langle id, -, -\rangle$ into $Lstable(p)$. So, if no tuple $\langle id, -, -\rangle$ exists in $Lstable(p)_i$, we are done.

Otherwise, some tuple $\langle id, -, ttl\rangle$ exists in $Lstable(p)_i$ (*i.e.*, at the beginning of and during Round i) with $ttl \in \{0, ..., \Delta\}$. Now, as $ttl \leq \Delta$, $Lstable(p)[id].ttl$ necessarily reaches 0 during Round $i + \max(0, ttl - 1)$ since if positive, it decreases by one at each round (Line 8). Then, the tuple corresponding to id is removed from $Lstable(p)$; see Line 20. Therefore, no tuple $\langle id, -, -\rangle$ exists in $Lstable(p)_{i+\max(1,ttl)}$. As such tuple can neither be inserted until Round $i + \Delta$, this proves that no tuple $\langle id, -, -\rangle$ exists in $Lstable(p)_{i+\Delta}$. □

For the previous lemma, we have the following corollary.

Corollary 13. *Let $i > 1$. Let p be a process of V. If p receives no message $\langle id, -, -\rangle$ during Δ consecutive rounds starting at Round i (i.e., from Round i to $i + \Delta - 1$), then $id \notin Lstable(p)_{i+\Delta}$ or $id = id(p)$.*

The following result about globally stable processes is very similar to the previous lemma; and so is the proof.

Lemma 5. *Let $i > 1$. Let p be a process of V and $id \in IDSET$. If p receives no message $\langle -, LSPs, - \rangle$ with $id \in LSPs$ during Δ consecutive rounds starting from Round i (i.e., from Round i to $i + \Delta - 1$), then $id \notin Gstable(p)_{i+\Delta}$ or $id = id(p)$.*

Proof. Let $i > 1$, $p \in V$, and id be a process identifier distinct from $id(p)$. Assume that p receives no message $\langle -, LSPs, - \rangle$ with $id \in LSPs$ from Round i to $i + \Delta - 1$.

The only way to insert a tuple $\langle id, -, - \rangle$ into $Gstable(p)$ is by Line 6 (this inserts $Lstable(p)[id(p)]$ but $Lstable(p)[id(p)].id = id(p) \neq id$) or by receiving some message $\langle id', LSPs, - \rangle$ with $id' \neq id(p)$ (see Line 17): this line inserts every tuple $\langle id'', susp, \Delta \rangle$ such that $\langle id'', susp, - \rangle \in LSPs$ and $id'' \neq id(p)$. So, as no message $\langle -, LSPs, - \rangle$ with $id \in LSPs$ is received, no message $\langle id, -, - \rangle$ is inserted into $Gstable(p)$ during Round i, ..., $i + \Delta - 1$.

If no tuple $\langle id, -, - \rangle$ exists in $Gstable(p)_i$, we are done. Otherwise, some tuple $\langle id, -, ttl \rangle$ exists in $Gstable(p)_i$ (*i.e.*, at the beginning of and during Round i) with $ttl \in \{0, ..., \Delta\}$. Now, as $ttl \leq \Delta$, $Gstable(p)[id].ttl$ necessarily reaches 0 during Round $i + \max(0, ttl - 1)$ since if positive, it decreases by one at each round; see Line 10. Then, the tuple corresponding to id is removed from $Gstable(p)$; see Line 22. Therefore, no tuple $\langle id, -, - \rangle$ exists in $Gstable(p)_{i+\max(1,ttl)}$. As such tuple can neither be inserted until Round $i + \Delta$, this proves that no tuple $\langle id, -, - \rangle$ exists in $Gstable(p)_{i+\Delta}$. □

Lemma 6. *Let $i > 1$ and $j \in \{i, ..., i + \Delta - 1\}$. Let p be a process of V. Let $id \in IDSET$ distinct from $id(p)$. If p receives a message $\langle id, -, \Delta + i - j \rangle$ during Round j, then $id \in Lstable(p)_{i+\Delta}$.*

Proof. Let $i > 1$ and $j \in \{i, ..., i + \Delta - 1\}$, $p \in V$, $id \in IDSET$ such that $id \neq id(p)$. Assume that p receives a message $\langle id, -, \Delta + i - j \rangle$ during Round j.

