Self-stabilizing Systems in Spite of High Dynamics[⋆]

Karine Altisen

Université Grenoble Alpes, VERIMAG

Stéphane Devismes

Université de Picardie Jules Verne, MIS

Anaïs Durand

Université de Clermont Auvergne, LIMOS

Colette Johnen

Université de Bordeaux, LaBRI

Franck Petit

Sorbonne Université, LIP6

Abstract

We initiate research on self-stabilization in highly dynamic identified message-passing systems where dynamics is modeled using time-varying graphs (TVGs). More precisely, we address the self-stabilizing leader election problem in three wide classes of TVGs: the class $\mathcal{TC}^{\mathcal{B}}(\Delta)$ of TVGs with temporal diameter bounded by Δ, the class $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ of TVGs with temporal diameter quasi-bounded by Δ, and the class $\mathcal{TC}^{\mathcal{R}}$ of TVGs with recurrent connectivity only, where $\mathcal{TC}^{\mathcal{B}}(\Delta) \subseteq \mathcal{TC}^{\mathcal{Q}}(\Delta) \subseteq \mathcal{TC}^{\mathcal{R}}$. We first study conditions under which our problem can be solved. We introduce the notion of size-ambiguity to show that the assumption on the knowledge of the number n of processes is central. Our results reveal that, despite the existence of unique process identifiers, any deterministic self-stabilizing leader election algorithm working on the class $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ or $\mathcal{TC}^{\mathcal{R}}$ cannot be size-ambiguous, justifying why our solutions for those

⋆This study has been partially supported by the French ANR project ANR-22-CE25-0008-01 (SkyData).

classes assume the exact knowledge of n. We then present three self-stabilizing leader election algorithms for Classes $\mathcal{TC}^{\mathcal{B}}(\Delta)$, $\mathcal{TC}^{\mathcal{Q}}(\Delta)$, and $\mathcal{TC}^{\mathcal{R}}$, respectively. Our algorithm for $\mathcal{TC}^{\mathcal{B}}(\Delta)$ stabilizes in at most 3Δ rounds. However, we show that stabilization time cannot be bounded for the leader election problem in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ and $\mathcal{TC}^{\mathcal{R}}$. Nevertheless, we circumvent this issue by showing that our solutions are speculative in the sense that their stabilization time in $\mathcal{TC}^{\mathcal{B}}(\Delta)$ ($\subseteq \mathcal{TC}^{\mathcal{Q}}(\Delta) \subseteq \mathcal{TC}^{\mathcal{R}}$) is $O(\Delta)$ rounds.

Keywords: self-stabilization, time-varying graphs, leader election, speculation
2010 MSC: 68W15, 68M15

1. Introduction

1.1. Context

Starting from an arbitrary configuration, a *self-stabilizing algorithm* [1] makes a distributed system reach within finite time a configuration from which its behavior is correct. Essentially, self-stabilizing algorithms tolerate *transient failures*, since by definition such failures last a finite time (as opposed to crash failures, for example) and their frequency is low (as opposed to intermittent failures). Indeed, the arbitrary initial configuration can be seen as the result of a finite number of transient faults, and after those faults cease, we can expect a sufficiently large time window without any fault so that the system recovers and then exhibits a correct behavior for a long time.

Even though self-stabilization is not inherently suited to handle other failure patterns, *a.k.a.*, *intermittent* and *permanent* failures, several works show that in many cases self-stabilization can still be achieved despite such faults occur. Indeed, strong forms of self-stabilization have been proposed to tolerate permanent failures, *e.g.*, *fault-tolerant self-stabilization* [2, 3] to cope with process crashes, and *strict stabilization* [4, 5] to withstand Byzantine failures. Furthermore, several self-stabilizing algorithms [6, 7] withstand intermittent failures such as frequent lost, duplication, or reordering of messages, meaning their convergence is still effective despite such faults continue to often occur in the system. Hence, even if at the first glance guaranteeing a convergence property may seem to be contradictory with a high failure rate, the literature shows that self-stabilization may be a suitable answer even in such cases.

All these aforementioned works assume static communication networks. Nevertheless, self-stabilizing algorithms dedicated to arbitrary network topologies tolerate, up to a certain extent, some topological changes (*i.e.*, the addition or the removal of communication links or nodes). Precisely, if topological changes are eventually detected locally at involved processes and if the frequency of such events is low enough, then they can be considered as transient faults. Actually, a number of works, *e.g.*, [8, 9, 10] (*n.b.*, [8] deals with leader election), use this kind of argument to claim that they are suited for the dynamic context. Furthermore, several approaches, like *superstabilization* [11] and *gradual stabilization* [12], aims at additionally providing countermeasures to efficiently treat topological changes when they are both spatially and timely sparse. However, all these aforementioned approaches [8, 9, 11, 10, 12] become totally ineffective when the frequency of topological changes drastically increase, in other words when topological changes are intermittent rather than transient. Actually, in the intermittent case, the network dynamics should be no more considered as an anomaly but rather as an integral part of the system nature.

Clearly, many of today's networks are highly dynamic, *e.g.*, MANET (*Mobile Ad-Hoc Networks*), VANET (*Vehicular Ad-Hoc Networks*), and DTN (*Delay-Tolerant Networks*), to only quote a few. Several works aim at proposing a general graph-based model to capture the network dynamics. In [13], the network dynamics is represented as a sequence of digraphs called *evolving graphs*. In [14], the topological evolution of the network is modeled by a *Time-Varying Graph* (TVG, for short) which basically consists of a (fixed) digraph and a presence function. In the digraph, nodes represent participating processes and the edges are communication links that may appear during the lifetime of the network. Moreover, the presence function indicates whether or not a given arc of the digraph exists at a given time. TVGs are typically gathered and ordered into classes according to the temporal characteristics of edge presence they satisfy [14].

The obtained taxonomy together with possibility/impossibility results allows to compare them according to their expressive power, *i.e.*, the power with respect to the set of problems that can be solved.

In highly dynamic distributed systems, an expected property is *self-adaptiveness*, *i.e.*, the ability of a system to accommodate to sudden and frequent changes of its environment. By definition, achieving self-stabilization in highly dynamic networks is a suitable answer to self-adaptiveness. There exist other approaches for self-adaptiveness, *e.g.*, the *graceful degradation* [15] and *speculation* [16]. Graceful degradation is a best effort approach which consists in designing an algorithm that solves a given problem under the highest possible dynamics and, in the other way around, computes an approached (meaningful) solution to the problem when considering a dynamic level higher enough to make the original problem impossible to solve. *Speculation* [16] is another possible approach for self-adaptiveness. Roughly speaking, it guarantees that the system satisfies its requirements for all executions, but also exhibits significantly better performances in a subset of more probable executions. The main idea behind speculation is that worst possible scenarios are often rare (even unlikely) in practice. So, a speculative algorithm is assumed to self-adapt its performances *w.r.t.* the "quality" of the environment, *i.e.*, the more favorable the environment is, the better the complexity of the algorithm should be. Interestingly, Dubois and Guerraoui [17] have investigated speculation in self-stabilizing, yet static, systems. They illustrate this property with a self-stabilizing mutual exclusion algorithm whose stabilization time is significantly better when the execution is synchronous.

1.2. Contribution

We initiate research on self-stabilization in *highly dynamic identified message-passing systems where the dynamics is modeled using TVGs* to obtain solutions tolerating both transient faults and high dynamics. In our model, processes can only communicate through local broadcast primitives: at each round, every process can send a common message to its unknown set of current neighbors (if any).

We reformulate the definition of self-stabilization to accommodate TVGs, and investigate the self-stabilizing leader election problem. This problem is fundamental in distributed computing since it allows to synchronize and self-organize a network. Thus, leader election is a basic component in many protocols, *e.g.*, spanning tree constructions, broadcasts, and convergecasts.

We study self-stabilizing leader election in three wide classes of TVGs, respectively denoted by $\mathcal{TC}^{\mathcal{B}}(\Delta)$, $\mathcal{TC}^{\mathcal{Q}}(\Delta)$, and $\mathcal{TC}^{\mathcal{R}}$, where $\mathcal{TC}^{\mathcal{B}}(\Delta) \subseteq \mathcal{TC}^{\mathcal{Q}}(\Delta) \subseteq \mathcal{TC}^{\mathcal{R}}$: $\mathcal{TC}^{\mathcal{B}}(\Delta)$ is the class of TVGs with temporal diameter bounded by Δ [18], $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ is the class of TVGs with temporal diameter quasi-bounded by Δ (introduced here), and $\mathcal{TC}^{\mathcal{R}}$ is the class of TVGs with recurrent connectivity [14]; this latter class is the most general infinite TVG class introduced so far [14, 19].[1]

We first study conditions under which our problem can be solved. Actually, our results show that the assumption on the knowledge of the number n of processes is central. To see this, we introduce the notion of size-ambiguity, which formalizes the fact that some subsets of processes do not share enough initial knowledge on n to detect that the system is not limited to themselves. In other words, such an ambiguity comes from the fact that n is only partially known by the processes (*e.g.*, when processes only know an upper bound on n). Our results show that, despite the existence of unique process identifiers, any deterministic self-stabilizing leader election algorithm working on the class $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ or $\mathcal{TC}^{\mathcal{R}}$ cannot be size-ambiguous. Hence, to make the problem solvable in those classes, we will assume each process knows the exact value of n.

We then propose self-stabilizing leader election algorithms for the three considered classes. In more detail, we present a self-stabilizing leader election algorithm for Class $\mathcal{TC}^{\mathcal{B}}(\Delta)$ with a stabilization time of at most 3Δ rounds, assuming every process knows Δ, yet using no information on n. This in

[1]Considering finite TVGs, *i.e.*, dynamic systems whose lifetime is limited, does not really make sense in the self-stabilizing context, since commonly self-stabilizing algorithms do not terminate [20, 21].

particular shows that our necessary condition is tight. Then, we propose a self-stabilizing leader election algorithm for Class $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ assuming every process knows Δ and n. We show that, in general, stabilization time for the leader election problem cannot be bounded in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$; nevertheless we show that the algorithm is speculative since its stabilization time in $\mathcal{TC}^{\mathcal{B}}(\Delta)$ is at most 2Δ rounds. Finally, we propose a self-stabilizing leader election algorithm for Class $\mathcal{TC}^{\mathcal{R}}$, where only n is known, yet requiring unbounded local memories. Finding a self-stabilizing solution in this class was rather challenging, since there is no guarantee on message delivery timeliness at all (*n.b.*, by definition of the class, there is no bound on the temporal diameter). Again, we show that, in general, stabilization time for the leader election problem cannot be bounded in $\mathcal{TC}^{\mathcal{R}}$; yet we establish that the algorithm is speculative since its stabilization time in $\mathcal{TC}^{\mathcal{B}}(\Delta)$ is at most $\Delta + 1$ rounds.

1.3. Related Work

Ensuring convergence in highly dynamic networks regardless of the initial configuration may seem to be very challenging, even impossible in many cases [22]. However, there are a few works [23, 24, 25] that deal with this issue, yet in widely different models and assumptions than ours.

A recent work [23] deals with 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. Note that, contrary to [23], the three classes studied in the present paper never enforce the network to have a particular topology at a given time.

In [24], 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 pair 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 infinitely often reachable is infinitely often reached. We do not make such an assumption here. Actually, Cai *et al.* show that, in their model, self-stabilizing leader election is deterministically solvable if and only if the number of processes n is known, despite processes being *anonymous*. In our model, even with the knowledge of n, (deterministic) self-stabilizing leader election cannot be solved if processes are anonymous.[2] Moreover, our results show that (maybe surprisingly) even with process identifiers, the knowledge of n is necessary to solve self-stabilizing leader election in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ and $\mathcal{TC}^{\mathcal{R}}$.

Finally, Dolev *et al.* [25] 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. We should remark that techniques used here can be reengineered to implement their input black box routing protocol.

1.4. Roadmap

In Section 2, we define the computational model. In Section 3, we propose and justify our definition of self-stabilization for highly dynamic environments; we then study the impact of the knowledge of n on the solvability of the self-stabilizing leader election. In the three next sections, we present, prove, and analyze our self-stabilizing algorithms. The last section is dedicated to conclusions and perspectives.

2. Preliminaries

2.1. Time-varying Graphs

A *time-varying graph* (TVG for short) [14] is a tuple $\mathcal{G} = (V, E, \mathcal{T}, \rho)$ where V is a (static) set of nodes, E is a (static) set of arcs between pairwise nodes, $\mathcal{T}$

[2]Every static network is a very particular case of TVG, which belongs to all classes studied here, so the impossibility result of Angluin [26] still applies.

is an interval over $\mathbb{N}^*$ (the set of positive integers) called the *lifetime* of $\mathcal{G}$, and $\rho : E \times \mathcal{T} \to \{0, 1\}$ is the *presence* function that indicates whether or not a given arc exists at a given time. We denote by $o_{\mathcal{T}} = \min \mathcal{T}$ the first instant in $\mathcal{T}$.

From a global viewpoint, the evolution of $\mathcal{G}$ is described as a sequence of digraphs, called *snapshots*: the *snapshot* of $\mathcal{G}$ at time $t \in \mathcal{T}$ is the digraph $G_t = (V, \{e \in E : \rho(e, t) = 1\})$.

Let $[t, t'] \subseteq \mathcal{T}$. The *temporal subgraph* of $\mathcal{G}$ for the interval $[t, t']$, denoted by $\mathcal{G}_{[t,t']}$, is the TVG $(V, E, [t, t'], \rho')$ where ρ' is ρ restricted to $[t, t']$. Roughly speaking, $\mathcal{G}_{[t,t']}$ is itself a TVG that reproduces all the interactions present in the original TVG $\mathcal{G}$, yet for the time window $[t, t']$.

