

Self-stabilizing Systems in Spite of High Dynamics

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

▶ **To cite this version:**

Karine Altisen, Stéphane Devismes, Anaïs Durand, Colette Johnen, Franck Petit. Self-stabilizing Systems in Spite of High Dynamics. 22nd International Conference on Distributed Computing and Networking, ICDCN'21, Jan 2021, Nara, Japan. pp.156-165, 10.1145/3427796.3427838 . hal-02376832v3

HAL Id: hal-02376832
https://hal.science/hal-02376832v3

Submitted on 15 May 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Self-stabilizing Systems in Spite of High Dynamics

Karine Altisen
Université Grenoble Alpes, VERIMAG, UMR 5104,
karine.altisen@univ-grenoble-alpes.fr

Stéphane Devismes
Université Grenoble Alpes, VERIMAG, UMR 5104, France
stephane.devismes@univ-grenoble-alpes.fr

Anaïs Durand
Université Clermont Auvergne, LIMOS, UMR 6158, France
anais.durand@uca.fr

Colette Johnen
Université de Bordeaux, LaBRI, UMR 5800, France
colette.johnen@u-bordeaux.fr

Franck Petit
Sorbonne Université, Paris, LIP6, UMR 7606, France
franck.petit@lip6.fr

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 in the class $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ or $\mathcal{TC}^{\mathcal{R}}$ cannot be size-ambiguous, justifying why our solutions for those 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. In $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ and $\mathcal{TC}^{\mathcal{R}}$, stabilization time cannot be bounded, except for trivial specifications. However, we show that our solutions are speculative in the sense that their stabilization time in $\mathcal{TC}^{\mathcal{B}}(\Delta)$ is $O(\Delta)$ rounds.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases self-stabilization, time-varying graphs, leader election, speculation

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

1 Introduction

Context. Starting from an arbitrary configuration, a *self-stabilizing algorithm* [18] 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* [6] to cope with process crashes, and *strict stabilization* [30] to withstand Byzantine failures. Furthermore, several self-stabilizing algorithms, *e.g.*, [17], 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.*, [16, 19, 24] (*n.b.*, in particular [16] deals with leader election), use this kind of argument to claim that they are suited for the dynamic context. Furthermore, several approaches, like *superstabilization* [23] and *gradual stabilization* [2], aims at additionally providing specific countermeasures to efficiently treat topological changes when they are both spatially and timely sparse. However, all these aforementioned approaches, *e.g.*, [2, 16, 19, 23, 24], 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.

Several works aim at proposing a general graph-based model to capture the network dynamics. In [32], the network dynamics is represented as a sequence of graphs called *evolving graphs*. In [13], the topological evolution of the network is modeled by a (fixed) graph where the nodes represent participating processes and the edges are communication links that may appear during the lifetime of the network. Each edge is labeled according to its presence during the lifetime of the network. Such graphs are called *Time-Varying Graphs* (TVGs, for short).

In highly dynamic distributed systems, an expected property is *self-adaptiveness*, *i.e.*, the ability of a system to accommodate with sudden and frequent changes of its environment. By definition, achieving self-stabilization in highly dynamic networks is a suitable answer to self-adaptiveness. *Speculation* [29] is another possible approach for self-adaptiveness. Roughly speaking, speculation 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 [26] 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.

Contribution. We initiate research on self-stabilization in *highly dynamic identified message-passing systems where dynamics is modeled using TVGs* to obtain solutions tolerating both transient faults and high dynamics. We reformulate the definition of self-stabilization to accommodate with 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 [27], $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ 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 temporal connectivity [13]; this latter class is the most general infinite TVG class introduced so far [11, 13].[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 shared 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. Our results show that, despite the existence of unique process identifiers, any deterministic self-stabilizing leader election algorithm working in 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 exactly 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 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. In general, stabilization time 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 timeliness guarantee at all (*n.b.*, by definition of the class, there is no bound on the temporal diameter). Again, in general, stabilization time cannot be bounded in $\mathcal{TC}^{\mathcal{R}}$, yet we show that the algorithm is speculative since its stabilization time in $\mathcal{TC}^{\mathcal{B}}(\Delta)$ is at most $\Delta + 1$ rounds.