According to Line 14 and 15, $\langle id, -, ttl \rangle \in Lstable(p)_{j+1}$ with $ttl \geq \Delta + i - j$ (it is not removed at Line 20 since $\Delta + i - j > 0$). The timer associated to id in $Lstable(p)$ is decremented at most by one (Line 8) during a round (*n.b.*, when the tuple $\langle id, -, - \rangle$ is updated in $Lstable(p)$, its timer does not decrease; see Lines 14-15) and a tuple $\langle id, -, ttl' \rangle$ is removed from $Lstable(p)$ at some round only if ttl' reaches 0 during that round (Line 20). As the timer of id in $Lstable(p)_{j+1}$ is: $ttl \geq \Delta + i - j$, a tuple $\langle id, -, - \rangle$ remains in $Lstable(p)$ during at least $\Delta + i - j - 1$ rounds, *i.e.*, from Round $j + 1$ to Round $\Delta + i$ (since $j + 1 + (\Delta + i - j) - 1 = \Delta + i$). □

Lemma 7. *Let $i > 1$ and $p \in V$. Let $id \in IDSET$ distinct from $id(p)$ and $LSPs \in MapType$ such that $id \in LSPs$. If p receives a tuple $\langle -, LSPs, - \rangle$ during Round i, then $\forall j \in \{i + 1, ..., i + \Delta\}$, $id \in Gstable(p)_j$.*

Proof. Let $i > 1$. Let $p \in V$. Let $id \in IDSET$ such that $id \neq id(p)$ and $LSPs \in MapType$ such that $id \in LSPs$. Assume that p receives a tuple $\langle -, LSPs, - \rangle$ during Round i.

According to Line 17, Process p inserts $\langle id, -, \Delta \rangle$ into $Gstable(p)$ during Round i. So $\langle id, -, \Delta \rangle \in Gstable(q)_{i+1}$ (it is not removed at Line 22 since $\Delta > 0$). The timer associated to id in $Gstable(p)$ is decremented at most by one (Line 10) during a round (*n.b.*, when the tuple $\langle id, -, - \rangle$ is updated $Gstable(p)$, its timer is not decremented; see Line 17), and a tuple $\langle id, -, ttl \rangle \in Gstable(p)$ is removed during a round only if ttl reaches 0 during that round; see Line 22. As the timer of id in $Gstable(p)_{i+1}$ is Δ, a tuple $\langle id, -, - \rangle$ remains in $Gstable(p)$ during at least $\Delta - 1$ rounds, *i.e.*, from Round $i + 1$ to Round $\Delta + i$ (since $i + 1 + \Delta - 1 = \Delta + i$). □

5.3 Removing Fake IDs

Lemma 8. *Let f be a fake ID, $p \in V$, $i > 4\Delta$, $msg \in msgs(p)_i$ be a message. We have: (a) $msg.id \neq f$, (b) $f \notin Lstable(p)_i$ (c) $f \notin msg.LSPs$, and (d) $f \notin Gstable(p)_i$.*

Proof. Let f be a fake ID.

(a) First, any process only initiates messages with its own ID (Line 26). So no message $\langle f, -, - \rangle$ are ever initiated. Then, if some $msgs(q)$ with $q \in V$ contains some message $\langle f, -, ttl \rangle$ at the beginning of the execution, then thanks to the timer mechanism, the timer in this message and in all its copies broadcasted into the network will reach 0 within at most Δ rounds. Then, those messages will not be sent (Line 2) and finally will be removed from every $msgs$ set where they appear (Line 24). Hence, $\forall i_1 > \Delta$, $msgs(p)_{i_1}$ contains no message $\langle f, -, - \rangle$, for every $p \in V$.

(b) Let $p \in V$. By Corollary 13, for every Round $i_2 \geq i_1 + \Delta > 2\Delta$, $f \notin Lstable_{i_2}(p)$.