A *journey* $\mathcal{J}$ can be thought as a path over time from a source p_1 to a destination q_k, *i.e.*, $\mathcal{J}$ is a sequence $(e_1, t_1), (e_2, t_2), \ldots, (e_k, t_k)$ where $\forall i \in \{1, \ldots, k\}$, $e_i = (p_i, q_i) \in E$ satisfies $\rho(e_i, t_i) = 1$ and $i < k \Rightarrow q_i = p_{i+1} \wedge t_i < t_{i+1}$. We respectively denote by $departure(\mathcal{J})$ and $arrival(\mathcal{J})$ the *starting time* t_1 and the *arrival time* t_k of $\mathcal{J}$. A *journey from p to q* is a journey whose source is p and destination is q. Let $\mathcal{J}(p, q)$ be the set of journeys in $\mathcal{G}$ from p to q. Let $\rightsquigarrow$ be the binary relation over V such that $p \rightsquigarrow q$ if $p = q$ or there exists a journey from p to q in $\mathcal{G}$.

The *temporal length* of a journey $\mathcal{J}$ is equal to $arrival(\mathcal{J}) - departure(\mathcal{J}) + 1$. Let $t \geq o_{\mathcal{T}} - 1$. By extension, we define the *temporal distance* from p to q at t, denoted by $\hat{d}_{p,t}(q)$, as follows: $\hat{d}_{p,t}(q) = 0$, if $p = q$, $\hat{d}_{p,t}(q) = \min\{arrival(\mathcal{J}) - t : \mathcal{J} \in \mathcal{J}(p, q) \wedge departure(\mathcal{J}) > t\}$ otherwise (by convention, we let $\min \emptyset = +\infty$). Roughly speaking, the temporal distance from p to q at time t gives the minimum timespan for p to reach q after t. The *temporal diameter* at t is the maximum temporal distance between any two nodes at t.

We define $ITVG(\mathcal{G})$ to be the predicate that holds if $\mathcal{T}$ is a right-open interval, in which case $\mathcal{G}$ is said to be an *infinite* TVG; otherwise $\mathcal{G}$ is called a *finite* TVG.

2.2. TVG Classes

Let $\mathcal{G} = (V, E, \mathcal{T}, \rho)$ be a TVG. We consider the following four TVG classes.

Class $\mathcal{TC}$ (Connectivity over Time), also denoted by $\mathcal{C}_3$ in [14]: every node can reach all the others at least once through a journey. Formally, $\mathcal{G} \in \mathcal{TC}$ if $\forall p, q \in V, p \rightsquigarrow q$.

Class $\mathcal{TC}^{\mathcal{R}}$ (Recurrent Connectivity), denoted by $\mathcal{C}_5$ in [14]: at any point in time, every node can reach all the others through a journey. Formally, $\mathcal{G} \in \mathcal{TC}^{\mathcal{R}}$ if $ITVG(\mathcal{G}) \wedge \forall t \in \mathcal{T}$, $\mathcal{G}_{[t,+\infty)} \in \mathcal{TC}$.

Class $\mathcal{TC}^{\mathcal{B}}(\Delta)$ with $\Delta \in \mathbb{N}^*$ (Bounded Temporal Diameter), denoted by $\mathcal{TC}(\Delta)$ in [18]: at any point in time, every node is at temporal distance at most Δ from all other nodes, *i.e.*, the temporal diameter is bounded by Δ. Formally, $\mathcal{G} \in \mathcal{TC}^{\mathcal{B}}(\Delta)$ if $ITVG(\mathcal{G}) \wedge \forall t \in \mathcal{T}, \mathcal{G}_{[t,t+\Delta)} \in \mathcal{TC}$.

Class $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ with $\Delta \in \mathbb{N}^*$ (Quasi Bounded Temporal Diameter): every node is infinitely often at temporal distance at most Δ from each other node. Formally, $\mathcal{G} \in \mathcal{TC}^{\mathcal{Q}}(\Delta)$ if $ITVG(\mathcal{G}) \wedge \forall p, q \in V, \forall t \in \mathcal{T}, \exists t' \geq t-1, \hat{d}_{p,t'}(q) \leq \Delta$.

Notice that, $\forall \Delta \in \mathbb{N}^*$, $\mathcal{TC}^{\mathcal{B}}(\Delta) \subseteq \mathcal{TC}^{\mathcal{Q}}(\Delta) \subseteq \mathcal{TC}^{\mathcal{R}} \subseteq \mathcal{TC}$, by definition. Furthermore, we say that a TVG class $\mathcal{R}$ is *recurring* if $\mathcal{R}$ only contains infinite TVGs and, for every $\mathcal{G} \in \mathcal{R}$ and every $t \geq o_{\mathcal{T}}$, $\mathcal{G}_{[t,+\infty)} \in \mathcal{R}$. In other words, every recurring TVG class is suffix-closed. The three classes we will consider hereafter (*i.e.*, $\mathcal{TC}^{\mathcal{R}}$, $\mathcal{TC}^{\mathcal{B}}(\Delta)$, $\mathcal{TC}^{\mathcal{Q}}(\Delta)$) are actually recurring.

2.3. Computational Model

We consider the computational model defined in [27, 28]. We assume a *distributed system* made of a set of *n processes*, denoted by V. Each process has a local memory, a local sequential and deterministic algorithm, and message exchange capabilities. By deterministic, we mean that each process step dictated by the algorithm is uniquely determined by the local memory of the process and the messages it received. We assume that each process p holds a unique identifier (ID for short) in its local memory. The identifier of p is denoted by $id(p)$ and taken in an arbitrary domain $IDSET$ totally ordered by $<$. We assume that each identifier is stored using B bits. In the sequel, we denote by ℓ the process of minimum identifier. Processes are assumed to communicate by message-passing through an interconnected network that evolves over the time. The topology of the network is then conveniently modeled by an infinite TVG $\mathcal{G} = (V, E, \mathcal{T}, \rho)$. Processes execute their local algorithms in *synchronous rounds*. For every $i > 0$, the communication network at Round i is defined by $G_{o_\mathcal{T}+i-1}$, *i.e.*, the snapshot of $\mathcal{G}$ after $i-1$ instants elapse from the initial time $o_\mathcal{T}$. So, $\forall p \in V$, we denote by $\mathcal{N}(p)^i = \{q \in V : \rho((p,q), o_\mathcal{T} + i - 1) = 1\}$, the set of p's neighbors at Round i. $\mathcal{N}(p)^i$ is assumed to be unknown by process p, whatever the value of i is.

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 $\mathcal{A}(p)$. We denote by $\mathcal{S}^V_\mathcal{A}(p)$ the non-empty set of p's possible local states in $\mathcal{A}$. Some variables may be constants in which case their values are predefined. A *configuration* of $\mathcal{A}$ for V is a vector of n components $(s_1, s_2, \dots, s_n)$, where s_1 to s_n represent the states of the processes in V. Let $\mathcal{C}^V_\mathcal{A}$ be the set of all possible configurations of $\mathcal{A}$ for V.

Let γ_0 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-1} to some configuration γ_i, where γ_{i-1} (resp. γ_i) is referred to as the configuration *at the beginning of Round i* (resp. *at the 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-1} using the primitive `SEND()`,
2. using Primitive `RECEIVE()`, p receives all messages sent by processes in $\mathcal{N}(p)^i$, and
3. p computes its state in γ_i.

An *execution* of a distributed algorithm $\mathcal{A}$ in the TVG $\mathcal{G} = (V, E, \mathcal{T}, \rho)$ is an infinite sequence of configurations $\gamma_0, \gamma_1, \dots$ of $\mathcal{A}$ for V such that $\forall i > 0$, γ_i is obtained by executing a synchronous round of $\mathcal{A}$ on γ_{i-1} based on the communication network at Round i, *i.e.*, the snapshot $G_{o_\mathcal{T}+i-1}$.

3. Self-stabilization in Highly Dynamic Environments

3.1. Definition

In the following, we define a *specification* as a predicate over configuration sequences.

Self-stabilization has been originally defined for static networks. In the reference book of Dolev [21], self-stabilization is defined as follows. An algorithm is *self-stabilizing for a specification SP* if there exists a set of so-called *legitimate* configurations satisfying the following two properties: (1) every execution of the algorithm in the considered system eventually reaches a legitimate configuration (*Convergence*); and (2) every possible execution suffix starting from a legitimate configuration satisfies SP (*Correctness*). Below, we accommodate this definition to highly dynamic environments.

Definition 1 (Self-stabilization). *An algorithm $\mathcal{A}$ is* self-stabilizing *for the specification SP on a class $\mathcal{I}$ of infinite TVGs 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:*

1. *for every $\mathcal{G} \in \mathcal{I}$ 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 legitimate configuration $\gamma' \in \mathcal{L}$ (*Convergence*), and*
2. *for every $\mathcal{G} \in \mathcal{I}$ with set of processes V, every $t \geq o_{\mathcal{T}}$, every legitimate configuration $\gamma \in \mathcal{L}$, and every execution e in $\mathcal{G}_{[t,+\infty)}$ 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.*

Remark 1. *In the case of a recurring class of TVG, the definition of self-stabilization for an algorithm $\mathcal{A}$ and a specification SP can be slightly simplified. Indeed, the correctness property can be equivalently rewritten as follows: given a* *set of processes V and a set of configurations $\mathcal{L}$ on V, for every $\mathcal{G} \in \mathcal{I}$ with set of processes V, every legitimate configuration $\gamma \in \mathcal{L}$, and every execution e in $\mathcal{G}$ starting from γ, $SP(e)$ holds (*Recurring-Correctness*).*

It is worth noting that Definition 1, as the one given in the reference book of Dolev [21], does not include the notion of *closure*: intuitively, a set of configurations $\mathcal{S}$ is *closed* if every step of the algorithm starting in a configuration of $\mathcal{S}$ leads to a configuration of $\mathcal{S}$; see Definition 2 for a formal definition. Now, when dealing with high-level models (such as the atomic-state model), closure is most of the time present in definitions of self-stabilization. However, in the more practical message-passing model, closure is usually simply given up; see, *e.g.*, [29, 30, 31]. Even if this absence is never motivated, this may be explained by the lack of functional significance of the closure property as compared to the convergence and correctness properties. Closure is rather a nice property that often helps to write elegant, and so simpler, proofs. Moreover, closure may be sometimes too restrictive, as we will show in Theorem 1 for example. Below, we reformulate closure in the context of TVGs.

Definition 2 (Closure). *Let $\mathcal{A}$ be a distributed algorithm, $\mathcal{I}$ be an infinite TVG class, V be a set of processes, and $\mathcal{S}$ be a subset of configurations of $\mathcal{A}$ for V. $\mathcal{S}$ is* closed *in $\mathcal{I}$ if for every $\mathcal{G} \in \mathcal{I}$ with set of processes V, every $t \geq o_{\mathcal{T}}$, and every configuration $\gamma \in \mathcal{S}$, every execution of $\mathcal{A}$ in $\mathcal{G}_{[t,+\infty)}$ starting from γ only contains configurations of $\mathcal{S}$.*

Remark 2. *Again, when the considered class of TVGs is recurring, the definition of closure can be slightly simplified. If $\mathcal{A}$ is a distributed algorithm, $\mathcal{R}$ is a recurring TVG class, V is a set of processes, and $\mathcal{S}$ is a subset of configurations of $\mathcal{A}$ for V, then $\mathcal{S}$ is* closed *in $\mathcal{R}$ if for every $\mathcal{G} \in \mathcal{R}$ with set of processes V and every configuration $\gamma \in \mathcal{S}$, every execution of $\mathcal{A}$ in $\mathcal{G}$ starting from γ only contains configurations of $\mathcal{S}$.*

3.2. Self-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 in an output variable, denoted here by $lid(p)$.

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 self-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 (legitimate) configuration γ from which a unique process is forever adopted as leader by all processes whatever be the execution suffix, *i.e.*, $\exists p \in V$ such that $\forall q \in V, lid(q) = id(p)$ forever in all possible execution suffixes starting from γ. Hence, the leader election specification SP_{LE} can be formulated as follows: a sequence of configurations $\gamma_0, \gamma_1, \ldots$ satisfies SP_{LE} if and only if $\exists p \in V$ such that $\forall i \geq 0$, $\forall q \in V$, the value of $lid(q)$ in configuration γ_i is $id(p)$.

In the sequel, we say that an algorithm is a *self-stabilizing leader election algorithm* for the class of infinite TVG $\mathcal{I}$ if it is self-stabilizing for SP_{LE} on $\mathcal{I}$.