Related Work. Ensuring convergence in highly dynamic networks regardless of the initial configuration may seem to be very challenging, even impossible in many cases [9]. However, there are a few works [8, 10, 22] that deal with this issue, yet in widely different models and assumptions than ours.

A recent work [8] 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 [8], the three classes studied the present paper never enforce the network to have a particular topology at a given time.

In [10], 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. Authors assume global fairness, which means that each pair of processes interacts infinitely often. This dynamic pattern is identified as Class $\mathcal{C}13$ in [13] and is a particular subclass of $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ (for $\Delta = 1$), which is itself included in $\mathcal{TC}^{\mathcal{R}}$; both classes being studied here. Notice also that our communication primitive is weaker, since we assume processes can only communicate using a local broadcast that allows the process to send at each round a common message to its unknown set of current neighbors. 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,

[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 [1, 20].

[2] Every static network is a very particular case of TVG, which belongs to all classes studied here, so the impossibility result of Angluin [3] still applies.

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.* [22] 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.

Roadmap. In Section 2, we present 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 our algorithms. The last section is dedicated to conclusions and perspectives.

2 Preliminaries

Time-varying Graphs. A *time-varying graph* (TVG for short) [13] 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, $\mathcal{T}$ 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. Notice that our definition of TVG is close to the model, called *evolving graphs*, defined in [32]. We denote by $o_{\mathcal{T}} = \min \mathcal{T}$ the first instant in $\mathcal{T}$. The *snapshot* of $\mathcal{G}$ at time $t \in \mathcal{T}$ is the graph $G_t = (V, \{e \in E : \rho(e, t) = 1\})$. 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 $[t, t']$. A *journey* is a sequence of ordered pairs $\mathcal{J} = (e_1, t_1), (e_2, t_2), \ldots, (e_k, t_k)$ where $\forall i \in \{1, \ldots, k\}$, $e_i = (p_i, q_i) \in E$ satisfies $\rho(e_i, t_i) = 1$ and $i < k \Rightarrow q_i = p_{i+1} \wedge t_i < t_{i+1}$. Nodes p_1 and q_k are respectively called the *initial* and *final extremities* of $\mathcal{J}$. 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 initial and final extremities are p and q, respectively. 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 $arrival(\mathcal{J}) - departure(\mathcal{J}) + 1$. By extension, we define the *temporal distance* from p to q at time $t \geq o_{\mathcal{T}} - 1$, denoted $\hat{d}_{p,t}(q)$, 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 $\min \emptyset = +\infty$). The *temporal diameter* at time $t \geq 0$ is the maximum temporal distance between any two nodes at time 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.

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

Class $\mathcal{TC}$ *(Temporal Connectivity)*, also denoted by $\mathcal{C}_3$ in [13] and $\mathcal{F}_2$ in [12]: 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 Temporal Connectivity)*, denoted by $\mathcal{C}_5$ in [13]: 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 [27]: at any point in time, every node can reach all the others through a journey of temporal length at most Δ, *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 can always eventually reach each other node through a journey of temporal length at most Δ. 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 $\mathcal{C}$ is *recurring* if $\mathcal{C}$ only contains infinite TVGs and, for every $\mathcal{G} \in \mathcal{C}$ and every $t \geq o_{\mathcal{T}}$, $\mathcal{G}_{[t,+\infty)} \in \mathcal{C}$. (In other words, every recurring TVG class is suffix-closed.) The three classes we will consider (*i.e.*, $\mathcal{TC}^{\mathcal{R}}$, $\mathcal{TC}^{\mathcal{B}}(\Delta)$, $\mathcal{TC}^{\mathcal{Q}}(\Delta)$) are recurring.