(c) Let $p \in V$. Let $msg = \langle id, LSPs, ttl \rangle$ be a message in $msgs(p)$ at the beginning of Round $i_3 > 3\Delta$. By Lemma 2, there is a process $q \in V$ such that $id = id(q)$ and $LSPs = Lstable(q)_{i_3 - \Delta + ttl}$. As $i_3 - \Delta + ttl > 2\Delta$, we apply (b): $f \notin Lstable(q)_{i_3 - \Delta + ttl} = LSPs$.

(d) Let $p \in V$. Since (c) holds, p receives no message $\langle -, LSPs, - \rangle$ such that $f \in LSPs$, from Round i_3; we can apply Lemma 5: for every Round $i_4 \geq i_3 + \Delta > 4\Delta$, $f \notin Gstable(p)_{i_4}$. □

5.4 Stable Memorization of some Processes in the Map $Gstable$

Let $TSources$ be the set of timely sources of $\mathcal{G}$. Timely sources (and may be some other processes) increment their suspicion variable only finitely many times (Lemma 10). Let p be a process whose suspicion value is constant from Round t_p (*i.e.*, a process of $\Diamond Const$); We show that p is memorized forever by every process q at least from Round $i \geq t_p + \Delta + 2$ (*i.e.*, $p \in Gstable(q)_i$); see Lemma 12. In Subsection 5.5, we establish that the finally elected process is a process of $\Diamond Const$.

Definition 6. *We define $TSources$ as the set of timely sources of $\mathcal{G}$. Namely, for a process $r \in V$,*

$$r \in TSources \text{ if and only if } \forall p \in V, \forall i \in \mathbb{N}^*, \hat{d}_i(r, p) \leq \Delta$$

Note that $TSources \neq \emptyset$ since $\mathcal{G} \in \mathcal{J}^B_{1,*}(\Delta)$.

Lemma 9. *Let $r \in TSources$. $\forall k > \Delta + 1$, $\forall p \in V$, $id(r) \in Lstable(p)_k$.*

Proof. The proof is done with $k = i + \Delta$, for some $i > 1$. Let $i > 1$, $r \in TSources$ and $p \in V$. According to Remark 5.(a), $id(r) \in Lstable(r)_{i+\Delta}$. Assume that $p \neq r$ and $\hat{d}_i(r, p) = d$. As r is a source and $p \neq r$, we have $1 \leq d \leq \Delta$. By Lemma 3, p receives a message $\langle id(r), -, \Delta - d + 1 \rangle$ during Round $i + d - 1$. According to Lemma 6, we have $id(r) \in Lstable(p)_{i+\Delta}$. □

Definition 7. *For every $i \in \mathbb{N}^*$ and $p \in V$, let $suspicion(p)_i$ be defined as follows:*

$$suspicion(p)_i = \begin{cases} -\infty & \textbf{if } id(p) \notin Lstable(p)_i, \\ Lstable(p)_i[id(p)].susp & \textbf{otherwise}. \end{cases}$$

Notice that, by Remark 5.(a), $suspicion(p)_i \neq -\infty$ for every $i > 1$.

Definition 8. *We define the set $\Diamond Const$ of processes in V. Let p be a process of V.*

$p \in \Diamond Const$ if and only if $\exists maxSuspicion(p) \in \mathbb{N}, \exists t_p \in \mathbb{N}^*, \forall i \geq t_p, suspicion(p)_i = maxSuspicion(p)$.

Namely, $\Diamond Const$ is the set of processes whose suspicion counter is eventually constant.

Lemma 10. $TSources \subseteq \Diamond Const \neq \emptyset$ *and* $\forall p \in TSources, t_p \leq 2\Delta + 1$.

Proof. Let $r \in TSources$. Let $i > 2\Delta + 1$. We first show that for every processes q and p, if $\langle id(q), LSPs, ttl \rangle \in msgs(p)_i$ for some $ttl \in \{0, ..., \Delta\}$ and $LSPs \in MapType$, then $r \in LSPs$. Indeed, by Lemma 2, this message was initiated by Process q and $LSPs = Lstable(q)_{i-X}$ with $X = \Delta - ttl$. Lemma 9 allows us to conclude that $r \in LSPs$, since $i - X > \Delta + 1$.