3.3. Knowledge of n and Closure in $\mathcal{TC}^{\mathcal{B}}(\Delta)$

We now advocate that closure of legitimate configurations may be cumbersome in $\mathcal{TC}^{\mathcal{B}}(\Delta)$ since to achieve it, any (deterministic) self-stabilizing leader election algorithm somehow requires the exact knowledge of n (the number of processes), *i.e.*, even partial knowledge such as an upper bound on n is not sufficient; see Theorem 1. To that goal, we need to first define what we mean by *not exactly knowing n.*

When an algorithm $\mathcal{A}$ uses the number of its processes, this means that this information is given as an input in the local state of each process. So, the definition of the set of possible local states of each process is adjusted according to the size of the system it belongs to. Conversely, if an algorithm $\mathcal{A}$ does not know its exact size, this means that there are sizes of systems that cannot be distinguished by part of its processes using their local inputs (and so their possible local states). More precisely, for a given set of processes V executing $\mathcal{A}$, there should exist a size $k < |V|$ for which the processes of any k-subset U of V do not share enough initial information to distinguish whether the system is made of the process-set V or U. Below, we formalize this intuitive idea by the notion of *size-ambiguity.*

Definition 3 (Size-Ambiguity). *Let V be a set of processes and $k \in \mathbb{N}$. A distributed algorithm $\mathcal{A}$ (in particular) defined for V is (k, V)-*ambiguous *if $0 < k < |V|$ and for every $U \subset V$ such that $|U| = k$, $\mathcal{C}_{\mathcal{A}}^{U}$ is well-defined and for every $p \in U$, $\mathcal{S}_{\mathcal{A}}^{U}(p) = \mathcal{S}_{\mathcal{A}}^{V}(p)$. We simply say that $\mathcal{A}$ is* size-ambiguous *if there exists V and k such that $\mathcal{A}$ is (k, V)-ambiguous.*

Consider now a few examples. First, if each process has a constant input whose value is the number n of processes in the system (*i.e.*, each process "exactly knows n"), then from our definition, the algorithm is not size-ambiguous since, in this case, the set of possible local states of any process differs from one size of system to another, at least because of the input storing n. Conversely, if the processes do not know the exact number of processes but its parity, then we can choose any set V of at least three processes and any positive value $k < |V|$ with same parity as $|V|$: for every subset U of V such that $|U| = k$, the constant input giving the parity will be the same at each process of U whether running its algorithm in a TVG with process-set V or U. Consequently, every process $p \in U$ will have the same set of possible local states in both TVGs; hence the size-ambiguity. Similarly, an algorithm is size-ambiguous if each process p only knows an upper bound $N_p \geq 2$ on the number of processes in the TVG (*n.b.*, processes may not know the same bound) since the property can be achieved with any set V of at least two processes and any value k such that $0 < k < |V|$.

Theorem 1. *Let $\mathcal{A}$ be a deterministic self-stabilizing leader election algorithm for $\mathcal{TC}^{\mathcal{B}}(\Delta)$ with $\Delta \geq 2$, V be a set of processes, $\mathcal{L}$ be a set of legitimate* *configurations of $\mathcal{A}$ for V, and $k \in \mathbb{N}$. $\mathcal{L}$ is not closed in $\mathcal{TC}^{\mathcal{B}}(\Delta)$ if $\mathcal{A}$ is (k,V)-ambiguous.*

Proof: Let $n = |V|$ and $V = \{p_0, \dots, p_{n-1}\}$. Assume, by the contradiction, that $\mathcal{A}$ is (k,V)-ambiguous, but $\mathcal{L}$ is closed in $\mathcal{TC}^{\mathcal{B}}(\Delta)$. Let $\mathcal{G} = (V, E, \mathcal{T}, \rho)$ be an infinite TVG such that

1. $E = \{(p_i, p_j) : p_i, p_j \in V \wedge i \neq j\}$
2. $\forall t \geq o_{\mathcal{T}}$, $\forall (p_i, p_j) \in E$, $\rho((p_i, p_j), t) = 1$ if and only if either t is odd, or $i \notin \{\frac{t}{2} \bmod n, \dots, (\frac{t}{2} + n - k - 1) \bmod n\}$ and $j \notin \{\frac{t}{2} \bmod n, \dots, (\frac{t}{2} + n - k - 1) \bmod n\}$.

Notice first that, $\forall t \geq o_{\mathcal{T}}$, the snapshot G_t of $\mathcal{G}$ is fully connected when t is odd. Consequently, $\mathcal{G}$ belongs to $\mathcal{TC}^{\mathcal{B}}(\Delta)$, with $\Delta \geq 2$. Then, by definition, we have:

Claim 1: *For every $x \in \{0, \dots, n-1\}$ and every $i \geq 0$, in the snapshot $G_{t_{x,i}}$ of $\mathcal{G}$ at time $t_{x,i} = 2((i + o_{\mathcal{T}}).n + x)$, the set $V \setminus \{p_x, \dots, p_{(x+n-k-1) \bmod n}\}$ is fully connected and the set $\{p_x, \dots, p_{(x+n-k-1) \bmod n}\}$ is independent.*

Let $\gamma \in \mathcal{L}$ and $p_\ell \in V$ be the elected process in γ. $\forall i \geq 0$, we inductively define Configuration γ^i as follows. $\gamma^0 = \gamma$. $\forall i > 0$, γ^i is the configuration at the end of the first round of the execution of $\mathcal{A}$ in $\mathcal{G}_{[t_{\ell,i}, +\infty)}$ starting from γ^{i-1}. Since $\mathcal{TC}^{\mathcal{B}}(\Delta)$ is recurring, we can use the closure of $\mathcal{L}$ to show by induction Claim 2 below (*n.b.*, Claim 2 is the only result of the proof where closure of $\mathcal{L}$ is used).

Claim 2: *$\forall i \geq 0$, γ^i is legitimate and $\forall p_j \in V$, $lid(p_j) = p_\ell$ in γ^i.*

Proof of the claim: By induction on i. The induction is trivial for $i = 0$. Consider now the case where $i > 0$. By induction hypothesis, γ^{i-1} is legitimate and $\forall p_j \in V$, $lid(p_j) = p_\ell$. By definition, $\mathcal{G}_{[t_{\ell,i}, +\infty)} \in \mathcal{TC}^{\mathcal{B}}(\Delta)$ since $\mathcal{TC}^{\mathcal{B}}(\Delta)$ is recurring. So, since $\mathcal{L}$ is closed in $\mathcal{TC}^{\mathcal{B}}(\Delta)$, γ^i is legitimate. Moreover, by applying the correctness property to the execution of $\mathcal{A}$ in $\mathcal{G}_{[t_{\ell,i}, +\infty)}$ starting from γ^{i-1}, we deduce that $\forall p_j \in V$, $lid(p_j) = p_\ell$ in γ^i; see the definition of SP_{LE}.

Let $V^- = V \setminus \{p_\ell, \ldots, p_{(\ell+n-k-1) \bmod n}\}$ and $E^- = \{(p_i, p_j) : p_i, p_j \in V^- \wedge i \neq j\}$. Let $\mathcal{G}^- = (V^-, E^-, \mathcal{T}, \rho^-)$ be the infinite TVG having k processes such that $\forall t \geq o_\mathcal{T}$, $\forall (p_i, p_j) \in E^-$, we have $\rho((p_i, p_j), t) = 1$. In other word, $\mathcal{G}^-$ is a static fully connected network. Consequently, $\mathcal{G}^-$ in particular belongs to $\mathcal{TC}^\mathcal{B}(\Delta)$ with $\Delta \geq 2$ (actually, it also belongs to $\mathcal{TC}^\mathcal{B}(\Delta)$ with $\Delta = 1$). Let γ_0^- be the configuration of $\mathcal{A}$ for V^- where each process has the same state as in the configuration γ^0 (such a configuration exists by definition of (k,V)-ambiguity). We now consider the execution $e = \gamma_0^-, \ldots, \gamma_i^-, \ldots$ of $\mathcal{A}$ in $\mathcal{G}^-$ starting from the configuration γ_0^-.

Claim 3: $\forall i \geq 0$, *the state of each process in* V^- *in* γ_i^- *is the same as in* γ^i.

Proof of Claim 3: By induction on i. The base case $i = 0$ is trivial. Consider now the case where $i > 0$. γ_{i-1}^- is the configuration at the beginning of Round i in e. By induction hypothesis, the state of each process in V^- in γ_{i-1}^- is the same as in γ^{i-1}. By Claim 1, each process of V^- has the same neighborhood in G_{i-1}^- and in $G_{t_{\ell,i}}$. Hence, during Round i they receive the same set of messages as during the first round of $\mathcal{A}$ in $\mathcal{G}_{[t_{\ell,i},+\infty)}$ starting from γ^{i-1}. So, since $\mathcal{A}$ is deterministic, each process of V^- behaves exactly as in the first round of $\mathcal{A}$ in $\mathcal{G}_{[t_{\ell,i},+\infty)}$ starting from γ^{i-1}. Thus, in γ_i^- at the end of Round i, the state of each process of V^- is the same as in γ^i.

By Claims 2 and 3, for every process p_j in V^-, in every configuration γ_i^-, we have $lid(p_j) = p_\ell \notin V^-$, *i.e.*, $lid(p_j)$ is a fake ID (for V^-). Hence, no suffix of e satisfies SP_{LE}. As a consequence, $\mathcal{A}$ is not a self-stabilizing leader election algorithm for $\mathcal{TC}^\mathcal{B}(\Delta)$ (with $\Delta \geq 2$), a contradiction. $\square$

Remark 3. *The condition* $\Delta \geq 2$ *is necessary in Theorem 1, indeed if* $\Delta = 1$*, there is a trivial deterministic self-stabilizing leader election algorithm for* $\mathcal{TC}^\mathcal{B}(\Delta)$ *that does not need information on* n *and has a closed set of legitimate configurations: it simply consists of all processes sending their own IDs at each round; since* $\Delta = 1$*, all processes learn the exact set of all IDs present in the network at each round and just have to choose,* e.g.*, the smallest one,* $id(\ell)$*.*

The legitimate configurations are then all configurations where every process p satisfies $lid(p) = id(\ell)$.

According to Theorem 1, the set of legitimate configurations of our solution for $\mathcal{TC}^{\mathcal{B}}(\Delta)$ (Algorithm 1) is not closed, since by making no assumption on n, this algorithm is size-ambiguous. The contrapositive of Theorem 1 is given in Corollary 1. This latter justify the need of the exact knowledge of the number of processes to obtain a closed set of legitimate configurations in a deterministic self-stabilizing leader election algorithm for $\mathcal{TC}^{\mathcal{B}}(\Delta)$, with $\Delta \geq 2$.

Corollary 1. *Let $\mathcal{A}$ be a deterministic self-stabilizing leader election algorithm for $\mathcal{TC}^{\mathcal{B}}(\Delta)$ (with $\Delta \geq 2$), V be a set of processes, and $\mathcal{L}$ be a set of legitimate configurations of $\mathcal{A}$ for V. If $\mathcal{L}$ is closed in $\mathcal{TC}^{\mathcal{B}}(\Delta)$, then $\mathcal{A}$ should not be size-ambiguous.*

Remark 4. *The scheme used in the proof of Theorem 1 can be adapted to handle other problems consisting in computing a constant output whose value depends on the set of processes. For example, one can show that no deterministic self-stabilizing size-ambiguous algorithm for $\mathcal{TC}^{\mathcal{B}}(\Delta)$ can both compute the exact number of processes and achieve the closure of its legitimate configurations.*

3.4. Knowledge of n and Closure in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$

We now show that every execution of a self-stabilizing algorithm for a so-called recurring specification (see Definition 4 below) in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ necessarily converges to a closed set of legitimate configurations; see Theorem 2. Consequently, no deterministic self-stabilization leader election algorithm for $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ can be size-ambiguous since SP_{LE} is recurring (Theorem 3 and Corollary 3); justifying why algorithms in Sections 5 and 6 assume the exact knowledge of n.

Informally, a specification is recurring if whenever an execution satisfies it, all its suffixes also satisfy it.

Definition 4. *[Recurring Specification] We say that a specification SP is* recurring *if for every sequence of configurations $\gamma_0, \gamma_1, \ldots$, $SP(\gamma_0, \gamma_1, \ldots) \Rightarrow (\forall i \geq 0$, $SP(\gamma_i, \gamma_{i+1}, \ldots))$.*

SP_{LE} (as most of specifications used in self-stabilization) is a recurring specification.