Computational Model. We consider the computational model defined in [5, 14]. 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. We assume that each process p has a unique identifier (ID for short). The identifier of p is denoted by $id(p)$ and taken in an arbitrary domain $IDSET$ totally ordered by $<$. 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). 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 $\mathcal{S}_{\mathcal{A}}^{V}(p)$ the 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 $(s_1, s_2, \ldots, s_n)$, where s_1 to s_n represent the states of the processes in 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 $\gamma_0, \gamma_1, \ldots$ 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

Definition. Self-stabilization has been originally defined for static networks. In [18, 20], it is defined as follows: an algorithm is self-stabilizing if, starting from an arbitrary configuration, it makes the system converge to a so-called *legitimate* configuration from which every possible execution suffix satisfies the intended specification (*n.b.*, we define a specification as a predicate over configuration sequences). Following the reference book of Dolev [20], we accommodate this concept with highly dynamic environments by splitting the definition into two properties: the *convergence* property, which requires every execution of the algorithm in the considered system to eventually reach a legitimate configuration; and the *correctness* property, which requires every possible execution suffix starting from a legitimate configuration to satisfy the specification.

▶ **Definition 1** (Self-stabilization). *An algorithm $\mathcal{A}$ is* self-stabilizing *for the specification SP on a class $\mathcal{C}$ 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 $\mathcal{G} \in \mathcal{C}$ with set of processes V and every configuration γ of $\mathcal{A}$ for V, every execution of $\mathcal{A}$ in $\mathcal{G}$ starting from γ contains a legitimate configuration $\gamma' \in \mathcal{L}$ (*Convergence*), and*
2. *for every $\mathcal{G} \in \mathcal{C}$ 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 2. 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 $\mathcal{G} \in \mathcal{C}$ 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 noticing that Definition 1, as the one given in the reference book of Dolev [20], 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 3 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.*, [4, 25, 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 6 for example. Below, we reformulate closure in the context of TVGs.

▶ **Definition 3** (Closure)**.** *Let $\mathcal{A}$ be a distributed algorithm, $\mathcal{C}$ be an infinite TVG class, V be a set of processes, and $\mathcal{S}$ be a set of configurations of $\mathcal{A}$ for V. $\mathcal{S}$ is* closed *in $\mathcal{C}$ if for every $\mathcal{G} \in \mathcal{C}$ 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 4. Again, when the considered class of TVGs is recurring, the definition of closure can be slightly simplified. If $\mathcal{A}$ is a distributed algorithm, $\mathcal{C}$ is a recurring TVG class, V is a set of processes, and $\mathcal{S}$ is a set of configurations of $\mathcal{A}$ for V, then $\mathcal{S}$ is *closed* in $\mathcal{C}$ if for every $\mathcal{G} \in \mathcal{C}$ 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 $\mathcal{S}$.

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 $lid(p)$. In the following, we call *fake ID* any value $v \in IDSET$ (recall that $IDSET$ is the definition domain of the identifiers) such that v is not assigned as a process identifier in the system, *i.e.*, there is no process $p \in V$ such that $id(p) = v$. In the 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 configuration from which a unique process is forever adopted as leader by all processes, *i.e.*, $\exists p \in V$ such that $\forall q \in V, lid(q) = id(p)$ forever. 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 $\mathcal{C}$ if it is self-stabilizing for SP_{LE} on $\mathcal{C}$.

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 in the network), *i.e.*, even partial knowledge such as an upper bound on n is not sufficient; see Theorem 6. 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 5** (Size-Ambiguity). *Let V be a set of processes and $k \in \mathbb{N}$. A distributed algorithm $\mathcal{A}$ is* (k, V)-ambiguous *if $0 < k < |V|$ and for every $U \subset V$ such that $|U| = k$ and every $p \in U$, $\mathcal{S}^U_{\mathcal{A}}(p) = \mathcal{S}^V_{\mathcal{A}}(p)$. 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 some 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 6.** *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 $V = \{p_0, \ldots, p_{n-1}\}$. Assume, by the contradiction, that $\mathcal{L}$ 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 $i \notin \{\frac{t}{2} \bmod n, \ldots, (\frac{t}{2} + n - k - 1) \bmod n\}$ and $j \notin \{\frac{t}{2} \bmod n, \ldots, (\frac{t}{2} + n - k - 1) \bmod n\}$.