Therefore, every message received by r during any Round $i > 2\Delta + 1$ contains r in its source field. So r does not augment its suspicion value during any Round $i > 2\Delta + 1$ (see Line 18); hence $\forall i > 2\Delta + 1, suspicion(r)_i = suspicion(r)_{2\Delta+1}$. □

Lemma 11. *Let* $p \in \Diamond Const$. $\forall i > t_p$, $\forall r \in TSources$, $id(p) \in Lstable(r)_i$.

Proof. Let $p \in \Diamond Const$. Let $r \in TSources$. Assume, by the contradiction, that $id(p) \notin Lstable(r)_i$ with $i > t_p$. Note that $p \neq r$, by Remark 5.(a). We have $\hat{d}_i(r, p) = d \in \{1, ..., \Delta\}$ since $p \neq r$ and $r \in TSources$. By Lemma 3, p receives the message $\langle id(r), Lstable(r)_i, \Delta - d + 1 \rangle$ during Round $j = i + d - 1$. As $id(p) \notin Lstable(r)_i$, the value of $Lstable(p)[p].susp$ is incremented (Line 18) during Round j. Since the suspicion value of a process can never decrease, $suspicion_i(p) \leq suspicion_j < suspicion(p)_{j+1}$ and $j \in \{i, ..., i + \Delta - 1\}$, *i.e.*, $j \geq i > t_p$. A contradiction to the definition of $\Diamond Const$. □

Lemma 12. *Let* $p \in \Diamond Const$. $\forall i \geq t_p + \Delta + 1$, $\forall q \in V$, $id(p) \in Gstable(q)_i$.

Proof. Let $p \in \Diamond Const$. Let $q \in V$.

- If $q = p$, since $i > 1$, $id(p) \in Gstable(q)_i$; see Remark 5.(b).
- If $q \neq p$ and $q \in TSources$, by Lemma 11, we have that $\forall j > t_p$, $id(p) \in Lstable(q)_j$. Now, as $id(p) \neq id(q)$, we use the contraposition of Corollary 13: let $j > \max(\Delta + 1, t_p)$; there exists a round $k \in \{j - \Delta, ..., j - 1\}$ during which q receives a message $\langle id(p), LSPs, ttl \rangle$.

  By Remark 5.(d), $id(p) \in LSPs$ since $k \geq j - \Delta > 1$. Thus, by Lemma 7, $\forall x \in \{k + 1, ..., k + \Delta\}$, $id(p) \in Gstable(q)_x$. So $id(p) \in Gstable(q)_j$ whatever be the value of k (indeed, take $x = k + (j - k)$ since $1 \leq j - k \leq \Delta$). Thus, $\forall j > \max(\Delta + 1, t_p)$, $id(p) \in Gstable(q)_j$. Since, $\max(\Delta + 1, t_p) \leq t_p + \Delta$, we have $\forall j \geq t_p + \Delta + 1$, $id(p) \in Gstable(q)_j$.
- If $q \neq p$ and $q \notin TSources$, let $r \in TSources$: $\forall j \in \mathbb{N}^*$, $\hat{d}_j(r, q) = d \leq \Delta$.

  Let $j > t_p$. According to Lemma 3, q receives a message $\langle id(r), Lstable(r)_j, \Delta - d + 1 \rangle$ during Round $j + d - 1$. As $j > t_p$, by Lemma 11, we have $id(p) \in Lstable(r)_j$. According to Lemma 7, $\forall k \in \{j + d, ..., j + d - 1 + \Delta\}$, we have $id(p) \in Gstable(q)_k$. This result holds for every $j > t_p$. Thus, $\forall i \geq j + d \geq j + \Delta \geq t_p + \Delta + 1$, $id(p) \in Gstable(q)_i$.

Hence, and in any case, $\forall i \geq t_p + \Delta + 1$, $p \in Gstable(q)_i$.