Definition 5 (Sequential Composition). *Let $\mathcal{G} = (V, E, \mathcal{T}, \rho)$ be an infinite TVG and $\mathcal{G}' = (V', E', [a, b], \rho')$ be a finite TVG. The* sequential composition *of $\mathcal{G}'$ and $\mathcal{G}$, denoted by $\mathcal{G}' \rhd \mathcal{G}$, is the infinite TVG $\mathcal{G}'' = (V'', E'', \mathcal{T}'', \rho'')$ such that $V'' = V \cup V'$, $E'' = E \cup E'$, $\mathcal{T}'' = [a, +\infty)$, and $\forall e \in E''$,*

- *$\forall t \in [a, b]$, $\rho''(e, t) = 1$ if and only if $e \in E' \wedge \rho'(e, t) = 1$, and*
- *$\forall t > b$, $\rho''(e, t) = 1$ if and only if $e \in E \wedge \rho(e, o_{\mathcal{T}} + t - b - 1) = 1$.*

Property 1. *Let $\mathcal{G} = (V, E, \mathcal{T}, \rho) \in \mathcal{TC}^{\mathcal{Q}}(\Delta)$ and $\mathcal{G}' = (V', E', \mathcal{T}', \rho')$ be a finite TVG. If $V' \subseteq V$, $\mathcal{G}' \rhd \mathcal{G} \in \mathcal{TC}^{\mathcal{Q}}(\Delta)$.*

Theorem 2. *Let SP be a recurring specification, $\mathcal{A}$ be a self-stabilizing algorithm for SP on $\mathcal{TC}^{\mathcal{Q}}(\Delta)$, and V be a set of processes. There exists a set of legitimate configurations of $\mathcal{A}$ for V which is closed in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$.*

Proof: Assume, by the contradiction, that every set of legitimate configurations of $\mathcal{A}$ for V is not closed in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$. Let $\mathcal{L}$ be the set of legitimate configurations of $\mathcal{A}$ for V defined as follows: for every configuration γ of $\mathcal{A}$ for V, $\gamma \in \mathcal{L}$ if and only if for every $\mathcal{G} \in \mathcal{TC}^{\mathcal{Q}}(\Delta)$ with set of processes V and every execution e of $\mathcal{A}$ in $\mathcal{G}$ starting from γ, $SP(e)$ holds. Since $\mathcal{L}$ is not closed in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$, there exists $\gamma_0 \in \mathcal{L}$, $\mathcal{G} = (V, E, \mathcal{T}, \rho) \in \mathcal{TC}^{\mathcal{Q}}(\Delta)$ with V as set of processes, and an execution $\gamma_0, \ldots, \gamma_i, \ldots$ in $\mathcal{G}$ starting from γ_0 which contains a configuration $\gamma_i \notin \mathcal{L}$. By definition of $\mathcal{L}$, there exists $\mathcal{G}' \in \mathcal{TC}^{\mathcal{Q}}(\Delta)$ with set of processes V and an execution e' in $\mathcal{G}'$ starting from γ_i such that $\neg SP(e')$ (otherwise γ_i should be in $\mathcal{L}$). Now, $\mathcal{G}_{[o_{\mathcal{T}}, o_{\mathcal{T}}+i-1]} \rhd \mathcal{G}' \in \mathcal{TC}^{\mathcal{Q}}(\Delta)$, by Property 1. Consequently, $\gamma_0, \ldots, \gamma_{i-1}, e'$ is an execution of $\mathcal{A}$ in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ that starts from γ_0 and violates SP since $\neg SP(e')$ and SP is recurring. By the correctness property of the self-stabilizing definition (see Remark 1), γ_0 cannot be a legitimate configuration, a contradiction. □

Corollary 2. *Let $\mathcal{A}$ be any self-stabilizing leader election algorithm for $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ and V be a set of processes. There exists a set of legitimate configurations of $\mathcal{A}$ for V which is closed in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$.*

Since $\mathcal{TC}^{\mathcal{B}}(\Delta) \subseteq \mathcal{TC}^{\mathcal{Q}}(\Delta)$, from Corollaries 1 and 2, we deduce Theorem 3 and Corollary 3:

Theorem 3. *No deterministic self-stabilizing leader election algorithm for $\mathcal{TC}^{\mathcal{Q}}(\Delta)$, with $\Delta \geq 2$, can be size-ambiguous.*

Corollary 3. *No deterministic self-stabilizing leader election algorithm for $\mathcal{TC}^{\mathcal{R}}$ can be size-ambiguous.*

Remark 5. *Like in Remark 4, one can show, for example, that no deterministic self-stabilizing size-ambiguous algorithm for $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ (resp. $\mathcal{TC}^{\mathcal{R}}$) can compute the exact number of processes.*

4. Class $\mathcal{TC}^{\mathcal{B}}(\Delta)$ with Δ known

The code of our self-stabilizing algorithm for $\mathcal{TC}^{\mathcal{B}}(\Delta)$ is given in Algorithm 1.

4.1. Overview of Algorithm 1

Each process p maintains two variables: the output $lid(p)$ will eventually contain the ID of the leader; $ttl(p)$ represents the *degree of suspicion* of p in $lid(p)$ and allows to eliminate messages containing fake IDs. At each round, if p receives some messages, the value of $ttl(p)$ is updated according to the received messages; otherwise, $ttl(p)$ is incremented if $lid(p) \neq id(p)$. The value of $ttl(p)$ can increase up to $2\Delta - 1$. Process p never increments $ttl(p)$ from $2\Delta - 1$ to 2Δ; instead it locally *resets* and declares itself as the leader: $lid(p) := id(p)$ and $ttl(p) := 0$; see Lines u1-u3.

At each round i, p first sends its leader ID together with its degree of suspicion; see Line 2. Then, the behavior of p depends on whether or not it receives some messages in the current round.

Assume first that p receives no message. In this case, p increments $ttl(p)$ if $lid(p) \neq id(p)$; see Line 5.

Assume now that p receives some messages. First, p selects the received message $\langle lid, ttl\rangle$ which is minimum according to the lexicographic order (*i.e.*, the message the lowest ID and with the lowest ttl to break ties); see Line 7. Then, if lid is smaller than $lid(p)$, p adopts lid as leader; see Lines 8-9. If $lid = lid(p)$, it updates the $ttl(p)$ by taking the smallest value between $ttl(p)$ and ttl (in this way, p may decrease its suspicion in $lid(p)$); see Line 11-12. Finally, in all cases, $ttl(p)$ is incremented if $lid(p) \neq id(p)$; see Lines 10, 12, and 14.

Recall that if an increment makes $ttl(p)$ bypass 2Δ, p systematically resets; see Function $updateTTL(v)$, Lines u1-u3.

Finally, if $lid(p) \geq id(p)$, p systematically resets; see Line 15-17. So, if p believes to be the leader at the end of Round i (*i.e.*, $lid(p) = id(p)$), then it sends its own ID together with a degree of suspicion 0 at the beginning of the next round, $i + 1$.

The reset mechanism allows to remove all fake IDs within at most 2Δ rounds. From that time, the lower ID process, ℓ, satisfies $(lid(\ell), ttl(\ell)) = (id(\ell), 0)$ forever (Corollary 4). So, after 2Δ rounds, ℓ sends $\langle id(\ell), 0\rangle$ at each round and all processes will receive messages $\langle id(\ell), d\rangle$, with $d \leq \Delta < 2\Delta$ (since $\Delta \in \mathbb{N}^*$), at least every Δ rounds since the temporal diameter is upper bounded by Δ. Thus, within at most Δ additional rounds, they will all adopt ℓ as leader and never more reset, ensuring that ℓ will remain the leader forever (Lemma 3). Hence, Algorithm 1 is a self-stabilizing leader election for $\mathcal{TC}^{\mathcal{B}}(\Delta)$ and its stabilization time is at most 3Δ rounds (Corollary 5).

4.2. Self-stabilization and Complexity

First, by definition of the algorithm, the next remark follows.

Remark 6. *Since the end of the first round, $\forall p \in V$, we have $lid(p) \leq id(p) \wedge (lid(p) = id(p) \Rightarrow ttl(p) = 0)$.*

Algorithm 1: Self-stabilizing leader election for $\mathcal{TC}^{\mathcal{B}}(\Delta)$, for each process p.

```
Inputs:
  $\Delta \in \mathbb{N}^*$            :  upper bound on the temporal diameter
  $id(p) \in IDSET$    :  ID of $p$
Local Variables:
  $lid(p) \in IDSET$              :  ID of the leader
  $ttl(p) \in \{0, \ldots, 2\Delta - 1\}$   :  degree of suspicion in $lid(p)$

Macros:
updateTTL($v$):
u1:    if $v \geq 2\Delta$ then // Reset
u2:        $lid(p) := id(p)$
u3:        $ttl(p) := 0$
u4:    else if $lid(p) \neq id(p)$ then  $ttl(p) := v$

1:  Repeat Forever
2:      SEND($\langle lid(p), ttl(p)\rangle$)
3:      mailbox := RECEIVE()
4:      if $mailbox = \emptyset$ then
5:          $updateTTL(ttl(p) + 1)$
6:      else
7:          $\langle lid, ttl\rangle := \min\{\text{messages in mailbox}\}$
8:          if $lid < lid(p)$ then
9:              $lid(p) := lid$
10:             $updateTTL(ttl + 1)$
11:         else if $lid = lid(p)$ then
12:             $updateTTL(\min(ttl(p), ttl) + 1)$
13:         else
14:             $updateTTL(ttl(p) + 1)$
15:     if $lid(p) \geq id(p)$ then // Reset
16:         $lid(p) := id(p)$
17:         $ttl(p) := 0$
```

Lemma 1. *Let f be a fake ID. For every $i \geq 1$, at the beginning of Round i, $\forall p \in V, lid(p) = f \Rightarrow ttl(p) \geq i - 1$.*

Proof: By induction on i: the base case, $i = 1$ is trivial since by definition

$ttl(p) \geq 0$. For the induction step, if $i > 1$, by induction hypothesis, $\forall p \in V, lid(p) = f \Rightarrow ttl(p) \geq i - 2$ at the beginning of Round $i - 1$. Notice that a process p can only change the value of $lid(p)$ to f if p receives a message containing f. Moreover, from the code of the algorithm, we know that for every process p, if $ttl(p) = 0$ at the beginning of Round i, then $lid(p) = id(p) \neq f$. So, *(*) if $lid(p) = f$ at beginning of Round i, then p did not reset during Round $i - 1$.*

Let $p \in V$ such that $lid(p) = f$ at beginning of Round i. There are two cases to consider.

1. If $lid(p) = f$ at the beginning of the Round $i-1$, then $ttl(p) \geq i-2$ at the beginning of the Round $i - 1$ and either p increments the value of $ttl(p)$ during Round $i - 1$ (Line 5, 12, or 14) or p sets $ttl(p)$ to $t + 1$ such that p received a message $m = \langle f, t \rangle$ from a neighbor q at Round $i - 1$ (Line 12). In the latter case, $lid(q) = f$ and so $ttl(q) = t \geq i - 2$ at the beginning of Round $i - 1$. Hence, in both cases, $ttl(p) \geq i - 1$ at the beginning of Round i by *(*)*.
2. If $lid(p) \neq f$ at the beginning of Round $i-1$, p receives a message $m = \langle f, t \rangle$ from some neighbor q and p sets $ttl(p)$ to $t+1$ during Round $i-1$ (Line 10). Now, $lid(q) = f$ and so $ttl(q) = t \geq i - 2$ at the beginning of Round $i - 1$. Thus, $ttl(p) \geq i - 1$ at the beginning of Round i by *(*)*.

□

Lemma 1 implies that for every $i > 0$ and every fake ID f, $\forall p \in V, lid(p) = f \Rightarrow ttl(p) \geq i$ at the *end* of Round i. We define a *quasi-legitimate* configuration of Algorithm 1 as any configuration where $lid(\ell) = id(\ell)$ and $ttl(\ell) = 0$ and there is no fake ID in the system (*i.e.*, $\forall p \in V$, $lid(p)$ is not a fake ID). So, from Lemma 1 and thanks to the reset mechanism of Algorithm 1, we deduce the following corollary.

Corollary 4. *At the end of Round 2Δ, the configuration is quasi-legitimate.*

Proof: By Lemma 1 and since the maximum value of *ttl* is $2\Delta - 1$, we have

$\forall p \in V$, $lid(p)$ is not a fake ID at the beginning of Round $2\Delta + 1$ and so at the end of Round 2Δ. Moreover, by definition, $id(\ell)$ is the smallest non-fake ID. So, $\forall p \in V$, $lid(p) \geq id(\ell)$ at the end of Round 2Δ. This is in particular true for process ℓ: $lid(l) \geq id(\ell)$ at the end of Round 2Δ. By Remark 6, we conclude that $lid(\ell) = id(\ell)$ and $ttl(\ell) = 0$ at the end of Round 2Δ (*n.b.*, $2\Delta > 1$ since $\Delta \in \mathbb{N}^*$). □

The proof of the next lemma consists in showing that for every set of processes V, the set of quasi-legitimate configurations of Algorithm 1 for V is closed in $\mathcal{TC}^{\mathcal{B}}(\Delta)$.

Lemma 2. *Let e be an execution of Algorithm 1 in an arbitrary TVG that starts from a quasi-legitimate configuration. The configuration reached at the end of every round of e is quasi-legitimate.*

Proof: Consider any step from γ to γ' such that γ is quasi-legitimate. First, since γ contains no fake ID, no message containing a fake ID can be sent in the step from γ to γ', and γ' contains no fake ID too. Moreover, $id(\ell)$ is the smallest non-fake ID. So, $\forall p \in V$, $lid(p) \geq id(\ell)$ in γ'. Finally, by Remark 6, we conclude that $lid(\ell) = id(\ell)$ and $ttl(\ell) = 0$ in γ'. Hence, γ' is quasi-legitimate. □

A process p has a *legitimate state* if and only if $lid(p) = id(\ell)$, $ttl(p) \leq \Delta$, and $p = \ell \Rightarrow ttl(p) = 0$. We define a *legitimate configuration* of Algorithm 1 as any configuration where every process has a legitimate state. By definition, every legitimate configuration is also quasi-legitimate.

Lemma 3. *Let $\mathcal{G}$ be a TVG of Class $\mathcal{TC}^{\mathcal{B}}(\Delta)$, $t \geq o_{\mathcal{T}}$, and e be an execution of Algorithm 1 in $\mathcal{G}_{[t,+\infty)}$ starting in a quasi-legitimate configuration. For every $r \geq \Delta$, the configuration at the end of Round r in e is legitimate.*

Proof: First, remark that for every $j > 0$, the communication network at Round j in e is G_{t+j-1}. Then, the proof of the lemma is based on the claim below.

Claim (*): *for every $i \geq 0$, $d \geq 0$, every process p such that $\hat{d}_{\ell,t+i-1}(p) \leq d$ satisfies: $\forall j \in \{1, \ldots, \Delta - \hat{d}_{\ell,t+i-1}(p) + 1\}$, $lid(p) = id(\ell)$ and $ttl(p) \leq$*

$\hat{d}_{\ell,t+i-1}(p) + j - 1$ *at the beginning of Round* $\big(i + j + \hat{d}_{\ell,t+i-1}(p)\big)$ *of* e.

Proof of Claim (*): By induction on d.

Base Case: If $d = 0$ and some $p \in V$ satisfies $\hat{d}_{\ell,t+i-1}(p) = d$, then $p = \ell$. Then, it is immediate since the initial configuration is quasi-legitimate (by hypothesis) and every subsequent configuration is quasi-legitimate too (by Lemma 2).