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 $x \in \{0, \ldots, n-1\}$ 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 $V \setminus \{p_x, \ldots, p_{(x+n-k-1) \bmod n}\}$ is fully connected and all processes in the set $\{p_x, \ldots, p_{(x+n-k-1) \bmod n}\}$ are isolated.*

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 is well-defined 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; this ends the proof of Claim 3.

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. ◄

▶ Remark 7. The condition $\Delta \geq 2$ is necessary in Theorem 6, 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 receive 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 6, 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 6 is given in Corollary 8. 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 8.** *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 9. The scheme used in the proof of Theorem 6 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.

Knowledge of n and Closure in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$. We now show that every execution of a self-stabilizing algorithm for a recurring specification in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ necessarily converges to a closed set of (legitimate) configurations; see Theorem 12. Consequently, no deterministic self-stabilization leader election algorithm for $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ can be size-ambiguous (Theorem 14 and Corollary 15); justifying why algorithms presented in Sections 5 and 6 assume the exact knowledge of n.

▶ **Definition 10** (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 11** (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 12.** *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 2), γ_0 cannot be a legitimate configuration, a contradiction. ◀

▶ **Corollary 13.** *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 8 and 13, we deduce Theorem 14 and Corollary 15:

▶ **Theorem 14.** *No deterministic self-stabilizing leader election algorithm for $\mathcal{TC}^{\mathcal{Q}}(\Delta)$, with $\Delta \geq 2$, can be size-ambiguous.*

▶ **Corollary 15.** *No deterministic self-stabilizing leader election algorithm for $\mathcal{TC}^{\mathcal{R}}$ can be size-ambiguous.*

▶ Remark 16. Like in Remark 9, 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

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 mistrust* of p in $lid(p)$ and allows to eliminate messages containing fake IDs. The value $ttl(p)$ increases at each round if p does not receive a message; otherwise it is updated thanks to the received messages. The value of $ttl(p)$ can increase up to $2\Delta - 1$. Process p never increases $ttl(p)$ 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 Line u1).

At each round i, p first sends its leader ID together with its degree of mistrust; see Line 2. Then, p selects the received message $\langle id, ttl\rangle$ which is minimum using the lexicographic order (*i.e.*, the message with the lowest ID and with the lowest ttl to break ties, if any; see Line 6). If id is smaller than $lid(p)$, p updates its leader $lid(p)$; see Line 7. If $id = lid(p)$, it updates the $ttl(p)$ by taking the smallest value between $ttl(p)$ and ttl (in this way, p may decrease its mistrust in $lid(p)$; see Line 8). In either case, $ttl(p)$ is then incremented if $lid(p) \neq id(p)$. Finally, if $lid(p) \geq id(p)$, p systematically resets; see Line 10. 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 mistrust 0 at the beginning of the next round, $i + 1$.

Eventually, the elected process is the process of lowest ID, ℓ. Once elected, ℓ sends $\langle id(\ell), 0\rangle$ at each round and since the temporal diameter is upper bounded by Δ, all processes will regularly receive messages $\langle id(\ell), d\rangle$, with $d \leq \Delta < 2\Delta$ (since $\Delta \in \mathbb{N}^*$). Consequently, they will no longer reset, ensuring that ℓ will remain the leader forever.