□

5.5 Accuracy of the memorized suspicion values

In this section, we establish that if $id(p) \in Gstable(q)$ at Round $t + 4\Delta - 2$, then the suspicion value of p memorized by q at that round (*i.e.*, $Gstable(q)_{t+4\Delta-2}[p].susp$) is the suspicion value of p at a previous round not early than round t (Lemma 16). Then, using Lemma 12, we can deduce that at any round $t \geq t_p + 4\Delta - 2$, every process has memorized $maxSuspicion(p)$ as the suspicion value of p, for every

$p \in \Diamond Const$. Finally, the suspicion value of processes not belonging to $\Diamond Const$ never stop to increase along the execution, ex. Hence, we can conclude that $\ell = \min\{id(p) : p \in \Diamond Const \wedge maxSuspicion(p) = min_{r \in \Diamond Const}\{maxSuspicion(r)\}\}$ is eventually elected by every process forever (Theorem 8).

Lemma 13. *Let q be a process of V. $\forall i \geq \Delta + 1$, for every tuple $\langle id, LSPs, - \rangle$ received by q during Round i, $\exists p \in V$ such that $id = id(p)$, $id(p) \in LSPs$ and $\exists t \in \{i - \Delta + 1, ..., i\}$, $LSPs[id(p)].susp = suspicion(p)_t$.*

Proof. Let q be a process of V. Let $ttl \in \{0, ..., \Delta\}$ and $i \geq \Delta + 1$. Assume that q receives a tuple $msg = \langle id, LSPs, ttl \rangle$ during Round i. Hence $ttl \geq 1$ and Corollary 12 applies: there exists a process $p \in V$ such that $id = id(p)$, $LSPs = Lstable(p)_t$ with $t = i - \Delta + ttl$. We have $t \in \{i - \Delta + 1, ..., i\}$. As $t > 1$, by Remark 5.(a), $id(p) \in Lstable(p)_t = LSPs$. Moreover, $LSPs[id(p)].susp = Lstable(p)_t[id(p)].susp = suspicion(p)_t$, by definition. □

Lemma 14. *Let p and q be two distinct processes of V. $\forall i \geq 2\Delta + 1$, if $id(p) \in Lstable(q)_i$, then there exists $t \in \{i - 2\Delta + 1, ..., i - 1\}$ such that $Lstable(q)_i[id(p)].susp = suspicion(p)_t$.*

Proof. Let $p, q \in V$ such that $p \neq q$. Let $i \geq 2\Delta + 1$. Let $s \in \mathbb{N}$.

First, at least one tuple $\langle id(p), -, - \rangle$ is inserted into $Lstable(q)$ by q at Line 15 during the last Δ round, by Lemma 4. Such insertions are due to receptions at q of messages $\langle id(p), -, - \rangle$ during the same rounds.

Consider then any message $\langle id(p), LSPs, - \rangle$ received by q at some round $X' \in \{i - \Delta, ..., i - 1\}$. By Lemma 13, and since $X' \geq \Delta + 1$, $id(p) \in LSPs$ and there exists $t \in \{X' - \Delta + 1, ..., X'\}$ (*i.e.*, $t \in \{i - 2\Delta + 1, ..., i - 1\}$) such that $LSPs[id(p)].susp = suspicion(p)_t$. This is in particular true for each of these messages that causes an insertion of into $Lstable(q)$ for $id(p)$. Since this is the only to modify the suspicion value of p at q, we are done.

□

Lemma 15. *Let q be a process of V. Let $i \geq 3\Delta$. For every message $\langle -, LSPs, - \rangle$ received by q at Round i, if there exists a process $p \in V$ such that $id(p) \in LSPs$, then there exists $t \in \{i - 3\Delta + 2, ...i\}$ such that $LSPs[id(p)].susp = suspicion(p)_t$.*

Proof. Let $p, q \in V$. Let $i \geq 3\Delta$. Assume that q receives a message $\langle id, LSPs, ttl \rangle$ during Round i with $id(p) \in LSPs$. First, $ttl \in \{1, ..., \Delta\}$, by definition of the algorithm. By Corollary 12 and as $i > \Delta$, there is a process x such that $id = id(x)$ and $LSPs = Lstable(x)_t$ with $t = i - \Delta + ttl$. Note that $i - \Delta + 1 \leq t \leq i$.

If $p \neq x$, then since $t \geq i - \Delta + 1 \geq 2\Delta + 1$, we have $LSPs[id(p)].susp = Lstable(x)_t[id(p)].susp = suspicion(p)_{t'}$ with $t' \in \{t - 2\Delta + 1, ..., t - 1\}$, by Lemma 14.