Induction step: consider any process p such that $\hat{d}_{\ell,t+i-1}(p) \leq d$ with $d > 0$. If $\hat{d}_{\ell,t+i-1}(p) < d$, the property is direct from the induction hypothesis. Consider now the case where $\hat{d}_{\ell,t+i-1}(p) = d$. There is a journey $\mathcal{J} \in \mathcal{J}(\ell, p)$ such that $departure(\mathcal{J}) > t + i - 1$ and $arrival(\mathcal{J}) = d + t + i - 1$. We denote $\mathcal{J}$ by $\{(e_0, t_0), (e_1, t_1), \ldots, (e_k, t_k)\}$ where $t_k = d + t + i - 1$. Let q be the process such that $e_k = (q, p)$. By definition, $\hat{d}_{\ell,t+i-1}(q) < \hat{d}_{\ell,t+i-1}(p) = d$. Hence, by induction hypothesis, $\forall j \in \{1, \ldots, \Delta - \hat{d}_{\ell,t+i-1}(q) + 1\}$, $lid(q) = id(\ell)$ and $ttl(q) \leq \hat{d}_{\ell,t+i-1}(q) + j - 1$ at the beginning of Round $i + j + \hat{d}_{\ell,t+i-1}(q)$. Let $j' = \hat{d}_{\ell,t+i-1}(p) - \hat{d}_{\ell,t+i-1}(q)$. Since $j' \in \{1, \ldots, \Delta - \hat{d}_{\ell,t+i-1}(q) + 1\}$,[3] we can instantiate the previous property with j'. We obtain $lid(q) = id(\ell)$ and $ttl(q) \leq \hat{d}_{\ell,t+i-1}(q) + j' - 1 = \hat{d}_{\ell,t+i-1}(p) - 1 = d - 1$ at the beginning of Round $i + j' + \hat{d}_{\ell,t+i-1}(q) = i + \hat{d}_{\ell,t+i-1}(p) = i + d$. Now, since $\rho(e_k, t_k) = 1$ and $t_k = d + t + i - 1$, q sends a message $\langle id(\ell), ttl_q\rangle$ to p during Round $i + d$ with $ttl_q \leq d - 1$, where ttl_q is the value of $ttl(q)$ at the beginning of Round $i + d$.

By definition of $id(\ell)$ and since there is no fake IDs (Lemma 2) the minimum message received by p in Round $i + d$ is $\langle id(\ell), ttl\rangle$ with $ttl \leq ttl_q \leq d - 1$. From the algorithm, $lid(p) = id(\ell)$ and $ttl(p) = ttl + 1 \leq d$ at the end of Round $i + d$, and so at the beginning of Round $i{+}d{+}1$. Then, by induction on $j \in \{1, \ldots, \Delta{-}d{+}1\}$, $ttl(p) \leq d + j - 1 \leq \Delta \leq 2\Delta - 1$ at the beginning of Round $i + \hat{d}_{\ell,t+i-1}(p) + j$

[3]More precisely, we have $j' \in \{1, \ldots, \Delta - \hat{d}_{\ell,t+i-1}(q)\}$.

since $ttl(p)$ is at most incremented by one during the previous round and p does not reset. So, $lid(p)$ remains equal to $id(\ell)$ at the beginning of Round $i + \hat{d}_{\ell,t+i-1}(p) + j$.

Let $r \geq \Delta \in \mathbb{N}^*$. We now apply Claim (*) to $d = \Delta$ so that every process p is taken into account by the claim: with $i = r - \Delta$, $j = \Delta - \hat{d}_{\ell,t+i-1}(p) + 1$, we obtain that $lid(p) = id(\ell)$ and $ttl(p) \leq \Delta$ at the beginning of Round $r + 1$; in addition, $ttl(\ell) = 0$ at the beginning of Round $r + 1$, by Remark 6. Hence, the configuration at the end of Round r is legitimate. $\square$

As a direct consequence of Corollary 4 and Lemma 3, we obtain the convergence.

Corollary 5. *Let $\mathcal{G}$ be a TVG of $\mathcal{TC}^{\mathcal{B}}(\Delta)$. For every $i \geq 3\Delta$, at the end of Round i of any execution of Algorithm 1 in $\mathcal{G}$, the configuration is legitimate.*

Lemma 4. *Let $\mathcal{G}$ be a TVG of $\mathcal{TC}^{\mathcal{B}}(\Delta)$, $t \geq o_{\mathcal{T}}$, and e be an execution of Algorithm 1 for $\mathcal{G}_{[t,+\infty)}$ starting in a legitimate configuration. For every $r \in \{1, ..., \Delta - 1\}$, the configuration e has reached at the end of Round r satisfies $lid(p) = id(\ell)$ and $ttl(p) \leq \Delta + r$, for every process p.*

Proof: First, the lemma trivially holds for $\Delta \leq 1$. So, we now show by induction on r that the lemma holds in the case where $\Delta > 1$.

Base case: At the beginning of Round 1, $\forall p \in V$, $lid(p) = id(\ell)$ and $ttl(p) \leq \Delta$ as the first configuration of e is legitimate. According to the algorithm, at the end of Round 1, $\forall p \in V$, $lid(p) = id(\ell)$ and $ttl(p) \leq \Delta + 1 < 2\Delta$ (since $\Delta > 1$).

Induction step: let $r \in \{2, ..., \Delta - 1\}$. At the end of Round $r - 1$, hence at the beginning of Round r, $\forall p \in V$, $lid(p) = id(\ell)$ and $ttl(p) \leq \Delta + r - 1 < 2\Delta - 1$, by induction hypothesis. According to the algorithm, and since $\Delta + r < 2\Delta$, no process can reset. Hence, at the end of Round r, $\forall p \in V$, $lid(p) = id(\ell)$ and $ttl(p) \leq \Delta + r$, and we are done.

□

Theorem 4 below is a direct consequence of Lemmas 3 and 4.

Theorem 4. *For every $\mathcal{G} = (V, E, \mathcal{T}, \rho) \in \mathcal{TC}^{\mathcal{B}}(\Delta)$, for every legitimate configuration γ of Algorithm 1 for V, the execution of Algorithm 1 in $\mathcal{G}$ starting from γ satisfies SP_{LE}.*

Proof: Let $e = \gamma_0 ... \gamma_i ...$ be an execution of Algorithm 1 in $\mathcal{G}$ such that γ_0 is legitimate. First, as γ_0 is legitimate, we have $lid(p) = id(\ell)$, for every process p (by definition). Then, by Lemma 4, for every $r \in \{1, ..., \Delta - 1\}$, at the end of Round r, *i.e.*, in Configuration γ_r, we have $lid(p) = id(\ell)$, for every process p. Finally, since γ_0 is quasi-legitimate (by definition, every legitimate configuration is also quasi-legitimate), Lemma 3 applies: for every $r \geq \Delta$, the configuration γ_r at the end of Round r is legitimate, so for every process p, $lid(p) = id(\ell)$ in γ_r. Hence, $SP_{LE}(e)$ holds. □

By Corollary 5 and Theorem 4, we have the next corollary.

Corollary 6. *Algorithm 1 is a self-stabilizing leader election algorithm for $\mathcal{TC}^{\mathcal{B}}(\Delta)$. Its stabilization time is at most 3Δ rounds. It requires $O(B + \log \Delta)$ bits per process and messages of size $O(B + \log \Delta)$ bits, where B is the number of bits used to store an ID.*

5. Class $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ with Δ and n known

The code of our self-stabilizing algorithm for $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ is given in Algorithm 2.

5.1. Overview of Algorithm 2

Each process p collects IDs in its variable $members(p)$. Actually, $members(p)$ is a (FIFO) queue containing at most n pairs $\langle id, t \rangle$, where id is an identifier and t is a timestamp, *i.e.*, an integer value less than or equal to Δ. In the following, we denote by $members(p)[id]$ the timestamp associated to the identifier id belonging to $members(p)$.

At each round i, p sends all pairs $\langle id, t\rangle$ of $members(p)$ such that $t < \Delta$ at the end of Round $i-1$ (Line 2). (The timestamps allow to eventually remove all fake IDs.) Then, p updates $members(p)$ by calling Function $insert(p, \langle id, t\rangle)$ for each received pair $\langle id, t\rangle$ such that $id \neq id(p)$ (Lines 4-5).

Each call to insertion function, $insert(p, \langle id, t\rangle)$, works as follows. If id already appears in $members(p)$, then the old pair $\langle id, t'\rangle$ is removed first from the queue (Lines i1-i2) and then $\langle id, \min(t, t')\rangle$ is appended at the tail of the queue (Lines i3). Otherwise, $\langle id, t\rangle$ is appended at the tail of the queue (Lines i6). However, since the size of $members(p)$ is limited, if the queue is full, its head is removed beforehand to make room for the new value (Line i5). Using this FIFO mechanism, initial spurious values eventually vanish from $members(p)$.

After all received pairs have been managed, the timestamps of all pairs in the queue are incremented (Line 6) and $\langle id(p), 0\rangle$ is systematically inserted at the tail of the queue (Line 7). This mechanism ensures two main properties. First, every timestamp associated to a fake ID in a variable $members$ is eventually forever greater than or equal to Δ (Lemma 5); and consequently, eventually no message containing fake IDs is sent (Corollary 7). Second, by definition of $\mathcal{TC}^{\mathcal{Q}}(\Delta)$, for every two distinct processes p and q, there are journeys of length at most Δ infinitely often, so each process p receives infinitely often messages containing $id(q)$ with timestamps smaller than Δ. Thus, eventually $members(p)$ exactly contains all IDs of the networks (Lemma 7). Now, at the end of each round, p updates its leader variable with the smallest ID in $members(p)$ (Line 8). Hence, the process of lowest ID, ℓ, is eventually elected.

5.2. Self-stabilization

Lemma 5. *Let f be a fake ID. For every $i \geq 1$, at the beginning of Round i, the following property holds: $\forall p \in V$ if f is in $members(p)$, then $members(p)[f] \geq \min(\Delta, i-1)$.*

Proof: By induction on $i \geq 1$. The base case, $i = 1$, is trivial since $members(p)[f]$ is a natural integer and $\Delta \geq 1$. For the induction step, assume that $i > 1$.

Algorithm 2: Self-stabilizing leader election for $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ for each process p.

```
Inputs:
  $n \in \mathbb{N}$              :  number of processes
  $\Delta \in \mathbb{N}^*$             :  recurrent bound on the temporal distance between processes
  $id(p) \in IDSET$      :  ID of $p$
Local Variables:
  $members(p)$          :  queue of at most $n$ elements
                           contain pairs $\langle id, t\rangle \in IDSET \times \{0, \ldots, \Delta\}$
  $lid(p) \in IDSET$     :  ID of the leader

Macros:
insert$(p, \langle id, t\rangle)$:
i1:   if $\exists t'$, $\langle id, t'\rangle \in members(p)$ then
i2:       remove $\langle id, t'\rangle$ from $members(p)$
i3:       push $\langle id, \min(t, t')\rangle$ at the tail of $members(p)$
i4:   else
i5:       if $|members(p)| = n$ then remove the head of $members(p)$
i6:       push $\langle id, t\rangle$ at the tail of $members(p)$

1: Repeat Forever
2:     SEND($\{\langle id, t\rangle \in members(p) : t < \Delta\}$)
3:     mailbox := RECEIVE()
4:     forall pair $\langle id, t\rangle$ in a message of mailbox do
5:         if $id \neq id(p)$ then insert$(p, \langle id, t\rangle)$
6:     forall $\langle id, t\rangle \in members(p) : t < \Delta$ do $members(p)[id] + +$
7:     insert$(p, \langle id(p), 0\rangle)$
8:     $lid(p) := \min\{id : \langle id, \_\rangle \in members(p)\}$
```

By induction hypothesis, we have: *(*)* $\forall p \in V$, *if* f *is in* $members(p)$, *then* $members(p)[f] \geq \min(\Delta, i-2)$ *at the beginning of Round* $i-1$. Let $p \in V$ such that f is in $members(p)$ at the beginning of Round i. There are two cases to consider.

710 1. Assume that $f \notin members(p)$ at the beginning of Round $i-1$. So, p received some pairs $\langle f, t\rangle$ during Round $i-1$. Each of those pairs satisfies $t \geq \min(\Delta, i-2)$ (by *(*)* and from the code of the algorithm) and is inserted into $members(p)$. Let $\langle f, t_{\min}\rangle$ be the message received by p at Round

$i-1$ with the smallest timestamp $t_{\min}$. By executing Line i3 or i6 for each received pair, we can deduce that $members(p)[f] = t_{\min} \geq \min(\Delta, i-2)$ right after the insertions at Round $i-1$. Hence, after executing Line 6, $members(p)[f] \geq \min(\Delta, i-1)$, and so is at the beginning of Round i.

2. Assume that f is in $members(p)$ at the beginning of Round $i-1$. Let t_α be the value of $members(p)[f]$ at the beginning of Round $i-1$. By *(*)*, $t_\alpha \geq \min(\Delta, i-2)$. There are two cases:

   (i) p does not receive any pair $\langle f, \_\rangle$ during Round $i-1$.

   After executing Line 6, $members(p)[f] = \min(\Delta, t_\alpha+1) \geq \min(\Delta, i-1)$, and so is at the beginning of Round i.