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 mistrust in $lid(p)$

Macros:
$updateTTL(v)$**:**
u1: **if** $v \geq 2\Delta$ **then** $lid(p) := id(p)$; $ttl(p) := 0$ `// Reset`
u2: **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** $updateTTL(ttl(p) + 1)$
5: **else**
6: $\langle lid, ttl\rangle := \min\{$messages in mailbox$\}$
7: **if** lid<$lid(p)$ **then** $lid(p) := lid$; $updateTTL(ttl + 1)$
8: **else if** lid=$lid(p)$ **then** $updateTTL(\min(ttl(p), ttl)+1)$
9: **else** $updateTTL(ttl(p) + 1)$
10: **if** $lid(p) \geq id(p)$ **then** $lid(p) := id(p)$; $ttl(p) := 0$ `// Reset`

Self-stabilization and Complexity. First, by definition of the algorithm, the next remark follows.

▶ Remark 17. Since the end of the first round, $\forall p \in V$, we have $lid(p) \leq id(p) \wedge (lid(p) = id(p) \Rightarrow ttl(p) = 0)$.

▶ **Lemma 18.** *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.

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 $i - 1$, either p increments the value of $ttl(p)$ during Round $i-1$ (Line 4, 8, or 9), 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 7 or 8). In the latter case, at the beginning of Round $i - 1$, $lid(q) = f$ and $ttl(q) = t$, and, by induction hypothesis, $t \geq i - 2$. In both cases, $ttl(p) \geq i - 1$ at the beginning of Round i.

(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 at Round $i-1$. Similarly to Case (1), $ttl(p) \geq i-1$ at the beginning of Round i. ◀

Lemma 18 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 18 and thanks to the reset mechanism of Algorithm 1, we deduce the following corollary.

▶ **Corollary 19.** *At the end of Round 2Δ, the configuration is quasi-legitimate.*

Proof. By Lemma 18 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 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 17, 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 20.** *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'. By Remark 17, 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* iff $lid(p) = id(\ell)$, $ttl(p) \leq \Delta$, 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 21.** *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. The base case occurs 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 20).

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 $j' \in \{1, \ldots, \Delta - \hat{d}_{\ell,t+i-1}(q) + 1\}$, 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 20) 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 $i + \hat{d}_{\ell,t+i-1}(p) + j$ 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 $i + \hat{d}_{\ell,t+i-1}(p) + j$. This ends the proof of Claim (*).

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 17. Hence, the configuration at the end of Round r is legitimate. ◀

As a direct consequence of Corollary 19 and Lemma 21, we obtain the convergence.

▶ **Corollary 22.** *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 23.** *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 $\Delta > 1$. For the 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 $i \in \{2, ..., \Delta - 1\}$. At the end of Round $i - 1$, hence at the beginning of Round i, $\forall p \in V$, $lid(p) = id(\ell)$ and $ttl(p) \leq \Delta + i - 1 < 2\Delta - 1$, by induction hypothesis. According to the algorithm, and since $\Delta + i < 2\Delta$, no process can reset. Hence, at the end of Round i, $\forall p \in V$, $lid(p) = id(\ell)$ and $ttl(p) \leq \Delta + i$, and we are done. ◀

Theorem 24 below is a direct consequence of Lemmas 21 and 23.

▶ **Theorem 24.** *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 23, 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 21 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 22 and Theorem 24, we have the following corollary.

▶ **Corollary 25.** *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.*

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

Overview of Algorithm 2. Each process p uses a variable $members(p)$ to collect IDs. 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. (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 $insert$ on each received pair $\langle id, t\rangle$ such that $id \neq id(p)$ (Lines 4-5).

The insertion function $insert$ works as follows: if id already appears in $members(p)$, then the old pair tagged with id is removed first from the queue (Line i1), and in either case, $\langle id, t\rangle$ is appended at the tail of the queue (Lines i1 and i4). In particular, since the size of $members(p)$ is limited, if the queue is full, its head is removed to make room for the new value (Line i3). 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 then, $\langle id(p), 0\rangle$ 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; and consequently, eventually no message containing fake IDs is sent. 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 regularly receives messages containing $id(q)$ with timestamps smaller than Δ. Thus, eventually $members(p)$ exactly contains all IDs of the networks. Now, at the end of each round, p updates its leader variable with the smallest ID in $members(p)$ (Line 7). Hence, the process of lowest ID, ℓ, is eventually elected.