If $p = x$ then by Remark 5, $LSPs[id(p)].susp = Lstable(x)_t[id(p)].susp = suspicion(p)_{t'}$ for $t' = t$.

Since $t \in \{i - \Delta + 1, \ldots, i\}$, in both cases, we have $t' \in \{i - 3\Delta + 2, ..., i\}$, and we are done. □

Lemma 16. *Let p and q be two distinct processes. Let $i \geq 4\Delta$. If $id(p) \in Gstable(q)_i$ then there exists $t \in \{i - 4\Delta + 2, ..., i - 1\}$ such that $Gstable(q)_i[id(p)].susp = suspicion(p)_t$.*

Proof. Let p and q be two distinct processes. Let $i \geq 4\Delta$. Assume $id(p) \in Gstable(q)_i$.

By Lemma 5, since $id(p) \in Gstable(q)_i$, $id(p) \neq id(q)$ and $i - \Delta > 1$: there exists $X \in \{1, ..., \Delta\}$ such that q receives some message $\langle -, LSPs, - \rangle$ with $id(p) \in LSPs$ at Round $i - X$. We note

$X_m = \min\{X : 1 \leq X \leq \Delta$ and q receives some message $\langle -, M, - \rangle$ such that $id(p) \in M$ during Round $i - X\}$

(*i.e.*, $i - X_m$ is the latest round when q receives such a message until Round i).

Let $\langle -, LSPs, - \rangle$ be any message received by q during Round $i - X_m$ such that $id(p) \in LSPs$. As $i - X_m \geq 3\Delta$, we have $LSPs[id(p)].susp = suspicion(p)_t$ with $t \in \{i - X_m - 3\Delta + 2, ..., i - X_m\}$, by Lemma 15. Hence, during Round $i - X_m$, $Gstable(q)[id(p)].susp$ is set to $suspicion(p)_t$; see Line 17. Note that $t \in \{i - 4\Delta + 2, ..., i - 1\}$.

By construction of X_m, no message $\langle -, M, - \rangle$ such that $id(p) \in M$ is received during Round $i - X_m + 1$ to Round $i - 1$. Furthermore, since $id(p) \in Gstable(q)_i$, the information of $id(p)$ has not been removed from $Gstable(q)$ meanwhile. Hence, $Gstable(q)_i[p].susp = suspicion(p)_t$. □

By definition of the $\Diamond Const$ set, we have the following remark.

Remark 6. *$\exists T \in \mathbb{N}^*$ such that $\forall i \geq T$,*

a) $\forall p \in V \setminus \Diamond Const$ we have $suspicion(p)_i > min_{r \in \Diamond Const}\{maxSuspicion(r)\}$

b) $\forall p' \in \Diamond Const$, we have $suspicion(p')_i = maxSuspicion(p')$

Notice that $T \geq t_{p'}, \forall p' \in \Diamond Const$.

Theorem 8. *$\forall i \geq T + 4\Delta + 1$, $\forall q \in V$, $lid(q) = \ell = \min\{id(p) : p \in \Diamond Const \wedge maxSuspicion(p) = min_{r \in \Diamond Const}\{maxSuspicion(r)\}\}$ at the beginning of Round i.*

Proof. Let q be a process of V. Let $i \geq T + 4\Delta + 1$. By Lemma 8, follows.

(a) For every $id \in Gstable(q)_i$, there exists a process $p \in V$ such that $id(p) = id$.

Then, we also have the following property.

(b) For every process $p \in V$, if $id(p) \in Gstable(q)_i$, then there exists $t \in \{T, ..., i\}$ such that $Gstable(q)_i[id(p)].susp = suspicion(p)_t$.

Indeed, if $p = q$ then by Remark 5.(b), $Gstable(q)_i[id(q)].susp = Lstable(q)_i[id(q)].susp = suspicion(q)_i$, as $i \geq 2$. Otherwise, $p \neq q$ and by Lemma 16, $Gstable(q)_i[id(p)].susp = suspicion(p)_t$ with $i - 4\Delta + 2 \leq t \leq i - 1$.