   (ii) p receives some pairs $\langle f, t\rangle$ during Round $i-1$. Each of those pairs satisfies $t \geq \min(\Delta, i-2)$ (by *(*)* and from the code of the algorithm) and is inserted into $members(p)$. Let $\langle f, t_{\min}\rangle$ be the message received by p at Round $i-1$ with the smallest timestamp $t_{\min}$.

   By executing Line i3 or i6 for each received pair, we can deduce that $members(p)[f] = \min(t_\alpha, t_{\min}) \geq \min(\Delta, i-2)$ right after the insertions at Round $i-1$. Hence, after executing Line 6, $members(p)[f] \geq \min(\Delta, i-1)$, and so is at the beginning of Round i.

□

Since a process p does not send a pair $\langle id, t\rangle$ of $members(p)$ with $t \geq \Delta$, we have:

Corollary 7. *In any round $\Delta + i$ with $i \geq 1$, no process receives a message containing fake IDs.*

Lemma 6. *$\forall p, q \in V$, if $id(q)$ is inserted into $members(p)$ during Round $\Delta + i$ with $i \geq 1$, $id(q)$ remains into $members(p)$ forever.*

Proof: If an ID id is in $members(p)$, it can only be removed from $members(p)$ if function $insert(p, \langle id', t\rangle)$ is called and one of the following two situations occurs:

- Line i2: if $id = id'$ but in this case id is immediately added at the tail of $members(p)$,
- Line i5: if $id \neq id'$, id is the head of the queue, and the size of $members(p)$ is already n.

So, after id is inserted (at tail) into $members(p)$, it requires the insertion of n distinct IDs that are not into $members(p)$ different from id in order to get id at the head of the queue and remove it. If id is inserted during Round $\Delta + i$, it is not a fake ID and the only other IDs that can be inserted into $members(p)$ are IDs of processes in V since p will not receive any fake ID (Corollary 7). Thus, at most $n-1$ distinct IDs different from id can be inserted after the insertion of id. Hence, id cannot be removed from $members(p)$. □

By definition of class $\mathcal{TC}^{\mathcal{Q}}(\Delta)$, for every pair of processes p and q, there exists $t \geq \Delta$ such that $\hat{d}_{q,o_\mathcal{T}+t-1}(p) \leq \Delta$. We denote by $t(q,p)$ the minimum value t that satisfies the above property, namely $t(q,p)$ represents the first date after $\Delta + o_\mathcal{T} - 1$ (*i.e.*, after Δ rounds) from which q can broadcast an information to p in no more than Δ rounds.

Lemma 7. *$\forall p, q \in V$, by the end of Round $t(q,p) + \Delta$, $id(q)$ is in $members(p)$ forever.*

Proof: Let $q \in V$. Remark, first, that $id(q) \in members(q) \wedge members(q)[q] = 0$ by the end of Round 1, by definition of Algorithm 2; see Line 7.

Let $p \in V$. If $q = p$ then using the remark above and since $t(q,p) + \Delta \geq 1$, we are done. We now assume $q \neq p$. As $\hat{d}_{q,o_\mathcal{T}+t(q,p)-1}(p) \leq \Delta$, there exists a journey $\mathcal{J} = \{(e_1, t_1), ..., (e_k, t_k)\}$ and a sequence of processes $p_0, \ldots, p_k$ such that $t_1 > o_\mathcal{T} + t(q,p) - 1$, $t_k = t(q,p) + \hat{d}_{q,o_\mathcal{T}+t(q,p)-1}(p) + o_\mathcal{T} - 1 \leq t(q,p) + \Delta + o_\mathcal{T} - 1$ and for every $i \in \{1, ..., k\}$, $e_i = (p_{i-1}, p_i)$ with $p_0 = q$ and $p_k = p$. To simplify the notations, let $\tau_i = t_i - o_\mathcal{T} + 1$ for every i in $\{1, ..., k\}$ such that the edge $e_i = (p_{i-1}, p_i)$ is present during Round τ_i. We have $\tau_1 > t(q,p)$, $\tau_k \leq t(q,p) + \Delta$, and $\tau_i - \tau_1 < \Delta$.

We prove by induction on i that for all $i \in \{1, ..., k\}$, (1) $id(q)$ is forever in $members(p_i)$ by the end of Round τ_i and (2) $members(p_i)[q] \leq \tau_i - \tau_1 + 1$ at the end of Round τ_i.

Base case: for $i = 1$, the edge (q, p_1) exists at Round τ_1. Using the first remark in the proof, at the beginning of Round τ_1, since $\tau_1 > t(q,p) \geq \Delta \geq 1$, we have $id(q) \in members(q) \wedge members(q)[q] = 0$. Hence, at Round τ_1, q sends $\langle id(q), 0\rangle$ in its message to p_1. Following the algorithm, p_1 inserts $id(q)$ in $members(p_1)$ during Round $\tau_1 > \Delta$. So, $id(q)$ is forever in $members(p_1)$ by the end of Round τ_1; see Lemma 6. Still following the algorithm, $members(p_1)[q] = 1$ at the end of Round τ_1.

Induction Step: Let $i > 1$. We assume the result holds for $i-1$: $id(q)$ is forever in $members(p_{i-1})$ by the end of Round τ_{i-1} and $members(p_{i-1})[q] \leq \tau_{i-1} - \tau_1 + 1$ at the end of Round τ_{i-1}. Hence, at the beginning of Round τ_i (and so, at the end of Round $\tau_i - 1$), we have: $id(q)$ in $members(p_{i-1})$ and as the timestamps are at most incremented by one at the end of each round, $members(p_{i-1})[q] \leq \tau_{i-1} - \tau_1 + 1 + \tau_i - 1 - \tau_{i-1} = \tau_i - \tau_1 < \Delta$.

During Round τ_i, the edge $e_i = (p_{i-1}, p_i)$ is present and p_{i-1} sends in its message to p_i a pair $\langle id(q), t_q\rangle$ such that $t_q \leq \tau_i - \tau_1$ since $\tau_i - \tau_1 < \Delta$. As p_i receives it, it inserts $id(q)$ in $members(p_i)$ in Round τ_i. Since $\tau_i > \tau_1 > \Delta$, Lemma 6 ensures that $id(q)$ remains forever in $members(p_i)$ by the end of Round τ_i. Moreover, following the algorithm, at the end of Round τ_i, we have $members(p_i)[q] \leq \tau_i - \tau_1 + 1$.

With $i = k$, $id(q)$ is forever in $members(p)$ by the end Round $\tau_k \leq t(q,p) + \Delta$. □

Let V be a set of processes. We define a *legitimate* configuration of Algorithm 2 for V as any configuration of Algorithm 2 for V where for every process p, we have $lid(p) = id(\ell)$ and $\{id : \langle id, \_\rangle \in members(p)\} = \{id(q) : q \in V\}$. Remark that the set of legitimate configurations of Algorithm 2 for V is closed in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$. Indeed, by definition of the algorithm, no message containing a fake

ID can be sent from such a configuration. Hence, the set $members(p)$ of every process p remains constant, $\min\{id : \langle id, \_\rangle \in members(p)\} = id(\ell)$ forever, and the next lemma follows.

Lemma 8. *Any execution of Algorithm 2 that starts from a legitimate configuration in an arbitrary TVG satisfies SP_{LE}.*

The next lemma is a direct consequence of Corollary 7 and Lemma 7.

Lemma 9. *$\exists t \geq \Delta$ such that the configuration reached at the end of Round $t + \Delta$ is legitimate.*

Proof: Corollary 7 ensures that for every $i > \Delta$ and $p \in V$, no fake ID is inserted in $members(p)$ at Round i. We let $T = \max\{t(q,p) : q, p \in V\}$. By definition, $T \geq \Delta$. By Lemma 7, we have that for every $p, q \in V$, $members(p)$ contains $id(q)$ at the end of Round $T + \Delta$. Let $p \in V$. As the size of $members(p)$ is bounded by the number n of processes, $members(p)$ is exactly the set of IDs of every process and, by Line 8, we also have $lid(p) = id(\ell)$. $\square$

By Lemmas 8-9, follows.

Theorem 5. *Algorithm 2 is a self-stabilizing leader election algorithm for $\mathcal{TC}^{\mathcal{Q}}(\Delta)$. It requires $O(n(B + \log \Delta))$ bits per process and messages of size $O(n(B + \log \Delta))$ bits.*

5.3. Time Complexity

As for Algorithm 1, we would like to exhibit a bound on the stabilization time of Algorithm 2. However, we will show below that it is impossible. Actually, we will even establish a stronger result since we will show that in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$, we cannot bound the time before any *pseudo-stabilizing* leader election algorithm definitely elects a leader. Pseudo-stabilization is a weak variant of self-stabilization initially introduced by Burns *et al.* [32] in the context of static networks. Intuitively, an algorithm is pseudo-stabilizing if all its executions (starting from arbitrary configurations) have a suffix satisfying the intended specification. In [33], we have accommodated this concept to the highly dynamic context as follows.

Definition 6 (Pseudo-stabilization). *An algorithm $\mathcal{A}$ is* pseudo-stabilizing *for the specification SP on a class $\mathcal{I}$ of infinite TVGs if for every set of processes V, every $\mathcal{G} \in \mathcal{I}$ 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 $\gamma_0, \gamma_1, \ldots$ is the minimum index i such that $SP(\gamma_i, \gamma_{i+1}, \ldots)$ holds. The* pseudo-stabilization time *in rounds is the maximum length of a pseudo-stabilization phase over all possible executions.*

Remark 7. *By definition, if an algorithm $\mathcal{A}$ is* self-stabilizing *for SP on a class $\mathcal{I}$ of infinite TVGs, then $\mathcal{A}$ is also* pseudo-stabilizing *for SP on $\mathcal{I}$ (but the reverse is not necessarily true [32]).*

Moreover, the length of the pseudo-stabilization phase is less than or equal to that of the stabilization phase in a given execution of $\mathcal{A}$; and so are its the pseudo-stabilization and stabilization time.

The idea behind the next impossibility result is that if we consider any infinite TVG $\mathcal{G}$ of $\mathcal{TC}^{\mathcal{Q}}(\Delta)$, then we can construct another infinite TVG by prefixing it with a finite TVG only constituted of independent sets, *i.e.*, graphs only constituted of isolated nodes without any link. By definition, the obtained infinite TVG still belongs to $\mathcal{TC}^{\mathcal{Q}}(\Delta)$. 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 $\mathcal{TC}^{\mathcal{Q}}(\Delta)$. There exists no function $f : \mathbb{N}^* \times \mathbb{N}^* \to \mathbb{N}$ such that $\forall \mathcal{G} \in \mathcal{TC}^{\mathcal{Q}}(\Delta)$ 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 $\mathcal{G} = (V, E, \mathcal{T}, \rho) \in \mathcal{TC}^{\mathcal{Q}}(\Delta)$ such that $|V| = n$ and whose prefix of length $f(n, \Delta)$ is only constituted of independent sets (no edge), *i.e.*, $\forall i \in \{1, ..., f(n, \Delta)\}$, the snapshot $G_{o_{\mathcal{T}}+i-1} = (V, E_{o_{\mathcal{T}}+i-1})$ is an independent set, *i.e.*, $E_{o_{\mathcal{T}}+i-1} = \emptyset$. Such a TVG exists in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$, by definition.

Let $e = \gamma_0, \gamma_1, ...$ be any execution of $\mathcal{A}$ in $\mathcal{G}$. By hypothesis, there exists a unique leader from Configuration $\gamma_{f(n,\Delta)}$. Let $p_\ell \in V$ 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 $V' = V \setminus \{p_\ell\} \cup \{v\}$. Let γ'_0 be any configuration of $\mathcal{A}$ for V' such that

1. v has any local state in γ'_0, and
2. $\forall p \in V' \setminus \{v\}$, the local state of p is the same in γ'_0 and γ_0.

The only difference between γ_1 and γ'_1 is that p_ℓ has been replaced by v (with an arbitrary local state). We now consider the execution $e' = \gamma'_0, \gamma'_1, ...$ of $\mathcal{A}$ in the infinite TVG $\mathcal{G}'$ which is identical to $\mathcal{G}$ except that p_ℓ has been replaced by v. Let $\mathcal{G}' = (V', E', \mathcal{T}, \rho')$, where $E' = \{(p,q) \in E : p \neq p_\ell \wedge q \neq p_\ell\}$ and $\forall (p,q) \in E', \forall i \in \mathcal{T}$

- if $p = v$, then $\rho'(p,q,i) = \rho(p_\ell, q, i)$,
- else if $q = v$, then $\rho'(p,q,i) = \rho(p, p_\ell, i)$,
- else $\rho'(p,q,i) = \rho(p,q,i)$.

Of course, $\mathcal{G}' \in \mathcal{TC}^{\mathcal{Q}}(\Delta)$ since $\mathcal{G} \in \mathcal{TC}^{\mathcal{Q}}(\Delta)$.

Claim (*): $\forall p \in V' \setminus \{v\}$, $\forall i \in \{0, ..., f(n,\Delta)\}$, *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 $i = 0$ is trivial, by construction of γ'_0. Let $i \in \{0, ..., f(n,\Delta) - 1\}$. 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 $i+1$ both in e and e'. So, q receives no message and takes the same state in Round $i+1$ 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 (*), $\forall q \in V' \setminus \{v\}$, $lid(q) = id(p_\ell)$ at the end of Round $f(n,\Delta)$ in e'. Now, $p_\ell \notin V'$ (in other words, $id(p_\ell)$ is a fake ID for V'). So, the suffix of

e' starting from $\gamma'_{f(n,\Delta)}$ does not satisfy SP_{LE}, *i.e.*, the length of the pseudo-stabilizing phase of the execution e' in the infinite TVG $\mathcal{G}' \in \mathcal{TC}^{\mathcal{Q}}(\Delta)$ is greater than $f(n, \Delta)$, a contradiction. $\square$

From Remark 7, we have the following corollary.