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 contains 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** remove $\langle id, t'\rangle$ from $members(p)$; push $\langle id, \min(t, t')\rangle$ at the tail of $members(p)$

i2: **else**

i3: **if** $|members(p)| = n$ **then** remove the head of $members(p)$

i4: 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: $insert(p, \langle id(p), 0\rangle)$; $lid(p) := \min\{id : \langle id, _\rangle \in members(p)\}$

Self-stabilization.

▶ **Lemma 26.** *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 $members(p)[f] \geq i - 1$.*

Proof. By induction on $i \geq 1$. The base case, $i = 1$, is trivial since $members(p)[f]$ is a natural integer. For the induction step, assume that $i > 1$. By induction hypothesis, at the beginning of Round $i-1$, we have: $\forall p \in V$, if f is in $members(p)$, then $members(p)[f] \geq i-2$. Let $p \in V$ such that f is in $members(p)$ at the beginning of Round i. There are two cases to consider.

(1) Assume that $f \notin members(p)$ at the beginning of Round $i-1$. So, p received the pair $\langle f, t\rangle$ during Round $i-1$ with $t \geq i-2$, and then $\langle f, tM\rangle$ is added to $members(p)$ with $tM = t$ by executing either Line i1 or i4. In both cases, after executing Line 6, $members(p)[f] \geq i-1$, and so is at the beginning of Round i.

(2) Assume that f is in $members(p)$ at the beginning of Round $i-1$. There are two cases: (i) p does not receive any pair $\langle f, _\rangle$ during Round $i-1$. So, by executing Line 6 and by induction hypothesis, $members(p)[f] \geq i-1$ at the beginning of Round i. (ii) p receives some pair $\langle f, t\rangle$ during Round $i-1$. So, by executing Line i1, $members(p)[f] := min(t, members(p)[f])$. Again, by assumption $t \geq i-2$ and $members(p)[f] \geq i-2$ at the beginning of Round $i-1$. So, in both cases, $members(p)[f]) \geq i-1$ 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 27.** *In any round $\Delta + i$ with $i \geq 1$, no process receives a message containing fake IDs.*

▶ **Lemma 28.** *$\forall p, q \in V$, 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 i1: if $id = id'$ but in this case id is immediately added at the tail of $members(p)$,
- Line i3: if $id \neq id'$, id is the head of the queue, and the size of $members(p)$ is already n.

After id is inserted (at tail) into $members(p)$, it requires the insertion of n different IDs that are not into $members(p)$ (and that are 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 27). Thus, at most $n-1$ 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 there exists a temporal journey from q to p of length no more than Δ.

▶ **Lemma 29.** *$\forall p, q \in V$, 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 $\mathcal{J} = \{(e_1, t_1), ..., (e_k, t_k)\}$ 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 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 28. 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 28 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 and $\min\{id : \langle id, _\rangle \in members(p)\} = id(\ell)$ forever. Hence:

▶ **Lemma 30.** *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 27 and Lemma 29.

▶ **Lemma 31.** *$\exists t \geq \Delta$ such that the configuration reached at the end of Round $t + \Delta$ is legitimate.*

Proof. Corollary 27 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 29, 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 (see Line i3), $members(p)$ is exactly the set of IDs of every process. By Line 7, we also have $lid(p) = id(\ell)$. ◀

By Lemmas 30-31, follows.