Now, as the suspicion value of any process p cannot decrease after the first round, $Gstable(q)_i[id(p)].susp \geq suspicion(p)_t \geq suspicion(p)_T$, for every $id(p) \in Gstable(q)_i$. Hence, by definition of T, we have

(c) $\forall p \notin \Diamond Const$, $Gstable(q)_i[id(p)].susp > min_{r \in \Diamond Const}\{maxSuspicion(r)\}$.

Let $p \in \Diamond Const$. By Lemma 12, $id(p) \in Gstable(q)_i$, since $i \geq t_p + \Delta + 1$. Moreover, by (b), there exists $t \in \{T, ..., i\}$ such that $Gstable(q)_i[id(p)].susp = suspicion(p)_t$. Now, as $t \geq t_p$ (by definition of $\Diamond Const$), $suspicion(p)_t = maxSuspicion(p)$, and follows.

(d) $\forall p \in \Diamond Const$, $id(p) \in Gstable(q)_i$ and $Gstable(q)_i[id(p)].susp = maxSuspicion(p)$.

Thus, at the beginning of Round i, $Gstable(q)_i$

- only contains records tagged with identifiers of processes of V by (a);
- may contain records tagged with identifiers of non-members of $\Diamond Const$, but their associated suspicion values are strictly greater than $min_{r \in \Diamond Const}\{maxSuspicion(r)\}$, by (c); and
- contains a record for each member p of $\Diamond Const$ with an associated suspicion value equal to the maximum suspicion value of p, by (d).

Hence, at the end of Round $i - 1$ (and so, at the beginning of Round i), the process elected by q, $lid(q)_i$, is ℓ; see Line 27. □

Corollary 14. *Algorithm $\mathcal{LE}$ is a deterministic pseudo-stabilizing leader election algorithm for $\mathcal{J}^B_{1,*}(\Delta)$.*

5.6 Speculation

The pseudo-stabilization time of Algorithm $\mathcal{LE}$ cannot be bounded in $\mathcal{J}^B_{1,*}(\Delta)$ by Theorem 5. However, Algorithm $\mathcal{LE}$ is speculative in the sense that its pseudo-stabilization time in $\mathcal{J}^B_{*,*}(\Delta) \subset \mathcal{J}^B_{1,*}(\Delta)$ is bounded. Indeed, by definition, in $\mathcal{J}^B_{*,*}(\Delta)$, $\forall p \in V$, $p \in TSources$ (*i.e.*, $V \setminus \Diamond Const = \emptyset$). Then, by Lemma 10, $\forall p \in V$, $t_p = 2\Delta + 1$. So, $T = 2\Delta + 1$, and $\forall i \geq 6\Delta + 2$, $lid(p) = \ell$ at the beginning of Round i, by Theorem 8.

6 Conclusion

We have studied the expressive power, w.r.t. the (deterministic) leader election problem, of pseudo- and self-stabilization in nine classes of dynamic graphs. Our results show that, self-stabilizing leader election can only be solved in the three classes studied in [2]. Furthermore, even pseudo-stabilizing leader election cannot be solved in the remaining classes, except in the class where at least one process is a timely source.

We have defined those classes by analogy with classes mainly studied for the Ω problem in crash-prone partially synchronous systems. We have simply skipped several dynamic patterns classically used in those systems, *e.g.*, patterns related to eventual timeliness and the notion of bi-source. Actually, as explained in [12], the fact that the bound immediately holds (timeliness) or only eventually (eventually timeliness) as no impact on stabilizing systems: just consider the first configuration from which the bound is guaranteed as the initial point of observation. Similarly, dealing with dynamic graphs where at least one process is a bi-source (*i.e.*, a process which is both a source and a sink) is not an issue in the expressive point of view. Indeed, the existence of a bi-source makes those dynamic graphs belonging to the class $\mathcal{J}_{*,*}$ since any bi-source acts as a hub during a flooding.

Concerning now the efficiency of solutions, our results give a broad answer in terms of time complexity, in particular about stabilization time. However, the space complexity aspect remains widely open. In particular, two of our stabilizing solutions, namely those for $\mathcal{J}_{*,*}$ (proposed in [2]) and $\mathcal{J}^B_{*,1}(\Delta)$ (proposed here), use an infinite memory. Actually, we conjecture that this drawback cannot be precluded.