Corollary 8. *Let $\Delta \in \mathbb{N}^*$ and $n \geq 2$. Let $\mathcal{A}$ be a deterministic self-stabilizing leader election algorithm for $\mathcal{TC}^{\mathcal{Q}}(\Delta)$. There exists no function $f : \mathbb{N}^* \times \mathbb{N}^* \to \mathbb{N}$ such that $\forall \mathcal{G} \in \mathcal{TC}^{\mathcal{Q}}(\Delta)$ 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 $f(n, \Delta)$.*

From the previous corollary, we know that the stabilization time of Algorithm 2 cannot be bounded in general. However, there are relevant favorable cases where it can be; in other words Algorithm 2 is speculative. To see this, we now show that the stabilization time of Algorithm 2 is at most 2Δ rounds in $\mathcal{TC}^{\mathcal{B}}(\Delta) \subseteq \mathcal{TC}^{\mathcal{Q}}(\Delta)$. Remark first that the proof of self-stabilization given in Subsection 5.2 holds for $\mathcal{TC}^{\mathcal{B}}(\Delta)$ since $\mathcal{TC}^{\mathcal{B}}(\Delta) \subseteq \mathcal{TC}^{\mathcal{Q}}(\Delta)$. Furthermore, for every processes p and q, $t(q, p)$ is exactly Δ in $\mathcal{TC}^{\mathcal{B}}(\Delta)$. Hence in the proof of Lemma 9, we have $T = \max\{t(q, p) : q, p \in V\} = \Delta$; ensuring that in $\mathcal{TC}^{\mathcal{B}}(\Delta)$, the configuration reached at the end of Round 2Δ is legitimate. Hence, follows.

Theorem 7. *The stabilization time of Algorithm 2 in Class $\mathcal{TC}^{\mathcal{B}}(\Delta)$ is at most 2Δ rounds.*

6. Class $\mathcal{TC}^{\mathcal{R}}$ with n known

The code of our self-stabilizing algorithm for $\mathcal{TC}^{\mathcal{R}}$ is given in Algorithm 3.

6.1. Overview of Algorithm 3

Contrary to $\mathcal{TC}^{\mathcal{Q}}(\Delta)$, we have no timing guarantee at all for journeys in $\mathcal{TC}^{\mathcal{R}}$. Consequently and in contrast to Algorithm 2, we cannot use any bound on timestamps to stop forwarding fake IDs. Thus, contrary to Algorithm 2, Algorithm 3 will require an unbounded memory: it will use unbounded timestamps and each process will only store up to n IDs, those timestamped with the smallest values. Finally, notice that using this latter policy, we do no more need any FIFO mechanism to remove fake IDs: the regular growth of their timestamps will ensure that they will be eventually replaced by real IDs in the memory storage of all processes. Hence, similarly to Algorithm 2, each process p uses a variable $members(p)$ to collect IDs. However, this time, $members(p)$ is a map that can contain up to n IDs, each of them being associated with a timestamp (we denote by $members(p)[id]$ the timestamp associated to the identifier id belonging to $members(p)$).

At each round i, p sends the content of $members(p)$ (Line 2). Then, p updates $members(p)$ by calling Function *insert* on each received pair $\langle id, t\rangle$ such that $id \neq id(p)$ (Lines 4-5). The function *insert* works as follows: if id already appears in $members(p)$, then the associated timestamp is updated by keeping the smallest value (Line i1). Otherwise, p tries to insert $\langle id, t\rangle$ in the map. Actually, $\langle id, t\rangle$ is inserted in the map if the map is not full (Line i2) or t is smaller than the greatest timestamp tM in the map (Lines i3-i7). In this latter case, $\langle id, t\rangle$ overwrites any value having this timestamp in $members(p)$ (Line i6-i7). This overwriting mechanism allows to eventually remove all fake IDs from $members(p)$, since their timestamps will regularly increase. After $members(p)$ has been updated, all timestamps of $members(p)$ are incremented (Line 6) and then, $\langle id(0), 0\rangle$ is systematically inserted in the map (Line 7).

Actually, Algorithm 3 guarantees two main properties. First, at the beginning of any round i, any timestamp associated to a fake ID is greater than or equal to $i - 1$; see Lemma 10. Second, by definition of $\mathcal{TC}^{\mathcal{R}}$, at any point in time, every process can reach all the others through a journey. The key property is then to show that if some broadcast initiated by process p reaches a process q at Round i, then the value of the timestamp in the message is small enough to ensure the insertion of $id(p)$ into $members(q)$; see Lemma 11. These two properties ensure that eventually $members(p)$ exactly contains all IDs of the network. Now, at the end of each round, p updates its leader variable with the smallest ID in $members(p)$ (Line 8). Hence, the process of lowest ID, ℓ, is eventually elected.

Algorithm 3: Self-stabilizing leader election for $\mathcal{TC}^{\mathcal{R}}$, for each process p.

```
Inputs:
  $n \in \mathbb{N}$                  :  number of processes
  $id(p) \in IDSET$          :  ID of $p$
Local Variables:
  $members(p)$            :  map of size at most $n$, contain pairs $\langle id, t\rangle \in IDSET \times \mathbb{N}$
  $lid(p) \in IDSET$         :  ID of the leader

Macros:
$max(p)$:
m1:   if $|members(p)| < n$ then return $\perp$
m2:   else return $\langle id, t\rangle \in members(p)$ with maximum timestamp $t$

$insert(p, \langle id, t\rangle)$:
i1:   if $\langle id, \_\rangle \in members(p)$ then $members(p)[id] := min(t, members(p)[id])$
i2:   else if $max(p) = \perp$ then add $\langle id, t\rangle$ in $members(p)$
i3:   else
i4:       $\langle idM, tM\rangle := max(p)$
i5:       if $t < tM$ then
i6:           remove $\langle idM, tM\rangle$ from $members(p)$;
i7:           add $\langle id, t\rangle$ in $members(p)$

1: Repeat Forever
2:     SEND($\langle members(p)\rangle$)
3:     mailbox := RECEIVE()
4:     forall pair $\langle id, t\rangle$ in a message of mailbox do
5:         if $id \neq id(p)$ then $insert(p, \langle id, t\rangle)$
6:     forall $id$ : $\langle id, \_\rangle \in members(p)$ do $members(p)[id]++$
7:     $insert(p, \langle id(p), 0\rangle)$
8:     $lid(p) := \min\{id : \langle id, \_\rangle \in members(p)\}$
```

6.2. Self-stabilization

The lemma below can be shown using an induction similar to the one used in the proof of Lemma 5, page 27.

Lemma 10. *Let f be a fake ID. For every $i \geq 1$, at the beginning of Round i, the following holds: $\forall p \in V$, if f is in $members(p)$, then $members(p)[f] \geq i - 1$.*

Lemma 11. *For every $i \geq 1$, at the end of Round i, the following property holds: $\forall p, q \in V$, if $\hat{d}_{p,o_\mathcal{T}}(q) \leq i-1$, then $id(p)$ is in $members(q)$ and $members(q)[p] \leq i-1$.*

Proof: By induction on $i \geq 1$.

Base case: In Round 1, p tries to insert $\langle p, 0\rangle$ in $members(p)$ (Line 7). Since the timestamp associated with every other ID in $members(p)$ has been incremented beforehand (line 6), $\langle p, 0\rangle \in members(p)$ by the end of the first round.

Induction step: Assume that $i > 1$. By induction, at the end of Round $i-1$, we have, for every $p, q \in V$ such that $\hat{d}_{p,o_\mathcal{T}}(q) \leq i-2$, that $id(p)$ is in $members(q)$ and $members(q)[p] \leq i-2$. Let $p, q \in V$ such that $\hat{d}_{p,o_\mathcal{T}}(q) \leq i-1$. There are two cases to consider.

1. If $\hat{d}_{p,o_\mathcal{T}}(q) \leq i-2$ then, by induction hypothesis, at the end of Round $i-1$, $id(p)$ is in $members(q)$ and $members(q)[p] \leq i-2$. During Round i, $id(p)$ cannot be removed from $members(q)$. Indeed, by Lemma 10, the timestamps associated to fake IDs are greater than or equal to $i-1$. Now, timestamps are incremented during Round i (Line 6), thus $members(q)[p] \leq i-1$ at the end of Round i.
2. If $\hat{d}_{p,o_\mathcal{T}}(q) = i-1$ then this means that the arrival of the journey from p to q which provides $\hat{d}_{p,o_\mathcal{T}}(q)$ occurs at time $o_\mathcal{T}+\hat{d}_{p,o_\mathcal{T}}(q) = o_\mathcal{T}+i-1$. So, $\exists r \in V$ such that $\hat{d}_{p,o_\mathcal{T}}(r) \leq i-2$ and edge (r,p) exists at instant $o_\mathcal{T}+i-1$. Hence, (r,q) is present at the beginning of Round i and so q receives a message from r during Round i. By induction hypothesis, at the end of Round $i-1$, $id(p)$ is in $members(r)$ and $members(r)[p] \leq i-2$. Hence, q receives the pair $\langle p, tM\rangle$ with $tM \leq i-2$ during Round i. For the same reasons as in Case (1), this pair is not rejected but inserted into $members(q)$. Then, timestamps are incremented (Line 6), hence $members(q)[p] \leq i-1$ at the end of Round i.

□

Let V be a set of processes. We define a *legitimate* configuration of Algorithm 3 for V as any configuration of Algorithm 3 for V where $lid(p) = id(\ell)$ and $\{id : \langle id, \_\rangle \in members(p)\} = \{id(q) : q \in V\}$, for every process p. By definition of the algorithm, no message containing a fake ID can be sent from such a configuration. So, from any legitimate configuration, the set $members(p)$ of every process p is constant and $\min\{id : \langle id, \_\rangle \in members(p)\} = id(\ell)$ forever. Thus, the set of legitimate configurations of Algorithm 3 for V is closed in $\mathcal{TC}^{\mathcal{R}}$ and we have:

Lemma 12. *Any execution of Algorithm 3 that starts from a legitimate configuration in an arbitrary TVG satisfies SP_{LE}.*

Theorem 8 below is a direct consequence of Lemmas 11 and 12.

Theorem 8. *Algorithm 3 is a self-stabilizing leader election algorithm for $\mathcal{TC}^{\mathcal{R}}$.*

Proof: Let $p \in V$. By definition of $\mathcal{TC}^{\mathcal{R}}$, $\forall q \in V$, $\exists \mathcal{J} \in \mathcal{J}(p,q)$ such that $departure(\mathcal{J}) > o_{\mathcal{T}}$. The temporal length of $\mathcal{J}$ is finite. Thus, $\exists\ \delta(p) \in \mathbb{N}$ such that $\forall q \in V$, $\hat{d}_{p,o_{\mathcal{T}}}(q) \leq \delta(p)$. Thus, at the end of Round $\delta(p)+1$, $\forall q \in V$, $id(p)$ is forever in $members(q)$ by Lemma 11. Since $members(q)$ contains at most n entries, after $\max_{p\in V} \delta(p) + 1$ rounds, $members(q)$ contains the ID of every process and no fake ID. So q chooses $id(\ell)$ as leader at the end of Round $\max_{p\in V} \delta(p) + 1$. Hence, the system is in a legitimate configuration at the end of this round and, by Lemma 12, we are done. □

6.3. Time Complexity

Similarly to $\mathcal{TC}^{\mathcal{Q}}(\Delta)$, stabilization time of leader election algorithms cannot be bounded in $\mathcal{TC}^{\mathcal{R}}$, as shown below.

Lemma 13. *Let $n \geq 2$. Let $\mathcal{A}$ be a deterministic pseudo-stabilizing leader election algorithm for $\mathcal{TC}^{\mathcal{R}}$. There exists no function $g : \mathbb{N}^* \to \mathbb{N}$ such that $\forall \mathcal{G} \in \mathcal{TC}^{\mathcal{R}}$ 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 lemma is false. Then, there exists a function $g : \mathbb{N}^* \to \mathbb{N}$ such that $\forall \mathcal{G} \in \mathcal{TC}^{\mathcal{R}}$ 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 $\mathcal{TC}^{\mathcal{Q}}(\Delta) \subseteq \mathcal{TC}^{\mathcal{R}}$, this claim also holds for $\mathcal{TC}^{\mathcal{Q}}(\Delta)$. Let $f(n, \Delta) = g(n)$. We have then $\forall \mathcal{G} \in \mathcal{TC}^{\mathcal{Q}}(\Delta)$ 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 $f(n, \Delta)$, contradicting Corollary 8. □

From Remark 7, we have the following corollary.

Corollary 9. *Let $n \geq 2$. Let $\mathcal{A}$ be a deterministic self-stabilizing leader election algorithm for $\mathcal{TC}^{\mathcal{R}}$. There exists no function $f : \mathbb{N}^* \to \mathbb{N}$ such that $\forall \mathcal{G} \in \mathcal{TC}^{\mathcal{R}}$ 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 $f(n)$.*

Similarly to Algorithm 2, we now show that Algorithm 3 is speculative in the sense that we cannot bound its stabilization time in $\mathcal{TC}^{\mathcal{R}}$, but in a more favorable case, precisely in $\mathcal{TC}^{\mathcal{B}}(\Delta) \subseteq \mathcal{TC}^{\mathcal{R}}$, its stabilization time is at most $\Delta + 1$ rounds, despite Δ is unknown. The proof of the theorem below is the same as the one of Theorem 8 but as we consider a TVG in $\mathcal{TC}^{\mathcal{B}}(\Delta)$, for every $p \in V$, $\delta(p) \leq \Delta$. Hence the system reaches a legitimate configuration at the end of Round $\max_{p \in V} \delta(p) + 1 = \Delta + 1$.