▶ **Theorem 32.** *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.*

Speculative Stabilization. Stabilization time cannot be bounded in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$, except for trivial specifications. Indeed, even if there exist infinitely many journeys of length bounded by Δ between any pair of distinct processes and Δ is known by all processes, the time between any two consecutive such journeys is unbounded, by definition of $\mathcal{TC}^{\mathcal{Q}}(\Delta)$. Consequently, we cannot bound the time necessary to route any piece of information from some process p to another process q, making the stabilization time unbounded in any case. We now show that Algorithm 2 is speculative in the sense that its stabilization time cannot be bounded in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$, but in a more favorable case, actually in $\mathcal{TC}^{\mathcal{B}}(\Delta) \subseteq \mathcal{TC}^{\mathcal{Q}}(\Delta)$, its stabilization time is at most 2Δ rounds. To see this, remark that the previous proof holds for $\mathcal{TC}^{\mathcal{B}}(\Delta)$ since $\mathcal{TC}^{\mathcal{B}}(\Delta) \subseteq \mathcal{TC}^{\mathcal{Q}}(\Delta)$. Yet, for every processes p and q, $t(q,p)$ is exactly Δ in this class of TVGs. Hence in the proof of Lemma 31, we have $T = \max\{t(q,p) : q, p \in V\} = \Delta$; ensuring that in Class $\mathcal{TC}^{\mathcal{B}}(\Delta)$, the configuration reached at the end of Round 2Δ is legitimate.

▶ **Theorem 33.** *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

Overview of Algorithm 3. 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-i5). In this latter case, $\langle id, t\rangle$ overwrites any value having this timestamp in $members(p)$ (Line i5). 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 2 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 34. 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 35. 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 7). 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; contains 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** remove $\langle idM, tM\rangle$ from $members(p)$; 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: $insert(p, \langle id(p), 0\rangle)$; $lid(p) := \min\{id : \langle id, _\rangle \in members(p)\}$

Self-stabilization. The lemma below and its proof are identical to those of Lemma 26.

▶ **Lemma 34.** *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 35.** *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 $i \geq 1$. For the 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 34, 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 $\exists r \in V$ such that $\hat{d}_{p,o_\mathcal{T}}(r) \leq i-2$. This means that the arrival of the journey from p to q which provides $\hat{d}_{p,o_\mathcal{T}}(q)$ occurs at time $o_\mathcal{T} + \hat{d}_{p,o_\mathcal{T}}(q) = o_\mathcal{T} + i - 1$. 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 36.** *Any execution of Algorithm 3 that starts from a legitimate configuration in an arbitrary TVG satisfies SP_{LE}.*

▶ **Theorem 37.** *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 in $members(q)$ by Lemma 35. 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 36, we are done. ◀

Speculative Stabilization. Similarly to $\mathcal{TC}^\mathcal{Q}(\Delta)$, stabilization time cannot be bounded in $\mathcal{TC}^\mathcal{R}$, except for trivial specifications (*n.b.*, $\mathcal{TC}^\mathcal{Q}(\Delta) \subseteq \mathcal{TC}^\mathcal{R}$). 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 37 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 38.** *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 noticing that the impossibility result of Braud-Santoni *et al.* [9] 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 $n \geq 0$ and $\Delta \geq n-1$. Precisely, this result forbids the existence of *silent* self-stabilizing solutions for many non-trivial static problems. Actually, silent self-stabilization additionally requires all processes to eventually keep their local state constant [21]. Now, leader election is a static problem. We have chosen to avoid this issue by proposing non-silent, *a.k.a.*, *talkative* [7] 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, our future work will focus on studying expressiveness of self-stabilization in TVGs. To that goal, we plan to first investigate broadcast problems, 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 [15, 28].

References

1 Karine Altisen, Stéphane Devismes, Swan Dubois, and Franck Petit. *Introduction to Distributed Self-Stabilizing Algorithms*, volume 8 of *Synthesis Lectures on Distributed Computing Theory*. Morgan & Claypool Publishers, 2019. arXiv:https://doi.org/10.2200/S00908ED1V01Y201903DCT015.

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

3 Dana Angluin. Local and global properties in networks of processors. In *Proceedings of STOC'80*, pages 82–93, 1980.

4 Baruch Awerbuch, Boaz Patt-Shamir, George Varghese, and Shlomi Dolev. Self-stabilization by local checking and global reset (extended abstract). In *Distributed Algorithms, 8th International Workshop, WDAG*, pages 326–339, 1994.