References

[1] Marcos Kawazoe Aguilera, Carole Delporte-Gallet, Hugues Fauconnier, and Sam Toueg. On implementing omega in systems with weak reliability and synchrony assumptions. *Distributed Comput.*, 21(4):285–314, 2008.

[2] Karine Altisen, Stéphane Devismes, Anaïs Durand, Colette Johnen, and Franck Petit. Self-stabilizing Systems in Spite of High Dynamics. In *the 22th International Conference on Distributed Computing and Networking (ICDCN 2021)*, Nara, Japan, 5-8 January 2021. to appear.

[3] Karine Altisen, Stéphane Devismes, Anaïs Durand, and Franck Petit. Gradual stabilization. *JPDC*, 123:26–45, 2019.

[4] Matthieu Barjon, Arnaud Casteigts, Serge Chaumette, Colette Johnen, and Yessin M. Neggaz. Maintaining a distributed spanning forest in highly dynamic networks. *Comput. J.*, 62(2):231–246, 2019.

[5] Marjorie Bournat, Ajoy K. Datta, and Swan Dubois. Self-stabilizing robots in highly dynamic environments. *Theor. Comput. Sci.*, 772:88–110, 2019.

[6] Nicolas Braud-Santoni, Swan Dubois, Mohamed-Hamza Kaaouachi, and Franck Petit. The next 700 impossibility results in time-varying graphs. *IJNC*, 6(1):27–41, 2016.

[7] James E. Burns, Mohamed G. Gouda, and Raymond E. Miller. Stabilization and pseudo-stabilization. *Distributed Computing*, 7(1):35–42, 1993.

[8] Shukai Cai, Taisuke Izumi, and Koichi Wada. How to prove impossibility under global fairness: On space complexity of self-stabilizing leader election on a population protocol model. *Theory Comput. Syst.*, 50(3):433–445, 2012.

[9] Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *Inter. J. of Parall., Emergent and Dist. Systems*, 27(5):387–408, 2012.

[10] Bernadette Charron-Bost and Shlomo Moran. The firing squad problem revisited. In *STACS*, pages 20:1–20:14, 2018.

[11] Ajoy K. Datta and Lawrence L. Larmore. Self-stabilizing leader election in dynamic networks. *Theory Comput. Syst.*, 62(5):977–1047, 2018.

[12] Carole Delporte-Gallet, Stéphane Devismes, and Hugues Fauconnier. Stabilizing leader election in partial synchronous systems with crash failures. *JPDC*, 70(1):45–58, 2010.

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

[14] Shlomi Dolev. Optimal time self-stabilization in uniform dynamic systems. *Parallel Process. Lett.*, 8(1):7–18, 1998.

[15] Shlomi Dolev, Ariel Hanemann, Elad Michael Schiller, and Shantanu Sharma. Self-stabilizing end-to-end communication in (bounded capacity, omitting, duplicating and non-fifo) dynamic networks - (extended abstract). In *SSS 2012*, volume 7596, pages 133–147. Springer, 2012.

[16] Shlomi Dolev and Ted Herman. Superstabilizing protocols for dynamic distributed systems. *Chicago Journal of Theoretical Computer Science*, 1995.

[17] Shlomi Dolev, Amos Israeli, and Shlomo Moran. Self-stabilization of dynamic systems assuming only read/write atomicity. *Dist. Comput.*, 7(1):3–16, 1993.

[18] S. Dubois and R. Guerraoui. Introducing speculation in self-stabilization: an application to mutual exclusion. In *PODC*, pages 290–298, 2013.

[19] Carlos Gómez-Calzado, Arnaud Casteigts, Alberto Lafuente, and Mikel Larrea. A connectivity model for agreement in dynamic systems. In *Euro-Par*, pages 333–345, 2015.

[20] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. L. Wong. Zyzzyva: Speculative byzantine fault tolerance. *ACM Trans. Comput. Syst.*, 27(4):7:1–7:39, 2009.

[21] B. Xuan, A. Ferreira, and A. Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *IJFCS*, 14(02):267–285, 2003.