Theorem 9. *The stabilization time of Algorithm 3 in $\mathcal{TC}^{\mathcal{B}}(\Delta)$ is at most $\Delta + 1$ rounds.*

7. Conclusion

We have addressed self-stabilization in highly dynamic identified message-passing systems by proposing self-stabilizing leader election algorithms for three major classes of time-varying graphs: $\mathcal{TC}^{\mathcal{B}}(\Delta)$, $\mathcal{TC}^{\mathcal{Q}}(\Delta)$, and $\mathcal{TC}^{\mathcal{R}}$.

It is worth noting that the impossibility result of Braud-Santoni *et al.* [22] applies to the class of *always connected over the time TVGs* of n processes which is actually included and so stronger than $\mathcal{TC}^{\mathcal{R}}$, as well as $\mathcal{TC}^{\mathcal{B}}(\Delta)$ and $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ for every $\Delta \geq \max(1, n-1)$. Precisely, this result forbids the existence of *silent* self-stabilizing solutions for most of non-trivial static problems and is due to the impossibility to distinguish in the considered class an edge appearing infinitely often from an edge appearing only a finite number of times. Actually, silent self-stabilization additionally requires all processes to eventually keep their local state constant [34]. Now, leader election is a static problem. We have chosen to avoid this issue by proposing non-silent, *a.k.a.*, *talkative* [35] solutions, *i.e.*, in our algorithms, a small part of the local state of each process (namely, the timestamps) is modified infinitely often.

Beyond extending our results to other particular problems, further research could focus on studying expressiveness of self-stabilization in TVGs. To that goal, broadcast problems should be investigated, again in very general TVG classes. Indeed, coupled with our leader election solutions, they should allow to build generic transformers, following, for example, the approaches proposed in [36, 37].

References

[1] E. W. Dijkstra, Self-stabilizing systems in spite of distributed control, Commun. ACM 17 (11) (1974) 643–644. `doi:10.1145/361179.361202`.
URL `https://doi.org/10.1145/361179.361202`

[2] J. Beauquier, S. Kekkonen-Moneta, On ftss-solvable distributed problems, in: J. E. Burns, H. Attiya (Eds.), Proceedings of the Sixteenth Annual ACM Symposium on Principles of Distributed Computing, Santa Barbara, California, USA, August 21-24, 1997, ACM, 1997, p. 290. `doi:10.1145/259380.259515`.
URL `https://doi.org/10.1145/259380.259515`

[3] C. Delporte-Gallet, S. Devismes, H. Fauconnier, Stabilizing leader election in partial synchronous systems with crash failures, J. Parallel Distributed

Comput. 70 (1) (2010) 45–58. `doi:10.1016/j.jpdc.2009.09.004`.
URL `https://doi.org/10.1016/j.jpdc.2009.09.004`

[4] M. Nesterenko, A. Arora, Dining philosophers that tolerate malicious crashes, in: Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02), Vienna, Austria, July 2-5, 2002, IEEE Computer Society, 2002, pp. 191–198. `doi:10.1109/ICDCS.2002.1022256`.
URL `https://doi.org/10.1109/ICDCS.2002.1022256`

[5] S. Dubois, T. Masuzawa, S. Tixeuil, Maximum metric spanning tree made byzantine tolerant, Algorithmica 73 (1) (2015) 166–201. `doi:10.1007/s00453-014-9913-5`.
URL `https://doi.org/10.1007/s00453-014-9913-5`

[6] S. Delaët, B. Ducourthial, S. Tixeuil, Self-stabilization with r-operators revisited, J. Aerosp. Comput. Inf. Commun. 3 (10) (2006) 498–514. `doi:10.2514/1.19848`.
URL `https://doi.org/10.2514/1.19848`

[7] A. K. Datta, S. Devismes, L. L. Larmore, V. Villain, Self-stabilizing weak leader election in anonymous trees using constant memory per edge, Parallel Process. Lett. 27 (2) (2017) 1750002:1–1750002:18. `doi:10.1142/S0129626417500025`.
URL `https://doi.org/10.1142/S0129626417500025`

[8] A. K. Datta, L. L. Larmore, Self-stabilizing leader election in dynamic networks, Theory Comput. Syst. 62 (5) (2018) 977–1047. `doi:10.1007/s00224-017-9758-9`.
URL `https://doi.org/10.1007/s00224-017-9758-9`

[9] S. Dolev, Optimal time self-stabilization in uniform dynamic systems, Parallel Process. Lett. 8 (1) (1998) 7–18. `doi:10.1142/S0129626498000043`.
URL `https://doi.org/10.1142/S0129626498000043`

[10] S. Dolev, A. Israeli, S. Moran, Self-stabilization of dynamic systems assuming only read/write atomicity, Distributed Comput. 7 (1) (1993) 3–16. `doi:10.1007/BF02278851`.
URL `https://doi.org/10.1007/BF02278851`

[11] S. Dolev, T. Herman, Superstabilizing protocols for dynamic distributed systems, Chic. J. Theor. Comput. Sci. 1997.
URL `http://cjtcs.cs.uchicago.edu/articles/1997/4/contents.html`

[12] K. Altisen, S. Devismes, A. Durand, F. Petit, Gradual stabilization, J. Parallel Distributed Comput. 123 (2019) 26–45. `doi:10.1016/j.jpdc.2018.09.002`.
URL `https://doi.org/10.1016/j.jpdc.2018.09.002`

[13] B. Bui-Xuan, A. Ferreira, A. Jarry, Computing shortest, fastest, and foremost journeys in dynamic networks, Int. J. Found. Comput. Sci. 14 (2) (2003) 267–285. `doi:10.1142/S0129054103001728`.
URL `https://doi.org/10.1142/S0129054103001728`

[14] A. Casteigts, P. Flocchini, W. Quattrociocchi, N. Santoro, Time-varying graphs and dynamic networks, Int. J. Parallel Emergent Distributed Syst. 27 (5) (2012) 387–408. `doi:10.1080/17445760.2012.668546`.
URL `https://doi.org/10.1080/17445760.2012.668546`

[15] M. Biely, P. Robinson, U. Schmid, M. Schwarz, K. Winkler, Gracefully degrading consensus and k-set agreement in directed dynamic networks, Theor. Comput. Sci. 726 (2018) 41–77. `doi:10.1016/j.tcs.2018.02.019`.
URL `https://doi.org/10.1016/j.tcs.2018.02.019`

[16] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, E. L. Wong, Zyzzyva: Speculative byzantine fault tolerance, ACM Trans. Comput. Syst. 27 (4) (2009) 7:1–7:39. `doi:10.1145/1658357.1658358`.
URL `https://doi.org/10.1145/1658357.1658358`

[17] S. Dubois, R. Guerraoui, Introducing speculation in self-stabilization: an application to mutual exclusion, in: P. Fatourou, G. Taubenfeld (Eds.), ACM Symposium on Principles of Distributed Computing, PODC '13, Montreal, QC, Canada, July 22-24, 2013, ACM, 2013, pp. 290–298. doi:10.1145/2484239.2484246.
URL https://doi.org/10.1145/2484239.2484246

[18] C. Gómez-Calzado, A. Casteigts, A. Lafuente, M. Larrea, A connectivity model for agreement in dynamic systems, in: J. L. Träff, S. Hunold, F. Versaci (Eds.), Euro-Par 2015: Parallel Processing - 21st International Conference on Parallel and Distributed Computing, Vienna, Austria, August 24-28, 2015, Proceedings, Vol. 9233 of Lecture Notes in Computer Science, Springer, 2015, pp. 333–345. doi:10.1007/978-3-662-48096-0_26.
URL https://doi.org/10.1007/978-3-662-48096-0_26

[19] A. Casteigts, A Journey through Dynamic Networks (with Excursions), 2018.
URL https://tel.archives-ouvertes.fr/tel-01883384

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

[21] S. Dolev, Self-Stabilization, MIT Press, 2000.
URL http://www.cs.bgu.ac.il/%7Edolev/book/book.html

[22] N. Braud-Santoni, S. Dubois, M. Kaaouachi, F. Petit, The next 700 impossibility results in time-varying graphs, Int. J. Netw. Comput. 6 (1) (2016) 27–41.
URL http://www.ijnc.org/index.php/ijnc/article/view/116

[23] M. Bournat, A. K. Datta, S. Dubois, Self-stabilizing robots in highly dynamic environments, Theor. Comput. Sci. 772 (2019) 88–110. doi:10.1016/j.

tcs.2018.11.026.
URL https://doi.org/10.1016/j.tcs.2018.11.026

[24] S. Cai, T. Izumi, K. 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) (2012) 433–445. doi:10.1007/s00224-011-9313-z.
URL https://doi.org/10.1007/s00224-011-9313-z

[25] S. Dolev, A. Hanemann, E. M. Schiller, S. Sharma, Self-stabilizing end-to-end communication in (bounded capacity, omitting, duplicating and non-fifo) dynamic networks - (extended abstract), in: A. W. Richa, C. Scheideler (Eds.), Stabilization, Safety, and Security of Distributed Systems - 14th International Symposium, SSS 2012, Toronto, Canada, October 1-4, 2012. Proceedings, Vol. 7596 of Lecture Notes in Computer Science, Springer, 2012, pp. 133–147. doi:10.1007/978-3-642-33536-5_14.
URL https://doi.org/10.1007/978-3-642-33536-5_14

[26] D. Angluin, Local and global properties in networks of processors (extended abstract), in: R. E. Miller, S. Ginsburg, W. A. Burkhard, R. J. Lipton (Eds.), Proceedings of the 12th Annual ACM Symposium on Theory of Computing, April 28-30, 1980, Los Angeles, California, USA, ACM, 1980, pp. 82–93. doi:10.1145/800141.804655.
URL https://doi.org/10.1145/800141.804655

[27] M. Barjon, A. Casteigts, S. Chaumette, C. Johnen, Y. M. Neggaz, Maintaining a distributed spanning forest in highly dynamic networks, Comput. J. 62 (2) (2019) 231–246. doi:10.1093/comjnl/bxy069.
URL https://doi.org/10.1093/comjnl/bxy069

[28] B. Charron-Bost, S. Moran, The firing squad problem revisited, Theor. Comput. Sci. 793 (2019) 100–112. doi:10.1016/j.tcs.2019.07.023.
URL https://doi.org/10.1016/j.tcs.2019.07.023

[29] B. Awerbuch, B. Patt-Shamir, G. Varghese, S. Dolev, Self-stabilization by local checking and global reset (extended abstract), in: G. Tel, P. M. B. Vitányi (Eds.), Distributed Algorithms, 8th International Workshop, WDAG '94, Terschelling, The Netherlands, September 29 - October 1, 1994, Proceedings, Vol. 857 of Lecture Notes in Computer Science, Springer, 1994, pp. 326–339. `doi:10.1007/BFb0020443`.
URL `https://doi.org/10.1007/BFb0020443`

[30] S. Dolev, A. Israeli, S. Moran, Resource bounds for self-stabilizing message-driven protocols, SIAM J. Comput. 26 (1) (1997) 273–290. `doi:10.1137/S0097539792235074`.
URL `https://doi.org/10.1137/S0097539792235074`

[31] G. Varghese, Self-stabilization by counter flushing, SIAM J. Comput. 30 (2) (2000) 486–510. `doi:10.1137/S009753979732760X`.
URL `https://doi.org/10.1137/S009753979732760X`

[32] J. E. Burns, M. G. Gouda, R. E. Miller, Stabilization and pseudo-stabilization, Distributed Comput. 7 (1) (1993) 35–42. `doi:10.1007/BF02278854`.
URL `https://doi.org/10.1007/BF02278854`

[33] K. Altisen, S. Devismes, A. Durand, C. Johnen, F. Petit, On Implementing Stabilizing Leader Election with Weak Assumptions on Network Dynamics, Research report, Université Grenoble Alpes, VERIMAG, UMR 5104, France ; LIMOS, Université Clermont Auvergne, CNRS, UMR 6158, France ; Université de Bordeaux, LaBRI, UMR 5800, France ; Sorbonne Université, Paris, LIP6, UMR 7606, France, to appear in PODC'21, ACM Symposium on Principles of Distributed Computing (October 2020).
URL `https://hal.archives-ouvertes.fr/hal-02979166`

[34] S. Dolev, M. G. Gouda, M. Schneider, Memory requirements for silent stabilization, Acta Informatica 36 (6) (1999) 447–462. `doi:10.1007/`

s002360050180.
URL https://doi.org/10.1007/s002360050180

[35] L. Blin, S. Tixeuil, Compact deterministic self-stabilizing leader election on a ring: the exponential advantage of being talkative, Distributed Comput. 31 (2) (2018) 139–166. doi:10.1007/s00446-017-0294-2.
URL https://doi.org/10.1007/s00446-017-0294-2

[36] A. Cournier, A. K. Datta, S. Devismes, F. Petit, V. Villain, The expressive power of snap-stabilization, Theor. Comput. Sci. 626 (2016) 40–66. doi:10.1016/j.tcs.2016.01.036.
URL https://doi.org/10.1016/j.tcs.2016.01.036

[37] S. Katz, K. J. Perry, Self-stabilizing extensions for message-passing systems, Distributed Comput. 7 (1) (1993) 17–26. doi:10.1007/BF02278852.
URL https://doi.org/10.1007/BF02278852