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

6 Joffroy Beauquier and Synnöve Kekkonen-Moneta. On FTSS-solvable distributed problems. In *PODC*, page 290, 1997.

7 Lélia Blin and Sébastien Tixeuil. Compact deterministic self-stabilizing leader election on a ring: the exponential advantage of being talkative. *Distributed Computing*, 31(2):139–166, 2018.

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

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

10 Shukai Cai, Taisuke Izumi, and Koichi Wada. How to prove impossibility under global fairness: On space complexity of self-stabilizing leader election on a population protocol model. *Theory Comput. Syst.*, 50(3):433–445, 2012. doi:10.1007/s00224-011-9313-z.

11 Arnaud Casteigts. *A Journey through Dynamic Networks (with Excursions)*. HDR, Université de Bordeaux, 2018.

12 Arnaud Casteigts, Serge Chaumette, and Afonso Ferreira. Characterizing topological assumptions of distributed algorithms in dynamic networks. In *SIROCCO*, volume 5869, pages 126–140, 2009.

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

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

15 Alain Cournier, Ajoy Kumar Datta, Stéphane Devismes, Franck Petit, and Vincent Villain. The expressive power of snap-stabilization. *Theor. Comput. Sci.*, 626:40–66, 2016.

16 Ajoy K. Datta and Lawrence L. Larmore. Self-stabilizing leader election in dynamic networks. *Theory Comput. Syst.*, 62(5):977–1047, 2018. doi:10.1007/s00224-017-9758-9.

17 Sylvie Delaët, Bertrand Ducourthial, and Sébastien Tixeuil. Self-stabilization with r-operators revisited. *J. of Aerospace Comp., Inf., and Comm.*, 3(10):498–514, 2006.

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

19 Shlomi Dolev. Optimal time self-stabilization in uniform dynamic systems. *Parallel Process. Lett.*, 8(1):7–18, 1998. doi:10.1142/S0129626498000043.

20 Shlomi Dolev. *Self-Stabilization*. MIT Press, 2000.

21 Shlomi Dolev, Mohamed G. Gouda, and Marco Schneider. Memory requirements for silent stabilization. *Acta Inf.*, 36(6):447–462, 1999.

22 Shlomi Dolev, Ariel Hanemann, Elad Michael Schiller, and Shantanu Sharma. Self-stabilizing end-to-end communication in (bounded capacity, omitting, duplicating and non-fifo) dynamic networks - (extended abstract). In Andréa W. Richa and Christian Scheideler, editors, *Stabilization, Safety, and Security of Distributed Systems - 14th International Symposium, SSS 2012, Toronto, Canada, October 1-4, 2012. Proceedings*, volume 7596 of *Lecture Notes in Computer Science*, pages 133–147. Springer, 2012. doi:10.1007/978-3-642-33536-5_14.

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

24 Shlomi Dolev, Amos Israeli, and Shlomo Moran. Self-stabilization of dynamic systems assuming only read/write atomicity. *Distributed Comput.*, 7(1):3–16, 1993. doi:10.1007/BF02278851.

25 Shlomi Dolev, Amos Israeli, and Shlomo Moran. Resource bounds for self-stabilizing message-driven protocols. *SIAM J. Comput.*, 26(1):273–290, 1997.

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

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

28 Shmuel Katz and Kenneth J. Perry. Self-stabilizing extensions for message-passing systems. *Distributed Computing*, 7(1):17–26, 1993.

29 R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. L. Wong. Zyzzyva: Speculative byzantine fault tolerance. *ACM Trans. Comp. Syst.*, 27:1–39, 2009.

30 Mikhail Nesterenko and Anish Arora. Dining philosophers that tolerate malicious crashes. In *ICDCS*, pages 191–198, 2002.

31 George Varghese. Self-stabilization by counter flushing. *SIAM Journal on Computing*, 30(2):486–510, 2000.

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